Computer Algebra Implementations of Whitham's Averaged Lagrangian Technique in the Field of Nonlinear Optics.

A thesis presented by Thomas David Arbuckle in part fulfilment of the requirements for the degree of Doctor of Philosophy to the Faculty of Engineering, Glasgow University, June 1992.

Acknowledgements.

Many people have contributed towards this thesis in some way or other. If I have referred to their work, or comments, in the text, I have acknowledged their contribution there. There remain, however, many people who have made a contribution and who have not been thanked and it is the purpose of this section to acknowledge and thank these people.

Firstly I would like to thank my supervisor, Dr. John Arnold, for his patience. It has taken many years to produce this thesis and I suspect that there must have been times when he must have wondered ... I would like to thank him for his encouragement in the earlier stages of my research, and for his sympathy when things other than research were foremost in my mind.

I must also acknowledge the contribution to the thesis made by the late Professor John Lamb, head of department. It was at his urging that I decided to commence my researches in this field and with his knowledge that I was permitted to continue my researches long after the allotted time might have been said to have passed. Of course, the current head of department, Professor Peter Laybourn must also be thanked for allowing me to continue for the final few months of my research.

Dr. Frances Goldman and Dr. John Nimmo of the mathematics department did their best to revitalise my failing memory. John was also helpful in suggesting further lines of research and in giving out sympathy. The brief section on the inverse scattering transform is at least partly inspired by his lecture notes which managed to make a difficult subject seem relatively easy. I would also like to thank Dr. Dan Martin whose attempts to teach me differential geometry were, I suspect, doomed from the start.

Alan Dawson, Elizabeth McAlavey and John Buchanan of the computing service department are probably quite tired of seeing me. Nevertheless, I'd like to thank Elizabeth, particularly, as contact person for Manchester and Lancaster, for all her help. It's been interesting ... The computing services of Manchester and Lancaster Universities must also be thanked for allowing me to use the MAPLE package on their mainframes. I'm sorry about the 70 megabyte output files, honest, Lancaster.

Dr. Michael Monagan, ETH, Zurich, and Dr. Tony Scott, ITAMP, Harvard, made several comments on the internal workings of the package which I hope to implement one day. We'll see how things go.

I'd like to thank Dr. David Harper, the former Computer Algebra Support Officer, currently of Queen Mary and Westfield College, University of London for his help and encouragement. Without his advice I'd probably still be trying to write the program and I suspect that without his help I would never have got this finished.

I should acknowledge the fact that just under half of the time that it took to produce this work was funded by an S.E.R.C. quota award. The rest was, of course, funded by the author.

Finally I'd like to thank my more intimate circle of acquaintances for their forbearance. My parents have managed to put up with me for almost thirty years: I suspect their patience is wearing thin. They have certainly subsidised these researches by supplying most of life's necessities free of charge. I'd like to thank my friends Barbara Denison and Betty Kerr for believing in me when it didn't seem as if others did. I'd like to thank my other friends for putting up with me and for making my life bearable. To the Adult and Continuing Education Department: "Thank you for supplying me with distractions." (Halt the cutbacks!) The list is continued below. I suppose the last person I have to thank is myself. Without my perseverance, I don't know how I would have managed.

Additional advice and contributions from:

Professor Samuel J. Thomson, Dr. Andrew Porte, (My turn for lunch.), Dr. Brian Webster, Dr. Archie Baker, (all Department of Chemistry);

Dr. Andrew Davies, Dr. Jim Hough, Dr. Bill Hogg, (all Department of Physics and Astronomy);

Dr. T. R. Foord, (Thank you.), Professor Richard de la Rue, Dr. John Davies, Dr. J. H. Marsh, (all Department of Electronics and Electrical Engineering);

Dr. Hazel Arnot, Dr. Pete Ansbro, Dr. Bindy Bhumbra, Dr. Grahame Maxwell, Dr. Clive Reeves, Dr. Kim Lee, Dr. Tin Cheng, Dr. Veli Airaksinen, Jon Frost, Ian McIntyre, (those who have gone before), Dr. Ivair Gontijo, Gary Gray, Andrew Smart, Dr. Susan Brunt, Wei Chen, and somebody else (for sharing an office with me);

The staff of Stow College, Department of Mathematics and Computing;

The staff of the Central College of Commerce, Department of Accounting, Computing and Office Studies, especially Alan Robertson and Liz Rae; (Thanks for employing

me folks.) Special mention here for Barbara Turner who said she would never talk to me again if I didn't put her name in my acknowledgements list.

My pupils, whose thanks to me have made it seem worthwhile sometimes, and whose appalling behaviour in some cases have made me realise what it must be like to be a teacher;

and,

Caroline Thaung, Takako Motoyama, Alan and Arlene Dee, Christine and Ian Stephenson, John Porter, Geddes Thomson, Stephen Mulrine and the writers' group, Ken MacLeod (PAX Associates), Dr. Helmut Morsbach and Kazue Kurebayashi, Kazuko Matsuda-Dow;

Adrian Cruden, Doreen MacMillan, Stephanie Denison, Joyce Fraser, Dr. Kazuo Kyuma, Tom and Amy Harris (for T.A.), Mary Black, fig rolls and coffee.

和美へ

Contents

Précis of Thesis

The work contained in the thesis consists of a number of topic areas.

1) It attempts to describe and explain in general terms the concept of a soliton and to enumerate examples of the use of the concept of the soliton in the field of nonlinear optics. A survey of the literature is made and references both theoretical and experimental are discussed.

2). After reviewing the concept of the soliton it describes physical systems in the field of nonlinear optics which are allied to those having solitonic solutions and describes how these allied systems may be solved or described mathematically. The Maxwell-Bloch equations are derived and the relevance of the lagrangian model to the work of Professor R. K. Bullough and coworkers is discussed. The mathematical machinery for describing quantized multi-level systems in terms of Lie basis sets is discussed with a view to incorporating such machinery in future lagrangian models. The work of Eberly and Hioe is especially relevant.

3). In deriving mathematical descriptions of physical situations, there seems to be no generic approach which may can be applied to a problem regardless of the manner in which it is posed. This results in an *ad hoc* addition of terms or of solution methods being applied to each problem on an individual basis. Nevertheless, a complicated and thorough model of nonlinear propagation due to Kodama and Hasegawa is discussed and contrasted with the lagrangian method to be implemented.

4). We describe Whitham's averaged lagrangian method and its variants, especially that due to Kawahara, in order that its application may illuminate the manner in which the physical properties of a medium give rise to additional terms within the equations of motion derived from the averaged lagrangian.

5). We construct two lagrangian descriptions of nonlinear optical systems, one classical and one semiclassical and, in addition, use the Boussinesq equation of water wave theory. The first of the optical models is a variant of the linear Lorentz model which

incorporates a nonlinear potential function. The second is based on a quantized atomic system which has a number of discrete levels. the occupations of these levels are described by a formalism allied to that of density matrices.

6). We attempt to use such descriptions to derive nonlinear describing equations. This is carried out by using the lagrangians derived as source data for algebraic programs.

7). We describe two computer algebra programs written to perform the task of generating such equations. The first of these is written using a package called REDUCE and is designed to calculate the averaged lagrangian for a given system. It is found that implementing a version of this program to calculate results for a lagrangian supplied to the program as data is not a trivial task. The second program is therefore written using a package called MAPLE and it is this program which is then used to calculate the equations of motion for the three lagrangians mentioned previously. In theory, it should be possible to do so for any lagrangian supplied to the program as data although there are limitations as to the complexity of the lagrangians it can handle using reasonable resources.

8). We present the results generated by these computer programs. Although the results are complicated, their derivation marks a step forward in that they could not be reasonably be derived by other methods.

9). The computer programs used are listed at the end of the thesis.

Yet with great toil all that I can attain
By long experience, and in learnèd schools,
Is for to know my knowledge is but vain,
And those that think them wise, are greatest fools.

The Tragedy of Croesus
Alexander, Sir William, Earl of Stirling, 1567-1640.

Chapter 1

We think as we do, mainly because other people think so.

Samuel Butler, 1835-1902.

1.1. Introduction.

The contents of this thesis can be split up into several distinct topic areas. This thesis aims:-

1). to describe and explain in general terms the concept of a soliton and to enumerate examples of the use of the concept of the soliton in the field of nonlinear optics;

2). to describe physical systems in the field of nonlinear optics which are allied to those having solitonic solutions and to describe how these allied systems may be solved or described mathematically;

3). to point out the problems that the above mentioned descriptions engender and to examine alternatives;

4). to describe Whitham's method and its variants especially that due to Kawahara;

5). to construct lagrangian descriptions of nonlinear optical systems;

6). to attempt to use such descriptions to derive nonlinear describing equations;

7). to describe two computer algebra programs written to perform the task of generating such equations; and finally

8). to present the results generated by these computer programs, commenting on their relationship with results already derived.

In reviewing the literature of nonlinear optics, it becomes apparent that the methods used in deriving describing equations for nonlinear optical systems tend to be somewhat *ad hoc* or, alternatively, such derivations tend to be application specific. Although the describing equation for nonlinear fibres, for example, is almost always some variant of the nonlinear Schrödinger equation, the way in which the various physical effects combine to produce the eventual equation is not always explored. The construction of a generic method which could be used to investigate a wide range of similar physical systems would be useful when examining the way in which the physical medium controls the final form of the describing equation.

Whitham's averaged lagrangian method is a very powerful method used to investigate the behaviour of waveforms in nonlinear systems. A variant of Whitham's method due to Kawahara can be used to systematically investigate any system obeying certain weak conditions and to generate a series of describing equations of motion to any

required order of accuracy. Its main disadvantage is that it requires the consideration of a great number of terms, tens of thousands for higher level results, so that use of the method on a daily basis is somewhat discouraged. Even for lower level results, results of the type which have been derived already, the researcher is required to work with several hundred terms. This increases the possibility of error in calculation and renders the method suitable only for occasional use on important problems. Its use as a comparative method for examining the effect of physical phenomena on the describing equations of nonlinear systems has, until now, not been considered.

The use of computer algebra systems to perform complicated mathematical calculations has occurred only comparatively recently. Using such systems it is possible to carry out calculations which would be impossible by hand because of their sheer size. In addition, computer algebra systems are less likely to make errors. Once a program has been constructed which will calculate the required answer, the likelihood of there being an error is much smaller than when the same result is calculated by hand; indeed, although the results produced by computer algebra systems can sometimes be wrong because of errors in the system's coding, these errors are rare and comparatively well documented in the packages used in this thesis. Furthermore, the errors tend to occur in new or little used parts of the packages, perhaps those to do with some uncommonly used special functions. When computer algebra systems have been used to check the results published in tables of integrals, for instance, errors have been discovered which were attributable to the book's author(s) and not to the computer algebra package. Results involving commonly used functions are very likely to be correct.

This thesis presents two computer algebra programs, one written for a package called REDUCE, the other for a package called MAPLE, which attempt to automate the processes of calculating averaged lagrangians and, in the second case, calculating the equations of motion of these averaged lagrangians. Using such programs the idea of examining the effects to high orders of different terms within the lagrangian by carrying out Kawahara's variant of Whitham's averaged lagrangian method becomes a reasonable proposition.

Using two models taken from the field of nonlinear optics as examples, it is shown how the programs can derive the averaged lagrangians describing the systems (given an original lagrangian) and in the case of the MAPLE program, how this averaged lagrangian may then be used to find the equations of motion of the averaged lagrangian. The first of these models is a classical one in which the Lorentz model of linear optics is modified to include a nonlinear potential term. The second model is a

semiclassical model in which a lagrangian formed in terms of density matrices models the behaviour of the medium, the wave component being kept in its classical form. The derivations of these lagrangians are indicated as well as the results produced by the program.

1.2. Solitons - General

A topic which will be of key importance in this thesis is the concept of the soliton. The soliton, which can be loosely described as a wave pulse having particle-like properties, is of interest because of its potential use in communication systems, not to mention the fact that solitons can also be used to describe a wide variety of phenomena from other disciplines. In the remainder of this chapter, we will go on to examine the concept of the soliton, its mathematical background, its applications and other pertinent topics.

1.2.1. Solitons I. What is a Soliton?

The definition of a soliton essentially depends upon the vocational area within which it is being discussed. What a soliton means to a mathematician is different to what it means to an engineer. To compound the situation, objects which might be described as solitons are often given different names in particular application areas. Throughout this thesis, we will use the term soliton to mean a soliton according to its mathematical definition, or, especially when describing applications of solitons in real systems, to mean a solitary wave - a wave which is similar to a soliton in shape but lacks its special mathematical properties. It will generally be clear from the context whether it is a true soliton or a solitary wave which is the subject of the reference.

What is a soliton? The following description is one of the most comprehensive [1].

A 'soliton' is not precisely defined, but is used to describe any solution of a nonlinear equation or system which (i) represents a wave of permanent form; (ii) is localized, decaying or becoming constant at infinity; and (iii) may interact strongly

with other solitons so that after the interaction it retains its form, almost as if the principle of superposition were valid.

In other words, crudely, a soliton is a wave pulse of some particular and unchanging shape which can interact with other pulses of the same kind and emerge from the interaction essentially as if the interaction had not occurred.

The discovery of the soliton is attributed to J. Scott Russell [2]. His description of how he first observed the great wave of translation from the banks of the Forth and Clyde canal has by now become famous: it serves little purpose to repeat it here. The interested reader can refer to either of the two references given above.

However, although studies into the general area of water waves were made thereafter by such authors as Boussinesq and Rayleigh, and indeed one of the most important describing equations in the field, the Korteweg-de Vries equation was subsequently discovered, it was not until numerical experiments performed on this Korteweg-de Vries equation by Zabusky and Kruskal in 1965 that the special position of such nonlinear equations began to be realised.

In a linear system it is well known that, given two wave profiles travelling at different velocities, each wave profile may be considered to be a discrete entity. Even at the time of their collision or overlap, whilst the overall picture of the two waves may be extremely complicated, it may be described by combining the descriptions of the waves when each is described separately. This is called the principle of superposition and it is the cornerstone of linear wave physics. Were the principle of superposition invalid then almost all of the tools we routinely use to synthesize wave forms would be invalid also. The principle of superposition does, however, only apply to linear systems - that is systems whose describing equations are linear. If the describing equation contains nonlinear terms, the principle of superposition is no longer valid and whilst Fourier analysis of the wave form is certainly possible, the Fourier components of the waveform no longer propagate as independent entities as they would in a linear system. This, then is the reason why solitons are so important.

Even though the equations which describe solitonic systems are nonlinear, and indeed they must be so for solitons to form, the wave profiles which we call solitons can interact within this nonlinear system essentially unchanged. For example, if a soliton of low amplitude is overtaking one of high amplitude (in the Korteweg-de Vries system) then as the two approach each other, they can be seen to be separate entities. As they merge, the waveform may be relatively complex - a double hump for example.

However, as they separate, the two pulses draw apart from each other unchanged in shape, the only change being a change of phase. All of this occurs in a nonlinear system.

Subsequent work discovered that solitons were the property of a few special equations, or hierarchies of equations, having special mathematical properties. Such equations are described as being completely integrable and have unusual behaviour under close mathematical scrutiny. We shall now go on to look at some of these equations in greater detail.

1.2.2. Solitons II. Some equations supporting solitons.

Perhaps the simplest exploration of the concept of the soliton is to be found in a review paper by Bullough [3]. For more detailed information, however, the interested reader should consult Drazin [1] or the book by Dodd, Eilbeck, Gibbon and Morris [4]. The review paper by Scott, Chu and McLaughlin [5] is useful for obtaining a historical perspective, it being one of the earliest review papers on the topic. For a reference concerning the use of solitons in condensed matter physics, the reader can refer to Bishop, Krumhansl and Trullinger [6]. A very mathematically slanted paper is that written by Bullough and Dodd [7].

The first equation to be solved to give solitonic solutions was the Korteweg-de Vries equation

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \tag{1.1}$$

This is the form usually given for the equation although there are others related to this one by transformations, for example

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + au\frac{\partial u}{\partial x} + \mu\frac{\partial^3 u}{\partial x^3} = 0 \tag{1.2}$$

It can be shown that the solitonic solution takes the form of a sech^2 pulse. The Korteweg-de Vries equation is used to describe water waves which are both weakly dispersive and weakly nonlinear. As such, we mention it here only for completeness.

Of the other equations known to have solitonic solutions, the general ones which will be of greatest importance in this thesis are the nonlinear Schrödinger equation and the sine-Gordon equation. The nonlinear Schrödinger equation is usually given by

$$i\frac{\partial u}{\partial t}+\frac{1}{2}\frac{\partial^2 u}{\partial x^2}+a|u|^2u=0,\ a \text{ constant} \tag{1.3}$$

and it is used to describe systems with strong dispersion coupled with weak nonlinearity. It forms the basis of equations which describe the propagation of pulses in optical fibres (as well as other physical systems) and can be shown to have a solution which is a sech envelope modulating a carrier wave.

Lastly the sine-Gordon equation is usually given by

$$\frac{\partial^2 u}{\partial x^2}-\frac{\partial^2 u}{\partial t^2}=\sin u \tag{1.4}$$

and it is associated with processes with both strong dispersion and nonlinearity. It has alternative forms, for example

$$\frac{\partial^2 u}{\partial x \partial t}=\pm\sin u \tag{1.5}$$

to which the first equation is related by a simple transformation.

It should perhaps be pointed out here that solitons do not form general solutions to these equations: they are particular ones. For example, the Korteweg-de Vries equation can be shown to have a wave solution in terms of Jacobian elliptic functions.

1.2.3. Solitons III. Why are they special?

We have already stated that solitons possess the special property that they can strongly interact with one another without being substantially changed by this interaction. The obvious question to ask is "Why is this the case?" This is a question which has not yet been fully answered. Any attempt to answer it in detail must be couched in highly mathematical language. Rather than becoming overly concerned with the mathematics, we refer the interested reader to the references. We attempt to give a brief overview as follows.

The use of conservation laws to solve physical problems is a technique well known to every physics undergraduate. That conservation laws in general can be said to

stem from one theorem is not so well known. This theorem, Noether's theorem [8][1], can be stated in many mathematical forms and is intimately bound up with much modern mathematics. For example, it is possible to state and prove the theorem in the language of symmetry groups of equations. Olver [9] gives such a statement in his book on the applications of Lie groups to differential equations. Alternatively one can restate the theorem in terms of differential geometry. A statement and proof of Noether's theorem is given in terms of manifolds and diffeomorphisms by Arnold [10].

It is perhaps simpler to think of it in a way allied to that of Lie theory. One imagines a lagrangian or similar functional which describes a physical situation. If there exists some contact (continuous) transformation of the independent variables which leaves the value of the functional unchanged, then there will exist a function which will remain constant throughout the evolution of the motion. This function is referred to as a conserved quantity. In many physical problems there will be at least one constant of the motion and there may be several. The functions representing energy, momentum, angular momentum and so on are common examples.

Now in physical problems, although there may be some conserved quantities, there are rarely more than a couple. If one thinks of a map of the evolution of the function in configuration space, one can imagine the value of the conserved quantity remaining constant as the evolution progresses. If one thinks of an equation which has solitonic solutions however, the situation is different. Such an equation will have the same simple types of conservation relations as those mentioned previously. It is however possible to show that for equations which support solitons, there exists an infinite number of such conserved quantities. In fact, this has been adopted as a necessary (but not sufficient) condition that the solutions to an equation are of solitonic type. One can therefore imagine an interaction between two solitons in some configuration space. Not only do they have the few usual conserved quantities determining the outcome of their interaction, but an infinite series of them, each of which must be obeyed throughout the collision. It is not difficult to see that such equations must be very special and likely to be rare.

That such conserved quantities exist is the basis of the method most widely used to determine the equations' solutions. This method is called the inverse scattering transform method and is the topic of the next section.

[1] An English translation of this paper can be found in *Transport Theory and Statistical Physics* 1,186-207(1971).

1.2.4. The Inverse Scattering Transform.

The inverse scattering transform (technique) is the subject of an extensive literature. Again we refer the interested reader to the literature for more detailed discussion. We present only an overview in this section. One of the fullest references in this area is given by Ablowitz and Segur in their book *Solitons and the Inverse Scattering Transform* [11]. In it they discuss the inverse scattering transform in great detail, presenting the results in mathematical completeness. They also describe some of the alternative methods to the inverse scattering transform method and attempt to place the method in its context. They make an interesting comment in the prologue to the book.

Certain nonlinear problems have a surprisingly simple underlying structure, and can be solved by essentially linear methods.

The mathematics behind this is somewhat complicated. The idea, however, is relatively simple. In solving linear problems, use is often made of the Fourier transform. If we wish to solve some linear evolution problem, solving the direct problem is often difficult. In order to simplify the problem, therefore, one transforms the equation into a simpler space in which the evolution can be followed more easily, Fourier space, and then transforms back again to find out the eventual result of the evolution. This is shown in diagram 1.1.

The idea behind the inverse scattering transform method is similar although it does involve more complicated mathematics. It can be shown that the method will reduce to the Fourier transform method for linear equations. Given a nonlinear equation such as the nonlinear Schrödinger equation or the Korteweg-de Vries equation, one wishes to solve the equation by finding out the solutions which will correctly describe the evolution of a pulse, or of any waveform. (Solitons are a particular solution.) This is essentially a difficult nonlinear problem. However, given the form of the describing equation and initial conditions -data which describes a solution to the equation before evolution- it is possible to use techniques borrowed from the field of quantum mechanics to transform the equation in such a way that the time evolution can be carried out simply.

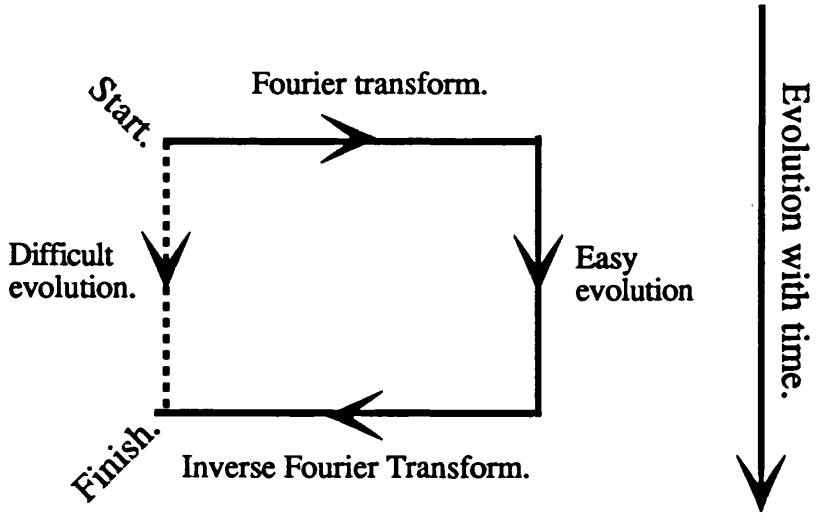


Diagram 1.1.

Using a Fourier Transform to Simplify a Calculation.

This is not meant to imply that the equations to be solved need have anything to do with quantum mechanics. Similar nonlinear equations may be derived in many different topic areas and the same method may be applied to these equations regardless of their source. However, problems in which quantum mechanical waves are scattered by some form of potential have been the subject of extensive research and techniques exist which enable the form of the potential to be calculated given the scattering data or, alternatively, for the scattering data to be calculated given the potential. These are the methods which are used in the inverse scattering transform technique.

The technique uses the initial data for the nonlinear evolution equation. Mathematical manipulations allow the data to be used to construct a function which describes a scattering potential in a time-independent Schrödinger equation. The scattering data has an easily calculated evolution. The transformation can then be inverted to give the form of the equation after the evolution. This will represent the solution of the nonlinear evolution equation.

This process is somewhat more complicated than has been indicated here. In order to carry out these manipulations, certain conditions must hold concerning the form of the initial data and, in addition, some complicated integrals must be carried out. The analogy with Laplace transforms may serve to give some indication of what is involved although, in the case of the inverse scattering transform technique, the additional complexity arising from the nonlinearity necessitates additional steps in carrying out the

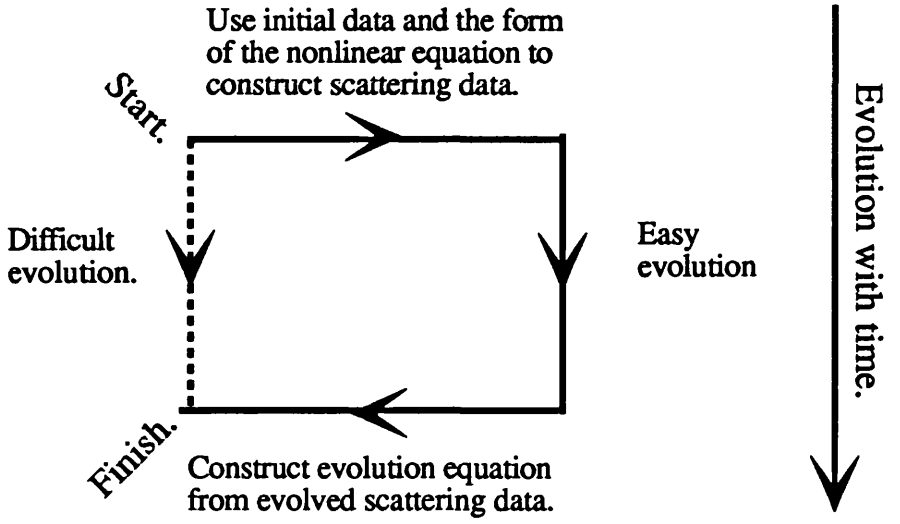


Diagram 1.2.
The Inverse Scattering Transform Method.

transformation. Fundamentally, it is the fact that, after transformation, the time variable is essentially reduced to an adjustable parameter which allows the simple evolution of the transformed data to be carried out.

A fuller description of this analogy can be found in Dodd, Eilbeck, Gibbon and Morris [4]. The transformation is explored in Drazin [1] and Newell also covers it in his book [12]. We should point out here that the explanation given above is highly simplified and that the process itself involves rather more mathematical manipulation and greater mathematical sophistication than that involved in a Fourier transform.

For one of the original papers in the field, the reader should consult Zakharov and Shabat [13]. In their paper, they discuss the solution of the nonlinear Schrödinger equation using the inverse scattering transform. A method of testing the complete integrability of nonlinear equations is presented by Chen, Lee and Liu [14]. (An equation is completely integrable if it has an infinite number of nontrivial conserved quantities and it is therefore possible that it has solitonic solutions.) Also of interest is a paper by Lewis [15] in which W.K.B. solutions for the scattering problem are used to examine soliton creation. This method represents a means by which the complexity of the calculations can be reduced. A further extension to the method is given by Elgin and Kaup [16] in which the authors use stochastic initial potentials. That is, the authors assume that there is a statistical uncertainty as to the exact form of the initial potential (which is formed from the describing nonlinear equation). This has potential uses in

modeling laser systems. There is, in addition, a sizeable literature in which perturbations of the scattering data are used to model perturbing terms appended to completely integrable equations. We shall see some of these references later.

Having quickly skimmed through the concepts and methods of the areas of research concerned with solitons, we will now go on to refer to these concepts and methods in reviewing their application in the field of nonlinear optics, especially in discussing models of pulse propagation of optical fibres.

1.3. Solitons in Nonlinear Optics.

Although the previous discussion of solitons has been in the context of research in the field of water waves, solitons are completely general objects which occur in many diverse fields. This is simply because the same equations arise in describing a wide range of physical phenomena. The review papers mentioned in the previous section point this out. The application area which is of interest here is that of nonlinear optics. Two of the equations which arise frequently in this area have already been mentioned:- the nonlinear Schrödinger equation and the sine-Gordon equation. There are others which have been solved in this field, notably the reduced Maxwell-Bloch equations which will be discussed in the next chapter. Our aim in the remainder of this chapter is to examine the ways in which the propagation of light pulses has been described by equations which are, or are similar to, completely integrable ones. The reason for this examination of the literature is partly to point out the great diversity of such equations, all of which describe essentially the same thing, and partly to show how the nonlinear phenomenon of the soliton has been put to practical use in constructing high capacity communications systems.

1.3.1. Solitonic Equations in Optics. I.

The field of linear optics has a long and varied history and it is not our purpose to review it here. The field of nonlinear optics is, however, a comparatively recent one, the first experimental demonstration of frequency doubling in a crystal being in 1961[2]. Our field of interest is yet more specialised than this. We wish to consider models which describe the propagation of a light beam, considered as a wave, within a nonlinear

[2] The experiment is described in P. A. Franken *et al*, Phys. Rev. Lett.7,118(1961).

medium. We wish also to consider associated phenomena associated with such models such as scattering (Brillouin and Raman), self-induced transparency and self-induced phase shifts, the competing effects of nonlinearity and dispersion. Our interest in these phenomena stems not from an intrinsic interest in the phenomena themselves, but rather from an interest in the effect such phenomena have on the propagation of light (pulses) within these media. To be more specific still, we wish to look at the effect such physical phenomena have on the equations which describe the light propagation.

This of course begs the question of what is to be done with these equations once they have been derived. There are three alternatives: the equations can be solved analytically; they can be solved numerically; or approximations can be made which will allow the solution of the simplified equations by one of the former two means. Although we will briefly consider numerical methods later in this chapter, the focus of this work is on solving the describing equations as exactly as possible. We wish to find analytic solutions, preferably exact ones. For problems as complicated as those which describe nonlinear light propagation, exact answers are generally impossible and so we usually resort to finding asymptotic solutions or to using perturbation methods.

We have mentioned that we will go on to discuss numerical techniques and phenomena derived by perturbation methods later in this section. Before this, however, it is necessary to look at the equations which have been derived to describe pulse propagation in nonlinear media, particularly nonlinear monomode fibres.

1.3.2. Solitonic Equations in Optics. II.

Linear theories describing pulse propagation in linear fibres and in linear layered media are well known and it is not our purpose to describe them here. The interested reader may consult the book by Gowar [17]. Other related topics include the use of the Beam Propagation Method (see later), the effective index method (for example see Chrostowski and Chelkowski [18]) and other methods which use Fourier synthesis to describe wave propagation. (See for example Marcuse [19].)

Although we will wish to discuss the sine-Gordon and reduced Maxwell-Bloch equations since they are used to describe some particular situations in nonlinear optics, we will delay this until the next chapter. The equation we will wish to discuss here is the nonlinear Schrödinger equation and its variants. This is because this is the equation which is used to describe the situation in which strong dispersion is coupled with weak nonlinearity in a physical system.

The equation can be derived in several ways and describes models of many disparate physical situations. Derivations are discussed in many texts on nonlinear optics. For example, see Dodd, Eilbeck, Gibbon and Morris [4] or Gowar [17]. As we have also pointed out, the equation can also be given in several alternative forms. Its usual one is

$$i\frac{\partial u}{\partial t}+\alpha\frac{\partial^2 u}{\partial x^2}+\beta|u|^2u=0,\ \alpha,\beta \text{ constants.} \tag{1.6}$$

Further variants of the nonlinear Schrödinger equation are possible. In a paper by Blow, Doran and Cummins [20], the authors discuss the effects of having third order dispersive terms in the absence of the more usual second order term. By assuming the usual dependence for the nonlinear refractive index

$$n=n_0+n_2|E|^2 \tag{1.7}$$

and using

$$E=\phi(z,t)R(t)\exp\left[i(\beta_0 z-\omega t)\right] \tag{1.8}$$

as the expansion for the electric field, substituting a Taylor expansion for the phase constant β and using the slowly varying envelope approximation gives

$$\left(i\frac{\partial}{\partial z}+i\beta_0'\frac{\partial}{\partial t}-\beta_0''\frac{\partial^2}{\partial t^2}-i\frac{\beta_0'''}{6}\frac{\partial^3}{\partial t^3}-i\frac{\beta_0'\beta_0''}{2\beta_0}\frac{\partial^3}{\partial t^3}+i\gamma+\frac{\beta_0 n_2}{n_0}|\phi|^2+\frac{2i\beta_0 n_2}{\omega_0 n_0}\frac{\partial}{\partial t}|\phi|^2\right)\phi=0 \tag{1.9}$$

which, if the second order dispersion term β_0'' is assumed zero gives, after a transformation, (see paper for details and units)

$$i\frac{\partial \upsilon}{\partial z'}-\frac{i}{6}\frac{\partial^3\upsilon}{\partial t^3}+i\Gamma\upsilon+|\upsilon|^2\upsilon+i\mathrm{E}\left(\frac{\partial}{\partial t}\right)\left(|\upsilon|^2\upsilon\right)=0 \tag{1.10}$$

The last term in the above expression has negligible effect on their calculations and the third term is a loss term which serves to broaden the pulses. If these two terms are removed, the similarity of the above expression to a nonlinear Schrödinger equation will be seen. After applying a numerical analysis to this equation, the authors find that the solitonic pulses spread out after being injected into the fibre. This obviously has important consequences for the capacity of a communication system.

1.3.3. Solitons III. Experimental Observations

Although we have described these effects in theoretical terms, they may be observed experimentally. There is a multitude of papers in the literature which describes experiments which are designed to explore the validity of the equations which have been derived and although our interest is theoretical rather than experimental, a brief examination of the types of papers which have been published is justified. Only by examining the success or failure of the theory by comparison with experiment can we hope to perfect the models which we use to describe the phenomena.

A good introductory reference to the experimental aspects can be found in a paper by Mollenauer [21]. In this paper, a brief review of the theory of solitons is presented as well as a description of experiments which have been conducted to observe solitons in optical fibres. One of the points raised is that one of the assumptions made in deriving the nonlinear Schrödinger equation to describe the pulse propagation in fibres is the assumption that the nonlinearity is instantaneous. If very short pulses are being propagated, the light pulse will require a quantized treatment as the number of cycles (of the carrier wave) drops. If, for example, there are, say, ten cycles of the carrier in the pulse, does it make sense to describe the pulse as being comprised of a carrier modulated by an envelope?

In a later section, we will see how the use of solitons in high capacity communication systems looks promising. In a paper by Mollenauer and Smith [22] the authors demonstrate that solitons may be transmitted over more than 4000 km in a fibre where the fibre loss is compensated by using the Raman effect. Another section will show how copropagating solitons exhibit forces between each other- sometimes attractive, sometimes repulsive. A paper by Mitschke and Mollenauer [23] demonstrates these forces experimentally.

We will later discuss papers which consider how the Raman effect would influence communication systems. An example of the difficulties this effect might engender has been observed experimentally by Mitschke and Mollenauer [23]. In their paper, solitonic pulses were injected into a length of fibre and the output pulses observed. They found that, at a power just above that necessary to form the first order soliton, a second pulse appeared. They state that one of these pulses is the soliton down-shifted in frequency by self-Raman pumping and that the other is the dispersive (non-

soliton) part of the radiation which travels at a different velocity due to group velocity dispersion. They call this effect the soliton self-frequency shift.

1.3.4. Vector Solitons

Throughout the above discussion, we have not mentioned the rôle of polarization. Some studies have been carried out in which birefringent fibres are modelled. The solitons which propagate in such systems are "vector solitons" which cannot be described by the simple nonlinear Schrödinger equation. The point made is that each of the polarizations behaves differently. A series of papers which discuss this subject has been written by Menyuk although other authors had researched the effect before these papers.

In the first of these papers [24], Menyuk uses two coupled nonlinear Schrödinger equations

$$\begin{aligned} i\left(\frac{\partial u}{\partial \xi}+\delta\frac{\partial u}{\partial s}\right)+\frac{1}{2}\frac{\partial^2 u}{\partial s^2}+\left(|u|^2+\frac{2}{3}|v|^2\right)u &= -i\gamma u \\ i\left(\frac{\partial v}{\partial \xi}+\delta\frac{\partial v}{\partial s}\right)+\frac{1}{2}\frac{\partial^2 v}{\partial s^2}+\left(\frac{2}{3}|u|^2+|v|^2\right)v &= -i\gamma v \end{aligned} \tag{1.11}$$

to model propagation in a birefringent fibre and makes the comment that all real monomode fibres are actually bimodal due to the presence of birefringence. This is important because the presence of birefringence can cause pulse splitting which would have obvious implications for communication systems.

In a further pair of papers [25, 26], he again discusses the effect of birefringence on soliton propagation. He uses numerical calculations based on the equations given previously to show that the nonlinearity can have the effect of binding solitonic pulses of each polarization together if the amplitudes of the pulses in each polarization are the same. They travel either as a single soliton or as a breather (two coupled solitons oscillating about a common centre). If the amplitudes in each polarization are unequal, he states that the breathers formed will eventually break apart. He finds that these processes are essentially unaffected by dissipative mechanisms.

These equations are also the subject of investigations by Christodoulides and Joseph [27] in which the authors discuss the related equations

$$i\frac{\partial u}{\partial z}+\frac{1}{2}\frac{\partial^2 u}{\partial \tau^2}+\left(|u|^2+A|v|^2\right)u+Bv^2u^*\exp(-4i\kappa z)=0$$
$$i\frac{\partial v}{\partial z}+\frac{1}{2}\frac{\partial^2 v}{\partial \tau^2}+\left(|v|^2+A|u|^2\right)v+Bu^2v^*\exp(+4i\kappa z)=0 \qquad (1.12)$$

They find that under certain conditions, a new class of soliton can result from propagation in birefringent fibres. (The harmonic terms are also included by Menyuk in his papers although he subsequently discards them.) The actual forms of these new solitons are given by the authors and they are not the usual sech profiles. The authors do state, however, that the objects they term solitons are only solitary waves: the special properties which we have assigned to true solitons are not proven for such objects.

Finally, we refer to a later use for birefringent effects given in a paper by Stolen and coworkers [28] in a subsequent section.

1.3.5. The Beam Propagation Method (B.P.M.).

We have mentioned that the equations derived to describe nonlinear propagation can be solved numerically. One of the methods which is commonly used for linear as well as nonlinear problems is that of Beam Propagation. Essentially, the idea is to discretize the medium through which the propagation is to take place and then to model the propagation in steps. At each step, the Fourier transform of the beam profile from the previous step is propagated linearly for the short distance covered by the grid.It is then inverted to give the new profile after one propagation step. After this, however, corrections are applied to this new profile to account for effects which are not modelled by the simple propagation such as nonlinear effects. Although the corrections applied are not exact, the propagation distances are so small that, given sufficiently short a distance, the error introduced by taking only the first order correction is relatively small. It will, of course, be appreciated that this method is highly computationally intensive. The method is also known as the split-step Fourier method.

B.P.M. has the advantage that it can model complex physical shapes such as couplers or layered structures. The first of these is modelled in a paper by Trillo and coworkers [29]. A dual core fibre is modelled using the technique and nonlinear switching is investigated. The second case mentioned is given by Leine and coworkers [30] in a paper which discusses the effect of loss on pulse propagation in a layered structure. We mentioned in the last section a paper discussing the interaction forces between solitons [23]. A short paper by Hermansson and Yevick [31] numerically

models this interaction using the beam propagation method. An alternative to B.P.M. is discussed in a paper by Yevick and Hermansson [32].

1.4. Communication Systems.

We have now looked at several of the effects and theories relevant to the study of solitons in general. We have also hinted that one of the main reasons why solitons are the object of so much current research is that their use in communication systems may lead to systems which have a much higher information carrying capacity and which will require fewer repeaters to propagate pulses without degradation.

There are several papers in the literature which discuss the use and design of soliton communication systems. It is the purpose of this section of the thesis to consider these references with a view to explaining why further research on equations describing soliton or soliton-like pulse propagation is required.

1.4.1. Early Papers on the use of Solitons in Communication Systems.

The first prediction that solitons would be used in communication systems was made in a famous paper by Hasegawa and Tappert [33]. To use their words:

A critical limitation in realizing the full-bandwidth capability of optical transmission systems is pulse distortion due to dispersive spreading. To overcome this difficulty, we propose to utilise the nonlinear dependence of the index of refraction on intensity that is intrinsic to glass and other dielectric materials employed in presently existing optical fibres.

Their model uses the cubic nonlinearity of glass generally given by the equation

$$n = n_0(\omega) + i\chi(\omega) + n_2\overline{E^2} \tag{1.13}$$

together with the following expression for the transverse electric field intensity

$$E(x,r,t) = R(r)\Re\left\{\phi(x,t)\exp\left[i(k_0 x - \omega_0 t)\right]\right\} \tag{1.14}$$

By assuming that the envelope amplitude function, ϕ, varies slowly compared to the carrier and by further assuming that the dispersive and nonlinear effects are weak, they obtain the following equation to describe the evolution of the pulse envelope.

$$i\left(\frac{\partial\phi}{\partial t}+\omega_0'\frac{\partial\phi}{\partial x}+v_0\phi\right)+\tfrac{1}{2}\omega_0''\frac{\partial^2\phi}{\partial x^2}+\frac{\alpha\omega_0 n_2}{n_0}|\phi|^2\phi=0 \tag{1.15}$$

They further state that equation (1.15) may be solved, under the assumption that the solution is localised and stationary, to give

$$\phi(x,t)=E_s\,\mathrm{sech}\left(\frac{t-t_0-x/v_g}{\tau_0}\right)\exp[i(\mathrm{K}x-\Omega t)] \tag{1.16}$$

The interested reader may consult the paper for the units and notation used. The important points, however, are that the describing equation used (1.3) is a variant of the nonlinear Schrödinger equation and that the localised solution used is simply a variant of the sech pulse envelope modifying a carrier.

Using the equations presented above, the authors go on to carry out numeric computer calculations in which they consider the effects of noise, absorption, and perturbations. They show that the pulses remain stable with respect to these perturbations and that the power required to produce them in real fibres is within the power ranges of currently available communication lasers. In a second paper [34], they carry out similar calculations for so-called dark solitons, solutions of the form

$$\phi(x,t)=E_s\tanh\left(\frac{t-t_0-x/v_g}{\tau_0}\right)\exp[i(\mathrm{K}x-\Omega t)] \tag{1.17}$$

obtained when the sign of the sign of the dispersion term $\partial^2\omega/\partial k^2$ is positive.

This paper presented an idea which many other authors then used as the basis for further research. The fact that one of the main limitations of current communication systems, the available bandwidth, might be overcome by using the inherent disadvantage of dispersion together with material nonlinearity was regarded as a breakthrough. Its importance can be judged by the numbers of papers which followed on the topic.

Another of the major limitations of fibres is considered by Smith in a paper from around the same period as the Hasegawa and Tappert papers [35]. The two processes of

stimulated Brillouin and Raman scattering can serve to degrade the information-carrying capacities of fibres. The Raman process serves to frequency-shift part of the light wave, the light generated by the process travelling either back (towards the source) or forwards. The Brillouin effect serves only to reflect light backwards towards the source. These processes occur spontaneously at all powers but at higher powers are stimulated: thus the effect becomes power dependent. Accordingly, if sufficient power is to be injected into the fibre guide in order to utilise the nonlinear effect of the medium for solitonic pulse propagation, the processes of stimulated Brillouin and Raman scattering will have to be considered. The equations describing the pulse propagation will have to be modified to take account of these effects by introducing additional terms.

The physical properties of fibres are discussed in a review article by Cotter [36] in which he also discusses many of the advantages and disadvantages of the use of optical communication systems. An experimental investigation of the effect of stimulated Brillouin scattering on pulse propagation in optical fibres is the topic of a paper by Ippen and Stolen [37]. In it they give experimental evidence to show that stimulated Brillouin scattering can be a problem even at comparatively low power levels (less than 1 W).

The design of a solitonic communication system is discussed in another paper by Hasegawa and Kodama [38] which we will review in a later chapter. Basically this paper presents a recipe for designing nonlinear optical communication systems.

As a further demonstration of the problems to be faced, Kapron's paper [39], discusses the fact that even when the material dispersion is zero, additional higher order effects will still affect pulse propagation.

In a paper by Christodoulides and Joseph [40] the authors present a model for pulse propagation in which they include fourth order effects. If care is taken to operate the fibre near certain 'operating points', they claim that the bandwidth of the fibre may be exploited more fully than in conventional systems, pulses of femtosecond duration being used as the information carriers. The equation they derive

$$\frac{\partial^2\phi}{\partial z^2}-\beta_0'^2\frac{\partial^2\phi}{\partial t^2}+2\beta_0\left[i\left(\frac{\partial\phi}{\partial z}+\beta_0'\frac{\partial\phi}{\partial t}\right)-\frac{\beta_0''}{2}\frac{\partial^2\phi}{\partial t^2}-i\frac{\beta_{0e}'''}{6}\frac{\partial^3\phi}{\partial t^3}+\frac{\beta_{0e}^{IV}}{24}\frac{\partial^4\phi}{\partial t^4}\right]$$
$$+\frac{2n_2\beta_0^2}{\alpha n_0}\left[|\phi|^2\phi+\frac{2i}{\omega_0}\frac{\partial}{\partial t}\left(|\phi|^2\phi\right)-\frac{1}{\omega_0^2}\frac{\partial^2}{\partial t^2}\left(|\phi|^2\phi\right)\right]=0 \tag{1.18}$$

is presented here for later comparison. It may be seen by inspection that it is similar to the nonlinear Schrödinger equation with the nonlinear and dispersive terms carried to higher orders.

1.4.2. More Recent Work.

In a short paper by Bava and coworkers [41], the effect of laser fluctuations on soliton propagation in optical communication systems is considered. The authors consider the initial scattering data of the inverse scattering transform to be subject to random perturbations which they attribute to fluctuations in the laser output. The effect produced is that the solitons produced do not all have the same velocity but have a range of velocities. This will obviously lead to problems with any communication system but, as they point out, the problem will affect most severely those systems having a long separation between repeater links.

Another effect which has been studied is the effect of frequency chirp on the performance of optical communication systems. Agarwal and Potasek [42] use a linear model with supergaussian pulses They claim that this more closely represents a directly modulated semiconductor laser.Their finding is that in the regime of anomalous dispersion (necessary for solitons) the effect of chirp is to broaden the pulses which, of course, impairs the information carrying capacity of the system. The modeling equation they use is (linear)

$$i\left(\frac{\partial A}{\partial z}+\gamma A\right)-\tfrac{1}{2}\beta^{(2)}\frac{\partial^2 A}{\partial t^2}=0 \tag{1.19}$$

A further study of source chirping is to be found in a paper by Desem and Chu [43]. They use inverse scattering theory to show that large chirping leads to much increased mutual interaction between solitons although, for more moderate chirping, they find that this interaction effect can be lessened by letting the solitons be of unequal amplitudes.

In an earlier paper on the same subject [44], Desem and Chu used a numerical study based on a simultaneous solution of the inverse scattering transform eigenvalue equations of the nonlinear Schrödinger equation to show that, given certain conditions, a soliton will still form and propagate in the presence of frequency chirp. They found, however, that more energy was transferred to the dispersive tail of the soliton and that chirping the pulse also has the effect of simulating a lower input power. If the input power drops below a critical level then a true soliton may fail to form.

The effect of noise on a soliton based communication system is the subject of a paper by Gordon and Haus [45]. Using results from noise theory, the authors calculate the limits placed on communication systems by amplifier noise. They find that such noise will limit the bit-rate length product by introducing a random shift in the soliton's carrier frequency.

Two papers by Wai and coworkers [46, 47] describe nonlinear pulse propagation near the zero-dispersion wavelength of monomode fibres. In the first of these papers [46] the nonlinear Schrödinger equation with dissipative and higher order dispersive terms

$$i\frac{\partial q}{\partial \xi} \pm \frac{1}{2}\frac{\partial^2 q}{\partial \tau^2} + |q|^2 q = -i\Gamma q + i\beta\frac{\partial^3 q}{\partial \tau^3} \tag{1.20}$$

is used to model breathers (higher order solitons) launched near the zero of the group-dispersion curve. The authors find that the effect of the higher order dispersion is to split the breathers into separate lower order solitons which obviously has implications for communications systems. This splitting will occur for breathers whose period of oscillation is short: for longer breathers, the third order term does not appear to affect the propagation of the pulses overmuch. The first order soliton is also found to be well behaved even in the presence of the third order dispersion term.

In the second of the papers , the authors describe a possible alternative to the use of the nonlinear Schrödinger equation to describe the pulse propagation by suggesting that the pulses are launched at exactly the zero of the group-velocity dispersion curve. They point out that at the zero, the nonlinear Schrödinger equation is no longer an accurate description of the situation. The equation that they suggest is

$$i\frac{\partial q}{\partial \xi} - i\frac{1}{6}\frac{\partial^3 q}{\partial s^3} + |q|^2 q = -i\Gamma q \tag{1.21}$$

However, their calculations suggest that a soliton, or rather a solitary wave, will be formed despite this. The scheme has the advantage that the power required to launch these solitary waves is lower than that required to launch true solitons into the same fibre.

1.5. Other Points

In a paper by Suzuki and coworkers [48] the use of an Er^{3+}-doped fibre for optical communication systems is discussed. The use of erbium as a dopant in a fibre allows it to act as an amplifier when it is pumped using a laser correctly tuned to the two transition wavelengths for the erbium. (Erbium can be considered to be a three level quantum system.) Using a short length of specially prepared fibre as a fibre amplifier allowed the authors to propagate pulses without change of shape over a much longer length of dispersion shifted fibre. The use of an erbium amplifier additionally allows the operation of the fibres at a minimum in their loss curve since it is at this point in the curve that the fibre amplifiers operate.

Self-focussing in fibres is a nonlinear effect in which the intense fields produced by short pulses cause the pulses to focus within the fibre due to the nonlinearity inherent in the fibre medium. This effect has been observed experimentally in a broad multimode fibre (see, for example, Baldeck and coworkers [49]). It has also been modelled by Manassah, Baldeck and Alfano [50].

1.5.1. Sticky Solitons.

One of the main impediments to the implementation of high capacity soliton communication systems that has been discovered so far is the presence of interaction forces between the solitons as they propagate along fibres. It was shown by Gordon [51] that there exist interaction forces between neighbouring solitons propagating in a fibre which are exponentially dependent on the separation between the solitons and which vary sinusoidally with their phase. Obviously if each soliton is being used to represent a bit in a digital communication system then this would create a limit in the length of the duty cycle although this might be counteracted somewhat by the choice of an appropriate coding scheme.

Numerical investigations of the effect were made by Hermansson and Yevick [31].The interaction was subsequently observed experimentally by Mitschke and Mollenauer [23].

In a paper by Blow and Doran [52], the authors discuss the effects of loss on the propagation of solitonic pulses. They use the perturbative approach based on the inverse scattering transform and developed by Karpman and Solovev [53] to examine the effects

of loss on two solitons propagating down an optical fibre. They find that the presence of loss in the system will enhance the interaction of the two solitons so that they are more likely to coalesce. Taken in conjunction with the previous papers concerning soliton interactions, it can be seen that the implementation of a soliton based communication system will not be without its difficulties.

1.5.2. Other Effects and Applications.

Even with solitonic communication systems, the fact that the system in which the "soliton" is propagating does not conform exactly to one of the completely integrable equations (probably the nonlinear Schrödinger equation) means that, even if a solitonic pulse is injected into the system, by the time it has traversed the fibre it is likely that the shape of the pulse will have been degraded and part of the energy originally present within the pulse is likely now to form a pedestal around its base. In a paper by Stolen, Botineau and Ashkin [28], a method is described for removing such pedestals by utilising birefringent fibres. In such fibres, the state of polarization can be made intensity dependent so that polarizing filters can be used to separate the intense pulse from its less intense background. The use of twisted birefringent optical fibres for intensity discrimination is discussed by Winful and Hu [54].

Another paper in which the change in distribution of energy within pulse trains is discussed is given by Blow and coworkers [55]. The purpose of their investigation is to explain the generation of series of compressed pulses by amplified nonlinear dispersive systems. The equation which they consider is

$$i\frac{\partial u}{\partial z} = \tfrac{1}{2}\frac{\partial^2 u}{\partial t^2} + i\alpha\frac{\partial^3 u}{\partial t^3} + u^2u^* + i\Gamma u + i\mu\frac{\partial^2 u}{\partial t^2} \tag{1.22}$$

Here Γ is a function which measures the amplification (or loss) present in the system and μ is related to how quickly the effect decreases away from the fastest growing mode. In numerical investigations, they investigate the manner in which such a description may be used to model the trapping of energy by solitons. Essentially, therefore, the method is based on energy balance.

We have already mentioned the phenomenon of the soliton self-frequency shift observed by Mitschke and Mollenauer [23]. The theory for this effect has been given by Gordon [56]. In the paper, Gordon modifies the nonlinear Schrödinger equation to

allow for a delayed response term which can be attributed to the Raman effect which causes the effect as we have already discussed.

The Raman effect can also have the effect of breaking up higher order solitons. Using equations derived by Hasegawa and Kodama which will be discussed in the next chapter, Tai and coworkers [57] both model the pulse break up and observe the effects experimentally.

References.

1. Drazin, P.G., *Solitons*. 1 ed. London Mathematical Society Lecture Note Series, ed. James, I. M. Vol. 85. 1983, Cambridge University Press. 136.pp.

2. Russell, J.S., *Report on Waves*. Rep. 14th Meet. Brit. Assoc. Adv. Sci., York., 1844. : p. 311-390.

3. Bullough, R.K., *Solitons*. Phys. Bull., 1978. **2**: p. 78-82.

4. Dodd, R.K., *et al.*, *Solitons and Nonlinear Wave Equations*. 1st (with corrections) ed. 1984, Academic Press. 630.

5. Scott, A.C., F.Y.F. Chu, and D.W. McLaughlin, *The Soliton: A New Concept in Applied Science*. Proc. I.E.E.E., 1973. **61**(10): p. 1443-1483.

6. Bishop, A.R., J.A. Krumhansl, and S.E. Trullinger, *Solitons in Condensed Matter: A Paradigm*. Physica D, 1980. **1**: p. 1-44.

7. Bullough, R.K. and R.K. Dodd, *Solitons in Mathematics: Brief History*, in *Solitons and Condensed Matter Physics*, A.R. Bishop and T. Schneider, Editor. 1978, Springer-Verlag: Oxford. p. 2-21.

8. Noether, E., *Invariante Variationsprobleme*. Nachr. König. Gesell. Wissen. Göttingen, Math.-Phys. Kl., 1918. : p. 235-257.

9. Olver, P.J., *Applications of Lie Groups to Differential Equations.* Graduate Texts in Mathematics, ed. P.R. Halmos, F.W. Gehring, and C.C. Moore. 1986, Springer-Verlag. 497.

10. Arnold, V.I., *Mathematical Methods of Classical Mechanics.* 4th printing ed. Graduate Texts in Mathematics, ed. F.W. Gehring, P.R. Halmos, and C.C. Moore. Vol. 60. 1978, New York, Heidelberg, Berlin: Springer-Verlag. 462.pp.

11. Ablowitz, M.J. and H. Segur, *Solitons and the Inverse Scattering Transform.* 1 ed. S.I.A.M. Studies in Applied Mathematics, ed. J.A. Nohel. Vol. 1. 1981, Philadelphia: Society for Industrial and Applied Mathematics. 425.pp.

12. Newell, A.C., *Solitons in Mathematics and Physics.* CBMS-NSF Regional Conference Series in Applied Mathematics, ed. C.B.o.t.M. Sciences. Vol. 48. 1985, Society for Industrial and Applied Mathematics. 244.

13. Zakharov, V.E. and A.B. Shabat, *Exact Theory of Two-dimensional Self-focusing and One-dimensional Self-modulation of Waves in Nonlinear Media.* Sov. Phys. JETP, 1972. **34**(1): p. 62-69.

14. Chen, H.H., Y.C. Lee, and C.S. Liu, *Integrability of Nonlinear Hamiltonian Systems by Inverse Scattering Method.* Phys. Scripta, 1979. **20**: p. 490-492.

15. Lewis, Z.V., *Semiclassical solutions of the Zakharov-Shabat Scattering Problem for Phase Modulated Potentials.* Phys. Lett. A., 1985. **112**(3,4): p. 99-103.

16. Elgin, J.N. and D.J. Kaup, *Inverse Scattering Theory with Stochastic Initial Potentials.* Opt. Comm., 1982. **43**(4): p. 233-236.

17. Gowar, J., *Optical Communication Systems.* 1st ed. Prentice Hall International Series in Optoelectronics, 1984, Englewood Cliffs/Prentice Hall International.

18. Chrostowski, J. and S. Chelkowski, *Analysis of an Optical Rib Waveguide with a Nonlinear Substrate.* Opt. Lett., 1987. **12**(7): p. 528-530.

19. Marcuse, D., *Pulse Distortion in Single-mode Fibers.* Appl. Opt., 1980. **19**(10): p. 1653-1660.

20. Blow, K.J., N.J. Doran, and E. Cummins, *Nonlinear Limits on Bandwidth at the Minimum Dispersion in Optical Fibres.* Opt. Comm., 1983. **48**(3): p. 181-184.

21. Mollenauer, L.F., *Solitons in Optical Fibres and the Soliton Laser.* Phil. Trans. Roy. Soc. Lond. A, 1985. **315**: p. 437-450.

22. Mollenauer, L.F. and K. Smith, *Demonstration of Soliton Transmission over more than 4000 km in Fibre with Loss Periodically Compensated by Raman Gain.* Opt. Lett., 1988. **13**(6): p. 675-677.

23. Mitschke, F.M. and L.F. Mollenauer, *Experimental Observation of Interaction Forces between Solitons in Optical Fibres.* Opt. Lett., 1987. **12**(5): p. 355-357.

24. Menyuk, C.R., *Nonlinear Pulse Propagation in Birefringent Optical Fibres.* Opt. Lett., 1987. **23**(2): p. 174-176.

25. Menyuk, C.R., *Stability of Solitons in Birefringent Optical Fibers. I: Equal Propagation Amplitudes.* Opt. Lett., 1987. **12**(8): p. 614-616.

26. Menyuk, C.R., *Stability of Solitons in Birefringent Optical Fibers. II. Arbitrary Amplitudes.* J. Opt. Soc. Am. B, 1988. **5**(2): p. 392-402.

27. Christodoulides, D.N. and R.I. Joseph, *Vector Solitons in Birefringent Nonlinear Dispersive Media.* Opt. Lett., 1988. **13**(1): p. 53-55.

28. Stolen, R.H., J. Botineau, and A. Ashkin, *Intensity Discrimination of Optical Pulses with Birefringent Fibers.* Opt. Lett., 1982. **7**(10): p. 512-514.

29. Trillo, S., *et al.*, *Soliton Switching in Fiber Nonlinear Directional Couplers.* Opt. Lett., 1988. **13**(8): p. 672-674.

30. Leine, L., *et al.*, *Propagation of Nonlinear Film Guided Waves in a Configuration with Material Losses: a Numerical Analysis*. Opt. Lett., 1987. **12**(9): p. 747-749.

31. Hermansson, B. and D. Yevick, *Numerical Investigation of Soliton Interaction*. Electron. Lett., 1983. **19**: p. 570.

32. Yevick, D. and B. Hermansson, *New Approach to Perturbed Optical Waveguides*. Opt. Lett., 1986. **11**(2): p. 103-105.

33. Hasegawa, A. and F. Tappert, *Transmission of Stationary Nonlinear Optical Pulses in Dispersive Dielectric Fibers. I. Anomalous Dispersion*. Appl. Phys. Lett., 1973. **23**(3): p. 142-144.

34. Hasegawa, A. and F. Tappert, *Transmission of Stationary Nonlinear Optical Pulses in Dispersive Dielectric Fibers. II. Normal Dispersion*. Appl. Phys. Lett., 1973. **34**(4): p. 171-172.

35. Smith, R.G., *Optical Power Handling Capacity of Low Loss Optical Fibers as Determined by Stimulated Raman and Brillouin Scattering*. Appl. Opt., 1972. **11**(11): p. 2489-2494.

36. Cotter, D., *Fibre Nonlinearities in Optical Communications*. Opt. Quant. Electron., 1987. **19**: p. 1-17.

37. Ippen, E.P. and R.H. Stolen, *Stimulated Brillouin Scattering in Optical Fibers*. Appl. Phys. Lett., 1972. **21**(11): p. 539-541.

38. Hasegawa, A. and Y. Kodama, *Signal Transmission by Optical Solitons in Monomode Fiber*. Proc. IEEE, 1981. **69**(9): p. 1145-1150.

39. Kapron, F.P., *Maximum Information Carrying Capacity of Fibre Optic Waveguides*. Electron. Lett., 1977. **13**: p. 96-97.

40. Christodoulides, D.N. and R.I. Joseph, *Femtosecond Solitary Waves in Optical Fibers - Beyond the Slowly Varying Envelope approximation.* Appl. Phys. Lett, 1985. **47**(2): p. 76-78.

41. Bava, G.P., G. Ghione, and I. Maio, *Influence of Laser Fluctuations on Soliton Propagation in Optical Fibres.* Electron. Lett., 1984. **20**(24): p. 1002-1003.

42. Agarwal, G.S. and S. Dutta Gupta, *Effect of Nonlinear Boundary Conditions on Nonlinear Phenomena in Optical Resonators.* Opt. Lett., 1987. **12**(10): p. 829-831.

43. Desem, C. and P.L. Chu, *Soliton Propagation in the Presence of Source Chirping and Mutual Interaction in Single-mode Optical Fibres.* Electron. Lett., 1987. **23**(6): p. 260-262.

44. Desem, C. and P.L. Chu, *Effect of Chirping on Solution Propagation in Single-mode Fibers.* Opt. Lett., 1986. **11**(4): p. 248-250.

45. Gordon, J.P. and H.A. Haus, *Random Walk of Coherently Amplified Solitons in Optical Fiber Transmission.* Opt. Lett., 1986. **11**(10): p. 665-667.

46. Wai, P.K.A., *et al.*, *Nonlinear Pulse Propagation in the Neighborhood of the Zero-dispersion Wavelength of Monomode Optical Fibers.* Opt. Lett., 1986. **11**(7): p. 464-466.

47. Wai, P.K.A., *et al.*, *Soliton at the Zero-group-dispersion Wavelength of a Single-modal Fibre.* Opt. Lett., 1987. **12**(8): p. 628-630.

48. Suzuki, K., Y. Kimura, and M. Nakazawa, *Subpicosecond Soliton Amplification and Transmission Using Er^{3+} Doped Fibers Pumped by InGaAsP Laser Diodes.* Opt. Lett., 1989. **14**(16): p. 865-867.

49. Baldeck, P.L., F. Raccah, and R.R. Alfano, *Observation of Self-focusing in Optical Fibers with Picosecond Pulses.* Opt. Lett., 1987. **12**(8): p. 588-589.

50. Manassah, J.T., P.L. Baldeck, and R.R. Alfano, *Self-focussing and Self-phase Modulation in a Parabolic Graded-index Fiber*. Opt. Lett, 1988. **13**(7): p. 589-591.

51. Gordon, J.P., *Interaction Forces among Solitons in Optical Fibers*. Opt. Lett., 1983. **8**(11): p. 596-598.

52. Blow, K.J. and N.J. Doran, *Bandwidth Limits of Nonlinear (Soliton) Optical Communication Systems*. Opt. Lett., 1983. **19**(11): p. 429-430.

53. Karpman, V.I. and V.V. Solov'ev, *A Perturbational Approach to the Two-soliton Systems*. Physica D, 1981. **3**: p. 487-502.

54. Winful, H.G. and A. Hu, *Intensity Discrimination with Twisted Birefringent Optical Fibres*. Opt. Lett., 1986. **11**(10): p. 668-670.

55. Blow, K.J., N.J. Doran, and D. Wood, *Trapping of Energy into Solitary Waves in Amplified Nonlinear Dispersive Systems*. Opt. Lett., 1987. **12**(12): p. 1011-1013.

56. Gordon, J.P., *Theory of the Soliton Self-frequency Shift*. Opt. Lett., 1986. **11**(10): p. 662-664.

57. Tai, K., A. Hasegawa, and N. Bekki, *Fission of Optical Solitons Induced by Stimulated Raman Effect*. Opt. Lett., 1988. **1988**(13): p. 5.

Chapter 2

In all things, success depends on previous preparation, and without such preparation there is sure to be failure.

Confucius, 550-478 B.C..

2.1. Introduction.

We have seen that the nonlinear describing equations which give rise to solitonic behaviour currently form an important area of research. Not only may such equations be used to describe propagation of pulses in optical fibres, they may also be used for describing nonlinear processes in integrated optical devices. We have also seen how many different approaches have been used in investigating these equations in order to research different physical phenomena and have looked at the practical importance of many of them. We have seen how the mathematical technique of the inverse scattering transform of Zakharov and Shabat has been supplemented by other approaches more suited to physical problems and have looked at some of the predictions of these theories. We must now go on to look at some areas of research which are particularly closely related to the main topics of this thesis.

In carrying out this research, one of the main motivations has been to derive more exact describing equations for the propagation of light pulses in nonlinear optical media. There are already several descriptions of light pulse propagation in such systems. Some of these are quite representative of the physical situation being modelled and go to some trouble to include as many details of the effects to be modelled within the mathematical description. Others seek to introduce new effects by using quantum models of the interactions although the models produced may not be entirely realistic from physical considerations. In the next few sections, we will consider some of these models with a view both to justifying the research to be performed and to supplying the necessary background.

2.2. The Work of Hasegawa and Kodama.

We have already mentioned the two papers of Hasegawa and Tappert [1, 2] in which they suggest the use of solitons in communication systems. The importance of these papers has been stressed in that they were the first to suggest the use of solitons in communication systems. This pair of papers can, however, be regarded as the first in a series of papers by Hasegawa in which he makes a significant contribution to the field.

In fact, together with Kodama, Hasegawa has developed one of the most complete descriptions of optical pulse propagation in optical fibres - a description so complete that it might be argued that further refinement of the mathematical description of pulse propagation is no longer required. Bearing in mind the fact that the work of this thesis was carried out in order to create more accurate mathematical models for such nonlinear systems as are described by the model of Hasegawa and Kodama, it is obviously necessary to examine this model in order to compare its approach with the one presented in this thesis.

2.2.1. Earlier Work of Hasegawa and Kodama.

The aim of the next section is to describe a detailed model of pulses in fibres derived by Hasegawa and Kodama. However, each author made many studies within the area of nonlinear pulse propagation before this model was derived. In this section we will describe some of their work.

The first paper which is to be discussed concerns the use of solitons for signal transmission in optical fibres. In a 1981 paper by Hasegawa and Kodama [3], the authors describe how the use of solitons as a means of transmission of information forms a means of overcoming the limitations imposed by dispersive effects in linear systems. As we have mentioned in the last chapter, this paper basically contains a recipe for designing a nonlinear communication system which uses solitonic pulses as the information carriers.

The describing equation used in this paper is

$$i\frac{\partial q}{\partial \xi}+\frac{1}{2}\frac{\partial^2 q}{\partial \tau^2}+|q|^2 q=-i\Gamma q+i\beta\frac{\partial^3 q}{\partial \tau^3} \tag{2.1}$$

where we can see that this is a variant of the nonlinear Schrödinger equation which contains additional terms to model loss and higher order dispersion. (The interested reader can refer to the paper for the units and symbols used.)

Using the nonlinear Schrödinger equation, they describe the properties of soliton pulses in an ideal fibre. They then go on to discuss the changes in behaviour introduced by the perturbing dissipative and higher order dispersive terms. In their conclusion they present a method of designing a communication system having the maximum bit rate for the minimum power. Although some of the predictions made were later shown not to be

fully borne out in practice, the paper represents a major step forward in design methodology for communication systems.

In a subsequent series of papers by Hasegawa and Kodama different approaches for maintaining signal quality by means of periodic amplification are discussed. In the first of these, Hasegawa and Kodama [4] suggest a method whereby conventional repeaters could theoretically be eliminated by injecting a pump beam along with the signal. The use of continuous wave radiation within the fibre itself as an amplifier has been suggested in other contexts by other authors but this paper represents the first suggestion of the method in the literature. Unfortunately, the method presented here requires that the pump radiation be injected in phase with the soliton carrier beam. This would be difficult to achieve in a practical sense.

In the second paper in this series [5], the problem of the dispersive wave generated when the solitons are amplified locally is addressed. The authors find that if the separation between the repeaters is short enough, then the dispersive waves can be maintained within the soliton structure itself so that the efficiency of the system is not greatly compromised. In the third paper [6], the authors show that even if the repeaters have a random error in the magnitude of their gains, the solitonic pulses remain remarkably stable even after lengthy propagation distances.

It is also possible to avoid the limitation imposed on the repeater distances (by the dispersive waves produced by local amplification) by using a stimulated Raman process to amplify the solitonic pulses throughout the whole range of their propagation [7]. Since the amplification is essentially global, there is no sudden increase in the solitons' amplitudes and so there is no need for them to adjust by generating the dispersive waves which determined the repeater distance in the previous schemes. Instead, the repeater distance is determined by the magnitude of the loss terms. This is the topic of the final paper in the series.

In a further paper by Hasegawa [8], he models the use of the stimulated Raman process (mentioned in the last paragraph) to periodically amplify the soliton waveform. Using numerical calculations based on the nonlinear Schrödinger equation with gain and loss terms

$$i\frac{\partial q}{\partial \xi}+\frac{1}{2}\frac{\partial^2 q}{\partial \tau^2}+|q|^2 q=-i\Gamma q+i\alpha I q \tag{2.2}$$

Hasegawa carries out several calculations in which the effects of amplifier spacing, duty cycle, coupling loss and other features affect the efficiency of a nonlinear solitonic

communication system. The amplification is carried out by injecting a continuous radiation pump wave at precalculated distances and allowing the pump and soliton carrier waves to interact via the Raman effect intrinsic to the fibre. The requirement that the pump and carrier be in phase (as in the earlier papers) is therefore no longer necessary.

A further paper by Hasegawa [9] discusses the production of a train of solitonic pulses by using the modulational instability intrinsic to the fibres themselves. Hasegawa uses the nonlinear Schrödinger equation including a loss term

$$i\frac{\partial q}{\partial \xi}+\frac{1}{2}\frac{\partial^2 q}{\partial \tau^2}+|q|^2 q=-i\Gamma q \tag{2.3}$$

to model the situation in which an initially injected sine wave is externally modulated. The pulse rate and width can be modified as desired although the end result of the process still contains the initial waveform in addition to the desired pulses. There are, however, methods which will allow the separation of this background from the pulses.

2.2.2. The Detailed Model of Kodama and Hasegawa.

In the previous section and also in the first chapter we have seen that there is a wide variety of equations which can be used to describe a soliton or solitary wave propagating in a monomode fibre. The effects of third order dispersion and of dissipation are commonly added to render the models more accurate and still further terms may be added to model systems in which the emphasis of the model lies in a particular effect. What has been lacking up to this point is a derivation of a general equation which would fit any given situation and would model the propagation of a light pulse to high accuracy. Such a model has been derived by Kodama and Hasegawa.

In the first of two papers on the topic [10], Kodama describes how the derivation of a generic describing equation may be made to high order. His paper consists of three sections. In the first of these sections, Kodama derives the equation

$$i\frac{\partial q}{\partial Z}+\frac{1}{2}\frac{\partial^2 q}{\partial T^2}+|q|^2 q=\varepsilon i\left(\beta_1\frac{\partial^3 q}{\partial T^3}+\beta_2|q|^2\frac{\partial q}{\partial T}+\beta_3 q^2\frac{\partial q^*}{\partial T}\right)-i\Gamma q \tag{2.4}$$

to describe pulse propagation in a fibre. His derivation starts with Maxwell's equations, and, as can be seen, the equation derived to describe the pulses is a modified version of

the nonlinear Schrödinger equation. The equation includes terms to describe loss and higher order dispersion and, moreover, has been derived in a systematic manner using the reductive perturbation method of Taniuti and coworkers. (This method is described in Jeffrey and Kawahara's book [11].)

In his derivation, Kodama makes several important points. The first of these is that for the nonlinear problem, the electric field should not be considered to be TE mode. In many derivations, the assumption that the field may be considered to be TE is used as a simplifying assumption. Secondly, Kodama goes on to derive the full vector equations in cylindrical coordinates for the description of the wave inside the fibre. By using the reductive perturbation method, he is then able to construct a series of equations which will describe the propagation of a pulse scaled to describe the behaviour when the nonlinearity and dispersion are balanced. (The equation given is derived at the fourth order of the small parameter ε.)

Kodama's subsequent descriptions are of another perturbative technique applied to the describing equation presented above and of an investigation of the equation using the inverse scattering transform formalism. Kodama goes on to discuss the effects of the perturbing terms, those on the right hand side of (2.4), with respect to their effect on a solitonic communication system. His main conclusions are that loss is the effect most responsible for the destruction of soliton shape and that, if periodically amplified, the soliton is capable of maintaining its shape over long distances.

By themselves, these conclusions are not particularly startling. What is impressive is the mathematical rigour with which they have been derived. The mathematics involved is very complicated and somewhat impenetrable but, even so, it is obvious that the aim of the paper is to set out a derivation which is as systematic as possible.

In the second paper which must be considered [12], Kodama and Hasegawa repeat the calculation correcting an error in the earlier derivation and also make several comments on the effect of physical phenomena on the propagation of pulses described by their equation. The equation they derive is given by

$$i\left(\frac{\partial q_1}{\partial z}+k_1'\frac{\partial q_1}{\partial t}\right)-\frac{1}{2}k_1''\frac{\partial^2 q_1}{\partial t^2}-\frac{1}{6}k_1''\frac{\partial^3 q_1}{\partial t^3}+ik_i q_1$$
$$+\frac{\partial k_1}{\partial E_0^2}E_0^2 q_1+ia_1\partial/\partial t\left(q_1E_0^2\right)+ia_2q_1\frac{\partial E_0^2}{\partial t}=0 \qquad (2.5)$$

This may be written in normalised form as

$$i\frac{\partial q}{\partial Z}+\frac{1}{2}\frac{\partial^2 q}{\partial T^2}+|q|^2 q+\varepsilon i\left(\beta_1\frac{\partial^3 q}{\partial T^3}+\beta_2\frac{\partial}{\partial T}\left(|q|^2 q\right)+\beta_3 q\frac{\partial}{\partial T}\left(|q|\right)^2\right)=0 \tag{2.6}$$

where the dissipative term is actually β_3 which is complex. Its imaginary part models the loss within the guide. They also point out that this equation may be transformed into

$$i\frac{\partial Q}{\partial Z}+\frac{1}{2}\frac{\partial^2 Q}{\partial T^2}+|Q|^2 Q+\varepsilon i\beta_1\left(\frac{\partial^3 Q}{\partial T^3}+6|Q|^2\frac{\partial Q}{\partial T}\right)=O(\varepsilon^2) \tag{2.7}$$

which, if the $O(\varepsilon^2)$ terms are neglected, is a higher order nonlinear Schrödinger equation which may be shown to be completely integrable and thus have solitonic solutions.

They investigate these equations. Their main findings are that the higher order dispersive terms serve to split higher order solitons and that the effect of loss is to produce a downward shift in the frequency of the carrier. This is interpreted as being due to the Raman process mentioned earlier. They also find that their results agree well with experiment.

The importance of these equations can be judged by the fact that they are then used as the starting point for further investigations. In a paper by Potasek [13] the equations derived by Kodama and Hasegawa are used to investigate the phenomenon of modulation instability. Potasek finds that the presence of loss actually diminishes the effect of modulation instability but that the retardation term, the additional term added to correct Kodama's first paper, significantly alters the effect. He points out that for a certain value of the derivative of the nonlinear term, the modulation instability disappears.

Kodama himself (together with Nozaki) goes on to use the equations derived to investigate soliton interaction in fibres [14]. They find, using both analytic and numerical methods, that the effect of higher order terms is to split higher order solitons into first order solitons travelling with different velocities. They also state that their results are in close agreement with those obtained by experiment.

In a paper by Tai, Hasegawa and Bekki [15] the effect of the stimulated Raman effect term mentioned earlier is investigated.They find that the Raman term will lead to soliton fission with the solitons being ejected in different directions so as to obey

"momentum" conservation. Thus the separation of the pulses is actually larger than that predicted earlier on the basis only of the carrier frequency shift induced by this term.

2.2.3. Implications of Kodama and Hasegawa's Papers.

It has already been mentioned that the results derived by Hasegawa and Kodama in the two papers given prevalence in the last section [10, 12] are derived with a high degree of mathematical rigour. What has not been fully discussed is the implications that these two papers have on the work to be performed in this thesis.

It has already been stated that one of the main reasons for undertaking this work was to derive the describing equations for a pulse in a nonlinear fibre in a more systematic manner. Yet here is a paper which, aside from the assumptions that the fibre is monomode and polarization preserving, derives, accurately and systematically, an equation which describes pulse propagation in a nonlinear fibre to high order. The model which we will present considers nonlinear wave propagation in a semi-infinite medium without boundary conditions and requires the existence of a lagrangian description of the system before calculations can even begin.

It is obviously necessary to contrast the two methods in order to justify further work on the problem. Firstly, the manner in which the model equations is derived from the Maxwell equations is hard to fault. Their assumptions that the fibre is monomode and polarization preserving are hardly restrictive. The use of the reductive perturbation method to derive the equations means that they have been derived systematically using a method that would be amenable to implementation as a computer algebra program. In fact, were higher orders to be required, it would not be impossible to determine them using such a method.

If the derivation of the equations does contain a flaw, it is this: the model used to derive the equations is a macroscopic one[1]. It would be quite difficult to modify the derivation given by Kodama and Hasegawa to model quantum effects for example. The description of the medium is inserted into the behaviour of the electric field. If the medium were to change in type from classical to quantized, it is difficult to see how the model could be easily modified. This does in no way imply that the description given by Kodama and Hasegawa is wrong. The point to be made is that, given such a complicated description, it is model-dependent in a manner in which the lagrangian description is not. Essentially the same manipulations are carried out in the quantized

[1] A. P. Ansbro. Private Communication.

and classical cases in the lagrangian description. In the Kodama model, the manipulations would be different. In the final analysis, it is likely that the same results could be obtained by either method. For the purposes of describing a pulse in a nonlinear fibre, the description given by Kodama and Hasegawa will be difficult to better. As a method to be applied to conceptually different physical models, it presents some difficulty.

Perhaps the matter might be better expressed as there being two tools for two different jobs. For the investigation of pulses in a classical nonlinear fibre, the description given by Hasegawa and Kodama is accurate and useful. As a means of investigating the effects of different materials and material models on the propagation characteristics, the lagrangian formalism is to be preferred. Although the lagrangian technique's description of the modal properties of the fibre is nonexistent (so far) it is suited to the adjustment of the model describing the material with a view to observing the changes in the equations describing the pulse. In this way, the two techniques can be considered to be separate attempts to describe similar situations which have different strengths and different weaknesses.

2.3. The Maxwell-Bloch Equations and their Variants.

In our previous discussion, we have mentioned that one may extend the validity of models of processes in nonlinear media by including quantum effects. Although it is possible to construct models which are fully quantized in nature - that is both medium and electromagnetic radiation are quantized - such models tend to be somewhat impractical for predicting effects in physical situations. Furthermore, the additional complexity introduced by quantization of the light field tends rather to obscure the effects being sought. The additional rigour is not warranted for the problems of interest to us since the mathematical complexity and difficulty of solution add little to the types of solution to be found. Instead most authors are content to use a semiclassical approach in which the atomic field is quantized but the light field is not. The foremost model in this area is given in terms of the Maxwell-Bloch equations.

In this section of the thesis, we will go on to discuss the Maxwell-Bloch equations and subsidiary equations which may either be derived from it or are closely related to it in that they describe special cases within the physical situations described by the Maxwell-Bloch equations. We will discuss research performed in deriving solitonic

solutions to these equations (if they exist) and will also describe how the assumptions inherent in the derivation of the Maxwell-Bloch equations can lead to the equations becoming invalid under certain experimental conditions.

2.3.1. Bloch-type Equations.

There is a multitude of equations which describe optical pulse propagation in nonlinear media. As has already been mentioned, the most general of these are those known as the Maxwell-Bloch or Optical Bloch equations. By making some fairly reasonable simplifying assumptions, one can derive a set of equations known as the reduced Maxwell-Bloch equations which can be shown to be completely integrable whereas the full Maxwell-Bloch equations are not. The self-induced transparency or S.I.T. equations are a related set of equations which are of importance in describing nonlinear phenomena. It may be shown that these equations are also related to the nonlinear Schrödinger equation and the sine-Gordon equation in appropriate limits. In other words, there is a hierarchy of related equations which describe electromagnetic pulse propagation in nonlinear media with greater or less accuracy, some of these equations being valid only in particular limits.

Essentially, the situation being modelled is as shown in diagram 2.1. The electromagnetic radiation is modelled as a classical wave; the medium is, however, modelled as a quantized medium having two distinct quantum levels, a ground state and an excited state. The electromagnetic radiation will induce electronic transitions between these two states according to a coupling term which is introduced to model the interaction between the medium and the radiation. This is, of course, a very simplified description of the manipulations to be carried out. We will now go on to look at a derivation of the Maxwell-Bloch equations in detail. This will prove useful when we later wish to look at nonlinear wave propagation in nonlinear quantized media in terms of our lagrangian model.

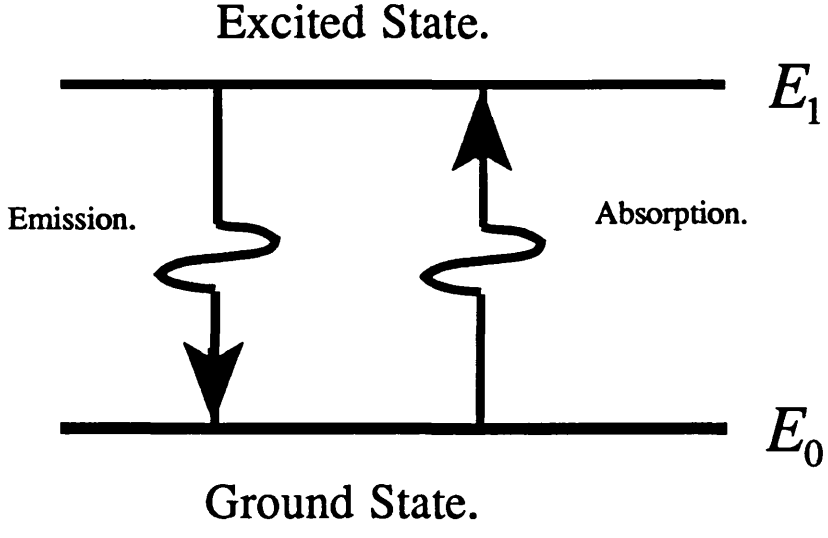


Diagram 2.1
A diagram of the quantum model.

2.3.2. The Maxwell-Bloch Equations.

In this section we will use the techniques of quantum mechanics to derive a set of equations which will describe the propagation of electromagnetic radiation through a quantized medium which is coupled to the radiation by means of some dipole interaction. It is assumed that the reader is familiar with the techniques and terminology of quantum mechanics: there is a multitude of standard texts which the interested reader may consult if necessary, for example Mathews and Venkatesan [16]. The use of density matrices is also discussed in standard texts and in the paper by Fano [17]. In any case, the derivation we will be following is that given in the book by Dodd, Eilbeck, Gibbon and Morris [18]. An alternative presentation of the derivation can be found in the book by Allen and Eberly [19].

We shall now derive the Maxwell-Bloch equations describing electromagnetic wave propagation in a two-level atomic medium. It is assumed that the physical situation is as given above in diagram 2.1 with the two energy levels being labelled E_0 and E_1 such that $E_1 = E_0 + \hbar\omega$. It is further assumed that the eigenfunctions for the two energy levels are known and given by the functions ϕ_0 and ϕ_1. Accordingly we can write

$$\hat{H}_0\phi_0 = E_0\phi_0 \tag{2.8}$$

and

$$\hat{H}_0\phi_1 = E_1\phi_1 \tag{2.9}$$

where $\hat{H}_0$ is the hamiltonian operator which is assumed to be time-independent. The influence of the external classical field is then introduced by modifying the hamiltonian

$$\hat{H} = \hat{H}_0 + \hat{H}_I \tag{2.10}$$

where $\hat{H}$ is the total hamiltonian and $\hat{H}_I$ is the interaction hamiltonian which is assumed to be explicitly time dependent. The final assumption made is that ϕ_0, ϕ_1 can be used as a basis for constructing the time-dependent wave function so that the wave function of the time dependent problem can be written as

$$\psi = a_0\phi_0 + a_1\phi_1 \tag{2.11}$$

The time dependent Schrödinger equation can then be written as

$$\hat{H}\psi = i\hbar\frac{\partial\psi}{\partial t} \tag{2.12}$$

The coefficients a_i are the probability amplitudes.

The substitution of (2.11) in (2.12) followed by multiplication from the left by ϕ_0, and then, separately, ϕ_1 gives the following two expressions

$$i\hbar\frac{\partial a_0}{\partial t} = \sum_{m=0}^{1}\tilde{H}_{0m}a_m \tag{2.13}$$

$$i\hbar\frac{\partial a_1}{\partial t} = \sum_{m=0}^{1}\tilde{H}_{1m}a_m \tag{2.14}$$

which can be combined to give

$$i\hbar\frac{\partial a_n}{\partial t} = \sum_{m=0}^{1}\tilde{H}_{nm}a_m \tag{2.15}$$

We can show that the expectation value of some operator $\hat{A}$ is (see appendix 1)

$$\begin{aligned}\left\langle \hat{A} \right\rangle_{\psi} &= \langle \psi | \hat{A} | \psi \rangle \\ &= \sum_{m,n=0}^{1} a_n^* a_m \langle \phi_n | \hat{A} | \phi_m \rangle\end{aligned} \tag{2.16}$$

which, for some observable $\hat{A}$, may be expressed in matrix form as

$$\left\langle \hat{A} \right\rangle_{\psi} = \mathrm{Tr}(\tilde{\rho}\tilde{\mathbf{A}}) \tag{2.17}$$

The self-adjoint matrices $\tilde{\mathbf{A}}_{nm}$ and $\tilde{\rho}_{nm}$ are given by

$$\tilde{\mathbf{A}}_{nm} = \langle \phi_n | \hat{A} | \phi_m \rangle \tag{2.18}$$

and

$$\tilde{\rho}_{nm} = a_n a_m^* \tag{2.19}$$

It may be shown (Appendix 2) that

$$i\hbar \frac{\partial a_n}{\partial t} = \sum_{m=0}^{1} \tilde{\mathrm{H}}_{nm} a_m \Rightarrow i\hbar \frac{\partial \tilde{\rho}}{\partial t} = \left[\tilde{\mathrm{H}}, \tilde{\rho}\right] \tag{2.20}$$

and the implied equation is known as the Liouville equation. (The square brackets here indicate the commutator.)

We now use the Pauli (spin) matrices

$$\tilde{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \quad \tilde{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}; \quad \tilde{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}; \tag{2.21}$$

together with the 2 by 2 unit matrix $\mathbf{I}_2$ and the result

$$(\sigma.\mathbf{A})(\sigma.\mathbf{B}) = (\mathbf{A}.\mathbf{B})\mathbf{I}_2 + i\sigma.(\mathbf{A} \times \mathbf{B}) \tag{2.22}$$

to reformulate the problem. Here σ is the column vector of matrices given by

$$\sigma = \begin{pmatrix} \tilde{\sigma}_x \\ \tilde{\sigma}_y \\ \tilde{\sigma}_z \end{pmatrix}$$

(2.23)

We find it useful to write the hamiltonian and density matrices as (see Appendix 3)

$$\tilde{H} = \tfrac{1}{2}\hbar(\omega_0 I_2 + \omega.\sigma)$$

(2.23)

and

$$\tilde{\rho} = \tfrac{1}{2}(I_2 + \rho.\sigma)$$

(2.24)

where ω_0 is the number

$$\omega_0 = \frac{1}{\hbar}(\mathrm{Tr}\tilde{H})$$

(2.25)

and ρ and ω are vectors given by

$$\rho = \mathrm{Tr}(\tilde{\rho}\sigma)$$

(2.26)

and

$$\omega = \frac{1}{\hbar}\mathrm{Tr}(\tilde{H}\sigma)$$

(2.27)

respectively. We can then show that the Liouville equation (2.20) can be written as

$$\tfrac{1}{2}i\hbar\frac{\partial\rho}{\partial t}.\sigma = \tfrac{1}{2}i\hbar(\omega\times\rho).\sigma$$

(2.28)

or alternatively as

$$\frac{\partial\rho}{\partial t} = \omega\times\rho$$

(2.29)

and this is the compact form of the Bloch equations without broadening. Much of the simplification in the notation is achieved by the use of the Pauli spin matrices as a basis representation for the motion: this will be of interest in a future section on the work of

Eberly and Hioe. The equations are, however, not in a particularly useful form. To rectify this we need to use a more concrete description of the interaction term.

The total hamiltonian was defined as

$$\hat{H} = \hat{H}_0 + \hat{H}_I \tag{2.30}$$

and we now define the interaction term, $\hat{H}_I$, as

$$\hat{H}_I = -\mathbf{E}.\hat{\mathbf{P}} \tag{2.31}$$

where $\hat{\mathbf{P}}$ is the dipole operator given by

$$\hat{\mathbf{P}} = -e\hat{\mathbf{x}} \tag{2.32}$$

We note that the macroscopic polarization $\mathbf{P}$ is given by the expectation value of the polarization operator

$$\begin{aligned} \mathbf{P} = \langle \hat{\mathbf{P}} \rangle &= \mathrm{Tr}(\rho\tilde{\mathbf{P}}) \\ &= \mathrm{Tr}(\rho\mathcal{P}\tilde{\sigma}_x) \\ &= \mathcal{P}\mathrm{Tr}(\rho\tilde{\sigma}_x) = \mathcal{P}\rho_1 \end{aligned} \tag{2.33}$$

We note here that the phases of the wavefunctions have been chosen to render the components of the polarization operator $\hat{\mathbf{P}}$ real. If this had not been done, the imaginary component of the polarization would have required us to use $\tilde{\sigma}_y$ in the matrix representation of the polarization matrix $\tilde{\mathbf{P}}$.

Using this information, we can then modify the representation of the total hamiltonian given in (2.30). We write the hamiltonian matrix as

$$\tilde{H} = \begin{bmatrix} E_0 + \hbar\omega & 0 \\ 0 & E_0 \end{bmatrix} - \mathbf{E}.\mathcal{P}\tilde{\sigma}_x \tag{2.34}$$

which we can represent in terms of Pauli matrices as

$$\tilde{H} = \tfrac{1}{2}(2E_0 + \hbar\omega)\mathbf{I}_2 + \tfrac{1}{2}\hbar\omega\tilde{\sigma}_z - \mathbf{E}.\mathcal{P}\tilde{\sigma}_x \tag{2.35}$$

If we remember that the vector ω can be written as

$$\omega = \frac{1}{\hbar}\mathrm{Tr}(\tilde{\mathrm{H}}\sigma) \tag{2.36}$$

then we see that we can write ω as

$$\omega = \begin{pmatrix} -2/\hbar \mathbf{E}.\mathcal{P} \\ 0 \\ \omega \end{pmatrix} \tag{2.37}$$

If we now refer to the compact form of the Bloch equations (2.29), we see that, if we write the vector ρ as

$$\rho = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{pmatrix} \tag{2.38}$$

and use the definition of the vector cross product to write

$$\omega \times \rho = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -2/\hbar \mathbf{E}.\mathcal{P} & 0 & \omega \\ \rho_1 & \rho_2 & \rho_3 \end{vmatrix} \tag{2.39}$$

then the final form of the Bloch equations we require is

$$\begin{aligned} \frac{\partial \rho_1}{\partial t} &= -\omega \rho_2 \\ \frac{\partial \rho_2}{\partial t} &= \omega \rho_1 + 2/\hbar \mathbf{E}.\mathcal{P} \rho_3 \\ \frac{\partial \rho_3}{\partial t} &= -2/\hbar \mathbf{E}.\mathcal{P} \rho_2 \end{aligned} \tag{2.40a,b,c}$$

In order to then model the electric field component, we must also add an additional equation describing the electric field. This is derived from Maxwell's equations as

$$\begin{aligned} \frac{\partial^2 \mathbf{E}}{\partial t^2} - c^2 \nabla^2 \mathbf{E} &= -\frac{N_0}{\varepsilon_0} \frac{\partial^2 \mathbf{P}}{\partial t^2} \\ &= -\frac{N_0 \mathcal{P}}{\varepsilon_0} \frac{\partial^2 \rho_1}{\partial t^2} \end{aligned} \tag{2.41}$$

where we have simply assumed that the classical polarization can be replaced by the expectation value of the microscopic quantum polarization times the density of the atomic sites N_0 so that the final set becomes

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t} &= -\omega \rho_2 \\
\frac{\partial \rho_2}{\partial t} &= \omega \rho_1 + \tfrac{2}{\hbar}\mathbf{E}.\mathcal{P}\rho_3 \\
\frac{\partial \rho_3}{\partial t} &= -\tfrac{2}{\hbar}\mathbf{E}.\mathcal{P}\rho_2 \\
\frac{\partial^2 \mathbf{E}}{\partial t^2} - c^2\nabla^2\mathbf{E} &= -\frac{N_0\mathcal{P}}{\varepsilon_0}\frac{\partial^2 \rho_1}{\partial t^2}
\end{aligned}$$

(2.42a,b,c,d)

It is this set which is known as the Maxwell-Bloch equations. Slight differences arise if cgs units are used. The equations have also been modified from those appearing in Dodd, Eilbeck, Gibbon and Morris's book [18] to make them compatible with other authors' results and to correct some sign errors.

There is an additional reason why we have carried out these manipulations. The use of spin matrices in forming a basis set for the representation considerably simplified the analysis. We will see in a subsequent section how such techniques can be extended to consider atoms having a greater number of quantum levels and we will subsequently suggest how such representations might be used within the context of a lagrangian model of nonlinear wave propagation.

We point out here that the derivation so far presented can be found in papers by Feynman [20] and Bullough [21] as well as in the books by Dodd, Eilbeck, Gibbon and Morris [18] and by Allen and Eberly [19]. However, the emphasis in each of these references is slightly different. The book by Dodd goes on to show how the Maxwell-Bloch equations can then be reduced to the sine-Gordon equation which we have already pointed out is a completely integrable equation having solitonic solutions. The book by Allen and Eberly goes on to use rotating observation frames to simplify the analysis and to show how the introduction of phenomenological relaxation times can be used to model effects leading to line broadening. Feynman's paper uses the Schrödinger picture in the derivation rather than the Heisenberg one: the results obtained are the same. We shall refer to the paper by Bullough, Caudrey, Eilbeck and Gibbon [21] since it has relevance to the next section of work.

After deriving the equations, Bullough *et al* go on to make some important points about the assumptions inherent in the model. They state the following.

Important physics is missing:

1). damping, radiative or relaxational;

2). atomic structure - many levels, degeneracy;

3). molecular or grosser structure - crystalline order, local molecular correlation leading to local fields;

4). molecular motions - Doppler motions in gases and vapours, phonon interactions in solids;

5). effects of surfaces (at high density $\gtrsim 10^{18}$ *atoms per c.c.only);*

6). 'real pulses' which include diffraction effects due to finite beam apertures, three-dimensional propagation.

They also state that the theory ignores correlation effects such as those between electrons on adjacent atoms or between the actual atoms themselves. The theory is therefore a better description of the physical situation when modelling situations having a low density of atoms since under such conditions the correlations between the atoms are minimized.

The theory can, however, be modified to take some of the relaxational and dissipational effects into account. This is done by introducing the time constants T_1, T_2, T_2^*. The constants T_1 and T_2 are due to homogeneous broadening whereas that referred to as T_2^* is due to inhomogeneous broadening. The term T_1 is used to model the way in which the excited atoms decay from the excited state to the ground state. The term T_2 is used to describe the decay of the dipole moments which can be different from the decay time of the population inversion. This is because processes such as collisions in a gas or phonon scattering in a solid can affect the dipole oscillations of the atoms without affecting their energy, that is, without affecting whether they are in the ground or excited states. The terms T_1 and T_2 are called the homogeneous lifetimes since they affect all of the atoms equally. The term T_2^* however, is called the inhomogeneous lifetime since it models the decay rate of processes which are local in nature such as the Doppler effect in gases or strain effects in solids. The inhomogeneous damping arising from these effects results in a spread of resonant frequencies and thus the oscillations of the dipoles of each of the atoms will tend to become out of phase with each other.

If the spread of resonant frequencies ω_s has a distribution $g(\omega_s)$ which is normalised so that

$$\int_0^{\infty} g(\omega_s)\mathrm{d}\omega_s = 1$$

(2.43)

where the width of $g(\omega_s)$ is $1/T_2^*$, then we can write a new set of equations where the describing functions ρ_1,ρ_2,ρ_3 all now become functions of the resonant frequency ω_s. This is because we can think of the function $g(\omega_s)$ as defining the probability that a particular atom will have a particular resonant frequency ω_s. The macroscopic dipole will therefore require to be averaged over this distribution so that we can write

$$\mathbf{P}_{mac} = N_0 \mathcal{P} \int_0^{\infty} g(\omega_s)\rho_1 \mathrm{d}\omega_s = N_0 \mathcal{P}\langle \rho_1 \rangle$$

(2.44)

where $\mathbf{P}_{mac}$ is the macroscopic polarization and the function ρ_1 is now to be considered a function of the resonant frequencies ω_s. The Maxwell-Bloch equations can therefore be modified as follows

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t} &= -\omega_s \rho_2 \\
\frac{\partial \rho_2}{\partial t} &= \omega_s \rho_1 - 1/T_2 \rho_2 + 1/\hbar \mathbf{E}.\mathcal{P}\rho_3 \\
\frac{\partial \rho_3}{\partial t} &= -(1+\rho_3)1/T_1 - 1/\hbar \mathbf{E}.\mathcal{P}\rho_2 \\
\frac{\partial^2 \mathbf{E}}{\partial t^2} - c^2\nabla^2\mathbf{E} &= -\frac{N_0 \mathcal{P}}{\varepsilon_0}\frac{\partial^2 \langle \rho_1 \rangle}{\partial t^2}
\end{aligned}$$

(2.45a,b,c,d)

The electric field is independent of the resonant frequency ω_s and should not be thought of as a local variable in the same sense as ω_s.

The above equations (2.45) when reduced to one dimension and time are referred to as the phenomenologically broadened Maxwell-Bloch equations. It is possible to show that T_1 and T_2 can be related to the Einstein A coefficient for spontaneous emission provided that the electric field is coherent. This is stated in the paper by Bullough [21] as well as the fact that for pulses of picosecond duration T_1 and T_2 can be ignored.

Having derived these equations, one needs then to go on to find their solutions. If we refer briefly to the form of equation (2.29), we can see that it has the same form as that for a charged spinning top in a magnetic field [22]. Alternatively, we can say that the equation refers to the precession of a spinning body acted on by a known torque ω.

The vector ρ is being acted on by the torque ω. If we refer to the components of the vector ω given in (2.37) reproduced below

$$\omega = \begin{pmatrix} -2/\hbar \mathbf{E}.\mathcal{P} \\ 0 \\ \omega \end{pmatrix} \tag{2.46}$$

we see that the torque vector has two non-zero components. It can be shown, see Allen and Eberly [19], that the first of these components is always smaller than the third for all physical situations of interest. If the external field is strong enough to make the two components of approximately equal magnitudes, then the external field must be of the same order of magnitude as the binding field for the electron. In other words, if the two components are of equal magnitude, the application of the field will ionise the atoms and the model will cease to be meaningful. This therefore implies that the vector ω points roughly in the direction of ρ_3 and using this will allow us to considerably simplify the analysis.

The technique used is to simplify the Bloch part of the Maxwell-Bloch equations by rewriting them in a rotating coordinate frame. The Maxwell part of the equations can be simulated by using a harmonic field in the dipole term. This is explained by Allen and Eberly [19]. They give the form of the Bloch equations as

$$\begin{aligned} \frac{\partial s_1}{\partial t} &= -\omega_0 s_2 \\ \frac{\partial s_2}{\partial t} &= \omega_0 s_1 + \kappa E s_3 \\ \frac{\partial s_3}{\partial t} &= -\kappa E s_2 \end{aligned} \tag{2.47}$$

These equations are essentially identical to those obtained before. The coefficient κ can be thought of as being given by

$$-2/\hbar \mathbf{E}.\mathcal{P} = -\kappa E \tag{2.48}$$

The electric field is assumed to be a harmonic of frequency ω almost resonant with the transition frequency ω_0. That is

$$E = E(t)\left[e^{i\omega t} + c.c.\right] \tag{2.49}$$

This is where we can apply physical arguments to the simplification. We have already stated that the vector given by ω, or in their notation where Ω is given by

$$\Omega = \begin{pmatrix} -\kappa E \\ 0 \\ \omega_0 \end{pmatrix} \tag{2.50}$$

points in roughly the same direction as s_3 or ρ_3 in our notation. We can rewrite the vector Ω as the sum of three other vectors

$$\Omega = \Omega^+ + \Omega^- + \Omega^0 \tag{2.51}$$

where

$$\begin{aligned} \Omega^0 &= (0,0,\omega_0) \\ \Omega^+ &= (-\kappa E\cos\omega t, -\kappa E\sin\omega t, 0) \\ \Omega^- &= (-\kappa E\cos\omega t, \kappa E\sin\omega t, 0) \end{aligned} \tag{2.52a,b,c}$$

The terms Ω^+ and Ω^- are small by our physical argument so that Ω^0 is the main term. If we think of the vector **s** as precessing about the 3-axis, then Ω^+ and Ω^- will rotate in opposite directions at angular velocity ω. Therefore, if one uses a rotating coordinate frame which rotates in the same direction as Ω^+ then Ω^- will rotate in the opposite direction at angular velocity 2ω. In such a case, the effect of Ω^- will be reversed at twice the light frequency and can essentially be ignored. This is called the rotating wave approximation. If we ignore Ω^- the equations (2.52) become

$$\begin{aligned} \frac{\partial s_1}{\partial t} &= -\omega_0 s_2 - \kappa E s_3 \sin\omega t \\ \frac{\partial s_2}{\partial t} &= \omega_0 s_1 + \kappa E s_3 \cos\omega t \\ \frac{\partial s_3}{\partial t} &= -\kappa E\left[s_2\cos\omega t - s_2\sin\omega t\right] \end{aligned} \tag{2.53a,b,c}$$

We define a vector ρ' having components u, v, w in the rotating frame by

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \cos\omega t & \sin\omega t & 0 \\ -\sin\omega t & \cos\omega t & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \tag{2.55}$$

We can therefore rewrite the equations of motion for the components of ρ' as

$$\frac{\partial u}{\partial t} = -(\omega_0 - \omega)v$$

$$\frac{\partial v}{\partial t} = +(\omega_0 - \omega)u + \kappa E w$$

$$\frac{\partial w}{\partial t} = -\kappa E v \tag{2.56a,b,c}$$

These equations can again be written as a single vector equation

$$\frac{\partial \rho'}{\partial t} = \Omega' \times \rho' \tag{2.57}$$

where

$$\Omega' = (-\kappa E, 0, \omega_0 - \omega) \tag{2.58}$$

We can see by examination of (2.57) and (2.58) that all of the optical frequencies have been removed from the problem by making the rotating wave approximation if the assumption is made that the frequency of the applied electric field and the transition frequency are similar. We can also see from (2.56) that the components u and v are components of the polarization. Since v is coupled to w, it must relate to the change in population of the two levels. The component w relates to the energy of the atoms. It determines the relative probability that an atom will occupy the ground or excited state. The final component u is that part of the polarization which is out of phase with the driving field and will therefore be dispersive in its effect. One can also show that the conservation of probability implies that

$$u^2 + v^2 + w^2 = 1 \tag{2.59}$$

The equations (2.56) can be modified to take into account the loss mechanisms mentioned previously. The equations become

$$\frac{\partial u}{\partial t} = -\Delta v - \frac{u}{T_2}$$
$$\frac{\partial v}{\partial t} = \Delta u - \frac{v}{T_2} + \kappa E w$$
$$\frac{\partial w}{\partial t} = -\frac{w - w_{eq}}{T_1} - \kappa E v$$

(2.60)

Here Δ is the detuning, and w_{eq} is the equilibrium value of the inversion w. Provided the value of the amplitude of the electric field E remains constant these equations may be solved exactly since they are linear first-order differential equations. This work was carried out by Torrey [23] who presented detailed solutions of the equations by means of Laplace transform techniques.

In the case of the Bloch equations without broadening (2.56), provided the amplitude of the electric field retains a constant value, the equations may also be solved analytically. Such a solution was given by Rabi [24]. By means of a complicated transformation, it is possible to find a rotating frame in which the components of the vector which describe the motion are stationary. In this way, it is possible to describe the motion of the vector components u, v, w as observed in the original rotating frame. If an atom starts initially in its ground state so that we can write

$$u_0 = v_0 = 0, w_0 = -1$$

(2.61)

then the general solution can be restricted to give

$$w = -1 + \frac{2(\kappa E)^2}{(\kappa E)^2 + \Delta^2} \sin^2 \sqrt{(\kappa E)^2 + \Delta^2} \frac{t}{2}$$

(2.62)

By examining this formula, we can see that, given a sufficiently small detuning Δ, quite a large proportion of the population will be excited to the upper level in each cycle.

It should not be thought that this set of equations represents the only way to proceed. In a subsequent section, it will be shown how a modified set of equations, the reduced Maxwell-Bloch equations can be solved using inverse scattering transform techniques. However, if one wishes to consider direct solutions, the papers by Smith [25, 26] may prove to be of interest. In them he discusses numerical solutions of broadened Maxwell-Bloch equations, in the first case under conditions of homogeneous

broadening, and in the second case under conditions of inhomogeneous and mixed broadening.

The Maxwell-Bloch equations have also been solved using a power-series solution method by Matulic and Palmer [27]. The method they introduce reproduces already known solutions but also gives rise to new solutions which are dependent on the initial conditions used. The method requires the simultaneous solution of large numbers of differential equations and thus the authors use numerical techniques to obtain their results.

Finally, a series of three papers by Zi-zhao and Guo-zhen [28-30] considers the possibility that equations similar to the Maxwell-Bloch equations could be used to model light propagation in semiconductors. The conclusion of the first of these papers [28] is that the interband transition in semiconductors can be described by an equation which is formally identical to the Bloch equations for a two-level system provided that the interaction between the electrons may be neglected. The second paper [29] discusses the similarity between excitons in semiconductors and excited atoms in the simple Bloch model whilst the third [30] goes on to suggest that self-induced transparency (S.I.T.) can be shown in semiconductors contrary to the predictions of other workers.

2.3.3. Why the Maxwell-Bloch Equations May Become Invalid.

It was mentioned previously that there were situations in which the Maxwell-Bloch equations fail to correctly describe physical situations. This failure is in addition to the simplifications involved in reducing the problem to one which is semi-classical in nature. This failure has been pointed out in a series of papers and it is our purpose to review these papers here. The intention is to point out that, although the Maxwell-Bloch equations do model a wide variety of phenomena correctly, there are situations in which their use is not justified and in which they give results which are not consistent with experiment. It will therefore be realised that any equations derived from the Maxwell-Bloch equations, or closely related to them, will have these limitations inherent within them since they are present within the structure of the model itself.

Although the failure of the Bloch equations had been pointed out before in the literature, it had been tacitly assumed by many researchers that the equations could be considered to be universally valid. However, as a result of experiments carried out by

DeVoe and Brewer [31] on low temperature ion impurities within Lanthanum Fluoride, the failure of the Bloch equations was brought to the attention of researchers.

In the previous section, it was pointed out that the Maxwell-Bloch equations could be extended from the form derived from the Liouville equations to include relaxation processes by including two relaxation times, T_1 and T_2. The first of these relaxation times relates to the rate at which excited electrons will relax down from the excited state to the ground state. The second of these relaxation times relates to the manner in which oscillations in the electron population between the two levels become out of phase with respect to each other. DeVoe and Brewer's paper makes an experimental test of the Bloch equations under conditions in which the relaxation times, assumed to be constants, become dependent upon the laser intensity. Under such conditions, it it unremarkable that the Bloch equations fail.

After the publication of this paper, further references were published which attempted to correct the Maxwell-Bloch equations or to give reasons and explanations for their invalidity. One of the first of these is to be found in a paper by Yamanoi and Eberly [32] (although see the next reference). In this paper, the authors derive alternative equations similar to the Maxwell -Bloch equations. The modifications made allow the equations to describe situations in which the intensity of the applied light is high and begins to saturate the medium. The authors then make some simplifying assumptions which allow them to solve the equations to obtain expressions which agree well with the experimental data given by DeVoe and Brewer.

In an earlier paper on the subject by Yamanoi and Eberly [33] in which they present an abbreviated version of their results, they present results which are the subject of comments by DeVoe and Brewer. DeVoe and Brewer assert that, although they agree that Yamanoi and Eberly have in principle modelled the physical situation correctly, in their neglect of terms they oversimplify the equations and calculate results which are contradictory to those of DeVoe and Brewer's own theories.

In a paper by Berman [34], validity conditions for the Maxwell-Bloch equations are calculated. In his paper, Berman uses probability theory to model the relaxation processes and finds that, even under conditions in which interactions between atoms can be assumed to take place in infinitesimally short times, the Bloch equations are still invalid because of modifications to the transition frequency caused by interactions between the atomic system and the perturber bath - perturbations introduced to cause the relaxation processes. Berman points out firstly that the Bloch equations can fail under conditions in which there is no incident radiation and secondly that, in his opinion, the

Bloch equations might be better considered as a set of equations which will describe two-level atoms interacting with an electromagnetic field only in exceptional limiting circumstances.

In a subsequent paper [35], Berman goes on to use the theories presented in the previous paper and to apply them to various experimental situations including that pertaining to the experiments performed by DeVoe and Brewer. His analysis is extremely mathematically complex but his equations do lead to results which depart from those of the Maxwell-Bloch equations. However, his solutions are obtained perturbatively and cannot therefore be directly compared with the experiment of DeVoe and Brewer since the experiment used strong fields which cannot be modelled correctly by perturbative analysis.

An additional reference in which a situation outwith the realm of validity of the Maxwell-Bloch equations is considered is presented in a paper by Singh and Agarwal [36]. They present numerical calculations in which they model a nonlinear process, four-wave mixing, under conditions which render the Bloch equations invalid. Again the importance of the relaxation processes in determining the validity of the Bloch equations is emphasized.

2.3.4. The Hierarchy of Equations of Maxwell-Bloch Type

We remarked previously that an attempt to solve the Maxwell-Bloch equations directly was not the only way to proceed. Instead, by using some reasonable simplifying assumptions, it is possible to derive a similar set of slightly simpler equations which have been called the reduced Maxwell-Bloch equations. By making further assumptions and simplifications one can derive other equations, the self-induced transparency equations and the sine-Gordon equation. It can further be shown that these equations, together with the reduced form of the Maxwell-Bloch equations, admit solitonic solutions.

It is our purpose in the next few sections to discuss these equations since they are related to the lagrangian model which will subsequently be derived. Most of the references were written by a group working at the University of Manchester's Institute of Science and Technology in the late 1960's and early 1970's under Professor R. K. Bullough.

2.3.4.1 The Reduced Maxwell-Bloch and Self-Induced Transparency Equations

The Maxwell-Bloch equations are used to describe an electromagnetic wave propagating in a field of two-level atoms. We have derived the equations and have indicated how they may be solved given certain simplifying assumptions. A paper of further interest is that by Eilbeck and Bullough [37] in which they discuss whether the Maxwell-Bloch equations are physically reasonable and show that they are incapable of producing optical shocks. However, as is stated in the paper by Bullough and coworkers [21], the only known analytic pulse solutions of these equations are hyperbolic secants having both extreme intensity and narrow width. These solutions are not of physical interest. It therefore makes sense to simplify the equations to allow their solution for wave types which are known to exist by experiment. Furthermore, it has been shown that the Maxwell-Bloch equations do not form a completely integrable system since numerical experiments have shown that pulses colliding according to the Maxwell-Bloch equations do not conserve their identity and should therefore not be considered as solitons [38].

Bearing this in mind, it is common to simplify the Maxwell-Bloch equations by making some assumptions about the behaviour of the physical systems involved. At a first level of simplification this results in a set of equations which are referred to as the reduced Maxwell-Bloch equations. An alternative simplification method results in a set of equations which are referred to as the self-induced transparency equations and it may be further shown that the reduced Maxwell-Bloch equations will simplify to give the self-induced transparency equations if certain limiting assumptions hold. These equations may also be simplified further. It is possible to show that the sine-Gordon equation can be obtained from these equations under further simplification for the resonant case.

As pointed out by Dodd, Eilbeck, Gibbon and Morris in their book [18] (page 548), the nonlinear Schrödinger equation is an unlikely equation to describe resonant propagation. For nonresonant propagation, however, one can derive the equation as a limiting case and this equation too has solitonic solutions and may be shown to be completely integrable.

The derivation of these equations is perhaps best presented in a pair of review papers by Bullough and some of his coworkers [21, 39]. The earlier paper is slightly more explicit but does contain incorrect suppositions which are corrected in the later

paper. The authors also choose to use c.g.s. units and for this reason we reproduce their version of the Maxwell-Bloch equations which we have just derived.

The Maxwell-Bloch equations are given as

$$\frac{\partial u}{\partial t} = -\omega_s v$$

$$\frac{\partial v}{\partial t} = \omega_s u + 2p\hbar^{-1}Ew$$

$$\frac{\partial w}{\partial t} = -2p\hbar^{-1}Ev$$

$$\nabla^2 E - \frac{\partial^2 E}{\partial t^2} = 4\pi nc^{-2}p\mathbf{u}\frac{\partial^2 u}{\partial t^2}$$

(2.63a,b,c,d)

where the factor of 4π comes from the use of c.g.s. units. It will be seen that this is identical to the version of the Maxwell-Bloch equations given in equations (2.47). The broadened form of the last equation can be written as

$$\nabla^2 E - \frac{\partial^2 E}{\partial t^2} = 4\pi nc^{-2}p\mathbf{u}\left\langle\frac{\partial^2 u}{\partial t^2}\right\rangle$$

(2.64)

where the angle brackets denote averaging with respect to the atomic population using the definition

$$\langle f\rangle = \int_0^\infty f(\omega_s')g(\omega_s')\mathrm{d}\omega_s'$$

(2.65)

in common with the previous discussion of homogeneous broadening. One then derives the reduced Maxwell-Bloch equations from (2.63) by using only the forward going characteristic of (2.63d) and using a scaling argument to eliminate terms valid for higher densities. The resulting equations

$$\frac{\partial u}{\partial t} = -\omega_s' v$$

$$\frac{\partial v}{\partial t} = \omega_s' u + 2p\hbar^{-1}Ew$$

$$\frac{\partial w}{\partial t} = -2p\hbar^{-1}Ev$$

$$\frac{\partial E}{\partial x} + c^{-1}\frac{\partial E}{\partial t} = -2\pi np\left\langle\frac{\partial u}{\partial t}\right\rangle = 2\pi np\langle\omega_s' v\rangle$$

(2.66a,b,c)

are known as the reduced Maxwell-Bloch equations. In the sharp line limit where there is only one resonant frequency ω_s these equations have been shown to form a completely integrable system and to have solitonic solutions [40]. In fact they can also be shown to have solitonic solutions when inhomogeneous broadening is retained [41]. They have also been shown to have multisoliton solutions by Gibbon and coworkers [42, 43].

On the other hand, the self-induced transparency equations may be derived from either the Maxwell-Bloch equations or the reduced Maxwell-Bloch equations by assuming that

$$E = \hbar p^{-1} \mathcal{E} \cos\Phi$$
$$\Phi = k_s x - \omega_s t + \phi, \ \omega_s = c k_s$$
$$u = Q\cos\Phi + P\sin\Phi$$

(2.67a,b,c)

where the functions $P,Q,u,\phi,\Phi,\mathcal{E}$ are all functions of x,t. We also use the slowly varying envelope and phase approximation. By this it is meant that it is assumed that the envelope and phase of the waveform vary on a slow time scale with respect to the carrier. Higher harmonic terms of the carrier frequency are also neglected. This approximation is called the rotating wave approximation. This gives rise to the self-induced transparency equations

$$\frac{\partial \mathcal{E}}{\partial x} + c^{-1}\frac{\partial \mathcal{E}}{\partial t} = \alpha\langle P\rangle$$
$$\mathcal{E}\left(\frac{\partial \phi}{\partial x} + c^{-1}\frac{\partial \phi}{\partial t}\right) = -\alpha\langle Q\rangle$$
$$\frac{\partial Q}{\partial t} = -\left(\Delta\omega' + \frac{\partial \phi}{\partial t}\right)P$$
$$\frac{\partial P}{\partial t} = \mathcal{E}\mathcal{N} + \left(\Delta\omega' + \frac{\partial \phi}{\partial t}\right)Q$$
$$\frac{\partial \mathcal{N}}{\partial t} = -\mathcal{E}P$$

(2.68a,b,c,d,e)

where $\Delta\omega'$ is given by $\Delta\omega' = \omega_s' - \omega_s$ and the angle brackets denote averaging over the atomic population as usual. The symbol $\mathcal{N}$ is used to represent the inversion previously given by w. The number α is given by $\alpha c = 2\pi n p^2 \omega_s \hbar^{-1}$. It may be shown that these equations also represent a completely integrable system with solitonic solutions. In fact,

even in the broadened case, it may be shown that the self-induced transparency equations and the reduced Maxwell-Bloch equations have multisoliton solutions [41].

If it is further assumed that $g(\Delta\omega')$ is even and Q odd and that ϕ is constant then the equations reduce to

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \chi}+c^{-1}\frac{\partial \mathcal{E}}{\partial t}&=\alpha\langle P\rangle\\ \frac{\partial P}{\partial t}&=\mathcal{E}\mathcal{N}+\Delta\omega' Q\\ \frac{\partial Q}{\partial t}&=-\Delta\omega' Q\\ \frac{\partial \mathcal{N}}{\partial t}&=-\mathcal{E}P\end{aligned} \tag{2.69}$$

Although these equations have exactly the same mathematical form as the reduced Maxwell-Bloch equations, here the term $\Delta\omega'$ replaces the term ω_i' in the reduced Maxwell-Bloch equations. The form of the solutions will obviously be the same for both systems. If it is further assumed that there is no broadening then the equations (2.69) reduce to

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \chi}+c^{-1}\frac{\partial \mathcal{E}}{\partial t}&=\alpha P\\ \frac{\partial P}{\partial t}&=\mathcal{E}\mathcal{N}\\ \frac{\partial \mathcal{N}}{\partial t}&=-\mathcal{E}P\\ Q&\equiv 0\end{aligned} \tag{2.70}$$

and by making the substitutions $P=-\sin\sigma$, $\mathcal{N}=-\cos\sigma$, $\mathcal{E}=\dfrac{\partial\sigma}{\partial t}$ one obtains (after rescaling) the equation

$$\frac{\partial^2\sigma}{\partial\xi^2}-\frac{\partial^2\sigma}{\partial\tau^2}=\sin\sigma \tag{2.71}$$

which may be recognised as the sine-Gordon equation in laboratory coordinates.

2.3.4.2. The Relationship Between The Equations.

We have seen how the Maxwell-Bloch equations can be modified to give the reduced Maxwell-Bloch equations and how these equations, or the original equations, can then be simplified to give the self-induced transparency equations. We have also seen how a limiting solution of the resonant case is the sine-Gordon equation. However, we have yet to explain why these equations are relevant to the work of the thesis and to clarify the relationship between the equations.

In a 1973 paper [44], Eilbeck, Gibbon, Caudrey and Bullough discuss many of the results they have obtained concerning the Maxwell-Bloch, reduced Maxwell-Bloch, self-induced transparency and sine-Gordon equations. They discuss not only the limitations of the approximations made in order to obtain the solutions already presented, but also the transformations necessary in order to compare the solutions of the equations for the purpose of investigating the physical significances of the solutions.

Firstly, the full Maxwell-Bloch equations provide the most accurate descriptions of the physical situation although it is possible to derive a more complicated set of equations which describe the same physical situation but with the additional inclusion of local field effects within the dielectric [21]. (The invalidity of the Maxwell-Bloch equations under certain experimental circumstances has already been pointed out in section 2.3.3..) The reduced Maxwell-Bloch equations provide the next most accurate description since the approximations made amount to a neglect of backscattering and an assumption that the density of active sites is low as we have previously explained. It is important to realise that these equations are both used to model the electric field together with the behaviour of the dielectric. The self-induced transparency equations model the envelope of the electric field.

Under a list of six assumptions given in the Eilbeck paper [44], one can then derive the self-induced transparency equations. The main approximations are the slowly varying envelope and phase approximation and the rotating wave approximation. We point out here that the slowly varying envelope and phase approximation assumes that the envelope and phase of the wave vary slowly with respect to the carrier and that this assumption is essentially made in the application of the averaged lagrangian technique which also assumes a separation of scales between the envelope and carrier. This will be discussed more fully later.

We stated earlier that the known pulse solution of the Maxwell-Bloch equation was not physically interesting. In fact, the single soliton solution of the reduced

Maxwell-Bloch equations corresponds to this solution of the Maxwell-Bloch equations [39]. The two soliton solution of the reduced Maxwell-Bloch equations, the so-called breather solution, can be shown to be equivalent to the solitonic solution of the self-induced transparency equations [39]. It will be remembered that the self-induced transparency equations have a structure which is mathematically similar to the reduced Maxwell-Bloch equations so that this correspondence is not overly surprising. In addition, the fact that the sine-Gordon equation can be derived as a limiting case for the resonant situation has already been mentioned. The paper by Bullough [39] also discusses the derivation of the nonlinear Schrödinger equation in connection with self-focussing. The nonlinear Schrödinger equation is often found to be applicable in describing nonresonant nonlinear phenomena.

All of these equations are nonlinear and all describe the same type of physical situation. Since this work was concurrent with Whitham's work on the averaged lagrangian method, it would be surprising if the idea of using the averaged lagrangian techniques to investigate these equations had not arisen within Bullough's group. This was indeed the case and, in view of the fact that we shall subsequently use Whitham's method to examine similar equations, this is a further reason why a consideration of these equations was necessary. Not only do these equations form a considerable part of the work describing nonlinear pulse propagation in nonlinear media, they are also, in principle, amenable to study using Whitham's averaged lagrangian techniques. In section 3.3.1.4 , we discuss the thesis by Jack [45] in which the use of the averaged lagrangian technique is contrasted with the use of the slowly varying envelope and phase approximation.

However, our main purpose in considering this body of work has been to derive the equations which describe electromagnetic wave propagation in two-level atomic media. In subsequent sections, we will present a similar model within the context of a lagrangian description of essentially the same physical situation: having examined the derivation necessary to derive the equations for a two-level system, and furthermore looked briefly at the similarities between the equations so derived, we will be better informed to make comparisons between the results obtained by each method.

2.4. Modelling Quantum Systems.

We have seen how there has been a considerable quantity of work which studies the behaviour of two-level atomic systems interacting with incident electromagnetic radiation. In our derivation of the Maxwell-Bloch equations, we saw how the use of the Pauli spin matrices as a basis set for the representation of the motion considerably simplified the analysis. In fact, this derivation could be considered a simple prototype calculation for systems with more quantum levels and the use of Pauli spin matrices could be considered the simplest form of a technique of calculation involving matrix representations of Lie-groups. For systems with more than two quantum levels, it proves possible to make some progress by extending both techniques as we have indicated. It is the purpose of this section to explore how such calculations have been used to simplify calculations involving several quantum levels - especially by Eberly and by Hioe - and to suggest how such techniques might be subsequently incorporated into a lagrangian model of electromagnetic wave propagation in nonlinear quantized media.

2.4.1. Feynman's Paper and Two-level Systems.

We have already mentioned the paper of Feynman, Vernon and Hellwarth [20] during the derivation of the Maxwell-Bloch equations. As they point out in their introduction, their paper "does not obtain results which are inaccessible to straightforward calculation, [although] the simplicity of the pictorial representation enables one to gain physical insight and obtain results quickly which display the main features of interest." Nevertheless, their paper did have a substantial influence on the development of work in this field. It was the "pictorial", or, rather, the geometric approach, which lead to important simplifications in the investigations of problems involving two-level atoms.

Although the original paper by Feynman uses the Schrödinger picture in describing the wave-functions, and although it does not use Pauli matrices as a basis set for the representation of the functions describing the behaviour of the vector components, the paper contains all of the information necessary to carry out the derivation given previously as far as equation (2.29) $\partial\rho/\partial t = \omega \times \rho$. To make full use of the method, however, it is easier to use the Liouville equation, density matrices and the

Heisenberg picture as we have done in the derivation of the Maxwell-Bloch equations. The reason we have followed the latter derivation is that the same approach can be modified to consider three or multi-level systems.

Before going on to consider three level systems, it is interesting to compare the simplicity of the derivation of the Maxwell-Bloch equations given earlier with a fully quantized approach to the same problem given by Tavis and Cummings [46, 47]. In the first of these papers [46], the authors derive equations which describe the interaction of a field of two level atoms with a single mode quantized radiation field. Their exact solution is complicated and they use a computer in order to graph their results. In the second paper [47], they discuss various approximations to their results in order to judge the validity of the approximation schemes used by other authors.

2.4.2. Three Level Systems I. Alternatives.

There are many models and formalisms which have been developed to describe three-level systems. Although our interest lies in one particular modelling scheme to be presented in the next section, it is useful to briefly look at some of the other schemes available. We shall not examine these schemes in detail but it is important to bear in mind that there are alternatives to the modelling techniques to be presented which give valid results although they do so in a manner which is possibly not as æsthetically appealing.

One of the first models of three-level media was that of Tan-no, Yokoto and Inaba [48, 49]. The first of these two papers derives Bloch equations for a three-level system and couples them with Maxwell's equations, in the form previously given for the two level system, to give a set of equations analogous to the Maxwell-Bloch equations for two levels. They give these equations as

$$\frac{\partial \rho_\mu}{\partial t} = \kappa \varepsilon_\lambda \varepsilon_\nu v_\mu - (\rho_\mu - \rho_\mu^0)/T_1$$

$$\frac{\partial u_\mu}{\partial t} = -\left[\Delta\omega - \frac{\partial \phi_\mu}{\partial t} + \left(\mu_\lambda^2 \varepsilon_\lambda^2 + \mu_\nu^2 \varepsilon_\nu^2\right)/4\hbar\Delta\omega_0\right] v_\mu - u_\mu/T_2$$

$$\frac{\partial v_\mu}{\partial t} = \left[\Delta\omega - \frac{\partial \phi_\mu}{\partial t} + \left(\mu_\lambda^2 \varepsilon_\lambda^2 + \mu_\nu^2 \varepsilon_\nu^2\right)/4\hbar\Delta\omega_0\right] u_\mu - \kappa \varepsilon_\lambda \varepsilon_\nu \rho_\mu - v_\mu/T_2$$

$$\frac{\partial \varepsilon_\lambda}{\partial z} + \frac{n_\lambda}{c}\frac{\partial \varepsilon_\lambda}{\partial t} = -\tfrac{1}{2}\beta_\lambda \langle \varepsilon_\nu v_\mu \rangle$$

$$\frac{\partial \varepsilon_\nu}{\partial z} + \frac{n_\nu}{c}\frac{\partial \varepsilon_\nu}{\partial t} = -\tfrac{1}{2}\beta_\nu \langle \varepsilon_\lambda v_\mu \rangle$$

$$\left(\frac{\partial \phi_\lambda}{\partial z} + \frac{n_\lambda}{c}\frac{\partial \phi_\lambda}{\partial t}\right)\mu_\nu \varepsilon_\lambda = -\tfrac{1}{2}\beta_\lambda \langle 2\mu_\lambda \varepsilon_\lambda \rho_\lambda + \mu_\nu \varepsilon_\nu u_\mu \rangle$$

$$\left(\frac{\partial \phi_\nu}{\partial z} + \frac{n_\nu}{c}\frac{\partial \phi_\nu}{\partial t}\right)\mu_\lambda \varepsilon_\nu = -\tfrac{1}{2}\beta_\lambda \langle 2\mu_\nu \varepsilon_\nu \rho_\nu + \mu_\lambda \varepsilon_\lambda u_\mu \rangle$$

(2.72a-g)

where

$$\kappa = \mu_\lambda \mu_\nu / 2\hbar^2 \Delta\omega_0, \ \beta_i = 2\pi N_0 \Omega_i \mu_\lambda \mu_\nu / c\hbar n_i \Delta\omega_0, \ \phi_\mu = \phi_\lambda + \phi_\nu$$

(2.73)

and where damping effects have been introduced phenomenologically as before. The meaning of the symbols can be obtained from the reference itself. The main points are, firstly, the similarity of the form of these equations to the two-level Maxwell-Bloch equations when the equation describing the electric field is written separated into its two characteristics and, secondly, that there are two incident light waves whose sum frequency is equal to the transition frequency between the first and final states. The second paper [49] contains numerical calculations based on the formulae calculated in the first paper. In view of the complicated nature of the equations for these systems, we will refrain from reproducing further examples until the next section.

The paper by Brewer and Hahn [50] presents results in terms of a density matrix formalism. They consider a molecular three-level system in which the lower two levels are degenerate and where the Stark effect can be used to switch the transitions in and out of resonance with the applied laser field. They derive Bloch like equations which describe the behaviour of the system as it is brought in and out of resonance as well as presenting a steady state calculation. They also consider the case in which a three level system in a ladder configuration has the upper and lower levels tuned to resonance with the sum frequency of two oppositely directed laser beams, again by using the Stark

effect. The equations they present are all given in terms of differential equations for the components of the density matrix which has nine components for the three-level case. Since we will be considering similar equations in the next section, we do not give the equations here.

A further paper of interest is that by Grischkowsky, Loy and Liao [51]. This paper also serves to put other theories, including that of Brewer and Hahn, in context. The density matrix formalism of Brewer and Hahn necessarily involves a vector description of the evolution. What was not immediately apparent from their paper was that this vector description is related to that of Feynman, Vernon and Hellwarth [20] for two-level systems.

In their paper they derive the density matrix equations which describe the behaviour of a three level system. By using a transformation in which the equations are considered in a doubly rotating frame of reference, they are able to write the hamiltonian matrix in a form which can, after approximation, be diagonalized thus solving for the evolution. One can then approximate the three-level system by a two-level one since the behaviour of the intermediate state can, again by approximation, be decoupled from the behaviour of the initial and final states. Alternatively, it is possible to start from an approximately equivalent two-level system by using a transformation that uses the intermediate state(s) to modify the initial and final states in such a way that they couple correctly to the applied field. This means that the model can then proceed essentially as for a two-level system.

Having derived the necessary equations, the authors then show how the approximate equations they have obtained can be shown to correspond to the vector model of Feynman, Vernon and Hellwarth as modified for two-photon systems. By making a further approximation, they then go on to use the equations they have obtained to calculate some physically relevant results such as expressions for the polarization. Thus although this paper represents an approximation to three-level systems, it can be thought of as an intermediate stage between two-level vector models,especially that given by Feynman and coworkers and between true three-level models.

A model by Makhviladze, Sinitsyn, and Shelepin [52] is used to calculate the changes in profile as a pulse propagates through a three-level system. They use numerical techniques to solve a complicated set of ten differential equations describing the evolution.

Finally, before we go on to discuss the geometrical model for three-level systems, it is useful to have a brief look at fully quantized models. The first of these is

given in a paper by Radmore and Knight [53]. The authors use fields quantized in number states and obtain a hamiltonian which, they report, could have been obtained by using semi-classical theory and unitary transformations. They then go on to examine and graph the populations of the levels under different conditions. Finally, they include an appendix in which they justify the assumptions they have made in order to include phenomenological damping.

The second model is given in a paper by Obada and Abdel-Hafez [54]. The model is fully quantized. It considers a three-level system coupled to a radiation field in which annihilation and creation operators are used to describe the behaviour of photons in each of the two modes. They are also able to obtain statistical measures of the behaviour of the system and can make predictions as to the evolution of the system under different initial conditions. It is possibly not surprising that the model is written in terms of spin-1 operators and this correspondence between number of levels and spin systems is what makes possible the analysis to be presented in the next section.

2.4.3. Three Level Systems II. Geometrical Approach.

We have hinted that the description of three-level systems using a geometrical approach is a useful one. We will now go on to see why this is so. We will find that by using a vector description based on a density matrix formalism one can recast the equations in a particularly elegant form by using higher dimensional analogues of the Pauli spin matrices. These matrices in fact form matrix representations of Lie algebras and it is their commutation relations which can be used to simplify the equations of motion for the three-level case. We will find that the application of these techniques to higher level situations is also possible. The techniques make use of the three-level Maxwell-Bloch analogues derived in the previous section and can be generalized to include more levels.

The first paper which is directly connected with the geometrical formalism under consideration is given by J. N. Elgin [55]. Again, the starting point for the analysis is the paper by Feynman, Vernon and Hellwarth [20] where the two-level rotation vector was first introduced in the field of optics. In the paper Elgin demonstrates that, for a three-level system, the three-dimensional analogue of the Feynman vector can be shown to obey a torque equation identical in form with that for the two level system. He further shows how the rotations of the state vector can now be written in terms of generators of the group SU(3) rather than SU(2) for the two-level case. To use Elgin's mathematical

terminology: "the state vector now moves in a three-dimensional Hilbert space, and the rotations belong to the group SU(3)." Since the rotations are now being modelled in terms of the matrix representations of the group SU(3), it can be shown that there must be two operators, the Casimir operators, which will commute with all the generators (matrices) of the group and therefore there must be two conserved quantities which correspond to these operators. It should be noted that, in the three-level case, there are two light fields, each of which is assumed to be slightly detuned from resonance and that furthermore the transitions are also assumed to be nondegenerate.

In the second paper in this series [56], Hioe and Eberly generalize their formalism to model N-level systems, the rotations now being given in terms of the generators of the group SU(N) for the N-dimensional Hilbert space. The level spacings between the levels are arbitrary and the external electric field can have "arbitrary strength time dependence and resonance character." We will now show how the N-level equations may be written in compact form using the formalism as presented by Hioe and Eberly in this paper. We have made no changes to the notation and our purpose in reproducing the formalism here is to show how the much more complicated general case for an N-level system can be simply and compactly written using this formalism.

Hioe and Elgin begin by assuming that they are dealing with an N-level system which has unperturbed energies E_m corresponding to eigenstates $|m\rangle$. The system is perturbed by a possibly time dependent term $\hat{V}(t)$ which may be of arbitrary magnitude. This means that the hamiltonian for the system can be written as

$$\hat{H} = \sum_m E_m \hat{P}_{mm} + \sum_{m,n} V_{mn}(t)\hat{P}_{mn} \tag{2.74}$$

They use carets for operators and define the $\hat{P}_{mn}$ by

$$\hat{P}_{mn} = |m\rangle\langle n| \tag{2.75}$$

where, by convention,

$$V_{mn}(t) = \langle m|\hat{V}(t)|n\rangle \tag{2.76}$$

(2.75) can be used to prove that

$$\left[\hat{P}_{kl}, \hat{P}_{mn}\right] = \hat{P}_{kn}\delta_{lm} - \hat{P}_{ml}\delta_{nk} \tag{2.77}$$

The Liouville equation can be written as

$$i\hbar\,\partial\hat{\rho}/\partial t = \left[\hat{H},\hat{\rho}\right] \tag{2.78}$$

and this can be recast in the Heisenberg picture by use of the relation

$$\left\langle \hat{P}_{mn}(t)\right\rangle = \mathrm{Tr}\left\{\hat{P}_{mn}\hat{\rho}(t)\right\} = \rho_{nm}(t) \tag{2.79}$$

Using the definitions

$$\hat{u}_{jk} \equiv \hat{P}_{jk} + \hat{P}_{kj}, \hat{v}_{jk} \equiv -i\left(\hat{P}_{jk} + \hat{P}_{kj}\right), \hat{w}_l \equiv -[2/l(l+1)]^{1/2}\left(\hat{P}_{11} + \cdots + \hat{P}_{ll} - l\hat{P}_{l+1,l+1}\right) \tag{2.80}$$

where $1 \le j < k \le N$, $1 \le l \le N-1$ one can write down a vector $\vec{\hat{s}}$ which is given by the ordered array

$$\vec{\hat{s}} = \left(\hat{u}_{12},\ldots,\hat{v}_{12},\ldots,\hat{w}_1,\ldots,\hat{w}_{N-1}\right) \tag{2.81}$$

The vector $\vec{\hat{s}}$ has components $\hat{s}_i$ which satisfy the relation

$$\left[\hat{s}_j,\hat{s}_k\right] = 2if_{jkl}\hat{s}_l = 2i\sum_{1}^{N^2-1} f_{jkl}\hat{s}_l \tag{2.82}$$

where the f_{jkl} are the completely antisymmetric structure constants of the SU(N) group. Using the above definitions, one can now write the density matrix and the hamiltonian operators as

$$\hat{\rho}(t) = N^{-1}\hat{I} + \frac{1}{2}\sum_{j=1}^{N^2-1} S_j(t)\hat{s}_j \tag{2.83}$$

and

$$\hat{H}(t) = \frac{\hbar}{2}\sum_{j=1}^{N^2-1}\Gamma_j(t)\hat{s}_j \tag{2.84}$$

where

$$S_j(t) = \mathrm{Tr}\left\{\hat{\rho}(t)\hat{s}_j\right\} \tag{2.85}$$

and

$$\hbar\Gamma_j(t) = \mathrm{Tr}\left\{\hat{H}(t)\hat{s}_j\right\} \tag{2.86}$$

where we have used the relation

$$\mathrm{Tr}\left\{\hat{s}_j,\hat{s}_k\right\} = 2\delta_{jk} \tag{2.87}$$

Using this relation to obtain the components of the Liouville equation (2.78), one obtains the result sought which is valid for any N

$$\frac{\partial S_i}{\partial t} = f_{ijk}\Gamma_j S_k \tag{2.88}$$

By working through this derivation for N=2, one can verify that the above equation reduces to the vector form of the Bloch equations given previously with $f_{ijk} \rightarrow \varepsilon_{ijk}$.where ε_{ijk} is the Levi-Civita permutation symbol. The authors then prove the existence of several constants of the motion which serve to constrain the trajectory of the state vector in the Hilbert space. Finally, they apply their formalism to the three-level case and show how the constraints inherent within the formalism can serve to rule out certain population distributions between the levels.

That the formalism is compact is beyond question: it is also utilitarian. In a subsequent paper by Hioe and Eberly [57], they use the formalism just developed for the case SU(3), that is for a three-level system, and derive the following set of equations for two photon resonance. That is, the condition $\Delta_{12} = -\Delta_{23} \equiv \Delta$ relating to the detunings pertaining to each level must hold. These are the three-level analogues of the Bloch equations.

$$\frac{\partial u_{12}}{\partial t} = \Delta v_{12} + \beta v_{13}$$
$$\frac{\partial u_{23}}{\partial t} = \Delta v_{23} - \alpha v_{13}$$
$$\frac{\partial u_{13}}{\partial t} = \beta v_{12} - \alpha v_{23}$$
$$\frac{\partial v_{12}}{\partial t} = -\Delta u_{12} + \beta u_{13} + 2\alpha w_1$$
$$\frac{\partial v_{23}}{\partial t} = \Delta u_{23} + \alpha u_{13} - \beta w_1 + \sqrt{3}\beta w_2$$
$$\frac{\partial v_{13}}{\partial t} = -\beta u_{12} + \alpha u_{23}$$
$$\frac{\partial w_1}{\partial t} = -2\alpha v_{12} + \beta v_{23}$$
$$\frac{\partial w_2}{\partial t} = -\sqrt{3}\beta v_{23}$$

(2.89)

As Hioe and Eberly point out, the two level equations can be recovered by removing any symbol containing the index 1 or 3. In these equations $\alpha(t)$ and $\beta(t)$ are given by the equations

$$\alpha(t) = \frac{\vec{d}_{12} \cdot \vec{E}_{12}(t)}{\hbar} = \tfrac{1}{2}\Omega_{12}(t)$$
$$\beta(t) = \frac{\vec{d}_{23} \cdot \vec{E}_{23}(t)}{\hbar} = \tfrac{1}{2}\Omega_{23}(t)$$

(2.90a,b)

and the detunings Δ_{jk} are given by $\Delta_{jk} = \nu_{jk} - \omega_{jk}$. If the further assumption is made that the two functions $\alpha(t)$ and $\beta(t)$ have the same time dependence, the authors then show how the use of the matrix formalism can facilitate the simplification of the eight dimensional vector equation for the state vector. This is achieved by block diagonalizing the vector equation so that the vector can then be analysed in terms of three separate subspaces of dimensions 1,3 and 4. This is equivalent to saying that, under the conditions given, there are three exact conserved nonlinear quantities. If the two frequencies are tuned to the two resonances then the four dimensional space splits again into two two-dimensional spaces. We do not give the full workings of this reduction since we will present a similar one shortly.

The determination of constants of the motion for three-level systems is the topic of several papers. We have already seen how the block diagonalization of the matrix

equations lead to the introduction of conserved quantities and we noted that the block diagonalization was dependent on the assumption that the two functions $\alpha(t)$ and $\beta(t)$ had the same time dependence. A paper by Gottlieb [58] describes how the correct choice of phase difference between each of these two equations leads to a matrix equations which is no longer block diagonal but is soluble for the equations of motion. In addition, he derives two linear constants of the motion for this special case.

This work is subsequently discussed by Hioe [59] in a paper in which he rederives Gottlieb's results using a method which he claims is simpler. Gottlieb refutes this assertion somewhat in a subsequent paper [60]. However, in addition to carrying out several matrix calculations pertaining to Gottlieb's examples, Hioe also mentions the work contained in an earlier paper by Hioe [61] which discusses the use of different subgroupings within the general SU(3) framework. ($\mathbf{SU(3) \supset SU(2) \times U(1)}$ or $\mathbf{SU(3) \supset O(3)}$.)

The main point of the paper by Hioe is that different representations of the same group can be employed in models of the same systems under different experimental conditions and that the choice of the correct representation can serve to simplify the analysis. The analysis employed is essentially that presented in the earlier paper [56]. The analyses differ in the choice of the matrix representations and in these representations commutation relations. Hioe also mentions that the block diagonalized form of the evolution matrices can be related to simulton propagation. We refer briefly to simultons in the next section.

The idea that the description of the evolution of the state vector can be simplified by the block diagonalization of the matrix is used in a paper by Oreg, Hioe and Eberly [62]. Their paper considers a system having $N^2 - 1$ levels and shows that the describing space has an (N-1) dimensional subspace whose basis vectors are given explicitly in terms of the hamiltonian matrix elements.

Before moving on to consider an interesting paper by Aravind [63] we briefly mention a rather complicated paper by Hioe [64] which considers the lossless propagation of optical pulses in N-level systems. It uses the irreducible tensorial sets of Racah in conjunction with Wigner 6-*j* symbols to simplify the calculations: we have already observed the correspondence between spin systems and multiple level atoms. Their calculations represent a general solution to the situation in which a pulse traverses a medium having the necessary symmetry. Their main conclusion is that for an N-level system, there are N-1 sets of conditions, each of which would permit the propagation of

simultons, multiple wavelength solitons. It is possible that such formalism could also be incorporated into a lagrangian model were it to be deemed necessary.

Finally, we return to the paper of Aravind [63]. This paper uses the pseudospin formalism presented by Hioe and Eberly [56, 57] in the rotating wave approximation. We have already seen how the eight dimensional matrix equations which describe the motion can be block diagonalised so that the motion can be described in terms of three separate subspaces of dimensions 4,3 and 1 [57]. Aravind gives a solution which is valid for arbitrary initial conditions within the restriction of exact two-photon resonance and the rotating wave approximation. Aravind also derives results which include relaxational effects. We will not examine the results which concern these relaxational effects in detail. His derivation of the block diagonal form is worth examining since it improves somewhat on that presented in the paper by Hioe [57].

Using the projection formula given as (2.85), one can write the components of the pseudospin vector as

$$\begin{aligned} S_1 &\equiv u_{12} = \rho_{12} + \rho_{21} \\ S_2 &\equiv u_{23} = \rho_{23} + \rho_{32} \\ S_3 &\equiv u_{13} = \rho_{13} + \rho_{31} \\ S_4 &\equiv v_{12} = -i(\rho_{12} - \rho_{21}) \\ S_5 &\equiv v_{23} = -i(\rho_{23} - \rho_{32}) \\ S_6 &\equiv v_{13} = -i(\rho_{13} - \rho_{31}) \\ S_7 &\equiv w_1 = -(\rho_{11} - \rho_{22}) \\ S_8 &\equiv w_2 = -\frac{1}{\sqrt{3}}(\rho_{11} + \rho_{22} - 2\rho_{33}) \end{aligned} \tag{2.91}$$

and under the assumptions given, this means that the equation of motion for S can be written as

$$\frac{dS_i}{dt} = \sum_{j,k} f_{ijk}\Gamma_j S_k, \; i,j,k = 1,\ldots,8 \tag{2.92}$$

where the vector Γ is given by

$$\Gamma = \left(-2\Omega_1, -2\Omega_2, 0, 0, 0, 0, -\Delta, \Delta/\sqrt{3}\right) \tag{2.93}$$

and f_{ijk} are the structure constants.The constants Ω_1, Ω_2 and Δ are given by

$$\hbar\Omega_1 = E_{12}\hat{\varepsilon}_{12}\cdot\mathbf{d}_{12},\ \hbar\Omega_2 = E_{23}\hat{\varepsilon}_{23}\cdot\mathbf{d}_{23},\ \Delta = \nu_{12} - \omega_{12} \tag{2.94}$$

where the $\mathbf{d}_{ij}$ are dipole matrix elements. The equation of motion becomes

$$\frac{dS}{dt} = \mathfrak{M}S \tag{2.95}$$

where $\mathfrak{M}$ has its nonzero components given by

$$\mathfrak{M}_{14} = \Delta, \mathfrak{M}_{16} = \Omega_2, \mathfrak{M}_{25} = -\Delta, \mathfrak{M}_{26} = -\Omega_1, \mathfrak{M}_{34} = \Omega_2,$$
$$\mathfrak{M}_{35} = -\Omega_1, \mathfrak{M}_{47} = 2\Omega_1, \mathfrak{M}_{57} = -\Omega_2, \mathfrak{M}_{58} = \sqrt{3}\Omega_2 \tag{2.96}$$

They now transform the original pseudospin vector S into a vector $T = \mathfrak{R}S$. The matrix $\mathfrak{R}$ is given by

$$\mathfrak{R} = \frac{1}{\varepsilon}\begin{pmatrix}
\Omega_1 & \Omega_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -\Omega_1 & \Omega_2 & 0 & 0 & 0 \\
0 & 0 & \varepsilon^{-1}\Omega_1\Omega_2 & 0 & 0 & 0 & -\tfrac{1}{2}\varepsilon^{-1}\left(2\Omega_1^2+\Omega_2^2\right) & \varepsilon^{-1}\frac{\sqrt{3}}{2}\Omega_2^2 \\
0 & 0 & 0 & 0 & 0 & \varepsilon & 0 & 0 \\
\Omega_2 & -\Omega_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \Omega_2 & \Omega_1 & 0 & 0 & 0 \\
0 & 0 & \varepsilon^{-1}\left(\Omega_2^2-\Omega_1^2\right) & 0 & 0 & 0 & -\varepsilon^{-1}\Omega_1\Omega_2 & -\varepsilon^{-1}\sqrt{3}\Omega_1\Omega_2 \\
0 & 0 & -\sqrt{3}\varepsilon^{-1}\Omega_1\Omega_2 & 0 & 0 & 0 & -\frac{\sqrt{3}}{2}\varepsilon^{-1}\Omega_2^2 & \tfrac{1}{2}\varepsilon^{-1}\left(2\Omega_1^2-\Omega_2^2\right)
\end{pmatrix} \tag{2.97}$$

where $\varepsilon = \left(\Omega_1^2+\Omega_2^2\right)^{1/2}$. The equation of motion now becomes

$$\frac{dT}{dt} = \mathfrak{S}T,\ \mathfrak{S} = \mathfrak{R}\mathfrak{M}\mathfrak{R}^{-1} \tag{2.98}$$

One finds that this transformation has put the matrix $\mathfrak{S}$ in block-diagonal form such that $\mathfrak{S} = \mathfrak{S}_3 \oplus \mathfrak{S}_4 \oplus \mathfrak{S}_1$. The blocks are given by

$$\mathfrak{S}_3 = \begin{pmatrix} 0 & -\Delta & 0 \\ \Delta & 0 & 2\varepsilon \\ 0 & -2\varepsilon & 0 \end{pmatrix}$$

$$\mathfrak{S}_4 = \begin{pmatrix} 0 & -\varepsilon & 0 & 0 \\ \varepsilon & 0 & \Delta & 0 \\ 0 & -\Delta & 0 & -\varepsilon \\ 0 & 0 & \varepsilon & 0 \end{pmatrix}, \quad \mathfrak{S}_1 = (0) \tag{2.99}$$

and since $\mathfrak{S}$ is, for our purposes constant, (2.98) can be integrated as

$$T(t) = \exp(\mathfrak{S}t)T(0) \tag{2.100}$$

or

$$S(t) = \mathfrak{R}^T \exp(\mathfrak{S}t)\mathfrak{R}\mathfrak{S}(0) \tag{2.101}$$

We can therefore write

$$\begin{aligned} \exp(\mathfrak{S}t) &= \exp(\mathfrak{S}_3 t) \oplus \exp(\mathfrak{S}_4 t) \oplus \exp(\mathfrak{S}_1 t) \\ &= \mathfrak{A} \oplus \mathfrak{B} \oplus \mathfrak{C} \end{aligned} \tag{2.102}$$

and use the Cayley-Hamilton theorem to solve the matrix equations to give explicit expressions for all of the elements of the matrices $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$.

This analysis has perhaps demonstrated that the use of the pseudospin formalism can be a powerful tool in the description of multilevel systems. Our purpose in introducing it has been to suggest that its use in a subsequent lagrangian model could prove to afford valuable simplification in the analysis.

2.4.4. Many Level Systems and Simultons.

The use of the matrix techniques to simplify equations for multi-level atoms is also possible. However, there is a quantity of literature, which predates the formalism, which nevertheless models multi-level systems. In this section, we will look briefly at such models as well as those involving vector models adapted for multiple levels. We will also look briefly at the concept of the simulton. Although the lagrangian model to be presented is applied to the two-level case alone, the extension to the multi-level case is obvious and might be profitably couched in the formalism used in this and previous sections.

Some of the earliest papers to cover the multiple level case in some generality are to be found in a series of papers published by Eberly, Shore and coworkers [65-67]. The first of these papers [65] relates to a series of numerical experiments involving an imagined multi-level atom. In their introduction, the authors give some reasons why the customary two-level models are inadequate for modelling some experimental situations and discuss the other possibility, that of modelling the levels in pairs using a rate-equation approach. Their model assumes that the N-level system has lasers tuned to the transition frequencies between the levels which are assumed not to change in frequency due to the irradiation. The model also fails to take into account relaxation or dissipative processes. They then go on to present graphs of the results of their numerical calculations in which they show population changes in 2,3,4,5, and 10 level systems.

In the second paper in this series [66], a mathematical model is presented in which it is shown that, under sufficiently strict conditions, it is possible to model an N-level system pumped as described in terms of Jacobi matrices and that the solutions for these systems can be written in terms of special functions. The assumptions are rather restrictive, however, calling for a complete neglect of losses and nonresonant interactions as well as invoking a global rotating wave approximation.

The third paper [67] considers an analytically soluble case which does include detuning and loss. Their model relies on the analogy between an N-level system and a spin-J system where N=2J+1. By using this analogy, all of the techniques for modelling spin systems become available and the authors are able to find periodic analytic solutions which describe the population dynamics.

We have already mentioned the concept of the simulton in connection with the paper by Hioe [64]. This paper contains results concerning simulton propagation couched in the language of the group theory formalism of the previous section. There were, however, predictions of simulton propagation prior to the application of the group theory formalism in this area. One of the first of these was by Konopnicki, Drummond and Eberly [68]. In this paper, they use Maxwell-Bloch type equations for an imagined N-level system and show that one of the solutions involves the simultaneous propagation of a group of different wavelength optical solitons which they name a simulton. They also calculate the conditions which must hold for simulton propagation to take place. The conditions that are required to be satisfied are that the N-pulses (in an N+1 level system) which comprise the simulton must be nearly coincident; that the absorbing medium must, in general, be excited out of its ground state; and that the pulse amplitudes must satisfy certain relations. They also use numerical models to investigate

the effect of loss mechanisms on the stability of the pulses together with their sensitivity to initial conditions.

In a second paper by Konopnicki and Eberly [69] the authors present the results for the three level case in some detail. They present a derivation of the three-level Maxwell-Bloch equations in an appendix and then use these equations to calculate analytic solutions for simulton propagation in the three-level medium. They then go on to use numerical models to indicate the effects of the variation of various system parameters on the evolution as well as to investigate the effects of loss mechanisms as before. They present a conservation law which is obeyed by the vector components and is reminiscent of the conservation of probability conservation law for the two level system.

We have now discussed the Maxwell-Bloch equations for two level systems; the derivation and applicability of related equations such as the self-induced transparency equations; the limitations of the Maxwell-Bloch equations; the extension of the Maxwell-Bloch equations to consider three- and multi-level systems; the use of Pauli spin matrices in the two-level case and the extension of this technique to the Lie group techniques for equations possessing SU(N) symmetry. We have seen how there is a rich variety of phenomena which can be investigated in two- or multi-level systems. We must now go on to examine a further technique used in the investigation of pulses described by nonlinear partial differential equations - Whitham's averaged lagrangian technique and its variants.

References.

1. Hasegawa, A. and F. Tappert, *Transmission of Stationary Nonlinear Optical Pulses in Dispersive Dielectric Fibers. II. Normal Dispersion.* Appl. Phys. Lett., 1973. **34**(4): p. 171-172.

2. Hasegawa, A. and F. Tappert, *Transmission of Stationary Nonlinear Optical Pulses in Dispersive Dielectric Fibers. I. Anomalous Dispersion.* Appl. Phys. Lett., 1973. **23**(3): p. 142-144.

3. Hasegawa, A. and Y. Kodama, *Signal Transmission by Optical Solitons in Monomode Fiber.* Proc. IEEE, 1981. **69**(9): p. 1145-1150.

4. Hasegawa, A. and Y. Kodama, *Amplification and Reshaping of Optical Solitons in a Glass Fibre - I.* Opt. Lett., 1982. 7(6): p. 285-287.

5. Kodama, Y. and A. Hasegawa, *Amplification and Reshaping of Optical Solitons in Glass Fibre II.* Opt. Lett., 1982. 7(7): p. 339-341.

6. Kodama, Y. and A. Hasegawa, *Amplification and Reshaping of Optical Solitons in Glass Fibre III. Amplifiers with Random Gain.* Opt. Lett., 1983. 8(6): p. 342-344.

7. Hasegawa, A., *Amplification and Reshaping of Optical Solitons in a Glass Fibre IV: Use of the Stimulated Raman Process.* Opt. Lett., 1983. 8(12): p. 650.

8. Hasegawa, A., *Numerical Study of Optical Soliton Transmission Amplified Periodically by the Stimulated Raman Process.* Appl. Opt., 1984. 23(19): p. 3302-3309.

9. Hasegawa, A., *Generation of a Train of Soliton Pulses by Induced Modulational Instability in Optical Fibers.* Opt. Lett., 1984. 9(7): p. 288-290.

10. Kodama, Y., *Optical Solitons in a Monomode Fiber.* J. Stat. Phys., 1985. 39(5/6): p. 597-624.

11. Jeffrey, A. and T. Kawahara, *Asymptotic Methods in Nonlinear Wave Theory.* 1 ed. Applicable Mathematics Series, Pitman Advanced Publishing Program, 1982, Boston, London, Melbourne: Pitman Books Limited. 256 pp.

12. Kodama, Y. and A. Hasegawa, *Nonlinear Pulse Propagating in a Monomode Dielectric Guide.* I.E.E.E. J. Quant. Electron., 1987. QE-23(5): p. 510-524.

13. Potasek, M.J., *Modulation Instability in an Extended Nonlinear Schrodinger Equation.* Opt. Lett., 1987. 12(11): p. 921-923.

14. Kodama, Y. and K. Nozaki, *Soliton Interaction in Optical Fibers.* Opt. Lett., 1987. 12(12): p. 1038-1040.

15. Tai, K., A. Hasegawa, and N. Bekki, *Fission of Optical Solitons Induced by Stimulated Raman Effect.* Opt. Lett., 1988. **1988**(13): p. 5.

16. Mathews, P.M. and K. Venkatesan, *A Textbook of Quantum Mechanics.* 1 ed. 1976, New Delhi: Tata McGraw-Hill. 387 pp.

17. Fano, U., *Description of States in Quantum Mechanics by Density Matrix and Operator Techniques.* Rev. Mod. Phys., 1957. **29**(1): p. 74-93.

18. Dodd, R.K., *et al., Solitons and Nonlinear Wave Equations.* 1st (with corrections) ed. 1984, Academic Press. 630 pp.

19. Allen, L. and J.H. Eberly, *Optical Resonance and Two-level Atoms.* Dover ed. 1987, Dover Publications, Inc. 233.

20. Feynman, R.P., F.L. Vernon, and R.W. Hellwarth, *Geometrical Representation of the Schrödinger Equation for Solving Maser Problems.* J. Appl. Phys., 1957. **28**(1): p. 49-52.

21. Bullough, R.K., *et al., A General Theory of Self-induced Transparency.* Opto-electronics, 1974. **6**: p. 121-140.

22. Pantell, R.H. and H.E. Puthoff, *Fundamentals of Quantum Electronics.* 1 ed. 1969, New York, London, Sidney, Toronto: John Wiley and Sons Inc. 361 pp.

23. Torrey, H.C., Phys. Rev., 1949. **76**: p. 1059.

24. Rabi, I.I., Phys. Rev., 1937. **51**: p. 652.

25. Smith, R.A., *Excitations between Atomic or Molecular Energy Levels by Monochromatic Laser Radiation. I. Excitation Under Conditions of Strictly Homogeneous Line Broadening.* Proc. R. Soc. Lond. A., 1978. **362**: p. 1-12.

26. Smith, R.A., *Excitations of Transitions between Atomic or Molecular Energy Levels by Monochromatic Laser Radiation. II. Excitation Under Conditions of Strictly*

Inhomogeneous Line Broadening and Mixed Homogeneous and Inhomogeneous Broadening. Proc. R. Soc. Lond. A., 1978. **362**: p. 13-25.

27. Matulic, L. and C. Palmer, *Power-series Solutions to the Optical Maxwell-Bloch Equations*. J. Opt. Soc. Am. B, 1989. **6**(3): p. 356-363.

28. Zi-zhao, G. and Y. Guo-zhen, *Theory of Coherent Propagation of a Light Wave in Semiconductors I*. Phys. Rev. B., 1982. **26**(12): p. 6826-6832.

29. Zi-zhao, G. and Y. Guo-zhen, *Theory of Coherent Propagation of a Light Wave in Semiconductors II*. Phys. Rev. B, 1982. **26**(12): p. 6833-6843.

30. Zi-zhao, G. and Y. Guo-zhen, *Theory of Coherent Propagation of a Light Wave in Semiconductors III*. Phys. Rev. B., 1982. **26**(12): p. 6844-6852.

31. DeVoe, R.G. and R.G. Brewer, *Experimental Test of the Optical Bloch Equations for Solids*. Phys. Rev. Lett., 1983. **50**(17): p. 1269-1272.

32. Yamanoi, M. and J.H. Eberly, *Relaxation Terms for Strong-field Optical Bloch Equations*. J. Opt. Soc. Am. B, 1984. **1**(5): p. 751-755.

33. Yamanoi, T. and J.H. Eberly, *Optical Bloch Equations for Low-Temperature Solids*. Phys. Rev. Lett., 1984. **52**(15): p. 1353-1354.

34. Berman, P.R., *Validity Conditions for the Optical Bloch Equations*. J. Opt. Soc. Am. B, 1986. **3**(4): p. 564-571.

35. Berman, P.R., *Markovian Relaxation Processes for Atoms in Vapours and in Solids: Calculation of Free-induction Decay in the Weak Field Limit*. J. Opt. Soc. Am. B, 1986. **5**(4): p. 572-586.

36. Singh, S. and G.S. Agarwal, *Four-wave Mixing Under Conditions When Optical Bloch Equations Fail*. J. Opt. Soc. Am. B, 1988. **5**(2): p. 254-258.

37. Eilbeck, J.C. and R.K. Bullough, *The Method of Characteristics in the Theory of Resonant or Nonresonant Nonlinear Optics*. J. Phys. A, 1972. **5**: p. 820-829.

38. Caudrey, P.J. and J.C. Eilbeck, *Numerical Evidence for Breakdown of Soliton Behaviour in Solutions of the Maxwell-Bloch Equations*. Phys. Lett., 1977. **62A**(2): p. 65-66.

39. Bullough, R.K., *et al.*, *Solitons in Laser Physics*. Phys. Scripta., 1979. **20**: p. 364-381.

40. Caudrey, P.J., J.C. Eilbeck, and J.D. Gibbon, *Exact Multisoliton Solution of the Reduced Maxwell-Bloch Equations of Non-linear Optics*. J. Inst. Maths. Applics., 1974. **14**: p. 375-386.

41. Caudrey, P.J., *et al.*, *Exact Multisoliton Solution of the Inhomogeneously Broadened Self-induced Transparency Equations*. J. Phys. A., 1973. **6**: p. L53-L56.

42. Gibbon, J.D., *et al.*, *An N-soliton Solution of a Nonlinear Optics Equation Derived by a General Inverse Method*. Lett. Nuovo Cim., 1973. **8**(13): p. 775-779.

43. Gibbon, J.D., *et al.*, *N-Soliton Solutions of some Non-linear Dispersive Wave Equations of Physical Significance*, in *Ordinary and Partial Differential Equations.*, 1974.

44. Eilbeck, J.C., *et al.*, *Solitons in Nonlinear Optics I. A More Accurate Description of the 2π Pulse in Self-induced Transparency*. J. Phys. A., 1973. **6**: p. 1337-1347.

45. Jack, P.M., *Ph. D. Thesis*. 1978, University of Manchester:

46. Tavis, M. and F.W. Cummings, *Exact Solution for an N-Molecule-Radiation-Field Hamiltonian*. Phys. Rev., 1968. **170**(2): p. 379-384.

47. Tavis, M. and F.W. Cummings, *Approximate Solutions for an N-Molecule-Radiation-Field Hamiltonian*. Phys. Rev., 1969. **188**(2): p. 692-695.

48. Tan-no, N., K. Yokoto, and H. Inaba, *Two-photon self-induced transparency in a resonant medium I. Analytical Treatment.* J. Phys. B, 1975. **8**(3): p. 339-348.

49. Tan-no, N., K. Yokoto, and H. Inaba, *Two-photon Self-induced Transparency in a Resonant Medium II. Transient Pulse Behaviour.* J. Phys. B, 1975. **8**(3): p. 349-357.

50. Brewer, R.G. and E.L. Hahn, *Coherent Two-photon Processes: Transient and Steady-state Cases.* Phys. Rev. A, 1975. **11**(3): p. 1641-1649.

51. Grischkowsky, D., M.M.T. Loy, and P.F. Liao, *Adiabatic Following Model for Two-photon Transitions: Nonlinear Mixing and Pulse Propagation.* Phys. Rev. A, 1975. **12**(6): p. 2514-2533.

52. Makhviladze, T.M., I.G. Sinitsyn, and L.A. Shelepin, *Transient Processes in Propagation of an Ultrashort Pulse in a Three-level Medium.* J. Sov. Quantum Electron., 1977. **7**(6): p. 739-741.

53. Radmore, P.M. and P.L. Knight, *Population Trapping and Dispersion in a Three-level System.* J. Phys. B, 1982. **15**: p. 561-573.

54. Obada, A.-S.F. and A.M. Abdel-Hafez, *Time-evolution for a Three-level Atom in Interaction with Two Modes.* J. Mod. Opt., 1987. **34**(5): p. 665-677.

55. Elgin, J.N., *Semiclassical Formalism for the Treatment of Three-level Systems.* Phys. Lett. A, 1980. **80A**(2,3): p. 140-142.

56. Hioe, F.T. and J.H. Eberly, *N-Level Coherence Vector and Higher Conservation Laws in Quantum Optics and Quantum Mechanics.* Phys. Rev. Lett., 1981. **47**(12): p. 838-841.

57. Hioe, F.T. and J.H. Eberly, *Nonlinear Constants of Motion for Three-level Quantum Systems.* Phys. Rev. A, 1982. **25**(4): p. 2168-2171.

58. Gottlieb, H.P.W., *Linear Constants of Motion for a Three-level Atom Excited by Two Modulated Electromagnetic Waves*. Phys. Rev. A, 1982. **26**(6): p. 3713-3715.

59. Hioe, F.T., *Linear and Nonlinear Constants of Motion for Two-photon Processes in Three-level Systems*. Phys. Rev. A, 1984. **29**(6): p. 3434-3436.

60. Gottlieb, H.P.W., *Second Invariant in an Excited Three-level System*. Phys. Rev. A, 1985. **32**(1): p. 653-654.

61. Hioe, F.T., *Dynamic Symmetries in Quantum Electronics*. Phys. Rev. A, 1983. **28**(2): p. 879-886.

62. Oreg, J., F.T. Hioe, and J.H. Eberly, *Adiabatic Following in Multilevel Systems*. Phys. Rev. A, 1984. **29**(2): p. 690-697.

63. Aravind, P.K., *Matrix Solution of Pseudospin Equations for Three-level Systems*. J. Opt. Soc. Am. B, 1986. **3**(7): p. 1025-1032.

64. Hioe, F.T., *Lossless Propagation of Optical Pulses through N-level Systems with SU(2) Symmetry*. J. Opt. Soc. Am. B, 1989. **6**(3): p. 335-343.

65. Eberly, J.H., *et al.*, *Coherent Dynamics of N-level Atoms and Molecules. I. Numerical Experiments*. Phys. Rev. A, 1977. **16**(5): p. 2038-2047.

66. Bialynicka-Birula, Z., *et al.*, *Coherent Dynamics of N-level Atoms and Molecules. II. Analytic Solutions*. Phys. Rev. A, 1977. **16**(5): p. 2048-2054.

67. Cook, R.J. and B.W. Shore, *Coherent Dynamics of N-level Atoms and Molecules. III. An Analytically Soluble Periodic Case*. Phys. Rev. A, 1979. **20**(2): p. 539-544.

68. Konopnicki, M.J., P.D. Drummond, and J.H. Eberly, *Theory of Lossless Propagation of Simultaneous Different-Wavelength Optical Pulses*. Opt. Comm., 1981. **36**(4): p. 313-316.

69. Konopnicki, M.J. and J.H. Eberly, *Simultaneous propagation of Short Different-wavelength Optical Pulses*. Phys. Rev. A, 1981. **24**(5): p. 2567-2583.

Chapter 3

Comparer, c'est comprendre.

To compare is to understand.

French Proverb

3.1 Introduction.

In the previous two chapters we have seen how nonlinear equations are extremely important in describing the effects that may be observed within optical systems. The concept of the soliton has proven to be extremely useful in describing the behaviour of (light) pulses. Moreover, the multitude of papers describing the use of the nonlinear Schrödinger equation in the field of nonlinear optics would lead us to expect that any description of the physical behaviour in this field would involve the solution of non-linear equations. Even the Maxwell-Bloch equations and their derivatives and analogues may be shown to give rise to behaviour which is nonlinear.

Nonlinear partial differential equations are notoriously difficult to solve. If the quantization of the physical systems is also introduced then the level of complexity of the equations increases again. In order to solve such equations the usual approach has been to derive an equation which describes the problem; to simplify it, possibly by using simplifying assumptions; and then to use a computer to solve the equation(s) numerically. Each of these steps will generally have some assumptions built in to it so that the eventual result, although it may give a reasonable and useful description of the overall behaviour, will not necessarily be sufficiently accurate to model the observed behaviour in detail.

In the case of fibre optic transmission systems, the nonlinear effect is much smaller than the other physical effects and it is only the distances involved which allow the effect to be manifested. Similarly, in an integrated optics device, the nonlinear effect is large but the distance of transmission small. The point of course is that in such models it is the presence of comparatively small terms which give rise to the features of interest. Accordingly, it is in the best interests of workers in this field to obtain equations which are as accurate as possible.

There are several ways in which the results obtained can be improved. The accuracy of numerical calculations can always be improved. This however is unlikely to give rise to any qualitatively new behaviour since the underlying equations are unchanged. One can reduce the number of approximations involved in deriving the describing equations from the physical situation or alternatively increase the number of variables in the initial description. This result has paid off in the case of those who have sought to increase the accuracy of the equations describing light pulses in optical fibres

as we have already seen. However, the procedure of adding additional terms still remains somewhat *ad hoc*. The trouble with removing simplifying assumptions is that they have generally been made to make the equations tractable in the first place.

Ideally one would like to see the most exact equation possible being solved as exactly as possible. The contribution that will be made here lies in the derivation of alternative descriptions of the physical problem in the hope that such descriptions will lead to new, and hopefully more productive, ways of considering the equations which describe them. In order to do so, it is necessary to look at solution methods for nonlinear equations. The work of Hasegawa and Kodama discussed in the last chapter, more specifically the work contained in Kodama and Hasegawa's paper [1] , was frighteningly complicated. If any further progress is to be made in describing the systems covered by their equation, it would realistically have to involve some other method which was sufficiently accurate to be able to retain all of the information modelled by their work but which involved some simplification in the process of the mathematics.

It is to this end that we now examine some of the mathematical techniques involved in the solution of nonlinear partial differential equations.

3.2. Techniques for Nonlinear Differential Equations.

Finding solutions to nonlinear partial differential equations is extremely difficult and if exact solutions are required there are only a few special methods which may be applied [2]. For first order nonlinear partial differential equations, Charpit's method and Jacobi's method may be applied to obtain solutions. However, only in special cases is the solution obtained easily since the methods both involve the solution of auxiliary equations which may be difficult to obtain. For second order nonlinear equations the situation is even worse. To quote Sneddon [2], "It is only in special cases that a [nonlinear] partial differential equation... of the second order can be integrated."

The work of Bluman and Cole [3] brought to the attention of researchers the fact that symmetry methods could be applied in an attempt to simplify the solution of differential equations. For first order equations, the determination of the symmetry group of transformations possessed by the equation allows the equation to be solved by quadratures. For higher order systems, however, the determination of the symmetry group of the equation allows one only to reduce the order of the equation by one with the addition of a supplementary equation. If only a general solution of the equations is

required, there is no need to know or use boundary conditions and, in addition, the equations considered may be nonlinear.

This might conceivably be of use in the determination of solutions to the type of partial differential equations which we are required to consider. However, the mathematical machinery for determining the symmetry groups has a highly technical background [4]. Even though there exists a computer algebra (REDUCE) package, SPDE [5], for the determination of the infinitesimal symmetry groups of an equation or systems of equations, the information produced from this tends to be of little help in solving the equations. The problem, of course, is that the equations which we are required to consider are nonlinear, complicated, and have few symmetries. Given the difficulties involved and the relatively unhelpful information which would probably be obtained it was decided that the use of such symmetry methods was likely to be unrewarding given the effort involved.

Of course, it has already been pointed out in the previous chapter that there is one class of nonlinear partial differential equations for which solutions may be found namely those equations which have solitonic solutions and are amenable to treatment using inverse scattering transform techniques. However, solitonic solutions are by no means general, and proving that an equation is completely integrable and is thus likely to have solitonic solutions is again a very difficult process.

In order to solve the equations governing wave propagation in a nonlinear medium, it is therefore necessary to employ some sort of asymptotic method which will approach the true solution to whatever degree of accuracy required. There are several asymptotic methods available for the solution of nonlinear partial differential equations. We now go on to discuss asymptotic solution methods in general. We will then make detailed examinations of some of the methods.

3.2.1. Perturbation Methods.

Again one of the most useful references on the topic is that by Jeffrey and Kawahara [6]. In their book they describe all of the methods used in this thesis and, moreover, give a brief summary of the background knowledge necessary to understand the uses of the methods. Although we will use this section to briefly summarise their main points, the interested reader should refer to the book for a more detailed discussion. We follow Jeffrey and Kawahara's book [6] closely.

One of the most surprising facts about asymptotic expansions concerns their convergence. Suppose that we have two functions $f(\varepsilon)$ and $g(\varepsilon)$. If

$$|f(\varepsilon)/g(\varepsilon)| \text{ is bounded as } \varepsilon \to 0 \tag{3.1}$$

then we can say that $f=O(g)$ and if

$$|f(\varepsilon)/g(\varepsilon)| \to 0 \text{ as } \varepsilon \to 0 \tag{3.2}$$

then we can say that $f=o(g)$. The symbols O and o are known as Landau order symbols and using them we may define what an asymptotic sequence is. If a sequence of functions $\{\phi_i(\varepsilon)\}$ which involves the small parameter ε has

$$\phi_{i+1}(\varepsilon) = o(\phi_i(\varepsilon)) \text{ as } \varepsilon \to 0, \ \forall i \geq 0 \tag{3.3}$$

then the sequence is called an asymptotic sequence. If a function $f(x;\varepsilon)$ which additionally depends on the variable x is approximated by the series

$$\sum_{i=0}^{j} \phi_i(\varepsilon) f_i(x) \text{ as } \varepsilon \to 0 \tag{3.4}$$

so that

$$f(x;\varepsilon) = \sum_{i=0}^{j} \phi_i(\varepsilon) f_i(x) + o(\phi_j(\varepsilon)) \text{ as } \varepsilon \to 0 \tag{3.5}$$

then this expression is called an asymptotic approximation to $j+1$ terms.

If this expression holds for every $j \geq 0$, then it is called an asymptotic expansion of $f(x;\varepsilon)$ as $x \to 0$. Moreover, if the expression holds uniformly for all x, then it is said to be uniformly valid and nonuniformly valid otherwise. Having more or less transcribed the definitions as given by Jeffrey and Kawahara, we now move on to the main points.

The first of these points is that, although convergent series are always asymptotic, the converse is not always true. In fact, asymptotic series are usually divergent. However, in an asymptotic series, one does not require an infinite series to describe the function to be approximated. Only the first few terms of the asymptotic expansion are required if ε is small enough. Secondly, differentiation and multiplication of asymptotic expansions is not always valid. Throughout this thesis, we assume that

any multiplication or differentiation of series expansions is valid. This should be borne in mind as a possible source of error.

It has been pointed out that the type of problem we are dealing with is a singular perturbation problem. We have not yet explained what this is. If a series expansion of the type given previously is valid throughout the range of x as $\varepsilon \to 0$, then it is called a regular perturbation problem. If, on the other hand, it does not have a uniform limit in some regions of the variable x as $\varepsilon \to 0$, then the expansion is called a singular perturbation problem. The regions in which regular perturbations break down are called regions of nonuniformity.

There are several classifications of singular perturbation problems. We follow Jeffrey and Kawahara in quoting Nayfeh [7]. Singular perturbation problems may be attributed to differing classifications depending on whether

1). sources of nonuniformities appear in relation to an infinite domain (for example, the appearance of secular terms in nonlinear oscillations);

2). a small parameter multiplies the highest-order derivative term in a differential equation;

3). there is a change of type of a partial differential equation;

4). the presence, or occurrence, of singularities.

All of the problems that we consider in this thesis fall into the first of these classifications. That is, the appearance of secular terms within a regular perturbation expansion renders such methods unusable. Instead there is a variety of techniques specially developed to deal with these methods.

Essentially, the reason why a regular expansion of an equation of secular type fails is that a regular power series expansion uses the linearised period of oscillation in the expansion. At large times, however, the difference in phase between the exact and linearised phase becomes arbitrarily large even when the difference between the periods is comparatively small. Hence to construct an expansion free of secular terms involves the use of periodic functions that have a period which is closer to, or exactly, the correct period. This obviously involves a change of coordinates and there are several ways in which this coordinate change can be introduced.

The method of optimal coordinates introduces a scale transformation in the period of the oscillation where the change of scale is based on the difference between the first order estimate of the result and the linearised period. The dependent function is then expanded in a power series in some small parameter. If the physical quantities as well as

the dependent function are expanded in a power series in a small parameter and the period scaled in this way, the method is called Poincaré's method. Another related method is called the method of strained coordinates. In fact the method of strained coordinates can be considered to be a generalisation of Poincaré's method

All of these methods are described fully in Jeffrey and Kawahara [6]. We shall not use these methods and so we refer the interested reader to this reference. There are some further methods which are described, however, which do have a bearing on the version of Whitham's method that we will eventually use and so we now go on to discuss them briefly.

3.2.2. Methods Related to Whitham's Method.

There are several methods which are related to Whitham's method. Again, we follow Jeffrey and Kawahara closely. The method of averaging is used when the solution to a wave equation may be described in terms of two scales - a rapidly varying part and a slowly varying part. The rapidly varying part will generally contain information which is largely superfluous in terms of physical applications and so, by integrating over a period of the fast scale, we can dispense with the information carried on this scale and examine instead the behaviour of the envelope. The scale used is generally one of time: it can occasionally be useful to think of it being the distance scale as in the case of finely layered media. In order to apply this method to give higher order solutions, one is required to make estimates of the order of derivative terms and thus the method can be considered to be less systematic than the previously listed ones.

The Krylov-Bogliubov-Mitropolsky (K.B.M.) method is a systematic way of generating periodic solutions to nonlinear oscillation problems. In the case of the K.B.M. method, the method may be summarised by a formal expansion scheme. The solution to a one dimensional problem given with the small parameter ε set to 0 is assumed to be $f = a\cos(t+\Phi)$ where a is a constant amplitude and Φ a constant phase Suppose that the amplitude and phase are then assumed to be slowly varying functions of t. Then the derivatives with respect to t are slowly varying quantities and so we can say that

$$\frac{da}{dt} = O(\varepsilon), \quad \frac{d\Phi}{dt} = O(\varepsilon) \tag{3.6}$$

If the perturbation expansion for the solution is given as

$$f = a\cos\theta + \sum_{n=1}^{\infty} \varepsilon^n f_n(a,\theta),\ \theta = t + \Phi \tag{3.7}$$

then one may introduce the slow variations in amplitude and phase by taking

$$\frac{da}{dt} = \sum_{n=1}^{\infty} \varepsilon^n A_n(a) \tag{3.8}$$

together with

$$\frac{d\theta}{dt} = 1 + \sum_{n=1}^{\infty} \varepsilon^n \Theta_n(a) \tag{3.9}$$

Then the second order derivatives can be shown to be of order ε^2. Higher orders behave similarly. If one goes on to apply this expansion scheme, one finds that to avoid secular terms, additional conditions must be satisfied. This method is completely systematic and can be applied to any required order and in more dimensions.

It will be noted that in the K.B.M. method, it was the definition of the derivative terms in terms of a power series in the small parameter which allowed the solution to be determined. A final method which is especially closely connected to the final variant of Whitham's method eventually chosen is that of the method of multiple scales in which the expansion of derivative terms is made a property of the derivative operators themselves rather than by defining individual derivatives in terms of power series.

In the case of a one dimensional problem (x,t) the time variable t is replaced by a sequence of variables of increasing scale. We define

$$t_n = \varepsilon^n t, \qquad n = 0..N \tag{3.10}$$

and expand the dependent variable by using

$$f(t;\varepsilon) = \sum_{n=0}^{N} \varepsilon^n f_n(t_0, t_1, \ldots t_n) + O\left(\varepsilon^{N+1}\right) \tag{3.11}$$

where each f_n is assumed to be a function of each of the new time variables. Then we have

$$\frac{d}{dt} = \sum_{n=0}^{N} \varepsilon^n \frac{\partial}{\partial t_n} \tag{3.12}$$

This replacement of the derivative operator with a series of derivative operators gives rise to the name of the variant of the multiple scales technique presented here: the derivative expansion method. Again this method is completely general and may be extended to any number of dimensions and dependent functions. It is on this method combined with the method of averaging that the Kawahara variant of the Whitham technique is based.[1]

3.3 Whitham's Method.

We have seen how the investigation of singular-perturbation problems may be tackled using a variety of solution methods. We have also seen how the use of these methods involves the elimination of secular terms by means of the solution of subsidiary equations as a prerequisite to the solution of the main equation. However, what is required is a method that will allow us to solve partial differential equations without involving us in complicated mathematical manipulations. If there were some method that avoided the generation of secular terms, the solution process would be considerably simplified.

Whitham's method does exactly this. It provides a method of asymptotically solving wave equations which avoids generating secular terms. More to the point, it provides a general and powerful solution method which may be usefully applied to nonlinear equations providing certain qualifying conditions are satisfied.

[1] For an example of secular terms being generated in a derivation involving the use of the method of multiple scales, the derivation of the nonlinear Schrödinger equation in a periodic structure by Sipe [Sipe, J. E. and H. G. Winful. "Nonlinear Schrödinger Solitons in a Periodic Structure." Opt. Lett. 13(2): 132-134, 1988.] may prove to be of interest.

3.3.1. Whitham's Original Method.

There are several variants of Whitham's method. As will be seen, the original exposition of the method as made by Whitham was flawed in one important respect. However, several authors sought to improve upon the method by introducing variants of their own. Some of these variants were quite general in their application whereas others were adapted in order to describe particular feature which might be found in a physical situation such as dissipation. We now go on to describe Whitham's original method, the method upon which the other variants are based. We will save the description of variants for later sections.

3.3.1.1. Introduction.

In previous sections we have seen how perturbation methods may be used to solve nonlinear differential wave equations. We have also noted how the manipulations involved tend to be tedious, involving the solution of subsidiary equations to eliminate secular terms. In the 1950's and 1960's, a fruitful time for research into nonlinear solution methods, an elegant method was put forward by the mathematician Gerald B. Whitham which would avoid the need to eliminate these secular terms. All of the required manipulations are handled within the formalism of the method. This method is put forward by Whitham in a series of papers which starts in 1965.

In the first paper in the series [8] , Whitham introduces a method which he describes as an extension of "the Krylov-Bogliubov averaging technique for non-linear vibrations". The theory is presented as a general method for studying the variations in behaviour of nonlinear wavetrains based on the method of averaging. An investigation of conserved quantities within the method leads Whitham to suggest that there might be some connection between his method and lagrangian dynamics although the bulk of the paper is concerned with the presence or absence of shock waves in the solutions to some sample equations considered within the paper. At this point there is no use of lagrangians within the method.

However, later in the same year, a second paper by Whitham [9] describes how lagrangians may be incorporated within the averaging method discussed in the previous paper. By noting the similarity between certain expressions obtained in the earlier paper and those to be found in Lagrange-Hamilton dynamics, Whitham is able to formulate a

method which will describe the propagation of slowly varying wavetrains. The restriction that the wavetrains be slowly varying is required in that it is a necessary prerequisite for averaging: if the solution is not slowly varying, the averaging process will discard meaningful information.

In the paper, Whitham points out that the equations derived could be obtained from the averaging process alone but that the use of the lagrangian method gives them a deeper significance. A further point of note is that Whitham uses Legendre transformations to show how the averaged lagrangians he obtains may be written in hamiltonian form. The use of the hamiltonian form within the method will later prove to be of some utility. Finally Jeffrey and Kawahara [6] trace the use of a variational principle of this type to a paper written by Sturrock [10] in 1958.

In his next paper on the topic [11], Whitham uses his newly formulated theory to examine a general lagrangian theory for the propagation of water waves. Using a lagrangian given by Luke [12] in a paper which additionally includes a justification of Whitham's method to first order as its first section, Whitham goes on to derive several equations dealing with water wave propagation and to compare them with equations derived by other methods. He also mentions a paper by Lighthill [13] which attempts to suggest experiments by means of which Whitham's method could be verified. There is also a paper by Bretherton [14] in which the author gives some justification of Whitham's method.

The next paper in the series [15] discusses and summarises Whitham's previous results with a bias towards applications in the field of water waves. Whitham discusses the relationship of the method to others in the field and also discusses links with stability theory and wavetrains with multiple periodicities. However, the mechanics of the method remain unchanged.

The final paper of the series [16], gives a formal justification of the method as the first order result in a perturbation expansion. Not only does Whitham give the method a more rigourous justification than in previous papers but he also gives a fuller exposition of the method and the use of Legendre transformations in applying the method to physically useful equations. It should be remembered here that the paper by Luke [12] has already verified that Whitham's method is a valid one, at least to first order. As Whitham points out, however, the proof which he supplies is more intuitive although it covers the same ground. This paper contains the most explicit explanation of the method given by Whitham in his papers.

In the sections of Whitham's book [17] which refer to the averaged lagrangian method, Whitham restates more or less the same work as that in this final paper although it is slightly more detailed. One of the examples used concerns the application of the theory to a model of electromagnetic wave propagation in which a field of electrons is considered to be held in a nonquadratic potential field. This means that the equations of motion for the electrons and hence the polarisability are nonlinear. This model is described further by Small in his Ph.D. thesis [18].

One point made by Whitham is that his method contains an inaccuracy in that it is incapable of describing higher order effects. Whitham suggests how this may be rectified but goes into little detail. The best explanation of why Whitham's method will fail in its unmodified form is given by Newell in his book [19]. The best exposition of the method is arguably given either by Jeffrey and Kawahara [6] or by Whitham himself [17].

3.3.1.2. The Method: an Introduction.

As Newell [19] points out, the idea behind the Whitham method is actually quite simple. Suppose that for some fully nonlinear partial differential equation, there exists a known travelling wave solution

$$f(\theta, A), \text{ where } \theta = kX - \omega T \tag{3.13}$$

Then it is possible to use this solution to obtain a further class of solutions in the case where the wave train is slowly varying. If the wavetrain is slowly varying then the former constants of wavenumber and frequency can be redefined as

$$k = \theta_X, \quad \omega = -\theta_T \tag{3.14}$$

Thus the frequency and amplitude now become functions of X and T, the position and time. We also allow the amplitude A to vary, again as a function of X and T. We can now (implicitly) define new space and time scales by setting

$$x = \varepsilon X, \quad t = \varepsilon T, \text{ where } \varepsilon << 1 \tag{3.15}$$

We now have three unknowns, ω, k, and A and so we will require three equations to solve for them.

The first of the three equations simply expresses the conservation of wave number

$$k_t + \omega_x = 0 \tag{3.16}$$

and is a consequence of the equivalence of mixed derivatives. One obtains the others by substituting some guess for the form of the solution into the defining partial differential equation and then imposing the condition that the periodicity of the waveform is fixed. The first order result, that arising when no variation of the three parameters is taken into account, or alternatively the relationship arising when the lowest power of the small parameter ε is involved, is called the dispersion relation. It relates the three unknowns algebraically. The next order result, together with the imposition of periodicity, results in a differential equation which must be satisfied if the equations are to be solved.

The manner in which the method is implemented is slightly different in that the suggested form for the solution is not substituted into the defining differential equation but into a variational principle which defines the differential equation by virtue of having it as its equation of motion. The resulting expression is then averaged by integrating it over a period of the original waveform and it is this step which results in there being no secular conditions. The integration over a fixed period fixes the relationship between the variables in a way that simulates the elimination of secular terms in other methods.

It will be noted that these manipulations make certain basic assumptions in order to make the method feasible. The first of these assumptions is that it is possible to impose periodicity on the waveform. This is vital to the success of the method. If there is no fixed periodicity, that is if the periodicity is allowed to be slowly varying say, then it is impossible to derive additional conditions which will determine the system. Another way of saying this is that the problem of secularity is again felt.

The second assumption made is that there can be some meaningful separation of scales. In other words, in defining the slow and fast variables, it is necessary to introduce the small parameter ε. This parameter is a measure of the difference between the scales. On the finer, fast scale, that involving X and T, it is assumed that the defining equation can be assumed to be locally valid without recourse to assuming that the wavenumber and frequency are slowly varying. That is, on this scale, the solution will be some local variant of the travelling wave solution from which the fixed periodicity is taken. The behaviour on the scale involving x and t will describe the macroscopic behaviour of the wave: it will describe the wave envelope. Thus, in order to apply the method successfully, there must be some sort of physical situation of the sort shown in diagram 3.1.

The final assumption made is that a lagrangian description of the problem exists. It is well known that lagrangians describing particular physical situations are not unique. There may be more than one lagrangian whose derived equation of motion is the equation to be studied. On the other hand, there is no method for deriving a lagrangian to describe a physical problem or to derive a lagrangian from a given equation. Lagrangians tend to be found by experience or trial and error. In a paper by Seliger and Whitham [20], the authors discuss this problem and give some lagrangians for some physical situations, that for the Maxwell equations for example. They state that in determining lagrangians, the use of a potential representation for the physical problem can often be productive. It seems that the representation chosen for the problem can often be decisive in determining the ease with which a lagrangian can be found or utilised.

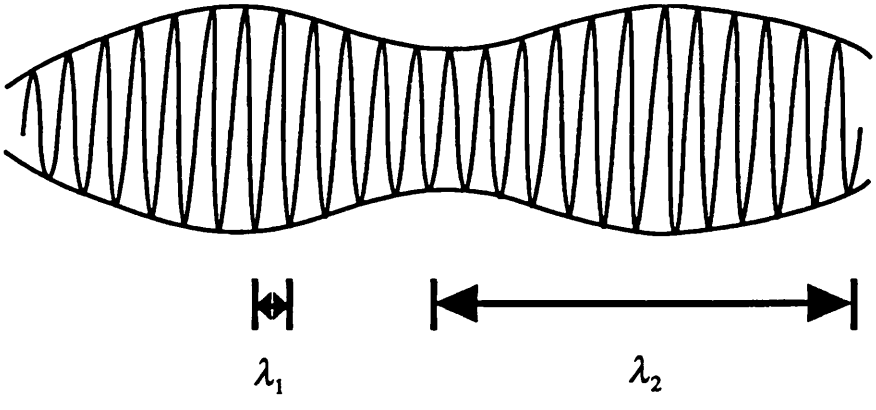


Diagram 3.1.
The necessary separation of scales.

The requirements necessary for the use of the Whitham technique (or its variants) may therefore be summarised as follows.

1). There must be a lagrangian representation for the problem to be studied.

2). The solution to be sought must involve a separation of scales. In other words, the method is to be applied where there is a slow variation of the defining parameters of the equation but where the wave motion may be considered to be described locally as having those parameters fixed.

3). The underlying waveform must have a fixed periodicity in order that the averaging may be performed.

If these three points hold, then Whitham's method may be applied. There is also a further assumption made in the use of a lagrangian and that is that the system being described is conservative. Although we will see a version of the method in which dissipation is incorporated into the formalism, Whitham's original method did not allow for nonconservative lagrangians.

3.3.1.3. The Method.

We now describe Whitham's original method following the description given in Jeffrey and Kawahara [6] very closely. We will subsequently use the example they present, that of the Boussinesq equation, to explain and exemplify the use of the Kawahara variant of Whitham's method. Accordingly, the solution of the same equation using the original approach is given for comparison.

For a lagrangian density given by

$$\mathcal{L}(\Phi_t, \Phi_x, \Phi) \tag{3.17}$$

(where subscripts denote partial differentiation) we assume that the variational principle given by

$$\partial \int\int \mathcal{L}(\Phi_t, \Phi_x, \Phi) dx dt = 0 \tag{3.18}$$

gives rise to the equations of motion for the physical system under consideration. The Euler-Lagrange equations for this lagrangian are given by

$$\frac{\partial}{\partial t}\left(\frac{\partial \mathcal{L}}{\partial \Phi_t}\right) + \frac{\partial}{\partial x}\left(\frac{\partial \mathcal{L}}{\partial \Phi_x}\right) - \frac{\partial \mathcal{L}}{\partial \Phi} = 0 \tag{3.19}$$

It is assumed that the first order solution to the equations has the form

$$\Phi = \phi(\theta) = a\cos(\theta + \eta_0), \text{ where } \theta = kx - \omega t \tag{3.20}$$

where the amplitude, wavenumber and frequency are assumed to be slowly varying functions of space and time.

One substitutes the above expression into the lagrangian and then averages over the rapidly varying phase θ. One obtains the averaged variational principle

$$\delta\int\int \mathcal{L}(\Phi_t,\Phi_x,\Phi)\mathrm{d}x\mathrm{d}t = 0 \tag{3.21}$$

where

$$\bar{\mathcal{L}} = \frac{1}{2\pi}\int_0^{2\pi} \mathcal{L}\mathrm{d}\theta,\ \mathcal{L} = \mathcal{L}(\omega,k,a,\theta) \tag{3.22}$$

The averaging is carried out with respect to θ keeping the variables ω,k,a constant. We state that

$$k = \frac{\partial\theta}{\partial x},\ \omega = -\frac{\partial\theta}{\partial t} \tag{3.23}$$

so the averaged variational principle should now be thought of as a variational principle for a and θ. We can now use the Euler-Lagrange equations for these variables to obtain the averaged equations of motion for the system as

$$\frac{\partial\bar{\mathcal{L}}}{\partial a} = 0 \tag{3.24}$$

and

$$\frac{\partial}{\partial t}\left(\frac{\partial\bar{\mathcal{L}}}{\partial\theta_t}\right) + \frac{\partial}{\partial x}\left(\frac{\partial\bar{\mathcal{L}}}{\partial\theta_x}\right) = 0 \tag{3.25}$$

with the additional condition that

$$\frac{\partial k}{\partial t} + \frac{\partial\omega}{\partial x} = 0 \tag{3.26}$$

Equations (3.24) to (3.26) represent a solution of the physical problem in that (3.24) is the dispersion relation for the problem, (3.25) is the equation which describes the modulation of the waveform and (3.26) expresses the conservation of wavenumber. Although these results are presented for one function dependent on one dimension plus time, they may easily be generalised to multiple equations in three-dimensions. Higher

order derivatives may also be introduced with a corresponding change in the form of the Euler-Lagrange equations. The necessary form can be found in books on the calculus of variations, for example Elsgolc [21].

For a concrete example, we use the Boussinesq equation

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} - \mu \frac{\partial^4 u}{\partial x^2 \partial t^2} = \frac{1}{2} \frac{\partial^2 u^2}{\partial x^2} \tag{3.27}$$

We mentioned in the previous section that, in finding a lagrangian representation of an equation, the potential representation for the problem can often be helpful [20]. Accordingly we take

$$u = f_x \tag{3.28}$$

and , after an integration, we obtain (denoting partial differentiation by subscripts)

$$f_{tt} - c^2 f_{xx} - \mu f_{xxtt} - f_x f_{xx} = 0 \tag{3.29}$$

as the potential description of the equation. If one then assumes that the lagrangian has the form

$$\mathcal{L} = \tfrac{1}{2} f_t^2 - \tfrac{1}{2} c^2 f_x^2 + \tfrac{1}{2} \mu f_{xt}^2 - \tfrac{1}{6} f_x^3 \tag{3.30}$$

such that

$$\mathcal{L} = \mathcal{L}(f_t, f_x, f_{xt}) \tag{3.31}$$

then, using the Euler-Lagrange equation

$$\frac{\partial}{\partial t}\left(\frac{\partial \mathcal{L}}{\partial f_t}\right) + \frac{\partial}{\partial x}\left(\frac{\partial \mathcal{L}}{\partial f_x}\right) - \frac{\partial^2}{\partial x \partial t}\left(\frac{\partial \mathcal{L}}{\partial f_{xt}}\right) = 0 \tag{3.32}$$

which corresponds to the variational principle

$$\delta \iint \mathcal{L}(f_t, f_x, f_{xt}) \mathrm{d}x\mathrm{d}t = 0 \tag{3.33}$$

one may reobtain the potential description as its equation of motion.

We note here that the linearised version of (3.29)

$$f_{tt} - c^2 f_{xx} - \mu f_{xxtt} = 0 \tag{3.34}$$

has as its dispersion relation

$$D(\omega,k) \equiv c^2k^2 - \omega^2 - \mu\omega^2k^2 = 0 \tag{3.35}$$

which may be calculated by making the substitution $f = a\cos(\theta+\beta)$ where $\theta = kx - \omega t$ in (3.34). The definition of θ also implies (3.23). The dispersion relation will be used for notational convenience in subsequent results.

The expansion for the nonlinear problem is assumed to be a perturbed plane wave which can be written as

$$f = \varepsilon a\cos(\theta+\beta) + \varepsilon^2 \sum_{n=2}^{\infty} a_n \cos(n\theta+\beta_n) + O(\varepsilon^3) \tag{3.36}$$

where $a,k,\omega,\beta,a_n,\beta_n$ are assumed to be slowly varying functions of x,t. We see from the form of the lagrangian that the expansions of the derivatives $f_t^2, f_x^2, f_{tt}^2, f_x^3$ are required. For example f_t^2 is given by

$$f_t^2 = \varepsilon^2\left(\frac{\partial a}{\partial t}\right)^2 \cos^2(\theta+\beta) + \varepsilon^2 a^2 \sin^2(\theta+\beta)\left(-\omega + \frac{\partial \beta}{\partial t}\right)^2$$
$$-2\varepsilon^2 a \frac{\partial a}{\partial t}\cos(\theta+\beta)\sin(\theta+\beta)\left(-\omega + \frac{\partial \beta}{\partial t}\right) + O(\varepsilon^3) \tag{3.37}$$

and similarly for the other derivatives. However, this result retains too many terms and so we discard all higher order derivatives and products of derivatives. We obtain, within this approximation to $O(\varepsilon^4)$

$$f_x^2 = \varepsilon^2 a^2 k^2 \sin^2(\theta+\beta) + 2\varepsilon^2 a^2 k \frac{\partial \beta}{\partial x} \sin^2(\theta+\beta) - 2\varepsilon^2 ak \frac{\partial a}{\partial x} \cos(\theta+\beta)\sin(\theta+\beta)$$

$$f_t^2 = \varepsilon^2 a^2 \omega^2 \sin^2(\theta+\beta) + 2\varepsilon^2 a^2 \omega \frac{\partial \beta}{\partial t} \sin^2(\theta+\beta) - 2\varepsilon^2 a\omega \frac{\partial a}{\partial t} \cos(\theta+\beta)\sin(\theta+\beta)$$

$$\begin{aligned} f_{xt}^2 &= \varepsilon^2 a^2 k^2 \omega^2 \cos^2(\theta+\beta) \\ &+2\varepsilon^2 a \frac{\partial a}{\partial x} \cos(\theta+\beta)\sin(\theta+\beta) k\omega^2 \\ &-2\varepsilon^2 a \frac{\partial a}{\partial t} \cos(\theta+\beta)\sin(\theta+\beta) k^2 \omega \\ &+2\varepsilon^2 a^2 k \omega^2 \frac{\partial \beta}{\partial x} \cos^2(\theta+\beta) \\ &+2\varepsilon^2 a^2 k^2 \omega \frac{\partial \beta}{\partial t} \cos^2(\theta+\beta) \end{aligned}$$

$$f_x^3 = \varepsilon^3 a^3 k^3 \sin^3(\theta+\beta)$$

(3.38a,b,c,d)

If we substitute the above expressions for the derivatives into the expression for the lagrangian and then perform averaging over θ according to

$$\overline{L} = \frac{1}{2\pi} \int_0^{2\pi} L \mathrm{d}\theta$$

(3.39)

we will obtain, making use of standard reduction formulæ

$$\begin{aligned} \overline{L} &= \tfrac{1}{4}\varepsilon^2 \left(\omega^2 - c^2 k^2 + \mu k^2 \omega^2\right) a^2 \\ &-\tfrac{1}{2}\varepsilon^2 \left\{ \left(\omega\left(1+\mu k^2\right) \frac{\partial \beta}{\partial t} \right) + \left(k\left(c^2 - \mu\omega^2\right) \frac{\partial \beta}{\partial x} \right) \right\} a^2 + O\left(\varepsilon^4\right) \end{aligned}$$

(3.40)

Changes of β in x and t are assumed to be of order ε so that the second term in the above expression is of order ε^3.

The averaged variational principle is now given by

$$\delta \iint \overline{L}(\omega, k, a, \beta) \mathrm{d}x\mathrm{d}t = 0$$

(3.41)

where x,t are regarded as slow variables. In this variational principle ω and k cannot be varied independently since they are related by the equation (3.26) expressing conservation of wavenumber. Thus the variational equations must be derived from the variations with respect to a and θ. To find the variation with respect to θ we use the definitions of ω and k. That is, instead of using the Euler-Lagrange equation

$$\frac{\partial}{\partial t}\left(\frac{\partial \bar{L}}{\partial \theta_t}\right)+\frac{\partial}{\partial x}\left(\frac{\partial \bar{L}}{\partial \theta_x}\right)=0 \tag{3.42}$$

we use it in the form

$$-\frac{\partial}{\partial t}\left(\frac{\partial \bar{L}}{\partial \omega}\right)+\frac{\partial}{\partial x}\left(\frac{\partial \bar{L}}{\partial k}\right)=0 \tag{3.43}$$

After substitution for the form of $\partial\bar{L}/\partial\omega$ and $\partial\bar{L}/\partial k$ (3.43) becomes

$$\begin{aligned}&\varepsilon^2\left[-\frac{\partial}{\partial t}\left\{\tfrac{1}{2}\left(1+\mu k^2\right)\omega a^2\right\}+\frac{\partial}{\partial x}\left\{\tfrac{1}{2}\left(\mu\omega^2-c^2\right)ka^2\right\}\right]\\&+\varepsilon^2\left[\frac{\partial}{\partial t}\left\{\tfrac{1}{2}\left(1+\mu k^2\right)\frac{\partial\beta}{\partial t}a^2-\mu k\omega\frac{\partial\beta}{\partial x}a^2\right\}\right.\\&\left.-\frac{\partial}{\partial x}\left\{-\tfrac{1}{2}\left(\mu\omega^2-c^2\right)\frac{\partial\beta}{\partial x}a^2+\mu\omega k\frac{\partial\beta}{\partial t}a^2\right\}\right]\\&+O\left(\varepsilon^5\right)=0\end{aligned} \tag{3.44}$$

The corresponding Euler-Lagrange equation for the variable a yields

$$\tfrac{1}{2}\varepsilon^2\left(\omega^2-c^2k^2+\mu\omega^2k^2\right)a-\varepsilon^2\left[\omega\left(1+\mu k^2\right)\frac{\partial\beta}{\partial t}+k\left(c^2-\mu\omega^2\right)\frac{\partial\beta}{\partial x}\right]a=0 \tag{3.45}$$

We notice that the lowest order consequence of (3.45) is the linearised dispersion relation derived previously. The equations at the next order are, according to Jeffrey and Kawahara [6]

$$\frac{\partial\beta}{\partial t}+V_g\frac{\partial\beta}{\partial x}=0 \tag{3.46}$$

from the dispersion relation equation and

$$\frac{\partial}{\partial t}\left(a^2\right)+\frac{\partial}{\partial x}\left(V_g a^2\right)=0 \tag{3.47}$$

from the modulation equation. Together with the linear dispersion relation and the equation expressing the conservation of wavenumber, these equations are the lowest

order results of Whitham theory applied to the Boussinesq equation. We discuss their comments on these equations in the next section.

3.3.1.4. Failures and Limitations of the Original Method.

We have already pointed out (section 3.3.1.1) that Whitham was aware that his method contained an inaccuracy in that it did not correctly model higher order effects. A further more explicit description of the failure of Whitham's method to retain all of the original information present in the describing equations is given in the book by Newell [19]. That the equations derived by using Whitham's original method discard information is further restated by Jeffrey and Kawahara [6] whilst commenting on the equations derived in the previous section. Although (3.46) and (3.47) are correct to the order given, the neglect of differential terms means that the results up to this order are incapable of describing the wave modulation which gives rise to the nonlinear Schrödinger equation. In order for the formalism to retain all of the information inherent within the mathematical model, it is necessary for it to retain all of the higher order terms, and further, to do so in a systematic way.

A practical demonstration of the limitations of the averaged lagrangian method is given in the thesis by Jack [22].This contains details of the similarity between the Maxwell-Bloch, reduced Maxwell-Bloch and self-induced transparency equations as well as results relating to the application of Whitham's method to them. We mentioned this thesis in the section which discussed these equations. The main topic of Jack's work is the similarity between the slowly varying envelope and phase approximation (which we used both in the derivation of the two-level Maxwell-Bloch equations and in the section discussing the Lie group representation of the three- and N-level systems) and the assumption that there is a separation of scales inherent in the use of the averaged lagrangian techniques. Jack states:

The slowly varying envelope and phase approximation assumes that the field consists of a slowly varying envelope modifying a carrier. [Higher order derivatives are subsequently neglected to simplify the equations.] *The averaged lagrangian technique assumes that the field is given locally by a uniform travelling wave, although the amplitude and frequency may vary over longer intervals. The amplitude, frequency and wave number of the travelling wave are related by a nonlinear dispersion relation whereas the slowly varying envelope and phase*

approximation treats the envelope and pulse as separate quantities and does not make use of a dispersion relation.

Her conclusion is that:

The averaged lagrangian method provides a mathematical basis for the slowly varying envelope and phase approximation. For problems in which it can be implemented fully, it has the advantage of collecting all the information on the slowly varying amplitude, frequency, and wave number in an averaged variational principle. However, its application to problems in nonlinear optics above zero order results in practical difficulties. Although the method can usually be applied at zero order, this is not sufficient to describe many phenomena, such as self-induced transparency, and in such cases alternative methods such as the slowly varying envelope and phase approximation must be used.

Thus, in addition to the difficulty in obtaining suitable lagrangians to describe the physical situation to be modelled, it was also discovered that the application of Whitham's averaged lagrangian techniques to problems in nonlinear optics led to equations which did not fully describe the physical situation, since it was necessary to neglect higher order terms in order for the calculations to proceed.

We will see in subsequent sections that there is a variant of Whitham's method due to Kawahara in which the higher order terms are retained in a systematic manner. The above comments by Jack should not be thought of as applying to the Kawahara variant since it manifestly is capable of retaining sufficient information for the correct limiting cases to be derived from the equations generated. For example, the Kawahara variant does give rise to the nonlinear Schrödinger equation in the appropriate limits. However, Jack's comments do apply to Whitham's original method. It is this limitation which renders the original method unsuitable for the detailed modelling of nonlinear phenomena: insufficient information is retained by the technique.

3.3.2. Variants of Whitham's Method.

We have already seen that there are several variants of Whitham's method. We have mentioned a variant used by Small [18] already. There are many others: we will mention the variants by Dewar [23], Dysthe [24], and a variant which might be

considered a separate entity by Chu and Mei [25] only in passing. There are, however, other variants which deserve further mention.

The first reference which should be mentioned is not strictly speaking a variant of Whitham's method at all but rather a paper in which it is stated that there is a connection between Whitham's method and the techniques based on the inverse scattering transform [26]. The treatment given by Kamchatnov is one in which the inverse scattering formalism is used to derive the solutions for the wave motion and it is these solutions which are then inserted into Whitham's method and solved for particular cases. Equations are derived describing the formations of solitons at the front of a long light pulse, the equations describing the system being the self-induced transparency equations discussed in chapter 2. There is little discussion of the actual connection between the two methods, however, and the author is not convinced of the utility of Whitham's method. Thus his contention that there is a link between the two methods can be considered to be speculative. There appears to be little research concerning possible links between the two methods at present.

As we have already stated, there are several variants of Whitham's method. We will mention a few briefly. The variant by Newell [19], for example, is similar to that of Kawahara [27] in that it seeks to systematically include higher order terms. Newell's variant is not as systematic as Kawahara's and so we have not chosen to use it in implementing the computer program which was eventually written to investigate higher order effects.

There are variants by Ostrovsky and Pelinovsky [28, 29], and by Lavenda [30, 31], which seek to extend the Whitham technique to deal with nonconservative fields. There is also a variant of the Whitham technique by Ablowitz and Benney [32, 33] which seeks to allow multiple periodicities within the waveform to be considered by the averaging technique. We now go on to discuss these variants in greater detail since it is conceivable that the extension of the program to deal with dissipation and multiple periodicity might be useful.

3.3.2.1. Variants for use with Dissipative Systems.

A description of wave propagation in dissipative systems would be useful in the study of, for example, light propagation in optical fibres. However, the application of lagrangian techniques to nonconservative systems is not as simple as that for conservative ones since, for conservative systems, the lagrangian can be shown to

follow from an established physical principle, the principle of least action. For nonconservative systems, it has been a commonly held misconception that variational techniques are not applicable.

This is not the case. In a paper by Van Der Vaart [34], it is shown that for a dissipative system having one degree of freedom, a lagrangian exists. For systems having more than one degree of freedom, it may be shown that a lagrangian may be constructed only if a certain relationship holds between matrices of the coefficients of the dissipative system. Whilst this does not prove that the type of dispersive, dissipative system to be found in nonlinear optics will necessarily have lagrangian descriptions, it does mean that their existence is not ruled out.

In fact there have been two modifications of Whitham's method which have been designed to consider dissipative systems. The first of these is based on a thermodynamic approach. Using formalism developed for the description of processes in nonequilibrium thermodynamics [30], a paper by Lavenda and coworkers [31] generalizes Whitham's method to include nonlinear dissipation. The authors make a distinction between linear and nonlinear dissipation. In the case where the dissipative term is linear, their method is similar to the next method to be discussed although it appears to include a simplifying assumption. In the case where the dissipative term is nonlinear, the method is mathematically rather messy and dependent upon the equation being considered. The authors' contention is that the presence of dissipation rules out the use of mechanical models and means that the variational principles to be used should be those of irreversible thermodynamics.

A second approach is suggested by Ostrovsky and Pelinovsky [28, 29]. They suggest modifying Whitham's method to cover the case of dissipation by utilising averaged Rayleigh dissipation functions. Essentially, the lagrangian formalism is modified so that the virtual work done when taking the variation of the lagrangian functional is no longer zero but some function which describes the averaged work done by the system against the dissipative forces. It may be seen how there is some similarity between this method and the method of Lavenda when applied to systems having linear dissipative terms. There is some additional discussion of this method in a review paper by Gorschkov [35] in which the method is compared with other methods in the field of asymptotic nonlinear field theory. Jeffrey and Kawahara also discuss the method in their book [6] where they expand slightly upon the derivations given by Ostrovsky and Pelinovsky.

As has already been mentioned, it would conceivably be useful to implement some model which could successfully model dissipation within nonlinear dispersive systems. The similarity of the technique to that of Whitham means that this is the modification which would be most easily implemented as part of the computer algebra program to calculate the equations of motion for averaged lagrangians. Some discussion of the modifications which would be required to the program to implement this variant of the Whitham method are given in chapter 6.

3.3.2.2. Other Variants.

The variant of Whitham's method due to Newell has already been mentioned as being one in which the author sought to extend Whitham's method to successfully cover higher order effects. Newell describes a variant of the Whitham method in his book [19] as well as giving some useful information as to the reasons why the original version of Whitham's method failed. Following remarks given by Ablowitz and Benney [32], he explicitly includes higher order terms in his expansion series for the functions within the lagrangian. Again this method is correct in its approach but does not have a formalism which is sufficiently systematic to be suited to its implementation as a computer algebra program.

The other variant of Whitham's method which is worthy of note is that due to Ablowitz and Benney [32, 33]. In Whitham's theory, the presence of a unique period is assumed in the definition of the averaged lagrangian. It will be remembered how the presence of this unique period was one of the conditions which were required to be satisfied before the method could be applied to a system of equations. If a unique period for the fast scale oscillations does not exist then it is impossible to find the averaged lagrangian which, by definition is

$$\overline{L} = \frac{1}{2\pi} \int_0^{2\pi} L d\theta \tag{3.48}$$

This integration represents an integration over a complete period of the oscillation of the underlying carrier wave, the fast scale oscillation.

In the method presented by Ablowitz and Benney [32], the Whitham method is extended to cover the case where several interacting nonlinear waves are present, each having a distinct period. The method also includes the higher order correction terms

which we have already pointed out are necessary in order to correctly model certain effects. The authors present results for the general case of multiple periodicities as well as the case where there are only two distinct periods. In a further paper [33], Ablowitz gives several examples in which the results produced by the single and multiphase theories are compared.

The method of Ablowitz and Benney is systematic in its approach and would be suited to implementation as a computer algebra program if this were to be required. The main difficulty with the method is that the complexity of the calculations again increases as a result of the multiple periodicities. The manipulations involved are similar to those carried out in Kawahara's variant of the method but not sufficiently similar that implementing the method as a computer algebra program would allow the use of code from the MAPLE computer algebra program which was finally written in order to carry out the desired calculations. The extension of the method to cover waves with multiple periodicities would be useful in modelling such phenomena as pulsed four-wave mixing. In addition, there would be an argument for implementing this method if it were desired to study interactions in media where the nonlinear interactions resulted in sufficient frequency doubling that there would be a significant component of light at a different frequency from the pump frequency. Whatever the merits of the method, the technique finally decided on was the variant by Kawahara which we will now discuss.

3.4. Kawahara's Variant of Whitham's Method.

We have seen how the original form of the Whitham method was flawed in that it did not handle the higher order terms correctly. This was known to Whitham and he pointed out [17] that modifications to the method would be required if the method were to be applied to problems in which these higher order terms were important. His student Small used such a modification in his thesis [18]. However, the method applied by Small is not systematic in its approach in the sense that it could be described as procedural. The most systematic variant of the Whitham method is that derived by Kawahara which incorporates within it the concept of a sequence of multiple scales by virtue of its incorporation of the formalism of the derivative expansion variant of the method of multiple scales. It is this method which has been implemented as a computer algebra program. We therefore continue by giving a description of the method.

3.4.1 Kawahara's Method.

For our given equation, we start by finding its lagrangian form. If the equation does not have a lagrangian form, Whitham's method will obviously be inapplicable. We assume that our equation has a lagrangian form which can be expressed as

$$L = L(x,t,f,f_x,f_{xx},f_{xxx},\ldots,f_t,f_{tt},f_{ttt},\ldots) \tag{3.49}$$

where there may be more than one dependent function f and we allow higher order derivatives to appear in the lagrangian. (Non-numeric subscripts appended to functions will represent partial differentiation with respect to the subscripted variable unless stated otherwise. Numeric subscripts will be used to indicate the order of expansion to which a function belongs. Usually it will be obvious from the context of the equation whether a subscript represents a differentiation or not.)

We start by defining the set of multiple scales

$$\begin{aligned} t &\rightarrow t_1,t_2,t_3,t_4, \quad \ldots \\ x &\rightarrow x_1,x_2,x_3,x_4, \quad \ldots \end{aligned} \tag{3.50}$$

and define

$$\begin{aligned} k(x_1,t_1,x_2,t_2,\ldots) &= \frac{\partial\theta}{\partial x} \\ \omega(x_1,t_1,x_2,t_2,\ldots) &= -\frac{\partial\theta}{\partial t} \end{aligned} \tag{3.51}$$

The dependent function (or functions) we redefine as

$$f(x,t) \rightarrow f(\theta,x_1,x_2,x_3,\ldots,t_1,t_2,t_3,\ldots) \tag{3.52}$$

We must also redefine the differential operators. We use the multiple scales

$$\begin{aligned} \frac{\partial}{\partial t} &\rightarrow \frac{\partial}{\partial t_1},\frac{\partial}{\partial t_2},\frac{\partial}{\partial t_3},\frac{\partial}{\partial t_4}, \quad \ldots \\ \frac{\partial}{\partial x} &\rightarrow \frac{\partial}{\partial x_1},\frac{\partial}{\partial x_2},\frac{\partial}{\partial x_3},\frac{\partial}{\partial x_4}, \quad \ldots \end{aligned} \tag{3.53}$$

and define the operators as

$$\frac{\partial}{\partial x} \to k\frac{\partial}{\partial \theta} + \varepsilon\frac{\partial}{\partial x_1} + \varepsilon^2\frac{\partial}{\partial x_2} + \varepsilon^3\frac{\partial}{\partial x_3} + \ldots$$
$$\frac{\partial}{\partial t} \to -\omega\frac{\partial}{\partial \theta} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2} + \varepsilon^3\frac{\partial}{\partial t_3} + \ldots \tag{3.54}$$

The equivalence of mixed derivatives therefore now gives us

$$\frac{\partial k}{\partial x_1} + \frac{\partial \omega}{\partial t_1} = 0, \qquad \frac{\partial k}{\partial x_2} + \frac{\partial \omega}{\partial k_2} = 0, \qquad \frac{\partial k}{\partial x_3} + \frac{\partial \omega}{\partial k_3} = 0, \qquad \ldots \tag{3.55}$$

We now substitute for all of the dependent variables with expansions of the form

$$f = \sum_{n=0}^{\infty} \varepsilon^{n+1} f_n, \; 0 < \varepsilon << 1 \tag{3.56}$$

where ε is a small parameter which is a measure of the difference between the scales of the carrier and of the envelope and where

$$f_0 = A\exp(i\theta) + A^*\exp(-i\theta) \tag{3.57}$$

and

$$f_n = \sum_{l=1}^{\infty} A_{l-1}^n \exp(il\theta) + A_{l-1}^{n\,*}\exp(-il\theta), \text{ for } n \neq 0 \tag{3.58}$$

It is possible to add a non-oscillating quantity to each f_n as well as to the lowest order function f_0. This is because, in some cases, the lagrangian is a potential representation of the equation which is being considered. We will dispense with this additional complication since such terms tend to appear at higher orders and can usually be set to zero anyway. For functions which deviate only slightly from the fundamental frequency we can also considerably simplify this last expression (3.58) since we can assume that no frequencies other than the fundamental are excited. This means that we can replace the expansion given above with

$$f_n = A_0^n \exp(i\theta) + A_0^{n\,*}\exp(-i\theta) \tag{3.59}$$

which will considerably simplify our analysis. It should perhaps be pointed out that if one carries out the calculations with the higher harmonic terms included, one tends to find that they appear only as corrections to the lower order expressions and it is only at higher orders in the small parameter that their presence is felt.

Inserting the expansions into the lagrangian and applying the new derivative operators gives us an expanded form of the lagrangian which we can write as

$$L = \sum_{n=0}^{\infty} \varepsilon^n L_n \tag{3.60}$$

For conventional types of lagrangians, the first two orders are often found to be zero. However, in subsequent sections, where we use quantized lagrangians, we will find it convenient to use a different expansion scheme in which it will be possible to have the first two orders of lagrangian non-zero.

The next step in the procedure is to integrate the lagrangian in order to perform the required averaging. We integrate over a period of the carrier with respect to θ.

$$\overline{L} = \frac{1}{2\pi} \int_0^{2\pi} L d\theta \tag{3.61}$$

This, by definition, is the averaged lagrangian which can also be written as

$$\overline{L} = \sum_{n=0}^{\infty} \varepsilon^n \overline{L}_n \tag{3.62}$$

What we have now is a series of lagrangians, one for each order of the small parameter, ε, to which we can now apply the Euler-Lagrange equation appropriate to the lagrangian functional as determined by the presence of variables and their derivatives in its functional form. This of course means that at each order of the small parameter, ε, we will have a series of equations of motion which will require to be satisfied if the equation is to correctly describe the behaviour to that order. We can then go on to solve these equations by some other means. As we have already seen, obtaining these equations is in itself an important aim since, even without solving them, their general form will suggest solution types or alternatively, if the equations are completely integrable, perhaps after a coordinate transformation, the problem will already be solved.

3.4.2. The Boussinesq Equation: an Example.

We shall now discuss the application of Kawahara's variant of the Whitham method to the Boussinesq equation, an equation taken from the field of water waves. The reason for choosing this equation is that a calculation for it has been performed by several different methods for comparison purposes. We have also derived the first order results using Whitham's original method. Further examples of the use of Whitham's method can be found in papers by Armstrong [36] and Dougherty [37]. Armstrong's paper discusses the use of Whitham's method in the field of nonlinear optics, applying the method to a system of two-level atoms to obtain results relating to the phenomenon of self-induced transparency. The paper by Dougherty mainly discusses the application of the method in the field of plasma physics but also touches on the use of the method in other areas.

We now give a corrected version of the solution of the Boussinesq equations presented in the book by Jeffrey and Kawahara [6]. The similarity of the method with the original one presented by Whitham can be gauged as can the complexity of the required manipulations. Our starting point is the lagrangian for the potential form of the Boussinesq equation presented in section 3.3.1.3 which we restate here as

$$\mathcal{L} = \tfrac{1}{2} f_t^2 - \tfrac{1}{2} c^2 f_x^2 + \tfrac{1}{2} \mu f_{xt}^2 - \tfrac{1}{6} f_x^3 \tag{3.63}$$

We then require to calculate the form of the differential operators $f_t^2, f_x^2, f_{xt}^2, f_x^3$ where the form of the first order operators is given by

$$\begin{aligned} \frac{\partial}{\partial x} &\to k\frac{\partial}{\partial \theta} + \varepsilon\frac{\partial}{\partial x_1} + \varepsilon^2\frac{\partial}{\partial x_2} + \varepsilon^3\frac{\partial}{\partial x_3} + \ldots \\ \frac{\partial}{\partial t} &\to -\omega\frac{\partial}{\partial \theta} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2} + \varepsilon^3\frac{\partial}{\partial t_3} + \ldots \end{aligned} \tag{3.64}$$

This definition was given previously as (3.54). For example f_x^2 is given by

$$f_x^2 = k^2\left(\frac{\partial f}{\partial \theta}\right)^2 + 2\varepsilon k\frac{\partial f}{\partial \theta}\frac{\partial f}{\partial x_1} + \varepsilon^2\left(\frac{\partial f}{\partial x_1}\right)^2 + 2\varepsilon^2 k\frac{\partial f}{\partial \theta}\frac{\partial f}{\partial x_2} + \ldots + O(\varepsilon^3) \tag{3.65}$$

The mixed derivative expression for f_{xt}^2 is quite complicated

$$
\begin{aligned}
f_{tt}^2 &= k^2\omega^2\left(\frac{\partial^2 f}{\partial\theta^2}\right)^2 - 2\varepsilon k^2\omega\frac{\partial^2 f}{\partial\theta^2}\frac{\partial^2 f}{\partial\theta\partial t_1} + 2\varepsilon k\omega^2\frac{\partial^2 f}{\partial\theta^2}\frac{\partial^2 f}{\partial\theta\partial x_1} \\
&-2\varepsilon^2 k^2\omega\frac{\partial^2 f}{\partial\theta^2}\frac{\partial^2 f}{\partial\theta\partial t_2} - 2\varepsilon^2 k\omega\frac{\partial^2 f}{\partial\theta^2}\frac{\partial^2 f}{\partial x_1\partial t_1} \\
&+2\varepsilon^2 k\omega^2\frac{\partial^2 f}{\partial\theta^2}\frac{\partial^2 f}{\partial\theta\partial t_2} + 2\varepsilon k\omega\frac{\partial^2 f}{\partial\theta^2}\frac{\partial\omega}{\partial x_1}\frac{\partial f}{\partial\theta} \\
&+2\varepsilon^2 k\omega\frac{\partial^2 f}{\partial\theta^2}\frac{\partial\omega}{\partial x_2}\frac{\partial f}{\partial\theta} - 2\varepsilon^2 k\omega\frac{\partial^2 f}{\partial\theta\partial t_1}\frac{\partial^2 f}{\partial\theta\partial x_1} \\
&-2\varepsilon^2 k\frac{\partial^2 f}{\partial\theta\partial t_1}\frac{\partial\omega}{\partial x_1}\frac{\partial f}{\partial\theta} + \varepsilon^2 k^2\left(\frac{\partial^2 f}{\partial\theta\partial t_1}\right)^2 \\
&+\varepsilon^2\omega^2\left(\frac{\partial^2 f}{\partial\theta\partial x_1}\right)^2 + 2\varepsilon^2\omega\frac{\partial^2 f}{\partial\theta\partial x_1}\frac{\partial\omega}{\partial x_1}\frac{\partial f}{\partial\theta} + \varepsilon^2\left(\frac{\partial\omega}{\partial x_1}\right)^2\left(\frac{\partial f}{\partial\theta}\right)^2 + \ldots + O(\varepsilon^3)
\end{aligned}
\tag{3.66}
$$

Similar types of expressions can be found for the other two derivative forms. We see that to first order they correspond with those found earlier using Whitham's original method (3.37). It now remains to make the substitutions for the function f. If we refer to the equation given previously as (3.56)

$$
f = \sum_{n=0}^{\infty}\varepsilon^{n+1} f_n,\ 0 < \varepsilon << 1
\tag{3.67}
$$

we see that the definition of the function f has been altered from that given in the book. If one refers to Kawahara's original paper [27] one finds that the form of the lagrangian for the potential form of the Boussinesq equation solved, and from which the results are transcribed into the book [6], is

$$
\mathcal{L} = \tfrac{1}{2} f_t^2 - \tfrac{1}{2} c^2 f_x^2 + \tfrac{1}{2}\mu f_{tt}^2 - \tfrac{1}{6}\varepsilon f_x^3
\tag{3.68}
$$

The difference lies in the introduction of the scaling parameter ε explicitly in the final term. Alternatively, we can adjust the scaling by changing the definition of the function f describing the wave motion but retaining the original lagrangian. This is why the substitution for the series starts at ε^1 in the expression (3.67).

If we use the substitution

$$
f_0 = \varepsilon\left(Ae^{i\theta} + A^* e^{-i\theta} + B\right)
$$

(3.69)

together with the definitions of the operators we have just calculated, we can obtain the lowest order result for the lagrangian as

$$\begin{aligned}\mathcal{L}_0 = {} & \tfrac{1}{2}\omega^2\left\{-A^2e^{2i\theta} - A^{*2}e^{-2i\theta} + 2AA^*\right\} \\ & -\tfrac{1}{2}c^2k^2\left\{-A^2e^{2i\theta} - A^{*2}e^{-2i\theta} + 2AA^*\right\} \\ & +\tfrac{1}{2}\mu\omega^2k^2\left\{A^2e^{2i\theta} + A^{*2}e^{-2i\theta} + 2AA^*\right\}\end{aligned}$$

(3.70)

so that after averaging one obtains the expression

$$\begin{aligned}\overline{\mathcal{L}}_0 &= \left(\omega^2 - c^2k^2 + \mu\omega^2k^2\right)AA^* \\ &= D(\omega,k)|A|^2\end{aligned}$$

(3.71)

so that the first order result - the equation of motion for A^* - is simply the linear dispersion relation as before.

If one goes on to substitute for the whole expression for f, one obtains the very complicated expressions for the results given by Kawahara in his paper [27] or in the book [6]. We present them for the purposes of comparison with subsequent results. The averaged lagrangians are

$$\overline{\mathcal{L}}_0 \quad = \left(\omega^2 - c^2k^2 + \mu\omega^2k^2\right)AA^* = D(\omega,k)|A|^2$$

(3.72)

for the second order result which we have called $\overline{\mathcal{L}}_0$ since it is the lowest order lagrangian obtained from the analysis with the following expression for $\overline{\mathcal{L}}_1$

$$\overline{\mathcal{L}}_1 \quad = D\left(AA_1^{(1)^*} + A^*A_1^{(1)}\right) - \frac{i}{2}\frac{\partial D}{\partial\omega}\left(A\frac{\partial A^*}{\partial x_1} - A^*\frac{\partial A}{\partial x_1}\right)$$

(3.73)

and

$$
\begin{aligned}
\bar{L}_2 \quad &= D\left(AA_2^{(1)^*} + A^*A_2^{(1)}\right) \\
&-\frac{i}{2}\frac{\partial D}{\partial \omega}\left(A\frac{\partial A^*}{\partial t_2} - A^*\frac{\partial A}{\partial t_2} + A\frac{\partial A_1^{(1)^*}}{\partial t_1} - A^*\frac{\partial A_1^{(1)}}{\partial t_1} + A_1^{(1)}\frac{\partial A^*}{\partial t_1} - A_1^{(1)^*}\frac{\partial A}{\partial t_1}\right) \\
&+\frac{i}{2}\frac{\partial D}{\partial k}\left(A\frac{\partial A^*}{\partial x_2} - A^*\frac{\partial A}{\partial x_2} + A\frac{\partial A_1^{(1)^*}}{\partial x_1} - A^*\frac{\partial A_1^{(1)}}{\partial x_1} + A_1^{(1)}\frac{\partial A^*}{\partial x_1} - A_1^{(1)^*}\frac{\partial A}{\partial x_1}\right) \\
&+\frac{1}{2}\frac{\partial^2 D}{\partial \omega^2}\left(\frac{\partial A}{\partial t_1}\frac{\partial A^*}{\partial t_1}\right) + \frac{1}{2}\frac{\partial^2 D}{\partial k^2}\left(\frac{\partial A}{\partial x_1}\frac{\partial A^*}{\partial x_1}\right) + \mu\left(\frac{\partial k}{\partial t_1}\right)^2 |A|^2 \\
&+\frac{1}{4}\frac{\partial^2 D}{\partial \omega \partial k}\left(A\frac{\partial^2 A^*}{\partial t_1 \partial x_1} + A^*\frac{\partial^2 A}{\partial t_1 \partial x_1} - \frac{\partial A}{\partial t_1}\frac{\partial A^*}{\partial x_1} - \frac{\partial A^*}{\partial t_1}\frac{\partial A}{\partial x_1}\right) \\
&+\frac{1}{4}\frac{\partial}{\partial t_1}\left(\frac{\partial^2 D}{\partial \omega^2}\right)\left[A^*\frac{\partial A}{\partial t_1} + A\frac{\partial A^*}{\partial t_1}\right] + \frac{1}{4}\frac{\partial}{\partial x_1}\left(\frac{\partial^2 D}{\partial k^2}\right)\left[A^*\frac{\partial A}{\partial x_1} + A\frac{\partial A^*}{\partial x_1}\right] \\
&+\sum_{l=1}^{\infty} D(l\omega, lk)A_1^{(l)}A_1^{(l)^*} + \frac{1}{2}\left(\frac{\partial B}{\partial t_1}\right)^2 - \frac{1}{2}c^2\left(\frac{\partial \mathbf{B}}{\partial x_1}\right)^2 - ik^3\left(A^2A_1^{(2)^*} - A^{*2}A_1^{(2)}\right) \\
&-k^2AA^*\frac{\partial B}{\partial x_1}
\end{aligned}
\tag{3.74}
$$

for $\bar{L}_2$.where we have used obvious extensions to the notation for the dispersion relation.

This is the result given by Kawahara in the book by Jeffrey and Kawahara [6]. Although we have given only a flavour of the derivation, the algebraic complexity of the manipulations required to produce equation (3.74) is obvious: even allowing for the truncation of results involving high powers of the expansion parameter, several hundred terms have had to be considered.

It can be shown that the equations of motion for the lagrangian $\bar{L}_2$ can be reduced to a form of the nonlinear Schrödinger equation [6]. This therefore implies that the variant of Whitham's method presented by Kawahara is both general and capable of retaining sufficient information to correctly model the modulational effects which the original method could not. We have already pointed out that one of the failures of the original method, of which Whitham was aware, was the failure of the method to correctly account for these higher order terms. The Kawahara variant is therefore suitable for modelling problems in nonlinear optics.

The problem with this method lies in the large numbers of terms which require to be processed.The complexity of the manipulations can be reduced somewhat by

considering truncated expansion series and by being careful to discard additional terms at the earliest opportunity. Nevertheless, if one wishes to use this method to solve some problem, then that problem should be important enough to warrant the time required to perform the lengthy and tedious calculations required. Its use for everyday problems is prohibited by the complexity of the analysis required in order to obtain the forms of the averaged lagrangians. Once one has obtained them, yet further work requires to be done in order to attempt their solution.

It was in order to simplify the process of obtaining averaged lagrangians for nonlinear optics problems that the programs to be presented in the next two chapters were subsequently written. The above results give some idea of the complexity of the calculations required to be carried out if the solutions were to be obtained by hand. If results to higher orders were required, then the method simply becomes impractical. This limitation is not present to the same extent when running such problems as computer programs.

In the next chapter, we will go on to introduce lagrangians for use in nonlinear optics. However, in the next section, it is necessary that we consider a different but related method due to Bondeson, Anderson and Lisak.

3.5 The Variational Method of Bondeson, Anderson and Lisak.

We have so far discussed Whitham's method and several of its variants. There remains one important method due to Bondeson, Anderson and Lisak which it is necessary to discuss. The method is related to Whitham's technique and is used to study the variation of pulses when their describing parameters are perturbed. The variant comes in two slightly different varieties. In the first, a perturbed form of some exactly integrable system is investigated by using the exact solitonic solution of the system as a trial solution for the perturbed system. The soliton parameters are optimised and the resulting solutions compare well with those obtained using perturbed inverse scattering transform techniques. In the second of the two cases, the waveform used as a trial function is of gaussian form. Although a distinction has been made between these two methods, they are almost completely identical. In both methods a variational technique is used to model variations in solution parameters.

We have discussed in chapter 2 how the nonlinear optical systems can often be shown to give rise to nonlinear describing equations which may have solitonic solutions. The nonlinear Schrödinger equation (with additional terms) is the equation which is most commonly used in the description of light pulses in optical fibres; the sine-Gordon equation is commonly used in describing resonant phenomena. We have also explained how finding a systematic method of deriving the additional terms which are often added to the nonlinear Schrödinger equation forms a major motivation for this research. The variational method of Bondeson, Anderson and Lisak [38] can be used to examine the behaviour of solitonic pulses and thus in investigating the behaviour of pulses described by equations which are inherently nonlinear.

The authors claim that the method "may be considered as an extension of Whitham's method to the case of solitary waves" and that it can be shown to give "a concise derivation of the evolution equations for the parameters" describing the soliton propagation. If the solitonic solution of the unperturbed system is known then, given the lagrangian for the equation, one may introduce certain terms as perturbations of the original equation and use the method to study the effect the perturbation has on the propagation of the original waveform. The class of perturbations which is allowed is fairly broad and includes dissipative and dispersive terms. However, the assumption made that any perturbations are adiabatic means that studies cannot be made of processes which occur over short time scales such as soliton collisions. This is similar to the limitation in Whitham's method in that only information concerning the envelope of the waveform is available. The assumption that the perturbations are adiabatic leads to equations which are similar to those of Ostrovsky and Pelinovsky [29]. The concept of averaging is used in that the lagrangian form of the equations is integrated over the time variable.

Rather than carry out a variational analysis of the full lagrangian, a further step in the process is to consider only variations in the soliton parameters. The variational principle must therefore be recast in a manner in which the variables to be optimised form the parameters of the solitonic solution. In other words, the soliton solution is being used as a trial function with which to test the response of the perturbed system.

In the first paper which considers solitonic equations [38], the authors consider perturbed forms of several well-known solitonic systems including the nonlinear Schrödinger equation and the sine-Gordon equation. Their results agree well with results obtained using forms of inverse scattering theory which allow for perturbations. In a second paper [39] in which the variational method is applied with solitonic

solutions, the authors study the interaction forces between solitons. They investigate the problem by using two coupled nonlinear Schrödinger equations and do not use the two-soliton form of the exact solution. They are able to derive equations describing the separation of the two solitons which are similar to those which may be obtained using perturbed inverse scattering theory. Their analysis is, however, much simpler than that involved in carrying out the calculation using this latter method.

In a previous paper [40], the authors performed the analysis limiting the interaction between the carrier and the modulating waveform to the interaction term. They find that the deleterious effects caused by loss in the case of coherent pulse propagation are lessened thus improving the estimates for the performance of proposed soliton communication systems.

Finally, in a paper which partly summarises previous results [41], the authors discuss the importance of the decaying non-soliton part of the pulse in again degrading the performance of communications systems. The merging of the two radiative backgrounds will cause a high background pedestal which will be difficult to separate from the signal pulses.

The second variant of the technique was first introduced in the field of nonlinear optics in a paper by Anderson [42]. Strictly, this is not a new method: rather it is an application of an old one in a new area. The idea behind the technique is that, for a lagrangian describing the test system, a pulse may again be introduced in the form of a test function which has variable parameters. The optimisation may then be carried out with respect to the parameters of the test function rather than by attempting to solve the equations of motion for the lagrangian. This test function, however, does not necessarily represent a solution of the describing system but rather is the generic gaussian pulse shape.

This of course assumes that a lagrangian form for the system to be described exists. In a series of papers, Anderson and Lisak investigate lagrangians describing the evolution of light pulses in optical fibres, again described by variants of the nonlinear Schrödinger equation. They then investigate the behaviour of gaussian shaped pulses by optimising the pulse parameters. Whilst this is a useful way of investigating how different shaped pulses are affected during propagation, it does not help in determining what the ideal form of the pulses would be although it does give insights into qualitative behaviour. Since it is necessary to choose a test function in order to represent the pulse, behaviour in which the pulse changes shape cannot be modelled by this method: the

pulse must retain gaussian form throughout. Nevertheless, useful information may be obtained concerning the dynamic behaviour of the pulse as it propagates along the fibre.

The first paper in the field of nonlinear optics which uses the gaussian function as a trial function within the variational procedure is by Anderson [42]. Their point is that, although the solitonic solutions are exact, they represent only specialised solutions: they are not general solutions. Thus, to investigate pulse propagation, one may use some other alternative pulse shape to investigate the pulse evolution. The use of a gaussian pulse as a test function allows the approximate solution of the nonlinear Schrödinger equation for cases which do not satisfy the requirements for soliton propagation. The main problem with this approach is, as has been mentioned previously, the fact that the pulse shape is then constrained to have the same general form.

Using their method, they derive an interesting mechanical analogy using a potential function which describes the behaviour of a gaussian pulse. By varying the relative strengths of the dispersive and nonlinear terms they are able to use this analogy to predict that for certain values the nonlinearity will be unable to balance the dispersion and a pulse will broaden; for higher values of the nonlinearity, the breadth of the pulse will oscillate; and at a certain limit value the pulse shape will propagate unchanged. Their results compare well with results obtained using soliton theory in the cases where the initial shape of the gaussian pulse approximates well to that of a soliton.

In a later paper [43], Anderson and Lisak discuss the effect of loss on "soliton" propagation, again by using this variational approach with a gaussian trial function. They use analysis similar to that used in the previous paper cited except that they include a loss term. The main advantage of the analysis is that there is no requirement that the loss term need be small: the analysis is not perturbative in type. They find that the inclusion of loss at first causes the period of oscillation of the pulse breadth to increase until, at a critical distance, the pulse breaks up and spreads out dispersively. Their results agree with numerical simulations as well as with an approximate (W.K.B.) analytic solution.

In a final paper by Anderson and co-authors [44], the effect of linear frequency chirp combined with fibre nonlinearity on pulses is investigated. They again use a variational approach with a gaussian trial function modified to allow for the frequency chirp and seek to show that the combined result of the two effects is to produce pulse compression.

An additional paper by Kumar and coworkers [45] uses the variational technique, again with a gaussian trial function, to investigate the effect of higher order nonlinearity on the pulse shape. Essentially the analysis is the same as that carried out in the Anderson papers. The authors find that the pulse breakup predicted by models based on the nonlinear Schrödinger equation including a loss term is delayed by the presence of a fifth order nonlinear term as might be expected.

Whilst the papers by Anderson and coworkers consider the problem of wave propagation in fibres by modelling the nonlinear Schrödinger equation in lagrangian form, a paper by Caglioti, Crosignani and Di Porto [46] considers the problem in hamiltonian form. Again the assumption is made that the pulse shape does not alter its shape during the evolution. The authors state that little new information can be gained from the model but that the simplicity of the computations involved lends the method to numeric calculations and provides additional physical insight.

References

1. Kodama, Y. and A. Hasegawa, *Nonlinear Pulse Propagating in a Monomode Dielectric Guide*. I.E.E.E. J. Quant. Electron., 1987. **QE-23**(5): p. 510-524.

2. Sneddon, I.N., *Elements of Partial Differential Equations*. International Student Edition ed. International Series in Pure and Applied Mathematics, ed. W.T. Martin. 1964, McGraw-Hill Book Company. 327 pp.

3. Bluman, G.W. and J.D. Cole, *Similarity Methods for Differential Equations*. Applied Mathematics and Science, 1974, New York: Springer-Verlag. 332 pp.

4. Olver, P.J., *Applications of Lie Groups to Differential Equations*. Graduate Texts in Mathematics, ed. P.R. Halmos, F.W. Gehring, and C.C. Moore. 1986, Springer-Verlag. 497 pp.

5. Schwarz, F., *SPDE*. 1985, Rand Corporation: St. Augustin.

6. Jeffrey, A. and T. Kawahara, *Asymptotic Methods in Nonlinear Wave Theory.* 1 ed. Applicable Mathematics Series, Pitman Advanced Publishing Program, 1982, Boston, London, Melbourne: Pitman Books Limited. 256 pp.

7. Nayfeh, A.H., *Perturbation Methods.* 1973, New York: John Wiley & Sons.

8. Whitham, G.B., *Non-linear Dispersive Waves.* Proc. Roy. Soc. Lond. A, 1965. **283**: p. 283-261.

9. Whitham, G.B., *A General Approach to Linear and Non-linear Dispersive Waves Using a Lagrangian.* J. Fluid Mech., 1965. **22**(2): p. 273-283.

10. Sturrock, P.A., *A Variational Principle and an Energy Theorem for Small Amplitude Disturbances of Electron Beams and of Electron-ion Plasmas.* Ann. Phys., 1958. **4**: p. 306-324.

11. Whitham, G.B., *Nonlinear Dispersion of Water Waves.* J. Fluid Mech., 1966. **27**(2): p. 399-412.

12. Luke, J.C., *A Perturbation Method for Nonlinear Dispersive Waves.* Proc. Roy. Soc. A, 1966. **292**: p. 403-412.

13. Lighthill, M.J., *Contributions to the Theory of Non-linear Dispersive Systems.* J. Inst. Maths. Applics., 1965. **1**: p. 269-306.

14. Bretherton, F.P., *Propagation in Slowly Varying Waveguides.* Proc. Roy. Soc. A., 1968. **302**: p. 555-576.

15. Whitham, G.B., *Variational Methods and Applications to Water Waves.* Proc. Roy. Soc. Lond. A, 1967. **299**: p. 6-25.

16. Whitham, G.B., *Two-timing, Variational Principles and Waves.* J. Fluid Mech., 1970. **44**(2): p. 373-395.

17. Whitham, G.B., *Linear and Nonlinear Waves*. 1 ed. Pure and Applied Mathematics, 1974, New York, Chichester, Brisbane, Toronto, Singapore: John Wiley & Sons. 636 pp.

18. Small, R.D., *Nonlinear Dispersive Waves in Nonlinear Optics*. 1972, California Institute of Technology:

19. Newell, A.C., *Solitons in Mathematics and Physics*. CBMS-NSF Regional Conference Series in Applied Mathematics, ed. C.B.o.t.M. Sciences. Vol. 48. 1985, Society for Industrial and Applied Mathematics. 244 pp.

20. Seliger, R.L. and G.B. Whitham, *Variational Principles in Continuum Mechanics*. Proc. Roy. Soc. A, 1968. **305**: p. 1-25.

21. Elsgolc, L.E., *Calculus of Variations*. International Series of Monographs on Pure and Applied Mathematics, ed. I.N. Sneddon, M. Stark, and S. Ulam. 1961, Oxford, London, New York, Paris: Pergamon Press. 178 pp.

22. Jack, P.M., *Ph. D. Thesis*. 1978, University of Manchester:

23. Dewar, R.L., *A Lagrangian Theory for Nonlinear Wave Packets in a Collisionless Plasma*. J. Plasma Phys., 1972. **7**(2): p. 267-284.

24. Dysthe, K.B., *A Note on the Application of Whitham's Method to Nonlinear Waves in Dispersive Media*. J. Plasma Phys., 1974. **11**(1): p. 63-76.

25. Chu, V.H. and C.C. Mei, *On Slowly-varying Stokes Waves*. J. Fluid Mech., 1970. **41**(4): p. 873-887.

26. Kamchatnov, A.M., *Propagation of Long Light Pulses in a Two-level Resonant Medium*. Sov. Phys. JETP, 1988. **67**(10): p. 2047-2052.

27. Kawahara, T., *A Note on the Lagrangian Method for Nonlinear Dispersive Waves*. J. Plasma Phys., 1977. **18**(2): p. 305-316.

28. Ostrovsky, L.A. and E.N. Pelinovsky, *Averaging Method for Nonsinusoidal Waves*. Sov. Phys. Doklady, 1971. **15**(12): p. 1097-1099.

29. Ostrovsky, L.A. and E.N. Pelinovsky, *Method of Averaging and the Generalised Variational Principle for Nonsinusoidal Waves*. J. Appl. Math. Mech., 1972. **36**(1): p. 71-78.

30. Lavenda, B.H., *Principles and Representations of Nonlinear Thermodynamics*. Phys. Rev. A, 1974. **9**(2): p. 929-942.

31. Lavenda, B.H., G. Peluso, and A. Di Chiara, *Non-linear Dissipative and Dispersive Waves*. J. Inst. Maths. Applics., 1977. **20**: p. 117-131.

32. Ablowitz, M.J. and D.J. Benney, *The Evolution of Multi-phase Modes for Nonlinear Dispersive Waves*. Stud. Appl. Math., 1970. **49**(3): p. 225-238.

33. Ablowitz, M.J., *Applications of Slowly Varying Nonlinear Dispersive Wave Theories*. Stud. Appl. Math., 1971. **50**(4): p. 329-344.

34. Van Der Vaart, H.R., *Variational Principle for Certain Nonconservative Systems*. Am. J. Phys., 1967. **35**(5): p. 419-423.

35. Gorschkov, K.A., L.A. Ostrovsky, and E.N. Pelinovsky, *Some Problems of Asymptotic Theory of Nonlinear Waves*. Proc. I.E.E.E., 1974. **62**(11): p. 1511-1517.

36. Armstrong, J.A., *Averaged Lagrangian Method Applied to Resonant Nonlinear Optics. The Self-steepening of Light Pulses*. Phys. Rev. A, 1975. **11**(3): p. 963-972.

37. Dougherty, J.P., *Lagrangian Methods in Plasma Dynamics. I. General Theory of the Method of the Averaged Lagrangian*. J. Plasma Phys., 1970. **4**(4): p. 761-785.

38. Bondeson, A., M. Lisak, and D. Anderson, *Soliton Perturbations: A Variational Principle for the Soliton Parameters*. Phys. Scripta, 1979. **20**: p. 479-485.

39. Anderson, D. and M. Lisak, *Bandwidth Limits due to Mutual Pulse Interaction in Optical Soliton Communication Systems*. Opt. Lett., 1986. **11**(3): p. 174-176.

40. Anderson, D. and M. Lisak, *Bandwidth Limits due to Incoherent Soliton Interaction in Optical-fiber Communication Systems*. Phys. Rev. A., 1985. **32**(4): p. 2270-2274.

41. Anderson, D., M. Lisak, and T. Reichel, *Asymptotic Propagation Properties of Pulses in a Soliton-based Optical-fiber Communication System*. J. Opt. Soc. Am. B, 1988. **5**(2): p. 207-210.

42. Anderson, D., *Variational Approach to Nonlinear Pulse Propagation in Optical Fibres*. Phys. Rev. A., 1983. **27**(6): p. 3135-3145.

43. Anderson, D., *High Transmission Rate Communication Systems Using Lossy Optical Solitons*. Opt. Comm., 1983. **48**(2): p. 107-112.

44. Anderson, D., M. Lisak, and M. Anderson, *Nonlinearly Enhanced Chirp Pulse Compression in Single-mode Fibers*. Opt. Lett., 1985. **10**(3): p. 134-136.

45. Kumar, A., S.N. Sarkar, and A.K. Ghatak, *Effect of Fifth-order Nonlinearity in Refractive Index on Gaussian Pulse Propagation in Lossy Fibers*. Opt. Lett., 1986. **11**(5): p. 321-323.

46. Caglioti, E., B. Crosignani, and P. Di Porto, *Hamiltonian Description of Nonlinear Propagation in Optical Fibers*. Phys. Rev. A, 1988. **38**(8): p. 4036-4042.

Chapter 4

A belief is not true because it is useful.

Henri-Frédéric Amiel, 1828-1881.

4.1. Introduction.

In this chapter, we will discuss the derivation of the lagrangians to be used in determining the behaviour of nonlinear optical systems. We will consider both the classical model closely allied to the Lorentz model and semiclassical models for multi-level systems related to the Standard Lagrangian. We will discuss other ancillary matters such as the choice of gauge and the use of the density matrix formalism in posing the problem. We will then go on to discuss a first attempt at the solution of problems in lagrangian form by means of the Kawahara variant of Whitham's averaged lagrangian method as implemented as a REDUCE computer algebra program.

4.2. Lagrangians for Nonlinear Optics.

In this section we will discuss the derivation of the lagrangians to be investigated using the Kawahara variant of Whitham's method. These lagrangians will be used in Chapter 6 as the input to the MAPLE program written to implement the averaging technique and can be thought of as sample problems to be investigated using the program as a tool. We will touch upon some other related matters in their derivation but will assume knowledge of the material presented in previous chapters in justification of why one would wish to investigate such lagrangians.

4.2.1. Introduction.

We shall investigate two sets of lagrangians. Firstly we shall derive a classical lagrangian in which the fields are described in purely classical terms and the nonlinearity is provided by using a nonlinear restoring force on a field of pseudoelectrons associated with a field of neutral atoms. Secondly we will present a series of semiclassical lagrangians in which the field of atoms will be assumed to be quantized, having a number of preferred quantum states between which electrons may make transitions under the influence of an external electric field, the nonlinearity being introduced by the coupling between the field of quantized atoms and a classical electromagnetic field. The classical lagrangian is related to the Lorentz model of optics in which matter is assumed to be represented by a field of massive atoms each associated with an electron bound by a restoring force. Secondly we will derive a semiclassical model in which the function

describing the atoms may be modified to allow for a multiplicity of atomic levels, two or three say, thus allowing us to study occupation levels within the atomic field. This work is allied to the Maxwell-Bloch type equations described in Chapter 2.

4.2.2. Classical Lagrangians.

The idea that the averaged lagrangian technique could be used to shed light on equations of nonlinear optics is not new. Whitham himself considers one of the examples we use in his book [1], and his student R. D. Small covers the derivation of the lagrangian used in detail in his thesis [2]. On the other hand, the consideration of the equations carried out by both researchers is in terms either of Whitham's original method (Whitham) or of an adaptation of Whitham's method not seen elsewhere in the literature (Small). As we have already pointed out, Whitham was aware that his original method would not correctly describe higher order results and required to be modified in some manner in order to do so. Small makes a suitable modification in terms of explicitly introducing two-timing into his consideration of the higher order results of the equations so that he anticipates Kawahara's variant somewhat.

The lagrangian which they both consider models a classical electromagnetic field coupled to a field of neutral atoms. As such it simply represents the natural generalisation of the Lorentz linear classical model describing the same situation. The difference lies only in that the restoring force acting on the electrons is linear in Lorentz' model.

4.2.2.1. The Lorentz Model.

The idea of the Lorentz model is quite simple. One considers the medium to be an infinite field of pseudoatoms, each of which is sufficiently separated from the other that they may be considered to be independent entities. Each atom is given one electron which is bound to the nucleus by a linear restoring force. Thus the electron is modelled by having it behave as a simple harmonic oscillator. Each electron as it moves will create an electromagnetic field so that when an external electromagnetic field is applied as a driving force, the field produced by the oscillating electrons must be made consistent with that applied by the external field. That is, in driving the motion of the electrons, the field is itself modified by the fields created by the movements of the electrons which again modifies the manner in which the electrons oscillate and so on. Consistency may

be maintained by appealing to energy conservation whereby the equation describing the motion of the electrons is modified from that of a driven harmonic oscillator to that of a damped driven harmonic oscillator.

The equations are given in the book by Allen and Eberly [3] as

$$\frac{\partial^2 x_a}{\partial t^2} + \omega_a^2 x_a = \frac{e}{m} E(t, \mathbf{r}_a) \tag{4.1}$$

and

$$\frac{\partial^2 x_a}{\partial t^2} + \frac{2}{\tau_0}\frac{\partial x_a}{\partial t} + \omega_a^2 x_a = \frac{e}{m} E(t, \mathbf{r}_a) \tag{4.2}$$

We refer the reader to this book for their derivation and for the units used. What is important is the form of the equations. We will compare them with the form of the equations used by Whitham in his derivation of a classical lagrangian to describe the same situation.

4.2.2.2. The Three Dimensional Lagrangian.

The Lorentz model is linear in that the restoring force applied to the electrons is linear. In order to modify the model to describe nonlinearity, it is necessary to derive equations which allow the restoring force on the electrons to be nonlinear. Rather than the somewhat simplified approach given in the derivation of the Lorentz equations, the equations to describe the behaviour of the electrons under a nonlinear restoring force can be derived directly from Maxwell's equations by including source terms to describe the effects of the nonlinearity. By using the results of the analysis, we can then derive a lagrangian which will describe the classical behaviour of the electrons. Once we have applied a solution method to the lagrangian, we can then compare the classical description with the quantized one. The lagrangian derived is given by R. D. Small in his thesis [2] and we will follow his derivation closely.

Start with Maxwell's equations in S.I. units (as given, for example, in Lorrain and Corson [4] page 452), each symbol being given its usual meaning.

$$\begin{aligned} \nabla \times \mathbf{B} - \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} &= \mu_0 \mathbf{J}_m \\ \nabla . \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} + \frac{\partial \mathbf{B}}{\partial t} &= 0 \\ \nabla . \mathbf{E} &= \frac{\rho_t}{\varepsilon_0} \end{aligned}$$

(4.3a, b, c, d)

Replacing **B** using $\mathbf{B} = \mu \mathbf{H}$ and using the identity $\nabla \times (\mu \mathbf{H}) = \mu \nabla \times \mathbf{H} - \mathbf{H} \times \nabla \mu$ we obtain the equation

$$\nabla \times \mathbf{H} = \mathbf{J}_m + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}$$

(4.4)

from (4.3a) where we also assume that $\mu_0 \varepsilon_0 = 1/c^2$ and $\nabla \mu = 0$. In other words we have assumed that

$$\begin{aligned} \mathbf{B} &= \mu \mathbf{H} \\ \mathbf{D} &= \varepsilon \mathbf{E} \\ \mu &= \mu_0 \\ \varepsilon &= \varepsilon_0 \end{aligned}$$

(4.5a, b, c, d)

We shall return to these assumptions shortly. Again making the substitution $\mathbf{B} = \mu \mathbf{H}$ in equation (4.3b) and (4.3c) we obtain

$$\nabla . \mathbf{H} = 0$$

(4.6)

and

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t}$$

(4.7)

Equation (4.3d) remains unaltered as

$$\nabla . \mathbf{E} = \frac{\rho_t}{\varepsilon_0}$$

(4.8)

If we now return to the assumptions made in equations (4.5), we see that the assumptions we have made are those for a linear, homogeneous, isotropic medium. The nonlinearity is introduced by means of the source terms.

The assumptions now made by Small are that the dipole moment at each lattice site is to be considered to be that for a finite sized dipole with the positive particle fixed at the lattice point and the negatively charged particle held in a potential well with the centre of the well located at the positive particle. He further considers only the force acting on the negative particle to be that effected by the electric field. He neglects magnetic effects, the distortion of the lattice and dissipation. Since we wish to form a lagrangian model which will require the use of a conservative field, the neglect of dissipation is consistent with the model we are constructing. Of the other two assumptions, the first effect is small since the velocity of the electron will be small (if it is to remain bound to its nucleus) and the distortion of the lattice sites may be assumed to be small since the crystal being considered is only a model.

Small then takes the potential at a displacement **R** from the nucleus as being given by the function **U(R)**. If this potential function is taken as being quadratic, we obtain the Lorentz theory again as we will see. If the potential function is higher than quadratic, the force will be nonlinear as we require. Small therefore takes

$$m\frac{\partial^2 R_i}{\partial t^2}+\frac{\partial U}{\partial R_i}=-eE_i \tag{4.9}$$

for each component of the the vectors and where m is the effective mass of the electron and -e its charge. If we then define

$$\mathbf{P}=-Ne\mathbf{R} \tag{4.10}$$

as the polarisation then we may rewrite (4.9) as

$$\frac{\partial^2 P_i}{\partial t^2}-\frac{Ne}{m}\frac{\partial U}{\partial R_i}=\varepsilon_0\omega_p^2 E_i \tag{4.11}$$

for each component *i* where

$$\omega_p^2=\frac{Ne^2}{\varepsilon_0 m} \tag{4.12}$$

ω_p is called the plasma frequency. If we then define

$$V(\mathbf{P}) = \frac{N^2 e^2}{\varepsilon_0 m} U(\mathbf{R}) \tag{4.13}$$

then we can derive

$$\frac{\partial V}{\partial P_i} = -\frac{Ne}{m}\frac{\partial U(R_i)}{\partial R_i} \tag{4.14}$$

and so we can rewrite the equation (4.11) as

$$\frac{\partial^2 P_i}{\partial t^2} + \frac{\partial V}{\partial P_i} = \varepsilon_0 \omega_p^2 E_i \tag{4.15}$$

We now need to relate the sources **J** and ρ to the polarisation P so that we can make use of (4.15) within Maxwell's equations. This is done by defining the current density as being due to the motion of the negative particles and so

$$\mathbf{J} = -Ne\frac{\partial \mathbf{R}}{\partial t} \tag{4.16}$$

That is

$$\mathbf{J} = \frac{\partial \mathbf{P}}{\partial t} \tag{4.17}$$

Now since

$$\nabla.\mathbf{J} = -\frac{\partial \rho}{\partial t} \tag{4.18}$$

and so therefore

$$\nabla.\mathbf{J} = \partial/\partial t(\nabla.\mathbf{P}) \tag{4.19}$$

integrating gives

$$\nabla.\mathbf{P} = -\rho \tag{4.20}$$

Furthermore, since

$$\nabla.\mathbf{E} = \frac{\rho}{\varepsilon_0} \tag{4.21}$$

we have

$$\varepsilon_0 \nabla.\mathbf{E} = -\nabla.\mathbf{P} \tag{4.22}$$

If we now substitute the expression for **J** into Maxwell's equations (4.3) we obtain

$$\begin{aligned} \nabla \times \mathbf{H} &= \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \frac{\partial \mathbf{P}}{\partial t} \\ \nabla.\mathbf{H} &= 0 \\ \nabla \times \mathbf{E} &= -\mu_0 \frac{\partial \mathbf{H}}{\partial t} \\ \varepsilon_0 \nabla.\mathbf{E} &= -\nabla.\mathbf{P} \end{aligned} \tag{4.23a, b, c, d}$$

and it is this set which we will now use to obtain the classical form of the averaged lagrangian to which we will subsequently apply the Whitham averaged lagrangian technique.

In the paper by Seliger and Whitham [5], the use of potential representations as a way towards determining the lagrangian description of a problem was discussed. We should not be surprised, therefore, when the potentials for the electromagnetic field (vector potential **A** and scalar potential ϕ) are now introduced. Small uses the standard equations

$$\mu_0 \mathbf{H} = \nabla \times \mathbf{A} \tag{4.24}$$

and

$$\mathbf{E} = -\frac{\partial \mathbf{A}}{\partial t} - \nabla \phi \tag{4.25}$$

and of course this choice means that equations (4.23b) and (4.23c) are satisfied identically.

If we substitute the expressions for **E** and $\mu_0 \mathbf{H}$ into the equations (4.23a) and (4.23d) we obtain the equations

$$\nabla\times\nabla\times\mathbf{A}=-\mu_0\varepsilon_0\frac{\partial^2\mathbf{A}}{\partial t^2}-\mu_0\varepsilon_0\frac{\partial}{\partial t}(\nabla\phi)+\mu_0\frac{\partial\mathbf{P}}{\partial t} \tag{4.26}$$

and

$$\frac{\partial}{\partial t}(\nabla.\mathbf{A})+\nabla^2\phi=\frac{1}{\varepsilon_0}\nabla.\mathbf{P} \tag{4.27}$$

respectively. It is at this point that the choice of gauge becomes necessary since only $\nabla\times\mathbf{A}$ is determined. At this point Small chooses the Lorentz gauge. We shall perform the calculations using both the Lorentz and Coulomb gauges. The reason for this lies in the choice of gauge for the equivalent quantized lagrangian. The Standard Lagrangian of quantum-electrodynamics is given in Coulomb gauge. However, use of the Coulomb gauge presents problems in the determination of the classical lagrangian as we will see.

By using the (Lorentz) gauge condition

$$\nabla.\mathbf{A}=-\mu_0\varepsilon_0\frac{\partial\phi}{\partial t} \tag{4.28}$$

together with the identity

$$\nabla\times\nabla\times\mathbf{A}=\nabla\nabla.\mathbf{A}-\nabla^2\mathbf{A} \tag{4.29}$$

we can reduce equations (4.26) and (4.27) to

$$\nabla^2\mathbf{A}-\mu_0\varepsilon_0\frac{\partial^2\mathbf{A}}{\partial t^2}+\mu_0\frac{\partial\mathbf{P}}{\partial t}=0 \tag{4.30}$$

and

$$\nabla^2\phi-\mu_0\varepsilon_0\frac{\partial^2\phi}{\partial t^2}-\frac{1}{\varepsilon_0}\nabla.\mathbf{P}=0 \tag{4.31}$$

respectively. The equation describing the behaviour of the electrons (4.15) can also be modified to give

$$\frac{\partial^2\mathbf{P}}{\partial t^2}+\frac{\partial V}{\partial\mathbf{P}}=-\varepsilon_0\omega_p^2\left(\frac{\partial\mathbf{A}}{\partial t}+\nabla\phi\right) \tag{4.32}$$

The corresponding results in the Coulomb gauge $\nabla.\mathbf{A} = 0$ are

$$\nabla^2\mathbf{A} - \mu_0\varepsilon_0\frac{\partial^2\mathbf{A}}{\partial t^2} - \mu_0\varepsilon_0\frac{\partial}{\partial t}(\nabla\phi) + \mu_0\frac{\partial\mathbf{P}}{\partial t} = 0$$

$$\nabla^2\phi - \frac{1}{\varepsilon_0}\nabla.\mathbf{P} = 0$$

$$\frac{\partial^2\mathbf{P}}{\partial t^2} + \frac{\partial V}{\partial\mathbf{P}} = -\varepsilon_0\omega_p^2\left(\frac{\partial\mathbf{A}}{\partial t} + \nabla\phi\right)$$

(4.33a, b, c)

We note that equation (4.33b) does not imply that $\mathbf{P} = \nabla\varepsilon_0\phi$ since the curl of any gradient field may be added to **P** to give the same result when the divergence of both sides is taken.

Although desirable, there is no systematic way of determining a lagrangian given a set of equations of motion. Once the equations have been derived, the lagrangian is determined by trial and error tempered by experience. With this in mind, Small states that the lagrangian for the Lorentz gauge system which we present again as

$$\nabla^2\mathbf{A} - \mu_0\varepsilon_0\frac{\partial^2\mathbf{A}}{\partial t^2} + \mu_0\frac{\partial\mathbf{P}}{\partial t} = 0$$

$$\nabla^2\phi - \mu_0\varepsilon_0\frac{\partial^2\phi}{\partial t^2} - \frac{1}{\varepsilon_0}\nabla.\mathbf{P} = 0$$

$$\frac{\partial^2\mathbf{P}}{\partial t^2} + \frac{\partial V}{\partial\mathbf{P}} = -\varepsilon_0\omega_p^2\left(\frac{\partial\mathbf{A}}{\partial t} + \nabla\phi\right)$$

(4.34a, b, c)

is given by

$$L = \frac{\varepsilon_0}{2}\left(A_{i,t}^2 - c^2A_{i,x_k}^2\right) - A_{i,t}P_i - \frac{\varepsilon_0}{2c^2}\left(\phi_t^2 - c^2\phi_{x_k}^2\right) - \phi_{x_k}P_k + \frac{1}{\varepsilon_0\omega_p^2}\left(\frac{P_{i,t}^2}{2} - V(\mathbf{P})\right)$$

(4.35)

where the summations for i and k are to be taken to run from 1 to 3 to cover all three spatial dimensions. If one takes each of the functions **A**, ϕ, **P** to be functions of x, y, z and t and applies the appropriate Euler-Lagrange equation to the lagrangian given above (4.35) then one reobtains the equations of motion given as equations (4.34).

The choice of gauge has allowed Small to write the lagrangian simply as two terms describing wave motion for **A** and ϕ, two coupling terms and a final term to describe the nonlinearity of the polarisation **P** as a reflection of the nonlinear restoring force on the electrons. If one examines the equations of motion given in the two gauges, one notes the symmetry of the results as given in the Lorentz gauge. The results given in

the Coulomb gauge have a different coupling between the three dependent variables **A**, ϕ and **P**. It is the symmetry of the coupling which has allowed Small to write down the lagrangian. There is no obvious way of forming a three-dimensional lagrangian for the Coulomb gauge.

4.2.2.3. Reduction to Two Dimensions Plus Time

We have noted how the choice of gauge condition lead to difficulty in deriving an alternative lagrangian in the Coulomb gauge $\nabla.\mathbf{A} = 0$. The use of the Lorentz gauge, $\nabla.\mathbf{A} = -\mu_0\varepsilon_0\frac{\partial\phi}{\partial t}$, was instrumental in the construction of the lagrangian. If we suppose that there is no applied scalar potential, that is that there is no applied voltage across the device, then it is obvious that the lagrangians for the two gauges will be identical since the Lorentz gauge condition reduces to the Coulomb one. In the absence of an applied potential, the lagrangian becomes

$$L = \frac{\varepsilon_0}{2}\left(A_{i,t}^2 - c^2A_{i,x_k}^2\right) - A_{i,t}P_{ik} + \frac{1}{\varepsilon_0\omega_p^2}\left(\frac{P_{i,t}^2}{2} - V(\mathbf{P})\right) \tag{4.36}$$

The equations of motion reduce to the following set although (4.37b) cannot be derived from the lagrangian.

$$\nabla^2\mathbf{A} - \mu_0\varepsilon_0\frac{\partial^2\mathbf{A}}{\partial t^2} + \mu_0\frac{\partial\mathbf{P}}{\partial t} = 0$$

$$\nabla.\mathbf{P} = 0$$

$$\frac{\partial^2\mathbf{P}}{\partial t^2} + \frac{\partial V}{\partial\mathbf{P}} = -\varepsilon_0\omega_p^2\frac{\partial\mathbf{A}}{\partial t} \tag{4.37a, b, c}$$

Using (4.37a) and the equivalence $\mathbf{E} = -\frac{\partial\mathbf{A}}{\partial t}$ it is easy to show that **E** and **P** are coupled by the equation

$$\frac{\partial^2E_i}{\partial t^2} - c^2\nabla^2E_i = -\frac{1}{\varepsilon_0}\frac{\partial^2P_i}{\partial t^2} \tag{4.38}$$

In other words, there is no cross coupling of the components of the electric field and the polarisation. Thus, if we assume that the electric field **E** is propagating in the x-y plane

and has a component in the z direction only, we can then make a similar assumption for **P**. The electric field could look like Diagram 4.1 with the polarisation of the material being similar.

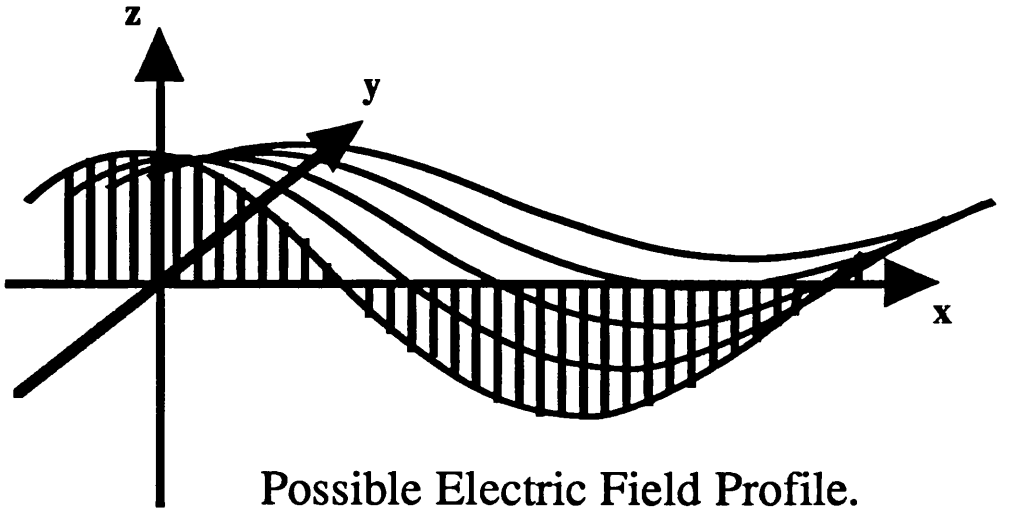


Diagram 4.1

A diagram of a possible electric field to show orientation.

If we assume, therefore that **E=E(x,y;t)** then **P=P(x,y;t)** and the equations of motion become

$$\frac{\partial^2 E}{\partial t^2} - c^2\left(\frac{\partial^2 E}{\partial x^2} + \frac{\partial^2 E}{\partial y^2}\right) = -\frac{1}{\varepsilon_0}\frac{\partial^2 P}{\partial t^2}$$

$$\frac{\partial^2 P}{\partial t^2} + \frac{\partial V}{\partial P} = \varepsilon_0 \omega_p^2 E$$

(4.39a, b)

where we have written E for the z component of the electric field and P for the z component of the polarisation. These equations are those used by Small in his thesis [2] and by Whitham in his book [1]. It may be seen that a lagrangian which correctly gives their equation of motion is

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2\left(A_x^2 + A_y^2\right)\right) - A_t P + \frac{1}{\varepsilon_0 \omega_p^2}\left(\frac{P_t^2}{2} - V(\mathbf{P})\right)$$

(4.40)

We will use this lagrangian later in the thesis as an example of a classical description of electromagnetic wave propagation in a medium. Note that the assumptions made in

deriving the lagrangian assume that the atoms are relatively dilute so that they have no effect on each other and so that the permeability and permittivity may be considered unchanged from the vacuum values. One can think of this as a medium which has a few dispersed active sites such as the lasing centres in ruby. One must also bear in mind that no provision is made within the lagrangian (4.40) for a potential to be applied across the medium which would be useful for modelling laser devices.

4.2.2.4. Boundary Conditions.

Throughout the derivation, no mention has been made of the use of boundary conditions. An infinite medium has been assumed throughout. Within the lagrangian formalism, it is possible to model bounded or layered media by optimising the lagrangian allowing the function to have cusps at the layer interfaces or, in the case of a bounded medium, by optimising with fixed end points. Such methods are discussed in chapter 2 of Elsgolc [6].

Using such a method, it would be possible to model a waveguide or other layered structure such as two coupled waveguides. The mathematics to perform such calculations is complicated and not amenable to hand calculation. However, it should not be beyond the capabilities of some future computer (algebra) program.

4.2.3. Quantized Lagrangians.

In the previous section we considered the derivation of a classical model of electromagnetic wave propagation in a nonlinear medium. The electrons of the model were constrained by a nonlinear restoring force which was proportional to the distance of the electron from the nucleus - essentially a "ball on a spring" mechanical model of the medium. It is our intention to present a semiclassical model allied to the classical one just presented in which the electrons will be permitted discrete energy levels and the nonlinearity will be introduced by the way the two (or more) quantum levels are coupled together. This model will be partly based on the hamiltonian descriptions used in the derivations of the Maxwell-Bloch equations presented earlier.

In addition, the use of lagrangians in quantum descriptions must be considered. There is an extensive literature which considers lagrangian descriptions of fully quantized or semiclassical systems. Some of this literature is relevant to the model and must therefore be considered. The use of the model touches on some fairly fundamental

problems in the use of quantum mechanics in the description of nonlinear optical processes and we will also touch on these briefly.

4.2.3.1. The Choice of Gauge.

In the derivation of the classical lagrangian describing the coupled radiation and atomic system, we mentioned that the choice of gauge provided an important means of simplifying the task of determining the lagrangian description. By using the Lorentz gauge

$$\nabla.\mathbf{A} = -\varepsilon_0\mu_0\frac{\partial\phi}{\partial t} \tag{4.41}$$

it was shown how a full three-dimensional classical lagrangian could be derived in order to describe the evolution of the coupled system. There was no obvious way in which the lagrangian for the system could be written in the Coulomb gauge

$$\nabla.\mathbf{A} = 0 \tag{4.42}$$

This is perhaps some indicator for the difficulties which must be faced when considering quantum descriptions of the electromagnetic field. The choice of the appropriate gauge condition can simplify the analysis necessary to determine the lagrangians or other quantum model. This is discussed in Loudon's book [7]. We will instead follow the treatment given by Cohen-Tannoudji and coworkers in their book "Photons and Atoms" [8]. One of the advantages of the choice of Lorentz gauge which we mention briefly here is the compact manner in which the equations can be written when the Lorentz gauge is used. Using the language of differential forms, all of classical electromagnetism can essentially be reduced to one equation $\partial_\mu A^\mu = 0$.

As the authors point out, however, when one wishes to quantize the classical description, the naïve approach to quantization of the theory simply replaces the classical variables with the quantum operators in the Coulomb gauge. The systematic derivation of the standard equations of quantum electrodynamics requires one to construct either a hamiltonian or a lagrangian for the the problem and then to study the conjugacy of the dynamical variables. The hamiltonian and lagrangian descriptions of any problem are related by means of a Legendre (hamiltonian) transformation. It turns out that, for the descriptions of physical problems, as opposed to abstract theoretical analysis, the

Coulomb gauge offers a simpler route to obtaining the quantized form of classical models of electromagnetism than the Lorentz gauge.

We start with the standard lagrangian

$$L=\sum_{\alpha}\frac{1}{2}m_{\alpha}\dot{\mathbf{r}}_{\alpha}^{2}+\frac{\varepsilon_0}{2}\int \mathrm{d}^3 r\left[\mathbf{E}^2(\mathbf{r})-c^2\mathbf{B}^2(\mathbf{r})\right]+\sum_{\alpha}\left[q_{\alpha}\dot{\mathbf{r}}_{\alpha}\cdot\mathbf{A}(\mathbf{r}_{\alpha})-q_{\alpha}U(\mathbf{r}_{\alpha})\right] \tag{4.43}$$

which describes a field of particles α at positions $\mathbf{r}_{\alpha}$ interacting with an electromagnetic field described by the potentials $\mathbf{A}$ and U. The fields $\mathbf{E}$ and $\mathbf{B}$ are given by the usual equations

$$\mathbf{E}(\mathbf{r})=-\nabla U-\dot{\mathbf{A}}(\mathbf{r})$$
$$\mathbf{B}(\mathbf{r})=\nabla\times\mathbf{A}(\mathbf{r}) \tag{4.44a,b}$$

The lagrangian is composed of three terms and so we can write

$$L=L_{\text{atoms}}+L_{\text{field}}+L_{\text{interaction}} \tag{4.45}$$

By introducing the charge density ρ and the current $\mathbf{j}$ one can write $L_{\text{interaction}}$ as

$$L_{\text{interaction}}=\int \mathrm{d}^3 r\left[\mathbf{j}(\mathbf{r})\cdot\mathbf{A}(\mathbf{r})-\rho(\mathbf{r})U(\mathbf{r})\right] \tag{4.46}$$

where the charge density and current are defined by

$$\rho(\mathbf{r},t)=\sum_{\alpha}q_{\alpha}\delta\left[\mathbf{r}-\mathbf{r}_{\alpha}(t)\right]$$
$$\mathbf{j}(\mathbf{r},t)=\sum_{\alpha}q_{\alpha}\mathbf{v}_{\alpha}\delta\left[\mathbf{r}-\mathbf{r}_{\alpha}(t)\right] \tag{4.47a,b}$$

so that the final form for the lagrangian can be written as

$$L=\sum_{\alpha}\frac{1}{2}m_{\alpha}\dot{\mathbf{r}}_{\alpha}^{2}+\int \mathrm{d}^3 r\mathcal{L}(\mathbf{r}) \tag{4.48}$$

where the lagrangian density $\mathcal{L}(\mathbf{r})$ is given by the expression

$$\mathcal{L}(\mathbf{r})=\frac{\varepsilon_0}{2}\left[\mathbf{E}^2(\mathbf{r})-c^2\mathbf{B}^2(\mathbf{r})\right]+\mathbf{j}(\mathbf{r})\cdot\mathbf{A}(\mathbf{r})-\rho(\mathbf{r})U(\mathbf{r}) \tag{4.49}$$

These expressions are given for comparisons with later results. However, it should be noted that the field described here is internal to the system and can not be fixed or varied

in the manner of an experimental parameter. If this is required, the lagrangian must be modified to take account of externally applied fields.

It is possible to show that the lagrangian (4.43) is invariant under translation, rotation and change of time origin. We know from Noether's theorem (see Chapter 1) that this implies that there must be three conserved quantities. They are the momentum, angular momentum and energy. It is further possible to show that the lagrangian given as (4.43) is the low-velocity limit of the relativistic description. We shall not concern ourselves with the relativistic model since it is unnecessary to use relativistic models at the energies we need to consider. A further point made is that a gauge transformation does not leave the lagrangian unchanged since changes occur to the interaction term. The result of these changes is to change the lagrangian into an equivalent lagrangian provided charge conservation holds.

The next step in the derivation of the quantized version of the standard lagrangian is to eliminate redundant variables. At each point in space, there are eight dynamical variables - the three vector potential components, the scalar potential, and their time-derivatives (the velocities). Since only four variables are required to describe the motion, for example the components of the electric and magnetic fields, there are too many degrees of freedom in these describing equations and therefore there must exist some constraining relations which will reduce the number of independent variables.

In fact, since the time derivative of the scalar potential does not appear in the lagrangian density, it is possible to eliminate the scalar potential from the equations thus leaving six independent components. In fact the further reduction required is achieved by the choice of gauge condition. The simplest choice for this gauge condition is to use the Coulomb gauge $\nabla.\mathbf{A} = 0$. One finds that after the elimination of the scalar potential, the lagrangian becomes

$$L = \sum_{\alpha} \frac{1}{2} m_{\alpha} \dot{\mathbf{r}}_{\alpha}^2 - V_{\text{Coul}} + \int \mathrm{d}^3 r \mathcal{L}_C \tag{4.50}$$

where V_{Coul} is the Coulomb energy of a system of particles given by

$$V_{\text{Coul}} = \sum_{\alpha} \frac{q_{\alpha}^2}{2\varepsilon_0 (2\pi)^3} \int \frac{\mathrm{d}^3 k}{k^2} + \sum_{\alpha > \beta} \frac{q_{\alpha} q_{\beta}}{4\pi\varepsilon_0 |\mathbf{r}_{\alpha} - \mathbf{r}_{\beta}|} \tag{4.51}$$

and the lagrangian density $\mathcal{L}_C$ is given by

$$\mathcal{L}_C = \frac{\varepsilon_0}{2}\left[\dot{\mathbf{A}}^2 - c^2(\nabla \times \mathbf{A})^2\right] + \mathbf{j}\cdot\mathbf{A}$$

(4.52)

Together with the gauge condition, these equations are now in a suitable form for quantization since they involve the correct number of independent variables. Once the transformation of these equations has been made into reciprocal space, the steps necessary in order to perform formal canonical quantization may be performed easily. We will not examine this process: our construction of a lagrangian does not rely on the details of the derivation of this fairly fundamental result.

4.2.3.2. Externally Applied Fields

Although we have considered the problem of a system of atoms interacting with an electromagnetic field in terms of the lagrangian formalism, there is obviously a corresponding hamiltonian description of the same problem and indeed such a description is more common than the lagrangian one. This hamiltonian can be formed from the lagrangian given earlier as

$$H = \sum_\alpha \frac{1}{2m_\alpha}\left[\mathbf{p}_\alpha - q_\alpha \mathbf{A}(\mathbf{r}_\alpha)\right]^2 + V_{\text{Coul}} + \frac{\varepsilon_0}{2}\int \mathrm{d}^3 r\left[\left(\frac{\Pi}{\varepsilon_0}\right)^2 + c^2(\nabla \times \mathbf{A})^2\right]$$

(4.53)

where

$$\Pi(\mathbf{r}) = \varepsilon_0 \dot{\mathbf{A}}(\mathbf{r})$$

(4.54)

and the conjugate momentum $\mathbf{p}_\alpha$ is given by

$$\mathbf{p}_\alpha = m_\alpha \dot{\mathbf{r}}_\alpha + q_\alpha \mathbf{A}(\mathbf{r}_\alpha)$$

(4.55)

It was previously pointed out that both the hamiltonian and the lagrangian descriptions of the fields refer to fields internal to the particle model. The fields do not refer to externally applied fields. In order to model a system where the field can be varied as an experimental parameter, it is necessary to introduce external sources for the desired fields. Thus both the hamiltonian and the lagrangian can be modified to take account of these additional fields. In fact, the form of the lagrangian is

$$L=\sum_{\alpha} m_{\alpha}\dot{\mathbf{r}}_{\alpha}^{2}+\frac{\varepsilon_0}{2}\int \mathrm{d}^3 r\left[\left(-\nabla U-\dot{\mathbf{A}}\right)^2-c^2(\nabla\times\mathbf{A})^2\right]+\int \mathrm{d}^3 r\left[\mathbf{j}_p\cdot(\mathbf{A}_e+\mathbf{A})-\rho_p(U_e+U)\right]$$

(4.56)

Here the subscript p denotes a quantity belonging to the system of particles only. It can be shown that the equations of motion which are derived from this lagrangian are

$$m_{\alpha}\ddot{\mathbf{r}}_{\alpha}=q_{\alpha}\left[\mathbf{E}_e+\mathbf{E}\right]+q_{\alpha}\dot{\mathbf{r}}_{\alpha}\times\left[\mathbf{B}_e+\mathbf{B}\right]$$

$$\Delta U+\nabla\cdot\dot{\mathbf{A}}=-\frac{\rho_p}{\varepsilon_0}$$

$$\ddot{\mathbf{A}}-c^2\Delta\mathbf{A}+c^2\nabla(\nabla\cdot\mathbf{A})+\nabla\dot{U}=\frac{\mathbf{j}_p}{\varepsilon_0}$$

(4.57a,b,c)

It can therefore be seen that the dynamics of the particles is determined by the total electromagnetic field whereas the dynamics of the fields A and U are determined only by the charge and current densities of the particles. The hamiltonian form of the lagrangian can be written as

$$H=H_{\text{atoms}}+H_{\text{field}}+H_{\text{interaction}}$$

(4.58)

The hamiltonian is composed of three parts. One part describes the atomic field, one part describes the behaviour of the electromagnetic field and the third part describes the coupling between the atoms and the electromagnetic field. The part which especially concerns us is the form of the interaction term. It can be written as

$$H_{\text{interaction}}=-\frac{q_{\alpha}}{m_{\alpha}}\sum_{\alpha}\mathbf{p}_{\alpha}^{e}\cdot\mathbf{A}(\mathbf{r}_{\alpha})+\frac{q_{\alpha}^2}{2m_{\alpha}}\mathbf{A}^2(\mathbf{r}_{\alpha})$$

(4.59)

where

$$\mathbf{p}_{\alpha}^{e}=\mathbf{p}_{\alpha}-q_{\alpha}\mathbf{A}_e$$

(4.60)

An expression similar to (4.59) can be found in Loudon [7] where it is referred to as the coupling terms of the minimal-coupling hamiltonian. The full hamiltonian of (4.58) can be transformed into a more convenient form by use of a unitary transformation which relies on the use of the Coulomb gauge condition. This is the Power-Zienau-Woolley transformation. Since our lagrangian model will involve a coupling term whose form is produced by an approximation based on expressions produced by this transformation we will consider it briefly in the next section.

4.2.3.3. The Form of the Interaction Term.

The transformation used to re-express the lagrangian in a more convenient form is called the Power-Zienau-Woolley transformation. We shall only sketch the derivation since the aim in introducing it is to discuss the terms which arise to describe the coupling between the two fields, not to use the derivation for any further purpose. The transformation is performed by adding to the Standard lagrangian an additional term which is a total derivative of a function F with respect to time which, since it is a total derivative, makes no difference to the final appearance of the equations of motion. The function F is given in the book by Cohen-Tannoudji and coworkers [8] as

$$F = -\int \mathrm{d}^3 r \mathbf{P}(\mathbf{r}) \cdot \mathbf{A}(\mathbf{r}) \tag{4.61}$$

where the polarization density $\mathbf{P}$ is given by

$$\mathbf{P}(\mathbf{r}) = \sum_{\alpha} \int_0^1 \mathrm{d}u q_\alpha \mathbf{r}_\alpha \delta(\mathbf{r} - u\mathbf{r}_\alpha) \tag{4.62}$$

Using the form of the Standard lagrangian given earlier (4.50) and defining the new lagrangian as

$$L' = L + \frac{\mathrm{d}F}{\mathrm{d}t} \tag{4.63}$$

we find that the form of the lagrangian can be given as

$$L' = \sum_{\alpha} \frac{1}{2} m_\alpha \dot{\mathbf{r}}_\alpha^2 - \sum_{\alpha > \beta} \frac{q_\alpha q_\beta}{4\pi\varepsilon_0 |\mathbf{r}_\alpha - \mathbf{r}_\beta|} - \sum_{\alpha} \varepsilon_{\mathrm{Coul}}^{\alpha} + \frac{\varepsilon_0}{2} \int \mathrm{d}^3 r \left[E_{\mathrm{Perp}}^2 - c^2 \mathbf{B}^2 \right]$$
$$+ \int \mathrm{d}^3 r \mathbf{M} \cdot \mathbf{B} + \int \mathrm{d}^3 r \mathbf{P} \cdot \mathbf{E}_{\mathrm{Perp}} \tag{4.64}$$

where $\varepsilon_{\mathrm{Coul}}^{\alpha}$ is given by

$$\varepsilon_{\mathrm{Coul}}^{\alpha} = \sum_{\alpha} \frac{q_\alpha^2}{2\varepsilon_0 (2\pi)^3} \int \frac{\mathrm{d}^3 k}{k^2} \tag{4.65}$$

and $\mathbf{M}$ the magnetization density, is given by

$$\mathbf{M}(\mathbf{r}) = \sum_{\alpha} u du q_{\alpha} \mathbf{r}_{\alpha} \times \dot{\mathbf{r}}_{\alpha} \delta(\mathbf{r} - u\mathbf{r}_{\alpha})$$

(4.66)

The last two terms are the interaction terms which can also be written as

$$L'_{\text{Int}} = \int d^3 r \mathbf{j} \cdot \mathbf{A} - \int d^3 r \left(\dot{\mathbf{P}} \cdot \mathbf{A} + \mathbf{P} \cdot \dot{\mathbf{A}} \right)$$

(4.67)

In order to rewrite this expression in terms of an expansion in powers of the wavelength, we define the vector potential in terms of an expansion in which the longer wavelength modes are separated from the shorter ones. (Here "longer" means large on the scale of a nucleus.) Thus

$$\mathbf{A}(\mathbf{r}) = \mathbf{A}^{<}(\mathbf{r}) + \mathbf{A}^{>}(\mathbf{r})$$

(4.68)

The transformation is redefined so that only the longer-wavelength modes are included in the interval. Using the definition

$$F = -\int d^3 r \mathbf{P} \cdot \mathbf{A}^{<}$$

(4.69)

one obtains the expressions

$$L'_{\text{Int}} = L^{<}_{\text{Int}} + L^{>}_{\text{Int}}$$
$$L^{>}_{\text{Int}} = \int d^3 r \mathbf{j} \cdot \mathbf{A}^{>}$$
$$L^{<}_{\text{Int}} = \int d^3 r \mathbf{M} \cdot \mathbf{B}^{<} + \int d^3 r \mathbf{P} . \mathbf{E}^{<}_{\text{Perp}}$$

(4.70a,b,c)

for the interaction terms. One then expands the fields $\mathbf{B}^{<}$ and $\mathbf{E}^{<}_{\text{Perp}}$ in a Taylor series. After evaluation of the integrals which include delta functions in the definitions of $\mathbf{M}$ and $\mathbf{P}$ one then obtains the following expression for $L^{<}_{\text{Int}}$

$$L^{<}_{\text{Int}} = \mathbf{d} \cdot \mathbf{E}^{<}_{\text{Perp}}(0) + \mathbf{m} . \mathbf{B}^{<}(0) + \sum_{i,j} q_{ij} \frac{\partial}{\partial x_i} E^{<}_{\text{Perp},j}(0) + \ldots$$

(4.71)

Here $\mathbf{d}$, $\mathbf{m}$, and q_{ij} are given by the following expressions.

$$\mathbf{d} = \sum_{\alpha} q_{\alpha} \mathbf{r}_{\alpha}$$

$$\mathbf{m} = \sum_{\alpha} \frac{1}{2} q_{\alpha} \mathbf{r}_{\alpha} \times \dot{\mathbf{r}}_{\alpha}$$

$$q_{ij} = \sum_{\alpha} \frac{1}{2} q_{\alpha} \left(r_{\alpha i} r_{\alpha j} - \frac{1}{3} \delta_{ij} \mathbf{r}_{\alpha}^2 \right)$$

(4.72a,b,c)

We see that the transformation has allowed us to write the interaction terms as a multipole expansion. The first term in this expansion is the electric dipole term and it is larger than the other two by the order of the fine structure constant. If one dispenses with the other two terms - the electric quadrupole term and the magnetic dipole term - this is termed the electric dipole approximation. It is this form of the coupling term which will be used in the subsequent lagrangian model and our purpose in following this derivation has been to introduce this term

Now since the two expressions describe the same experimental situation and are further related by a unitary transformation, it would be thought that results calculated using either of these two expressions would be identical. This is indeed the case. However, there is some discussion in the literature of there being differing results depending on which description is used in the calculation of the results. For example, a paper by Power and Zienau [9] discusses the use of the dipole coupling scheme and states that it seems to give agreement with experiment whereas that using vector potential coupling terms does not. A later paper by Fried [10] states that it may be shown that the two descriptions will give similar results provided certain conditions, namely the conservation of energy and the inclusion of all coherent processes to the required order, are met. In fact, the discussion of this topic in the book by Cohen-Tannoudji and coworkers [8] states that the two approaches are equivalent and that previous papers, in which it has been suggested that one of these approaches is to be preferred, have been the result of misinterpretations or misunderstandings of the equivalence between the two approaches. Essentially, the "mathematical formulation depends on the representation" of the problem: in one representation, the eigenfunctions may be easier to determine than in another. Use of one representation may introduce subtle errors during the course of the calculation because of the way in which the relative importance of the terms varies between the representations.

In essence, the problem may be said to be caused by an excessive reliance upon the describing formulæ. If one bears in mind that there is a fairly fundamental problem in retaining sufficient information in making calculations with either model, then either

model provides an accurate description of the physical situation and the choice of model may therefore be made by judging the facility with which each model lends itself to calculation of the physical results.

This difference in interpretation should also be borne in mind when considering the quantum lagrangian model presented in the next section. Furthermore, the model uses the electric dipole approximation. Since the electric dipole approximation is a good one, further terms being different by an order of the fine structure constant, for the purposes of comparison with the other qualitative model so far presented, the approximation should make no difference to the results obtained although, if quantitative models are to be developed, this inherent approximation may require correction by the inclusion of further terms.

4.2.3.4. The Quantum Lagrangian.

We now present a simple model for a two-level quantum system coupled to a classical electric field. This model has previously been given by Arnold [11]. It uses a vector approach to describe the occupation of the quantum level and matrices to represent the coupling between the electrons and the classical electric field. Essentially, the postulation of a lagrangian, or action-functional, having suitable equations of motion describing the behaviour of the polarization and occupations of the quantum levels enables a model of the system to be put into a suitable form for calculation using the Kawahara variant of Whitham's method.

Lagrangian formulations of Schrödinger's equation are well known. For example, in the book by Cohen-Tannoudji and coworkers, the following lagrangian is given as representing a charged particle in an electromagnetic field

$$\mathcal{L}=\frac{i\hbar}{2}\left(\psi^{*}\dot{\psi}-\dot{\psi}^{*}\psi\right)-\frac{1}{2m}\left[\left(-\frac{\hbar}{i}\nabla-q\mathbf{A}\right)\psi^{*}\right]\left[\left(\frac{\hbar}{i}\nabla-q\mathbf{A}\right)\psi\right]-(V(r)+qU)\psi^{*}\psi \tag{4.73}$$

Its equation of motion is given by

$$i\hbar\dot{\psi}-(V+Uq)\psi-\frac{1}{2m}\left(\frac{\hbar}{i}\nabla-q\mathbf{A}\right)^{2}\psi=0 \tag{4.74}$$

However, we wish to use a notation related to the density matrix formulation: much of the description of the derivation of the Maxwell-Bloch equations for the two-level system presented in chapter 2 used such a formalism.

The time-independent Schrödinger equation is

$$i\hbar\frac{\partial\psi}{\partial t}=\hat{H}\psi \tag{4.75}$$

We assume that the wavefunction ψ refers to that of an N-level system so that we can write

$$\psi=\sum_{k=1}^{N}v_k\psi_k \tag{4.76}$$

The v_k are the probabilities of occupation of the eigenstate ψ_k whose eigenfunctions are assumed to be known. Substituting the expression (4.76) in (4.75) and simplifying assuming that the states ψ_k are normalised and orthonormal gives

$$\sum_{i,j=1}^{N}v_i^*v_j\left\langle\psi_i^*\left|\hat{H}\right|\psi_j\right\rangle=\sum_{i,j=1}^{N}v_i^*v_j\hat{H}_{ij} \tag{4.77}$$

for the right hand side of (4.75) and

$$i\hbar\sum_{i=1}^{N}v_i^*\frac{\partial v_i}{\partial t} \tag{4.78}$$

for the left hand side. If we select the component k of the resulting equation

$$i\hbar\sum_{i=1}^{N}v_i^*\frac{\partial v_i}{\partial t}=\sum_{i,j=1}^{N}v_i^*v_j\hat{H}_{ij} \tag{4.79}$$

we obtain

$$i\hbar\frac{\partial v_k}{\partial t}=\sum_{j=1}^{N}\hat{H}_{kj}v_j \tag{4.80}$$

which can be written in vector form as

$$i\hbar\frac{\partial \mathbf{v}}{\partial t} = \hat{H}\mathbf{v}$$

(4.81)

In other words, the use of the occupation probabilities forms an alternative to the use of the density matrix which can be written as

$$\rho = \mathbf{v} \otimes \mathbf{v}^{\dagger}$$

(4.82)

where the dagger represents the hermitian conjugate. We can obviously write this in lagrangian form. Either by inspection or by carrying out a Legendre transformation we can see that the lagrangian form for the above equation (4.81) is

$$L = \mathbf{v}^{\dagger}\hat{H}\mathbf{v} - i\hbar\mathbf{v}^{\dagger}\frac{\partial \mathbf{v}}{\partial t}$$

(4.83)

If we apply the Euler-Lagrange equation appropriate to this lagrangian we obtain the conjugate expressions

$$-i\hbar\frac{\partial}{\partial t}\mathbf{v}^{\dagger} = \mathbf{v}^{\dagger}\hat{H}$$

$$i\hbar\frac{\partial}{\partial t}\mathbf{v} = \hat{H}\mathbf{v}$$

(4.84a,b)

for the equations of motion as required.

Now examine the form of the classical lagrangian given by Small which was presented in section 4.2.2.3. as (4.40).

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2A_x^2\right) - A_tP + \frac{1}{2\varepsilon_0\omega_p^2}\left(P_t^2 - V(P)\right)$$

(4.85)

The first term describes the field, the second the coupling between field and atoms and the third the behaviour of the atomic system. We can use the first term again to describe the behaviour of the field, replace the polarization in the coupling term with its quantum equivalent and replace the third term with the lagrangian for Schrödinger's equation. The equation becomes

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2A_x^2\right) - A_tN\left\langle \mathbf{v}^{\dagger}\middle|\hat{P}\middle|\mathbf{v}\right\rangle + N\left\langle \mathbf{v}^{\dagger}\middle|i\hbar\frac{\partial}{\partial t} - \hat{H}\middle|\mathbf{v}\right\rangle$$

(4.86)

where N is the density of the atomic sites.

4.2.3.5. The Form of the Matrix Operators.

We use the electric dipole approximation and assume that the hamiltonian for the problem can be written as

$$\hat{H} = \hat{H}_0 + \mathbf{d}\cdot\mathbf{E} \tag{4.87}$$

Since the eigenstates of the operator have fixed energy level , we can write

$$\hat{H}_0\psi_i = E_i\psi_i \tag{4.88}$$

Thus the full vector form of the equations can be written as

$$\begin{aligned}
&\begin{pmatrix} v_1^* & v_2^* & v_3^* & \dots \end{pmatrix}\left(i\hbar\frac{\partial}{\partial t}\right)\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} v_1^* & v_2^* & v_3^* & \dots \end{pmatrix}\begin{pmatrix} E_1 & H_{12} & H_{13} & \dots \\ H_{21} & E_2 & H_{23} & \\ H_{31} & H_{32} & E_3 & \\ \vdots & & & \ddots \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} v_1^* & v_2^* & v_3^* & \dots \end{pmatrix}\left(\operatorname{diag}(E_1,E_2,E_3\dots) + \begin{pmatrix} 0 & H_{12} & H_{13} & \dots \\ H_{21} & 0 & H_{23} & \\ H_{31} & H_{32} & 0 & \\ \vdots & & & \ddots \end{pmatrix}\right)\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix}
\end{aligned} \tag{4.89}$$

The terms H_{12} and so on are given by the dipole coupling term.For example, H_{12} is given by

$$\begin{aligned}
\langle\psi_1|\hat{H}|\psi_2\rangle &= \langle\psi_1|\hat{\mathbf{E}}\cdot\hat{\mathbf{d}}|\psi_2\rangle \\
&= E\langle\psi_1|\hat{\mathbf{d}}|\psi_2\rangle \\
&= Ed_{12}
\end{aligned} \tag{4.90}$$

where we have decorrelated the operators for the electric field and the polarization and have introduced an obvious extension to the notation. If we further assume that each

quantum level has a definite parity and choose the phases so that $d_{12} = d_{21}$ then the dipole matrix can be written as

$$\begin{pmatrix} 0 & d_{12} & 0 & \cdots \\ d_{21} & 0 & d_{23} & \\ 0 & d_{32} & 0 & \\ \vdots & & & \ddots \end{pmatrix} \tag{4.91}$$

This can be inserted into the lagrangian model by writing the lagrangian as

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2 A_x^2\right) - A_t N\begin{pmatrix} v_1^* & v_2^* & v_3^* & \cdots \end{pmatrix}\begin{pmatrix} 0 & d_{12} & 0 & \cdots \\ d_{21} & 0 & d_{23} & \\ 0 & d_{32} & 0 & \\ \vdots & & & \ddots \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix}$$

$$+i\hbar N\begin{pmatrix} v_1^* & v_2^* & v_3^* & \cdots \end{pmatrix}\frac{\partial}{\partial t}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix} - \begin{pmatrix} v_1^* & v_2^* & v_3^* & \cdots \end{pmatrix}\begin{pmatrix} E_1 & 0 & 0 & \cdots \\ 0 & E_2 & 0 & \\ 0 & 0 & E_3 & \\ \vdots & & & \ddots \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix} \tag{4.92}$$

Note that if we had wished, it would have been possible to express this using the appropriate matrix representation of SU(N). This is why the Lie group formalism was introduced in chapter 2 during the derivation of the Maxwell-Bloch equations.

4.2.3.6. The Quantum Lagrangian for a 2-level System.

The two-level version of the above lagrangian is obviously

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2 A_x^2\right) - A_t N\begin{pmatrix} v_1^* & v_2^* \end{pmatrix}\begin{pmatrix} 0 & d_{12} \\ d_{21} & 0 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + i\hbar N\begin{pmatrix} v_1^* & v_2^* \end{pmatrix}\frac{\partial}{\partial t}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

$$-\begin{pmatrix} v_1^* & v_2^* \end{pmatrix}\begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{4.93}$$

Here the independent variables are A for the field and v_1^* and v_2^*. The hermitian conjugates of the two vector components yield equations which are the conjugates of those generated by v_1^* and v_2^*. If we multiply the vector and matrix expressions to form a

scalar expression, we will have an expression which can then be treated as any other lagrangian in three variables or five if we include the conjugates.

In fact this is not strictly true. We will need to consider the series used to represent the components of the state occupation vector v. We will discuss this in chapter 6. In addition, the lagrangians could be rescaled to give forms in dimensionless variables. This has the immediate advantage that it is possible to see that the nonlinearity in both the classical and semi-classical cases comes from different sources. It is also possible to judge the effects of the various terms in the lagrangians. The quantum lagrangian presented here demonstrates this in that the small effects on the quantum scale and the large effects on the scale of the radiation are coupled together using the number density of the atomic sites, a large number.

4.3. Implementing Whitham's Method as a REDUCE Computer Program.

The example given in which the Boussinesq equation was investigated using the Kawahara variant of Whitham's method will have made obvious the prime disadvantage to using Whitham's method. In investigating any lagrangian, the number of terms which must be processed is very large. In the case of the Boussinesq equations, for example, it has been pointed out that the calculation necessary to obtain results from which the nonlinear Schrödinger equation could be derived took three weeks. The calculation was extremely tedious and was later discovered to be incorrect. If higher order results are required then the method simply becomes completely unmanageable. Before the advent of computer algebra systems the solution to this problem would simply have been to have chosen a different method or to have made some simplifying assumptions within the model being considered. However, as a first attempt at avoiding this predicament, it was decided to implement the Kawahara variant of Whitham's method using the REDUCE computer algebra package.

4.3.1. Introduction.

The REDUCE computer algebra system was developed by Anthony C. Hearn and is marketed and distributed by the Rand Corporation. It was the first computer algebra package to be widely distributed for general use although specialised systems

had been developed previously and, indeed, some are still in use in particular subject areas. The version of REDUCE used was version 3.3. To quote from the user's manual [12], REDUCE is "an algebraic programming system."

What this means is that REDUCE is a system whereby algebraic, that is symbolic, calculations may be performed on a computer. The necessary instructions to be followed are incorporated into a computer program just like FORTRAN or LISP. The system can understand the standard operations and functions of mathematics and if new functions are required, facilities exist for defining these functions in terms of standard programming concepts such as looping, pattern matching, substitution and so on.

REDUCE is written in the LISP programming language. It can be run in two possible modes symbolic mode and algebraic mode. Algebraic mode is the mode which is most used in that it makes assumptions about the purpose of input expressions which are not made in symbolic mode. Symbolic mode is actually a variant of the LISP programming language. Code written for use in symbolic mode is written in RLISP and therefore requires the user to be familiar with LISP programming whereas, in algebraic mode, the user may ignore the fact that the program is itself written in LISP and write code solely in the REDUCE programming language.

The code written to implement the Whitham procedure was written to run in REDUCE's algebraic mode. This meant that it was unnecessary for the author to learn LISP programming. A second point to make about symbolic mode programming is that whilst facilities exist to allow the passing of information between the two modes, writing code to run in symbolic mode requires a thorough knowledge of the internal workings of the REDUCE system and is not to be attempted lightly. If, for example, it were desired to modify the workings of the REDUCE kernel, it would be possible to do this by using code written in RLISP running in symbolic mode. However, such a modification is not easy to perform and would present a major programming task.

REDUCE was originally written to be used in solving problems in physics and so it has a bias which makes it somewhat easier to use for certain types of problems. It is also rule-based so that, in some ways, obtaining the answers to particular problems is easier than with procedural languages such as MAPLE. On the other hand, this means that it is easier to implement a separate program for each problem rather than writing a general solution.

In view of the above statement, we will now examine a program which is designed to solve the problem of obtaining the averaged lagrangian for the Boussinesq equation. This is chosen for three reasons. Firstly it may be compared with the working

given previously for this equation in which the Kawahara variant was explained and the steps necessary in the calculation were carried out. In this manner, the similarity of the REDUCE program to the working carried out by hand may be judged. Secondly, the REDUCE program may be compared with the MAPLE program to obtain the same result. Finally, the calculated result may be compared with that given in Jeffrey and Kawahara's book [13] as a check that the result obtained is correct.

It might be asked why the comparison is not made using a lagrangian from nonlinear optics. The reason for this is that, as explained above, to obtain a result for a lagrangian requires that part of the program be rewritten in the case of the REDUCE code. Moreover, as we will see, the result obtained in the case of the REDUCE program is not the equations of motion for the lagrangian. Only the averaged lagrangian is obtained. In the case of the MAPLE program which is the subject of the next chapter, it is the equations of motion which are obtained as a final result although obviously the averaged lagrangian must be obtained during the course of the calculation. In order to compare the results obtained by the two methods, therefore, one must either compare the averaged lagrangians themselves or calculate the equations of motion from the averaged lagrangian obtained by the REDUCE program. The number of terms which will be present in the averaged lagrangian is large. Thus the comparison of the two sets of output would either require a manual comparison or the writing of a further computer program to convert the output expressions of one program to those of the other. On the other hand, calculating the equations of motion from the averaged lagrangian would still be a lengthy and error prone exercise. Given that there is no reasonable way to make REDUCE calculate the equations of motion for any general lagrangian, it seems reasonable, therefore, to present results for the one example that was worked through using all three methods for comparison. The reason why it REDUCE cannot be used to calculate the equations of motion is the subject of a forthcoming section (4.3.4).

4.3.2. Implementation for Small's Lagrangian.

A complete listing of the program which determines the averaged lagrangian for the Boussinesq equation is given in the appendices. The program makes use of a separate file, called *AVER2*, which is read in to the program at execution time. This file contains the code which carries out the averaging of the lagrangian. The purpose of this modification was to separate utility code from code which was particular to the problem to be solved by the program. The program given in the appendix was written after this

modification was carried out and so has two separate files. Earlier versions of the program were not separated in this way. The *AVER2* file is also given separately since it is described in the next section and since it might conceivably be of use to other workers in this area.

The first action of the code is to define the *operators* to be used by the code. These are not operators in the strict mathematical sense: rather they are in some way a notational convenience for denoting certain operations to be carried out on variable arguments. The *DEEX* and *DEET* operators mimic the differentiation operators for x and t as might be expected. The operators *SIGMA1* and *SIGMA2* are used to represent summations which are formed in the expansion of the functions whose extremals are to be found. The functions of the operators *A1*, *A1STAR*, *A2* and *A2STAR* is, however, not so obvious. Rather than thinking of these entities as operators, it is better to consider them as indexed functions.

The purpose of the *ORDER* and *FACTOR* statements is really self-evident as is the list of *DEPEND* statements. The *ORDER* statements are not strictly necessary but are there for convenience when transcribing the results from the printout to paper for later manipulation by hand.

The rules given as *FOR* statements following the *COMMENT* line serve to implement the redefined differentiation operators. (*DF* is the REDUCE partial differentiation operator.) The subsequent two rules allow the differentiation operator to differentiate the two summation operators defined previously. Finally a *LET* statement:

LET EPSLN**7=0

is used to terminate the series produced by the program at some convenient point. The value of ε^7 is given above only as an example. For higher order calculations, one simply increases this limit. Of course higher order calculations do require the user to make sure that the *DEPEND* statements and summation rules have also been extended to generate all of the required terms.

What follows in the listing is the statement to read in the AVER2 file which is described separately. The program content relevant to the particular problem of the Boussinesq equation follows this definition of the *AVERAGE* procedure.

The remainder of the program is remarkably short. We define four functions to serve as the differentiations required in the lagrangian. For example, we define *FSUBT!↑2* to be the equivalent of the operator $\left(\frac{\partial F}{\partial t}\right)^2$ and similarly for the other operators. (The exclamation mark in the previous REDUCE expression has no

mathematical significance.) We can then define the function F in terms of the sum of terms given as

F:= TERM1+EPSLN*TERM2+EPSLN**2*TERM3

This will be used to form the summation which we have previously given as

$$f = \sum_{n=0}^{\infty} \varepsilon^{n+1} f_n,\ 0 < \varepsilon << 1 \tag{4.94}$$

where each f_n is given as

$$f_n = \sum_{l=1}^{\infty} A_{l-1}^n \exp(il\theta) + A_{l-1}^{n\,*} \exp(-il\theta) + B_n,\ \text{for n} \neq 0 \tag{4.95}$$

with f_0 given as

$$f_0 = A\exp(i\theta) + A^* \exp(-i\theta) + B \tag{4.96}$$

Note that the REDUCE expression starts at order ε^0 and that in (4.94) at ε^1. Earlier calculations carried out using the Kawahara variant used the REDUCE expression's scaling to agree with the expressions to be found in Jeffrey and Kawahara's book [13]. The discovery of an error in the book lead to the adjustment of the scaling to agree with equation (4.94) and this scaling was used in subsequent work.

The next action of the program is to define the lagrangian being used. The definition of the lagrangian is sufficient to trigger its evaluation in terms of the rules defined in the previous few lines. It therefore makes sense to ask the program what the coefficients of each power of ε are. This is included in the code to allow the operators for the determination of the lagrangian at each order to be calculated and displayed for comparison with those derived by hand.

Finally the code replaces the placeholders *TERM1*, *TERM2* and *TERM3* with the functions given in the previous two equations. Note how the summations given for f_1, f_2, and so on are kept as summations. As we will see in the next section, this is a source of difficulty in deriving the averaged form of the lagrangians since a means is required for simplifying products and powers of these summation objects.

Once the substitutions for the terms in the series F have been made, the final step is to apply the AVERAGE function to the series generated. This then calculates the form of the averaged lagrangian at each of the required orders as far as is possible within the limitations of the program. These limitations concern the handling of the summations

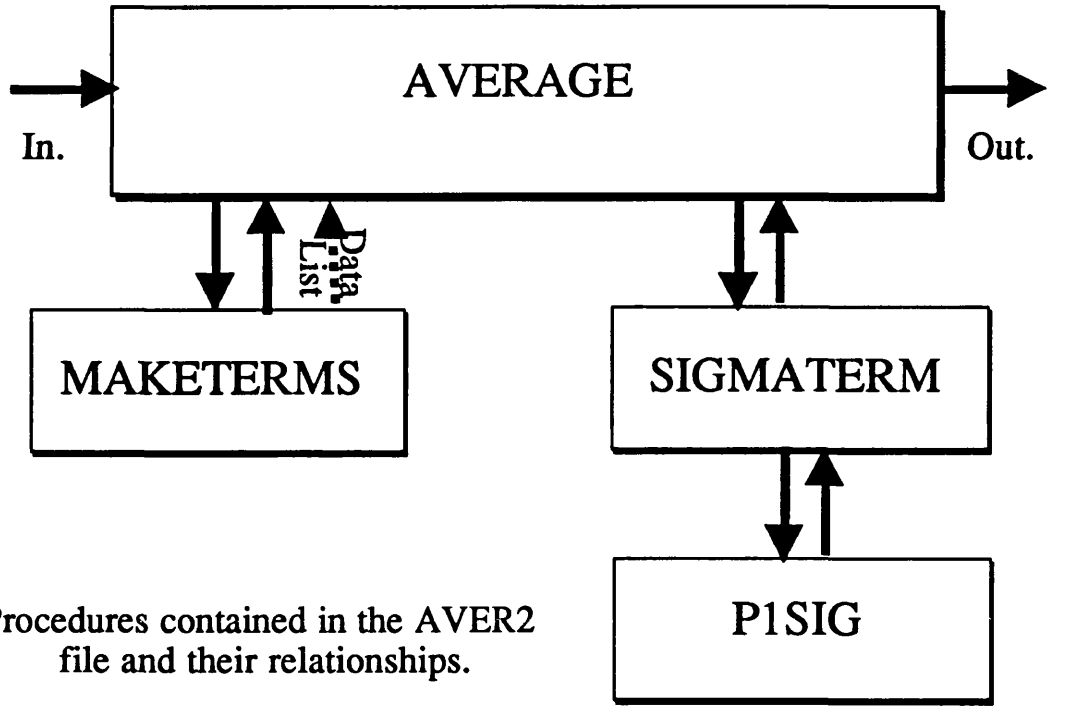


Diagram 4.2.
The Procedures Contained in the *AVER2* file and their Relationships.

SIGMA1 and *SIGMA2* together with limitations imposed by the package itself. We will now go on to discuss the AVER2 file. This will explain why simplifying the forms calculated for the averaged lagrangian causes difficulty. In section 4.3.4, we will discuss why the REDUCE program imposes its own limitations which render a general purpose program for deriving the equations of motion for an averaged lagrangian a matter of difficulty.

4.3.3. The *AVER2* File.

The *AVER2* file contains the code written to implement a new function in REDUCE namely that of *AVERAGE()*. The averaging function is itself implemented using two functions: *MAKETERMS* and *SIGMATERM*. *SIGMATERM* uses a procedure *P1SIG* in the course of its calculations. A simple diagram of the method of operation of the averaging function is given in diagram 4.2.

It can be seen from the diagram that the *AVERAGE* procedure is the top level procedure which uses *MAKETERMS* and *SIGMATERM* as service procedures. The purpose of the *AVERAGE* procedure is to use the *MAKETERMS* procedure to turn the polynomial representing the lagrangian into a list in which each member of the list contains one term of the polynomial. Once this has been done, it can decide which terms to keep or discard by examining them for the presence of factors involving exponentials. However, the terms in the lagrangian still involve summation operators and the procedure *SIGMATERM* is used to handle terms contained within a summation. The procedure *SIGMATERM* actually examines the terms to see if they may be simplified since only a certain range of expressions lies within the scope of the program. If simplification of a term involving a summation operator is possible, this simplification is performed by the procedure *P1SIG*. Otherwise, the expression is left unsimplified. We now go on to discuss each of the procedures in turn.

4.3.3.1. The *AVERAGE* Procedure.

The averaging procedure is itself quite short since most of the difficult work is performed by the other procedures. The first action of the code is to use the procedure *MAKETERMS* to convert the polynomial representing the lagrangian to a list in which each member of the list is one term of the polynomial. After the procedure has checked for an empty list as a safety feature, we discover one of shortcomings of the REDUCE package. In simulating integration, the code checks to see whether a term contains a power of an exponential. Any term which contains a harmonic term can be discarded since it will integrate to give zero. However, the *coeffn* operator, the operator which can be used to check for the presence of powers of harmonic terms, does not understand negative powers of an object. It is therefore necessary to check the numerator and denominator of each term separately.

The code therefore proceeds, as it reforms the sum which will eventually be the averaged lagrangian, by checking each of the members of the list in turn. If a member of the list contains an unresolved summation, the summation is replaced by *SIGMATERM* of the summation. The *SIGMATERM* function is used to resolve terms involving summation operators. Otherwise, the term may be considered to be a simple one. Checking proceeds by examining the denominator for the presence of harmonics. If a harmonic is found the term is discarded. Then the numerator is treated similarly. These actions must be performed in this order since, if the order is reversed and the numerator

is checked first, the presence of a negative power of a harmonic in the denominator will cause the REDUCE package to fail oddly. This represents a second difficulty experienced with the REDUCE package. We will meet further difficulties later.

Once each term in the list has been processed, the eventual result will be a polynomial in which each of the terms will have been averaged. The only exception to this will be where an expression involving summations has been passed to the *SIGMATERM* function which is too complicated for it to handle. In this case the result returned will contain these terms in an unsimplified form. In other words, the AVERAGE procedure takes a polynomial which will contain summation functions, and averages it by simulating integration and using the *SIGMATERM* procedure to expand the summations whilst in the course of the averaging procedure itself.

4.3.3.2. The *MAKETERMS* Procedure.

The purpose of the *MAKETERMS* procedure is to take a polynomial and to turn it into a list in which each member of the list is a term within the polynomial. This task seems relatively simple. However, the REDUCE system has certain vagaries which make this a nontrivial task. The problem arises because of the possible presence of a unary minus in the numerator of the polynomial. If the polynomial is premultiplied by a unary minus the REDUCE program will produce an error.

Accordingly, after performing some elementary error checking, the first action of this procedure is to examine the numerator of the polynomial for a unary minus. If this minus is found, it is stored so that it may be multiplied with the output from the remainder of the procedure. The numerator itself is negated so that the expression to be considered has a positively signed numerator. The code then simply checks to see that the modified numerator of the argument passed to it is a polynomial. If it is not, the procedure terminates with an error; if it is then the code forms the list by taking each term from the polynomial numerator, dividing it by the denominator and multiplying in the possible unary minus discussed earlier.

4.3.3.3. The *SIGMATERM* Procedure.

This is the procedure to which are passed any summation operators which are found in the lagrangian during averaging. We have already noted how the summations at each level are assigned to two summation operators *SIGMA1* and *SIGMA2*. If higher

orders of calculation were required then further summation operators would need to be defined. If a member of the list of terms contains a *SIGMA1* or a *SIGMA2* then the whole term is passed to the procedure *SIGMATERM* which will then decide if the term is to be simplified using the *P1SIG* procedure or to be returned unaltered.

The code begins by assuming that the term passed to it as an argument comprises a numerator having a numeric part and also some function of the SIGMA operators and a denominator which has only a numeric part. That is, there is no occurrence of functions or operators in the denominator of the term. This should be the form of the terms passed to the procedure anyway.

After checking for an unusual argument, the procedure can then go on to break the argument down into its constituent parts. Again, the first step is to check for the presence of a unary minus. If one is present then this is removed as a sign factor to be multiplied back in later. The next step is to separate any products of SIGMA's from any possible numeric part in the numerator and to form a list of numeric and summation operators to be found in the numerator. A problem arises here because if a numerator consists only of one of the summation operators then the next stage in the processing will split the summation from its exponent but treat the result as if the two entities were a product. That is, SIGMA1**2 would be treated as if it were SIGMA1*2. Accordingly two possible routes lead to the next result which is a list, called NEWLIST, which contains the SIGMA1 term, the SIGMA2 term, and the numeric part of the numerator divided by the denominator. Both of the summation operators may be raised to some power.

The program then removes the numeric part from its consideration and looks at the remainder of the list. If this remainder has length greater than one then the term will contain some product of the two summation functions. The procedure then gives up returning the whole term unchanged. If the power to which the summation has been raised is greater than one, the procedure again gives up returning the term unchanged. Finally, if the list contains one summation to the first power then the result passed back to the main procedure is the product of the sign term, the numeric part of the numerator divided by the denominator, and the summation operator as simplified by the P1SIG function. In other words, the function of the SIGMATERM function is to pass only first powers of some summation operator on to the procedure P1SIG for processing, leaving all other terms unchanged. It was intended that further additional procedures would be implemented to simplify higher powers of the summation operators but since the program was subsequently scrapped, these procedures were never written.

4.3.3.4 The *P1SIG* Procedure.

This procedure carries out the bulk of the work required in averaging summations involving either of the two sigma operators. The arguments passed to the procedure are a numeric part which comprises the numeric part of the numerator divided by the denominator of the term, followed by the summation operator which is raised to the first power only. The sign of the term is handled by having the result calculated within the SIGMATERM procedure as it returns its result. Thus the P1SIG procedure must only simplify the summation by considering the product of any exponentials contained within the term on the summation operator. Any exponential term retained will be the result of choosing a term from the series to be retained by the simulated integration.

The first task of the function, therefore, is to separate the numerator and denominator of the numeric argument and check to see if either contains any exponential functions. If there are none, then the whole term may be discarded since any integration of the exponentials contained within the summation will still result in a value of zero. The next step is again to check for the presence of unary minus signs before the summation operator. The sign of the summation operator is again separated off as a multiplicative factor. The procedure then obtains the argument of the summation function and uses the MAKETERMS procedure again to turn the sum in to a list (of two elements).

Again we must check for the sign of each term and split off any unary minus signs. The reason for this is that we will subsequently apply the *log* function to the exponentials and the logarithm function will obviously fail if it is asked to find the *log* of a negative number. We therefore create two lists, one containing the signs of the terms and one containing the positive parts of the terms. Taking the list containing the positive parts of the terms, we multiply by the exponential which premultiplied the summation and then take the log of the result. The result will be a list of two elements, each element containing logs and numbers formed by the cancellation of the exponentials. The log terms are discarded leaving only a list containing two elements. Each element will be of the form (index±number). The *solve* function is then applied to each of these elements, solving for the index that will give the result zero. The result is two equations in the relevant index for the term which will be retained We must then determine which one of these values to use.

The values determined will have the same numeric value but opposite sign and the value required is the positive one. The index to be determined does not include the sign of the exponent so that if the positive value of the solution is chosen, this will determine the correct solution in the case where the harmonic premultiplying the sum has a positive or a negative sign. Accordingly, the numeric parts of the solution equations are determined and the maximum of the two values is chosen. The element of the solution list from which the index value comes also determines which of the two forms within the summation will be used to form the term to be retained. Thus the correct form can be chosen, the numeric value of the index substituted for the index label, the sign and the numeric part of the term multiplied together with the form for the term, and the final result returned as an answer.

Although this may seem to be rather complicated, it merely reproduces the method which would be used to perform the calculation by hand. If the procedure which is used to simplify the expressions which involve summations cannot simplify any such expression, it is returned unchanged. This of course means that part of the result which may be returned has not been averaged. If this is the case, however, it is fairly easy to spot which terms must then be simplified by hand since they will contain powers or products of summations optionally premultiplied by some numeric factor. Such terms are rare in the expressions formed in this particular calculation and although it is conceivable that more of such terms could be generated given a different lagrangian, it is apparent that this particular solution method is not ideal for the calculation of averaged lagrangians for general problems and so development work on this program was terminated. We shall discuss this further in the next section.

4.3.4. Towards Applying the Program in Nonlinear Optics.

Having constructed a program which would determine the averaged lagrangian for a given problem it was desired to construct a system whereby the averaged lagrangian for a lagrangian not known to the program before execution could be derived and the equations of motion arising from this averaged lagrangian calculated. For a given general lagrangian, there may be more than simply one variable to be optimised. For example, in the case of the lagrangian constructed by Small in his Ph.D. thesis [2] and given in a simplified form by Whitham in his book [1], there are two variables which are required to be optimised, the field and the polarization. Given this then, series expansions for each variable must be constructed in the manner given previously with

summations of harmonics at each level of the expansion. What was required was some method of generating a series of indexed names depending on some root name. For example, given the variable name *A*, the program would be required to generate the names A1 A1STAR and B together with A1(1), A1STAR(1), A2, A2STAR(1) and other such terms. Some method was required which would perform this function.

There seemed to be no obvious such method available within the algebraic mode although, by writing a short procedure in symbolic mode, it proved possible to generate such names. However, whenever the program was required to use them, the REDUCE package would often fail in some unusual manner. It seemed likely that the names being generated by this short procedure were not being properly recognised by the Algebraic processor. In order to solve this problem, it would have been necessary to write large quantities of SLISP code in order to modify the behaviour of the REDUCE kernel and this would be both difficult and likely to be unsatisfactory. An examination of the MAPLE programming language showed that it might provide a suitable environment for rewriting the program. Access to the necessary mathematical tools was readily available. For example, indexed functions were already defined. Writing procedures to carry out the types of manipulation required would not involve the modification of any of the MAPLE kernel. MAPLE has a kernel which is written in C and which defines the most basic operators and syntax of the language. Thereafter, if additional functions are required, they may be easily added using the MAPLE language itself and indeed this is how the bulk of the package is written. Rewriting the program using the MAPLE package was the route that was chosen.

A comparison of the two programs reveals that the expansions implemented by each program are different. The MAPLE package makes the summations involved concrete and terminates them after a given number of terms rather than keeping the summations in their compact form. It might be asked if the REDUCE program might not have been simplified by using this method rather than the one implemented. The answer to this is that this would have been the case. Indeed, with the benefit of hindsight, using this method to generate the averaged lagrangian for a general problem would give the averaged lagrangian very quickly when compared with the time taken to write the section of the MAPLE program which generates the averaged lagrangian. REDUCE's rule based approach considerably lessens the programming input involved. On the other hand, writing the section of the program which would generate the equations of motion would have been difficult since the symbolic mode of REDUCE would have had to have been used to perform the Frechêt derivatives. It is difficult to see how REDUCE could

have been used simply to perform the calculation of the equations of motion. A possibility which was not considered at the time was to use REDUCE to form the averaged lagrangian and then to convert the result for use in MAPLE to determine the equations of motion. This might have been a possible solution. It does however seem somewhat unsatisfactory given that two packages would be required together with some sort of translation program between them. The MAPLE program which was eventually implemented seems to be a better solution in view of the fact that it is self-contained.

References.

1. Whitham, G.B., *Linear and Nonlinear Vaves.* 1 ed. Pure and Applied Mathematics, 1974, New York, Chichester, Brisbane, Toronto, Singapore: John Wiley & Sons. 636 pp.

2. Small, R.D., *Nonlinear Dispersive Waves in Nonlinear Optics.* 1972, California Institute of Technology:

3. Allen, L. and J.H. Eberly, *Optical Resonance and Two-level Atoms.* Dover ed. 1987, Dover Publications, Inc. 233 pp.

4. Lorrain, P. and D. Corson, *Electromagnetic Fields and Waves.* 2nd ed. 1970, San Francisco: W. H. Freeman and Company. 706 pp.

5. Seliger, R.L. and G.B. Whitham, *Variational Principles in Continuum Mechanics.* Proc. Roy. Soc. A, 1968. **305**: p. 1-25.

6. Elsgolc, L.E., *Calculus of Variations.* International Series of Monographs on Pure and Applied Mathematics, ed. I.N. Sneddon, M. Stark, and S. Ulam. 1961, Oxford, London, New York, Paris: Pergamon Press. 178.

7. Loudon, R., *The Quantum Theory of Light.* 1973, Oxford University Press.

8. Cohen-Tannoudji, C., J. Dupont-Roc, and G. Grynberg, *Photons and Atoms. Introduction to Quantum Electrodynamics*. 1989, John Wiley and Sons. 468 pp.

9. Power, E.A. and S. Zienau, *Coulomb Gauge in Non-relatavistic Quantum Electrodynamics and the Shape of Spectral Lines*. Phil. Trans. Roy. Soc. A, 1959. **251**: p. 427-454.

10. Fried, Z., *Vector Potential Versus Field Intensity*. Phys. Rev. A, 1973. **8**(6): p. 2835-2844.

11. Arnold, J.M., *The Lagrangian Approach to Nonlinear Wave Propagation*, in *Nonlinear Waves in Solid-state Physics*, A.D. Boardman *et al.*, Editors. 1991, Plenum Press: p. 259-274.

12. Hearn, A.C., *REDUCE User's Manual (Version 3.3)*. 1987, Santa Monica: Rand Corporation.

13. Jeffrey, A. and T. Kawahara, *Asymptotic Methods in Nonlinear Wave Theory*. 1 ed. Applicable Mathematics Series, Pitman Advanced Publishing Program, 1982, Boston, London, Melbourne: Pitman Books Limited. 256.

Chapter 5

Le plus souvent, quand on pense sortir d'une mauvaise affaire, on s'enfonce encore plus avant.

Very often, when one thinks of a way to get out of a difficulty, one gets into still greater difficulties.

Jean de la Fontaine
La Vieille et les Deux Servantes.

5.1 Introduction.

We have now examined how to determine the necessary lagrangians; how Whitham's method is applied; the problems inherent in Whitham's method; and how the REDUCE computer algebra package may be used to calculate an averaged lagrangian. We must now consider how to implement the Whitham algorithm using the MAPLE computer algebra package. We will find that there are considerable differences between the programs written in the two languages. REDUCE's rule based approach led to a quick but inflexible solution. The solution of this problem using MAPLE is more flexible and elegant. However, its procedure based approach means that the programming input is necessarily considerably higher.

5.2 Overview of the Program.

As a first step in expounding the MAPLE solution we recap on the method that we are required to automate. It will be assumed that the lagrangian form for the equation under consideration is known and that all of the functions which will appear within the variational principle are known. From previous chapters, it will be noticed that the sequence of operations required can be summarised as follows.

1). Define sets of multiple scales.

2). Redefine the differential operators in terms of these multiple scales. Each differential operator becomes a series in terms of a small parameter.

3). Substitute for all of the dependent variables - the functions within the variational principle - with expansions of complex harmonics in terms of a small parameter.

4). Insert the expansions for dependent variables and derivative operators into the lagrangian and then collect powers of the small parameter.

5). Average each lagrangian using the definition of the averaged lagrangian to obtain a sequence of lagrangians at each order of the small parameter.

6). Use the Euler-Lagrange equation appropriate to each lagrangian to determine the equations of motion at each order.

The final result of these calculations is a sequence of equations of motion, one for each variable, which describe the behaviour of that variable at that order. Once these equations have been obtained, they may be used to shed light on the solution for the full equation or be solved by other methods. The main difficulty in calculating the equations is the large number of terms involved in the calculations and this is why it is necessary to automate the procedure.

We have seen that the implementation of the above procedure as a REDUCE program considerably speeded up the calculation of the averaged lagrangian even if it proved impossible to implement the determination of the equation of motion without writing SLISP procedures to modify the workings of the REDUCE kernel. A complete automation of the whole process was required and so it was decided to rewrite the REDUCE program in MAPLE. An attempt to modify the REDUCE kernel would have been very difficult and such an attempt would probably be not worthwhile.

The full program listings are given in the appendices. However, the program is very long and complex. For this reason, the remainder of this chapter is devoted to an explanation of the MAPLE program. In overview, the program can probably be summarised by the following diagram (Diagram 5.1).

The program begins by initialising the MAPLE variables *printlevel* and *prettyprint*. These variables affect the levels of output generated by MAPLE as opposed to those generated by the program. The "words used" messages which often appear when using MAPLE are reduced in number by using one of MAPLE's internal functions *words*. The next step is to read in the input file. This file, which is generally no more than a dozen or so lines long, contains all of the necessary input data for the program. It will contain:

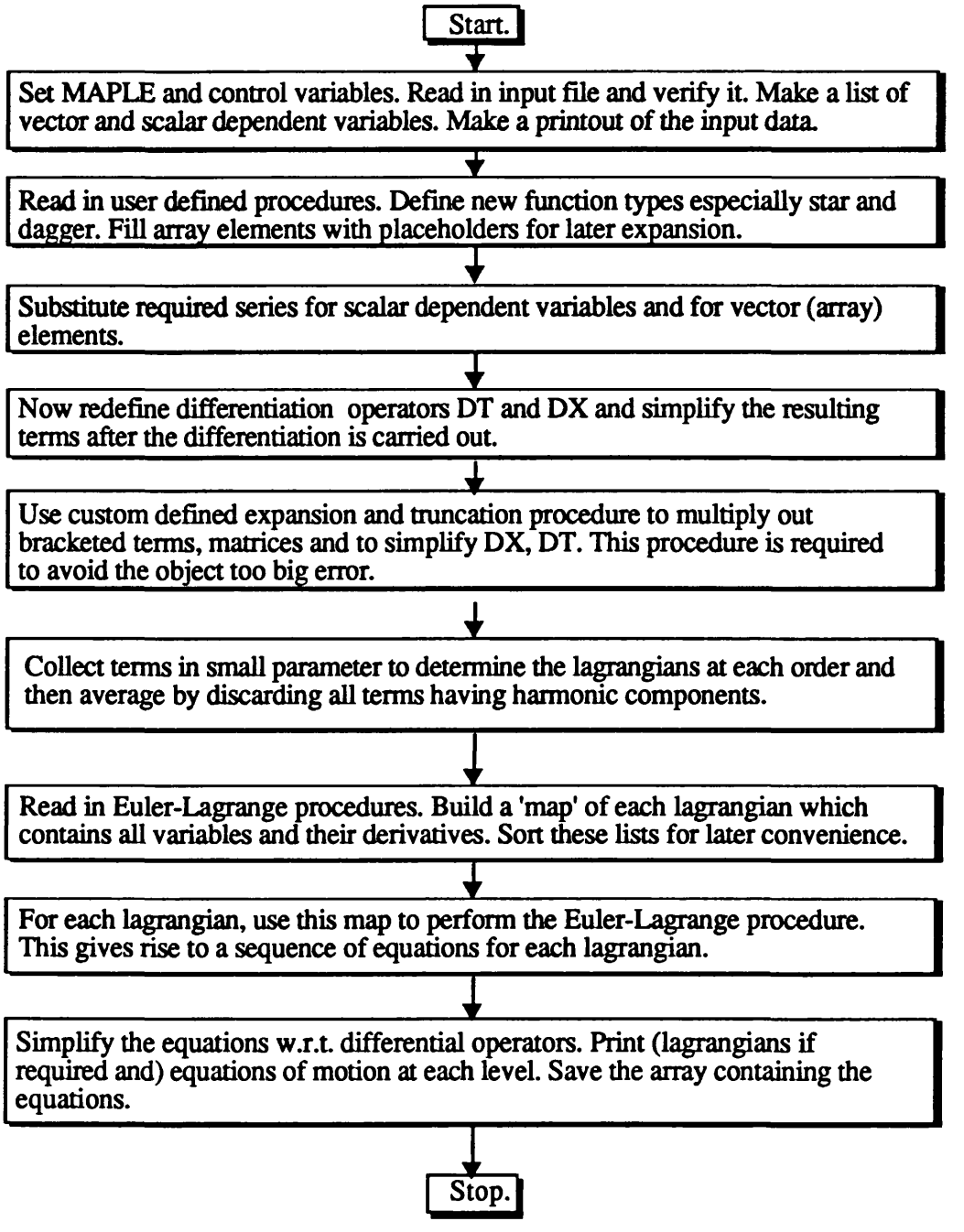


Diagram 5.1

1). the lagrangian itself;

2). the required order of expansion to which the program should carry out its calculations (*JMAX*);

3). a variable, *prinlvl*, set to an integer between zero and four which will affect the levels of output produced by the program as opposed to by MAPLE;

4). a variable, *errorswitch*, which affects whether or not the program will give long error messages for errors which concern the validation of the input file;

5). a set, *varset*, which contains the dependent variables (those whose equations of motion are to be found) within the lagrangian;

6). an optional set, *vset*, which contains a list of the vector dependent variables within the lagrangian;

7). a boolean variable, *hiharm*, which will determine whether the higher order harmonics are used in the expansion;

8). a boolean variable, *groundstate*, determining whether or not the groundstate of a vector variable will be treated as a perturbation from unity.

Having read in the input file, the program then performs some fairly elementary validation of the information. This has been written into the program so that the program can be used by persons other than its author. The checks performed are fairly simple. The data types of the control variables are checked to see that they are correct and the lagrangian is checked to see that it contains the variables which are mentioned as being present by virtue of their presence in the set *varset*. If vector quantities are included in the input file, the program also checks to see that at least one is also a dependent variable. Note that provided the checks made by the program are satisfied, any suitable lagrangian can be used: the program is quite general and does not restrict itself to any particular form of lagrangian, although assumptions have been made about the form of quantum lagrangians.

After printing out its input data, the program proceeds to perform the calculation. One point of note is that, since in MAPLE a variable cannot be assigned to a function of itself, it is necessary to create new names for each variable. The program does this by creating new lists of variables with an appended underbar (_) from the sets supplied in the input file. Its next step is to define some new data types such as *star*, *DX* or *DT* and to read in all of the user defined functions which will be required to handle them correctly and to perform all of the miscellaneous functions necessary in obtaining the averaged lagrangian. One of the most important of these is that which redefines the functions *dx* and *dt* which are implied in the input file in terms of a further undefined

function *Diff*. The trouble with allowing the program to calculate the differential series too early is that the variable θ is not yet included in the dependent variables since they have not yet been expanded. The full differentiation can be performed only after the series have been substituted and it is simple to do this by letting the undefined function *Diff* be equivalent to the MAPLE function *diff* which performs partial differentiation.

The first step in the calculation is to simplify the given lagrangian with respect to complex conjugation and hermitian conjugation. This is simply to insure that the lagrangian to be used is in its simplest form. The program then substitutes for the dependent variables series formed with the dummy variables and then reads in the file which permits the differentiation to be carried out.

This procedure with the redefinition of the differentiation operator is made necessary by MAPLE's lack of a DEPEND statement. In REDUCE it would have been possible to have said that the dependent variables were functions of θ and REDUCE would then not have discarded the terms in the summation. In MAPLE there is no transparent way to do this. The question of differentiation is worthy of further discussion since both the order in which the substitutions are made and the way in which subsequent simplifications are carried out require to be more fully explained. The definition of a differentiation type rather than a differentiation function leads to program behaviour which runs contrary to what is expected. This will be explained in a subsequent section.

The treatment of the hermitian transpose of a matrix causes further problems. If one tells MAPLE to simplify *dagger* of a matrix, MAPLE will simply apply *dagger* to each element of a matrix or array which is not what is intended. In order that MAPLE correctly transpose matrix objects when forming a hermitian conjugate, simplification of the *dagger* and *star* operations only is performed at this stage as well. This insures that matrix operations are carried out correctly, that the matrices and vectors are multiplied out in such a manner that a one element matrix can be set converted to an object a type other than matrix and that the eventual answer at this stage in the calculation is a (very) large polynomial in which all differentiations have been carried out and the implied differentiations inherent in the *DX* and *DT* types are applied only to the undefined indexed functions used to represent the unknown amplitude functions used in the expansions of the dependent functions within the lagrangian. A final application of the simplification procedures for *DT*, *DX* and *star* and the expanded lagrangian is ready to be multiplied out.

If one tells MAPLE to expand the lagrangian at this point, this results in one of three possible outcomes depending on the lagrangian and the operating system being used. Initially the program was run under the CMS operating system on an Amdahl 5980 at Manchester University. Under CMS, it is possible to define the amount of virtual memory available to a program although larger sizes are only available for batch jobs. The maximum size of virtual machine I was allowed (by special arrangement) for my virtual machine was 8 megabytes(M). Using VM/XA (an extended addressing scheme), one could request machine sizes of up to 64M for batch jobs. However, this memory area was not contiguous. The CMS operating system is loaded in between around 13M and 16M. This is fixed. In order to obtain a larger area of memory, it is necessary that software be loaded in above the operating system essentially leaving the lower 13M free. In order to do this, the software to be run in this upper area must be aware of the extended architecture addressing. MAPLE is not. Thus it cannot be run in XA mode and therefore it is limited to around 13M of memory.

An attempt to expand the lagrangian will inevitably result in the formation of very large intermediate expressions which will require a large amount of storage space. Thus, for small lagrangians and low orders of expansion, an application of MAPLE's expand function would work satisfactorily: an application of the function to a large or complicated lagrangian would result in a "Protection exception" error as MAPLE attempted to overwrite the operating system space.

To avoid this problem, work on the program was transferred to Lancaster University's Sequent Symmetry, a powerful Unix machine, where the limitations posed by the CMS operating system would no longer be a problem. Again, a naïve attempt at simplifying small expressions would work satisfactorily. This time, however, an attempt to simplify larger expressions resulted in MAPLE itself giving up. There is a limitation built in to the MAPLE kernel (although see later) which simply stated means that any one node of a MAPLE expression cannot have more than $2^{15}-1$ branches. This limitation is written into the MAPLE kernel itself and is a consequence of the data structure used to store MAPLE objects. Thus, if one is calculating a large expression, if at any stage more than this number of branches are needed from any node to describe the expression the calculation will fail. Even if the final expression itself would have had fewer than ≈32,000 terms, if this number is exceeded at any intermediate stage the calculation will halt.

One obvious way to get around this is to truncate the expressions being multiplied out so that terms higher than the maximum order of expansion are eliminated

before the calculation is carried out. One might think that a simple application of a truncation procedure would be all that would be required. However, this proved not to be the case. An application of the truncation procedure supplied by Dr. David Harper resulted only in the package failing oddly. The fault lies in the fact that the expression has not yet been expanded and so powers of the small parameters multiply polynomials themselves containing further powers of the small variable. To avoid this, it is necessary to expand the expression and then apply the truncation procedure which is precisely what one wishes to avoid.

To circumvent the "Object too big" error involved a major programming effort to write a procedure which would expand an object safely no matter what its eventual size. This procedure was called *smartexpand* and is the topic of a later section. That this procedure should be required is somewhat worrying. The procedure disposes of several terms in the process of its calculations and so it is very difficult to test to see whether the output from the procedure is what is expected. It would be likely that the most obvious source of bugs in this program would be this procedure. In future versions of the package, this limitation has been improved slightly (see later) and to some extent, therefore, the months spent writing this procedure represent wasted effort.

Nevertheless, having expanded and truncated the lagrangian, the next steps are to collect terms in the small parameter. Each polynomial at each order then represents the lagrangian for that order. Averaging is then carried out, not by integration but by the simple expedient of discarding all terms which have exponential factors within them. This has the same effect as integration for less cost in processing time. Finally, the evaluator *evalc* is called to simplify occurrences of powers of i and the averaged forms of the lagrangian are printed out and saved as a MAPLE internal file. This is the stage to which the previous REDUCE version of the program would proceed.

The MAPLE version of the program is able to continue since the facility exists for automatically creating a sequence of indexed names. It was this shortcoming in REDUCE which made it necessary to rewrite the program. The program's first step is to read in some additional procedures concerned with deriving the equations of motion.

Two of these concern the ordering of derivative terms. In order that the answer obtained by the program be unique, each derivative expression must have a unique ordering of derivative indices. The sort procedure contained in MAPLE allows one to supply a boolean procedure to judge whether or not two objects from the list of those being sorted are or are not to have their order reversed. This proves to be extremely

useful since a special procedure can then be implemented for dealing with the new *DX* and *DT* types. The other files directly concern the generation of the equations of motion.

The first step in determining the equation of motion is simply to make a set of all those dependent variables and their derivatives to be found within the lagrangian, each order of lagrangian being processed in turn. This is done to minimize the amount of work to be done by the procedure. Scalar and vector variables are handled separately in order to allow for separate processing to occur for each type of variable although there is no difference in the processing carried out. There is no need for the procedure to consider variables or derivatives which do not appear. The use of sets insures that there is no duplication of elements. Sets are determined describing the presence of the variables themselves, the presence of their *DX* type derivatives, the presence of their *DT* type derivatives and of the mixed *DX(DT(* derivatives. (The ordering of mixed derivatives is fixed and maintained by the simplification procedures.) Once these sets have been determined, they are converted to lists and sorted using yet another sorting procedure, this time to present the final results in an ordered list rather than in some arbitrary order.

Having determined these lists, the program can then go on to use them, by referring back to the original lagrangian from which they came, to create sets of the equations of motion. It does this by scanning the lagrangian for the presence of a dependent variable. Having found one, which is not itself part of a derivative operation such as *DX*, it performs a differentiation of the lagrangian with respect to the variable. This is not simply an application of the *diff* procedure since the object with respect to which the differentiation is being carried out is itself a compound object. Rather the differentiation is carried out by applying *diff* to "atomic" objects created using the MAPLE procedure *frontend*. This will form the basis of the equation of motion in most cases.

The procedure where the derivative operators *DX* or *DT* is involved is similar. Using the elements contained in the sets of variables having a derivative form in the lagrangian, the program uses the *frontend* procedure to form the derivative of the lagrangian with respect to the compound object. This is not the end of the procedure here since, for those objects which involve differentiation operators, a further differentiation must be carried out. This corresponds to the d/dt or d/dx operations that are carried out when finding the equations by hand. This differentiation is carried out by applying one of the *DX*, *DT* or *DX(DT(* operators in the usual way. Another point which must be considered is the sign of the terms generated in this manner. The sign

depends on the order of the derivative. This is handled correctly for each term by the procedure which finds the derivative. The final step, therefore, is simply to add together the terms produced by the differentiations and to store them in the arrays contained within sets for each order of the small parameter.

Having produced these expressions, a special procedure is required to print them out in a form which is recognisable as an equation of motion. This simply takes the expressions from the arrays and prints them out setting them equal to zero in order to turn them into equations. Finally the whole MAPLE session is saved in order that the equations can be further manipulated within MAPLE if this is required. It often happens that derivatives of the linear dispersion relation(s) can be made into factors which considerably simplify the final expressions. For the additional programming effort required, it was not considered to be a necessary addition to the program: such processing can be done under user control within MAPLE if required.

5.3 The Differentiation Procedures: Files *diffdtdx*, *simpdtdx* and *diftheta*.

An examination of the program listings given in the appendices show that a large part of the earlier section of the program simply concerns data validation and simple error checking. It is only after the input data has been checked and some user messages are printed that the program goes on to perform calculations. The first action that is performed is to convert the sets of vector variables, scalar variables and constants (probably predefined arrays) into lists so that the ordering of the variables is maintained. The next step is to define a new series of lists of variables which is the same as the original series of lists but for the fact that each variable in each list has an underbar (_) appended to it. This is because, in MAPLE, it is impossible to define a function of the form A=*f*(A) for some function *f* since this results in a recursive definition. An attempt to do so causes MAPLE to hang up as it circularly tries to redefine the name.

It is just after this point that the program reads in the user defined functions which it then uses to perform the calculation. The files which concern us here are called *diffdtdx*, *diftheta* and *simpdtdx*. It is the functions contained in these files which perform differentiation. In implementing the Whitham method, one is required to

redefine the differential operators $\partial/\partial x$ and $\partial/\partial t$ by replacing them with a series expansion in terms of the multiple scales in x and t. In the input file, the operations of differentiation with respect to x and t are represented by *dx* and *dt* respectively. What is not obvious from cursory examination of the program is that the action of reading in these files causes the redefinition of the differentiation operators *dx* and *dt* to take place, at least partially. In order to understand what is happening here, it is necessary to examine the *diffdtdx* file.

5.3.1 The *diffdtdx* and *diftheta* Files.

The *diffdtdx* file is quite short. It actually consists of the same definitions repeated twice, once for functions involving differentiation by x and once for those involving t. If we consider those involving t only say, we see that the first short procedure replaces the function *dt* with a sum of the differential of the argument of the *dt* function with respect to θ using a function *Diff* yet to be defined and another function *Dt*. The function *Dt* is then defined as a series which could be written

$$\sum_{j=1}^{JMAX} \varepsilon^j DT(f,j) \tag{5.1}$$

We see that this is effectively the same as the second part of the series defined earlier as

$$\begin{aligned}\frac{\partial}{\partial x} &\to k\frac{\partial}{\partial\theta} + \varepsilon\frac{\partial}{\partial x_1} + \varepsilon^2\frac{\partial}{\partial x_2} + \varepsilon^3\frac{\partial}{\partial x_3} + \ldots \\ \frac{\partial}{\partial t} &\to -\omega\frac{\partial}{\partial\theta} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2} + \varepsilon^3\frac{\partial}{\partial t_3} + \ldots\end{aligned} \tag{5.2}$$

where the index j included within the DT function in (5.1) is used to denote the particular scale with respect to which the differentiation is being performed. Included within the file *diffdtdx* are two procedures which are concerned with the differentiation of DX and DT type objects. These procedures are called *diff/DT* and *diff/DX* respectively and their effect is to append a further index to the arguments of the DX of DT function being differentiated. Their names show that they have been written to take advantage of the standard MAPLE differentiation function interface. That is, by appending the name of a function to the word *diff* (separated by a slash), the procedure

so named will be used whenever MAPLE is required to differentiate a function of this type.

For example, suppose we are required to differentiate the following expression

$$\frac{\partial}{\partial t_3}\left(\frac{\partial}{\partial t_1}(a\_[1,1])\right) \tag{5.3}$$

which can be represented as

$$DT(DT(a\_[1,1],1),3) \tag{5.4}$$

The result of carrying out the calculation will be

$$\frac{\partial^2}{\partial t_1 \partial t_3}(a\_[1,1]) \tag{5.5}$$

which is represented by

$$DT(a\_[1,1],1,3) \tag{5.6}$$

This, of course, implies that there must be some ordering of indices since the two possible forms of the example derivative must be equivalent. This ordering is performed later in the program.

Earlier in this section it was stated that the differentiations with respect to θ were performed in terms of a function *Diff* which had yet to be defined. The reason for this is that, if the differentiation were to be carried out at this point, several terms would be missed out of the expression. If the program is substituting for the expression *dt(f)*, where *f* is some subexpression, then the result of this substitution will be

$$-\omega * Diff(f,\theta) + \sum_{j=1}^{JMAX} \varepsilon^j DT(f,j) \tag{5.7}$$

if the *Diff* function is kept undefined. If we were to replace *Diff* with *diff* in the definition, the end result would be

$$\sum_{j=1}^{JMAX} \varepsilon^j DT(f,j) \tag{5.8}$$

where all of the terms which would have been generated by the differentiation with respect to θ have disappeared. This is, of course, because the function f is not known by MAPLE to be a function of the variable θ and so the partial differentiations performed by the *diff* procedure yield zero. The idea is then to substitute for the expressions within the differentiation operators with the series which replace these functions and then set the function *Diff* equal to the function *diff*. Once this substitution has been carried out, the dependence of the functions on the variable θ will be explicit and the differentiation will be carried out without error. This is the function of the file *diftheta*. It contains code which performs exactly such a substitution.

This sequence of actions- the partial redefinition of the differentiation function, the substitution for the differentiated function, the completion of the differentiation function's definition- poses the question of why the substitution of the variables is not performed first and the definition of the differentiation function not performed, in one stage, second. The answer to this is probably that it happened as a result of historical accident rather than as an act of conscious planning. It is likely that such a method would have simplified the coding slightly. At the time when the code was written, the equivalent REDUCE method was still in the author's mind. That there would have been much simplification is arguable however. Although the differentiation functions would have been easier to write, care would have to have been taken with the multiplication of (possibly hermitian or complex conjugates of) matrices within the functions to be multiplied together. It is likely that the same amount of effort would have been involved in either case. One possibility for future work is therefore to rewrite this section of code to see whether speed or efficiency improvements can be made by reversing the order in which these two operations are carried out.

5.3.2 The *simpdtdx* File: Simplification Procedures for the *DX* and *DT* types.

There remains one further set of procedures which deal with the *DX* and *DT* types namely those procedures contained in the file *simpDTDX*. The procedures dealing with the simplification of *DX* and *DT* types are contained in these files as the procedures *simplify/DX* and *simplify/DT* and again the standard MAPLE interface, this time for simplification procedures, is used. These routines are required because MAPLE does not know about the behaviour of the new types which have been defined in the *typedef* file. For example, the commutative properties of the functions *star* and *DT* are unknown

to the system. If MAPLE is required to perform a simplification of a user defined function, a simplification routine for the user defined function must exist or MAPLE will give up its simplification process and report the failure as a (non-fatal) error.

The simplification functions for the *DX* and *DT* types are not symmetric either. Although *DX(DT(f,1),2)* is formally equivalent to *DT(DX(f,2),1)*, MAPLE does not know this since the commutativity of the two operators is unknown to it and there is nothing to tell MAPLE that we require a canonical ordering of the terms. It is necessary to impose this ordering. Thus, whenever one of the simplification procedures meets a *DT(DX(f,x),y)* term it is changed into a *DX(DT(f,y),x)* term. Moreover, when an ordering has been imposed on the indices within the differentiation function, this will lead to a unique representation of any possible combination of differentiation operators.

We shall restrict the discussion to the *simplify/DT* procedure since, with the exception of the ordering of the derivative operators, the two procedures are nearly identical. The first simplification performed by the procedure is obvious enough but is a good example of how explicit one is required to be: $DT(0,x<,y,z,...>)=0$. The line following this is representative of a difference between the *expand* and *simplify* interfaces: if a user supplied procedure is used with *expand*, the function surrounding the argument is retained in the argument to the *expand* function; in the case of *simplify* however, the function surrounding the argument is removed. For example, if one is calculating *simplify/dummy(f)* the argument to *simplify/dummy* is f whereas if one is calculating *expand/dummy(f)* the argument to *expand/dummy* is *dummy(f)*. Another point of note is that the argument supplied need not necessarily be of type *DT*. It is therefore necessary to check for arguments to the function of other types.

Thus at first it might seem as if the workings of the simplification procedure are contrary to the rules of calculus. For example, if it were not known that the simplify procedure were looking for expressions to simplify rather than having an expression of the required type with the function prefixing it removed, then it might seem that the function of the first few lines of the program were to calculate products and powers in the following manner:

$$\begin{aligned} DT(a*b*c,1) &\rightarrow DT(a,1)*DT(b,1)*DT(c,1) \\ DT(p^n,1) &\rightarrow DT(p,1)^n \end{aligned} \tag{5.9}$$

This is plainly not competent mathematics. It might, however, be excused if it were known before hand that the functions a,b and c were likely to be functions of the

variables whose differentials were being found in at most one case. We need not concern ourselves with this matter further however.

The purpose of the first section of code is essentially to progress down the tree structure which represents the MAPLE object, cutting it up into finer and finer pieces, until a piece containing a *DT* function is detected. Once this happens, the series of alternatives to be found in the *elif* block is used. Each type of argument is considered in turn and the appropriate action for each is then carried out. The function of the procedure for *star* and *dagger* types is to take the *star* and *dagger* operators out of the *DT* function so that they can be handled separately by their own simplification procedures. For arrays, the *DT* procedure is mapped onto each element of the array and a similar mapping is carried out for sums of terms. For the case of a product, the product rule is implemented for any number of arguments and powers of functions are handled by the chain rule. There are exceptions to the way that powers are handled.The scaling variable ϵ is explicitly taken out of the remit of the differentiation operator *DT*. The case of powers of matrices or products of matrices is not simplified. This is because to do so would possibly be incorrect since the *dagger* operation might not yet have been applied. This feature was written more as a fail-safe feature than as a part of the calculating machinery.

Finally, the only other parts of the code that are required are those handling the differentiation of other *DT* objects and *DX* objects. If the simplification routine finds a *DT(DX(* then the order of the differentiations is reversed as explained previously and the simplification routine for *DX* objects is applied to the new object. If the procedure finds a *DT(DT(* within the data object, it combines the two sets of subsidiary arguments to the *DT* functions to produce a single *DT* function.

It will be noted here that the code uses the simplification routines for the DX and DT operators after the substitution of the summations for the dependent functions but before the full simplification of hermitian conjugates of matrices. Once the correct evaluation of the hermitian conjugate has been carried out, the matrices involved can then be multiplied together and (this is assumed) reduced to a scalar expression. Thus a second application of the simplification procedures after the simplification of hermitian conjugates and of matrix multiplication should result in the simplification of any expressions which were neglected in the first run through the code. This matter will be further discussed in the following section.

5.4 The Simplification and Expansion Procedures Involving the *star* and *dagger* Operations and the *expddson* Procedure.

In the same way that the differentiation procedures *DX* and *DT* required the procedures *simplify/DX* and *simplify/DT*, the procedures *star*, denoting complex conjugation, and *dagger*, denoting hermitian conjugation, also require simplification procedures. These procedures, *simplify/star* and *simplify/dagger* are implemented in a manner similar to the procedures pertaining to the *DX* and *DT* operators. In the case of the *star* and *dagger* operations, however, a further set of operations is required namely that concerning the expansion of the *star* and *dagger* operations. These are denoted *expand/star* and *expand/dagger*. In addition, there is a further procedure, *expddson*, which is used to handle the expansion and simplification of the expression after the simplification of hermitian conjugates and matrix expressions has been carried out. It performs a degree of simplification without invoking the *expand* operation on the expression. The use of the generic *expand* procedure on any usefully sized lagrangian would generally result in the failure of the MAPLE package with the error "Object too big". We now consider the simplification, expansion and *expddson* procedures in separate subsections.

5.4.1 The Simplification Procedures for the *star* and *dagger* operations.

In section 5.3.2, the operation of the simplification functions for the *DX* and *DT* operators was discussed. *DX* and *DT* are really new data types in just the same way that *star* and *dagger* are.We can therefore expect some degree of similarity between the two sets of simplification procedures. The comments of section 5.3.2 concerning the differences between the interfaces for the expand and simplify functions within MAPLE should also be noted. Since we are dealing with a set of simplify functions here, the procedure should be written to allow for the fact that the argument to the procedure may be any general function and not necessarily a function of the data type from which the procedure takes its name.

Since the functions of a hermitian conjugation operator and a complex conjugation operator are identical other than for matrix data types, there is an expected

degree of similarity between the two procedure codes. Taking the case of the *simplify/dagger* procedure first, we see that the fist section of the code is functionally equivalent to that in either the *simplify/DX* or *simplify/DT* procedures. Basically, the function of the code is to search through ever smaller sections of the argument expression seeking an sub-element which has type *dagger*. It does this by mapping the simplification procedure over sums or products or by considering the exponand within a power. Once it has found an expression involving the *dagger* operator, it uses the series of alternatives given in the second section of the code, one for each of the possible argument types to the *dagger* operator. Since each type has a procedure called *dagger<type>simp* associated with it, it is quite easy to see how the code functions.

The main problem with the *dagger* operator is the problem of non-commutativity. Strictly speaking, the definition of the hermitian conjugate of a product is

$$(A * B)^\dagger = B^\dagger * A^\dagger \tag{5.10}$$

When the objects forming the product are scalar, we can use the commutativity of scalar multiplication to rearrange the product so that we could write, for this case

$$(A * B)^\dagger = A^\dagger * B^\dagger \tag{5.11}$$

although we should bear in mind the fact that this rearrangement has been made. If the objects forming the product within the conjugation operation are matrices or vectors, however, this rearrangement is not possible since matrix multiplication is non-commutative.

All of the simplifications implemented in the *simplify/dagger* procedure assume that this rearrangement is possible. If the case of matrix multiplication does arise, the expression is left unsimplified. The only non-trivial simplifications which are performed regard the simple case

$$\left((A)^\dagger\right)^\dagger = A \tag{5.12}$$

and the hermitian conjugate of a complex exponential. This latter case uses the code from the *simplify/star* procedure. Further simplifications could be performed by appending further simplification procedures to the main routine. However, the code presented simplifies commonly used data satisfactorily: adding further simplification rules could be considered to come under the heading of further work.

In the case of the *simplify/star* procedure, similar arguments hold. The code structure is almost identical to that for the *simplify/dagger* procedure. The differences lie in the procedures which have been implemented for the simplification of the different types of operation within the *star* operator. This time the non-commutativity aspect of the complex conjugation operation has been taken into account. It does so by swapping the order of a non-commutative product of matrices and applying complex conjugation to each matrix. In the case of three matrices, the code relies on the MAPLE simplify procedure to map the simplification for two matrices suitably. The code also finds the complex conjugate of a complex exponential expression correctly.

5.4.2 The Expansion Procedures for the *star* and *dagger* operations.

As well as simplification procedures for the *star* and *dagger* operators, it was also necessary to implement expansion procedures for each. This was necessary because it was envisaged that MAPLE's *expand* function would be applied to the lagrangian before truncation and averaging. This proved to be unfeasible. However, the *expand* procedures were still necessary since it was vital that only scalar objects were passed into the truncation and averaging code. Forming such objects is discussed in the next section. In this section, however, we discuss the operation of the expansion procedures.

As in the case of the simplification procedures, the code for the expansion procedures are similar for the two types of operator, *star* and *dagger*. It will also be observed that this time each piece of code is appreciably shorter than in the case of the *simplify* operators. The case of the *dagger* operation is slightly more complicated since the hermitian conjugation operation must be applied correctly to objects of matrix or array type. We will consider the case of the *dagger* operator first.

In previous sections concerning the function of the simplification procedures, it has been noted that the operation of the interfaces to the simplification and expansion procedures differs. In the case of simplification procedures, the argument to the procedure need not be of the type associated with that procedure. For example, the argument to *simplify/star* need not be of type *star*. This also implies that the operation being simplified has not been stripped from the initial argument. To continue our example, the original argument to the simplify procedure need not be *star(X)* for some object *X*: intuitively we might expect the *star* operation to have been stripped on having the argument passed to the simplify procedure.

In the case of expansion procedures, the argument passed to the procedure must originally have had the same type as the expansion operation although this operation has been stripped from the argument in passing the argument. For example, if the argument to *expand/star* is X, then the object *star(X)* is to be found in the expression being expanded and the *star* function has been stripped in passing the expression to the *expand* procedure. This is not clearly explained anywhere in the reference manual [1]. This will, of course, affect the way in which expansion procedures are written. There is no need to seek subexpressions containing the required operation since the argument passed to the procedure is the argument of such an operation in the expression to be simplified.This is further explained in the table 5.1 which uses the *star* function as an example.

The operation of the *expand/dagger* procedure is fairly simple. Quite simply, if the argument passed to the *expand* procedure is a sum or a product, the procedure is mapped onto each of the arguments of the sum or product. If the object is a non-commutative product of arrays, the order of the multiplication is reversed and the *expand/dagger* operation applied to each of the terms. This is in accordance with the definition of a hermitian conjugate of a product (5.24). If the argument to the procedure is an array then the procedure uses a small routine called *dagarrexpd* which will correctly find the hermitian conjugate of an array using the star operation and a short routine called *ttrans*. (The *transpose* routine of the *linalg* library within MAPLE [1] is unsuitable.) If the argument is not one of these types then the *dagger* operation can be treated as if it were the *star* operation. Accordingly, other types of argument are passed on to the *expand/star* procedure for further processing.

The *expand/star* procedure is even simpler. In the case of the *dagger* operation, it was necessary to be careful when considering terms or expressions involving non-commutative products of arrays because of the need to apply a *transpose* operation

Function.	**Argument to Function in Expression.**	**Argument passed to code.**
simplify/star	*star(X)*	*star(X)*
simplify/star	*Y*	*Y*
expand/star	*star(Z)*	*Z*
expand/star	A	*expand* function not invoked.

Table 5.1.

The different actions of the *expand* and *simplify* procedures.

to the hermitian conjugate of an array. This meant that the *simplify* operation for *dagger* expressions was not as fully implemented as the *star* operation. This deficiency is corrected in the *expand* procedure. The *simplify/star* procedure is shorter because it need not implement many of the functions required by the *expand/dagger* procedure.

Again the bulk of the processing involves arrays. If the argument supplied involves an array, the procedure maps *star* onto each element of the array. If it involves a non-commutative product of arrays, the order of multiplication is reversed and *star* is applied to each of the arguments. If the type of the argument is *star* then this can only be because the expression originally involved the hermitian conjugate of a complex conjugate and the *expand/dagger* operation has replaced *dagger(star(* with *star(star(* in the course of its processing. The procedure endeavours to cover any possible case by reapplying *expand/dagger* to the *expand/star* operation. In any other case, the procedure simply returns again the *star* of its argument. This will cover the case where the complex conjugate of an indexed variable is to be found.

Having discussed the operation of the *expand* procedures, it will be noted from the code of the main program that there is no explicit call of the expansion procedures. As was explained earlier, this is because to apply the generic *expand* procedure to many of the lagrangian expressions would cause the "Object too big" error. It was to avoid this error in attempting to provide some degree of object expansion that the next procedure was implemented.

5.4.3 The *expddson* Procedure and its Implications.

The code contained in the *expddson* procedure was written to avoid calling the generic *expand* procedure. Some degree of expansion of the expressions generated was necessary, but provided the user defined types were expanded, later manipulations were expected to deal with the expansion of the expressions in a general sense. The purpose of the *expddson* procedure, therefore, is to expand all of the user defined types - *DT*, *DX*, *star*, *dagger* - whilst not expanding the expression by performing the distribution of products of expressions over sums of expressions. There is another reason why *expand* could not simply be applied to matrix expressions: the *expand* procedure would map *dagger* onto all matrix elements (without forming the transpose of the matrix) and this would be mathematically incorrect.

This has implications for the future operations to be performed on the expression. Whilst all of the user defined procedures defined above will have been

expanded and simplified, any bracketed terms in the expression will not have been multiplied out. Thus what may be expected is that the expression, after application of this procedure, will consist of a large number of sums of terms multiplied together as bracketed expressions. Once the multiplication of the brackets has been carried out, the remaining work will consist of factorisation and collection of similar terms and this is work that can be left to MAPLE itself: once these operations have been carried out, work can continue with the manipulations required for the averaged lagrangian method.

An examination of the code reveals its relative simplicity. Again, the first action of the procedure is to map itself over sums or products. Once these actions have been carried out, the next sections of the code apply to the user defined functions that have been implemented. The *star* and *dagger* operations are expanded and the *DX* and *DT* operations are simplified. This is where the *expand/dagger* and *expand/star* operations are invoked. The only other function of the *expddson* procedure again concerns non-commutative products. If a non-commutative product is encountered, the procedure insures that any *dagger* or *star* operations are applied correctly to any of the matrix types within the matrix multiplication expression. It does this by repeatedly calling itself on the subexpressions contained within the matrix multiplication expression. Then it multiplies the arrays together. It is this procedure, therefore, which is ultimately responsible for insuring that the expression which finally leaves this section of code is comprised only of scalar objects. There can be no vector expressions passed into the averaging section of the code.

5.5 The Summation Procedures and their Significance.

We have now discussed the simplification and expansion procedures necessary to obtain an expression comprising a number of sums of terms multiplied together to form the lagrangian. However, although the procedures which are used to form the substitutions for the dependent functions were mentioned in section 5.3.1., no explanation of their form or function has yet been made. It is therefore necessary to consider these procedures before we go on to consider the *smartexpand* procedure. It is necessary to examine what types of expression are likely to be formed before being passed to the simplification and expansion procedures we have just discussed. The significance of these expressions must also be considered.

The summation procedures are contained in the file called *sums*. There are four of these procedures:- *weeharmsum*, *bigharmsum*, *quantsum* and *bigquantsum*. Each of these procedures is used slightly differently but they are all very similar in structure. Those procedures having "*quant*" within their names are used in the expansion of quantum objects which are assumed to be vectors representing the occupation of the various quantum levels within the physical system. Thus for system having two quantized levels, two element vectors would be involved whereas for a three level system the vectors would have three elements and so on. When the user of the program is entering the input file, it is not necessary to specify the contents of these arrays. The program uses the size of the arrays to fill the elements with the sums constructed by one of the two quantized sums procedures. Thus the user must only supply the size of the arrays. The two other procedures are used in forming the series which are substituted for classical or scalar objects. These procedures substitute series based on the variable name for each occurrence of the scalar variable.

There are two procedures for each type- quantized and scalar. Those procedures having "*big*" in their names produce a larger expression than those which do not. It may be remembered from chapter 3 that, in the overview of the averaged lagrangian method, it was stated that in most cases only first harmonic terms need be considered since the effect of the higher harmonics was not generally felt until higher orders. A global variable *hiharm* determines whether expansions involving higher harmonics will be used in forming the sums which will be substituted for the dependent variables. A further global variable, *JMAX* is used to denote the level of expansion in terms of the small parameter ϵ to which the expansion should be carried out. The quantum procedures also require a further global variable which denotes whether or not the first element of vector objects is to be given a special significance. If the variable *groundstate* is set to true, the first element of a vector object is taken to represent the ground state of the physical system so that excited states can be considered a perturbation in which the upper levels are populated at the expense of the ground level. Thus, if the variable *groundstate* is set to true, the first element of all vector objects is set equal to unity minus a sum of terms rather than just a sum of terms.

Finally, in order to form these sums correctly, the summation procedures must have access to four further global variables, the lists *nlist*, *newnlist*, *klist*, and *newklist*. These contain the scalar dependent variables, their new names to be used in the substitution process, the vector variables and their new names.

5.5.1. The Summation Procedures for Classical (Scalar) Objects.

The procedures which are concerned with the expansion of classical objects are called *weeharmsum* and *bigharmsum*. Each of these procedures is quite short - about six or seven lines of code. It is important to be aware of their functions since they represent an important component of the averaged lagrangian method.

It will be recalled from the introductory section (5.1) that in the process of carrying out the calculation it is necessary to substitute for dependent variables with series expansions. The dependent variables are substituted for in terms of a series of the form

$$f = \sum_{n=0}^{\infty} \varepsilon^{n+1} f_n, \quad 0 < \varepsilon << 1 \tag{5.13}$$

where

$$f_0 = A\exp(i\theta) + A^*\exp(-i\theta) \tag{5.14}$$

and

$$f_n = \sum_{l=1}^{\infty} A_{l-1}^n \exp(il\theta) + A_{l-1}^{n\,*} \exp(-il\theta), \text{ for } n \neq 0 \tag{5.15}$$

as was previously explained in section 5.1.. It was also pointed out that, since higher harmonics did not usually play a part in the equations of motion until higher orders in the small parameter ε, that the higher harmonics could be ignored for many purposes with a useful saving in the amount of computing time and storage space required to run the program. This resulted in an expansion for higher orders of the form

$$f_n = A_0^n \exp(i\theta) + A_0^{n\,*} \exp(i\theta) \tag{5.16}$$

It is these two expressions (5.15, 5.16) which are implemented by the program as *bigharmsum* and *weeharmsum* respectively.

For example, suppose that the variable A is used in the input file to denote a classical variable. Then A will be present in the list *nlist* as a scalar dependent variable and the main program will have initialised the list *newnlist* to contain the name $A\_$ upon which the summation to replace the variable A will be based. Assume also that the

variable *JMAX* denoting level of expansion has been set to a small integer, 6 say. Then depending on the value of the boolean variable *hiharm* one of the two possible series above will be formed.

If the variable *hiharm* is set to true then the program will use the summation procedure *bigharmsum* and the series formed will be

$$A = \varepsilon\left(A\_[0,1]\exp(i\theta) + A\_^{*}[0,1]\exp(-i\theta)\right) + \sum_{j=1}^{6}\varepsilon^{j+1}\left(\sum_{k=1}^{12}\left(A\_[j,k]\exp(ik\theta) + A\_^{*}[j,k]\exp(-ik\theta)\right)\right) \tag{5.17}$$

However, if the variable hiharm is set to false, then the program will form a sum which does not involve the higher harmonics. Thus, to continue the example, the series formed will be

$$A = \sum_{j=0}^{6}\varepsilon^{j+1}\left(A\_[j,1]\exp(i\theta) + A\_^{*}[j,1]\exp(-i\theta)\right) \tag{5.18}$$

It is fairly obvious that the inclusion of higher order harmonics will drastically increase the number of terms being formed and will slow down the computation considerably.

At this point it should be pointed out that, in his exposition of the method [2], Kawahara points out that since in many cases the lagrangian is itself a potential function for the equation being modelled, it is natural to add a non-oscillating quantity to each function. In other words, the expansion of a function would have

$$f_0 = A\exp(i\theta) + A^{*}\exp(-i\theta) + B \tag{5.19}$$

where B is a function of the slow variables as the first order approximation and

$$f_n = \sum_{l=1}^{\infty} A_{l-1}^{n}\exp(il\theta) + A_{l-1}^{n\,*}\exp(-il\theta) + B_n, \text{ for } n \neq 0 \tag{5.20}$$

where the B_n are again functions of the slow variables for the higher order approximations. We see here that implementing this would be a relatively simple matter. It would merely be necessary to adjust the summation procedure accordingly.

5.5.2. The Summation Procedures for Quantum (Vector) Objects.

The case of quantized lagrangians is slightly more complicated than the case for purely classical ones. Not only must higher harmonics be included within the substitutions for each variable but, since the variables are themselves vector elements, each element must have an appropriate summation associated with it and further, the treatment of the first of these elements may need to differ from the treatment of the other elements depending on whether the summation is to be considered as a perturbation from unity or not.

For example, suppose we are dealing with a two level quantum system. Let us assume that the name of the vector describing the occupation probability of each level is *v*. In the input file for the summation, there will be a definition of the form

$$\text{v:= array(1..2,[]);} \tag{5.21}$$

This tells the code that the quantity *v* is a two dimensional array and if this quantity is included within the sets *varset* and *vset* in the input file it will be recognised as such and will be expanded by the program. The program will generate the symbol *v_* to represent the root of the name of all objects pertaining to *v* and then use this root in subsequent expansion. The lower of the two elements in the array will be used to represent the groundstate and will be designated *v_1* by the program whereas the upper element will be called *v_2*. Obviously, if there were more than two elements they would be numbered in sequence going up the array. So far this could be represented as

$$\text{v} = \text{array(1..2,[])} \rightarrow \text{v\_} \rightarrow \begin{bmatrix} v\_2 \\ v\_1 \end{bmatrix} \tag{5.22}$$

It is then necessary to substitute for the placeholders v_1 and v_2 with series just as in the case of scalar variables. There are, however, some subtle differences in the way in which the expansions are carried out. These do not merely concern the use of the variables *hiharm* and *groundstate* but in the way the harmonics themselves are built up. Depending on whether the level is given an odd or even number, the expansion will have the opposite sign applied to the exponential terms. Odd levels are arbitrarily given negative exponential harmonics whereas even levels are given positively signed harmonics. The harmonics themselves act at a frequency of half that assigned to scalar variables. This has been touched upon in chapter 4 and will be explained in chapter 6 but

it concerns the way in which the scalar variables are presumed to act in such a manner as to mimic the energy (and hence frequency) separation between the levels.

If we examine the code for the quantum summation procedures, that is for the procedures *quantsum* and *bigquantsum*, the differences between the summations for classical and quantum objects will become apparent. Firstly let us look at the code for the *quantsum* procedure. The first item of note is that the code uses the size of the array with which it is supplied as an argument as a means of determining the number of quantum levels which, of course, corresponds to the size of the array objects. It then uses the value of the boolean variable *groundstate* to see whether the first of the array elements should be treated as a perturbation from unity. If the variable is set to true, the code will generate a series of the following form for the groundstate element *v_1*

$$v\_1 \rightarrow 1 - \varepsilon * \sum_{j=0}^{6} \left(\varepsilon^j v\_1[j,1] * \exp(-i * \theta) \right) \tag{5.23}$$

If the variable is set to false, the code will generate the following summation

$$v\_1 \rightarrow 1 - \varepsilon * \sum_{j=0}^{6} \left(\varepsilon^j v\_1[j,1] * \exp(-i * \theta) \right) \tag{5.24}$$

where again we have assumed the variable *JMAX* is set to 6. A similar summation will be formed for the odd numbered levels within the array. The summations formed for the even numbered levels take the form

$$v\_2 \rightarrow \sum_{j=0}^{6} \left(\varepsilon^{j+1} v\_1[j,1] * \exp(i * \theta) \right) \tag{5.25}$$

The difference lies only in the sign of the exponent within the exponential terms.

Similarly to the case of the scalar variables, the procedure *bigquantsum* generates a series of terms which include higher harmonics and again the setting of the variable *hiharm* will affect how the first level term is expanded. The expansion will again involve only half integer products of the frequency θ used in the expansion for the scalar (classical) operators. If the variable *groundstate* is true, the summation formed by the *bigquantsum* procedure for the groundstate variable *v_1* is

$$v\_1 \rightarrow 1 - \varepsilon * \sum_{j=0}^{6} \left(\varepsilon^j \sum_{k=1}^{12} (v\_1[j,k] * \exp(-i(2k-1)\%)) \right) \tag{5.26}$$

whereas if the variable is false the summation produced will be

$$v\_1 \rightarrow \sum_{j=0}^{6} \left(\varepsilon^{j+1} \sum_{k=1}^{12} (v\_1[j,k] * \exp(-i(2k-1)\%)) \right) \tag{5.27}$$

and again we assume that *JMAX* is set to 6. For even numbered matrix elements, the summation used will be

$$v\_2 \rightarrow \sum_{j=0}^{6} \left(\varepsilon^{j+1} \sum_{k=1}^{12} (v\_1[j,k] * \exp(i(2k-1)\%)) \right) \tag{5.28}$$

The difference again lies in the sign of the exponent of the exponential term.

It should also be pointed out as usual that making changes to these summation procedures to accommodate alternative expansion schemes would be comparatively simple.

5.6 The *smartexpand* Procedure.

It was mentioned in the overview of the program in section 5.2 that one of the major problems in the implementation of the program was the fact that MAPLE will allow no more than ≈32,000 terms in any one expression or, to put it another way, that no object in MAPLE can have any more than ≈32,000 branches from any node. This is not strictly speaking true. This limitation applies only to version 4.2 of the package which was all that was available to the author during the period in which the program was written. Future versions of the program have since been released.

The author has since met one of the authors of the MAPLE package, Dr. Michael Monagan, and has been informed that newer versions of the MAPLE package have been modified in which this problem is somewhat ameliorated. The version of the MAPLE package which was used during most of the writing of the program was version 4.2. This version has a limitation of ≈32,000 terms. Version 4.3 has a limitation of ≈64,000

terms and version 5.0 a limitation of ≈128,000 terms. Further additions to the number of terms which are allowed are unlikely at present. There is also the additional argument that, for most users, 128,000 terms is sufficient and that to increase the size of MAPLE data objects in order to cater to the needs of those who wish to use larger objects means that the efficiency of the package would be impaired. There would be an overhead for the package in terms of the amount of memory used and the speed at which the package would run.

It has also since been pointed out by Dr. Tony Scott (ITAMP, Harvard) and by Dr. Michael Monagan (ETH Zurich), both of whom have been, or are, involved in development work on the MAPLE package itself, that some modifications to the programming method involving the use of the *collect* function in MAPLE might have avoided the use of the procedure which we will subsequently describe and would have resulted in a program which would have run more efficiently and faster. Modifications to the program to achieve this are an obvious part of the work which remains to be done on the program and this will be discussed at the end of the thesis under the topic of further work.

That there is a limitation to the number of terms which may be generated is still true even though that limit may have been raised. The problem of having too many terms in an expression is still one which will arise in work using the package and, although the solution which we are about to discuss is crude and direct and perhaps best avoided, the problem will still be encountered in the work of others and a discussion of the solution implemented is justified.

5.6.1 The *smartexpand* Procedure and its Component Parts.

The procedure which gave rise to the "object too big" error was the *expand* procedure. This is the procedure whose function it is to distribute products over sums. In other words, in order to multiply out all of the bracketed expressions, one applies the *expand* procedure. This invariably gives rise to a larger expression than its input expression although the size of the result is smaller, in general, than expressions which are generated within the function itself as a result of its computations. This effect of "intermediate result swell" is well known in computer algebra systems. If we assume that the expansion of the lagrangian expression is indeed necessary, and we have already pointed out that this may not be the case, then one must find some way of overcoming the limitation in the number of terms and this must be achieved in the user's software.

The obvious first thing to try is to truncate the size of the input expression using the *orderser* functions of Dr. David Harper. This was ineffective, however, since the *orderser* function expected an expression which was already simplified As a short term solution, the function could have been applied to the subexpressions in the lagrangian in a recursive manner. This would have been only a short term solution however, since, as the expression was expanded, higher order terms would again have been generated.

The fact that MAPLE expressions are tree structures would indicate that recursion would be likely to be necessary in any solution of the problem. Unfortunately, a MAPLE expression will have a tree structure in which the branches will be of unequal lengths and the branching ratio will be irregular. Implementing a procedure which uses recursion alone would be extremely difficult: the difficulty would lie not in truncation of the arguments to each operator but in encountering overflows at differing levels within the tree. Since the procedure which was required was envisaged as one which could successfully handle an expression of any desired size, the procedure which we now describe was the one finally decided upon.

Referring to the program listings given later in the thesis, one will note that the procedure *smartexpand* is very short. In fact the overall structure of the smartexpand procedure is as shown in the following diagram. As can be seen, the *smartexpand* procedure really comprises a number of smaller procedures. The procedure by which the whole is named simply acts as a harness for a further two procedures, *maptree* and *choptree*, which are themselves only harnesses for two further procedures *domaptree* and *dochoptree*. The *dochoptree* procedure uses three further procedures, *dochopplus*, *dochoptimes* and *dochoppow*, to carry out the necessary simplifications.

The procedure works as follows. After being invoked with the function to be simplified as its argument, the highest level procedure simply checks to see if it has been used before by checking for the presence of its data structures. If the procedure has been used previously, then the program will stop with an error message. This was implemented in order to prevent the use of the procedure more than once during the program. This restriction is probably unnecessary but was included in case the program executed loops involving this procedure during development. The highest level procedure also checks to see that the names used by its data structure have not been previously defined elsewhere in the program. In order that substantial amounts of data can be passed between the parts of this procedure, it was decided to make the data structure a global entity. Thus this error checking avoids what would be an unusual error.

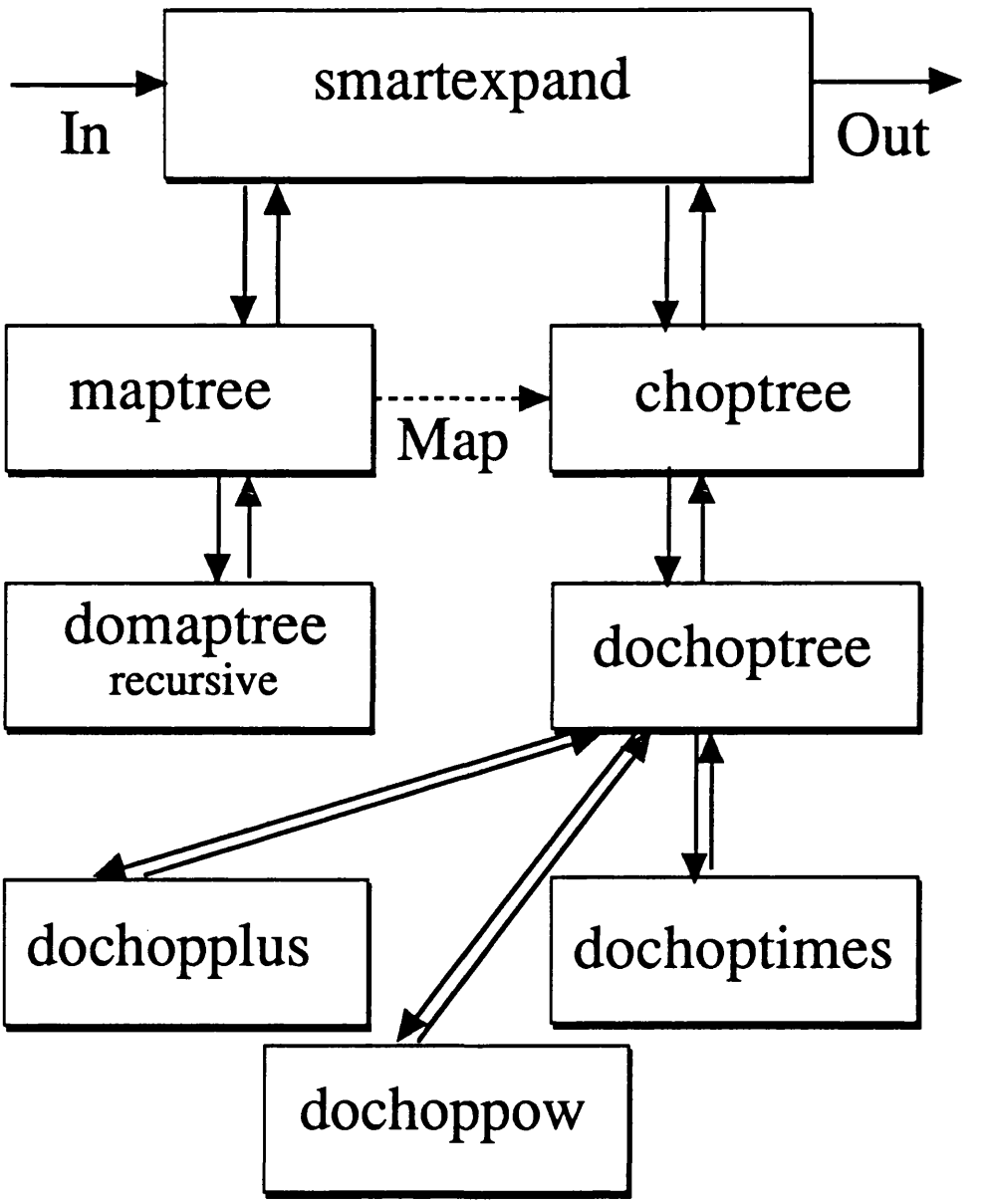


Diagram 5.2

Having performed this simple error checking, the procedure *smartexpand* invokes two further procedures in turn - namely *maptree*, which is given the simplifying function as an argument, and *choptree*, which acts on the data structures

created for it. The final function of the procedure is to examine the results of the simplifications performed on the data structures created by the *maptree* procedure and simplified by the *choptree* procedure. The function returns the simplified argument if the expression could be simplified, in its final form, within the ≈32,000 term limit. Otherwise, the function prints a message which informs the user that the result of the simplification could not be contained within one MAPLE object and reporting on the location of the results which have been stored in one of the global data structures mentioned previously.

As can be guessed from their names, the functions of the two procedures at the next level are to map the MAPLE data structure, that is fill the global arrays mentioned with sufficient data to completely describe the MAPLE expression, and to perform the required simplification by constructing the smallest possible expanded expression at each subsequent level of the tree structure.

5.6.2 The *maptree* Procedure.

The first of the two procedures, *maptree*, has the function of constructing a map of the MAPLE expression. However, the procedure *maptree* does not actually perform the mapping. Instead the procedure acts as a harness for the mapping procedure *domaptree* which actually performs the function required. The *maptree* procedure's function is to check to see if the global variable names required are available and to initialise the variables which are required for the *domaptree* procedure which is recursive. Obviously it would be counterproductive to try to write a recursive procedure which had a section of code which served to initialise it before use. It is much simpler to use an initialising harness as has been implemented. The data structures which are created are given in the table 5.2. They comprise three tables: *totnodes*, *typeinfo*, and *oprands*. The first of these will contain the total number of nodes at each level of the tree; the second will contain elements which describe the type of each node, each row of the array being used to hold the descriptions of one of the levels in the tree structure; the final table holds the operands of the nodes at that level which are already simplified. Thus each branch tip will result in the storage of operands in the *oprands* table. All three of the objects have been made tables. The *totnodes* table has been created as a sparse table to allow the maximum depth of the tree structure to be determined by the *choptree* procedure.

Name	Type	Function
totnodes	sparse table	Holds the total number of nodes at a given level in the data structure.
typeinfo	table	Holds a description of the type and the number of branches of each node at each level.
oprands	table	Holds the operands for any terminating node.

Table 5.2.

The global data structures used in the *smartexpand* procedure.

The *domaptree* code is given separately in appendix 4. It is relatively short . It functions as follows. It determines how many nodes are present at that level of the tree and saves it in the *totnodes* table. Then for each node in turn at that level it determines whether or not the node includes an addition operator anywhere within it. If it does not there is no point in continuing and the node is marked as simplified, its type and operands are saved in the relevant tables. If the node is one of addition, multiplication or exponentiation, the procedure notes the type of the node in the *typeinfo* table and then recursively descends the tree from the node it is currently considering. Finally, if the procedure is of any other type the procedure marks the node as simplified and saves its type and operands. This step, and the condition which seeks addition operators, insure that the recursion will end and that the next branch along from the node above can then be explored. This section of the code is probably fairly efficient and is subsequently implemented (in a modified form) when the structure of the expanded expression is again required when determining the equations of motion. The next figure shows a simple diagram of a sample expression showing the path taken by the procedure and table 5.3 shows the results which would be stored by the procedure when given the tree shown in the diagram as an argument.

Note the special significance given to the summation operator by the maptree code. It is the presence of the summation operator which primarily determines whether or not the code will descend any particular branch of a tree. If a branch, which requires to be mapped, does not contain a summation operator, the code will mark that branch as being simplified and will not continue further. This explains the differing behaviour of the code as shown in diagram 5.3. The term a*b*c does not contain, or arise from, a summation operator and so the code does not descend that branch. On the other hand, the expression a+b+c does arise from a summation operator and so the code investigates each term even though they are all atomic objects.

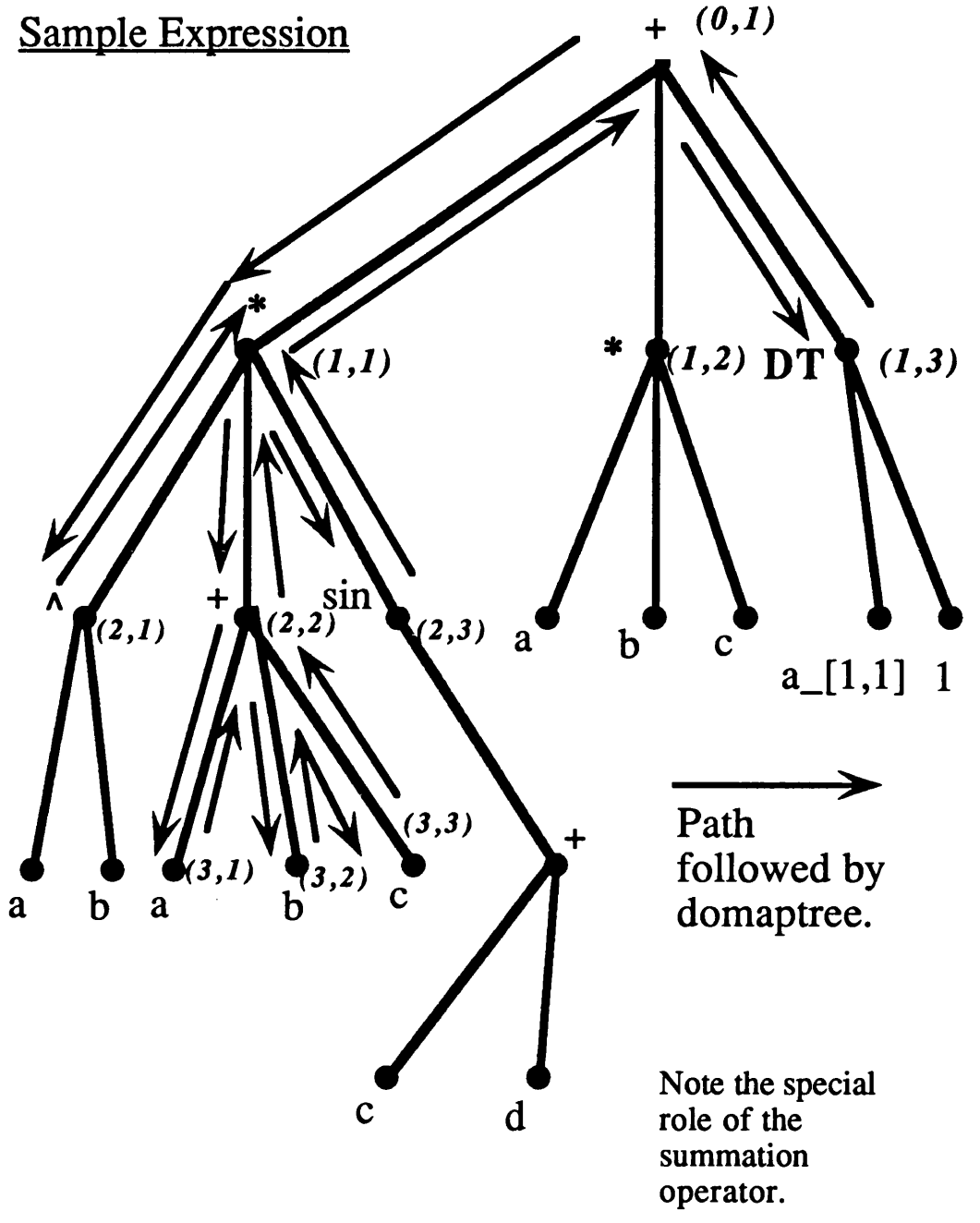
The data which is generated really is a form of map of the MAPLE expression and it is this map which is used to simplify the expression. Once the map has been generated, the structure of the expression is known and it then becomes possible to think of the tree structure in a different way. Since the number, type, location and arguments of each node are known, it becomes possible to think of the tree structure more as an array with a variable number of elements at each level. To put it another way, instead of thinking of a MAPLE structure as a traditional tree structure, think of the tree being "knocked sideways" in such a way that all of its branches grow to the right. Yet another way of thinking about the same thing is, instead of thinking of a MAPLE expression as a tree structure which has a vertical depth and a varying number of branches at each node, to think of the expression more as a stack of arrays which are left aligned and have a definite extent horizontally. This is shown in diagram 5.4.

Now, if one has obtained an expression of definite and known size, then the number of ways that one can deal with the simplification of that expression increases. Although the problem is far from being solved, it is easier to see that a solution which will avoid the "object too big" error is possible. Essentially what we will now proceed to do is to simplify the data structure we have just formed by treating it as a large array. By working upwards from the bottom of the array, we will simulate pruning the tree from the bottom. Any particularly insoluble branches can be left at the bottom of the tree as it shrinks to be combined in the answer at the final highest level. This is the function of the *choptree* and *dochoptree* procedures.

Table Name	Contents
totnodes	table(sparse,[(0)=1,(1)=3,(2)=3,(3)=3])
typeinfo	table([(1,2)=[S,0],(0,1)=[+,3],(3,2)=[S,0],(2,1)=[S,0], (2,2)=[+,3],(1,1)=[*,3},(2,3)=[S,0],(1,3)=[S,0](3,1)=[S,0], (3,3)=[S,0]])
oprands	TABLE([(1,2)=a*b*c,(3,2)=b,(2,1)=a^b,(2,3)=sin(c+d), (1,3)=DT(a_[1,1],1),(3,1)=a,(3,3)=c])

Table 5.2

Data structures generated by the maptree procedure for the sample input given in diagram 5.3.



Full Expression is:

$$a^b(a+b+c)\sin(c+d)+abc+DT(a\_[1,1],1)$$

Diagram 5.3

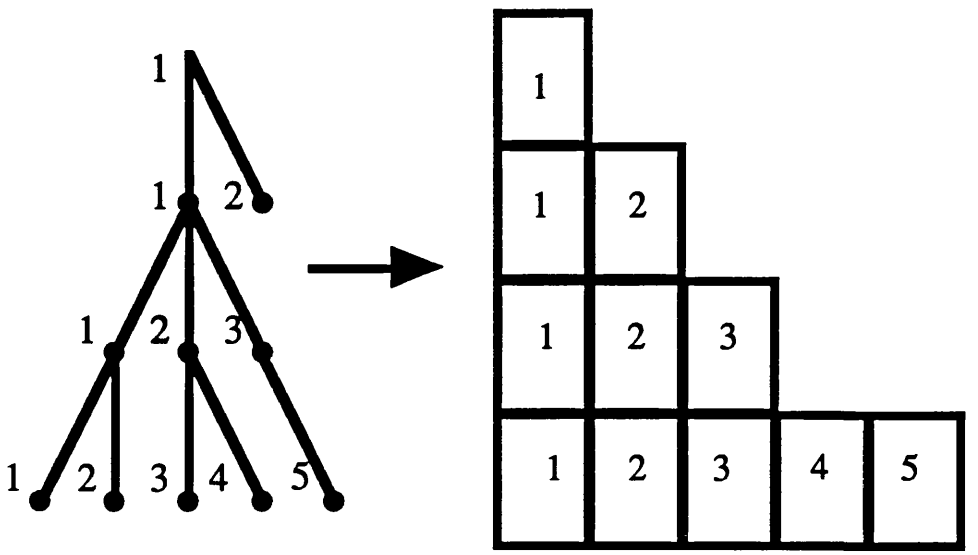


Figure 5.4

5.6.3 The *choptree* Procedure.

The function of the *choptree* procedure is rather more than simply a harness for the *dochop* procedure. Its function is to check that the *domaptree* procedure has supplied a map of the MAPLE expression, to set some local variables and to determine the height of the tree by using the *totnodes* table. The main local variables used by the procedure are called *tlevel* and *tnode*. They are used as counters to describe the current level and node of the tree under investigation. Once the size of the tree has been determined, the procedure uses two nested *do* loops to simplify the tree working from right to left, bottom to top. Since the size of the structure is not known in advance, the program uses MAPLE's *break* facility to jump out of the loops. After completing a traversal of each horizontal row, the procedure checks to see that the new type of each operand at that level is that of "simplified" and if this is not the case, this means that simplification within the ≈32000 term limit could not be achieved at that level.

The actual simplification itself is performed by calling the procedure *dochop* with the level and position of the node at that level as arguments. *dochop* uses the information contained in the *typeinfo* table to determine whether the node is one of addition, multiplication or exponentiation and uses the appropriate simplification procedure to simplify the node. The *dochop* procedure is accordingly rather short, the bulk of the work being done by the *dochoptimes*, *dochoppow* and *dochopplus* procedures.

Each of these three procedures takes as argument the level and position of the node measured from the top and from the left hand edge of the tree structure and also the number of operands possessed by the node. Before each procedure starts work, it must determine how far from the left hand edge of the structure the arguments of the node begin in the next level down. If a previous simplification has failed, then arguments pertaining to the node being simplified will be found two levels below the node. To complicate matters further, if two simplifications have failed, it cannot then be assumed that the arguments two levels down still start or end at known positions and the extent of the arguments must be calculated for each of the two levels. Whilst making these calculations, the procedure is also making an order of magnitude estimate of the final size of the answer and allotting storage space for intermediate results in table elements. The first step in the calculation of the simplified result involves truncating the arguments using the *orderser* procedure mentioned earlier and then copying the truncated operands of the node into a table created locally within the procedure for that purpose. This table has two levels, one corresponding to each of the two possible levels from which the arguments of the expression might have been taken (except in the case of *dochopplus* where only one is used). What happens then is dependent on the type of calculation to be performed. Each of the three possible operations involves a different algorithm. The three algorithms are closely related, especially in the cases of multiplication and exponentiation, but, even though large parts of the code were shared between the procedures, it proved to be simpler to create separate procedures to deal with each type and to write them out in full than to use subprocedures. This might also be an area in which future tidying work might be done.

5.6.3.1 The *dochopplus* Procedure.

The case of *dochopplus* is somewhat simpler than the other two. Since any previous overflows will have type "plus" it makes sense to copy all of the arguments from both levels into a table with only one level. The operands can then be added by

combining different table entries. This combination is not simply an addition of two elements from the table since this could cause an overflow. Instead, the procedure *dochopplus* calculates in advance the size of the result of adding the two elements together. If the result will not cause an overflow then simple addition is used. If an overflow is possible, then the two expressions are added on a term by term basis and the result checked after each addition to see whether or not an overflow is likely to occur at the next addition. The point is that if the expression already contains a similar term, the result of the addition will be a change in the size of a coefficient rather than the creation of a new term. Thus the actual size of the result will tend to be smaller than the calculated size of the result. If it is found that a further addition is likely to cause an overflow, then a new table element is started and further additions are stored in this element rather than the previous one. This process continues until all of the elements have been summed. This might be improved somewhat in that subexpressions guaranteed not to cause overflow could be added rather than individual terms. For example, the procedure could calculate the largest expressions guaranteed not to cause an overflow and repeatedly add such expressions to the original series rather than adding the expressions together term by term.

Once this occurs, the data table must be modified to reflect the results of the calculation. This further means that space will have to be allotted for the result in the data table. This is done by shifting the operands of the other nodes to the right of the node under consideration to allow for the size of the new object. This object may be smaller or larger than the original and it may also change its type. For example, if the sum has been successfully simplified within the operands limit then the type of the node will change from "plus" to "simplified". A large part of the remainder of the procedure thus consists of the operations required in order to correct the contents of the *oprands* and *typeinfo* tables. The *totnodes* table is not updated nor is the level containing arguments resulting from previous overflows tidied up.

5.6.3.2 The *dochoptimes* and *dochoppow* Procedures.

The *dochoptimes* procedure contains code which differs only in small details from that in the *dochopplus* code to determine the initial arguments, their size and location. However, in the case of both *dochoptimes* and *dochoppow*, three tables are used to hold the intermediate results of the calculations. The truncated arguments are copied into a holding table (having two levels as described previously) and the other two

tables are used to hold the results of the multiplication of the previous factor in the multiplication and as a work space to hold the result of the current multiplication. This is shown in diagram 5.5.

As can be seen from the diagram, each factor of the multiplication becomes an element of *table1* and *table2* is initially set equal to unity. *table2* is multiplied by the contents of the first element of *table1* (step 1) and the results then copied into *table3* (step 2). The element of *table3* is then multiplied by the expression forming the next factor of the multiplicative expression as stored in table1, each term of the factor being multiplied individually and the results again being stored in table2 (step 3). As each multiplication takes place, the procedure checks to see if the expression is getting too large and if so it creates another element and then continues the multiplication between the two elements (step 4). Once the next factor has been multiplied, the results are then copied out of table2 into table3 (step 5) and the process then repeats until all of the factors have been multiplied out (step 6).

The final result is held in a table as in the *dochopplus* procedure and code similar to that used in *dochopplus* is used to tidy up the data tables *typeinfo* and *oprands*. Essentially the same thing happens in the *dochoppow* procedure. The only difference in the *dochoppow* procedure lies in the fact that, since exponentiation is involved, the table elements are filled with copies of the factor whose power is to be found rather than the individual factors of a product.

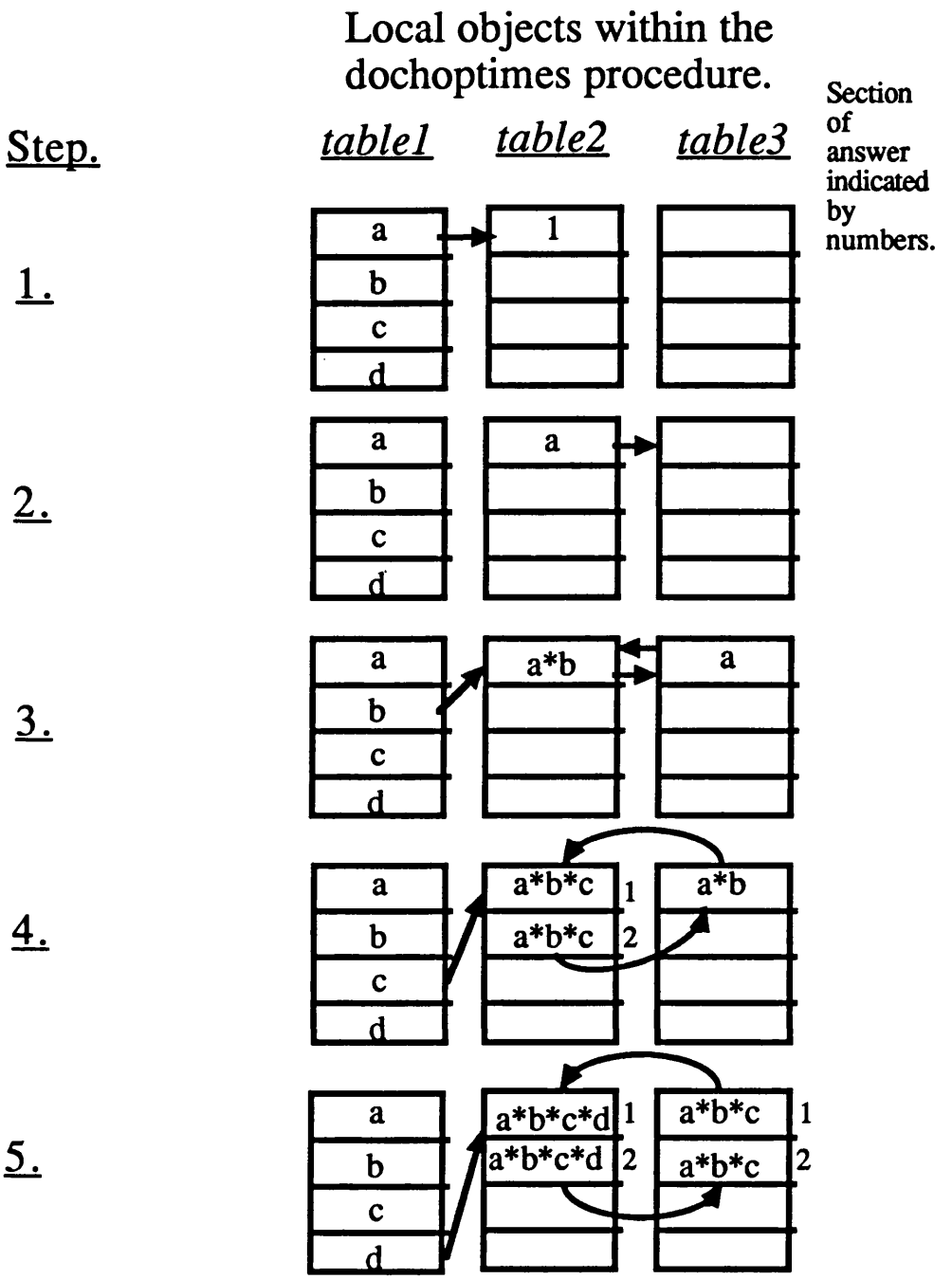

Local objects within the dochoptimes procedure.
Step.
table1
table2
table3
Section of answer indicated by numbers.
1.
2.
3.
4.
5.
6.
etc. Process continues until complete.
a
b
c
d
1
a
a*b
a
a*b*c
a*b*c
a*b
a*b*c*d
a*b*c*d
a*b*c
a*b*c

Diagram 5.5

5.7 The Averaging and Truncation Procedures

Extensive use is made of a truncation procedure, called *seriestrunc*, in truncating the expressions read into the *dochop* subprocedures and in truncating the final result which emerges form the *smartexpand* procedure. Once the full lagrangian has been evaluated, the MAPLE *collect* function is used to gather together all terms having the same power of the small parameter ϵ and then each series of terms at each order is removed from the expression and placed within an table element, one element for each power of the small parameter. Once this has been carried out, these expressions, the lagrangians at each order of expansion, are then passed on to the averaging procedure where the non-stationary terms are removed by simulated integration. We now go on to describe the two processes of truncation and averaging. The description of the truncation procedure is brief: the procedure was contributed by Dr. David Harper and is therefore not original work.

5.7.1. The Truncation Procedure *seriestrunc*[1]

MAPLE has its own series manipulation package [1] and this library of functions does indeed have its own truncation routines. Unfortunately, the machinery used for the manipulation of these series is somewhat clumsy for the general purpose which we require. It is also likely to be slower. The author was therefore advised that the library routines would be likely to be unsuitable. The series truncation routines supplied by Dr. Harper were therefore duly incorporated into the code especially within the expansion procedure *smartexpand*. The main procedure is called *seriestrunc* and it uses some short utility routines which are also present in the file. In order to invoke the procedure, one uses the calling statement

seriestrunc(f, [olist], maxorder)

where *f* is the function to be truncated, *olist* is a list of equality statements giving each small parameter a numeric weighting, and *maxorder* is the order at which the series will be truncated. The *seriestrunc* procedure expects that the argument *f* is a series which has been expanded so that there are no series within brackets to be multiplied together. If this is not the case, the procedure will not function (at all). Within the program, the procedure is utilised in the form

[1]The procedure was supplied and written by Dr.David Harper, currently of the Astronomy Unit, School of Mathematical Sciences, Queen Mary and Westfield College, University of London.

seriestrunc(L, [eps=1], JMAX)

where L is the lagrangian, *eps* is the small parameter ε, and *JMAX* is the maximum order of expansion supplied in the input file and present as a global variable.

If it were desired to allow the derivative operators to have one small parameter in their expansion and the functions another, then the relative weights that could be given to each could be adjusted using the orderlist, *[olist]*. Then the summation procedures could be relatively simply adjusted and the derivative operators given a different expansion parameter. Such an adaptation of the program would not take long and might be useful under certain circumstances. This is something which could be added to the program at a later date. A further modification which might be made is to let the scalar variables have one expansion parameter and the vector variables another. Again this would be relatively simple to implement and it could prove useful in that the scale on which quantum features might become apparent might be altogether different from those where classical features were to be found.

5.7.2 The Averaging Procedure.

It will be recalled that one of the main reasons for using Whitham's averaged lagrangian method is that the use of averaging gives results concerning the behaviour of the envelope of the carrier wave rather than giving results containing largely extraneous information concerning the behaviour of the carrier itself. The definition of the averaged lagrangian, as obtained from the expanded version of the lagrangian, is

$$\overline{L} = \frac{1}{2\pi}\int_0^{2\pi} L d\theta \tag{5.29}$$

The net effect of this integration is to remove any terms containing harmonic factors. In other words, any term which contains an exponential function is removed by the procedure.

To carry out a true integration of the lagrangian's thousands of terms would be grossly inefficient given that the much simpler expedient of setting exponential terms to zero exists. This is the course of action pursued by the code. The procedure simply maps itself over all terms within the lagrangian expression. It examines each term to see if it contains an exponential function: if it does, the term is set equal to zero; if it does not the term is retained unchanged. This should be contrasted with the procedure of the REDUCE program. Obtaining the full form of the lagrangian prior to averaging was

comparatively easy in REDUCE. The work of the coding had to be done in writing the averaging procedure. In MAPLE the opposite is true: obtaining the lagrangian's expanded form is difficult; it is the averaging which is easy.

5.8 Determining the Equations of Motion.

Having obtained the averaged lagrangian, it is then necessary to use this lagrangian to determine the equations of motion. These equations will describe the behaviour of the envelope of the carrier rather than the behaviour of the carrier itself since this is the function of the averaging procedure. Determining the equations themselves is, however, not as straightforward as might be supposed. Employing user defined differentiation types actually simplifies matters: using the true but unevaluated differentiation operator causes considerable difficulty when one wishes to implement an Euler-Lagrange procedure as other MAPLE users have observed in User Group correspondence. Nevertheless, a considerable amount of processing is required to extract the desired equations of motion from the averaged lagrangian. We now go on to describe this processing. That is, we describe how the Euler-Lagrange Procedure is implemented within the MAPLE program.

5.8.1. Introductory Overview.

The procedure used to determine the equations of motion for the averaged lagrangians exactly mirrors that used when determining the equations by hand. The fact that the equations have been averaged and that the lagrangian is not strictly a lagrangian but a lagrangian density makes no difference to the algorithm employed.

Suppose that we wish to determine the equations of motion of the example lagrangian

$$L = au_t^2 + bu_x^2 + u_{xxt} + u_{xt}^3 + v_t + cv_x + duv \tag{5.30}$$

If it were required that we carry out this calculation by hand, our first action would be to scan the lagrangian for the presence of the dependent variables. In the example above, the variables which we are required to deal with are u and v. The next action performed is to find all terms which contain functions of these variables and to collect them together

in some way as perhaps a list or set. Thus for the example above, the sets for u and v would be, respectively

$$\left\{au_t^2, bu_x^2, u_{xxt}, u_{xt}^3, duv\right\} \tag{5.31}$$

and

$$\left\{v_t, cv_x, duv\right\} \tag{5.32}$$

The next requirement is to construct the appropriate Euler-Lagrange or Ostrogradski equation [3] for each of these two sets. In order to do this, we scan the lagrangian for the presence of derivatives of the variables as well as the variables themselves. This will result in two further sets and these are given for the example as

$$\left\{u, u_t, u_x, u_{xt}, u_{xxt}\right\} \tag{5.33}$$

and

$$\left\{v, v_t, v_x\right\} \tag{5.34}$$

Having obtained these sets, the construction of the Euler-Lagrange equations is relatively simple. One will be required to form a special derivative of the lagrangian with respect to the variables in the sets (5.33) and (5.34) treating each derivative as if it were itself a variable. Once these derivatives are obtained, we then differentiate them according to the order and number of derivatives in terms of the independent variables x and t. In other words, firstly, we replace each of the members of these two sets by placeholders and then, continuing as if they were true functions, we find derivatives of these placeholders with respect to the x and t derivatives involved in the original terms.. In terms of the example, we form two set of replacements

$$\left\{u = u, u_t = p, u_x = q, u_{xt} = r, u_{xxt} = s\right\} \tag{5.35}$$

and

$$\left\{v = v, v_t = j, v_x = k\right\} \tag{5.36}$$

for each of the two sets above so that the lagrangian will then look like

$$L = ap^2 + bq^2 + s + r^3 + j + k + duv \tag{5.37}$$

The Euler-Lagrange equations will involve derivatives of the lagrangian with respect to these placeholders but modified by derivative operators involving x and t which for the moment we will represent as $D_1, D_2, \ldots$ operators. Thus the two Euler-Lagrange equations will be given as

$$\frac{\partial L}{\partial u} + D_1\left(\frac{\partial L}{\partial p}\right) + D_2\left(\frac{\partial L}{\partial q}\right) + D_3\left(\frac{\partial L}{\partial r}\right) + D_4\left(\frac{\partial L}{\partial s}\right) = 0 \tag{5.38}$$

and

$$\frac{\partial L}{\partial v} + D_5\left(\frac{\partial L}{\partial j}\right) + D_6\left(\frac{\partial L}{\partial k}\right) = 0 \tag{5.39}$$

We now need to determine each of the $D_1, D_2, \ldots$ operators and this is done by considering the x and t derivatives involved in the functions which are represented by the placeholders for the derivatives required and by considering the order of these derivatives for the sign of the derivative operator terms.

The placeholder p represents u_t and so the derivative operator required for D_1 is $\partial/\partial t$. The derivative operator for D_3 is u_{xt} and so the derivative operator required is $\partial^2/\partial x \partial t$ .One may continue in this manner constructing the necessary replacements for the other derivative operators to obtain the Euler-Lagrange equations

$$\frac{\partial L}{\partial u} - \frac{\partial}{\partial t}\left(\frac{\partial L}{\partial p}\right) - \frac{\partial}{\partial x}\left(\frac{\partial L}{\partial q}\right) + \frac{\partial^2}{\partial x \partial t}\left(\frac{\partial L}{\partial r}\right) - \frac{\partial^3}{\partial x^2 \partial t}\left(\frac{\partial L}{\partial s}\right) = 0 \tag{5.40}$$

and

$$\frac{\partial L}{\partial v} - \frac{\partial}{\partial t}\left(\frac{\partial L}{\partial j}\right) - \frac{\partial}{\partial x}\left(\frac{\partial L}{\partial k}\right) = 0 \tag{5.41}$$

The signs of the terms in the equations of motion are determined by examining the order of the associated derivative. All of the even order derivatives have positive signs and all of the odd order derivatives have negative signs. If we then carry out the operations indicated in equations (5.40) and (5.41), we will obtain the equations of motion for each of the two functions u and v. They are

$$-2au_x - 2bu_{xx} + 6u_{xxx}u_{xx} + 6u_x u_{xxx} + dv = 0 \tag{5.42}$$

for u and

$$du = 0 \tag{5.43}$$

for v.

Obviously this example is not very physical. It does, however, demonstrate the main points of the procedure. The steps can be summarised as follows:-

1). examine the lagrangian to determine the functions required for the determination of the extremal;

2). locate and classify all occurrences of derivatives of each of the functions and construct a set or list of all such objects;

3). substitute for all of the different derivative types with unique placeholders;

4a). use the derivative terms to be found in each of these sets to construct Euler-Lagrange equations;

4b). use the derivatives involved in forming each of the placeholders to determine the derivative operators in the Euler-Lagrange equations;

4c). use the order of derivative to determine the sign of the terms in the Euler-Lagrange equations;

5). apply the Euler-Lagrange equations to the lagrangian to determine the equation of motion.

This mirrors almost exactly the procedure used in determining the equations of motion for each lagrangian. There is, however, one additional point that has not been raised namely the equivalence of mixed derivatives with respect to the ordering of the derivatives. In carrying out the procedure for calculation of the equations of motion, the ordering of the derivative terms is not usually something which one would explicitly consider. It would be expected that each term would have its derivatives ordered in some particular manner and that, if the derivatives were not ordered, then the person doing the calculation would simply rearrange them. MAPLE is not aware of the equivalence of derivative terms whose derivatives are performed in differing orders since we are using a custom-designed scheme to denote differentiation. In order that MAPLE correctly treat terms which are identical but for order, some canonical ordering must be imposed on the indices within the derivative function. This is the topic of the next section.

5.8.2. Ordering the indices of *DX* and *DT* types: The *ordindex* procedure.

As has been pointed out before, in order that derivative terms be recognised as being equal, it is necessary to impose a canonical ordering on their indices. This is carried out by the procedure *ordindex*. For example, suppose we have the following expressions

$$\frac{\partial^3 f}{\partial x_1 \partial x_3 \partial t_1}, \quad \frac{\partial^3 f}{\partial x_3 \partial t_1 \partial x_1}, \quad \frac{\partial^3 f}{\partial t_1 \partial x_1 \partial x_3}$$
$$\frac{\partial^3 f}{\partial x_1 \partial t_1 \partial x_3}, \quad \frac{\partial^3 f}{\partial x_3 \partial x_1 \partial t_1}, \quad \frac{\partial^3 f}{\partial t_1 \partial x_3 \partial x_1} \tag{5.44}$$

These expressions are obviously mathematically identical. However the program will represent them as

$$DX(DT(f,1),3,1), \quad DX(DT(DX(f,1),1),3), \quad DT(DX(f,3,1),1)$$
$$DX(DT(DX(f,3),1),1), \quad DX(DT(f,1),1,3), \quad DT(DX(f,1,3),1) \tag{5.45}$$

Thus, when the MAPLE program comes to collecting similar terms, it will regard each of these expressions as distinct although they are not. To avoid this situation, it is necessary to impose a canonical ordering on the indices and also the ordering of mixed derivatives.

The ordering of mixed *DX* and *DT* terms has already been discussed in the section which discusses the simplification and expansion routines for the *DX* and *DT* operators. The *simplify/DX* and *simplify/DT* procedures will rearrange any *DT(DX(* terms to give *DX(DT(* terms. The simplify procedure will also place all derivatives of the same type together so that the only three possibilities are DX(, DT(, and DX(DT(. Thus, once an expression has passed through the expression simplifier, *DT(DX(* terms will be removed and we need no longer consider the relative ordering of mixed derivatives. The ordering of the indices within the *DX* and *DT* operators, however, is still not fixed. In terms of the example, the forms

$$DT(DX(f,3,1),1), \quad DT(DX(f,1,3),1) \tag{5.46}$$

are disallowed on the grounds that the DX and DT derivatives are in the wrong order and the terms

$$DX(DT(DX(f,1),1),3), \quad DX(DT(DX(f,3),1),1) \tag{5.47}$$

are disallowed because they involve unsimplified differentiations. The expressions

$$DX(DT(f,1),3,1), \qquad DX(DT(f,1),1,3), \tag{5.48}$$

are still allowed. To avoid any such ambiguity, it is necessary to apply an ordering to the indices themselves. This is relatively easy to achieve. We impose the condition that the indices should be ordered in increasing numerical value. Thus the only one of the example expressions which remains available to the program is

$$DX(DT(f,1),1,3) \tag{5.49}$$

and it will be seen that this ordering is unique given the two rules imposed.

The procedure which imposes this ordering of indices is quite short and compact. An examination of the code shows that, as usual, the code examines smaller and smaller parts of the argument expression until it finds a *DX* or *DT* type. For *DX* types it checks to see whether the object being differentiated is of type *DT*. If so it orders the indices of the *DT* procedure first using the built-in MAPLE function *sort* [1]. Once this has been achieved the indices of the *DX* part of the mixed *DX(DT(* expression are ordered in the same way. If the *DX(* expression does not have a *DT(* expression as its argument, then the indices of the *DX* operator are sorted. If the type of the argument expression is of type *DT*, the indices of the *DT(* expression are sorted. Otherwise the procedure does nothing.

Although it is important for the reasons discussed above that some canonical ordering be imposed, if this is not done, the answer obtained by the program will not be incorrect. It will merely have more terms than is necessary. In any case, higher order derivatives only tend to appear in the higher order lagrangians so that this imposition of canonical ordering will have little impact on the more common types of lagrangian.

5.8.3. The File *eqnmot*: Finding out the Variables and Derivatives Used.

It will be remembered from section 5.8.1. that the algorithm used to determine the equations of motion modelled exactly that used when determining the equations by hand. The function of the *eqnmot* file, which contains the *equationsofmotion*

procedure, and of the *mapmod* procedure, is to implement the first of the two steps to be taken in the determination of the equations of motion. In fact, the first of those steps, determining the variables with respect to which the extremal is to be found, has already been carried out since this information is placed in the input file as the set *varset*. The function of the first of the two procedures, therefore, is to determine not which variables are required in the calculation of the extremal but whether these variables are present in the lagrangian under consideration. The *mapmod* procedure is used to make up lists of the derivatives of these variables. These two procedures are now described in the following two sections.

5.8.3.1 The *equationsofmotion* Procedure.

The function of this procedure is to create sets containing the variables contained within the lagrangian supplied to the procedure as its argument. Obviously the procedure must know the variables which are likely to form these sets and this is achieved by using the global lists *newnlist* (the scalar variables), *newklist* (the vector variables) and the series expansion limit *JMAX* together with the *hiharm* boolean variable. By utilising the information in these global entities, the code can search only for indexed variables of the correct type. However, this assumes that these entities are defined. Thus the first action of the code is to check that this is indeed the case. It then checks to see that the five sets *scalvar*, *vecvar*, *DTvar*, *DXvar* and *DXDTvar* are not defined. If they are, it prints a warning message. Again this is to avoid unintentional looping. These sets will be used in the determination of the correct Euler-Lagrange equations.

The actual work of the procedure is then quite simply accomplished. It creates the sets *scalvar* and *vecvar* and the scans the lagrangian looking for all possible combinations of name and index within indexed functions. If scalar functions are found, they are added to the *scalvar* set using the *union* function for sets. Similarly, if vector variables are found, they are added to the set *vecvar*. The final act of the procedure code is to call the procedure *mapmod*.

5.8.3.2. The *mapmod* Procedure.

It will be recalled that in the implementation of the *smartexpand* procedure that a procedure called *maptree* was written to produce a map of the lagrangian expression. The procedure worked by recursively descending the tree structure representation of the lagrangian noting the type, number and arguments of the tree's nodes. The *mapmod* procedure is a modified version of this code whose purpose is this time to seek instances of the occurrence of *DX* or *DT* types within a lagrangian expression and to produce three sets of such objects, one set for each of the two types and one set for mixed derivatives. These sets are called *DXvar*, *DTvar* and *DXDTvar* respectively. These sets will then be converted to lists for use in obtaining the equations of motion.

In a similar way to the *maptree* procedure, the *mapmod* procedure serves only as a harness for the *domapmod* procedure. It simply creates the sets which will hold the different derivative types and initialises the level variable. The check to see whether the sets have been previously defined is simply a debugging measure to avoid unintended looping.

The code for the *mapmod* procedure is simpler than that for the *maptree* procedure in that the code merely searches for the *DX* and *DT* type operators. Since the lagrangian expression has already been simplified, only *DX(*, *DT(* and *DX(DT(* expressions will exist. If the mapmod procedure finds a *DT(* expression it must be of type *DT*: if it finds a *DX(* expression, it need only determine whether or not the argument to the *DX* operator is a *DT(* expression to decide whether the object is a mixed derivative. No other possibilities are allowed. The code in the main program then uses these sets together with the information obtained by the *equationsofmotion* procedure to produce the equations of motion.

5.8.4. Sorting the Variables.

Once the variables and the derivatives of these variables present in the lagrangian expressions have been determined, one could then go on to apply the code which produces the equations of motion to these sets directly. However, this would produce a series of equations in which the variables and their derivatives appeared in the equations of motion in an arbitrary order which simply mimicked the order in which the variables and their derivatives were to be found in the lagrangian expression. This is not entirely

satisfactory. Avoiding this problem involves the simple expedient of converting these sets of variables into lists (which preserve ordering whereas sets do not) and then sorting these lists into order. MAPLE's *sort* procedure has already been used in the *ordindex* procedure. However, in that procedure the ordering was based on whether one number was greater than or equal to another. The ordering of functions which themselves have possible nested layers of indices is not something which can be so simply dealt with.

Fortunately, the *sort* procedure has been written in such a manner that, if the objects to be sorted are of some unusual type, then the user can supply an alternative ordering procedure which will be used in the *sort* procedure's internal decision making process. The ordering procedure is supplied as an extra argument to the *sort* call and should be boolean. It should return a value of true if the objects it is being asked to compare are in the correct order and false if not.

In order that the equations of motion produced are always in the same order, therefore, the sets of variables produced - *scalvar*, *vecvar*, *DXvar*, *DTvar* and *DXDTvar* - are converted into lists and then sorted using two user supplied boolean procedures. These procedures are called *dordbool* and *fordbool* and they are for use in ordering derivative types or (function) variable types respectively. We now go on to consider the operation of these procedures.

5.8.4.1 The *dordbool* and Associated Procedures.

The function of the *dordbool* procedure is to compare two derivative expressions of the same type - *DX*, *DT* or mixed *DX(DT* - and to return a value of true if the two elements are to be considered to be correctly ordered and false otherwise. An examination of the code for this procedure shows that the function of the top level procedure *dordbool* is to insure that the two arguments supplied to it have matching types. The *dordbool* procedure then uses two further subprocedures to do the processing. The first of these is called *doordD1bool* and it deals with the case where the derivative terms are of either one of the two types *DX* or *DT*. The second procedure is called *doordDXDTbool* and it is used when ordering the mixed derivative type.

The *doordDXDTbool* procedure is quite short in that it uses the *doordD1bool* procedure to do most of the work. If the *DT* parts of its arguments, which are both *DX(DT(* expressions, are identical, the two are ordered using the *DX* portions. If they

are not identical, the arguments are ordered using the *DT* parts. If the *DT* parts have not been ordered before entering this procedure, this will lead to inconsistent behaviour.

This then leads us on to discuss the operation of the *doordD1bool* procedure. Before doing so, we must explain how three small procedures- *fordbool*, *szip* and *lzip*-function since they are utilised within the *doordD1bool* procedure. *fordbool* is a procedure used for ordering the indexed functions used in the expansion of the variables performed by the *sums* procedures. *fordbool* will order functions according to the following sequence of priorities: lexical ordering of names, ordering of order of function (the first index within the brackets), ordering of the harmonics (the second index). *lzip* takes two arguments and places them in a list without changing their order; *szip* does the same but places the elements in the list in sorted order. The names come from the fact that the library function *zip* [1] will be applied to lists of these objects.

The operation of the *doordD1bool* procedure is now easy to explain. Given two *DX* or *DT* type procedures, the code firstly compares the order of the two argument functions against the two arguments sorted according to the *fordbool* procedure. If the functions are identical, then the code must use the arguments to the functions to decide upon their ordering. Firstly it considers the length of the index lists. The shorter lists will be placed before the longer ones. Then, if the lengths of the two lists are equal, their elements must be used to order the two functions. Two temporary lists are manufactured. The first is a *zip*ped list of the index lists in which each of the sublists has been sorted. The second is a similar *zip*ped list in which the sublists have not been sorted. They are formed using the *szip* and *lzip* procedures mentioned previously. The code then compares the first elements of each of the two lists. Unless each of the elements is identical, there is some discrepancy between the lists and so the *DX* or *DT* functions must be in the wrong order. Again this assumes that the index lists have been sorted prior to their comparison: if the lists have not been sorted, this will lead to discrepancies in the behaviour of the sorting procedure.

5.8.5. Generating the Equations.

To recap on the topics already presented, we have seen how the variables within the lagrangian are identified and then placed in sets, and the manner in which the dependent variables, the functions to be used in finding the extremal, may be sorted into order using the *dordbool* and *fordbool* procedures. However, this only allows us to determine which variables we are required to use in constructing the equations of

motion. That is, we have achieved only the first of the two steps listed in the introductory section 5.8.1.. In the subsequent sections we will examine how the code in the main program examines these sets and applies some subsidiary procedures to determine the equations of motion.

5.8.5.1. The Main Program Code.

The actions required to determine the equations of motion assume that the program has calculated each of the averaged lagrangians at each order of the small parameter and saved them in the table *LC<number>bar*. The program is now required to determine the equations for each order of the small parameter as well as for each of the variables which have been the subject of expansion. At each order, therefore, there will be an increasing number of equations which will need to be saved in an identifiable way. There is the additional problem that, in the case of some lagrangians, the same equations are generated for different variables at each order. It is important, therefore, that the equations at each order be stored separately. Finally, although the processing involved in the determination of the equations for scalar and vector variables is essentially the same, some provision has been made to allow for differences in the processing in future versions. With these statements in mind, let us now examine the code contained in the final section of the main program.

The program first begins by reading in the necessary files. Most of these have already been discussed and deal with the determination of the variables within the lagrangian and with the ordering of these variables. Once this has been performed, the program enters a loop in which the lagrangians at each order of the small parameter are considered in turn. The code uses the *equationsofmotion* procedure (which calls the *mapmod* procedure) to create the five sets which contain the vector and scalar variables and all of the *DX(*, *DT(* and *DX(DT(* expressions. These sets must then be converted to lists since the ordering within a set is arbitrary and defined by the MAPLE program. The resulting lists are then sorted using the *fordbool* procedure as an argument to the internal *sort* procedure for the case of the lists of the scalar and vector functions. The other three lists of derivative terms are sorted using the *dordbool* procedure. It is then necessary to create the Euler-Lagrange equations for each lagrangian and, as has been pointed out, the scalar and vector variables are treated separately.

The program's next action is to create two sets, *scalareqnset* and *vectoreqnset* which will be used to hold tables of equations for each of the scalar and vector variables

respectively. It then uses the lists *newscalvar* and *newvecvar*, which contain the scalar and vector variables in list form, to examine the lagrangian for each scalar and then each vector variable in turn. Firstly for the scalar equations, the program uses the variables to be found within the set *newscalvar*. For each of the variables in combination with each of the levels of expansion (the power of the small expansion parameter ε) the program determines whether a table to hold the equations of motion which are to be generated exists within the set *scalareqnset*. If it does not, then the program creates one. The table has two indices, one for the power of expansion and one for the order of the harmonic. In this way, the equations generated for each of the variables created when the substitutions for the original dependent variables were made will have table elements assigned to them for the storage of their equations of motion. Once the table has been created (if this is necessary) the program's next action is to assign the table element corresponding to the variable under consideration to a function of that variable. The function used is called *makeELterm* and it is discussed in the next section. Essentially its function is to generate the terms arising from the section of the Euler-Lagrange equation denoted by its first argument when applied to the lagrangian which is its second argument.

The code then uses the information stored in the lists *newDTvar*, *newDXvar* and *newDXDTvar* in turn to examine the lagrangians at each order for derivative terms involving the derivatives of the variable taken from *newscalvar* which is currently under investigation. For each of these derivatives, the program will then use the procedure *makeELterm* to generate the subsequent parts of the Euler-Lagrange equation which involves derivative terms.

To be more explicit, suppose that the variable under investigation at one level of the expansion variable is A_[1,2]. Since it is a scalar variable it will be found in the list *newscalvar*. If a table called *eqtn<level>A_* does not exist, it is created. The element [1,2] of this table is then set equal to *makeELterm* applied to the lagrangian at that level. This corresponds to

$$\partial L / \partial A\_[1,2] \tag{5.50}$$

of the Euler-Lagrange equation. Then the lists *newDTvar*, *newDXvar* and *newDXDTvar* are examined in turn to see if the Euler-Lagrange equation contains terms containing derivatives of the variable *A_[1,2]*. If the list *newDTvar* contains *DT(A_[1,2],2)* for example, the function *makeELterm* will be applied to the lagrangian

with this as argument to generate the section of the Euler-Lagrange equation corresponding to

$$-\partial/\partial t_2\left(\partial L/\partial(\partial A\_[1,2]/\partial t_2)\right) \tag{5.51}$$

In this way all of the terms within the Euler-Lagrange equation will be generated without the program having to perform calculations for terms which will not be present in the final equations.

The code which applies to vector variables is identical in form. The variables are of course taken from the list *newvecvar* and the equations placed in the set *vectoreqnset*. Finally, having generated all of the equations for each of the scalar and vector variables at one order, the code uses a further procedure, *doprint2*, to print out the contents of the tables of equations of motion for each of the variables. The equations are then saved in machine readable form for future manipulation by the user if required.

5.8.5.2. The *makeELterm* and *derivord* Procedures.

As has already been pointed out in the previous section, the function *makeELterm* forms a vital part of the code necessary to determine the equations of motion for each of the variables to be found in each of the lagrangians. Its function is to calculate the Euler-Lagrange equation for each variable within each lagrangian on a term by term basis and then to apply the Euler-Lagrange term to the lagrangian to calculate additional terms within the equations of motion. If we refer to the sequence of operations listed in section 5.8.1, we see that this short procedure carries out the actions listed as 3, 4a, 4b and part of 5. That is, it:-

1). substitutes for all of the different derivative types with unique placeholders;

2). uses the derivative terms to be found in each of these sets to construct Euler-Lagrange equations on a term by term basis;

3). uses the derivatives involved in forming each of the placeholders to determine the derivative operators in the Euler-Lagrange equations again on a term by term basis;

4). uses the order of derivative to determine the sign of the terms in the Euler-Lagrange equations using a service procedure;

5). applies the Euler-Lagrange equations to the lagrangian to determine the equation of motion.

The procedure uses a service procedure, *derivord*, to calculate the sign of the term and the procedure *doprint2* is used to make the printout. However, now that all of the variables required for the calculation of the equations of motion have been determined and placed in the necessary sets and lists, it may be seen that the bulk of the remaining work required in forming the equations of motion is done by this procedure.

If we now examine the code for the procedure, we can see how the code operates. The procedure takes as arguments the function with respect to which the derivative in the Euler-Lagrange equation is to be found and the lagrangian to be used in the calculation. Its first operation is to determine the sign of the term within the Euler- Lagrange equation to be found. It does this using the procedure *derivord* which is discussed in the next section.

The procedure then uses the MAPLE function *frontend* [1] to both make unique substitutions for all variables and to perform the inner derivative within an Euler-Lagrange term. The function of the *frontend* procedure is "to extend the domain of computation" of many MAPLE functions. It does this by substituting for subexpressions within its main argument so that the function to be applied to this argument need not worry about the types of the subexpressions contained within the main expression. To put it more concisely, the function substitutes for subexpressions with atomic placeholders, performs some action and then replaces the placeholders with their original contents. The function call within the makeELterm procedure is

frontend(diff,[L,star(f)],[{`+`,`*`,`**`},{}]) (5.52)

What this says is the expression L is to be differentiated with respect to *star(f)* where types included in the first set are not to be replaced with unique placeholders. The expression *star(f)* is used in order to obtain the expression involving the variable rather than the conjugate expression involving the complex conjugate of the variable. This will perform the

$$\partial L \Big/ \partial\left(\partial A^{*}\_[1,2]/\partial t_2\right) \tag{5.53}$$

part of a

$$-\partial/\partial t_2\left(\partial L/\partial(\partial A^*\_[1,2]/\partial t_2)\right) \tag{5.54}$$

expression if the call made is

*frontend(diff, [L,star(DT(A*_[1,2],2))], [{`+`,`*`,`**`},{}])* (5.55)

If there is no *DX* or *DT* operator involved, this will find the complete Euler-Lagrange term.

Once the code has determined the inner part of the expression, it must perform the derivative corresponding to the outer part. For each of the three cases, *DT(*, *DX(*, *DX(DT(*, the code will perform this integration by placing the inner expression within an appropriate differential operator, the arguments to the operator being determined from the variable in use, and then using the *simplify/DX* and *simplify/DT* routines to simplify the resulting expression. This performs the outer differentiation. The final act of the procedure is to return $(-1)^k$ times the calculated result where k is the order of the derivative. This results in the calculation of the Euler-Lagrange terms as well as the result of applying them to the procedure at the same time.

The sign of the terms is calculated by using the procedure *derivord*. This is a remarkably simple procedure which returns the order of a derivative of any of the three allowed types by counting the number of indices. If the derivative type is *DX(DT(* then the derivative order is the sum of the *DX* and *DT* indices.

5.8.5.3. Printing the Equations: the Procedure *doprint2*

Having determined the equations of motion, they must be printed out in the correct form. Since the procedure is called from within the main loop of the main program, it will print out results for each lagrangian separately. Accordingly it need not take account of the level of expansion.

The code implemented is quite simple. Scalar and vector equation sets are handled separately. Having performed some simple error checking, the code merely uses the tables stored in the sets *scalareqnset* and *vectoreqnset*. Taking each element from each of these sets in turn, the program enters a double loop in which counters representing the two indices of the variables are incremented and the equations corresponding to the variable connected with the two indices is printed out.

The only point likely to cause confusion is the use of the MAPLE functions *substring* and *length*. The names of the tables are *eqtn<level>variable*. Assuming that the length of the string denoting the level is one character, the sixth and subsequent characters form the name of the variable which is being investigated. Truncating the table name in this way allows the program to printout a message informing the user which variable the equations refer to.

5.9 Concluding Remarks.

We have now examined in exhaustive detail the workings of the MAPLE code which implements Whitham's averaged lagrangian procedure. We have noted that the procedures involved are capable of calculating the equations of motion for a general lagrangian in multiple variables, either vector or scalar. The main assumption of the code is that the vector variables will have a particular significance as descriptors of the occupations of some state level scheme. We have also noted at which points the code could be modified to improve performance or functionality. What is now required is the application of the program to some lagrangians from the field of nonlinear optics.

References.

1. Char, B.W., *et al.*, *MAPLE Reference Manual.* 5th edition ed. 1988, Waterloo, Ontario, Canada: WATCOM Publications Limited, 415 Philip Street, Waterloo, Ontario, Canada. 403 pp.

2. Jeffrey, A. and T. Kawahara, *Asymptotic Methods in Nonlinear Wave Theory.* 1 ed. Applicable Mathematics Series, Pitman Advanced Publishing Program, 1982, Boston, London, Melbourne: Pitman Books Limited. 256 pp.

3. Elsgolc, L.E., *Calculus of Variations.* International Series of Monographs on Pure and Applied Mathematics, ed. I.N. Sneddon, M. Stark, and S. Ulam. 1961, Oxford, London, New York, Paris: Pergamon Press. 178 pp.

Chapter 6

All my possessions for a moment of time.

Queen Elizabeth I.

6.1. Introduction.

We have now examined all of the background material relevant to the thesis. We have touched on mathematical solution techniques for nonlinear partial differential equations, Maxwell-Bloch type equations, all of the variants of Whitham's technique and have derived several lagrangians, two of which describe models of electromagnetic wave propagation in nonlinear media. In addition, we have describe how two computer algebra programs, one written in REDUCE and the other in MAPLE, can be used to carry out the calculation of the averaged lagrangians associated with lagrangian models. The second of these models also carries out the calculation of the equations of motion associated with the calculated lagrangian. We now apply the MAPLE program to the lagrangian descriptions of the Boussinesq equation, the classical model of nonlinear propagation due to Small and the semi-classical model derived in the chapter 4. We then summarise the work contained in the thesis, make comments on the equations produced, and give suggestions for future work.

6.2. The Lagrangians.

We now present the results of the calculations based on the lagrangians derived in the previous chapters. The results of the calculation for the Boussinesq equation are given for comparison with the results obtained by hand and presented previously. The results for the classical lagrangian due to Small display some interesting recurrences in the forms of the equations of motion. The results of a calculation using the two-level semi-classical lagrangian derived in chapter 4 are presented together with an explanation of the modifications made to the Whitham method to allow for the inclusion of conjugate variables to describe the series expansions for the components of the state occupation vector **v**.

6.2.1. The Boussinesq Equation

We have already seen in chapter 3 that the Boussinesq equation was the first equation to which the Kawahara variant of Whitham's method was applied. The results of the calculation as performed by hand are presented in that chapter together with a sample calculation of the results for the same lagrangian as carried out by hand. Since the results for this equation are already available, it forms a good test of the MAPLE program's abilities. It should be noted that both an explanation of the workings of the MAPLE program and the results for the Boussinesq equation were reported in the paper by Arbuckle and Arnold [1].

Using the lagrangian density

$$L = \tfrac{1}{2} f_t^2 - \tfrac{1}{2} c^2 f_x^2 + \tfrac{1}{2} \mu f_{xt}^2 - \tfrac{1}{6} f_x^3 \tag{6.1}$$

the equations of motion as calculated by the program are as given in appendix 5. The first of these is obviously the linear dispersion relation

$$w^2 f\_[0,1] - k^2 f\_[0,1] c^2 + mu k^2 w^2 f\_[0,1] = 0 \tag{6.2}$$

where there are obvious changes to the notation to allow for the inability of the computer to deal with Greek symbols. The next two equations are somewhat complicated. They represent the equations of motion which correspond to the next two orders of expansion. It will be noticed that since the program is unaware of the linear dispersion relation derived as the first equation of motion, it cannot simplify the terms in the same manner as when carrying out the calculations by hand. This leads to complicated and messy equations which are difficult to check. Nevertheless, the two hours required to obtain the result represent a considerable saving over the three weeks taken to perform the calculations by hand.

6.2.2. The Classical Lagrangian.

The classical lagrangian given by Small was presented in chapter 4 as

$$L = \frac{\varepsilon_0}{2}\left(A_t^2 - c^2 A_x^2\right) - A_t P + \frac{1}{2\varepsilon_0 \omega_p^2}\left(P_t^2 - V(P)\right) \tag{6.3}$$

We assume that the potential function V(P) can be written in terms of a power series in P and take the lagrangian as

$$L=\frac{\varepsilon_0}{2}\left(A_t^2-c^2A_x^2\right)-A_tP+\frac{1}{2\varepsilon_0\omega_p^2}\left(P_t^2-v_0^2P^2+\tfrac{1}{2}\alpha P^4\right) \tag{6.4}$$

where v_0 and α are constants. We then carry out the Kawahara procedure using the MAPLE program, expanding the variables *A* and *P*. The results obtained are as given in appendix 6. Note how the same equations of motion migrate up the list of equations for each order, the equation for the highest order variable always being the linearised dispersion relation. Again the complexity of the equations is exacerbated by the lack of grouping of the terms or of the use of the linear dispersion relation and its derivatives to simplify the notation.

6.2.3. The Semi-Classical Lagrangian.

This lagrangian was given in chapter 4 as

$$L=\frac{\varepsilon_0}{2}\left(A_t^2-c^2A_x^2\right)-A_tN\begin{pmatrix}v_1^* & v_2^*\end{pmatrix}\begin{pmatrix}0 & d_{12}\\ d_{21} & 0\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix}+i\hbar N\begin{pmatrix}v_1^* & v_2^*\end{pmatrix}\frac{\partial}{\partial t}\begin{pmatrix}v_1\\ v_2\end{pmatrix}$$
$$-\begin{pmatrix}v_1^* & v_2^*\end{pmatrix}\begin{pmatrix}E_1 & 0\\ 0 & E_2\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix} \tag{6.5}$$

The expansion of the variable *A* representing the electric field can be carried out as before. However, the expansion of the conjugate variables for the state occupation vector elements requires further explanation.

We stated in chapter 5 that the expansion of the vector variables required different summation operations from those for the scalar variables. Not only does the summation involve expansion in odd half integer harmonics but there are sign changes made to the exponents as well as the addition of an extra program variable to force the program to calculate the expansion taking the first of the state occupation vector components, v_1, as the ground state and assigning the summation a different series.

The reason for the half integral harmonics can be found by examining the expression for the polarization (2-level)

$$\left\langle\hat{P}\right\rangle=v_1^*v_2d_{12}+v_2^*v_1d_{12} \tag{6.6}$$

We see that in order that the polarization not vanish as a result of averaging, it is necessary that the matrix elements contain harmonics which have integral powers since these will then cancel with those of the electric field's series expansion. Since we have chosen a system in which the odd levels have one parity and the even levels another, we can give different signs to the harmonic series of each level so that the exponentials of the terms of opposite polarity cancel. In other words we assign

$$v_1 = \varepsilon^1 v e^{-i\theta/2},\ v_2 = \varepsilon^2 v e^{i\theta/2},\ v_3 = \varepsilon^3 v e^{-i\theta/2} \tag{6.7}$$

to first order. The higher order terms can be assigned similarly so that for example

$$v_1 = \varepsilon^1 v e^{-i\theta/2} + \varepsilon^2 \sum_{l=1}^{\infty} {}^1v_1^{(l)} e^{-il\theta} e^{-i\theta/2} + \varepsilon^3 \sum_{m=1}^{\infty} {}^1v_2^{(m)} e^{-im\theta} e^{-i\theta/2} + \ldots \tag{6.8}$$

the prefix before the v-functions denotes the vector element to which the expansion belongs and in expansions for even numbered levels the half integer exponent has the opposite sign. It is assumed that the functions are normalised and that any normalisation constants are subsumed into the v-functions.

A further complication arises if the first of the v-functions represents a ground state. It is assumed that if this is so, under the experimental cases most often found, the ground state is generally the most highly populated level. It is therefore possible to set the expansion for the first of these v-functions to be a series as given above subtracted from unity.

The output for a sample calculation which uses a quantum lagrangian is given in appendix 7. It will be apparent from an examination of the output that none of the equations of motion generated involves any of the vector variables. They have all been removed during the course of the averaging process although the terms are generated by the summation mechanism. Obviously, what is desired is that equations of motion including the vector variables are generated by the program. It should therefore be understood that the extension of the program to deal with vector quantities is unfinished. Indeed, whilst the thesis was being completed, several modifications were made to the program to allow it to handle the vector quantities correctly: the use of indexed variables was giving rise to difficulties during the execution of the simplification routines simplify/DX and simplify/DT. The output should be treated only as indicative of the possibility that quantum results could be calculated by the program. This section is in need of further work.

It is further noted that this is only a tentative approach at solving the problem since the summations involved seem somewhat unsatisfactory. Nevertheless, since any corrected expansion required would require only the definition of a differing series summation within the program, this should not present any great problem.

6.3. Comments on the Equations.

The comment is often made that the output from computer algebra programs is generally more complex than is to be desired: that is certainly the case for the examples considered above. Nevertheless, these equations should not be considered as being worthless. They are no more complicated than similar expressions that have been obtained by working by hand and, indeed, the case of the equations of motion for the Boussinesq equation bears this out. It is at the level of approximation presented that the nonlinearity of the equations first begins to make itself felt and at which simplifying the equations gives rise to equations similar to the nonlinear Schrödinger equation and other such nonlinear equations.

The trouble with simplifying the equations of motion by substitutions of equations already obtained into higher order ones is that such a process is not algorithmic. In order to obtain the most utilitarian equations, it is necessary to make use of certain pieces of information obtained whilst not using others. For this reason, the results are stored in computer readable form so that the resources of the MAPLE system can be brought to bear on the results under the users control. However, we have seen that the use of the linear dispersion relation and its derivatives with respect to frequency and wavenumber gives rise to some simplification in the notation. We will mention this again in the next section.

It was our original intention that the results produced by the Kawahara variant could be used to investigate the effects of differently structured lagrangians on the equations of motion obeyed by the physical system. The capability of the system to retain sufficient information to make this a reasonable aim is demonstrated by the complexity of the equations produced. Since the time for a "production run" can now be measured in hours rather than weeks, it has become a sensible proposition to examine the effects of changes in the lagrangian by simply modifying the lagrangian and recalculating. In addition, if the summations used to carry out the series required for the

substitution need to be modified, it is a relatively simple task to do so. This is especially relevant in the case of the semi-classical lagrangian where different summation schemes can be used to perhaps represent different initial conditions

6.4. Summary.

The main result obtained by this work is the automation of the Kawahara variant of Whitham's averaged lagrangian method by the use of a computer algebra program written using the MAPLE computer algebra system. Using it, or a previous program written using the REDUCE system allows the calculation of averaged lagrangians, and in the case of the MAPLE program, their equations of motion within a few hours rather than the weeks which such calculations usually took. The actual time taken depends on the complexity of the lagrangian and the order of expansion but is generally of the order of hours. A program listing for each program is presented at the end of the thesis.

In addition, the thesis contains substantial background work relating to the future extensions to the program. The use of the SU(N) matrix representations for calculations involving vector systems could be used to simplify the notation for the calculations as well as affording more efficient calculation by means of the commutation rules for these representations. Although the block diagonalization achieved under certain circumstances is fortuitous, the use of matrix quantities, especially based on orthogonal basis sets, could allow calculations to be undertaken for systems yet more complicated than those already tackled.

The number of references attests to the depth of the literature surveyed: all of the work related to the main thread of the thesis has been discussed and evaluated. Some of the more complicated and fundamental issues are the subject of research which is beyond the scope of the thesis. Since the thesis is essentially multidisciplinary, discussion of these topics has been limited in order to keep the thesis to a reasonable length. Nevertheless, mention of such topics has been made wherever possible.

As usual in research, the attempt to answer one question has given rise to several more. Since the author's life, and funds, are limited, it has been thought prudent to finish the thesis at this point although there remain many more questions to be answered. It seems likely that further work on this program might produce a tool which

can be used in everyday situations to provide answers to some interesting physical problems.

6.5. Future Work.

References to future work can be found throughout the text. As with any piece of research, the problem under investigation grows to fill the available resources and one person can only hope to make a contribution to relevant results: only in rare cases does a researcher give definite and complete answers to a problem. The research here suggests many possible avenues for further work.

The most obvious avenue for further effort lies in the MAPLE program. We stated in chapter five that the *smartexpand* procedure represented one area where further effort was definitely required. Although efforts have been made by the MAPLE team to improve the limitation in the number of terms which may be held in any one expression, such improvements have practical limits. The efficiency of the MAPLE program itself would be compromised for problems not requiring many terms: there would be an overhead in processing data structures capable of holding many hundreds of thousands of terms if only a few hundred were present in the expression.

We further stated that the modification of the program to include boundary conditions would be a task which would be worth carrying out since the modelling of layered structures, such as waveguides and couplers, would mean that the program could be applied to physical configurations of engineering interest. This type of modification is conceivably possible but would require extensive modifications to the running of the program taking several more months or years. As the program stands, it represents two man years of effort (part-time).

The possible inclusion of code to allow the model to include dissipation was mentioned in chapter 3. The use of a Rayleigh dissipation function in the model would require modifications to the main program code in that the virtual work performed by the lagrangian functional would now be non-zero. The code would need to be modified in such a way that the Rayleigh dissipation function was absorbed into the lagrangian.

A further topic for investigation is the use of external potentials within the quantum and classical lagrangians. In the derivation of the lagrangian for the classical lagrangian we stated that the external potential was set to zero to allow the equivalence of

the Lorentz and Coulomb gauge lagrangians. Determination of a Coulomb gauge lagrangian for either problem would allow the modelling of laser devices.

The use of symmetry methods in simplifying the lagrangians was suggested in chapter 1. By determining the symmetry of the differential equations describing the physical situation, it is possible to reduce the order of the differential equations by introducing supplementary conditions. Two papers which have not been previously mentioned and might prove of interest are those by Hebda [2] and by Battle and coworkers [3].

The use of Legendre transformations to transform the system from lagrangian to hamiltonian form has a utility which has been little demonstrated in this thesis. By determining the conjugate variables, the correct quantization scheme can be implemented. Whitham also suggested [4] that the use of such transformations was useful in determining the optimum form for the lagrangian for its use in the averaged lagrangian technique.

Although every possible effort has been made to insure that the program is correct and operates as is intended, it will be appreciated that in a program of this size the possibility of errors in coding cannot be ruled out. Errors were discovered during the writing of the thesis which were not discovered earlier because of the unusual test data supplied to the program. Furthermore, since the program is the creation of only one programmer, the author, it is likely that the types of error which would be discovered were the program to have been produced by someone working as a member of a group have not been entirely eliminated. Since the problem of coding bugs is common to all programs, and this one is maintained by only one author rather than by a team, it will be appreciated that work was terminated when a stable program behaviour was found. There may be further bugs yet to be discovered.

Finally, although the utility of the program in calculating results previously unobtainable without excessive effort has been demonstrated, the use of these results in further research has not been carried out. A possibility which is not inconceivable is that the algebraic program could be used as the front end for a numerical model. Suitable lagrangians could be expanded and simplified as much as possible and their averaged form used as the input to a numerical program which could optimize the lagrangians produced numerically. Alternatively, the equations of motion produced could be used within some discrete modelling scheme, such as a nonlinear form of the Beam Propagation Method, to allow the calculation of propagation in more realistic configurations. It had been hoped that some numerical work could be performed which

would supplement the current work but limitations of time and money force the termination of the research at this point. The quotation at the beginning of this chapter seems especially apt.

References.

1. Arbuckle, T.D. and J.M. Arnold. *A MAPLE Implementation of Whitham's Averaged Lagrangian Method.* in *IMACS '91.* 1991. Dublin: Elsevier.

2. Hebda, P.W., *Treatment of Higher-order Lagrangians via the Construction of Dynamically Equivalent first-order Lagrangians.* J. Math. Phys., 1990. **31**(9): p. 2116-2125.

3. Battle, C., *et al., Equivalence between the Lagrangian and Hamiltonian Formalism for Constrained Systems.* J. Math. Phys., 1986. **27**(12): p. 2953-2962.

4. Whitham, G.B., *Linear and Nonlinear Waves.* 1 ed. Pure and Applied Mathematics, 1974, New York, Chichester, Brisbane, Toronto, Singapore: John Wiley & Sons. 636 pp.

Appendix 1

We wish to determine the expectation value of an operator $\hat{A}$ with respect to the wavefunction given by

$$\psi = a_0\phi_0 + a_1\phi_1 \tag{A1.1}$$

The expectation value of an operator is given by

$$\left\langle \hat{A} \right\rangle_\psi = \langle \psi | \hat{A} | \psi \rangle \tag{A1.2}$$

Simply substitute the expression for the wave function ψ into the expression for the expectation value. One obtains easily

$$\begin{aligned} \left\langle \hat{A} \right\rangle_\psi &= \langle a_0\phi_0 + a_1\phi_1 | \hat{A} | a_0\phi_0 + a_1\phi_1 \rangle \\ &= \sum_{n=0}^{1} a_n^* a_0 \langle \phi_n | \hat{A} | \phi_0 \rangle + a_n^* a_1 \langle \phi_n | \hat{A} | \phi_1 \rangle \\ &= \sum_{m,n=0}^{1} a_n^* a_m \langle \phi_n | \hat{A} | \phi_m \rangle \end{aligned} \tag{A1.3}$$

The expression can obviously be extended to cover additional quantum levels by extending the range of the summations.

Appendix 2

We need to show that Schrödinger's equation in the form

$$i\hbar\frac{\partial a_{n-1}}{\partial t} = \sum_{m,n=1}^{2}\tilde{H}_{nm}a_{m-1} \tag{A2.1}$$

implies that

$$i\hbar\frac{\partial\tilde{\rho}}{\partial t} = \left[\tilde{H},\tilde{\rho}\right] \tag{A2.2}$$

which is Liouville's equation. The implication is one way: the converse does not necessarily hold. That is

$$i\hbar\frac{\partial a_{n-1}}{\partial t} = \sum_{m,n=1}^{2}\tilde{H}_{nm}a_{m-1} \Rightarrow i\hbar\frac{\partial\tilde{\rho}}{\partial t} = \left[\tilde{H},\tilde{\rho}\right] \tag{A2.3}$$

Consider $\frac{\partial\tilde{\rho}}{\partial t}$. We can write for example

$$\frac{\partial\rho_{11}}{\partial t} = \frac{\partial a_0}{\partial t}a_0^* + a_0\frac{\partial a_0^*}{\partial t} \tag{A2.4}$$

Use the Schrödinger equation in the form

$$i\hbar\frac{\partial a_0}{\partial t} = \tilde{H}_{11}a_0 + \tilde{H}_{12}a_1 \tag{A2.5}$$

together with its complex conjugate to substitute for the derivative terms in (A2.4). We obtain

$$i\hbar\left\{\frac{\partial a_0}{\partial t}a_0^* + a_0\frac{\partial a_0^*}{\partial t}\right\} = \left\{a_0a_0^*\tilde{H}_{11} + a_1a_0^*\tilde{H}_{12} - a_0a_0^*\tilde{H}_{11}^* - a_0a_1^*\tilde{H}_{12}^*\right\} \tag{A2.6}$$

and similarly for the other components. By inspection we see that we can write

$$\left\{a_0 a_0^* \tilde{H}_{11} + a_1 a_0^* \tilde{H}_{12} - a_0 a_0^* \tilde{H}_{11}^* - a_0 a_1^* \tilde{H}_{12}^*\right\} = \left\{\rho_{11}\tilde{H}_{11} + \rho_{21}\tilde{H}_{12} - \rho_{11}\tilde{H}_{11} - \rho_{12}\tilde{H}_{21}\right\} \tag{A2.7}$$

and that this is simply the 1,1 component of $\left[\tilde{H},\tilde{\rho}\right]$. Thus, the implication

$$i\hbar\frac{\partial a_{n-1}}{\partial t} = \sum_{m,n=1}^{2} \tilde{H}_{nm} a_{m-1} \Rightarrow i\hbar\frac{\partial\tilde{\rho}}{\partial t} = \left[\tilde{H},\tilde{\rho}\right] \tag{A2.8}$$

is proven.

Appendix 3

We stated that the density matrix and hamiltonian operator could be written as

$$\tilde{\rho} = \tfrac{1}{2}(\mathrm{I}_2 + \rho \cdot \sigma)$$
$$\tilde{H} = \tfrac{1}{2}\hbar(\omega_0 \mathrm{I}_2 + \omega \cdot \sigma)$$

(A3.1a,b)

where

$$\omega_0 = \tfrac{1}{\hbar}\mathrm{Tr}\tilde{H}$$
$$\rho = \mathrm{Tr}(\tilde{\rho}\sigma)$$
$$\omega = \tfrac{1}{\hbar}\mathrm{Tr}(\tilde{H}\sigma)$$

(A3.2a,b,c)

We require the additional fact that

$$Tr(\tilde{\rho}) = 1$$

(A3.3)

We shall prove only (A3.1a) since (A3.1b) is almost identical.

$$\begin{aligned}
\tfrac{1}{2}(\mathrm{I}_2 + \rho \cdot \sigma) &= \tfrac{1}{2}(1 \cdot \mathrm{I}_2 + \rho \cdot \sigma) \\
&= \tfrac{1}{2}(\mathrm{Tr}(\tilde{\rho})\mathrm{I}_2 + \rho \cdot \sigma) \\
&= \tfrac{1}{2}(\mathrm{Tr}(\tilde{\rho})\mathrm{I}_2 + \mathrm{Tr}(\tilde{\rho}\sigma) \cdot \sigma) \\
&= \tfrac{1}{2}\left((\rho_{11} + \rho_{22})\mathrm{I}_2 + \left(\mathrm{Tr}\begin{pmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{pmatrix} \right) \cdot \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{pmatrix} \right)
\end{aligned}$$

Expanding this expression gives the expressions on the following page.

$$\tfrac{1}{2}(I_2+\rho\cdot\sigma)=\tfrac{1}{2}\left((\rho_{11}+\rho_{22})I_2+\left(\mathrm{Tr}\left(\begin{pmatrix}\rho_{12}&\rho_{11}\\ \rho_{22}&\rho_{21}\end{pmatrix}\begin{pmatrix}i\rho_{12}&-i\rho_{11}\\ i\rho_{22}&-i\rho_{21}\end{pmatrix}\begin{pmatrix}\rho_{11}&-\rho_{12}\\ \rho_{21}&-\rho_{22}\end{pmatrix}\right)\cdot\left(\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}0&-i\\ i&0\end{pmatrix}\begin{pmatrix}1&0\\0&-1\end{pmatrix}\right)\right)\right)$$

$$=\tfrac{1}{2}\left(\begin{pmatrix}\rho_{11}+\rho_{22}&0\\0&\rho_{11}+\rho_{22}\end{pmatrix}+\begin{pmatrix}\rho_{12}+\rho_{21}\\ i(\rho_{12}-\rho_{21})\\ \rho_{11}-\rho_{22}\end{pmatrix}\cdot\left(\begin{pmatrix}0&1\\1&0\end{pmatrix}\begin{pmatrix}0&-i\\ i&0\end{pmatrix}\begin{pmatrix}1&0\\0&-1\end{pmatrix}\right)\right)$$

$$=\tfrac{1}{2}\left(\begin{pmatrix}\rho_{11}+\rho_{22}&0\\0&\rho_{11}+\rho_{22}\end{pmatrix}\right.$$

$$\left.+\left[\begin{pmatrix}0&\rho_{12}+\rho_{21}\\ \rho_{12}+\rho_{21}&0\end{pmatrix}+\begin{pmatrix}0&\rho_{12}-\rho_{21}\\ \rho_{12}-\rho_{21}&0\end{pmatrix}+\begin{pmatrix}\rho_{11}-\rho_{22}&0\\0&\rho_{11}-\rho_{22}\end{pmatrix}\right]\right)$$

$$=\tilde{\rho}$$

The equivalence of the two expressions is thus demonstrated.

Appendix 4

There follows a listing of the *domaptree* code referred to in chapter 5 page 211.

```
domaptree:=proc(f)
   local node; node:=0;
   tlevel:=tlevel+1;
   do;
      node:=node+1;
      totnodes[tlevel]:=totnodes[tlevel]+1;
      if node>nops(f) then
         totnodes[tlevel]:=totnodes[tlevel]-1;
         tlevel:=tlevel-1;
         break;    #Does node exist?    #
      fi;
      if not hastype(op(node,f),`+`) then
         typeinfo[tlevel,totnodes[tlevel]]:=[`S`,0];
         oprands[tlevel,totnodes[tlevel]]:=op(node,f);
         next;          # Go on to next node at this tlevel. #
      elif type(op(node,f),{`+`,`*`,`**`}) then
         typeinfo[tlevel, totnodes[tlevel]]:=[whattype(op(node,f)),
                  nops(op(node,f))];
         domaptree(op(node,f)); #Continue down this branch. #
      else
         typeinfo[tlevel, totnodes[tlevel]]:=[`S`,0];
         oprands[tlevel, totnodes[tlevel]]:=op(node,f); #BUG 22/8/91 Was :=f;
         next; # This branch is ok already. #
      fi;
   od;
   RETURN();
end;
```

Appendix 5

There follows a listing of the output produced by the program for the Boussinesq equation. The output has been edited to remove some of the "words used" messages.

```
    |\^/|
._|\|   |/|_. University of Lancaster - Computer Ctr
 \  MAPLE  /  Version 4.3 --- Mar 1989
 <____ ____>  For on-line help, type  help();
      |
# ================================================================ #
#                                                                  #
#                                                                  #
#                                                                  #
# (C) Tom Arbuckle 1989, 1990                                      #
#                                                                  #
#                                                                  #
#                                                                  #
# ================================================================ #
#                                                                  #
#                                                                  #
#                                                                  #
# This program is the first in a suite of programs designed to deal #
#                                                                  #
# with the process of obtaining the equations of motion from an    #
#                                                                  #
# averaged lagrangian. This first program accepts as input the file #
#                                                                  #
# AVERIN DAT A (AVERIN.DAT in MAPLE) and then performs validation  #
#                                                                  #
# on the contents of this file. It then goes on to perform suitable #
#                                                                  #
# expansions to a level specified to the user, finally passing the #
#                                                                  #
# results on to the next progam as a data file.                    #
#                                                                  #
# The file AVERIN.DAT should contain the following:                #
#                                                                  #
#    1) the variable errorswitch (set to true or false);           #
#                                                                  #
#    2) the variable prinlvl (set to an integer 0<=prinlvl<=4);    #
#                                                                  #
#    3) the lagrangian,L;                                          #
#                                                                  #
#    4) the required order of expansion, jmax;                     #
#                                                                  #
#    5) a set, varset, containing the variables used in L;         #
#                                                                  #
#    6) an optional set, vset, containing all array type quantities #
#                                                                  #
#           in use within the Lagrangian;                          #
#                                                                  #
#    7) a variable, hiharm, determining whether or not the expansion #
#                                                                  #
#           used will include harmonics higher than the fundamental. #
#                                                                  #
#           This variable should either be set to true or false.   #
#                                                                  #
#    8) a variable, groundstate, determining whether or not the    #
#                                                                  #
#           groundstate of a vector quantity will be treated as a  #
#                                                                  #
#           perturbation from unity. Normally set to true.         #
#                                                                  #
# If either of the first two variables are not assigned, they are  #
#                                                                  #
# given the default values of false and 0.                         #
#                                                                  #
# The variable hiharm will be given the value false if not assigned. #
#
```

```
# The variable groundstate has a default of true if not assigned.    #
#                                                                     #
#                                                                     #
#
# N.B.  The variable ok must be assigned false whenever an error     #
#
# occurs. A top level exit is required in order that the output      #
#
# from the help file be displayed. Hence checking to see whether     #
#
# things are ok will correctly terminate the program on error.       #
#
#                                                                     #
#
# ================================================================== #
#
##
#
> printlevel:=-1;                # adjust level of output              #
>
# printlevel:=1;
#
> prettyprint:=2:                  # left justify printed output       #
>
> words(0);                        # switch off words used messages    #
>
> read `/user10/xt/xtb003/mplfiles/averinc.dat`;    # READ AVERIN DAT A  #
> ok:=true;
>
> if assigned(errorswitch) then
>
>    if type(errorswitch, boolean) then
>
>        if errorswitch=false then
>
>            print(`Error reporting not required by user`);
>
>        elif errorswitch=true then
>
>            print(`Extended error reporting switched on.`);
>
>            print(`Detailed error messages will be given by the program`);
>
>        else
>
>            print(`The variable errorswitch must be assigned true or false`);
>
>            ok:=false;
>
>        fi;
>
>    else
>
>        print(`The variable errorswitch must be assigned true or false`);
>
>        ok:=false;
>
>    fi;
>
> fi;
                    Extended error reporting switched on.

             Detailed error messages will be given by the program
```

```
>
> if ok=false then stop fi:
>
# Previous line keeps ok as a variable to use later.                    #
#
# Now load in callerr as the error handling procedure                   #
#
# callerr defined as an unevaluated function call                       #
#
> callerr:='readlib('callerr',`/user10/xt/xtb003/mplfiles/callerr.m`)';
Warning: Recursive definition of name
>
> errhndlr:='readlib('errhndlr',`/user10/xt/xtb003/mplfiles/errhndlr.m`)';
Warning: Recursive definition of name
>
> errtable:='readlib('errtable',`/user10/xt/xtb003/mplfiles/errtable.m`)';
Warning: Recursive definition of name
>
> if assigned(vset) then
>
>    if type(vset,set) then
>
>       print(`WARNING: Vector or array quantities implied in input file.`);
>
>    else
>
>       ok:=false;
>
>       callerr(`Variable vset must be a set`,5)
>
>    fi;
>
> else
>
>    vset:={}:
>
> fi;
>
> if not ok then stop fi;
>
> if assigned(hiharm) then
>
>    if type(hiharm,boolean) then
>
>       if hiharm=true then
>
>          print(`WARNING: Higher harmonics used in this calculation.`)
>
>       elif hiharm=false then
>
>          print(`Expansion using lower harmonics only.`)
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable hiharm must be either true or false.`,8);
>
>    fi;
```

```
>
> else
>
>    hiharm:=false:
>
> fi;
                    Expansion using lower harmonics only.
>
> if not ok then stop fi;
>
> if assigned(groundstate) then
>
>    if type(groundstate,boolean) then
>
>       if groundstate=true then
>
>          print(`The groundstate of vector variables will be taken.`);
>
>          print(`to be a perturbation from unity.`);
>
>       elif groundstate=false then
>
>          print(`WARNING: The groundstate of any vector variables will`);
>
>          print(`         not be taken to be a special case.`);
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable groundstate must be either true or false.`,9);
>
>    fi;
>
> else
>
>    groundstate:=true:
>
> fi;
>
> if not ok then stop fi:
>
> kset:={anames()} minus
>
>     {'prettyprint','L','varset','JMAX','errorswitch','prinlvl','hiharm',
>
>      'ok','vset',`errhndlr`,`callerr`,`errtable`,'groundstate'}
>
>      minus vset minus
>
>     {`_initvalsNestlist`,`table/initvals`,`print/matrix`};
>
> if kset<>{} then
>
>    ok:=false:
>
>    callerr(`You have assigned superfluous variables`,kset,1)
>
```

```
> fi;
>
> if not ok then stop fi:
>
> kset:='kset':
>
> if assigned(prinlvl) then
>
>    if type(prinlvl,integer) then
>
>       if prinlvl>=0 and prinlvl<=4 then
>
>          print(`The level of printed output has been set to`,prinlvl);
>
>       else
>
>          ok:=false:
>
>          callerr(`Valid print levels lie between 0 and 4`,2);
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`Prinlvl must be set to an integer`,3);
>
>    fi;
>
> else
>
>    prinlvl:=0;          # Default printlevel set to 0                #
>
> fi;
                The level of printed output has been set to, 3

>
> if not ok then stop fi;
>
# Check to see that L contains the variables given in varset         #
#
> for var in varset while ok=true do
>
>    if not has(L,var) then ok:=false fi
>
> od;
>
> if not ok then
>
>    callerr(`Lagrangian, L, does not contain the variable`,var,4);
>
> fi;
>
> if not ok then stop fi:
>
> if vset<>{} then
>
>    kset:=varset intersect vset:
>
>    if kset={} then
>
>       ok:=false:
>
>       callerr(`Improper use of vector quantities`,6)
```

```
>
>    else
>
>       print(`The following variables have been defined as vectors: `);
>
>       print(kset)
>
>    fi;
>
> else
>
>    kset:={};
>
> fi;
>
> if not ok then stop fi:
>
> for var in (vset minus kset) do;
>
>    if not type(var,matrix) then
>
>      ok:=false;
>
>      callerr(`Incorrect matrix/array type definitions in input file`,7);
>
>    fi;
>
> od;
>
> if not ok then stop fi:
>
# nset is the set of non-vector expansion variables                    #
#
> nset:=varset minus kset:
>
# read `datestam.mpl`;
#
# datestamp();
#
> print(`The input file has now been validated.`);
                    The input file has now been validated.

>
> if prinlvl>2 then
>
>    print(`The program will now go on to perform the expansion`);
>
>    print(`required as a preliminary to the averaging p_ocedure.`);
>
> fi;
             The program will now go on to perform the expansion

            required as a preliminary to the averaging procedure.

>
> if prinlvl>1 then
>
>    print(`The lagrangian used will be:- `);
>
>    print(L);
>
>    if hiharm=true then
>
>       print(`Higher harmonics will be used in the substitution.`)
>
```

```
>    else
>
>       print(`Only the fundamental will be used in the calculation.`)
>
>    fi;
>
>    if nops(vset)<>0 then
>
>       print(`The calculation assumes that the following quantities are`);
>
>       print(`vector variables which will require subsequent expansion`);
>
>       print(kset);
>
>       print(` and that these variables are the scalar variables`);
>
>       print(` be expanded`);
>
>       print(nset);
>
>       print(`The following quantities are arrays which are`);
>
>       print(`assumed to have values given by the user.`);
>
>       print(vset minus kset);
>
>       if groundstate=true then
>
>          print(`The groundstate of vector quantities will be`);
>
>          print(`taken to be near unity.`);
>
>       else
>
>          print(`Note that the groundstate of vector quantities will`);
>
>          print(`NOT be taken to be a special case and will be expanded`);
>
>          print(`in a similar manner to all of the other variables.`):
>
>       fi;
>
>       print();
>
>    else
>
>       print(`These variables will be taken to be those with which the`);
>
>       print(`expansion is to be performed  and with respect to which`);
>
>       print(`the subsequent variation is to be carried out.`);
>
>       print(nset); print();  # nset=varset for all scalars. #
>
>    fi;
>
> fi;
```

The lagrangian used will be:-

$$\frac{1}{2}\,dt(f)^2 - \frac{1}{2}\,dx(f)^2\,c^2 + \frac{1}{2}\,mu\,dx(dt(f))^2 - \frac{1}{6}\,dx(f)^3$$

Only the fundamental will be used in the calculation.

These variables will be taken to be those with which the

```
            expansion is to be performed  and with respect to which
                 the subsequent variation is to be carried out.
                                        {f}


>
> if prinlvl>2 then
>
>    print(`Calculation begins ...`);
>
> fi;
                              Calculation begins ...

>
# Convert sets to lists to maintain ordering.                          #
#
> nlist:=convert(nset,list); klist:=convert(kset,list);
>
# nlist:=[op(nset)]: klist:=[op(kset)]:
#
> tb:=table(): tc:=table():
>
> for i from 1 to nops(nlist) do;
>
>    assign(tb[i]=cat(nlist[i],_));
>
> od;
>
> for i from 1 to nops(klist) do;
>
>    assign(tc[i]=cat(klist[i],_));
>
> od;
>
> newnlist:=convert(tb,list):
>
> newklist:=convert(tc,list):
>
# op(newklist); op(newnlist);
#
> tb:='tb': tc:='tc':   # Unassign tb, tc                              #
>
> read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expdstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddson.mpl`;
>
#read `expdnc.mpl`;
> read `/user10/xt/xtb003/mplfiles/diffdtdx.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
>
# read `truncser.mpl`;
#
```

```
> read `/user10/xt/xtb003/mplfiles/orderser.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/sums.mpl`: # get summation procedures.
>   #
> read `/user10/xt/xtb003/mplfiles/average.mpl`;
>
# # read `smalexpd.mpl`; # Avoid `object too big.`                #
#
> read `/user10/xt/xtb003/mplfiles/smrtexpd.mpl`;
>
> readlib(evalm):
>
# Simplify the lagrangian to get rid of star(star(. etc.
#
> L:=simplify(L,dagger);
>
> L:=simplify(L,star);
>
> if prinlvl>=3 then
>    print(`The lagrangian has now been simplifed with respect to dagger`);
>
>    print(`and star.`);
> fi;
          The lagrangian has now been simplifed with respect to dagger

                                    and star.

> if prinlvl=4 then print(`The lagrangian is now`,L) fi;
> if klist<>[] then
>
>    for i from 1 to nops(klist) do;
>
>       assign(op(i,newklist)=array( op(2,op(op(i,klist)))) );
>
>    od;
>
>    if hiharm=false then
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),quantsum(op(i,newklist)));
>
>       od;
>
>    else
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),bigquantsum(op(i,newklist)));
>
>       od;
>
>    fi;
>
> fi;
>
> if hiharm=false then
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),weeharmsum(op(i,newnlist)));
>
>    od;
>
> else
```

```
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),bigharmsum(op(i,newnlist)));
>
>    od;
>
> fi;
>
# read in the Diff procedure now that the substitutions have been made.
#
> read `/user10/xt/xtb003/mplfiles/diftheta.mpl`;
>
> L:=simplify(L,DT,DX);
bytes used=1000176, alloc=360448, time=16.200
bytes used=2002104, alloc=425984, time=24.666
bytes used=3003588, alloc=507904, time=33.250
bytes used=4003988, alloc=614400, time=43.966
bytes used=5005104, alloc=614400, time=55.016
bytes used=6005352, alloc=614400, time=65.366
bytes used=7005844, alloc=614400, time=76.016
bytes used=8006192, alloc=614400, time=86.400
bytes used=9006432, alloc=614400, time=97.183
bytes used=10006860, alloc=638976, time=107.866
bytes used=11008780, alloc=638976, time=118.150
bytes used=12008904, alloc=638976, time=129.083
bytes used=13009688, alloc=655360, time=139.583
bytes used=14010972, alloc=655360, time=147.633
bytes used=15012780, alloc=655360, time=158.516
bytes used=16014084, alloc=655360, time=166.550
                                      test1

bytes used=17016192, alloc=655360, time=175.100
bytes used=18016572, alloc=786432, time=183.883
bytes used=19018884, alloc=917504, time=192.866
bytes used=20020116, alloc=917504, time=202.016
bytes used=21021044, alloc=925696, time=211.300
                                      test1

bytes used=22023344, alloc=925696, time=220.566
bytes used=23025196, alloc=925696, time=233.483
bytes used=24028040, alloc=925696, time=244.283
                                      test1

bytes used=25029792, alloc=925696, time=255.066
bytes used=26031664, alloc=925696, time=265.166
bytes used=27032488, alloc=1064960, time=274.983
bytes used=28033264, alloc=1204224, time=285.483
bytes used=29036128, alloc=1261568, time=296.233
bytes used=30041288, alloc=1261568, time=306.733
bytes used=31042268, alloc=1261568, time=317.133
bytes used=32044532, alloc=1261568, time=328.100
bytes used=33044840, alloc=1261568, time=338.216
bytes used=34045420, alloc=1261568, time=348.650
bytes used=35046396, alloc=1261568, time=358.800
bytes used=36046560, alloc=1261568, time=369.433
bytes used=37049096, alloc=1261568, time=379.900
bytes used=38051504, alloc=1261568, time=390.883
bytes used=39051680, alloc=1261568, time=401.083
bytes used=40051812, alloc=1261568, time=411.466
bytes used=41052352, alloc=1261568, time=421.666
bytes used=42053076, alloc=1261568, time=431.950
bytes used=43054872, alloc=1261568, time=442.466
bytes used=44055192, alloc=1261568, time=453.700
bytes used=45055528, alloc=1261568, time=463.966
```

```
bytes used=46056512, alloc=1261568, time=474.433
bytes used=47056976, alloc=1261568, time=484.633
bytes used=48057740, alloc=1261568, time=495.150
bytes used=49058772, alloc=1261568, time=505.666
bytes used=50061272, alloc=1261568, time=516.633
bytes used=51062612, alloc=1261568, time=526.766
bytes used=52063800, alloc=1261568, time=537.333
bytes used=53063928, alloc=1261568, time=547.883
bytes used=54064128, alloc=1261568, time=558.500
bytes used=55064940, alloc=1261568, time=568.933
                                     test1

bytes used=56065380, alloc=1261568, time=578.583
bytes used=57066280, alloc=1261568, time=587.116
>
# Should now have made the necessary assignments. Matrices should not be
#
# empty so can carry out necessary calculations.
#
#expand(expandoff); expand(expandon); expandoff(exp);
#
#printlevel:=2000;
#
> if prinlvl>=3 then
>
>    print(`The arrays have now been filled and the summations`);
>
>    print(`required for the expansion carried out.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian has become: `,L);
>
>    fi;
>
> fi;
              The arrays have now been filled and the summations

                   required for the expansion carried out.

>
# quit;
#
> L:=expddagstonly(L);
>
> if prinlvl>=3 then
>    print(`The lagrangian has been expanded with repsect to its`);
>
>    print(`dagger and star operations only.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian becomes: `);
>
>       lprint(L);
>
>    fi;
>
> fi;
             The lagrangian has been expanded with repsect to its

                       dagger and star operations only.

>
# If simply do expand then will fail when dagger is mapped onto
```

```
#
# matrices. Transpose will alter shape remember.
#
> L:=simplify(L,DT,DX,star); # simplify wrt DT,DX,star only.
bytes used=58068004, alloc=1261568, time=607.333
bytes used=59069180, alloc=1261568, time=615.733
bytes used=60069632, alloc=1261568, time=624.183
bytes used=61070712, alloc=1261568, time=633.216
bytes used=62071500, alloc=1261568, time=642.183
bytes used=63072932, alloc=1261568, time=651.283
bytes used=64075252, alloc=1261568, time=660.466
bytes used=65075396, alloc=1261568, time=678.816
bytes used=66076148, alloc=1261568, time=687.483
bytes used=67077048, alloc=1261568, time=697.066
bytes used=68077296, alloc=1261568, time=705.466
bytes used=69077876, alloc=1261568, time=714.466
bytes used=70079692, alloc=1261568, time=723.316
bytes used=71082452, alloc=1261568, time=732.083
bytes used=72084092, alloc=1261568, time=741.033
bytes used=73087008, alloc=1261568, time=749.916
bytes used=74088812, alloc=1261568, time=759.250
bytes used=75089108, alloc=1261568, time=767.983
bytes used=76089924, alloc=1261568, time=776.833
bytes used=77090864, alloc=1261568, time=785.666
bytes used=78092948, alloc=1261568, time=794.516
bytes used=79093240, alloc=1261568, time=803.550
bytes used=80094812, alloc=1261568, time=812.000
bytes used=81099776, alloc=1261568, time=821.033
bytes used=82102356, alloc=1261568, time=830.550
bytes used=83103668, alloc=1261568, time=838.550
bytes used=84105296, alloc=1261568, time=847.700
bytes used=85105656, alloc=1261568, time=856.533
bytes used=86106548, alloc=1261568, time=865.300
bytes used=87108188, alloc=1261568, time=874.183
bytes used=88109280, alloc=1261568, time=883.150
bytes used=89110968, alloc=1261568, time=892.433
bytes used=90111264, alloc=1261568, time=901.033
bytes used=91112080, alloc=1261568, time=909.833
bytes used=92112904, alloc=1261568, time=918.483
bytes used=93114988, alloc=1261568, time=927.316
bytes used=94115280, alloc=1261568, time=936.150
bytes used=95116852, alloc=1261568, time=944.883
bytes used=96121816, alloc=1261568, time=953.950
bytes used=97123712, alloc=1261568, time=962.633
bytes used=98124940, alloc=1261568, time=970.966
bytes used=99126476, alloc=1261568, time=979.766
bytes used=100126916, alloc=1261568, time=988.833
bytes used=101130324, alloc=1261568, time=997.633
bytes used=102131740, alloc=1261568, time=1007.800
bytes used=103133560, alloc=1261568, time=1015.833
                                      test1

bytes used=104135668, alloc=1261568, time=1024.833
bytes used=105137096, alloc=1261568, time=1033.733
bytes used=106141464, alloc=1261568, time=1042.983
bytes used=107141904, alloc=1261568, time=1052.016
bytes used=108142324, alloc=1261568, time=1061.283
                                      test1

bytes used=109145872, alloc=1261568, time=1070.433
bytes used=110148044, alloc=1261568, time=1083.933
bytes used=111150772, alloc=1261568, time=1095.316
                                      test1
```

```
bytes used=472046240, alloc=2514944, time=7044.516
bytes used=473046424, alloc=2514944, time=7059.083
bytes used=474046632, alloc=2514944, time=7073.716
bytes used=475046948, alloc=2514944, time=7088.133
bytes used=476047288, alloc=2514944, time=7102.583
bytes used=477047656, alloc=2514944, time=7117.266
bytes used=478048076, alloc=2514944, time=7131.450
bytes used=479048460, alloc=2514944, time=7145.933
bytes used=480048736, alloc=2514944, time=7160.166
bytes used=481049148, alloc=2514944, time=7174.916
bytes used=482049508, alloc=2514944, time=7189.083
bytes used=483049700, alloc=2514944, time=7203.266
bytes used=484050004, alloc=2514944, time=7217.950
bytes used=485050236, alloc=2514944, time=7232.216
bytes used=486052876, alloc=2514944, time=7247.033
bytes used=487053068, alloc=2514944, time=7261.433
bytes used=488053324, alloc=2514944, time=7274.866
bytes used=489053760, alloc=2514944, time=7287.683
bytes used=490054140, alloc=2514944, time=7301.183
bytes used=491054368, alloc=2514944, time=7314.033
bytes used=492054648, alloc=2514944, time=7327.433
bytes used=493054824, alloc=2514944, time=7340.483
                      Completely simplified at level: , 2

bytes used=494055068, alloc=2514944, time=7353.283
bytes used=495055308, alloc=2514944, time=7365.416
                      Completely simplified at level: , 1

                  Sucessfully simplified within operands limit.

#L:=collect(L,eps,recursive,expand);
# Truncate the expanded lagrangian.
#
> L:=seriestrunc(L,[eps=1],JMAX);
bytes used=496055584, alloc=2514944, time=7378.366
bytes used=497056160, alloc=2514944, time=7390.116
bytes used=498057584, alloc=2514944, time=7402.516
> if prinlvl=4 then
>
>    print(`The lagrangian has been expanded and simplified.`);
>
>    print(`Its final form is: `);
>
>    lprint(L);
>
> fi;
>
# Collect terms in `eps`                #
#
> L:=collect(L,eps):
>
> if prinlvl>=3 then
>
>    print(`The lagrangian has been rearranged in terms of the small`);
>
>    print(`variable eps.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian is now: `);
>
>       lprint(L);
>
>    fi;
```

```
>
> fi;
            The lagrangian has been rearranged in terms of the small

                                  variable eps.

>
> save `/scratch/xt/xtb003/t2/avlagq4t.m`;
>
# lprint(L);
> for i from 0 to JMAX do; #4 do;
>    L.i:=coeff(L,`eps`,i);
>
>    L.i.`bar`:=average(L.i);
>
>    LC.i.`bar`:=evalc(L.i.`bar`);
>    print(`The averaged lagrangian at order`,i,` is:`);
>
>    print(LC.i.`bar`);
> od;
                    The averaged lagrangian at order, 0,  is:

                                          0

                    The averaged lagrangian at order, 1,  is:

                                          0

                    The averaged lagrangian at order, 2,  is:

                2  2                                  2
           - c  k  star(f_[0, 1]) f_[0, 1] + w  star(f_[0, 1]) f_[0, 1]

                      2  2
                 + mu k  w  star(f_[0, 1]) f_[0, 1]

                    The averaged lagrangian at order, 3,  is:

     2  2                                2  2                  2
  - c  k  star(f_[1, 1]) f_[0, 1] - c  k  f_[1, 1] %1 + w  f_[1, 1] %1

           2  2                  2
      + mu k  w  f_[1, 1] %1 + w  star(f_[1, 1]) f_[0, 1]

           2  2                                2
      + mu k  w  star(f_[1, 1]) f_[0, 1] + (- c  star(DX(%2)) k f_[0, 1]

                                                          2
      - w f_[0, 1] star(DT(%2)) + DT(%2) w %1 + mu w  star(DX(%2)) k f_[0, 1]

           2                2                  2
      + mu k  DT(%2) w %1 + c  DX(%2) k %1 - mu w  DX(%2) k %1

           2                                  2
      - mu k  w f_[0, 1] star(DT(%2)) + mu w  %1 k f_[0, 1]

           2
      - mu k  %1 w f_[0, 1]) I

%1 :=
                                star(f_[0, 1])

%2 :=
                                  f_[0, 1], 1
```

```
bytes used=499058472, alloc=2514944, time=7426.300
                The averaged lagrangian at order, 4,  is:

   2  2                    2
- c  k  %3 f_[1, 1] + (- c  star(DX(%5)) k f_[0, 1]

        2                              2                      2
     - c  star(DX(%4)) k f_[0, 1] + c  f_[1, 1] k %2 + c  DX(%1) k %3

        2                    2                    2
     + c  DX(%4) k %2 + c  DX(%5) k %2 - mu w  DX(%4) k %2

          2                                  2
     - mu k  star(DT(%5)) w f_[0, 1] + mu w  star(DX(%4)) k f_[0, 1]

          2                                  2
     + mu w  star(DX(%5)) k f_[0, 1] + mu k  DT(%5) w %2

          2                          2                      2
     + mu w  star(DX(%1)) k f_[1, 1] - c  %3 k f_[0, 1] + mu k  DT(%1) w %3

                        2                      2
     + DT(%1) w %3 + mu k  DT(%4) w %2 - 2 mu k  %3 w f_[0, 1]

          2                    2
     - mu w  DX(%1) k %3 - mu k  %2 w f_[0, 1] - %3 w f_[0, 1]

          2                    2
     - mu w  DX(%5) k %2 - mu k  star(DT(%1)) w f_[1, 1]

                                    2
     - star(DT(%1)) w f_[1, 1] - mu k  star(DT(%4)) w f_[0, 1]

                                  2
     - star(DT(%4)) w f_[0, 1] + 2 mu w  %3 k f_[0, 1]

     - star(DT(%5)) w f_[0, 1] + f_[1, 1] w %2 + DT(%5) w %2 + DT(%4) w %2

          2                   2
     + mu w  %2 k f_[0, 1] - c  star(DX(%1)) k f_[1, 1]) I

          2  2
     + mu k  w  %3 f_[1, 1] - mu w f_[0, 1] k star(DT(%1))

     - mu w DX(%1) k star(DT(%1)) - mu w star(DX(%1)) k DT(%1)

          2  2
     + mu k  w  f_[0, 1] star(f_[2, 1]) - 2 mu w %2 k DT(%1)

                                        2  2
     + mu star(DX(DT(%1), 1)) k w f_[0, 1] - c  k  f_[2, 1] %2

                                   2  2
     + mu DX(DT(%1), 1) k w %2 - c  k  f_[0, 1] star(f_[2, 1])

          2  2                    2
     + mu k  w  f_[2, 1] %2 + w  f_[2, 1] %2 - mu w %2 k f_[0, 1]

                           2                                  2
     - mu w DX(%1) k %2 + w  f_[0, 1] star(f_[2, 1]) + w  %3 f_[1, 1]

           2                                        2
     + 2 mu w  %2 DX(%1) + DT(%1) star(DT(%1)) + mu k  DT(%1) %2

        2                           2                              2
```

```
      - c  star(DX(%1)) DX(%1) + mu k  DT(%1) star(DT(%1)) + 2 mu w  %2 f_[0, 1]

           2                                  2
      + mu w  star(DX(%1)) DX(%1) + mu w  star(DX(%1)) f_[0, 1]

%1 :=
                                    f_[0, 1], 1

%2 :=
                                  star(f_[0, 1])

%3 :=
                                  star(f_[1, 1])

%4 :=
                                    f_[0, 1], 2

%5 :=
                                    f_[1, 1], 1

>
> save '/scratch/xt/xtb003/t2/tavlag4q.m';
# quit;
#
# This was originally file eqnmotq2.mpl
# trying to make the equations of motion. #
# JMAX:=6; prinlvl:=4;
# v_:=array(1..2,[]); #array bound required for looping.
# newnlist:=[eta_]; newklist:=[v_]; hiharm:=false;
# LC0bar:=0; LC1bar:=0;
# read '/user10/xt/xtb003/results/averq2.m';
# printlevel:=-1;
# Define LC(2-4)bar;
> read '/user10/xt/xtb003/mplfiles/ordindex.mpl';
# read '/user10/xt/xtb003/mplfiles/typedef.mpl';
> read '/user10/xt/xtb003/mplfiles/dordbool.mpl';
> read '/user10/xt/xtb003/mplfiles/mapmod.mpl';
> read '/user10/xt/xtb003/mplfiles/eqnmot.mpl';
> read '/user10/xt/xtb003/mplfiles/derivord.mpl';
> read '/user10/xt/xtb003/mplfiles/makeELterm.mpl';
> read '/user10/xt/xtb003/mplfiles/doprint2.mpl';
>
> read '/user10/xt/xtb003/mplfiles/simpstar.mpl';
> read '/user10/xt/xtb003/mplfiles/simpdtdx.mpl';
# Have now read in necessary files.
>
> for levcount from 0 to JMAX do;
>    equationsofmotion('LC'.levcount.'bar');
# Create DTvar, DXvar, DXDTvar as sets.
>    DTvar:=convert(DTvar,'list');
>    DXvar:=convert(DXvar,'list');
>    DXDTvar:=convert(DXDTvar,'list');
>
>    scalvar:=convert(scalvar,'list');
>
>    vecvar:=convert(vecvar,'list');
>
>    newDTvar:=sort(DTvar,dordbool);
>
>    newDXvar:=sort(DXvar,dordbool);
>
>    newDXDTvar:=sort(DXDTvar,dordbool);
>
#   printlevel:=2000;
>    newscalvar:=sort(scalvar,fordbool);
```

```
>    newvecvar:=sort(vecvar,fordbool);
#   printlevel:=-1;
>    if prinlvl>3 then
>       print(`op(newscalvar)`,op(newscalvar));
>       print(`op(newvecvar)`,op(newvecvar) );
>       print(`op(newDTvar)`,op(newDTvar));
>       print(`op(newDXvar)`,op(newDXvar));
>       print(`op(newDXDTvar)`,op(newDXDTvar));
>    fi;
# printlevel:=2000;
>    lprint(`The lagrangian being used is :-`);
>    lprint(`LC`.levcount.`bar`);
# Generating the equations starts here.
>    scalareqnset:={}; vectoreqnset:={}; # Put the names of the arrays
# which will hold the equations in these two sets.
>    if nops(newscalvar)<>0 then
#      print(newscalvar,`TATEST3`);
>       for var in newscalvar do;
# Generate Euler-Lagrange for scalar variables.
>       print(`Investigating variable `,var,`at level`,levcount);
>       scalname:=op(0,var);
>       if not assigned(`eqtn`.levcount.scalname) then
>          scalareqnset:=scalareqnset union {`eqtn`.levcount.scalname};
>          `eqtn`.levcount.scalname:=table(sparse,[]) fi;
#      printlevel:=101; print(`101 #5`);
>       `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>           makeELterm(var,`LC`.levcount.`bar`);
#      printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #6`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #7`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #8`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       od;
#      print(`eqtn`.levcount.op(0,var)[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
```

```
>    if nops(newvecvar)<>0 then
#      print(newvecvar,`TATEST4`);
>       for var in newvecvar do;
#         printlevel:=11;
# Generate Euler-Lagrange for vector variables.
>       print(`Investigating vector variable`,var,`at level`,levcount);
>       vecname:=op(0,var);
>       if not assigned(`eqtn`.levcount.vecname) then
>          vectoreqnset:=vectoreqnset union {`eqtn`.levcount.vecname};
>          `eqtn`.levcount.vecname:=table(sparse,[]) fi;
# printlevel:=101; print(`101 #1`);
>       `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>           makeELterm(var,`LC`.levcount.`bar`);
# printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#              printlevel:=101;print(`101 #2`);
>               `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                  `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                  makeELterm(var2,`LC`.levcount.`bar`);
#              printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#         printlevl:=101; print(`101 #3`);
>               `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                  `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                  makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
#              printlvel:=101; print(`101 #4`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#     printlevel:=-1;
>             fi;
>          od;
>       fi;
>       od;
#      printlevel:=-1;
#      print(`eqtn`.levcount.vecname[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
#   printlevel:=11;
>    doprinteqns();
> od; # end of outer loop.
The lagrangian being used is :-
0
The lagrangian being used is :-
0
The lagrangian being used is :-
-c**2*k**2*star(f_[0,1])*f_[0,1]+w**2*star(f_[0,1])*f_[0,1]+mu*k**2*w**2*star(
f_[0,1])*f_[0,1]
                Investigating variable , f_[0, 1], at level, 2

            The equations of motion for the scalar variables are:-
```

```
                The equation of motion for , f_[0,1],  is:-

                    2  2              2                 2  2
                - c  k  f_[0, 1] + w  f_[0, 1] + mu k  w  f_[0, 1], =0


bytes used=500059236, alloc=2514944, time=7448.750
The lagrangian being used is :-
-c**2*k**2*star(f_[1,1])*f_[0,1]-c**2*k**2*f_[1,1]*star(f_[0,1])+w**2*f_[1,1]*
star(f_[0,1])+mu*k**2*w**2*f_[1,1]*star(f_[0,1])+w**2*star(f_[1,1])*f_[0,1]+mu*
k**2*w**2*star(f_[1,1])*f_[0,1]+(-c**2*star(DX(f_[0,1],1))*k*f_[0,1]-w*f_[0,1]*
star(DT(f_[0,1],1))+DT(f_[0,1],1)*w*star(f_[0,1])+mu*w**2*star(DX(f_[0,1],1))*k
*f_[0,1]+mu*k**2*DT(f_[0,1],1)*w*star(f_[0,1])+c**2*DX(f_[0,1],1)*k*star(f_[0,1
])-mu*w**2*DX(f_[0,1],1)*k*star(f_[0,1])-mu*k**2*w*f_[0,1]*star(DT(f_[0,1],1))+
mu*w**2*star(f_[0,1])*k*f_[0,1]-mu*k**2*star(f_[0,1])*w*f_[0,1])*I
                Investigating variable , f_[0, 1], at level, 3

                Investigating variable , f_[1, 1], at level, 3

            The equations of motion for the scalar variables are:-


                The equation of motion for , f_[1,1],  is:-

                    2  2              2                 2  2
                - c  k  f_[0, 1] + w  f_[0, 1] + mu k  w  f_[0, 1], =0


                The equation of motion for , f_[0,1],  is:-

   2  2              2                 2  2                                  2
- c  k  f_[1, 1] + w  f_[1, 1] + mu k  w  f_[1, 1] + (DT(%1) w + mu k  DT(%1) w

       2                2                2                    2
    + c  DX(%1) k - mu w  DX(%1) k + mu w  k f_[0, 1] - mu k  w f_[0, 1]) I

                                      2                   2
    - (- w f_[0, 1] - DT(%1) w - mu k  w f_[0, 1] - mu k  DT(%1) w) I

          2                2                2
    - (- c  k f_[0, 1] - c  DX(%1) k + mu w  DX(%1) k) I,

   =0

%1 :=
                                  f_[0, 1], 1


bytes used=501059520, alloc=2514944, time=7458.300
The lagrangian being used is :-
-c**2*k**2*star(f_[1,1])*f_[1,1]+(-c**2*star(DX(f_[1,1],1))*k*f_[0,1]-c**2*star
(DX(f_[0,1],2))*k*f_[0,1]+c**2*f_[1,1]*k*star(f_[0,1])+c**2*DX(f_[0,1],1)*k*
star(f_[1,1])+c**2*DX(f_[0,1],2)*k*star(f_[0,1])+c**2*DX(f_[1,1],1)*k*star(f_[0
,1])-mu*w**2*DX(f_[0,1],2)*k*star(f_[0,1])-mu*k**2*star(DT(f_[1,1],1))*w*f_[0,1
]+mu*w**2*star(DX(f_[0,1],2))*k*f_[0,1]+mu*w**2*star(DX(f_[1,1],1))*k*f_[0,1]+
mu*k**2*DT(f_[1,1],1)*w*star(f_[0,1])+mu*w**2*star(DX(f_[0,1],1))*k*f_[1,1]-c**
2*star(f_[1,1])*k*f_[0,1]+mu*k**2*DT(f_[0,1],1)*w*star(f_[1,1])+DT(f_[0,1],1)*w
*star(f_[1,1])+mu*k**2*DT(f_[0,1],2)*w*star(f_[0,1])-2*mu*k**2*star(f_[1,1])*w*
```

```
f_[0,1]-mu*w**2*DX(f_[0,1],1)*k*star(f_[1,1])-mu*k**2*star(f_[0,1])*w*f_[0,1]-
star(f_[1,1])*w*f_[0,1]-mu*w**2*DX(f_[1,1],1)*k*star(f_[0,1])-mu*k**2*star(DT(
f_[0,1],1))*w*f_[1,1]-star(DT(f_[0,1],1))*w*f_[1,1]-mu*k**2*star(DT(f_[0,1],2))
*w*f_[0,1]-star(DT(f_[0,1],2))*w*f_[0,1]+2*mu*w**2*star(f_[1,1])*k*f_[0,1]-star
(DT(f_[1,1],1))*w*f_[0,1]+f_[1,1]*w*star(f_[0,1])+DT(f_[1,1],1)*w*star(f_[0,1])
+DT(f_[0,1],2)*w*star(f_[0,1])+mu*w**2*star(f_[0,1])*k*f_[0,1]-c**2*star(DX(f_[
0,1],1))*k*f_[1,1])*I+mu*k**2*w**2*star(f_[1,1])*f_[1,1]-mu*w*f_[0,1]*k*star(DT
(f_[0,1],1))-mu*w*DX(f_[0,1],1)*k*star(DT(f_[0,1],1))-mu*w*star(DX(f_[0,1],1))*
k*DT(f_[0,1],1)+mu*k**2*w**2*f_[0,1]*star(f_[2,1])-2*mu*w*star(f_[0,1])*k*DT(f_
[0,1],1)+mu*star(DX(DT(f_[0,1],1),1))*k*w*f_[0,1]-c**2*k**2*f_[2,1]*star(f_[0,1
])+mu*DX(DT(f_[0,1],1),1)*k*w*star(f_[0,1])-c**2*k**2*f_[0,1]*star(f_[2,1])+mu*
k**2*w**2*f_[2,1]*star(f_[0,1])+w**2*f_[2,1]*star(f_[0,1])-mu*w*star(f_[0,1])*k
*f_[0,1]-mu*w*DX(f_[0,1],1)*k*star(f_[0,1])+w**2*f_[0,1]*star(f_[2,1])+w**2*
star(f_[1,1])*f_[1,1]+2*mu*w**2*star(f_[0,1])*DX(f_[0,1],1)+DT(f_[0,1],1)*star(
DT(f_[0,1],1))+mu*k**2*DT(f_[0,1],1)*star(f_[0,1])-c**2*star(DX(f_[0,1],1))*DX(
f_[0,1],1)+mu*k**2*DT(f_[0,1],1)*star(DT(f_[0,1],1))+2*mu*w**2*star(f_[0,1])*f_
[0,1]+mu*w**2*star(DX(f_[0,1],1))*DX(f_[0,1],1)+mu*w**2*star(DX(f_[0,1],1))*f_[
0,1]
```

Investigating variable , f_[0, 1], at level, 4

Investigating variable , f_[1, 1], at level, 4

Investigating variable , f_[2, 1], at level, 4

The equations of motion for the scalar variables are:-

The equation of motion for , f_[2,1], is:-

$$- c^2 k^2 f_{[0,1]} + w^2 f_{[0,1]} + mu\, k^2 w^2 f_{[0,1]}, =0$$

The equation of motion for , f_[1,1], is:-

$$\begin{aligned} &mu\, k^2 w^2 f_{[1,1]} + (2\, mu\, w^2 k\, f_{[0,1]} + DT(\%1)\, w - mu\, w^2\, DX(\%1)\, k \\ &\quad - w\, f_{[0,1]} - 2\, mu\, k^2 w\, f_{[0,1]} + mu\, k^2\, DT(\%1)\, w - c^2 k\, f_{[0,1]} \\ &\quad + c^2\, DX(\%1)\, k)\, I - c^2 k^2 f_{[1,1]} + w^2 f_{[1,1]} \\ &\quad - (- w\, f_{[0,1]} - DT(\%1)\, w - mu\, k^2 w\, f_{[0,1]} - mu\, k^2\, DT(\%1)\, w)\, I \\ &\quad - (- c^2 k\, f_{[0,1]} - c^2\, DX(\%1)\, k + mu\, w^2\, DX(\%1)\, k)\, I, \qquad =0 \end{aligned}$$

%1 :=

f_[0, 1], 1

The equation of motion for , f_[0,1], is:-

$$\begin{aligned} &2\, mu\, w^2 f_{[0,1]} + mu\, w^2\, DX(\%4) + mu\, k^2 w^2 f_{[2,1]} + 4\, mu\, DX(DT(\%4), 1)\, k\, w \\ &\quad - c^2 k^2 f_{[2,1]} + mu\, k^2\, DT(\%4) + (- mu\, w^2\, DX(\%2)\, k - mu\, k^2 w\, f_{[0,1]} \end{aligned}$$

```
          2                    2                  2                  2
+ mu w  k f_[0, 1] + mu k  DT(%1) w + mu k  DT(%2) w - mu w  DX(%1) k

   2             2             2
+ c  DX(%2) k + c  DX(%1) k + c  f_[1, 1] k + f_[1, 1] w + DT(%2) w

                      2                  2
+ DT(%1) w) I + w  f_[2, 1] - mu k  DT(%3) - DT(%3)

          2                    2
- (- mu k  w f_[1, 1] - mu k  DT(%2) w - f_[1, 1] w - DT(%2) w) I

          2                    2
- (- mu k  w f_[0, 1] - mu k  DT(%1) w - w f_[0, 1] - DT(%1) w) I

        2            2           2
- mu w  DX(%3) + c  DX(%4) + c  DX(%3)

         2              2              2
- (mu w  DX(%2) k - c  f_[1, 1] k - c  DX(%2) k) I

         2              2              2
- (mu w  DX(%1) k - c  k f_[0, 1] - c  DX(%1) k) I,                    =0
```

```
%1 :=
                                f_[0, 1], 2

%2 :=
                                f_[1, 1], 1

%3 :=
                               f_[0, 1], 1, 1

%4 :=
                                f_[0, 1], 1
```

```
> save `/scratch/xt/xtb003/t2/eqnmotq.m`;
bytes used=502587224, alloc=2514944, time=7482.933
> quit;
bytes used=502587404, alloc=2514944, time=7482.933
```

Appendix 6

There follows a listing of the output produced by the program for the classical lagrangian due to Small. The output has been edited to remove some of the "words used" messages.

```
    |\^/|
._|\|   |/|_. University of Lancaster - Computer Ctr
 \  MAPLE  /  Version 4.3 --- Mar 1989
 <____ ____>  For on-line help, type  help();
      |
# ================================================================== #
#                                                                    #
#                                                                    #
# (C) Tom Arbuckle 1989, 1990                                        #
#                                                                    #
#                                                                    #
# ================================================================== #
#                                                                    #
#                                                                    #
# This program is the first in a suite of programs designed to deal  #
#                                                                    #
# with the process of obtaining the equations of motion from an      #
#                                                                    #
# averaged lagrangian. This first program accepts as input the file  #
#                                                                    #
# AVERIN DAT A (AVERIN.DAT in MAPLE) and then performs validation    #
#                                                                    #
# on the contents of this file. It then goes on to perform suitable  #
#                                                                    #
# expansions to a level specified to the user, finally passing the   #
#                                                                    #
# results on to the next progam as a data file.                      #
#                                                                    #
# The file AVERIN.DAT should contain the following:                  #
#                                                                    #
#    1) the variable errorswitch (set to true or false);             #
#                                                                    #
#    2) the variable prinlvl (set to an integer 0<=prinlvl<=4);      #
#                                                                    #
#    3) the lagrangian,L;                                            #
#                                                                    #
#    4) the required order of expansion, jmax;                       #
#                                                                    #
#    5) a set, varset, containing the variables used in L;           #
#                                                                    #
#    6) an optional set, vset, containing all array type quantities  #
#                                                                    #
#           in use within the Lagrangian;                            #
#                                                                    #
#    7) a variable, hiharm, determining whether or not the expansion #
#                                                                    #
#           used will include harmonics higher than the fundamental. #
#                                                                    #
#           This variable should either be set to true or false.     #
#                                                                    #
#    8) a variable, groundstate, determining whether or not the      #
#                                                                    #
#           groundstate of a vector quantity will be treated as a    #
#                                                                    #
#           perturbation from unity. Normally set to true.           #
#                                                                    #
# If either of the first two variables are not assigned, they are    #
#                                                                    #
# given the default values of false and 0.                           #
#                                                                    #
# The variable hiharm will be given the value false if not assigned. #
#
```

```
# The variable groundstate has a default of true if not assigned.    #
#                                                                     #
#                                                                     #
#
# N.B.  The variable ok must be assigned false whenever an error     #
#
# occurs. A top level exit is required in order that the output      #
#
# from the help file be displayed. Hence checking to see whether     #
#
# things are ok will correctly terminate the program on error.       #
#
#                                                                     #
#
# ================================================================== #
#
##
#
> printlevel:=-1;                # adjust level of output              #
>
# printlevel:=1;
#
> prettyprint:=2:                  # left justify printed output       #
>
> words(0);                        # switch off words used messages    #
>
> read `/user10/xt/xtb003/mplfiles/avercl.dat`;    # READ AVERIN DAT A  #
> ok:=true;
>
> if assigned(errorswitch) then
>
>    if type(errorswitch, boolean) then
>
>        if errorswitch=false then
>
>            print(`Error reporting not required by user`);
>
>        elif errorswitch=true then
>
>            print(`Extended error reporting switched on.`);
>
>            print(`Detailed error messages will be given by the program`);
>
>        else
>
>            print(`The variable errorswitch must be assigned true or false`);
>
>            ok:=false;
>
>        fi;
>
>     else
>
>         print(`The variable errorswitch must be assigned true or false`);
>
>         ok:=false;
>
>     fi;
>
> fi;
                    Extended error reporting switched on.

             Detailed error messages will be given by the program
```

```
>
> if ok=false then stop fi:
>
# Previous line keeps ok as a variable to use later.                    #
#
# Now load in callerr as the error handling procedure                   #
#
# callerr defined as an unevaluated function call                       #
#
> callerr:='readlib('callerr',`/user10/xt/xtb003/mplfiles/callerr.m`)';
Warning: Recursive definition of name
>
> errhndlr:='readlib('errhndlr',`/user10/xt/xtb003/mplfiles/errhndlr.m`)';
Warning: Recursive definition of name
>
> errtable:='readlib('errtable',`/user10/xt/xtb003/mplfiles/errtable.m`)';
Warning: Recursive definition of name
>
> if assigned(vset) then
>
>    if type(vset,set) then
>
>       print(`WARNING: Vector or array quantities implied in input file.`);
>
>    else
>
>       ok:=false;
>
>       callerr(`Variable vset must be a set`,5)
>
>    fi;
>
> else
>
>    vset:={}:
>
> fi;
>
> if not ok then stop fi;
>
> if assigned(hiharm) then
>
>    if type(hiharm,boolean) then
>
>       if hiharm=true then
>
>          print(`WARNING: Higher harmonics used in this calculation.`)
>
>       elif hiharm=false then
>
>          print(`Expansion using lower harmonics only.`)
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable hiharm must be either true or false.`,8);
>
>    fi;
```

```
>
> else
>
>    hiharm:=false:
>
> fi;
>
> if not ok then stop fi;
>
> if assigned(groundstate) then
>
>    if type(groundstate,boolean) then
>
>       if groundstate=true then
>
>          print(`The groundstate of vector variables will be taken.`);
>
>          print(`to be a perturbation from unity.`);
>
>       elif groundstate=false then
>
>          print(`WARNING: The groundstate of any vector variables will`);
>
>          print(`         not be taken to be a special case.`);
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable groundstate must be either true or false.`,9);
>
>    fi;
>
> else
>
>    groundstate:=true:
>
> fi;
>
> if not ok then stop fi:
>
> kset:={anames()} minus
>
>     {'prettyprint','L','varset','JMAX','errorswitch','prinlvl','hiharm',
>
>      'ok','vset',`errhndlr`,`callerr`,`errtable`,'groundstate'}
>
>      minus vset minus
>
>     {`_initvalsNestlist`,`table/initvals`,`print/matrix`};
>
> if kset<>{} then
>
>    ok:=false:
>
>    callerr(`You have assigned superfluous variables`,kset,1)
>
> fi;
>
```

```
> if not ok then stop fi:
>
> kset:='kset':
>
> if assigned(prinlvl) then
>
>    if type(prinlvl,integer) then
>
>       if prinlvl>=0 and prinlvl<=4 then
>
>          print(`The level of printed output has been set to`,prinlvl);
>
>       else
>
>          ok:=false:
>
>          callerr(`Valid print levels lie between 0 and 4`,2);
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`Prinlvl must be set to an integer`,3);
>
>    fi;
>
> else
>
>    prinlvl:=0;          # Default printlevel set to 0                #
>
> fi;
                The level of printed output has been set to, 3

>
> if not ok then stop fi;
>
# Check to see that L contains the variables given in varset         #
#
> for var in varset while ok=true do
>
>    if not has(L,var) then ok:=false fi
>
> od;
>
> if not ok then
>
>    callerr(`Lagrangian, L, does not contain the variable`,var,4);
>
> fi;
>
> if not ok then stop fi:
>
> if vset<>{} then
>
>    kset:=varset intersect vset:
>
>    if kset={} then
>
>       ok:=false:
>
>       callerr(`Improper use of vector quantities`,6)
>
>    else
```

```
>
>       print(`The following variables have been defined as vectors: `);
>
>       print(kset)
>
>    fi;
>
> else
>
>    kset:={};
>
> fi;
>
> if not ok then stop fi:
>
> for var in (vset minus kset) do;
>
>    if not type(var,matrix) then
>
>      ok:=false;
>
>      callerr(`Incorrect matrix/array type definitions in input file`,7);
>
>    fi;
>
> od;
>
> if not ok then stop fi:
>
# nset is the set of non-vector expansion variables                    #
#
> nset:=varset minus kset:
>
# read `datestam.mpl`;
#
# datestamp();
#
> print(`The input file has now been validated.`);
                    The input file has now been validated.

>
> if prinlvl>2 then
>
>    print(`The program will now go on to perform the expansion`);
>
>    print(`required as a preliminary to the averaging procedure.`);
>
> fi;
              The program will now go on to perform the expansion

             required as a preliminary to the averaging procedure.

>
> if prinlvl>1 then
>
>    print(`The lagrangian used will be:- `);
>
>    print(L);
>
>    if hiharm=true then
>
>       print(`Higher harmonics will be used in the substitution.`)
>
>    else
>
```

```
>        print(`Only the fundamental will be used in the calculation.`)
>
>     fi;
>
>     if nops(vset)<>0 then
>
>        print(`The calculation assumes that the following quantities are`);
>
>        print(`vector variables which will require subsequent expansion`);
>
>        print(kset);
>
>        print(` and that these variables are the scalar variables`);
>
>        print(` be expanded`);
>
>        print(nset);
>
>        print(`The following quantities are arrays which are`);
>
>        print(`assumed to have values given by the user.`);
>
>        print(vset minus kset);
>
>        if groundstate=true then
>
>           print(`The groundstate of vector quantities will be`);
>
>           print(`taken to be near unity.`);
>
>        else
>
>           print(`Note that the groundstate of vector quantities will`);
>
>           print(`NOT be taken to be a special case and will be expanded`);
>
>           print(`in a similar manner to all of the other variables.`):
>
>        fi;
>
>        print();
>
>     else
>
>        print(`These variables will be taken to be those with which the`);
>
>        print(`expansion is to be performed  and with respect to which`);
>
>        print(`the subsequent variation is to be carried out.`);
>
>        print(nset); print();  # nset=varset for all scalars. #
>
>     fi;
>
> fi;
                          The lagrangian used will be:-

                                                                  2         2            4
                     2        2  2                          dt(P)   - mu02 P   + 1/2 alpha P
1/2 esplnO (dt(A)   - dx(A)   c  ) + dt(A) P + 1/2 ------------------------------
                                                                    epslnO omegap2

            Only the fundamental will be used in the calculation.

            These variables will be taken to be those with which the
```

```
            expansion is to be performed  and with respect to which

                 the subsequent variation is to be carried out.

                                        {A, P}


>
> if prinlvl>2 then
>
>    print(`Calculation begins ...`);
>
> fi;
                              Calculation begins ...

>
# Convert sets to lists to maintain ordering.                          #
#
> nlist:=convert(nset,list); klist:=convert(kset,list);
>
# nlist:=[op(nset)]: klist:=[op(kset)]:
#
> tb:=table(): tc:=table():
>
> for i from 1 to nops(nlist) do;
>
>    assign(tb[i]=cat(nlist[i],_));
>
> od;
>
> for i from 1 to nops(klist) do;
>
>    assign(tc[i]=cat(klist[i],_));
>
> od;
>
> newnlist:=convert(tb,list):
>
> newklist:=convert(tc,list):
>
# op(newklist); op(newnlist);
#
> tb:='tb': tc:='tc':   # Unassign tb, tc                              #
>
> read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expdstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddson.mpl`;
>
#read `expdnc.mpl`;
> read `/user10/xt/xtb003/mplfiles/diffdtdx.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
>
# read `truncser.mpl`;
#
```

```
> read `/user10/xt/xtb003/mplfiles/orderser.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/sums.mpl`: # get summation procedures.
>   #
> read `/user10/xt/xtb003/mplfiles/average.mpl`;
>
# # read `smalexpd.mpl`; # Avoid `object too big.`                #
#
> read `/user10/xt/xtb003/mplfiles/smrtexpd.mpl`;
>
> readlib(evalm):
>
# Simplify the lagrangian to get rid of star(star(. etc.
#
> L:=simplify(L,dagger);
>
> L:=simplify(L,star);
>
> if prinlvl>=3 then
>    print(`The lagrangian has now been simplifed with respect to dagger`);
>
>    print(`and star.`);
> fi;
          The lagrangian has now been simplifed with respect to dagger

                                    and star.

> if prinlvl=4 then print(`The lagrangian is now`,L) fi;
> if klist<>[] then
>
>    for i from 1 to nops(klist) do;
>
>       assign(op(i,newklist)=array( op(2,op(op(i,klist)))) );
>
>    od;
>
>    if hiharm=false then
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),quantsum(op(i,newklist)));
>
>       od;
>
>    else
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),bigquantsum(op(i,newklist)));
>
>       od;
>
>    fi;
>
> fi;
>
> if hiharm=false then
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),weeharmsum(op(i,newnlist)));
>
>    od;
>
> else
```

```
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),bigharmsum(op(i,newnlist)));
>
>    od;
>
> fi;
>
# read in the Diff procedure now that the substitutions have been made.
#
> read `/user10/xt/xtb003/mplfiles/diftheta.mpl`;
>
> L:=simplify(L,DT,DX);
bytes used=1000372, alloc=352256, time=16.116
bytes used=2000844, alloc=442368, time=24.766
bytes used=3001348, alloc=516096, time=33.450
bytes used=4001728, alloc=614400, time=42.550
bytes used=5004864, alloc=614400, time=51.500
bytes used=6006660, alloc=614400, time=60.516
bytes used=7006816, alloc=614400, time=69.400
                                    test1

bytes used=8007232, alloc=679936, time=78.366
bytes used=9007644, alloc=835584, time=87.633
bytes used=10008764, alloc=942080, time=96.916
bytes used=11010176, alloc=942080, time=106.233
bytes used=12010432, alloc=942080, time=115.633
                                    test1

bytes used=13014900, alloc=958464, time=125.516
bytes used=14017676, alloc=958464, time=134.050
bytes used=15019624, alloc=958464, time=142.500
bytes used=16020760, alloc=958464, time=151.216
bytes used=17021188, alloc=958464, time=160.000
bytes used=18021512, alloc=958464, time=168.733
bytes used=19023408, alloc=958464, time=177.233
bytes used=20030060, alloc=958464, time=185.833
                                    test1

bytes used=21034312, alloc=958464, time=195.133
bytes used=22034988, alloc=958464, time=204.416
bytes used=23036152, alloc=966656, time=213.950
bytes used=24036532, alloc=966656, time=223.516
bytes used=25036908, alloc=983040, time=233.150
>
# Should now have made the necessary assignments. Matrices should not be
#
# empty so can carry out necessary calculations.
#
#expand(expandoff); expand(expandon); expandoff(exp);
#
#printlevel:=2000;
#
> if prinlvl>=3 then
>
>    print(`The arrays have now been filled and the summations`);
>
>    print(`required for the expansion carried out.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian has become: `,L);
>
>    fi;
```

```
>
> fi;
                The arrays have now been filled and the summations

                     required for the expansion carried out.

>
# quit;
#
> L:=expddagstonly(L);
>
> if prinlvl>=3 then
>    print(`The lagrangian has been expanded with repsect to its`);
>
>    print(`dagger and star operations only.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian becomes: `);
>
>       lprint(L);
>
>    fi;
>
> fi;
              The lagrangian has been expanded with repsect to its

                        dagger and star operations only.

>
# If simply do expand then will fail when dagger is mapped onto
#
# matrices. Transpose will alter shape remember.
#
> L:=simplify(L,DT,DX,star); # simplify wrt DT,DX,star only.
bytes used=26039112, alloc=983040, time=245.683
bytes used=27041272, alloc=983040, time=253.616
bytes used=28043300, alloc=983040, time=261.933
bytes used=29045680, alloc=983040, time=270.200
bytes used=30046020, alloc=983040, time=278.633
bytes used=31046172, alloc=983040, time=287.333
bytes used=32047072, alloc=983040, time=295.983
bytes used=33048304, alloc=983040, time=304.833
bytes used=34048680, alloc=983040, time=313.633
bytes used=35049028, alloc=983040, time=322.216
bytes used=36053944, alloc=983040, time=330.166
bytes used=37055916, alloc=983040, time=339.283
bytes used=38056268, alloc=983040, time=347.550
                                       test1

bytes used=39056384, alloc=983040, time=356.766
bytes used=40059156, alloc=983040, time=365.916
bytes used=41060212, alloc=983040, time=375.266
bytes used=42061300, alloc=983040, time=384.583
bytes used=43061696, alloc=983040, time=394.066
                                       test1

bytes used=44062100, alloc=983040, time=403.866
bytes used=45063332, alloc=983040, time=412.383
bytes used=46064100, alloc=983040, time=420.750
bytes used=47066256, alloc=983040, time=429.283
bytes used=48069040, alloc=983040, time=437.950
bytes used=49072000, alloc=983040, time=446.600
bytes used=50073640, alloc=983040, time=455.116
bytes used=51075952, alloc=983040, time=463.633
```

```
                                    test1

bytes used=52076220, alloc=983040, time=472.783
bytes used=53077796, alloc=983040, time=481.950
bytes used=54082416, alloc=983040, time=491.466
bytes used=55082636, alloc=983040, time=500.983
bytes used=56083116, alloc=983040, time=510.550
bytes used=57083540, alloc=999424, time=520.566
>
> save `/scratch/xt/xtb003/cl/lagrngq4.m`;
#should now be able to truncate
#
#L:=smartexpand(L):
> L:=collect(L,eps,recursive,expand);
bytes used=58102000, alloc=999424, time=545.533
bytes used=59155548, alloc=1482752, time=562.733
bytes used=60424368, alloc=2179072, time=585.533
bytes used=61602752, alloc=2752512, time=608.833
bytes used=62861428, alloc=3268608, time=636.150
# Truncate the expanded lagrangian.
#
> L:=seriestrunc(L,[eps=1],JMAX);
> if prinlvl=4 then
>
>    print(`The lagrangian has been expanded and simplified.`);
>
>    print(`Its final form is: `);
>
>    lprint(L);
>
> fi;
>
# Collect terms in `eps`                #
#
> L:=collect(L,eps):
>
> if prinlvl>=3 then
>
>    print(`The lagrangian has been rearranged in terms of the small`);
>
>    print(`variable eps.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian is now: `);
>
>       lprint(L);
>
>    fi;
>
> fi;
           The lagrangian has been rearranged in terms of the small

                                variable eps.

>
> save `/scratch/xt/xtb003/cl/avlagq4t.m`;
bytes used=63970808, alloc=3776512, time=688.866
>
# lprint(L);
> for i from 0 to JMAX do; #4 do;
>    L.i:=coeff(L,`eps`,i);
>
>    L.i.`bar`:=average(L.i);
>
```

```
>    LC.i.`bar`:=evalc(L.i.`bar`);
>    print(`The averaged lagrangian at order`,i,` is:`);
>
>    print(LC.i.`bar`);
> od;
```

The averaged lagrangian at order, 0, is:

$$0$$

The averaged lagrangian at order, 1, is:

$$0$$

The averaged lagrangian at order, 2, is:

$$\begin{aligned}
&\mathrm{espln0}\, w^2\, A_[0,1]\, \mathrm{star}(A_[0,1]) - \mathrm{espln0}\, k^2\, c^2\, A_[0,1]\, \mathrm{star}(A_[0,1]) \\
&+ \frac{w^2\, P_[0,1]\, \mathrm{star}(P_[0,1])}{\mathrm{epsln0}\ \mathrm{omegap2}} - \frac{\mathrm{mu02}\, P_[0,1]\, \mathrm{star}(P_[0,1])}{\mathrm{epsln0}\ \mathrm{omegap2}} \\
&+ (-w\, A_[0,1]\, \mathrm{star}(P_[0,1]) + w\, \mathrm{star}(A_[0,1])\, P_[0,1])\, I
\end{aligned}$$

The averaged lagrangian at order, 3, is:

$$\begin{aligned}
&\mathrm{DT}(\%1)\, \%3 + \mathrm{star}(\mathrm{DT}(\%1))\, P_[0,1] - \mathrm{espln0}\, k^2\, c^2\, A_[0,1]\, \mathrm{star}(A_[1,1]) \\
&+ \frac{P_[0,1]\, w^2\, \mathrm{star}(P_[1,1])}{\mathrm{epsln0}\ \mathrm{omegap2}} + \frac{\%3\, w^2\, P_[1,1]}{\mathrm{epsln0}\ \mathrm{omegap2}} \\
&- \frac{\mathrm{mu02}\, P_[0,1]\, \mathrm{star}(P_[1,1])}{\mathrm{epsln0}\ \mathrm{omegap2}} - \frac{\mathrm{mu02}\, \%3\, P_[1,1]}{\mathrm{epsln0}\ \mathrm{omegap2}} \\
&+ \mathrm{espln0}\, A_[0,1]\, w^2\, \mathrm{star}(A_[1,1]) + \mathrm{espln0}\, \%2\, w^2\, A_[1,1] \\
&- \mathrm{espln0}\, k^2\, c^2\, \%2\, A_[1,1] + (\mathrm{espln0}\, k\, c^2\, \%2\, \mathrm{DX}(\%1) - w\, A_[1,1]\, \%3 \\
&+ w\, \mathrm{star}(A_[1,1])\, P_[0,1] - w\, A_[0,1]\, \mathrm{star}(P_[1,1]) + w\, \%2\, P_[1,1] \\
&- \frac{w\, P_[0,1]\, \mathrm{star}(\mathrm{DT}(P_[0,1],1))}{\mathrm{epsln0}\ \mathrm{omegap2}} + \frac{w\, \%3\, \mathrm{DT}(P_[0,1],1)}{\mathrm{epsln0}\ \mathrm{omegap2}} \\
&- \mathrm{espln0}\, w\, A_[0,1]\, \mathrm{star}(\mathrm{DT}(\%1)) + \mathrm{espln0}\, w\, \%2\, \mathrm{DT}(\%1) \\
&- \mathrm{espln0}\, k\, c^2\, A_[0,1]\, \mathrm{star}(\mathrm{DX}(\%1)))\, I
\end{aligned}$$

$$\%1 := A\_[0,1],\ 1$$

$$\%2 := \mathrm{star}(A\_[0,1])$$

$$\%3 := \mathrm{star}(P\_[0,1])$$

```
                    The averaged lagrangian at order, 4,  is:

               2                       2
espln0 %6 w   A_[2, 1] + espln0 w   A_[1, 1] %7

                         2                                  2  2
     + espln0 A_[0, 1] w   star(A_[2, 1]) - espln0 c   k   A_[1, 1] %7

                   2      2                    2          2
     - espln0 c   %6 k   A_[2, 1] - espln0 c   A_[0, 1] k   star(A_[2, 1])

                                                  2
     + A_[1, 1] %1 + %7 P_[0, 1] - espln0 c   DX(%5) star(DX(%5))

        %8 star(%8)                                  w %1 DT(P_[1, 1], 1)
     + -------------- + (- espln0 w A_[0, 1] %7 + --------------------
       epsln0 omegap2                                  epsln0 omegap2

       w P_[0, 1] %2                                      w %1 P_[1, 1]
     - -------------- - espln0 w A_[0, 1] star(DT(%3)) + --------------
       epsln0 omegap2                                     epsln0 omegap2

                  2                     2
     + espln0 c   k %7 DX(%5) + espln0 c   k %6 DX(%4)

                  2
     - espln0 c   k A_[0, 1] star(DX(%3)) + espln0 w %6 DT(%4)

     + espln0 w %6 A_[1, 1] + espln0 w %7 DT(%5)

     - espln0 w A_[1, 1] star(DT(%5)) + espln0 w %6 DT(%3)

                  2                                    2
     - espln0 c   k A_[1, 1] star(DX(%5)) - espln0 c   k A_[0, 1] %7

                  2
     - espln0 c   k A_[0, 1] star(DX(%4)) - espln0 w A_[0, 1] star(DT(%4))

                  2            w %1 DT(P_[0, 1], 2)
     + espln0 c   k %6 DX(%3) + --------------------
                                  epsln0 omegap2

       w P_[0, 1] star(DT(P_[1, 1], 1))      w %2 %8        w P_[1, 1] star(%8)
     - -------------------------------- + -------------- - -------------------
                epsln0 omegap2            epsln0 omegap2    epsln0 omegap2

                                                                  2
     + w star(A_[2, 1]) P_[0, 1] - w A_[2, 1] %1 + espln0 c   k %6 A_[1, 1]

     + w %7 P_[1, 1] - w A_[1, 1] %2 + w %6 P_[2, 1]

                                       w P_[0, 1] star(DT(P_[0, 1], 2))
     - w A_[0, 1] star(P_[2, 1]) - ---------------------------------) I
                                           epsln0 omegap2

           2                           2                         2
       %1 w   P_[2, 1]    P_[0, 1] w   star(P_[2, 1])    w   P_[1, 1] %2
     + -------------- + -------------------------- + --------------
       epsln0 omegap2        epsln0 omegap2          epsln0 omegap2

     + espln0 DT(%5) star(DT(%5)) + DT(%5) %2 + star(DT(%5)) P_[1, 1]

     + DT(%4) %1 + star(DT(%4)) P_[0, 1] + DT(%3) %1 + star(DT(%3)) P_[0, 1]

       mu02 P_[0, 1] star(P_[2, 1])    mu02 %1 P_[2, 1]    mu02 P_[1, 1] %2
```

```
      - --------------------------- - --------------- - ---------------
               epsln0 omegap2            epsln0 omegap2     epsln0 omegap2

                           2   2
                 alpha P_[0, 1]  %1
           + 3/2 --------------------
                  epsln0 omegap2

%1 :=
                                  star(P_[0, 1])

%2 :=
                                  star(P_[1, 1])

%3 :=
                                    A_[0, 1], 2

%4 :=
                                    A_[1, 1], 1

%5 :=
                                    A_[0, 1], 1

%6 :=
                                  star(A_[0, 1])

%7 :=
                                  star(A_[1, 1])

%8 :=
                                  DT(P_[0, 1], 1)

>
> save `/scratch/xt/xtb003/cl/tavlag4q.m`;
# quit;
#
# This was originally file eqnmotq2.mpl
# trying to make the equations of motion. #
# JMAX:=6; prinlvl:=4;
# v_:=array(1..2,[]); #array bound required for looping.
# newnlist:=[eta_]; newklist:=[v_]; hiharm:=false;
# LC0bar:=0; LC1bar:=0;
# read `/user10/xt/xtb003/results/averq2.m`;
# printlevel:=-1;
# Define LC(2-4)bar;
> read `/user10/xt/xtb003/mplfiles/ordindex.mpl`;
# read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
> read `/user10/xt/xtb003/mplfiles/dordbool.mpl`;
> read `/user10/xt/xtb003/mplfiles/mapmod.mpl`;
> read `/user10/xt/xtb003/mplfiles/eqnmot.mpl`;
> read `/user10/xt/xtb003/mplfiles/derivord.mpl`;
> read `/user10/xt/xtb003/mplfiles/makeELterm.mpl`;
> read `/user10/xt/xtb003/mplfiles/doprint2.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
> read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
# Have now read in necessary files.
>
> for levcount from 0 to JMAX do;
>    equationsofmotion(`LC`.levcount.`bar`);
# Create DTvar, DXvar, DXDTvar as sets.
>    DTvar:=convert(DTvar,`list`);
>    DXvar:=convert(DXvar,`list`);
>    DXDTvar:=convert(DXDTvar,`list`);
>
```

```
>    scalvar:=convert(scalvar,`list`);
>
>    vecvar:=convert(vecvar,`list`);
>
>    newDTvar:=sort(DTvar,dordbool);
>
>    newDXvar:=sort(DXvar,dordbool);
>
>    newDXDTvar:=sort(DXDTvar,dordbool);
>
#   printlevel:=2000;
>    newscalvar:=sort(scalvar,fordbool);
>    newvecvar:=sort(vecvar,fordbool);
#   printlevel:=-1;
>    if prinlvl>3 then
>       print(`op(newscalvar)`,op(newscalvar));
>       print(`op(newvecvar)`,op(newvecvar) );
>       print(`op(newDTvar)`,op(newDTvar));
>       print(`op(newDXvar)`,op(newDXvar));
>       print(`op(newDXDTvar)`,op(newDXDTvar));
>    fi;
# printlevel:=2000;
>    lprint(`The lagrangian being used is :-`);
>    lprint(`LC`.levcount.`bar`);
# Generating the equations starts here.
>    scalareqnset:={}; vectoreqnset:={}; # Put the names of the arrays
# which will hold the equations in these two sets.
>    if nops(newscalvar)<>0 then
#      print(newscalvar,`TATEST3`);
>       for var in newscalvar do;
# Generate Euler-Lagrange for scalar variables.
>       print(`Investigating variable `,var,`at level`,levcount);
>       scalname:=op(0,var);
>       if not assigned(`eqtn`.levcount.scalname) then
>          scalareqnset:=scalareqnset union {`eqtn`.levcount.scalname};
>          `eqtn`.levcount.scalname:=table(sparse,[]) fi;
#      printlevel:=101; print(`101 #5`);
>       `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>            makeELterm(var,`LC`.levcount.`bar`);
#      printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #6`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #7`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
```

```
#      printlevel:=101; print(`101 #8`);
>               `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                  `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                  makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>            fi;
>         od;
>       fi;
>       od;
#      print(`eqtn`.levcount.op(0,var)[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
>    if nops(newvecvar)<>0 then
#      print(newvecvar,`TATEST4`);
>       for var in newvecvar do;
#         printlevel:=11;
# Generate Euler-Lagrange for vector variables.
>       print(`Investigating vector variable`,var,`at level`,levcount);
>       vecname:=op(0,var);
>       if not assigned(`eqtn`.levcount.vecname) then
>          vectoreqnset:=vectoreqnset union {`eqtn`.levcount.vecname};
>          `eqtn`.levcount.vecname:=table(sparse,[]) fi;
# printlevel:=101; print(`101 #1`);
>       `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>           makeELterm(var,`LC`.levcount.`bar`);
# printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#               printlevel:=101;print(`101 #2`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#               printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#         printlevl:=101; print(`101 #3`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>            fi;
>         od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
#               printlvel:=101; print(`101 #4`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#     printlevel:=-1;
>            fi;
>         od;
>       fi;
>       od;
#      printlevel:=-1;
#      print(`eqtn`.levcount.vecname[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
#   printlevel:=11;
```

```
>    doprinteqns();
> od; # end of outer loop.
The lagrangian being used is :-
0
The lagrangian being used is :-
0
The lagrangian being used is :-
espln0*w**2*A_[0,1]*star(A_[0,1])-espln0*k**2*c**2*A_[0,1]*star(A_[0,1])+1/
epsln0/omegap2*w**2*P_[0,1]*star(P_[0,1])-1/epsln0/omegap2*mu02*P_[0,1]*star(P_
[0,1])+(-w*A_[0,1]*star(P_[0,1])+w*star(A_[0,1])*P_[0,1])*I
                Investigating variable , A_[0, 1], at level, 2

                Investigating variable , P_[0, 1], at level, 2

            The equations of motion for the scalar variables are:-


                 The equation of motion for , P_[0,1],  is:-

                   2
                  w  P_[0, 1]      mu02 P_[0, 1]
                 -------------- - -------------- - w A_[0, 1] I, =0
                 epsln0 omegap2    epsln0 omegap2


                 The equation of motion for , A_[0,1],  is:-

                   2                        2  2
          espln0 w  A_[0, 1] - espln0 k  c  A_[0, 1] + w P_[0, 1] I, =0


bytes used=64972068, alloc=3776512, time=707.816
The lagrangian being used is :-
DT(A_[0,1],1)*star(P_[0,1])+star(DT(A_[0,1],1))*P_[0,1]-espln0*k**2*c**2*A_[0,1
]*star(A_[1,1])+1/epsln0/omegap2*P_[0,1]*w**2*star(P_[1,1])+1/epsln0/omegap2*
star(P_[0,1])*w**2*P_[1,1]-1/epsln0/omegap2*mu02*P_[0,1]*star(P_[1,1])-1/epsln0
/omegap2*mu02*star(P_[0,1])*P_[1,1]+espln0*A_[0,1]*w**2*star(A_[1,1])+espln0*
star(A_[0,1])*w**2*A_[1,1]-espln0*k**2*c**2*star(A_[0,1])*A_[1,1]+(espln0*k*c**
2*star(A_[0,1])*DX(A_[0,1],1)-w*A_[1,1]*star(P_[0,1])+w*star(A_[1,1])*P_[0,1]-w
*A_[0,1]*star(P_[1,1])+w*star(A_[0,1])*P_[1,1]-1/epsln0/omegap2*w*P_[0,1]*star(
DT(P_[0,1],1))+1/epsln0/omegap2*w*star(P_[0,1])*DT(P_[0,1],1)-espln0*w*A_[0,1]*
star(DT(A_[0,1],1))+espln0*w*star(A_[0,1])*DT(A_[0,1],1)-espln0*k*c**2*A_[0,1]*
star(DX(A_[0,1],1)))*I
                Investigating variable , A_[0, 1], at level, 3

                Investigating variable , A_[1, 1], at level, 3

                Investigating variable , P_[0, 1], at level, 3

                Investigating variable , P_[1, 1], at level, 3

            The equations of motion for the scalar variables are:-


                 The equation of motion for , P_[1,1],  is:-

                   2
                  w  P_[0, 1]      mu02 P_[0, 1]
                 -------------- - -------------- - w A_[0, 1] I, =0
                 epsln0 omegap2    epsln0 omegap2
```

```
                The equation of motion for , P_[0,1],  is:-

                                2
                              w  P_[1, 1]     mu02 P_[1, 1]
          DT(A_[0, 1], 1) + -------------- - --------------
                             epsln0 omegap2   epsln0 omegap2

                 /              w DT(P_[0, 1], 1)\      w P_[0, 1] I
               + |- w A_[1, 1] + -----------------| I + --------------
                 \               epsln0 omegap2  /    epsln0 omegap2

                 w DT(P_[0, 1], 1) I
               + -------------------,                              =0
                  epsln0 omegap2


                The equation of motion for , A_[1,1],  is:-

                     2                       2  2
          espln0 w  A_[0, 1] - espln0 k  c  A_[0, 1] + w P_[0, 1] I, =0


                The equation of motion for , A_[0,1],  is:-

              2                          2  2
  espln0 w  A_[1, 1] - espln0 k  c  A_[1, 1]

                          2
       + (espln0 k c  DX(%1) + w P_[1, 1] + espln0 w DT(%1)) I - DT(P_[0, 1], 1)

                                                                   2
       + espln0 w A_[0, 1] I + espln0 w DT(%1) I + espln0 k c  A_[0, 1] I

                         2
       + espln0 k c  DX(%1) I,

      =0

%1 :=
                                     A_[0, 1], 1


bytes used=65974752, alloc=3776512, time=716.566
The lagrangian being used is :-
espln0*star(A_[0,1])*w**2*A_[2,1]+espln0*w**2*A_[1,1]*star(A_[1,1])+espln0*A_[0
,1]*w**2*star(A_[2,1])-espln0*c**2*k**2*A_[1,1]*star(A_[1,1])-espln0*c**2*star(
A_[0,1])*k**2*A_[2,1]-espln0*c**2*A_[0,1]*k**2*star(A_[2,1])+A_[1,1]*star(P_[0,
1])+star(A_[1,1])*P_[0,1]-espln0*c**2*DX(A_[0,1],1)*star(DX(A_[0,1],1))+1/
epsln0/omegap2*DT(P_[0,1],1)*star(DT(P_[0,1],1))+(-espln0*w*A_[0,1]*star(A_[1,1
])+1/epsln0/omegap2*w*star(P_[0,1])*DT(P_[1,1],1)-1/epsln0/omegap2*w*P_[0,1]*
star(P_[1,1])-espln0*w*A_[0,1]*star(DT(A_[0,1],2))+1/epsln0/omegap2*w*star(P_[0
,1])*P_[1,1]+espln0*c**2*k*star(A_[1,1])*DX(A_[0,1],1)+espln0*c**2*k*star(A_[0,
1])*DX(A_[1,1],1)-espln0*c**2*k*A_[0,1]*star(DX(A_[0,1],2))+espln0*w*star(A_[0,
1])*DT(A_[1,1],1)+espln0*w*star(A_[0,1])*A_[1,1]+espln0*w*star(A_[1,1])*DT(A_[0
,1],1)-espln0*w*A_[1,1]*star(DT(A_[0,1],1))+espln0*w*star(A_[0,1])*DT(A_[0,1],2
)-espln0*c**2*k*A_[1,1]*star(DX(A_[0,1],1))-espln0*c**2*k*A_[0,1]*star(A_[1,1])
-espln0*c**2*k*A_[0,1]*star(DX(A_[1,1],1))-espln0*w*A_[0,1]*star(DT(A_[1,1],1))
+espln0*c**2*k*star(A_[0,1])*DX(A_[0,1],2)+1/epsln0/omegap2*w*star(P_[0,1])*DT(
P_[0,1],2)-1/epsln0/omegap2*w*P_[0,1]*star(DT(P_[1,1],1))+1/epsln0/omegap2*w*
star(P_[1,1])*DT(P_[0,1],1)-1/epsln0/omegap2*w*P_[1,1]*star(DT(P_[0,1],1))+w*
```

```
star(A_[2,1])*P_[0,1]-w*A_[2,1]*star(P_[0,1])+espln0*c**2*k*star(A_[0,1])*A_[1,
1]+w*star(A_[1,1])*P_[1,1]-w*A_[1,1]*star(P_[1,1])+w*star(A_[0,1])*P_[2,1]-w*A_
[0,1]*star(P_[2,1])-1/epsln0/omegap2*w*P_[0,1]*star(DT(P_[0,1],2)))*I+1/epsln0/
omegap2*star(P_[0,1])*w**2*P_[2,1]+1/epsln0/omegap2*P_[0,1]*w**2*star(P_[2,1])+
1/epsln0/omegap2*w**2*P_[1,1]*star(P_[1,1])+espln0*DT(A_[0,1],1)*star(DT(A_[0,1
],1))+DT(A_[0,1],1)*star(P_[1,1])+star(DT(A_[0,1],1))*P_[1,1]+DT(A_[1,1],1)*
star(P_[0,1])+star(DT(A_[1,1],1))*P_[0,1]+DT(A_[0,1],2)*star(P_[0,1])+star(DT(
A_[0,1],2))*P_[0,1]-1/epsln0/omegap2*mu02*P_[0,1]*star(P_[2,1])-1/epsln0/
omegap2*mu02*star(P_[0,1])*P_[2,1]-1/epsln0/omegap2*mu02*P_[1,1]*star(P_[1,1])+
3/2/epsln0/omegap2*alpha*P_[0,1]**2*star(P_[0,1])**2
                Investigating variable , A_[0, 1], at level, 4

                Investigating variable , A_[1, 1], at level, 4

                Investigating variable , A_[2, 1], at level, 4

                Investigating variable , P_[0, 1], at level, 4

                Investigating variable , P_[1, 1], at level, 4

                Investigating variable , P_[2, 1], at level, 4

            The equations of motion for the scalar variables are:-


                 The equation of motion for , A_[2,1],  is:-

                  2                      2  2
        espln0 w  A_[0, 1] - espln0 k  c  A_[0, 1] + w P_[0, 1] I, =0


                 The equation of motion for , A_[1,1],  is:-

                   2                      2  2
P_[0, 1] + espln0 w  A_[1, 1] - espln0 k  c  A_[1, 1] + (- espln0 w A_[0, 1]

                                    2                          2
     + espln0 w DT(%1) - espln0 c  k A_[0, 1] + espln0 k c  DX(%1) + w P_[1, 1]

    ) I + espln0 w A_[0, 1] I + espln0 w DT(%1) I - DT(P_[0, 1], 1)

                  2                        2
     + espln0 k c  A_[0, 1] I + espln0 k c  DX(%1) I,

    =0

%1 :=
                                A_[0, 1], 1


                 The equation of motion for , A_[0,1],  is:-

       2                      2  2
espln0 w  A_[2, 1] - espln0 c  k  A_[2, 1] + (espln0 w A_[1, 1] + w P_[2, 1]

              2                                          2
     + espln0 c  k A_[1, 1] + espln0 w DT(%1) + espln0 c  k DX(%2)

              2
     + espln0 c  k DX(%1) + espln0 w DT(%2)) I + espln0 w A_[1, 1] I

     + espln0 w DT(%2) I - espln0 DT(A_[0, 1], 1, 1) - DT(P_[1, 1], 1)
```

```
     + esplnO w A_[0, 1] I + esplnO w DT(%1) I - DT(P_[0, 1], 2)

                 2                              2
     + esplnO c   DX(A_[0, 1], 1) + esplnO c   DX(A_[0, 1], 1, 1)

                 2                            2                        2
     + esplnO c   k A_[1, 1] I + esplnO c   k DX(%2) I + esplnO k c   A_[0, 1] I

                 2
     + esplnO c   k DX(%1) I,                                                    =0
%1 :=
                                  A_[0, 1], 2
%2 :=
                                  A_[1, 1], 1


               The equation of motion for , P_[2,1],  is:-

                 2
                w   P_[0, 1]      mu02 P_[0, 1]
               --------------- - --------------- - w A_[0, 1] I, =0
               epslnO omegap2    epslnO omegap2


               The equation of motion for , P_[1,1],  is:-

 /    w P_[0, 1]                        w DT(P_[0, 1], 1)\     mu02 P_[1, 1]
 |- --------------- - w A_[1, 1] + ------------------| I - ---------------
 \  epslnO omegap2                       epslnO omegap2  /    epslnO omegap2

        2
       w   P_[1, 1]                              w P_[0, 1] I    w DT(P_[0, 1], 1) I
     + --------------- + DT(A_[0, 1], 1) + --------------- + -------------------
       epslnO omegap2                           epslnO omegap2    epslnO omegap2

     ,

     =0


               The equation of motion for , P_[0,1],  is:-

A_[1, 1]

       /                 w DT(P_[0, 1], 2)    w DT(P_[1, 1], 1)      w P_[1, 1]  \
     + |- w A_[2, 1] + ------------------ + ------------------ + ---------------| I
       \                  epslnO omegap2        epslnO omegap2      epslnO omegap2/

                                                                  2
                           mu02 P_[2, 1]      alpha P_[0, 1]  star(P_[0, 1])
     + DT(A_[0, 1], 2) - --------------- + 3 --------------------------------
                           epslnO omegap2              epslnO omegap2

        2
       w   P_[2, 1]                            DT(P_[0, 1], 1, 1)    w P_[1, 1] I
     + --------------- + DT(A_[1, 1], 1) - ------------------ + ---------------
       epslnO omegap2                          epslnO omegap2       epslnO omegap2

       w DT(P_[1, 1], 1) I    w P_[0, 1] I    w DT(P_[0, 1], 2) I
```

```
    + ------------------- + ------------- + -------------------,
        epsln0 omegap2       epsln0 omegap2      epsln0 omegap2

    =0


> save '/scratch/xt/xtb003/cl/eqnmotq.m';
bytes used=67066528, alloc=3776512, time=729.550
> quit;
bytes used=67066668, alloc=3776512, time=729.550
```

Appendix 7

There follows a listing of the output produced by the program for the quantum lagrangian. The output has been edited to remove some of the "words used" messages.

Note that although the vector variables are included in the lagrangian, and the relevant summations are generated at intermediate stages, they are removed by the averaging procedure with the result that no equations of motion corresponding to vector variables appear in the printout. This suggests that some change should be made to the relevant summation procedure or, of course, that there may be some bug in the program.

```
    |\^/|
._|\|   |/|_. University of Lancaster - Computer Ctr
 \  MAPLE  /  Version 4.3 --- Mar 1989
 <____ ____>  For on-line help, type  help();
      |
# ================================================================ #
#                                                                  #
#                                                                  #
# (C) Tom Arbuckle 1989, 1990                                      #
#                                                                  #
#                                                                  #
# ================================================================ #
#                                                                  #
#                                                                  #
# This program is the first in a suite of programs designed to deal #
# with the process of obtaining the equations of motion from an    #
# averaged lagrangian. This first program accepts as input the file #
# AVERIN DAT A (AVERIN.DAT in MAPLE) and then performs validation  #
# on the contents of this file. It then goes on to perform suitable #
# expansions to a level specified to the user, finally passing the #
# results on to the next progam as a data file.                    #
# The file AVERIN.DAT should contain the following:                #
#    1) the variable errorswitch (set to true or false);           #
#    2) the variable prinlvl (set to an integer 0<=prinlvl<=4);    #
#    3) the lagrangian,L;                                          #
#    4) the required order of expansion, jmax;                     #
#    5) a set, varset, containing the variables used in L;         #
#    6) an optional set, vset, containing all array type quantities #
#           in use within the Lagrangian;                          #
#    7) a variable, hiharm, determining whether or not the expansion #
#           used will include harmonics higher than the fundamental. #
#           This variable should either be set to true or false.   #
#    8) a variable, groundstate, determining whether or not the    #
#           groundstate of a vector quantity will be treated as a  #
#           perturbation from unity. Normally set to true.         #
# If either of the first two variables are not assigned, they are  #
# given the default values of false and 0.                         #
# The variable hiharm will be given the value false if not assigned. #
#
```

```
# The variable groundstate has a default of true if not assigned.    #
#                                                                     #
#                                                                     #
#
# N.B.  The variable ok must be assigned false whenever an error     #
#
# occurs. A top level exit is required in order that the output      #
#
# from the help file be displayed. Hence checking to see whether     #
#
# things are ok will correctly terminate the program on error.       #
#
#                                                                     #
#
# ================================================================== #
#
##
#
> printlevel:=-1;                # adjust level of output              #
>
# printlevel:=1;
#
> prettyprint:=2:                  # left justify printed output       #
>
> words(0);                        # switch off words used messages    #
>
> read `/user10/xt/xtb003/mplfiles/averqn.dat`;    # READ AVERIN DAT A  #
> ok:=true;
>
> if assigned(errorswitch) then
>
>    if type(errorswitch, boolean) then
>
>        if errorswitch=false then
>
>            print(`Error reporting not required by user`);
>
>        elif errorswitch=true then
>
>            print(`Extended error reporting switched on.`);
>
>            print(`Detailed error messages will be given by the program`);
>
>        else
>
>            print(`The variable errorswitch must be assigned true or false`);
>
>            ok:=false;
>
>        fi;
>
>     else
>
>         print(`The variable errorswitch must be assigned true or false`);
>
>         ok:=false;
>
>     fi;
>
> fi;
                    Extended error reporting switched on.

              Detailed error messages will be given by the program
```

```
>
> if ok=false then stop fi:
>
# Previous line keeps ok as a variable to use later.                    #
#
# Now load in callerr as the error handling procedure                   #
#
# callerr defined as an unevaluated function call                       #
#
> callerr:='readlib('callerr',`/user10/xt/xtb003/mplfiles/callerr.m`)';
Warning: Recursive definition of name
>
> errhndlr:='readlib('errhndlr',`/user10/xt/xtb003/mplfiles/errhndlr.m`)';
Warning: Recursive definition of name
>
> errtable:='readlib('errtable',`/user10/xt/xtb003/mplfiles/errtable.m`)';
Warning: Recursive definition of name
>
> if assigned(vset) then
>
>    if type(vset,set) then
>
>       print(`WARNING: Vector or array quantities implied in input file.`);
>
>    else
>
>       ok:=false;
>
>       callerr(`Variable vset must be a set`,5)
>
>    fi;
>
> else
>
>    vset:={}:
>
> fi;
          WARNING: Vector or array quantities implied in input file.

>
> if not ok then stop fi;
>
> if assigned(hiharm) then
>
>    if type(hiharm,boolean) then
>
>       if hiharm=true then
>
>          print(`WARNING: Higher harmonics used in this calculation.`)
>
>       elif hiharm=false then
>
>          print(`Expansion using lower harmonics only.`)
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable hiharm must be either true or false.`,8);
```

```
>
>    fi;
>
> else
>
>    hiharm:=false:
>
> fi;
                    Expansion using lower harmonics only.

>
> if not ok then stop fi;
>
> if assigned(groundstate) then
>
>    if type(groundstate,boolean) then
>
>       if groundstate=true then
>
>          print(`The groundstate of vector variables will be taken.`);
>
>          print(`to be a perturbation from unity.`);
>
>       elif groundstate=false then
>
>          print(`WARNING: The groundstate of any vector variables will`);
>
>          print(`         not be taken to be a special case.`);
>
>       else
>
>          ok:=false
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`The variable groundstate must be either true or false.`,9);
>
>    fi;
>
> else
>
>    groundstate:=true:
>
> fi;
              The groundstate of vector variables will be taken.

                       to be a perturbation from unity.

>
> if not ok then stop fi:
>
> kset:={anames()} minus
>
>     {'prettyprint','L','varset','JMAX','errorswitch','prinlvl','hiharm',
>
>      'ok','vset',`errhndlr`,`callerr`,`errtable`,'groundstate'}
>
>      minus vset minus
>
>     {`_initvalsNestlist`,`table/initvals`,`print/matrix`};
>
```

```
> if kset<>{} then
>
>    ok:=false:
>
>    callerr(`You have assigned superfluous variables`,kset,1)
>
> fi;
>
> if not ok then stop fi:
>
> kset:='kset':
>
> if assigned(prinlvl) then
>
>    if type(prinlvl,integer) then
>
>       if prinlvl>=0 and prinlvl<=4 then
>
>          print(`The level of printed output has been set to`,prinlvl);
>
>       else
>
>          ok:=false:
>
>          callerr(`Valid print levels lie between 0 and 4`,2);
>
>       fi;
>
>    else
>
>       ok:=false:
>
>       callerr(`Prinlvl must be set to an integer`,3);
>
>    fi;
>
> else
>
>    prinlvl:=0;          # Default printlevel set to 0                #
>
> fi;
                The level of printed output has been set to, 4

>
> if not ok then stop fi;
>
# Check to see that L contains the variables given in varset         #
#
> for var in varset while ok=true do
>
>    if not has(L,var) then ok:=false fi
>
> od;
>
> if not ok then
>
>    callerr(`Lagrangian, L, does not contain the variable`,var,4);
>
> fi;
>
> if not ok then stop fi:
>
> if vset<>{} then
>
>    kset:=varset intersect vset:
```

```
>
>    if kset={} then
>
>       ok:=false:
>
>       callerr(`Improper use of vector quantities`,6)
>
>    else
>
>       print(`The following variables have been defined as vectors: `);
>
>       print(kset)
>
>    fi;
>
> else
>
>    kset:={};
>
> fi;
            The following variables have been defined as vectors:

                                        {v}

>
> if not ok then stop fi:
>
> for var in (vset minus kset) do;
>
>    if not type(var,matrix) then
>
>      ok:=false;
>
>      callerr(`Incorrect matrix/array type definitions in input file`,7);
>
>    fi;
>
> od;
>
> if not ok then stop fi:
>
# nset is the set of non-vector expansion variables                    #
#
> nset:=varset minus kset:
>
# read `datestam.mpl`;
#
# datestamp();
#
> print(`The input file has now been validated.`);
                    The input file has now been validated.

>
> if prinlvl>2 then
>
>    print(`The program will now go on to perform the expansion`);
>
>    print(`required as a preliminary to the averaging procedure.`);
>
> fi;
             The program will now go on to perform the expansion

            required as a preliminary to the averaging procedure.
```

```
>
> if prinlvl>1 then
>
>    print(`The lagrangian used will be:- `);
>
>    print(L);
>
>    if hiharm=true then
>
>       print(`Higher harmonics will be used in the substitution.`)
>
>    else
>
>       print(`Only the fundamental will be used in the calculation.`)
>
>    fi;
>
>    if nops(vset)<>0 then
>
>       print(`The calculation assumes that the following quantities are`);
>
>       print(`vector variables which will require subsequent expansion`);
>
>       print(kset);
>
>       print(` and that these variables are the scalar variables`);
>
>       print(` be expanded`);
>
>       print(nset);
>
>       print(`The following quantities are arrays which are`);
>
>       print(`assumed to have values given by the user.`);
>
>       print(vset minus kset);
>
>       if groundstate=true then
>
>          print(`The groundstate of vector quantities will be`);
>
>          print(`taken to be near unity.`);
>
>       else
>
>          print(`Note that the groundstate of vector quantities will`);
>
>          print(`NOT be taken to be a special case and will be expanded`);
>
>          print(`in a similar manner to all of the other variables.`):
>
>       fi;
>
>       print();
>
>    else
>
>       print(`These variables will be taken to be those with which the`);
>
>       print(`expansion is to be performed  and with respect to which`);
>
>       print(`the subsequent variation is to be carried out.`);
>
>       print(nset); print();  # nset=varset for all scalars. #
>
```

```
>    fi;
>
> fi;
```

The lagrangian used will be:-

$$\frac{1}{2}\,\mathrm{epsln0}\,(dt(A)^2 - c^2\,dx(A)^2) - N\,dt(A)\,(\mathrm{dagger}(v)\ \&*\ (\mathrm{delmat}\ \&*\ v)) + I\,\mathrm{hbar}\,N\,(\mathrm{dagger}(v)\ \&*\ dt(v)) - (\mathrm{dagger}(v)\ \&*\ (\mathrm{emat}\ \&*\ v))$$

Only the fundamental will be used in the calculation.

The calculation assumes that the following quantities are

vector variables which will require subsequent expansion

{v}

and that these variables are the scalar variables

be expanded

{A}

The following quantities are arrays which are

assumed to have values given by the user.

{delmat, emat}

The groundstate of vector quantities will be

taken to be near unity.

```
>
> if prinlvl>2 then
>
>    print(`Calculation begins ...`);
>
> fi;
```

Calculation begins ...

```
>
# Convert sets to lists to maintain ordering.                          #
#
> nlist:=convert(nset,list); klist:=convert(kset,list);
>
# nlist:=[op(nset)]: klist:=[op(kset)]:
#
> tb:=table(): tc:=table():
>
> for i from 1 to nops(nlist) do;
>
>    assign(tb[i]=cat(nlist[i],_));
>
> od;
>
> for i from 1 to nops(klist) do;
>
>    assign(tc[i]=cat(klist[i],_));
>
> od;
>
```

```
> newnlist:=convert(tb,list):
>
> newklist:=convert(tc,list):
>
# op(newklist); op(newnlist);
#
> tb:='tb': tc:='tc':   # Unassign tb, tc                          #
>
> read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddag.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expdstar.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/expddson.mpl`;
>
#read `expdnc.mpl`;
> read `/user10/xt/xtb003/mplfiles/diffdtdx.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
>
# read `truncser.mpl`;
#
> read `/user10/xt/xtb003/mplfiles/orderser.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/sums.mpl`: # get summation procedures.
>   #
> read `/user10/xt/xtb003/mplfiles/average.mpl`;
>
# # read `smalexpd.mpl`; # Avoid `object too big.`                #
#
> read `/user10/xt/xtb003/mplfiles/smrtexpd.mpl`;
>
> readlib(evalm):
>
# Simplify the lagrangian to get rid of star(star(. etc.
#
# printlevel:=2000;
> L:=simplify(L,dagger);
>
> L:=simplify(L,star);
>
# printlevel:=-1;
# quit;
> if prinlvl>=3 then
>    print(`The lagrangian has now been simplifed with respect to dagger`);
>
>    print(`and star.`);
> fi;
          The lagrangian has now been simplifed with respect to dagger

                                    and star.

# printlevel:=2000;
> if prinlvl=4 then print(`The lagrangian is now`,L) fi;
The lagrangian is now,

1/2 epsln0 ((

    /    d     \                      2               3               4
- w |-------- A| + eps DT(A, 1) + eps  DT(A, 2) + eps  DT(A, 3) + eps  DT(A, 4)
```

```
    \ dtheta   /

       2
)^2 - c

   /    d     \                      2               3               4
(k |-------- A| + eps DX(A, 1) + eps  DX(A, 2) + eps  DX(A, 3) + eps  DX(A, 4))
   \ dtheta   /

^2) - N (

    /    d     \                     2               3               4
- w |-------- A| + eps DT(A, 1) + eps  DT(A, 2) + eps  DT(A, 3) + eps  DT(A, 4)
    \ dtheta   /

) (dagger(v) &* (delmat &* v)) + I hbar N (dagger(v) &* (

    /    d     \                     2               3               4
- w |-------- v| + eps DT(v, 1) + eps  DT(v, 2) + eps  DT(v, 3) + eps  DT(v, 4)
    \ dtheta   /

)) - (dagger(v) &* (emat &* v))

> if klist<>[] then
>
>    for i from 1 to nops(klist) do;
>
>       assign(op(i,newklist)=array( op(2,op(op(i,klist)))) );
>
>    od;
>
>    if hiharm=false then
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),quantsum(op(i,newklist)));
>
>       od;
>
>    else
>
>       for i from 1 to nops(klist) do;
>
>          assign(op(i,klist),bigquantsum(op(i,newklist)));
>
>       od;
>
>    fi;
>
> fi;
# printlevel:=-1;
#
> if hiharm=false then
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),weeharmsum(op(i,newnlist)));
>
>    od;
>
> else
>
>    for i from 1 to nops(nlist) do;
>
>       assign(op(i,nlist),bigharmsum(op(i,newnlist)));
```

```
>
>    od;
>
> fi;
>
# read in the Diff procedure now that the substitutions have been made.
#
> read `/user10/xt/xtb003/mplfiles/diftheta.mpl`;
>
> L:=simplify(L,DT,DX);
bytes used=1001184, alloc=327680, time=15.533
bytes used=2003588, alloc=450560, time=24.283
    Removed bytes used messages.
bytes used=17033136, alloc=942080, time=156.233
>
# Should now have made the necessary assignments. Matrices should not be
#
# empty so can carry out necessary calculations.
#
#expand(expandoff); expand(expandon); expandoff(exp);
#
#printlevel:=2000;
#
> if prinlvl>=3 then
>
>    print(`The arrays have now been filled and the summations`);
>
>    print(`required for the expansion carried out.`);
>
>    if prinlvl=4 then
>
>       print(`The lagrangian has become: `,L);
>
>    fi;
>
> fi;
              The arrays have now been filled and the summations

                   required for the expansion carried out.

The lagrangian has become: ,

1/2 epsln0 ((- w (eps (A_[0, 1] I exp(I theta) - %15 I exp(- I theta))

      2
 + eps  (A_[1, 1] I exp(I theta) - %12 I exp(- I theta))

      3
 + eps  (A_[2, 1] I exp(I theta) - %9 I exp(- I theta))

      4
 + eps  (A_[3, 1] I exp(I theta) - %6 I exp(- I theta))

      5
 + eps  (A_[4, 1] I exp(I theta) - %2 I exp(- I theta))) + eps (eps (

DT(%37) exp(I theta) + A_[0, 1] DT(%33) + star(DT(%37)) exp(- I theta)

                          2
 + %15 DT(%31)) + %14 + eps  (DT(%36) exp(I theta) + A_[1, 1] DT(%33)

                                                        3
 + star(DT(%36)) exp(- I theta) + %12 DT(%31)) + %11 + eps  (

DT(%35) exp(I theta) + A_[2, 1] DT(%33) + star(DT(%35)) exp(- I theta)
```

```
 + %9 DT(%31)) + %8 + eps^4 (DT(%34) exp(I theta) + A_[3, 1] DT(%33)

 + star(DT(%34)) exp(- I theta) + %6 DT(%31)) + %5 + eps^5 (DT(%32) exp(I theta)

 + A_[4, 1] DT(%33) + star(DT(%32)) exp(- I theta) + %2 DT(%31))) + eps^2 (eps (

DT(%30) exp(I theta) + A_[0, 1] DT(%26) + star(DT(%30)) exp(- I theta)

 + %15 DT(%24)) + %14 + eps^2 (DT(%29) exp(I theta) + A_[1, 1] DT(%26)

 + star(DT(%29)) exp(- I theta) + %12 DT(%24)) + %11 + eps^3 (

DT(%28) exp(I theta) + A_[2, 1] DT(%26) + star(DT(%28)) exp(- I theta)

 + %9 DT(%24)) + %8 + eps^4 (DT(%27) exp(I theta) + A_[3, 1] DT(%26)

 + star(DT(%27)) exp(- I theta) + %6 DT(%24)) + %5 + eps^5 (DT(%25) exp(I theta)

 + A_[4, 1] DT(%26) + star(DT(%25)) exp(- I theta) + %2 DT(%24))) + eps^3 (eps (

DT(%23) exp(I theta) + A_[0, 1] DT(%19) + star(DT(%23)) exp(- I theta)

 + %15 DT(%17)) + %14 + eps^2 (DT(%22) exp(I theta) + A_[1, 1] DT(%19)

 + star(DT(%22)) exp(- I theta) + %12 DT(%17)) + %11 + eps^3 (

DT(%21) exp(I theta) + A_[2, 1] DT(%19) + star(DT(%21)) exp(- I theta)

 + %9 DT(%17)) + %8 + eps^4 (DT(%20) exp(I theta) + A_[3, 1] DT(%19)

 + star(DT(%20)) exp(- I theta) + %6 DT(%17)) + %5 + eps^5 (DT(%18) exp(I theta)

 + A_[4, 1] DT(%19) + star(DT(%18)) exp(- I theta) + %2 DT(%17))) + eps^4 (eps (

DT(%16) exp(I theta) + A_[0, 1] DT(%4) + star(DT(%16)) exp(- I theta)

 + %15 DT(%1)) + %14 + eps^2 (DT(%13) exp(I theta) + A_[1, 1] DT(%4)

 + star(DT(%13)) exp(- I theta) + %12 DT(%1)) + %11 + eps^3 (

DT(%10) exp(I theta) + A_[2, 1] DT(%4) + star(DT(%10)) exp(- I theta)

 + %9 DT(%1)) + %8 + eps^4 (

DT(%7) exp(I theta) + A_[3, 1] DT(%4) + star(DT(%7)) exp(- I theta) + %6 DT(%1)

) + %5 + eps^5 (
```

```
DT(%3) exp(I theta) + A_[4, 1] DT(%4) + star(DT(%3)) exp(- I theta) + %2 DT(%1)

                    2
)))^2 - c  (k (eps (A_[0, 1] I exp(I theta) - %15 I exp(- I theta))

      2
 + eps  (A_[1, 1] I exp(I theta) - %12 I exp(- I theta))

      3
 + eps  (A_[2, 1] I exp(I theta) - %9 I exp(- I theta))

      4
 + eps  (A_[3, 1] I exp(I theta) - %6 I exp(- I theta))

      5
 + eps  (A_[4, 1] I exp(I theta) - %2 I exp(- I theta))) + eps (eps (

DX(%37) exp(I theta) + A_[0, 1] DX(%33) + star(DX(%37)) exp(- I theta)

                                  2
 + %15 DX(%31)) + %14 + eps  (DX(%36) exp(I theta) + A_[1, 1] DX(%33)

                                                              3
 + star(DX(%36)) exp(- I theta) + %12 DX(%31)) + %11 + eps  (

DX(%35) exp(I theta) + A_[2, 1] DX(%33) + star(DX(%35)) exp(- I theta)

                                  4
 + %9 DX(%31)) + %8 + eps  (DX(%34) exp(I theta) + A_[3, 1] DX(%33)

                                                          5
 + star(DX(%34)) exp(- I theta) + %6 DX(%31)) + %5 + eps  (DX(%32) exp(I theta)

                                                                  2
 + A_[4, 1] DX(%33) + star(DX(%32)) exp(- I theta) + %2 DX(%31))) + eps  (eps (

DX(%30) exp(I theta) + A_[0, 1] DX(%26) + star(DX(%30)) exp(- I theta)

                                  2
 + %15 DX(%24)) + %14 + eps  (DX(%29) exp(I theta) + A_[1, 1] DX(%26)

                                                              3
 + star(DX(%29)) exp(- I theta) + %12 DX(%24)) + %11 + eps  (

DX(%28) exp(I theta) + A_[2, 1] DX(%26) + star(DX(%28)) exp(- I theta)

                                  4
 + %9 DX(%24)) + %8 + eps  (DX(%27) exp(I theta) + A_[3, 1] DX(%26)

                                                          5
 + star(DX(%27)) exp(- I theta) + %6 DX(%24)) + %5 + eps  (DX(%25) exp(I theta)

                                                                  3
 + A_[4, 1] DX(%26) + star(DX(%25)) exp(- I theta) + %2 DX(%24))) + eps  (eps (

DX(%23) exp(I theta) + A_[0, 1] DX(%19) + star(DX(%23)) exp(- I theta)

                                  2
 + %15 DX(%17)) + %14 + eps  (DX(%22) exp(I theta) + A_[1, 1] DX(%19)

                                                              3
 + star(DX(%22)) exp(- I theta) + %12 DX(%17)) + %11 + eps  (

DX(%21) exp(I theta) + A_[2, 1] DX(%19) + star(DX(%21)) exp(- I theta)
```

```
 + %9 DX(%17)) + %8 + eps^4 (DX(%20) exp(I theta) + A_[3, 1] DX(%19)
 + star(DX(%20)) exp(- I theta) + %6 DX(%17)) + %5 + eps^5 (DX(%18) exp(I theta)
 + A_[4, 1] DX(%19) + star(DX(%18)) exp(- I theta) + %2 DX(%17))) + eps^4 (eps (
DX(%16) exp(I theta) + A_[0, 1] DX(%4) + star(DX(%16)) exp(- I theta)
 + %15 DX(%1)) + %14 + eps^2 (DX(%13) exp(I theta) + A_[1, 1] DX(%4)
 + star(DX(%13)) exp(- I theta) + %12 DX(%1)) + %11 + eps^3 (
DX(%10) exp(I theta) + A_[2, 1] DX(%4) + star(DX(%10)) exp(- I theta)
 + %9 DX(%1)) + %8 + eps^4 (
DX(%7) exp(I theta) + A_[3, 1] DX(%4) + star(DX(%7)) exp(- I theta) + %6 DX(%1)
) + %5 + eps^5 (
DX(%3) exp(I theta) + A_[4, 1] DX(%4) + star(DX(%3)) exp(- I theta) + %2 DX(%1)
)))^2) - N (- w (eps (A_[0, 1] I exp(I theta) - %15 I exp(- I theta))
 + eps^2 (A_[1, 1] I exp(I theta) - %12 I exp(- I theta))
 + eps^3 (A_[2, 1] I exp(I theta) - %9 I exp(- I theta))
 + eps^4 (A_[3, 1] I exp(I theta) - %6 I exp(- I theta))
 + eps^5 (A_[4, 1] I exp(I theta) - %2 I exp(- I theta))) + eps (eps (
DT(%37) exp(I theta) + A_[0, 1] DT(%33) + star(DT(%37)) exp(- I theta)
 + %15 DT(%31)) + %14 + eps^2 (DT(%36) exp(I theta) + A_[1, 1] DT(%33)
 + star(DT(%36)) exp(- I theta) + %12 DT(%31)) + %11 + eps^3 (
DT(%35) exp(I theta) + A_[2, 1] DT(%33) + star(DT(%35)) exp(- I theta)
 + %9 DT(%31)) + %8 + eps^4 (DT(%34) exp(I theta) + A_[3, 1] DT(%33)
 + star(DT(%34)) exp(- I theta) + %6 DT(%31)) + %5 + eps^5 (DT(%32) exp(I theta)
 + A_[4, 1] DT(%33) + star(DT(%32)) exp(- I theta) + %2 DT(%31))) + eps^2 (eps (
DT(%30) exp(I theta) + A_[0, 1] DT(%26) + star(DT(%30)) exp(- I theta)
```

```
                                                                    2
 + %15 DT(%24)) + %14 + eps  (DT(%29) exp(I theta) + A_[1, 1] DT(%26)

                                                                                3
 + star(DT(%29)) exp(- I theta) + %12 DT(%24)) + %11 + eps  (

DT(%28) exp(I theta) + A_[2, 1] DT(%26) + star(DT(%28)) exp(- I theta)

                                    4
 + %9 DT(%24)) + %8 + eps  (DT(%27) exp(I theta) + A_[3, 1] DT(%26)

                                                                          5
 + star(DT(%27)) exp(- I theta) + %6 DT(%24)) + %5 + eps  (DT(%25) exp(I theta)

                                                                                   3
 + A_[4, 1] DT(%26) + star(DT(%25)) exp(- I theta) + %2 DT(%24))) + eps  (eps (

DT(%23) exp(I theta) + A_[0, 1] DT(%19) + star(DT(%23)) exp(- I theta)

                                    2
 + %15 DT(%17)) + %14 + eps  (DT(%22) exp(I theta) + A_[1, 1] DT(%19)

                                                                          3
 + star(DT(%22)) exp(- I theta) + %12 DT(%17)) + %11 + eps  (

DT(%21) exp(I theta) + A_[2, 1] DT(%19) + star(DT(%21)) exp(- I theta)

                                    4
 + %9 DT(%17)) + %8 + eps  (DT(%20) exp(I theta) + A_[3, 1] DT(%19)

                                                                          5
 + star(DT(%20)) exp(- I theta) + %6 DT(%17)) + %5 + eps  (DT(%18) exp(I theta)

                                                                                   4
 + A_[4, 1] DT(%19) + star(DT(%18)) exp(- I theta) + %2 DT(%17))) + eps  (eps (

DT(%16) exp(I theta) + A_[0, 1] DT(%4) + star(DT(%16)) exp(- I theta)

                                    2
 + %15 DT(%1)) + %14 + eps  (DT(%13) exp(I theta) + A_[1, 1] DT(%4)

                                                                          3
 + star(DT(%13)) exp(- I theta) + %12 DT(%1)) + %11 + eps  (

DT(%10) exp(I theta) + A_[2, 1] DT(%4) + star(DT(%10)) exp(- I theta)

                                    4
 + %9 DT(%1)) + %8 + eps  (

DT(%7) exp(I theta) + A_[3, 1] DT(%4) + star(DT(%7)) exp(- I theta) + %6 DT(%1)

           5
) + %5 + eps  (

DT(%3) exp(I theta) + A_[4, 1] DT(%4) + star(DT(%3)) exp(- I theta) + %2 DT(%1)

))) (dagger(v) &* (delmat &* v)) + I hbar N

                                    2               3               4
(dagger(v) &* (eps DT(v, 1) + eps  DT(v, 2) + eps  DT(v, 3) + eps  DT(v, 4)))

 - (dagger(v) &* (emat &* v))

%1 :=

                                exp(- I theta), 4
```

```
%2 :=
                              star(A_[4, 1])

%3 :=
                                A_[4, 1], 4

%4 :=
                              exp(I theta), 4

%5 :=
                   5
              eps  (A_[4, 1] exp(I theta) + %2 exp(- I theta))

%6 :=
                              star(A_[3, 1])

%7 :=
                                A_[3, 1], 4

%8 :=
                   4
              eps  (A_[3, 1] exp(I theta) + %6 exp(- I theta))

%9 :=
                              star(A_[2, 1])

%10 :=
                                A_[2, 1], 4

%11 :=
                   3
              eps  (A_[2, 1] exp(I theta) + %9 exp(- I theta))

%12 :=
                              star(A_[1, 1])

%13 :=
                                A_[1, 1], 4

%14 :=
                   2
              eps  (A_[1, 1] exp(I theta) + %12 exp(- I theta))

%15 :=
                              star(A_[0, 1])

%16 :=
                                A_[0, 1], 4

%17 :=
                             exp(- I theta), 3

%18 :=
                                A_[4, 1], 3

%19 :=
                              exp(I theta), 3

%20 :=
                                A_[3, 1], 3

%21 :=
                                A_[2, 1], 3
```

```
%22 :=
                                    A_[1, 1], 3

%23 :=
                                    A_[0, 1], 3

%24 :=
                                 exp(- I theta), 2

%25 :=
                                    A_[4, 1], 2

%26 :=
                                  exp(I theta), 2

%27 :=
                                    A_[3, 1], 2

%28 :=
                                    A_[2, 1], 2

%29 :=
                                    A_[1, 1], 2

%30 :=
                                    A_[0, 1], 2

%31 :=
                                 exp(- I theta), 1

%32 :=
                                    A_[4, 1], 1

%33 :=
                                  exp(I theta), 1

%34 :=
                                    A_[3, 1], 1

%35 :=
                                    A_[2, 1], 1

%36 :=
                                    A_[1, 1], 1

%37 :=
                                    A_[0, 1], 1

>
# quit;
#
# printlevel:=2000;
> L:=expddagstonly(L);
bytes used=18033508, alloc=950272, time=165.666
                                      dagarr

                                       tttt

                                       tttt

                                       tttt

                                       tttt
```

```
>    fi;
>
> fi;
```

The lagrangian has been rearranged in terms of the small variable eps.

The lagrangian is now:

```
(1/2*epsln0*star(A_[0,1])**2*DT(exp(-I*theta),1)**2+1/2*epsln0*A_[0,1]**2*DT(
exp(I*theta),1)**2-En1*exp(-1/2*I*theta)**2*v_1[2,1]**2+1/2*epsln0*star(DT(A_[0
,1],1))**2/exp(I*theta)**2-En1*exp(-1/2*I*theta)**2*v_1[1,1]**2+epsln0*DT(A_[0,
1],1)*star(DT(A_[0,1],1))-2*I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[1,1],2)+I*
hbar*N*v_1[1,1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),2)+I*hbar*N*v_1[0,1
]/exp(I*theta)*DT(v_1[2,1],2)+2*epsln0*c**2*star(A_[1,1])*DX(exp(-I*theta),1)*k
*star(A_[0,1])*I/exp(I*theta)-epsln0*w*A_[1,1]*I*exp(I*theta)**2*DT(A_[0,1],1)+
epsln0*A_[0,1]*DT(exp(I*theta),1)*star(A_[0,1])*DT(exp(-I*theta),1)+epsln0*w*
star(A_[1,1])*I/exp(I*theta)**2*star(DT(A_[0,1],1))-En2*exp(1/2*I*theta)**2*v_2
[1,1]**2+I*hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),4)+I*
hbar*N*v_1[1,1]/exp(I*theta)*DT(v_1[2,1],1)+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(
v_1[0,1],4)+I*hbar*N*v_2[0,1]*exp(I*theta)*v_2[1,1]+2*I*hbar*N*v_1[1,1]/exp(I*
theta)**(1/2)*v_1[2,1]*DT(exp(-1/2*I*theta),1)-3*I*hbar*N*v_1[0,1]/exp(I*theta)
*v_1[2,1]+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[1,1],3)+I*hbar*N*v_1[0,1]**2/
exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),2)+2*I*hbar*N*v_2[0,1]*exp(I*theta)**(
1/2)*v_2[1,1]*DT(exp(1/2*I*theta),1)+I*hbar*N*v_1[2,1]/exp(I*theta)*DT(v_1[0,1]
,2)+I*hbar*N*v_2[1,1]*exp(I*theta)*DT(v_2[0,1],1)+I*hbar*N*v_1[2,1]/exp(I*theta
)*DT(v_1[1,1],1)-2*N*star(A_[0,1])*DT(exp(-I*theta),1)*del12*v_1[1,1]*v_2[0,1]+
I*hbar*N*v_2[0,1]*exp(I*theta)*DT(v_2[0,1],2)-2*I*hbar*N*v_1[1,1]/exp(I*theta)*
DT(v_1[1,1],1)-2*N*star(A_[0,1])*DT(exp(-I*theta),1)*del12*v_1[0,1]*v_2[1,1]+I*
hbar*N*v_2[0,1]**2*exp(I*theta)**(1/2)*DT(exp(1/2*I*theta),2)-2*I*hbar*N*v_1[0,
1]/exp(I*theta)*DT(v_1[2,1],1)-2*I*hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2)*DT(
exp(-1/2*I*theta),3)+2*I*hbar*N*v_1[1,1]/exp(I*theta)**(1/2)*v_1[0,1]*DT(exp(-1
/2*I*theta),3)-2*I*hbar*N*v_1[1,1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),
1)+I*hbar*N*v_1[1,1]/exp(I*theta)*DT(v_1[0,1],1)-2*I*hbar*N*v_1[2,1]/exp(I*
theta)*DT(v_1[0,1],1)+2*N*w*A_[1,1]*I*exp(I*theta)*del12*v_1[0,1]*v_2[1,1]+2*N*
w*A_[1,1]*I*exp(I*theta)*del12*v_1[1,1]*v_2[0,1]-2*N*w*star(A_[0,1])*I/exp(I*
theta)*del12*v_1[0,1]*v_2[2,1]+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[3,1],1)+2*
N*w*star(A_[0,1])*I/exp(I*theta)*del12*v_1[1,1]*v_2[0,1]+I*hbar*N*v_2[0,1]*exp(
I*theta)*DT(v_2[1,1],1)-2*En2*exp(1/2*I*theta)**2*v_2[0,1]*v_2[2,1]+2*N*w*star(
A_[0,1])*I/exp(I*theta)*del12*v_1[0,1]*v_2[1,1]-2*N*w*star(A_[0,1])*I/exp(I*
theta)*del12*v_1[2,1]*v_2[0,1]+2*N*w*A_[0,1]*I*exp(I*theta)*del12*v_1[0,1]*v_2[
2,1]-2*N*w*star(A_[0,1])*I/exp(I*theta)*del12*v_1[1,1]*v_2[1,1]-4*I*hbar*N*v_1[
0,1]/exp(I*theta)**(1/2)*v_1[2,1]*DT(exp(-1/2*I*theta),1)-2*N*w*A_[0,1]*I*exp(I
*theta)*del12*v_1[0,1]*v_2[1,1]+2*N*w*A_[0,1]*I*exp(I*theta)*del12*v_1[1,1]*v_2
[1,1]-2*En1*exp(-1/2*I*theta)**2*v_1[1,1]*v_1[3,1]+4*En1*exp(-1/2*I*theta)**2*
v_1[0,1]*v_1[3,1]-2*N*w*A_[0,1]*I*exp(I*theta)*del12*v_1[1,1]*v_2[0,1]+2*N*w*A_
[0,1]*I*exp(I*theta)*del12*v_1[2,1]*v_2[0,1]-I*hbar*N*v_1[1,1]**2/exp(I*theta)-
2*I*hbar*N*v_1[1,1]/exp(I*theta)*DT(v_1[0,1],2)+I*hbar*N*v_1[0,1]/exp(I*theta)*
DT(v_1[0,1],2)-2*I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[0,1],3)+2*I*hbar*N*v_1[0
,1]/exp(I*theta)**(1/2)*v_1[1,1]*DT(exp(-1/2*I*theta),1)-2*N*star(DT(A_[1,1],1)
)/exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-2*N*star(A_[1,1])*DT(exp(-I*theta),1)*
del12*v_1[0,1]*v_2[0,1]-2*N*w*star(A_[1,1])*I/exp(I*theta)*del12*v_1[0,1]*v_2[1
,1]-2*N*A_[1,1]*DT(exp(I*theta),1)*del12*v_1[0,1]*v_2[0,1]+epsln0*A_[1,1]*DT(
exp(I*theta),1)*w*star(A_[0,1])*I/exp(I*theta)-2*N*DT(A_[0,1],1)*exp(I*theta)*
del12*v_1[1,1]*v_2[0,1]+2*epsln0*star(A_[1,1])*DT(exp(-I*theta),1)*w*star(A_[0,
1])*I/exp(I*theta)-epsln0*w*A_[0,1]*I*exp(I*theta)**2*DT(A_[1,1],1)-2*N*star(DT
(A_[0,1],1))/exp(I*theta)*del12*v_1[1,1]*v_2[0,1]-2*N*A_[0,1]*DT(exp(I*theta),1
)*del12*v_1[1,1]*v_2[0,1]-2*N*DT(A_[1,1],1)*exp(I*theta)*del12*v_1[0,1]*v_2[0,1
]-2*N*star(DT(A_[0,1],1))/exp(I*theta)*del12*v_1[0,1]*v_2[1,1]-2*N*star(DT(A_[0
,1],2))/exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-2*N*star(A_[0,1])*DT(exp(-I*theta)
,2)*del12*v_1[0,1]*v_2[0,1]+3*I*hbar*N*v_1[1,1]/exp(I*theta)*v_1[2,1]-2*N*DT(A_
[0,1],2)*exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-2*N*w*star(A_[1,1])*I/exp(I*theta
)*del12*v_1[1,1]*v_2[0,1]+2*I*hbar*N*v_1[0,1]/exp(I*theta)**(1/2)*v_1[3,1]*DT(
exp(-1/2*I*theta),1)-2*N*A_[0,1]*DT(exp(I*theta),1)*del12*v_1[0,1]*v_2[1,1]-2*N
*DT(A_[0,1],1)*exp(I*theta)*del12*v_1[0,1]*v_2[1,1]-2*N*star(A_[1,1])/exp(I*
```

```
theta)*del12*v_1[0,1]*v_2[0,1]-2*N*w*A_[1,1]*I*exp(I*theta)*del12*v_1[0,1]*v_2[
0,1]+2*N*star(A_[0,1])*DT(exp(-I*theta),1)*del12*v_1[0,1]*v_2[0,1]-2*N*A_[1,1]*
exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-epsln0*c**2*star(A_[0,1])*DX(exp(-I*theta)
,2)*k*A_[0,1]*I*exp(I*theta)-epsln0*c**2*star(DX(A_[0,1],1))/exp(I*theta)*A_[0,
1]*DX(exp(I*theta),1)+epsln0*c**2*k*star(A_[1,1])*I/exp(I*theta)*A_[0,1]*DX(exp
(I*theta),1)+epsln0*c**2*k*star(A_[0,1])*I*A_[1,1]-epsln0*c**2*star(A_[0,1])*DX
(exp(-I*theta),1)*k*A_[1,1]*I*exp(I*theta)-2*N*A_[0,1]*DT(exp(I*theta),2)*del12
*v_1[0,1]*v_2[0,1]+epsln0*c**2*k*star(A_[0,1])*I*DX(A_[0,1],2)-epsln0*c**2*k*A_
[0,1]*I*exp(I*theta)**2*A_[1,1]+epsln0*DT(A_[0,1],2)*w*star(A_[0,1])*I-1/2*
epsln0*c**2*k**2*star(A_[1,1])**2*I**2/exp(I*theta)**2-1/2*epsln0*c**2*k**2*A_[
1,1]**2*I**2*exp(I*theta)**2+epsln0*c**2*k**2*A_[1,1]*I**2*star(A_[1,1])-epsln0
*c**2*k**2*star(A_[2,1])*I**2/exp(I*theta)**2*star(A_[0,1])+epsln0*c**2*k**2*A_
[2,1]*I**2*star(A_[0,1])-epsln0*c**2*DX(A_[1,1],1)*exp(I*theta)**2*k*A_[0,1]*I-
epsln0*c**2*star(DX(A_[0,1],1))*k*A_[1,1]*I+epsln0*c**2*k**2*star(A_[2,1])*I**2
*A_[0,1]+epsln0*c**2*star(DX(A_[0,1],1))/exp(I*theta)**2*k*star(A_[1,1])*I+2*N*
DT(A_[0,1],1)*exp(I*theta)*del12*v_1[0,1]*v_2[0,1]+epsln0*c**2*star(A_[0,1])**2
*DX(exp(-I*theta),2)*k*I/exp(I*theta)+2*N*A_[0,1]*DT(exp(I*theta),1)*del12*v_1[
0,1]*v_2[0,1]-epsln0*c**2*k*A_[0,1]*I*exp(I*theta)**2*DX(A_[0,1],2)+2*N*w*A_[2,
1]*I*exp(I*theta)*del12*v_1[0,1]*v_2[0,1]+2*N*star(DT(A_[0,1],1))/exp(I*theta)*
del12*v_1[0,1]*v_2[0,1]-1/2*epsln0*c**2*DX(A_[0,1],1)**2*exp(I*theta)**2-2*
epsln0*c**2*A_[1,1]*DX(exp(I*theta),1)*k*A_[0,1]*I*exp(I*theta)-epsln0*c**2*
star(A_[0,1])*DX(exp(-I*theta),1)*A_[0,1]*DX(exp(I*theta),1)-1/2*epsln0*c**2*
star(DX(A_[0,1],1))**2/exp(I*theta)**2+epsln0*c**2*star(DX(A_[1,1],1))/exp(I*
theta)**2*k*star(A_[0,1])*I+epsln0*c**2*A_[1,1]*DX(exp(I*theta),1)*k*star(A_[0,
1])*I/exp(I*theta)-2*N*w*star(A_[2,1])*I/exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-2
*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1[4,1]+epsln0*star(A_[1,1])/exp(I*theta)**
2*w*star(A_[0,1])*I+epsln0*c**2*DX(A_[0,1],1)*k*star(A_[1,1])*I-epsln0*c**2*
star(DX(A_[1,1],1))*k*A_[0,1]*I-epsln0*c**2*A_[0,1]**2*DX(exp(I*theta),2)*k*I*
exp(I*theta)-epsln0*c**2*DX(A_[0,1],1)*exp(I*theta)*A_[0,1]*DX(exp(I*theta),1)+
epsln0*c**2*star(A_[1,1])/exp(I*theta)**2*k*star(A_[0,1])*I+epsln0*A_[0,1]*DT(
exp(I*theta),2)*w*star(A_[0,1])*I/exp(I*theta)-epsln0*c**2*DX(A_[0,1],1)*star(
DX(A_[0,1],1))+epsln0*c**2*A_[0,1]*DX(exp(I*theta),2)*k*star(A_[0,1])*I/exp(I*
theta)-epsln0*c**2*star(DX(A_[0,1],1))/exp(I*theta)*star(A_[0,1])*DX(exp(-I*
theta),1)-epsln0*c**2*star(A_[1,1])*k*A_[0,1]*I-epsln0*c**2*star(DX(A_[0,1],2))
*k*A_[0,1]*I+epsln0*c**2*star(DX(A_[0,1],2))/exp(I*theta)**2*k*star(A_[0,1])*I-
epsln0*c**2*DX(A_[0,1],1)*exp(I*theta)**2*k*A_[1,1]*I+epsln0*c**2*DX(A_[1,1],1)
*k*star(A_[0,1])*I-epsln0*c**2*k**2*A_[2,1]*I**2*exp(I*theta)**2*A_[0,1]+epsln0
*star(A_[0,1])**2*DT(exp(-I*theta),2)*w*I/exp(I*theta)-epsln0*c**2*DX(A_[0,1],1
)*exp(I*theta)*star(A_[0,1])*DX(exp(-I*theta),1)+1/2*epsln0*w**2*star(A_[1,1])
**2*I**2/exp(I*theta)**2-epsln0*DT(A_[0,1],2)*exp(I*theta)**2*w*A_[0,1]*I+
epsln0*w**2*A_[0,1]*I**2*exp(I*theta)**2*A_[2,1]-epsln0*star(A_[1,1])*DT(exp(-I
*theta),1)*w*A_[0,1]*I*exp(I*theta)+epsln0*star(DT(A_[0,1],2))/exp(I*theta)**2*
w*star(A_[0,1])*I-epsln0*star(DT(A_[0,1],2))*w*A_[0,1]*I+1/2*epsln0*w**2*A_[1,1
]**2*I**2*exp(I*theta)**2-epsln0*w**2*A_[2,1]*I**2*star(A_[0,1])-epsln0*w**2*A_
[0,1]*I**2*star(A_[2,1])-epsln0*star(DT(A_[1,1],1))*w*A_[0,1]*I+epsln0*w*star(
A_[1,1])*I/exp(I*theta)*A_[0,1]*DT(exp(I*theta),1)-epsln0*star(A_[0,1])*DT(exp(
-I*theta),2)*w*A_[0,1]*I*exp(I*theta)-epsln0*A_[0,1]**2*DT(exp(I*theta),2)*w*I*
exp(I*theta)-epsln0*star(A_[1,1])*w*A_[0,1]*I-epsln0*w*A_[0,1]*I*exp(I*theta)**
2*A_[1,1]+4*En1*exp(-1/2*I*theta)**2*v_1[1,1]*v_1[2,1]+3*I*hbar*N*v_1[0,1]/exp(
I*theta)*v_1[3,1]+I*hbar*N*v_1[1,1]/exp(I*theta)*DT(v_1[1,1],2)+epsln0*star(DT(
A_[1,1],1))/exp(I*theta)**2*w*star(A_[0,1])*I+2*N*w*star(A_[1,1])*I/exp(I*theta
)*del12*v_1[0,1]*v_2[0,1]+I*hbar*N*v_1[1,1]/exp(I*theta)*DT(v_1[0,1],3)-2*
epsln0*A_[1,1]*DT(exp(I*theta),1)*w*A_[0,1]*I*exp(I*theta)-2*En1*exp(-1/2*I*
theta)**2*v_1[0,1]*v_1[2,1]+epsln0*w*star(A_[1,1])*I*DT(A_[0,1],1)+I*hbar*N*v_1
[3,1]/exp(I*theta)*DT(v_1[0,1],1)+epsln0*DT(A_[0,1],1)*exp(I*theta)*star(A_[0,1
])*DT(exp(-I*theta),1)+epsln0*A_[0,1]*DT(exp(I*theta),1)*star(DT(A_[0,1],1))/
exp(I*theta)+epsln0*w*star(A_[0,1])*I*DT(A_[1,1],1)-epsln0*w**2*A_[1,1]*I**2*
star(A_[1,1])-epsln0*w*A_[1,1]*I*exp(I*theta)*star(A_[0,1])*DT(exp(-I*theta),1)
-1/2*epsln0*c**2*A_[0,1]**2*DX(exp(I*theta),1)**2-epsln0*w*A_[1,1]*I*star(DT(A_
[0,1],1))+epsln0*DT(A_[0,1],1)*exp(I*theta)*A_[0,1]*DT(exp(I*theta),1)+epsln0*
star(DT(A_[0,1],1))/exp(I*theta)*star(A_[0,1])*DT(exp(-I*theta),1)+epsln0*w**2*
star(A_[2,1])*I**2/exp(I*theta)**2*star(A_[0,1])+epsln0*w*star(A_[0,1])*I*A_[1,
1]-1/2*epsln0*c**2*star(A_[0,1])**2*DX(exp(-I*theta),1)**2+2*I*hbar*N*v_1[0,1]/
exp(I*theta)**(1/2)*v_1[2,1]*DT(exp(-1/2*I*theta),2)-epsln0*c**2*star(A_[1,1])*
DX(exp(-I*theta),1)*k*A_[0,1]*I*exp(I*theta)+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(
```

```
v_1[1,1],1)-4*I*hbar*N*v_1[0,1]/exp(I*theta)**(1/2)*v_1[1,1]*DT(exp(-1/2*I*
theta),2)+1/2*epsln0*DT(A_[0,1],1)**2*exp(I*theta)**2)*eps**4+(2*En1*exp(-1/2*I
*theta)**2*v_1[1,1]**2+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[1,1],2)+I*hbar*N*
v_2[0,1]**2*exp(I*theta)**(1/2)*DT(exp(1/2*I*theta),1)-2*I*hbar*N*v_1[0,1]/exp(
I*theta)*v_1[1,1]+I*hbar*N*v_2[0,1]*exp(I*theta)*DT(v_2[0,1],1)+I*hbar*N*v_1[0,
1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),1)-2*En2*exp(1/2*I*theta)**2*v_2
[0,1]*v_2[1,1]+epsln0*w**2*A_[0,1]*I**2*exp(I*theta)**2*A_[1,1]+I*hbar*N*v_1[0,
1]/exp(I*theta)*DT(v_1[0,1],1)-epsln0*w*A_[0,1]*I*exp(I*theta)*star(A_[0,1])*DT
(exp(-I*theta),1)-epsln0*c**2*k**2*star(A_[1,1])*I**2/exp(I*theta)**2*star(A_[0
,1])+2*N*w*star(A_[0,1])*I/exp(I*theta)*del12*v_1[0,1]*v_2[0,1]-2*N*w*A_[0,1]*I
*exp(I*theta)*del12*v_1[0,1]*v_2[0,1]+epsln0*c**2*k**2*A_[1,1]*I**2*star(A_[0,1
])-epsln0*c**2*k*A_[0,1]**2*I*exp(I*theta)*DX(exp(I*theta),1)+epsln0*c**2*k**2*
star(A_[1,1])*I**2*A_[0,1]-epsln0*c**2*star(DX(A_[0,1],1))*k*A_[0,1]*I+epsln0*c
**2*star(A_[0,1])**2*DX(exp(-I*theta),1)*k*I/exp(I*theta)+epsln0*c**2*DX(A_[0,1
],1)*k*star(A_[0,1])*I-epsln0*w*A_[0,1]*I*exp(I*theta)**2*DT(A_[0,1],1)-epsln0*
c**2*DX(A_[0,1],1)*exp(I*theta)**2*k*A_[0,1]*I+epsln0*c**2*star(DX(A_[0,1],1))/
exp(I*theta)**2*k*star(A_[0,1])*I-epsln0*c**2*k**2*A_[1,1]*I**2*exp(I*theta)**2
*A_[0,1]-epsln0*c**2*star(A_[0,1])*DX(exp(-I*theta),1)*k*A_[0,1]*I*exp(I*theta)
+epsln0*c**2*k*star(A_[0,1])*I/exp(I*theta)*A_[0,1]*DX(exp(I*theta),1)+epsln0*w
*star(A_[0,1])**2*I/exp(I*theta)*DT(exp(-I*theta),1)+epsln0*w*star(A_[0,1])*I/
exp(I*theta)*A_[0,1]*DT(exp(I*theta),1)+epsln0*w**2*star(A_[0,1])*I**2/exp(I*
theta)**2*star(A_[1,1])-epsln0*w*A_[0,1]**2*I*exp(I*theta)*DT(exp(I*theta),1)-2
*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1[1,1]+3*I*hbar*N*v_1[0,1]/exp(I*theta)*
v_1[2,1]-2*I*hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),2)+
epsln0*w*star(A_[0,1])*I/exp(I*theta)**2*star(DT(A_[0,1],1))+I*hbar*N*v_1[1,1]/
exp(I*theta)*DT(v_1[1,1],1)+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[2,1],1)+I*
hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),3)+I*hbar*N*v_1[1,1
]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),1)-2*I*hbar*N*v_1[1,1]/exp(I*
theta)*DT(v_1[0,1],1)+I*hbar*N*v_1[2,1]/exp(I*theta)*DT(v_1[0,1],1)-2*N*w*star(
A_[0,1])*I/exp(I*theta)*del12*v_1[1,1]*v_2[0,1]-2*N*w*star(A_[0,1])*I/exp(I*
theta)*del12*v_1[0,1]*v_2[1,1]+2*I*hbar*N*v_1[0,1]/exp(I*theta)**(1/2)*v_1[2,1]
*DT(exp(-1/2*I*theta),1)+2*N*w*A_[0,1]*I*exp(I*theta)*del12*v_1[0,1]*v_2[1,1]-2
*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1[3,1]+2*N*w*A_[0,1]*I*exp(I*theta)*del12*
v_1[1,1]*v_2[0,1]+I*hbar*N*v_1[1,1]**2/exp(I*theta)+I*hbar*N*v_1[1,1]/exp(I*
theta)*DT(v_1[0,1],2)-2*I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[0,1],2)+I*hbar*N*
v_1[0,1]/exp(I*theta)*DT(v_1[0,1],3)-4*I*hbar*N*v_1[0,1]/exp(I*theta)**(1/2)*
v_1[1,1]*DT(exp(-1/2*I*theta),1)+2*N*w*A_[1,1]*I*exp(I*theta)*del12*v_1[0,1]*
v_2[0,1]-2*N*star(A_[0,1])*DT(exp(-I*theta),1)*del12*v_1[0,1]*v_2[0,1]-epsln0*w
**2*star(A_[0,1])*I**2*A_[1,1]+epsln0*w*star(A_[0,1])*I*DT(A_[0,1],1)-epsln0*w*
A_[0,1]*I*star(DT(A_[0,1],1))-2*N*DT(A_[0,1],1)*exp(I*theta)*del12*v_1[0,1]*v_2
[0,1]-epsln0*w**2*A_[0,1]*I**2*star(A_[1,1])-2*N*A_[0,1]*DT(exp(I*theta),1)*
del12*v_1[0,1]*v_2[0,1]-2*N*star(DT(A_[0,1],1))/exp(I*theta)*del12*v_1[0,1]*v_2
[0,1]-2*En1*exp(-1/2*I*theta)**2*v_1[1,1]*v_1[2,1]-2*N*w*star(A_[1,1])*I/exp(I*
theta)*del12*v_1[0,1]*v_2[0,1]+4*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1[2,1]-2*I
*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[1,1],1)+2*I*hbar*N*v_1[0,1]/exp(I*theta)**
(1/2)*v_1[1,1]*DT(exp(-1/2*I*theta),2))*eps**3+(2*I*hbar*N*v_1[0,1]/exp(I*theta
)**(1/2)*v_1[1,1]*DT(exp(-1/2*I*theta),1)-2*N*w*star(A_[0,1])*I/exp(I*theta)*
del12*v_1[0,1]*v_2[0,1]-En2*exp(1/2*I*theta)**2*v_2[0,1]**2+2*N*w*A_[0,1]*I*exp
(I*theta)*del12*v_1[0,1]*v_2[0,1]-1/2*epsln0*c**2*k**2*star(A_[0,1])**2*I**2/
exp(I*theta)**2-2*I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[0,1],1)+I*hbar*N*v_1[0,
1]**2/exp(I*theta)**(1/2)*DT(exp(-1/2*I*theta),2)+I*hbar*N*v_1[0,1]/exp(I*theta
)*DT(v_1[1,1],1)-En1*exp(-1/2*I*theta)**2*v_1[0,1]**2-En1*exp(-1/2*I*theta)**2*
v_1[1,1]**2+1/2*epsln0*w**2*A_[0,1]**2*I**2*exp(I*theta)**2+4*En1*exp(-1/2*I*
theta)**2*v_1[0,1]*v_1[1,1]-2*I*hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2)*DT(exp(-
1/2*I*theta),1)+I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[0,1],2)+I*hbar*N*v_1[1,1]
/exp(I*theta)*DT(v_1[0,1],1)-epsln0*w**2*A_[0,1]*I**2*star(A_[0,1])+1/2*epsln0*
w**2*star(A_[0,1])**2*I**2/exp(I*theta)**2+epsln0*c**2*k**2*star(A_[0,1])*I**2*
A_[0,1]-2*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1[2,1]-I*hbar*N*v_1[0,1]**2/exp(I
*theta)+2*I*hbar*N*v_1[0,1]/exp(I*theta)*v_1[1,1]-1/2*epsln0*c**2*k**2*A_[0,1]
**2*I**2*exp(I*theta)**2)*eps**2+(I*hbar*N*v_1[0,1]/exp(I*theta)*DT(v_1[0,1],1)
+2*En1*exp(-1/2*I*theta)**2*v_1[0,1]**2-2*En1*exp(-1/2*I*theta)**2*v_1[0,1]*v_1
[1,1]+I*hbar*N*v_1[0,1]**2/exp(I*theta)+I*hbar*N*v_1[0,1]**2/exp(I*theta)**(1/2
)*DT(exp(-1/2*I*theta),1))*eps-En1*exp(-1/2*I*theta)**2*v_1[0,1]**2
>
> save `/scratch/xt/xtb003/qn2/avlagq4t.m`;
```

```
bytes used=139571608, alloc=3686400, time=1691.350
>
# lprint(L);
> for i from 0 to JMAX do; #4 do;
>    L.i:=coeff(L,`eps`,i);
>
>    L.i.`bar`:=average(L.i);
>
>    LC.i.`bar`:=evalc(L.i.`bar`);
>    print(`The averaged lagrangian at order`,i,` is:`);
>
>    print(LC.i.`bar`);
> od;
                    The averaged lagrangian at order, 0,  is:

                                        0

                    The averaged lagrangian at order, 1,  is:

                                        0

                    The averaged lagrangian at order, 2,  is:

          2                                          2  2
    epsln0 w  A_[0, 1] star(A_[0, 1]) - epsln0 c  k  A_[0, 1] star(A_[0, 1])

                    The averaged lagrangian at order, 3,  is:

                2  2                              2  2
      - epsln0 c  k  A_[1, 1] %2 - epsln0 c  k  star(A_[1, 1]) A_[0, 1]

                     2                              2
           + epsln0 w  A_[1, 1] %2 + epsln0 w  star(A_[1, 1]) A_[0, 1] + (

                    2                                   2
          - epsln0 c  star(DX(%1)) k A_[0, 1] + epsln0 c  DX(%1) k %2

           + epsln0 w %2 DT(%1) - epsln0 w A_[0, 1] star(DT(%1))) I

%1 :=
                                    A_[0, 1], 1

%2 :=
                                  star(A_[0, 1])

                    The averaged lagrangian at order, 4,  is:

                                          2  2
    epsln0 DT(%2) star(DT(%2)) - epsln0 c  k  A_[1, 1] %4

                2  2                              2  2
       - epsln0 c  k  A_[2, 1] %1 - epsln0 c  k  star(A_[2, 1]) A_[0, 1]

                    2                                  2
       - epsln0 c  DX(%2) star(DX(%2)) + epsln0 w  A_[2, 1] %1

                 2                                   2
       + epsln0 w  star(A_[2, 1]) A_[0, 1] + epsln0 w  A_[1, 1] %4 + (

              2                              2
      epsln0 c  k A_[1, 1] %1 + epsln0 c  k %1 DX(%5) + epsln0 DT(%5) w %1

                    2                                   2
       - epsln0 c  star(DX(%2)) k A_[1, 1] + epsln0 c  DX(%2) k %4
```

```
                2                                          2
        - epsln0 c  star(DX(%3)) k A_[0, 1] - epsln0 c  %4 k A_[0, 1]

                2                                          2
        - epsln0 c  star(DX(%5)) k A_[0, 1] + epsln0 c  DX(%3) k %1

        - epsln0 star(DT(%5)) w A_[0, 1] - epsln0 star(DT(%3)) w A_[0, 1]

        - epsln0 %4 w A_[0, 1] + epsln0 w %4 DT(%2) + epsln0 w %1 DT(%3)

        - epsln0 w A_[1, 1] star(DT(%2)) + epsln0 w A_[1, 1] %1) I

%1 :=
                                  star(A_[0, 1])

%2 :=
                                    A_[0, 1], 1

%3 :=
                                    A_[1, 1], 1

%4 :=
                                  star(A_[1, 1])

%5 :=
                                    A_[0, 1], 2

>
> save `/scratch/xt/xtb003/qn2/tavlag4q.m`;
# quit;
#
# This was originally file eqnmotq2.mpl
# trying to make the equations of motion. #
# JMAX:=6; prinlvl:=4;
# v_:=array(1..2,[]); #array bound required for looping.
# newnlist:=[eta_]; newklist:=[v_]; hiharm:=false;
# LC0bar:=0; LC1bar:=0;
# read `/user10/xt/xtb003/results/averq2.m`;
# printlevel:=-1;
# Define LC(2-4)bar;
> read `/user10/xt/xtb003/mplfiles/ordindex.mpl`;
# read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
> read `/user10/xt/xtb003/mplfiles/dordbool.mpl`;
> read `/user10/xt/xtb003/mplfiles/mapmod.mpl`;
> read `/user10/xt/xtb003/mplfiles/eqnmot.mpl`;
> read `/user10/xt/xtb003/mplfiles/derivord.mpl`;
> read `/user10/xt/xtb003/mplfiles/makeELterm.mpl`;
> read `/user10/xt/xtb003/mplfiles/doprint2.mpl`;
>
> read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
> read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
# Have now read in necessary files.
>
> for levcount from 0 to JMAX do;
>    equationsofmotion(`LC`.levcount.`bar`);
# Create DTvar, DXvar, DXDTvar as sets.
>    DTvar:=convert(DTvar,`list`);
>    DXvar:=convert(DXvar,`list`);
>    DXDTvar:=convert(DXDTvar,`list`);
>
>    scalvar:=convert(scalvar,`list`);
>
>    vecvar:=convert(vecvar,`list`);
>
>    newDTvar:=sort(DTvar,dordbool);
```

```
>
>    newDXvar:=sort(DXvar,dordbool);
>
>    newDXDTvar:=sort(DXDTvar,dordbool);
>
#   printlevel:=2000;
>    newscalvar:=sort(scalvar,fordbool);
>    newvecvar:=sort(vecvar,fordbool);
#   printlevel:=-1;
>    if prinlvl>3 then
>       print(`op(newscalvar)`,op(newscalvar));
>       print(`op(newvecvar)`,op(newvecvar) );
>       print(`op(newDTvar)`,op(newDTvar));
>       print(`op(newDXvar)`,op(newDXvar));
>       print(`op(newDXDTvar)`,op(newDXDTvar));
>    fi;
# printlevel:=2000;
>    lprint(`The lagrangian being used is :-`);
>    lprint(`LC`.levcount.`bar`);
# Generating the equations starts here.
>    scalareqnset:={}; vectoreqnset:={}; # Put the names of the arrays
# which will hold the equations in these two sets.
>    if nops(newscalvar)<>0 then
#      print(newscalvar,`TATEST3`);
>       for var in newscalvar do;
# Generate Euler-Lagrange for scalar variables.
>       print(`Investigating variable `,var,`at level`,levcount);
>       scalname:=op(0,var);
>       if not assigned(`eqtn`.levcount.scalname) then
>          scalareqnset:=scalareqnset union {`eqtn`.levcount.scalname};
>          `eqtn`.levcount.scalname:=table(sparse,[]) fi;
#      printlevel:=101; print(`101 #5`);
>       `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>           makeELterm(var,`LC`.levcount.`bar`);
#      printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #6`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #7`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
#      printlevel:=101; print(`101 #8`);
>                `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
```

```
>             fi;
>          od;
>       fi;
>       od;
#      print(`eqtn`.levcount.op(0,var)[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
>    if nops(newvecvar)<>0 then
#      print(newvecvar,`TATEST4`);
>       for var in newvecvar do;
#         printlevel:=11;
# Generate Euler-Lagrange for vector variables.
>       print(`Investigating vector variable`,var,`at level`,levcount);
>       vecname:=op(0,var);
>       if not assigned(`eqtn`.levcount.vecname) then
>          vectoreqnset:=vectoreqnset union {`eqtn`.levcount.vecname};
>          `eqtn`.levcount.vecname:=table(sparse,[]) fi;
# printlevel:=101; print(`101 #1`);
>       `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>           makeELterm(var,`LC`.levcount.`bar`);
# printlevel:=-1;
>       if nops(newDTvar)<>0 then
>          for var2 in newDTvar do;
>             if has(var2,var) then
#              printlevel:=101;print(`101 #2`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#               printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXvar)<>0 then
>          for var2 in newDXvar do;
>             if has(var2,var) then
#           printlevl:=101; print(`101 #3`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#      printlevel:=-1;
>             fi;
>          od;
>       fi;
>       if nops(newDXDTvar)<>0 then
>          for var2 in newDXDTvar do;
>             if has(var2,var) then
#             printlvel:=101; print(`101 #4`);
>                `eqtn`.levcount.vecname[op(1,var),op(2,`ar)]:=
>                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
>                   makeELterm(var2,`LC`.levcount.`bar`);
#     printlevel:=-1;
>             fi;
>          od;
>       fi;
>       od;
#      printlevel:=-1;
#      print(`eqtn`.levcount.vecname[op(1,var),op(2,var)]);
#      print(`=0`);
>    fi;
#   printlevel:=11;
>    doprinteqns();
> od; # end of outer loop.
                                  op(newscalvar)

                                  op(newvecvar)
```

```
                                    op(newDTvar)

                                    op(newDXvar)

                                   op(newDXDTvar)

The lagrangian being used is :-
0
                                   op(newscalvar)

                                   op(newvecvar)

                                    op(newDTvar)

                                    op(newDXvar)

                                   op(newDXDTvar)

The lagrangian being used is :-
0
                              op(newscalvar), A_[0, 1]

                                   op(newvecvar)

                                    op(newDTvar)

                                    op(newDXvar)

                                   op(newDXDTvar)

The lagrangian being used is :-
epsln0*w**2*A_[0,1]*star(A_[0,1])-epsln0*c**2*k**2*A_[0,1]*star(A_[0,1])
                  Investigating variable , A_[0, 1], at level, 2

              The equations of motion for the scalar variables are:-


                    The equation of motion for , A_[0,1],  is:-

                           2                      2  2
                  epsln0 w  A_[0, 1] - epsln0 c  k  A_[0, 1], =0


bytes used=140574080, alloc=3686400, time=1710.566
                        op(newscalvar), A_[0, 1], A_[1, 1]

                                   op(newvecvar)

                             op(newDTvar), DT(A_[0, 1], 1)

                             op(newDXvar), DX(A_[0, 1], 1)

                                   op(newDXDTvar)

The lagrangian being used is :-
-epsln0*c**2*k**2*A_[1,1]*star(A_[0,1])-epsln0*c**2*k**2*star(A_[1,1])*A_[0,1]+
epsln0*w**2*A_[1,1]*star(A_[0,1])+epsln0*w**2*star(A_[1,1])*A_[0,1]+(-epsln0*c
**2*star(DX(A_[0,1],1))*k*A_[0,1]+epsln0*c**2*DX(A_[0,1],1)*k*star(A_[0,1])+
epsln0*w*star(A_[0,1])*DT(A_[0,1],1)-epsln0*w*A_[0,1]*star(DT(A_[0,1],1)))*I
                  Investigating variable , A_[0, 1], at level, 3

                  Investigating variable , A_[1, 1], at level, 3
```

```
The equations of motion for the scalar variables are:-

The equation of motion for , A_[1,1],  is:-

                    2                      2  2
epsln0 w  A_[0, 1] - epsln0 c  k  A_[0, 1], =0

The equation of motion for , A_[0,1],  is:-

          2  2                    2
- epsln0 c  k  A_[1, 1] + epsln0 w  A_[1, 1]

                 2
      + (epsln0 c  DX(%1) k + epsln0 w DT(%1)) I + epsln0 w A_[0, 1] I

                                    2                          2
      + epsln0 w DT(%1) I + epsln0 c  k A_[0, 1] I + epsln0 c  k DX(%1) I, =0

%1 :=
                                  A_[0, 1], 1

                  op(newscalvar), A_[0, 1], A_[1, 1], A_[2, 1]

                                  op(newvecvar)

       op(newDTvar), DT(A_[0, 1], 1), DT(A_[0, 1], 2), DT(A_[1, 1], 1)

       op(newDXvar), DX(A_[0, 1], 1), DX(A_[0, 1], 2), DX(A_[1, 1], 1)

                                  op(newDXDTvar)

The lagrangian being used is :-
epsln0*DT(A_[0,1],1)*star(DT(A_[0,1],1))-epsln0*c**2*k**2*A_[1,1]*star(A_[1,1])
-epsln0*c**2*k**2*A_[2,1]*star(A_[0,1])-epsln0*c**2*k**2*star(A_[2,1])*A_[0,1]-
epsln0*c**2*DX(A_[0,1],1)*star(DX(A_[0,1],1))+epsln0*w**2*A_[2,1]*star(A_[0,1])
+epsln0*w**2*star(A_[2,1])*A_[0,1]+epsln0*w**2*A_[1,1]*star(A_[1,1])+(epsln0*c
**2*k*A_[1,1]*star(A_[0,1])+epsln0*c**2*k*star(A_[0,1])*DX(A_[0,1],2)+epsln0*DT
(A_[0,1],2)*w*star(A_[0,1])-epsln0*c**2*star(DX(A_[0,1],1))*k*A_[1,1]+epsln0*c
**2*DX(A_[0,1],1)*k*star(A_[1,1])-epsln0*c**2*star(DX(A_[1,1],1))*k*A_[0,1]-
epsln0*c**2*star(A_[1,1])*k*A_[0,1]-epsln0*c**2*star(DX(A_[0,1],2))*k*A_[0,1]+
epsln0*c**2*DX(A_[1,1],1)*k*star(A_[0,1])-epsln0*star(DT A_[0,1],2))*w*A_[0,1]-
epsln0*star(DT(A_[1,1],1))*w*A_[0,1]-epsln0*star(A_[1,1])*w*A_[0,1]+epsln0*w*
star(A_[1,1])*DT(A_[0,1],1)+epsln0*w*star(A_[0,1])*DT(A_[1,1],1)-epsln0*w*A_[1,
1]*star(DT(A_[0,1],1))+epsln0*w*A_[1,1]*star(A_[0,1]))*I
                  Investigating variable , A_[0, 1], at level, 4

                  Investigating variable , A_[1, 1], at level, 4

                  Investigating variable , A_[2, 1], at level, 4

               The equations of motion for the scalar variables are:-

                The equation of motion for , A_[2,1],  is:-

                      2                      2  2
               epsln0 w  A_[0, 1] - epsln0 c  k  A_[0, 1], =0
```

```
                The equation of motion for , A_[1,1],  is:-

          2  2                      2
- epsln0 c  k  A_[1, 1] + epsln0 w  A_[1, 1] + (

        2                     2
epsln0 c  DX(%1) k - epsln0 c  k A_[0, 1] - epsln0 w A_[0, 1] + epsln0 w DT(%1)

                                                                  2
) I + epsln0 w A_[0, 1] I + epsln0 w DT(%1) I + epsln0 c  k A_[0, 1] I

          2
 + epsln0 c  k DX(%1) I,

    =0

%1 :=
                                  A_[0, 1], 1

bytes used=141585324, alloc=3686400, time=1719.833

                The equation of motion for , A_[0,1],  is:-

          2  2                      2                      2
- epsln0 c  k  A_[2, 1] + epsln0 w  A_[2, 1] + (epsln0 c  k A_[1, 1]

             2                                          2
     + epsln0 c  k DX(%1) + epsln0 DT(%1) w + epsln0 c  DX(%2) k

     + epsln0 w DT(%2) + epsln0 w A_[1, 1]) I - epsln0 DT(A_[0, 1], 1, 1)

     + epsln0 w A_[1, 1] I + epsln0 w DT(%2) I + epsln0 w A_[0, 1] I

                                       2
     + epsln0 w DT(%1) I + epsln0 c  DX(A_[0, 1], 1)

             2                                  2
     + epsln0 c  DX(A_[0, 1], 1, 1) + epsln0 c  k A_[1, 1] I

             2                        2                      2
     + epsln0 c  k DX(%2) I + epsln0 c  k A_[0, 1] I + epsln0 c  k DX(%1) I, =0

%1 :=
                                  A_[0, 1], 2

%2 :=
                                  A_[1, 1], 1


> save `/scratch/xt/xtb003/qn2/eqnmotq.m`;
> quit;
bytes used=142043496, alloc=3686400, time=1728.350
```

Appendix 8

There follows a listing of the REDUCE program required to calculate results for the Small lagrangian. This is followed by a listing of the file containing the averaging procedure which begins on page 329.

```
COMMENT THIS IS A PROGRAM TO CALCULATE THE HIGHER ORDER RESULTS ;
COMMENT FOR THE SMALL LAGRANGIAN. ;
OPERATOR DEEX,DEET,SIGMA1,SIGMA2;
OPERATOR A1,A1STAR,A2,A2STAR,A3,A3STAR,A4,A4STAR;
OPERATOR B1,B1STAR,B2,B2STAR,B3,B3STAR,B4,B4STAR;
FACTOR EPSLN;
ORDER K,OMEGA,MUNOUGHT,ALPHA;
ORDER X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
ON REVPRI;
DEPEND XI,THETA,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND P,THETA,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND K,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND OMEGA,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND ASTAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND BSTAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A1,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A1STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B1,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B1STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A2,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A2STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B2,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B2STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A3,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A3STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B3,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B3STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A4,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND A4STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B4,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
DEPEND B4STAR,X1,T1,X2,T2,X3,T3,X4,T4,X5,T5;
COMMENT THIS SHOULD HAVE SET UP ALL THE REQUIRED DEPENDENCIES ;
COMMENT WE NOW GO ON WITH THE DERIVATIVE RULES ;
COMMENT !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! ;
FOR ALL F
    LET DEEX(F)=K*DF(F,THETA)+EPSLN*DF(F,X1)+EPSLN**2*DF(F,X2)
         +EPSLN**3*DF(F,X3)+EPSLN**4*DF(F,X4)+
         +EPSLN**5*DF(F,X5);
FOR ALL F
    LET DEET(F)=-OMEGA*DF(F,THETA)+EPSLN*DF(F,T1)+EPSLN**2*DF(F,T2)
         +EPSLN**3*DF(F,T3)+EPSLN**4*DF(F,T4)+
         +EPSLN**5*DF(F,T5);
%FOR ALL F,Q
%    LET DF(SIGMA1(F),Q)=SIGMA1(DF(F,Q));
%FOR ALL F,Q
%    LET DF(SIGMA2(F),Q)=SIGMA2(DF(F,Q));
COMMENT NOW SET HIGHER POWERS OF EPSLN THAN THREE TO ZERO BECAUSE ;
COMMENT WE WILL ONLY CONSIDER WORKING TO THIS ORDER TO START WITH ;
LET EPSLN**7=0;
ON LIST;
OFF NAT;
IN AVER2;
COMMENT MAKL8 CONTAINS THE CODE FOR THE AVERAGING PROCEDURE ;
COMMENT NOW GET ON WITH THE WORK ;
XISUBT!-2:=(DEET(XI))**2;
XISUBT:=DEET(XI);
XISUBX!-2:=(DEEX(XI))**2;
PSUBT!-2:=(DEET(P))**2;
L:=(R/2)*(XISUBT!-2-C**2*XISUBX!-2)-XISUBT*P+
              (S/2)*(PSUBT!-2-MUNOUGHT**2*P**2+(1/2)*ALPHA*P**4);
COMMENT IT MIGHT PROVE NECESSARY TO ALTER THIS FURTHER ;
COMMENT BY PUTTING IN THE ADDITIONAL CORRECTION TERMS ;
```

```
COMMENT SUCH AS A2(1),A3(1) ETC. WILL DO THIS IF NECESSARY ;
XI:=EPSLN*(A*E**(I*THETA)+ASTAR*E**(-I*THETA))+
    EPSLN**2*(A1(1)*E**(I*THETA)+
         A1STAR(1)*E**(-I*THETA))+
    EPSLN**3*(A2(1)*E**(I*THETA)+
         A2STAR(1)*E**(-I*THETA))+
    EPSLN**4*(A3(1)*E**(I*THETA)+
         A3STAR(1)*E**(-I*THETA))+
    EPSLN**5*(A4(1)*E**(I*THETA)+
         A4STAR(1)*E**(-I*THETA));
P:=EPSLN*(B*E**(I*THETA)+BSTAR*E**(-I*THETA))+
    EPSLN**2*(B1(1)*E**(I*THETA)+
         B1STAR(1)*E**(-I*THETA))+
    EPSLN**3*(B2(1)*E**(I*THETA)+
         B2STAR(1)*E**(-I*THETA))+
    EPSLN**4*(B3(1)*E**(I*THETA)+
         B3STAR(1)*E**(-I*THETA))+
    EPSLN**5*(B4(1)*E**(I*THETA)+
         B4STAR(1)*E**(-I*THETA));
COMMENT REMEMBER THAT THIS TIME THE EXPANSIONS FOR EACH ;
COMMENT PART STILL INCLUDE EPSLN AND SO MUST DO THE ;
COMMENT SEPARATION AT THIS STAGE ;
COMMENT ANOTHER POINT TO REMEMBER IS THAT WE MUST MULTIPLY;
COMMENT THE POWER SERIES EXPANSIONS FOR XI AND P BY EPSLN;
COMMENT IN ORDER TO MAKE SURE THERE IS SEPARATION OF ORDERS;
COMMENT IF WE DO NOT, WE MAKE THE PRODUCT OF THE TWO EXPANSIONS;
COMMENT OF THE SAME ORDER AS THE FIRST POWER OF EITHER. THIS IS;
COMMENT INCORRECT IN THAT THE INTERACTION TERM MUST ACT;
COMMENT NONLINEARLY, HENCE THE SCALING EMPLOYED HERE;
FIRST(COEFF(L,EPSLN));
SECOND(COEFF(L,EPSLN));
CLIST:=REST(REST(COEFF(L,EPSLN)));
COMMENT COEFF(L,EPSLN) HAS ITS FIRST TWO ELEMENTS EQUAL;
COMMENT TO ZERO, HENCE CLIST;
SL0:=FIRST(CLIST);
SL1:=SECOND(CLIST);
SL2:=THIRD(CLIST);
SL3:=THIRD(REST(CLIST));
SL4:=THIRD(REST(REST(CLIST)));
COMMENT SLI SHOULD BE THE ZERO TO SECOND ORDER RESULTS ;
COMMENT EPSLN**0 AND EPSLN**1 ARE BOTH ZERO;
COMMENT BE DARING ;
SL0BAR:=AVERAGE(SL0);
SL1BAR:=AVERAGE(SL1);
SL2BAR:=AVERAGE(SL2);
SL3BAR:=AVERAGE(SL3);
SL4BAR:=AVERAGE(SL4);
FACTOR R,S;
ON DIV;
COMMENT SAVE US DIVIDING THROUGHOUT BY TWO ETC. ;
SL0BAR;
SL1BAR;
SL2BAR;
SL3BAR;
SL4BAR;
OUT SMOUT5;
WRITE "OFF ECHO;";
SL0BARS:=SL0BAR;
SL1BARS:=SL1BAR;
SL2BARS:=SL2BAR;
SL3BARS:=SL3BAR;
SL4BARS:=SL4BAR;
WRITE ";END;"£
SHUT SMOUT5;
;END;
```

```
COMMENT WE ARE GOING TO NEED THE FOLLOWING SIMPLIFICATION RULES. ;
FOR ALL A,B LET LOG(A*B)=LOG(A)+LOG(B);
FOR ALL A,B LET LOG(A/B)=LOG(A)-LOG(B);
FOR ALL A,P LET LOG(A**P)=P*LOG(A);
COMMENT THIS SHOULD ALLOW FOR THE SIMPLIFICATION OF THE ARGUMENTS ;
COMMENT WITHIN THE SIGMA OPERATORS;
COMMENT THIS IS A NEW ATTEMPT AT THE PROBLEM. ;
COMMENT FIRST PROCEDURE TURNS POLY INTO A LIST OF ITS TERMS ;

ALGEBRAIC PROCEDURE MAKETERMS(POLY);
COMMENT INPUT IS A POLYNOMIAL WHOSE ARGLENGTH MAY BE ZERO ;
COMMENT MODIFIED FROM AVER1 ;
COMMENT SO THAT IT CAN HANDLE A NEGATIVE POLYNOMIAL;
COMMENT EG -(POLY)/DEN CAUSES A CRASH;
    BEGIN SCALAR NUMER,DENOM;
         SCALAR QLIST,COUNT,SSIGN;
         IF TERMS(POLY)=0 THEN REDERR "NO TERMS IN POLY";
         NUMER:=NUM(POLY); DENOM:=DEN(POLY);
         IF ARGLENGTH(NUMER)=0 THEN
              REDERR "MAKETERMS. ARGLENGTH IS ZERO";
         IF ARGLENGTH(NUMER)=1 THEN
              << IF PART(NUMER,1)=-NUMER THEN
                   <<NUMER:=-NUMER; SSIGN:=-1 >> >>
              ELSE <<SSIGN:=1 >> ;
COMMENT HAVE SSIGN=1 FOR POSITIVE POLYS ;
         IF ARGLENGTH(NUMER) NEQ TERMS(NUMER) THEN
              REDERR "MAKETERMS. SUM IS NOT TOP LEVEL OPERATOR";
         QLIST:= FOR I:= 1:ARGLENGTH(NUMER) COLLECT
              PART(NUMER,I)/(DENOM*SSIGN) ;
         IF LENGTH(QLIST) NEQ TERMS(NUMER) THEN
              << REDERR "MAKETERMS FAILS" >>;
         RETURN QLIST;
    END;

COMMENT THIS PROCEDURE IS DESIGNED TO HANDLE TERMS HAVING ;
COMMENT POWER SIGMA OF 1 AND A NUMERIC PART ;

ALGEBRAIC PROCEDURE P1SIG(NUMRIC,STERM);
    BEGIN SCALAR NUMER,DENOM,PWXP,SERMSG;
    SCALAR SARG,SLIST,SNUM,SDEN,ST1,ST2;
    SCALAR SGLIST,SOLLIST,SYPE,SLNL,SOLN;
    SCALAR FIR,SEC,ANS,SALIST;
    COMMENT BOTH ARGUMENTS MUST BE POSITIVE. SIGN IS HANDLED ;
    COMMENT BY SIGMATERM ;
    WRITE "DOING P1SIG";
    DENOM:=DEN(NUMRIC); NUMER:=NUM(NUMRIC);
    IF DEG(DENOM,(E**(I*THETA)))=0 AND DEG(NUMER,(E**(I*THETA)))=0
         THEN << PWXP:=0; WRITE "NO EXPONENTIAL IN P1SIG" >>
         ELSE << IF DEG(DENOM,(E**(I*THETA)))>DEG(NUMER,(E**(I*THETA)))
              THEN PWXP:=-DEG(DENOM,(E**(I*THETA)))
              ELSE PWXP:=DEG(NUMER,(E**(I*THETA))) >> ;
    IF PWXP=0 THEN GOTO LABEL1;
    COMMENT IF PWXP=0 THERE IS NO POINT IN KEEPING THE TERM ;
    COMMENT WE DO NOT CHANGE NUMRIC TO NUMRIC/E**(I*THETA) BECAUSE ;
    COMMENT WE ONLY USE THE FORM OBTAINED IN SALIST TO OBTAIN ;
    COMMENT THE SOLUTION AND SALIST DOES NOT CONTAIN THE EXP FACTOR ;
    COMMENT CORRECT FOR THE FACT THAT THE EXPONENTIAL PART IS TO ;
    COMMENT BE TAKEN INSIDE THE SIGMA OPERATOR ;
    IF ARGLENGTH(STERM) NEQ 1
         THEN REDERR "P1SIG. SIGMA NOT TOP LEVEL OPERATOR" ;
    OFF ALLFAC;
    COMMENT IF ALLFAC IS ON MUL FACTOR HAS HIGHEST PRECEDENCE IN SARG ;
    SARG:=PART(STERM,1);
    IF ARGLENGTH(SARG)=1 THEN
         << IF PART(SARG,1)=-SARG THEN
```

```
        << SARG:=-SARG; SERMSG:=-1 >> >>
ELSE SERMSG:=1 ;
WRITE "SARG IS ",SARG," WITH SERMSG ",SERMSG;
COMMENT MIGHT HAVE A UNARY MINUS ;
IF TERMS(SARG) NEQ 2 THEN REDERR "P1SIG. TERMS IN SIGMA'S ARG";
COMMENT GET THE ARGUMENT OF THE SIGMA OPERATOR AS SARG ;
SALIST:= MAKETERMS(SARG);
COMMENT USE MAKETERMS TO SPLIT UP THE SUMMATION WITHIN THE ARGUMENT ;
IF LENGTH(SALIST) NEQ 2 THEN
     REDERR "P1SIG. UNUSUAL ARGUMENT OF SIGMA";
WRITE "SALIST IS ",SALIST;
COMMENT THE SIGNS OF THE TERMS MAY BE NEGATIVE AND WE ;
COMMENT CANNOT TAKE THE LOG OF A NEGATIVE NUMBER. ;
COMMENT HENCE WE SHOULD TEST EACH OF THE TWO TERMS FOR ITS SIGN ;
COMMENT AND PUT THE SIGN IN A SEPARATE LIST CALLED SGLIST ;
SGLIST:={};
SLIST:= FOR EACH S IN SALIST COLLECT
     << IF ARGLENGTH(S)=1 THEN
          << IF PART(S,1)=-S THEN
               << SGLIST:=APPEND(SGLIST,{-1}); S:=-S >> >>
     ELSE << SGLIST:=APPEND(SGLIST,{1}); S:=S >> >> ;
WRITE "SLIST IS ",SLIST;
WRITE "SGLIST IS ",SGLIST ;
SLIST:= FOR EACH S IN SLIST COLLECT
     S*(E**(I*THETA*PWXP));
WRITE "NOW SLIST IS ",SLIST;
SLIST:= FOR EACH S IN SLIST COLLECT
     S:=LOG(S);
WRITE "SLIST IS NOW .... ",SLIST;
COMMENT SHOULD NOW HAVE NUMBERS MIXED WITH LOGS. ;
COMMENT THROW AWAY THE LOG TERMS AND KEEP THE OTHERS TO GIVE TO ;
COMMENT SOLVE IN ORDER TO THROW AWAY ONE HALF OF THE EXPRESSION ;
SOLLIST:= FOR EACH S IN SLIST COLLECT
     << MAKETERMS(S) >> ;
COMMENT NOW HAVE A LIST OF TWO LISTS ;
WRITE "SOLLIST SHOULD BE ",SOLLIST;
SOLLIST:= FOR EACH S IN SOLLIST COLLECT
     << S:= FOR EACH X IN S SUM
          << IF FREEOF(X,LOG) THEN
               << WRITE "KEEPING ",X; X:=X >>
                ELSE << WRITE "DISCARDING ",X; X:=0 >> >> >> ;
COMMENT SHOULD NOW HAVE TWO EXPRESSIONS WHICH WE WISH TO EQUATE ;
COMMENT TO ZERO TO SOLVE FOR A NON-NEGATIVE LINDEX OR LPINDEX ;
WRITE "SOLLIST IS ",SOLLIST;
FOR EACH S IN SOLLIST DO
     << IF FREEOF(S,LPINDEX)
          THEN SYPE:=LINDEX ELSE SYPE:=LPINDEX >> ;
WRITE "SYPE IS ",SYPE;
SLNL:= FOR EACH S IN SOLLIST JOIN
     << SOLVE(S=0,SYPE) >>;
WRITE "SLNL IS ",SLNL;
COMMENT WE WANT THE SOLUTION WHICH IS POSITIVE. ZERO NOT ALLOWED. ;
COMMENT SOLVE RETURNS A LIST OF SOLUTIONS ;
IF LENGTH(SLNL) NEQ 2 THEN REDERR "P1SIG. NUMBER OF SOLUTIONS" ;
IF NUMBERP(RHS(FIRST(SLNL))) THEN FIR:=RHS(FIRST(SLNL)) ELSE
     FIR:=LHS(FIRST(SLNL));
IF NUMBERP(RHS(SECOND(SLNL))) THEN SEC:=RHS(SECOND(SLNL)) ELSE
     SEC:=LHS(SECOND(SLNL));
COMMENT SOLVE RETURNS A LIST OF EQUATIONS - NOT NUMBERS ! ;
SOLN:=MAX(FIR,SEC); WRITE "SOLN IS ",SOLN;
COMMENT THIS SHOULD OBVIATE THE USE OF THE SUMMATION AND WE SHOULD ;
COMMENT NOW BE ABLE TO RETURN AN ANSWER. ;
IF SOLN=FIR THEN
     << ANS:=FIRST(SALIST)*SERMSG*NUMRIC >>
     ELSE << ANS:=SECOND(SALIST)*SERMSG*NUMRIC >> ;
```

```
    WRITE "ANS IS ... ",ANS;
    IF SOLN=FIR THEN ANS:=SUB(FIRST(SLNL),ANS)
         ELSE ANS:=SUB(SECOND(SLNL),ANS) ;
    WRITE "ANS IS ",ANS;
    GOTO LABEL2;
LABEL1: ANS:=0;
LABEL2: RETURN ANS;
    END;

COMMENT THIS PROCEDURE IS TO HANDLE TERMS CONTAINING A SIGMA;
COMMENT IT USES THE PROCEDURE P1SIG TO DO MOST OF THE WORK ;

ALGEBRAIC PROCEDURE SIGMATERM(TERM);
    BEGIN SCALAR NUMER,QLIST,NEWLIST,ANS;
    SCALAR DENOM,NN,NN1,NN2,TERMSG,SIGLIST;
    NUMER:=NUM(TERM); DENOM:=DEN(TERM);
    COMMENT DENOM MUST BE A NUMERIC TERM ;
    COMMENT NUMER WILL HAVE A NUMERIC PART, POSSIBLY 1, TIMES ;
    COMMENT POWERS OF SIGMAS ;
    IF ARGLENGTH(NUMER)<1 THEN REDERR "UNUSUAL NUMERATOR";
    IF ARGLENGTH(NUMER)=1 THEN
         << IF PART(NUMER,1)=-NUMER THEN
             << NUMER:=-NUMER; TERMSG:=-1 >> >>
    ELSE TERMSG:=1 ;
    WRITE "NUMER IS ",NUMER," WITH TERMSG ",TERMSG;
    IF LCOF(NUMER,SIGMA1)=1 OR LCOF(NUMER,SIGMA2)=1 OR
         LCOF(LCOF(NUMER,SIGMA2),SIGMA1)=1 THEN GOTO LABEL1;
    COMMENT ASSUME POWERS N,M OF SIGMA1**N*SIGMA2**M ARE NO HIGHER ;
    COMMENT THAN N=1, M=1, SO THAT CAN HAVE S1**1,S1**2,S2**1, ;
    COMMENT S2**2,S1*S2 AND NO OTHERS. ;
    QLIST:= FOR I:=1:ARGLENGTH(NUMER) COLLECT PART(NUMER,I);
    GOTO LABEL2;
LABEL1: QLIST:={NUMER}; WRITE "DONE GONE TO.";
    COMMENT SHOULD NOW HAVE A LIST OF FACTORS ;
    COMMENT IF LENGTH(QLIST)=1 THEN NUMER HAD A NUMERIC PART OF ;
    COMMENT 1 AND USING THE PART OPERATOR MIGHT HAVE SPLIT THE NEXT ;
    COMMENT LEVEL. E.G. SIGMA1**2 . ;
LABEL2: WRITE "LIST IS ",QLIST;
    NEWLIST:={}; NN:=0;
    IF LENGTH(QLIST) NEQ 1 THEN
         << NN1:=1;
         FOR EACH S IN QLIST DO
              << IF FREEOF(S,SIGMA1) AND
                   FREEOF(S,SIGMA2) THEN
                   << NN1:=NN1*S; WRITE "NN1 NOW IS ",NN1 >>
              ELSE << NEWLIST:=APPEND({S},NEWLIST) >> >> ;
         NEWLIST:=APPEND({NN1/DENOM},NEWLIST) >>
    ELSE << NEWLIST:=APPEND({1/DENOM},QLIST);
              WRITE "APPENDING DENOM",DENOM >> ;
    WRITE "NEWLIST IS ",NEWLIST;
    FOR EACH S IN NEWLIST DO
              << IF FREEOF(S,SIGMA1) AND FREEOF(S,SIGMA2)
               THEN NN:=NN+1 >> ;
    IF NN NEQ 1 THEN << WRITE "NN IS ",NN; REDERR "SIGMATERM FAILS" >> ;
    IF NOT FREEOF(FIRST(NEWLIST),SIGMA1)
         OR NOT FREEOF(FIRST(NEWLIST),SIGMA2) THEN
         REDERR "FIRST OF LIST NOT NUMERIC";
    COMMENT SHOULD NOW HAVE A LIST OF THE FOLLOWING FORM :-  ;
    COMMENT {NUMERIC,SIGMA1**N,SIGMA2**M} , IN THAT ;
    COMMENT ORDER. REMEMBER THAT THE OVERALL SIGN OF THE TERM ;
    COMMENT IS HELD IN TERMSG AND NEEDS TO BE MULTIPLIED IN LATER ;
    COMMENT NOW SEE WHAT THE OVERALL POWER OF THE SIGMA OPERATOR ;
    COMMENT IS BECAUSE IF IT IS HIGHER THAN ONE WE SHALL DO NO FURTHER ;
    COMMENT WORK ON THE TERM. ;
    SIGLIST:=REST(NEWLIST);
```

```
    IF LENGTH(SIGLIST) NEQ 1 THEN
       << WRITE "LENGTH SIGLIST IS ",LENGTH(SIGLIST); GOTO LABEL3 >> ;
    COMMENT THIS FOR MIXED SIGMAS ;
COMMENT    IF COEFFN(FIRST(SIGLIST),SIGMA1,1)=0 AND ;
COMMENT         COEFFN(FIRST(SIGLIST),SIGMA2,1)=0 THEN ;
    IF ARGLENGTH(FIRST(SIGLIST)) NEQ 1 THEN
        << WRITE "SIGMA POWER TOO HIGH"; GOTO LABEL3 >> ;
    COMMENT SHOULD NOW HAVE A SIGMA POWER OF 1 IN SIGLIST, ALSO ;
    COMMENT OF LENGTH 1. NOW NEED TO DO THE CLEVER BIT ! ;
    COMMENT REMEMBER THE SIGN OF THE TERM IN TERMSG ;
    ANS:=TERMSG*P1SIG(FIRST(NEWLIST),FIRST(SIGLIST));
    COMMENT TAKE THIS BIT OUT AND DO IT SEPARATELY ;
    GOTO LABEL4;
LABEL3: ANS:=TERM;
LABEL4: RETURN ANS;
    END;

ALGEBRAIC PROCEDURE AVERAGE(POLY);

    BEGIN SCALAR QLIST,HASDEN;
    SCALAR POLY2;
    QLIST:=MAKETERMS(POLY);
    IF LENGTH(QLIST)=0 THEN REDERR "EMPTY LIST";
        WRITE "LIST IS ... (AV) ", QLIST;
COMMENT THE COEFFN OPERATOR DOESN'T UNDERSTAND NEGATIVE POWERS ;
COMMENT NEXT BIT ASSUMES EACH S IS A POLY IN S ;
COMMENT NEED TO TEST FOR THE NEGATIVE POWERS OF EXP FACTOR FIRST ;
    POLY2:= FOR EACH S IN QLIST SUM
        << IF NOT FREEOF(S,SIGMA1) OR NOT FREEOF(S,SIGMA2)
            THEN << WRITE "RETAINING SIG TERM ",S; SIGMATERM(S) >>
            ELSE << IF COEFFN(DEN(S),E**(I*THETA),0)=0
                THEN << WRITE "DISCARDING TERM",S; 0 >>
                ELSE << IF COEFFN(S,E**(I*THETA),0)=0
                    THEN << WRITE "DISCARDING TERM",S; 0 >>
                    ELSE << WRITE "RETAINING TERM",S; S >> >>
     >>  >> ;
COMMENT THIS SHOULD NOW BE THE AVERAGED FORM WITH EXCEPTION OF ;
COMMENT THE SIGMA OPERATORS. DEAL WITH THESE LATER. ;
    WRITE "POLY2 IS ... (AV)", POLY2;
        RETURN  POLY2;
    END;
;END;
```

Appendix 9

There follows a listing of the MAPLE program required to calculate results for the Small lagrangian. All of the necessary files are given in the order in which they are presented in the main program file which is the first listing

The files and the order in which they appear is given below. This list is followed by an alphabetical list of the procedures by name.

The procedures are now listed alphabetically.

```
# ================================================================== #
#                                                                    #
# (C) Tom Arbuckle 1989, 1990, 1991                                  #
#                                                                    #
# ================================================================== #
#                                                                    #
# This program is the first in a suite of programs designed to deal  #
# with the process of obtaining the equations of motion from an      #
# averaged lagrangian. This first program accepts as input the file  #
# AVERIN DAT A (AVERIN.DAT in MAPLE) and then performs validation    #
# on the contents of this file. It then goes on to perform suitable  #
# expansions to a level specified to the user, finally passing the   #
# results on to the next progam as a data file.                      #
# The file AVERIN.DAT should contain the following:                  #
#    1) the variable errorswitch (set to true or false);             #
#    2) the variable prinlvl (set to an integer 0<=prinlvl<=4);      #
#    3) the lagrangian,L;                                            #
#    4) the required order of expansion, jmax;                       #
#    5) a set, varset, containing the variables used in L;           #
#    6) an optional set, vset, containing all array type quantities  #
#          in use within the Lagrangian;                             #
#    7) a variable, hiharm, determining whether or not the expansion #
#          used will include harmonics higher than the fundamental.  #
#          This variable should either be set to true or false.      #
#    8) a variable, groundstate, determining whether or not the      #
#          groundstate of a vector quantity will be treated as a     #
#          perturbation from unity. Normally set to true.            #
# If either of the first two variables are not assigned, they are    #
# given the default values of false and 0.                           #
# The variable hiharm will be given the value false if not assigned. #
# The variable groundstate has a default of true if not assigned.    #
#                                                                    #
# N.B.  The variable ok must be assigned false whenever an error     #
# occurs. A top level exit is required in order that the output      #
# from the help file be displayed. Hence checking to see whether     #
# things are ok will correctly terminate the program on error.       #
#                                                                    #
# ================================================================== #
#
printlevel:=-1;                  # adjust level of output            #
prettyprint:=2:                  # left justify printed output       #
words(0);                        # switch off words used messages    #
read `/user10/xt/xtb003/mplfiles/averinct.dat`;    # read data file   #
ok:=true;
if assigned(errorswitch) then
   if type(errorswitch, boolean) then
       if errorswitch=false then
           print(`Error reporting not required by user`),
       elif errorswitch=true then
           print(`Extended error reporting switched on.`);
           print(`Detailed error messages will be given by the program`);
       else
           print(`The variable errorswitch must be assigned true or false`);
           ok:=false;
       fi;
    else
        print(`The variable errorswitch must be assigned true or false`);
        ok:=false;
    fi;
fi;
if ok=false then stop fi:
# Now load in callerr as the error handling procedure                #
# callerr defined as an unevaluated function call                    #
callerr:='readlib('callerr',`/user10/xt/xtb003/mplfiles/callerr.m`)';
errhndlr:='readlib('errhndlr',`/user10/xt/xtb003/mplfiles/errhndlr.m`)';
```

```
errtable:='readlib('errtable',`/user10/xt/xtb003/mplfiles/errtable.m`)';
if assigned(vset) then
   if type(vset,set) then
      print(`WARNING: Vector or array quantities implied in input file.`);
   else
      ok:=false;
      callerr(`Variable vset must be a set`,5)
   fi;
else
   vset:={}:
fi;
if not ok then stop fi;
if assigned(hiharm) then
   if type(hiharm,boolean) then
      if hiharm=true then
         print(`WARNING: Higher harmonics used in this calculation.`)
      elif hiharm=false then
         print(`Expansion using lower harmonics only.`)
      else
         ok:=false
      fi;
   else
      ok:=false:
      callerr(`The variable hiharm must be either true or false.`,8);
   fi;
else
   hiharm:=false:
fi;
if not ok then stop fi;
if assigned(groundstate) then
   if type(groundstate,boolean) then
      if groundstate=true then
         print(`The groundstate of vector variables will be taken.`);
         print(`to be a perturbation from unity.`);
      elif groundstate=false then
         print(`WARNING: The groundstate of any vector variables will`);
         print(`         not be taken to be a special case.`);
      else
         ok:=false
      fi;
   else
      ok:=false:
      callerr(`The variable groundstate must be either true or false.`,9);
   fi;
else
   groundstate:=true:
fi;
if not ok then stop fi:
kset:={anames()} minus
    {'prettyprint','L','varset','JMAX','errorswitch','prinlvl','hiharm',
     'ok','vset',`errhndlr`,`callerr`,`errtable`,'groundstate'}
     minus vset minus
    {`_initvalsNestlist`,`table/initvals`,`print/matrix`};
if kset<>{} then
   ok:=false:
   callerr(`You have assigned superfluous variables`,kset,1)
fi;
if not ok then stop fi:
kset:='kset':
if assigned(prinlvl) then
   if type(prinlvl,integer) then
      if prinlvl>=0 and prinlvl<=4 then
         print(`The level of printed output has been set to`,prinlvl);
      else
         ok:=false:
```

```
         callerr('Valid print levels lie between 0 and 4',2);
      fi;
   else
      ok:=false:
      callerr('Prinlvl must be set to an integer',3);
   fi;
else
   prinlvl:=0;          # Default printlevel set to 0                #
fi;
if not ok then stop fi;
# Check to see that L contains the variables given in varset         #
for var in varset while ok=true do
   if not has(L,var) then ok:=false fi
od;
if not ok then
   callerr('Lagrangian, L, does not contain the variable',var,4);
fi;
if not ok then stop fi:
if vset<>{} then
   kset:=varset intersect vset:
   if kset={} then
      ok:=false:
      callerr('Improper use of vector quantities',6)
   else
      print('The following variables have been defined as vectors: ');
      print(kset)
   fi;
else
   kset:={};
fi;
if not ok then stop fi:
for var in (vset minus kset) do;
   if not type(var,matrix) then
     ok:=false;
     callerr('Incorrect matrix/array type definitions in input file',7);
   fi;
od;
if not ok then stop fi:
# nset is the set of non-vector expansion variables                  #
nset:=varset minus kset:
# read 'datestam.mpl';  # Valid only for CMS machines. #
# datestamp();          # Valid only for CMS machines. #
print('The input file has now been validated.');
if prinlvl>2 then
   print('The program will now go on to perform the expansion');
   print('required as a preliminary to the averaging procedure.');
fi;
# nset=varset for all scalars. #
if prinlvl>1 then
   print('The lagrangian used will be:- ');
   print(L);
   if hiharm=true then
      print('Higher harmonics will be used in the substitution.')
   else
      print('Only the fundamental will be used in the calculation.')
   fi;
   if nops(vset)<>0 then
      print('The calculation assumes that the following quantities are');
      print('vector variables which will require subsequent expansion');
      print(kset);
      print(' and that these variables are the scalar variables');
      print(' be expanded');
      print(nset);
      print('The following quantities are arrays which are');
      print('assumed to have values given by the user.');
```

```
      print(vset minus kset);
      if groundstate=true then
         print(`The groundstate of vector quantities will be`);
         print(`taken to be near unity.`);
      else
         print(`Note that the groundstate of vector quantities will`);
         print(`NOT be taken to be a special case and will be expanded`);
         print(`in a similar manner to all of the other variables.`):
      fi;
      print();
   else
      print(`These variables will be taken to be those with which the`);
      print(`expansion is to be performed  and with respect to which`);
      print(`the subsequent variation is to be carried out.`);
      print(nset); print();
   fi;
fi;
if prinlvl>2 then
   print(`Calculation begins ...`);
fi;
# Convert sets to lists to maintain ordering.                          #
nlist:=convert(nset,list); klist:=convert(kset,list);
tb:=table(): tc:=table():
for i from 1 to nops(nlist) do;
   assign(tb[i]=cat(nlist[i],_));
od;
for i from 1 to nops(klist) do;
   assign(tc[i]=cat(klist[i],_));
od;
newnlist:=convert(tb,list):
newklist:=convert(tc,list):
tb:='tb': tc:='tc':   # Unassign tb, tc                              #
read `/user10/xt/xtb003/mplfiles/typedef.mpl`;
read `/user10/xt/xtb003/mplfiles/simpstar.mpl`;
read `/user10/xt/xtb003/mplfiles/simpdag.mpl`;
read `/user10/xt/xtb003/mplfiles/expddag.mpl`;
read `/user10/xt/xtb003/mplfiles/expdstar.mpl`;
read `/user10/xt/xtb003/mplfiles/expddson.mpl`;
read `/user10/xt/xtb003/mplfiles/diffdtdx.mpl`;
read `/user10/xt/xtb003/mplfiles/simpdtdx.mpl`;
read `/user10/xt/xtb003/mplfiles/orderser.mpl`;
read `/user10/xt/xtb003/mplfiles/sums.mpl`;
read `/user10/xt/xtb003/mplfiles/average.mpl`;
read `/user10/xt/xtb003/mplfiles/smrtexpd.mpl`;
readlib(evalm):
# Simplify the lagrangian to get rid of star(star(. etc.
L:=simplify(L,dagger);
L:=simplify(L,star);
if prinlvl>=3 then
   print(`The lagrangian has now been simplifed with respect to dagger`);
   print(`and star.`);
fi;
if prinlvl=4 then print(`The lagrangian is now`,L) fi;
if klist<>[] then
   for i from 1 to nops(klist) do;
      assign(op(i,newklist)=array( op(2,op(op(i,klist)))) );
   od;
   if hiharm=false then
      for i from 1 to nops(klist) do;
         assign(op(i,klist),quantsum(op(i,newklist)));
      od;
   else
      for i from 1 to nops(klist) do;
         assign(op(i,klist),bigquantsum(op(i,newklist)));
      od;
```

```
   fi;
fi;
if hiharm=false then
   for i from 1 to nops(nlist) do;
      assign(op(i,nlist),weeharmsum(op(i,newnlist)));
   od;
else
   for i from 1 to nops(nlist) do;
      assign(op(i,nlist),bigharmsum(op(i,newnlist)));
   od;
fi;
# read in the Diff procedure now that the substitutions have been made.
read `/user10/xt/xtb003/mplfiles/diftheta.mpl`;
L:=simplify(L,DT,DX);
# Should now have made the necessary assignments. Matrices should not be
# empty so can carry out necessary calculations.
if prinlvl>=3 then
   print(`The arrays have now been filled and the summations`);
   print(`required for the expansion carried out.`);
   if prinlvl=4 then
      print(`The lagrangian has become: `,L);
   fi;
fi;
L:=expddagstonly(L);
if prinlvl>=3 then
   print(`The lagrangian has been expanded with repsect to its`);
   print(`dagger and star operations only.`);
   if prinlvl=4 then
      print(`The lagrangian becomes: `);
      lprint(L);
   fi;
fi;
# If simply do expand then will fail when dagger is mapped onto
# matrices. Transpose will alter shape remember.
L:=simplify(L,DT,DX,star); # simplify wrt DT,DX,star only.
save `/scratch/xt/xtb003/lagrngq4.m`;
# now able to expand using special procedure to avoid 'Object too big'.
L:=smartexpand(L):
# Truncate the expanded lagrangian.
L:=seriestrunc(L,[eps=1],JMAX);
if prinlvl=4 then
   print(`The lagrangian has been expanded and simplified.`);
   print(`Its final form is: `);
   lprint(L);
fi;
# Collect terms in `eps`                    #
L:=collect(L,eps):
if prinlvl>=3 then
   print(`The lagrangian has been rearranged in terms of the small`);
   print(`variable eps.`);
   if prinlvl=4 then
      print(`The lagrangian is now: `);
      lprint(L);
   fi;
fi;
save `/scratch/xt/xtb003/avlagq4t.m`;
for i from 0 to JMAX do; #4 do;
   L.i:=coeff(L,`eps`,i);
   L.i.`bar`:=average(L.i);
   LC.i.`bar`:=evalc(L.i.`bar`);
   print(`The averaged lagrangian at order`,i,` is:`);
   print(LC.i.`bar`);
od;
save `/scratch/xt/xtb003/tavlag4q.m`;
# Read in files required for determining equations of motion.
```

```
read `/user10/xt/xtb003/mplfiles/ordindex.mpl`;
read `/user10/xt/xtb003/mplfiles/dordbool.mpl`;
read `/user10/xt/xtb003/mplfiles/mapmod.mpl`;
read `/user10/xt/xtb003/mplfiles/eqnmot.mpl`;
read `/user10/xt/xtb003/mplfiles/derivord.mpl`;
read `/user10/xt/xtb003/mplfiles/makeELterm.mpl`;
read `/user10/xt/xtb003/mplfiles/doprint2.mpl`;
for levcount from 0 to JMAX do;
   equationsofmotion(`LC`.levcount.`bar`);
# Create DTvar, DXvar, DXDTvar as sets.
   DTvar:=convert(DTvar,`list`);
   DXvar:=convert(DXvar,`list`);
   DXDTvar:=convert(DXDTvar,`list`);
   scalvar:=convert(scalvar,`list`);
   vecvar:=convert(vecvar,`list`);
   newDTvar:=sort(DTvar,dordbool);
   newDXvar:=sort(DXvar,dordbool);
   newDXDTvar:=sort(DXDTvar,dordbool);
   newscalvar:=sort(scalvar,fordbool);
   newvecvar:=sort(vecvar,fordbool);
   if prinlvl>3 then
      print(`op(newscalvar)`,op(newscalvar));
      print(`op(newvecvar)`,op(newvecvar) );
      print(`op(newDTvar)`,op(newDTvar));
      print(`op(newDXvar)`,op(newDXvar));
      print(`op(newDXDTvar)`,op(newDXDTvar));
   fi;
   lprint(`The lagrangian being used is :-`);
   lprint(`LC`.levcount.`bar`);
# Generating the equations starts here. The names of the arrays which
# will hold the equations of motion will go in the following two sets.
   scalareqnset:={}; vectoreqnset:={};
   if nops(newscalvar)<>0 then
      for var in newscalvar do;
# Generate Euler-Lagrange for scalar variables.
      print(`Investigating variable `,var,`at level`,levcount);
      scalname:=op(0,var);
      if not assigned(`eqtn`.levcount.scalname) then
         scalareqnset:=scalareqnset union {`eqtn`.levcount.scalname};
         `eqtn`.levcount.scalname:=table(sparse,[]) fi;
      `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
          makeELterm(var,`LC`.levcount.`bar`);
      if nops(newDTvar)<>0 then
         for var2 in newDTvar do;
            if has(var2,var) then
               `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
                  makeELterm(var2,`LC`.levcount.`bar`);
            fi;
         od;
      fi;
      if nops(newDXvar)<>0 then
         for var2 in newDXvar do;
            if has(var2,var) then
               `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
                    `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
                  makeELterm(var2,`LC`.levcount.`bar`);
            fi;
         od;
      fi;
      if nops(newDXDTvar)<>0 then
         for var2 in newDXDTvar do;
            if has(var2,var) then
               `eqtn`.levcount.scalname[op(1,var),op(2,var)]:=
                  `eqtn`.levcount.scalname[op(1,var),op(2,var)]+
```

```
                    makeELterm(var2,`LC`.levcount.`bar`);
                fi;
             od;
          fi;
          od;
       fi;
       if nops(newvecvar)<>0 then
          for var in newvecvar do;
    # Generate Euler-Lagrange for vector variables.
          print(`Investigating vector variable`,var,`at level`,levcount);
          vecname:=op(0,var);
          if not assigned(`eqtn`.levcount.vecname) then
             vectoreqnset:=vectoreqnset union {`eqtn`.levcount.vecname};
             `eqtn`.levcount.vecname:=table(sparse,[]) fi;
          `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
              makeELterm(var,`LC`.levcount.`bar`);
          if nops(newDTvar)<>0 then
             for var2 in newDTvar do;
                if has(var2,var) then
                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
                      `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
                      makeELterm(var2,`LC`.levcount.`bar`);
                fi;
             od;
          fi;
          if nops(newDXvar)<>0 then
             for var2 in newDXvar do;
                if has(var2,var) then
                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
                      `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
                      makeELterm(var2,`LC`.levcount.`bar`);
                fi;
             od;
          fi;
          if nops(newDXDTvar)<>0 then
             for var2 in newDXDTvar do;
                if has(var2,var) then
                   `eqtn`.levcount.vecname[op(1,var),op(2,var)]:=
                      `eqtn`.levcount.vecname[op(1,var),op(2,var)]+
                      makeELterm(var2,`LC`.levcount.`bar`);
                fi;
             od;
          fi;
          od;
       fi;
       doprinteqns();
    od; # end of outer loop.
    save `/scratch/xt/xtb003/eqnmotq.m`;
    quit;
    # Should be the end of the matter.
```

```
callerr:=proc()
   if nargs=0 then
      ERROR(`No arguments supplied to callerr`)
   elif nargs=1 then
      ERROR(`Require more than one argument to callerr`)
   else
      print(args[1..nargs-1]);
      if errorswitch=true then
          errhndlr(args[nargs])
      fi;
   fi;
end;
save `/user10/xt/xtb003/mplfiles/callerr.m`;
```

```
# ================================================================== #
#                                                                    #
# Used as an error handler in APROG1Vi. Modified from ERH2 MPL       #
# Argument is an integer which must belong to the errortable.        #
# The integer will be used to look up the appropriate error message. #
#                                                                    #
# ================================================================== #
errhndlr:=proc(ern)
          local temp,plevel;
#         print(ern,errorswitch);
          if not type(ern,integer) then
       ERROR(`Last part of expression sequence must be an integer`)
          fi;
# errtable will have been defined by an implicit read in the main    #
# program. Now need to check that the error is known to the system.  #
          if not assigned(errtable[ern]) then
              ERROR(`This error not known to the system`)
          fi;
          temp:=cat(`mplfiles/H`,ern,`.HLP`):
#         plevel:=printlevel;  printlevel:=111;                      #
          read temp;
#         printlevel:=plevel;                                        #
          end;
save `/user10/xt/xtb003/mplfiles/errhndlr.m`;
```

```
# ================================================================== #
# Type definitions. Load Separately. Used in simplify and in         #
# expand.                                                            #
# ================================================================== #
`type/exp`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`exp` then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
`type/star`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`star` then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
`type/dagger`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`dagger` then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
`type/&*`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`&*` then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
`type/DX`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`DX` and nops(f)>=2 then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
`type/DT`:=proc(f);
   if not type(f,function) then RETURN(false) fi;
   if op(0,f)=`DT` and nops(f)>=2 then
      RETURN(true)
   else
      RETURN(false)
   fi;
end;
```

```
# ================================================================ #
#                                                                  #
# Procedure to handle the simplification of the star (complex conj.) #
# operator. The argument may be of many different kinds.           #
# Note that the file `typdef.mpl` must have been read in first.    #
# In addition, the file `simpdag.mpl` should have been read in:    #
# both files cross reference each other.                           #
#                                                                  #
# ================================================================ #
`simplify/star`:=proc(f)
   local temp,temp2,i;
   if nargs<>1 then
      ERROR(`More than one argument supplied.`);
   fi;
   if not type(f,`star`) then
      if type(f,`+`) then RETURN( map(`simplify/star`,f))
      elif type(f,`*`) then
         temp2:=1;
         for i to nops(f) do;
            temp2:=temp2*(`simplify/star`(op(i,f)))
         od;
         RETURN(temp2);
      elif type(f,`**`) then
         temp2:=op(1,f);
         if type(temp2,`star`) then
            temp2:=`simplify/star`(temp2);
            RETURN(subsop(1=temp2,f));
         else RETURN(f)
         fi;
      else
         RETURN(f)
      fi;
   elif type(f,`star`) then
      temp:=op(1,f);
      if type(temp,`+`) then
         RETURN( map(`simplify/star`, map(`star`,temp) ) );
      elif type(temp,`*`) then RETURN(starprodsimp(temp))
      elif type(temp,`**`) then RETURN(starpowersimp(temp))
      elif type(temp,`exp`) then RETURN(starexpsimp(temp));
      elif type(temp,`star`) then RETURN(op(1,temp));
      elif type(temp,`dagger`) then
        temp2:=simplify(dagger(star(op(1,temp))) );
         RETURN(temp2);
      elif temp=I then RETURN(-I);
      elif type(temp,`numeric`) then RETURN(temp);
      elif type(temp,`string`) and temp=`eps`
         then RETURN(temp); # names are vars #
      elif temp=0 then RETURN(0);
      else RETURN(f)
      fi;
   else RETURN(f)
   fi;
end;

starpowersimp:=proc(f)
   local powtemp;
   if nargs<>1 then
     ERROR(`Too many arguments.`);
   fi;
   powtemp:=op(1,f):
   if type(powtemp,`star`) then RETURN(subsop(1=op(1,powtemp),f))
   elif type(powtemp,`&*`) then RETURN(star(f))
   elif type(powtemp,`array`) then RETURN(subsop(1=star(op(1,f)),f));
   elif type(powtemp,`function`) then RETURN(subsop(1=star(op(1,f)),f));
   else RETURN(f); # Want star( a**3 ) to be a**3 #
```

```
   fi;
end;

starprodsimp:=proc(f)
   local i,stem;
   stem:=1:
   for i to nops(f) do;
      stem:=stem*(`simplify/star`(star(op(i,f))));
   od;
   RETURN(stem);
end;

starexpsimp:=proc(f)
   local reel,imeg,temp;
   reel:=evalc(Re(op(1,f)));
   imeg:=evalc(Im(op(1,f)));
   temp:=exp(reel-I*imeg):
   RETURN(temp);
end;

stararrsimp:=proc(f);
   RETURN( map(star, op(f)) );
end;

starncprodsimp:=proc(f)
   if nops(f)<>2 then RETURN(f) fi;
   RETURN( evalm(`&*`( `simplify/star`( star(op(2,f)) ),
         `simplify/star`( star(op(1,f)) ) )) );
end;
```

```
# ================================================================ #
#                                                                  #
# Procedure to handle the simplification of the dagger (Herm. conj.) #
# operator. The argument may be of many different kinds.           #
# Note that the file 'typedef.mpl' must be read in first and that  #
# the file containing 'simplify/star' must also have been read in. #
#                                                                  #
# ================================================================ #
'simplify/dagger':=proc(f)
   local temp,temp2,i;
   if nargs<>1 then
      ERROR('More than one argument supplied.');
   fi;
   if not type(f,'dagger') then
      if type(f,'+') then RETURN( map('simplify/dagger',f))
      elif type(f,'*') then
         temp2:=1;
         for i to nops(f) do;
            temp2:=temp2*('simplify/dagger'(op(i,f)))
         od;
         RETURN(temp2);
      elif type(f,'**') then
         temp2:=op(1,f);
         if type(temp2,'dagger') then
            temp2:='simplify/dagger'(temp2);
            RETURN(subsop(1=temp2,f));
         else RETURN(f)
         fi;
      elif type(f,'&*') then
         temp2:='simplify/dagger'(op(1,f));
         temp2:=&*(temp2,('simplify/dagger'(op(2,f))) );
         RETURN(temp2);
      else
         RETURN(f)
      fi;
   elif type(f,'dagger') then
      temp:=op(1,f);
      if type(temp,'+') then
         RETURN( map('simplify/dagger', map('dagger',temp) ) );
      elif type(temp,'*') then RETURN(daggerprodsimp(temp))
      elif type(temp,'**') then RETURN(daggerpowsimp(temp))
      elif type(temp,'exp') then RETURN(daggerexpsimp(temp));
      elif type(temp,'star') then RETURN(dagstarsimp(temp));
      elif type(temp,'dagger') then RETURN(op(1,temp));
      elif type(temp,'numeric') then RETURN(temp);
      elif type(temp,'&*') then RETURN(f);
      elif type(temp,'array') then RETURN(f); # RETURN(star(ttrans(temp)));
      elif temp=I then RETURN(-I);
      elif temp=0 then RETURN(NULL);
      else RETURN(star(temp));
      fi;
   else RETURN(f)
   fi;
end;
daggerpowsimp:=proc(f)
   if nargs<>1 then
     ERROR('Too many arguments.');
   fi;
   if type(op(1,f),array) then RETURN(dagger(f)) fi;
   RETURN('simplify/star'(subsop(1=star(op(1,f)),f)));
end;
daggerprodsimp:=proc(f)
   local i,stem;
   stem:=1:
   for i to nops(f) do;
```

```
      stem:=stem*(`simplify/dagger`(dagger(op(i,f))));
   od;
   RETURN(stem);
end;
daggerexpsimp:=proc(f)
   RETURN(starexpsimp(f)); # starexpsimp expects star stripped       #
end;
dagstarsimp:=proc(f)
   local temp,temp2,i,stem,bse;
   if nops(f)<>1 then ERROR(`Too many arguments`) fi;
   if not type(f,star) then  ERROR(`Not Star`) fi;
   temp:=f; `simplify/star`(temp);
   if type(temp,`+`) then RETURN(map(daggstarsimp,map(dagger,temp)));
   elif type(temp,`*`) then
      stem:=1;
      for i to nops(temp) do
         stem:=stem*daggstarsimp(dagger(op(i,temp)));
      od;
      RETURN(stem);
   elif type(temp,`star`) then
      temp2:=op(1,temp);   #star of something - strip off the star   #
      if type(temp2,`dagger`) then RETURN(star(op(1,temp2))); #dg(st(dg#
      elif type(temp2,`array`) then RETURN(dagger(temp));
      elif type(temp2,`&*`) then RETURN(dagger(temp));
      else RETURN(temp2);  # dagger(star())->();                       #
      fi;
   elif type(temp,`**`) then
      bse:=op(1,temp);
      if type(bse,`star`) then RETURN(subsop(1=op(1,bse),temp))
      else RETURN(subsop(1=star(bse),temp))
      fi;
   else RETURN(`simplify/dagger`(dagger(temp)));
   fi;
end;
```

```
# ================================================================== #
#                                                                    #
# Routine to expand the dagger operator. For arguments other than    #
# numeric, array, non-commutative product of arrays, the routine     #
# simply returns the star of its argument. For ncprods, it expands.  #
# For arrays, it maps the star operator onto the transpose of the    #
# to find the Hermitian conjugate.                                   #
# Note that the arguments hould have been simplified first and that  #
# the files 'expdstar.mpl' and typedef.mpl' are required by this     #
# section of the code.                                               #
#                                                                    #
# ================================================================== #
`expand/dagger`:=proc(f)
   local i,stem;
   if nargs<>1 then
      ERROR(`Too many arguments.`)
   fi;
   if type(f,`+`) then RETURN(map(`expand/dagger`,f))
   elif type(f,`*`) then
      stem:=1;
      for i from 1 to nops(f) do;
         stem:=stem*`expand/dagger`(op(i,f))
      od;
      RETURN(stem);
   fi;
   if type(f,`&*`) then
   readlib(evalm):
      RETURN( evalm(`&*`( `expand/dagger`( op(2,f) ),
         `expand/dagger`( op(1,f) ) ) ));
   elif type(f,array) then print(`dagarr`); RETURN( dagarrexpd(f))
   elif type(f,`star`) then
      RETURN(`expand/dagger`(`expand/star`(op(1,f))));
   elif type(f, numeric) then RETURN(f)
   else RETURN( star(f))
   fi;
end;
# ================================================================== #
#                                                                    #
# Routine to find Hermitian conjugate of an array.                   #
# Takes one array  argument.                                         #
#                                                                    #
# ================================================================== #
dagarrexpd:=proc(f)
   local tempm, tempm2, tempm3;
#   tempm:=op(f);
   if nargs<>1 then
      ERROR(`Too many arguments to dagger function`)
   elif not type(f,array) then
      ERROR(`Argument must be an array`)
   fi;
   tempm:=ttrans(f); # linalg[ transpose ](f);
   tempm2:=map(star, tempm);
   tempm3:=simplify(tempm2,star);
   RETURN( op(tempm3));
end;
ttrans:=proc(f)
   local ic, mmax, temp; # Added 12/6/92.
   if nargs<>1 or not type(f,`array`) then
      ERROR(`Arguments to ttrans.`);
   fi;
   mmax:=op(2,(op(2,(op(f)))));
   temp:=array(1..1,1..mmax);
   for ic from 1 to mmax do;
      temp[1,ic]:=f[ic];
```

```
   od;
#   print(temp);
   RETURN(op(temp));
end;
```

```
# ================================================================ #
#                                                                  #
# Procedure to handle the expansion of the star (complex conj.)    #
# operator. The argument may be of many different kinds.           #
# Note that in the case of expand the star operator is stripped    #
# whereas in the case of simplify it is not.                       #
# The arguments to the expand function should have been simplified #
# first and the files `simpdag.mpl` and `typedef.mpl` should have  #
# been read in before running this section of the code.            #
#                                                                  #
# ================================================================ #
`expand/star`:=proc(f)
   local temp;
   if nargs<>1 then
      ERROR(`More than one argument supplied.`);
   fi;
   if type(f,`array`) then RETURN(stararrexpd(f));
   elif type(f,`&*`) then RETURN(starncprodexpd(f));
   elif type(f,`star`) then
     RETURN(`expand/dagger`(`expand/star`(op(1,f))));
   elif type(f,numeric) then RETURN(f);
   else RETURN(star(f));
   fi;
end;
stararrexpd:=proc(f);
   RETURN( map(star, op(f)) );
end;
starncprodexpd:=proc(f)
   if nops(f)<>2 then RETURN(f) fi;
   readlib(evalm):
   RETURN( evalm(`&*`( `expand/star`( op(2,f) ),
         `expand/star`( op(1,f) ) )) );
end;
```

```
# ================================================================== #
#                                                                    #
# Expand with respect to dagger and star only.                       #
#                                                                    #
# ================================================================== #
expddagstonly:=proc(f)
   local i,temp,stem,mularr1,mularr2,mularr3,mularr4,narr1;
   option remember;
   if nargs>1 then ERROR(`Too many arguments`) fi;
   if type(f,`+`) then RETURN(map(expddagstonly,f))
   elif type(f,`*`) then
      stem:=1;
      for i from 1 to nops(f) do;
         stem:=stem*expddagstonly(op(i,f));
      od;
      RETURN(stem);
   elif type(f,`dagger`) then
      RETURN(`expand/dagger`(op(1,f)))
   elif type(f,`star`) then
      RETURN(`expand/star`(op(1,f)))
   elif type(f,`DT`) then
      RETURN(`simplify/DT`(f))
   elif type(f,`DX`) then
      RETURN(`simplify/DX`(f))
   elif type(f,`&*`) then
      readlib(evalm);
      temp:=f;
      if has(temp,{`star`,`dagger`,`&*`}) then
         mularr1:=expddagstonly(op(1,temp));
         mularr2:=expddagstonly(op(2,temp));
         if not type(mularr1,`matrix`) then
            mularr3:=evalm(mularr1)
         else
            mularr3:=mularr1
         fi;
         if not type(mularr2,`array`) then
            mularr4:=evalm(mularr2)
         else
            mularr4:=mularr2
         fi;
         narr1:=linalg[multiply](mularr3,mularr4);
         if type(narr1,`vector`) then
            if nops(op(2,op(narr1)))=2 and
               op(2,op(2,op(narr1)))=1 then
                  RETURN(narr1[1])
            else
               RETURN(narr1)
            fi;
         else
            RETURN(narr1)
         fi;
      else RETURN(f)
      fi;
   else RETURN(f)
   fi;
end;
```

```
# ================================================================== #
#                                                                    #
# These procedures defind the differentiation operations dx, dt in   #
# terms of other procedures namely: Dx,Dt,DEEX,DEET,DX,DT. DX, DT    #
# must be left undefined so that the remain unevaluated. These       #
# procedures assume that the varaible varset contains a set  of      #
# variables with respect to which differentiation is sensible and    #
# require JMAX to be defined as the level of expansion.              #
#   The types DT and DX are defined in typedef.mpl and the procedures#
# simplify/DT and simplify/DX are held in simpdtdx.mpl to deal with  #
# the commutation of these operators with star and dagger.           #
#                                                                    #
# ================================================================== #

dt := proc(f) -w*Diff(f,theta) + Dt(f) end;
dx := proc(f) +k*Diff(f,theta) + Dx(f) end;

Dt := proc(f)
  local j;
  sum('eps**j*DT(f,j)','j'=1..JMAX)
  end;

Dx := proc(f)
  local j;
  sum('eps**j*DX(f,j)','j'=1..JMAX)
  end;

`diff/DT`:=proc()
   local temp,y,f,i;
   f:=args[1]; y:=args[nargs];
   if nargs>=3 then
      for i from 2 to nargs-1 do;
         if not type(args[i],numeric) then RETURN(DT(args)) fi;
      od;
   else ERROR(`Insufficient arguments to diff/DT`)
   fi;
   temp:=Diff(f,y);
   RETURN(DT(temp,args[2..(nargs-1)]))
end;

`diff/DX`:=proc()
   local temp,y,f,i;
   f:=args[1]; y:=args[nargs];
   if nargs>=3 then
      for i from 2 to nargs-1 do;
         if not type(args[i],numeric) then RETURN(DX(args)) fi;
      od;
   else ERROR(`Insufficient arguments to diff/DX`)
   fi;
   temp:=Diff(f,y);
   RETURN(DX(temp,args[2..(nargs-1)]))
end;
```

```
# ================================================================ #
#                                                                  #
# These are procedures to handle simplification of the DX and DT   #
# operators. DT and DX must not be defined: they must remain as    #
# unevaluated function calls. The operators require simplifcation in #
# that DT and DX are not known by the system to commute with the   #
# star and dagger operations. Thus the function of each of these   #
# procedures is simply to take star and dagger operators outwith the #
# scope of the DX or DT operations. Note that the number of arguments#
# to DX or DT may be any number greater than 2. The second and     #
# subsequent arguments represent the subscripts of the differential #
# variables. For example, DX( A_[1,0] ,2) represents:              #
#                              d( A_[1,0])                         #
#                               dx2                                #
# Note that these subscripts will have to be put in some canonical #
# but that DX always should come before DT. ie DT(DX(.. makes sense #
# but will always be reaaranged to give DX(DT(..                   #
#                                                                  #
# ================================================================ #
`simplify/DT`:=proc(f)
   local temp,i,j,k,temp2,temp3,temp4;
   if f=0 then RETURN(0) fi;
   if not type(f,DT) then
      if not (hastype(f,`DT`) or hastype(f,`DX`) ) then RETURN(f)
      elif type(f,`+`) then RETURN(map(`simplify/DT`,f))
      elif type(f,`*`) then
         temp:=1;
         for i from 1 to nops(f) do;
            temp:=temp*(`simplify/DT`(op(i,f)))
         od;
         RETURN(temp);
# ie f=p**q. No mention of DT yet. DT may be in p.
      elif type(f,`**`) then
         RETURN( subsop(1=`simplify/DT`(op(1,f)),f) );
      elif type(f,`DX`) then
         RETURN(`simplify/DX`(f))
      elif type(f,`array`) then
         RETURN(map(`simplify/DT`, f)) # 15/6/92
      else
         RETURN(f)
      fi;
   elif type(f,`DT`) then
      temp2:=op(1,f);
#      whattype(temp2); # test 15/6/92
      if type(temp2,`star`) then
         temp3:=star(`simplify/DT`(DT(op(1,temp2),op(2..nops(f),f) )))
      elif type(temp2,`dagger`) then
         temp3:=dagger(`simplify/DT`(DT(op(1,temp2),op(2..nops(f),f) )))
      elif type(temp2,`array`) then
            print(`tttt`); # test15/6/92 if t then temp3=array.
         temp3:=`simplify/DT`(map(`DT`,op(temp2), op(2..nops(f),f) ));
      elif type(temp2,`+`) then
         temp3:=`simplify/DT`(map(`DT`,temp2,op(2..nops(f),f) ));
      elif type(temp2,`*`)then
         if nops(temp2)=2 then
            temp3:=
               `simplify/DT`(DT(op(1,temp2),op(2..nops(f),f)))*
                   op(2,temp2)+op(1,temp2)*
               `simplify/DT`(DT(op(2,temp2),op(2..nops(f),f)));
         else
            temp3:=`simplify/DT`(DT(op(1,temp2),op(2..nops(f),f)))*
                  product(op('j',temp2),'j'=2..nops(temp2))+
                  product(op('j',temp2),'j'=1..nops(temp2)-1)*
              `simplify/DT`(DT(op(nops(temp2),temp2),op(2..nops(f),f)));
            temp4:=0;
```

```
            for k from 2 to nops(temp2)-1 do;
               temp4:=temp4+
                   product(op('i',temp2),'i'=1..k-1)*
                   `simplify/DT`(DT(op(k,temp2),op(2..nops(f),f)))*
                     product(op('i',temp2),'i'=k+1..nops(temp2));
            od;
            temp3:=temp3+temp4;
         fi;
      elif type(temp2,`**`) then
         if type(op(1,temp2),`array`) then
            temp3:=f;
         elif type(op(1,temp2),`&*`) then
            temp3:=f
         elif type(op(1,temp2),`string`) and op(1,temp2)=`eps` then
            temp3:=temp2;
         elif type(op(1,temp2),{`+`,`*`}) then
            if not type(op(2,temp2),`integer`) then
               ERROR(`Must have integer powers.`) fi;
            temp3:=op(2,temp2)*
            `simplify/DT`(op(1,temp2))**(op(2,temp2)-1)*
            `simplify/DT`(DT(op(1,temp2),op(2..nops(f),f)));
# DT(f(x)**3) -> 3*f(x)**2*DT(f(x)) Note that  f(x) might contain DT
# so need to have two calls to simplify/DT one without s/DT(DT.
         elif type(op(1,temp2),`string`) then
            temp3:=0;
         else
            temp3:=f
         fi;
      elif type(temp2,`table`) then temp3:=f
      elif type(temp2,`indexed`) then temp3:=f
      elif type(temp2,`DX`) then
         temp4:=DX(DT(op(1,temp2),op(2..nops(f),f) ),
            op(2..nops(temp2),temp2));
         temp3:=`simplify/DX`(temp4);
      elif type(temp2,`DT`) then
         temp3:=DT(op(1,temp2),op(2..nops(temp2),temp2),
            op(2..nops(f),f) );
         elif type(temp2,`string`) then RETURN(0)
# Get round DT(-1,1) we hope. Lancaster modification.
         elif type(temp2,`numeric`) then RETURN(temp2)
         else RETURN(f); # temp3:=temp2
      fi;
      if type(temp3,`array`) then RETURN(op(temp3)) else
      RETURN(temp3)  fi; # Changed from only RETURN(temp3) 15/6/92
   elif type(f,`&*`) then
      RETURN(f)
   else RETURN(f); # DT(x,1)=0 unless x depends on T
   fi;
end;

`simplify/DX`:=proc(f)
   local i,j,k,temp,temp2,temp3,temp4,temp5;
   if f=0 then RETURN(0) fi;
   if not type(f,`DX`) then
      if not (hastype(f,`DX`) or hastype(f,`DT`) ) then RETURN(f)
      elif type(f,`+`) then RETURN(map(`simplify/DX`,f))
      elif type(f,`*`) then
         temp:=1;
         for i from 1 to nops(f) do;
            temp:=temp*(`simplify/DX`(op(i,f)))
         od;
         RETURN(temp);
      elif type(f,`**`) then
      print(`test1`);
         RETURN( subsop(1=`simplify/DX`(op(1,f)),f) );
```

```
        elif type(f,`DT`) then RETURN(`simplify/DT`(f));
        else RETURN(f)
        fi;
 # ie f=p**q. No mention of DX yet. DX may be in p.
     elif type(f,`DX`) then
        temp2:=op(1,f);
        if type(temp2,`star`) then
           temp3:=star(`simplify/DX`(DX(op(1,temp2),op(2..nops(f),f) )))
        elif type(temp2,`dagger`) then
           temp3:=dagger(`simplify/DX`(DX(op(1,temp2),op(2..nops(f),f) )))
        elif type(temp2,`array`) then
           temp3:=map(`DX`,temp2,op(2..nops(f),f) )
        elif type(temp2,`+`) then
           temp3:=`simplify/DX`(map(`DX`,temp2,op(2..nops(f),f) ));
        elif type(temp2,`*`)then
           if nops(temp2)=2 then
              temp3:=
                 `simplify/DX`(DX(op(1,temp2),op(2..nops(f),f)))*
                     op(2,temp2)+op(1,temp2)*
                 `simplify/DX`(DX(op(2,temp2),op(2..nops(f),f)));
           else
              temp3:=`simplify/DX`(DX(op(1,temp2),op(2..nops(f),f)))*
                    product(op('j',temp2),'j'=2..nops(temp2))+
                    product(op('j',temp2),'j'=1..nops(temp2)-1)*
                  `simplify/DX`(DX(op(nops(temp2),temp2),op(2..nops(f),f)));
              temp4:=0;
              for k from 2 to nops(temp2)-1 do;
                 temp4:=temp4+
                      product(op('i',temp2),'i'=1..k-1)*
                      `simplify/DX`(DX(op(k,temp2),op(2..nops(f),f)))*
                        product(op('i',temp2),'i'=k+1..nops(temp2));
              od;
              temp3:=temp3+temp4;
           fi;
        elif type(temp2,`**`) then
           if type(op(1,temp2),`array`) then
              temp3:=f;
           elif type(op(1,temp2),`&*`) then
              temp3:=f;
           elif type(op(1,temp2),`string`) and op(1,temp2)=`eps` then
              temp3:=temp2
           elif type(op(1,temp2),{`+`,`*`}) then
              if not type(op(2,temp2),`integer`) then
                 ERROR(`Must have integer powers.`) fi;
              temp3:=op(2,temp2)*
              `simplify/DX`(op(1,temp2))**(op(2,temp2)-1)*
              `simplify/DX`(DX(op(1,temp2),op(2..nops(f),f)));
 # DX(f(x)**3) -> 3*f(x)**2*DX(f(x)) Note that  f(x) might contain DX
 # so need to have two calls to simplify/DX one without s/DX(DX.
           elif type(op(1,temp2),`string`) then
              temp3:=0;
           else
              temp3:=f;
           fi;
        elif type(temp2,`table`) then temp3:=f;
        elif type(temp2,`indexed`) then temp3:=f
        elif type(temp2,`DT`) then
             temp4:=`simplify/DT`(temp2);
             if type(temp4,`DT`) then
                temp3:=DX(temp4,op(2..nops(f),f))
             else
                temp5:=DX(temp4,op(2..nops(f),f));
                temp3:=`simplify/DX`(temp5);
             fi;
        elif type(temp2,`DX`) then
```

```
         temp3:=DX(op(1,temp2),op(2..nops(temp2),temp2),
            op(2..nops(f),f) );
      elif type(temp2,`string`) then RETURN(0);
# Get round DT(-1,1) we hope. Lancaster modification.
         elif type(temp2,`numeric`) then RETURN(temp2)
      else RETURN(f);  #temp3:=temp2;
      fi;
      RETURN(temp3);
   elif type(f,`&*`) then
      RETURN(f)
   else RETURN(f) # DX(x,1)=0 unless x contains X.
   fi;
end;
```

```
#     Copyright notice
#     ----------------
#
#           Author   David Harper
#                    Computer Laboratory,
#                    University of Liverpool
#                    P.O. Box 147
#                    Liverpool L69 3BX
#                    ENGLAND
#
#                    (ALGEBRA @ UK.AC.LIVERPOOL)
#
#           Date     9 October 1987
#
#           Title    Series truncation routines
#
#     Copyright (c) David Harper 1987
#
#
#     Permission is granted to any individual or institution to
#     use, copy or re-distribute this software so long as it is
#     not sold for profit, provided that this copyright  notice
#     is retained and the file is not altered.
#
#     ---- End of copyright notice ----
#
# Series-handling routines
#
# Routine 'seriestrunc'
#
# This routine is designed to take a series expression (e.g. a Fourier
# series) and remove all terms whose order (determined by the given
# order-list) is higher than the specified maximum order.
#
# Usage : seriestrunc(f,olist,maxorder)
#
#         where f is any expression (which should be passed to 'expand'
#         beforehand for best effect), olist is an order-list (see below)
#         and maxorder is the highest order for retention of terms (a
#         non-negative integer).
#
# Order-lists
# -----------
#
# An order-list is a list of items of the form  name=integer which
# define the weight of each variable. For example, if an expression
# contains variables a,b,c where a is a first-order small quantity,
# b is a second-order small quantity and c is of zero-order (ie of the
# order of unity) then the following olist could be used
#
# olist := [a=1, b=2, c=0]
#
# Any name which does not appear in the olist is assumed to be of
# zero-order so that the olist [a=1, b=2] is equivalent to the one
# above.
#
seriestrunc := proc(f,orderlist,maxorder)
  local res,thisterm,nitems,k;
  if not type(maxorder,integer) then ERROR(`third arg not an integer`,
                                                    maxorder); fi;
  if maxorder<0 then ERROR(`max order must not be negative`,maxorder); fi;
  if type(f,`+`) then
    res := 0;
    nitems := nops(f);
    for k from 1 to nitems do
```

```
      thisterm := op(k,f);
      if termorder(thisterm,orderlist)<=maxorder then res := res+thisterm; fi;
    od;
    RETURN(res);
  else
    if termorder(f,orderlist)<=maxorder then RETURN(f);
                                        else RETURN(0);
    fi;
  fi;
end;
#
# Utility routine 'termorder'
#
# termorder returns the order of the first argument according to
# the order-list supplied as the second argument.
#
termorder := proc(f,orderlist)
  local currentorder,nitems,k,thisterm;
  if not type(orderlist,list) then ERROR(`second arg not a list`,orderlist);
  fi;
  for k from 1 to nops(orderlist) do
    if not type(op(k,orderlist),`=`) then ERROR(`illegal item in order list`,
                                          op(k,orderlist)); fi;
  od;
  if type(f,name) then RETURN(findorder(f,orderlist));
  elif type(f,`^`) then RETURN(op(2,f)*termorder(op(1,f),orderlist));
  elif type(f,`*`) then
    currentorder := 0;
    nitems := nops(f);
    for k from 1 to nitems do
      thisterm := op(k,f);
      currentorder := currentorder + termorder(thisterm,orderlist);
    od;
    RETURN(currentorder);
  else RETURN(0);
  fi;
end;
#
# Utility routine 'findorder'
#
# findorder returns the order assigned to the given name in the
# order-list.
#
# For example, findorder(a,[a=2,b=3,c=1]) yields 2
#
# whilst       findorder(q,[a=2,b=3,c=1]) yields 0 since q is not
#              in the olist
#
findorder := proc(f,orderlist)
  local thisname,k;
  if not has(orderlist,f) then RETURN(0);
  else
    for k from 1 to nops(orderlist) do
      thisname := op(k,orderlist);
      if f=op(1,thisname) then RETURN(op(2,thisname)); fi;
    od;
  fi;
  ERROR(`unable to match name in orderlist `,f, orderlist);
end;
```

```
# ================================================================ #
#                                                                  #
# Procedures necessary for calculating the required summations used  #
# replace the variables to be expanded. They require the global      #
# variables:-                                                      #
#    hiharm, groundstate, JMAX, nlist, newnlist, klist, newklist.    #
#    eps is the expansion parameter. theta is kx-wt.
#                                                                  #
# ================================================================ #
# Lancaster modification. Reverse order of variable name and row number
# of the array. eg. 1v_[0,1] causes a syntax error if entered at the
# keyboard but may be correctly determined by the program.

weeharmsum:=proc(var)
   local res,i;
   res:=sum('eps**(i+1)*(var[i,1]*exp(I*theta)+star(var[i,1])*exp(-I*theta))',
   'i'=0..JMAX);
   RETURN(res);
end;
bigharmsum:=proc(var)
   local res,i,j;
   res:=eps*(var[0,1]*exp(I*theta)+star(var[0,1])*exp(-I*theta))+
      sum('eps**(i+1)*sum('var[i,j]*exp(I*j*theta)
      +star(var[i,j])*exp(-I*j*theta)',
      'j'=1..2*JMAX)','i'=1..JMAX);
   RETURN(res);
end;
quantsum:=proc(var)
   local res,i,j,size;
   res:=table();
   if nargs<>1 then ERROR(`Procedure requires only one argument.`) fi;
   if not type(var,array) then ERROR(`Procedure requires an array as argument`)
   fi;
   size:=op(2,op(2,op(var)));   # upper array bound  #
   if not size>=2 then ERROR(`Array must have an index > 1.`) fi;
   if groundstate=true then
      res[1]:=(1-eps)*sum('eps**j*cat(var,1)[j,1]*exp(-I*theta/2)',
      'j'=0..JMAX);
      for i from 3 by 2 to size do;
         res[i]:=sum('eps**(j+1)*cat(var,i)[j,1]*exp(-I*theta/2)','j'=0..JMAX);
      od;
   else
      for i from 1 by 2 to size do;
         res[i]:=sum('eps**(j+1)*cat(var,i)[j,1]*exp(-I*theta/2)','j'=0..JMAX);
      od;
   fi;
   for i from 2 by 2 to size do;
      res[i]:=sum('eps**(j+1)*cat(var,i)[j,1]*exp(I*theta/2)','j'=0..JMAX);
   od;
   for i from 1 to size do;
      assign(var[i]=res[i]);
   od;
   RETURN(op(var));
end;
bigquantsum:=proc(var)
   local res,i,j,k,size;
   res:=table();
   if nargs<>1 then ERROR(`Procedure requires only one argument.`) fi;
   if not type(var,array) then ERROR(`Procedure requires an array as argument`)
   fi;
   size:=op(2,op(2,op(var)));   # upper array bound  #
   if not size>=2 then ERROR(`Array must have an index > 1.`) fi;
   if groundstate=true then
     res[1]:=(1-eps)*sum('eps**j*sum('cat(var,1)[j,k]*exp(-I*(2*k-1)*theta/2)',
      'k'=1..JMAX)','j'=0..JMAX);
```

```
      for i from 3 by 2 to size do;
         res[i]:=sum('eps**(j+1)*sum('cat(var,i)[j,1]*exp(I*(2*k-1)*theta/2)',
         'k'=1..JMAX)','j'=0..JMAX);
      od;
   else
      for i from 1 by 2 to size do;
         res[i]:=sum('eps**(j+1)*sum('cat(var,i)[j,1]*exp(I*(2*k-1)*theta/2)',
         'k'=1..JMAX)','j'=0..JMAX);
      od;
   fi;
   for i from 2 by 2 to size do;
      res[i]:=sum('eps**(j+1)*sum('cat(var,i)[j,1]*exp(I*(2*k-1)*theta/2)',
      'k'=1..JMAX)','j'=0..JMAX);
   od;
   for i from 1 to size do;
      assign(var[i]=res[i]);
   od;
   RETURN(op(var));
end;
```

```
# ================================================================ #
#                                                                  #
# This procedure is designed to perform Witham averaging on its argu-#
# ments. It checks to see whether a term contains the function exp.  #
# If it does, then the term is discarded. If not , it is retained.   #
# The procedure takes only one argument, the function to be averaged.#
#                                                                  #
# ================================================================ #
average:=proc(f)
   local i,j,temp;
   if nargs<>1 then ERROR(`Arguments to average incorrect.`) fi;
   if type(f,`+`) then RETURN(map(`average`,f))
   elif has(f,{`&*`,`array`}) then
      RETURN(f)
   elif type(f,`+`) then
      temp:=0;
      for j from 1 to nops(f) do;
         if has(op(j,f),`exp`) then
            temp:=temp+op(j,f)
         fi;
      od;
   RETURN(temp);
   elif has(f,`exp`) then
      RETURN(0)
   else
      RETURN(f)
   fi;
end;
```

```
# ================================================================== #
#                                                                    #
# This is a program which uses the procedures maptree and choptree - #
# contained in the files MAPTREE MPL and CHOPTREE MPL - to perform   #
# and expansion of a MAPLE    object without cousing the object too  #
# big error. Maptree maps the supplied argument and choptree -       #
# without arguments- then goes on to perform the expansion. The      #
# result is returned in the table oprands created by the choptree    #
# procedure. If the expansion has been produced a result which does  #
# not exceed the 32000 oprands restriction, the variable supplied as #
# an argument will be changed to give the result.                    #
#                                                                    #
# ================================================================== #
smartexpand:=proc(f)
   local temp,temp2;
   temp:=f;
   if assigned(totnodes) or assigned(typeinfo) or assigned(oprands) then
      if type(totnodes, `table`) or type(typeinfo, `table`) or
         type(oprands, `table`) then
         ERROR(`Smartexpand has been used previously during this run.`)
      else
         ERROR(`totnodes, typeinfo or oprands is already defined. `)
      fi;
   fi;
   maptree(temp);
   choptree();
   if totnodes[1]=0 then
      print(`Sucessfully simplified within operands limit.`);
      temp2:=oprands[0,1];
      RETURN(temp2);
   else
      print(`Operands limit exceeded during expansion.`);
      print(`Results to be found in table oprands[1,x].`);
      RETURN(NULL);
   fi;
end;

# ================================================================== #
#                                                                    #
# Use a recursive algorithm to map the tree structure for a given    #
# structure supplied as the sole argument. The results will be       #
# placed in three tables. The first table, totnodes, contains the    #
# highest or current node at that level. The second table,           #
# typeinfo, is a two dimensional table which stores the type         #
# information for each node at a given level. The type is either     #
# `+`, `*`, `**`, or `S` where `S` denotes that the expresssion is   #
# already in a suitable form. Thus typeinfo(level,node) contains the #
# information on the type of node given in the form of a one         #
# character string. The third table, oprands, contains the           #
# operands for each operator of type `S`. this table will be used to #
# reconstruct the expanded form of the expression since there is no  #
# obvious way to use a nexted form of nops with variable arguments.  #
#                                                                    #
# ================================================================== #
maptree:=proc(f);
   if assigned(tlevel) then ERROR(`tlevel is already assigned.`) fi;
   if assigned(totnodes) or assigned(typeinfo) or assigned(oprands) then
      ERROR(`Output tables for procedure maptree are already assigned.`)
   fi;
   totnodes:=table(sparse,[]);
   typeinfo:=table();
   oprands:=table();
   totnodes[0]:=1;
   if type(f,{`+`,`*`,`**`}) then
      typeinfo[0,1]:=[whattype(f),nops(f)]
```

```
    else
       typeinfo[0,1]:=['S',0]; oprands[0,1]:=f;
       RETURN()
    fi;
    tlevel:=0;
    domaptree(f);
    RETURN();
 end;
 domaptree:=proc(f)
    local node; node:=0;
    tlevel:=tlevel+1;
    do;
       node:=node+1;
       totnodes[tlevel]:=totnodes[tlevel]+1;
       if node>nops(f) then
          totnodes[tlevel]:=totnodes[tlevel]-1;
          tlevel:=tlevel-1;
          break;    #Does node exist?    #
       fi;
       if not hastype(op(node,f),'+') then
          typeinfo[tlevel,totnodes[tlevel]]:=['S',0];
          oprands[tlevel,totnodes[tlevel]]:=op(node,f);
          next;          # Go on to next node at this tlevel. #
       elif type(op(node,f),{'+','*','**'}) then
          typeinfo[tlevel, totnodes[tlevel]]:=[whattype(op(node,f)),
                   nops(op(node,f))];
          domaptree(op(node,f)); #Continue down this branch. #
       else
          typeinfo[tlevel, totnodes[tlevel]]:=['S',0];
          oprands[tlevel, totnodes[tlevel]]:=op(node,f); #BUG 22/8/91 Was :=f;
          next; # This branch is ok already. #
       fi;
    od;
    RETURN();
 end;

 # ================================================================ #
 #                                                                  #
 # This procedure is designed to take the information supplied by   #
 # maptree, namely the three tables of output, and to simplify the  #
 # information contained therein. It will truncate any operands to  #
 # given operators so that unnecessary terms are not retained. In   #
 # addition, it will check to see whether an answer about to be     #
 # generated will exceed the number of terms allowed which is given #
 # as maxterms. The output will be either the simplified expression #
 # or the tables themselves if the procedure fails to be able to    #
 # simplify the arguments it is supplied.                           #
 #                                                                  #
 # ================================================================ #
 choptree:=proc()
  local maxlevel,tlevel,i,tnode,ovrflo;
    if not (assigned(oprands) and assigned(totnodes) and assigned(typeinfo))
       then ERROR('Input table(s) not assigned.') fi;
    if not (type(oprands,'table') and type(typeinfo,'table')
       and type(totnodes,'table')) then
       ERROR('Input arguments to choptree are not tables as required.') fi;
    maxlevel:=0; ovrflo:=false;
    do;
       if totnodes[maxlevel]<>0 then maxlevel:=maxlevel+1
       else maxlevel:=maxlevel-1; break
       fi;
    od;
    tlevel:=maxlevel;
    for i from 1 to totnodes[tlevel] do;
       if not typeinfo[tlevel,i]=['S',0] then
```

```
            ERROR(`Choptree supplied with invalid map of tree.`) fi;
        od;  # All the highest level nodes would be simplified already.
        if tlevel>0 then tnode:=totnodes[tlevel-1]+1 fi;
        do;
           tlevel:=tlevel-1;
           if tlevel<0 then break fi;
           do;
              tnode:=tnode-1;
              if tnode<1 then # done this level.
                 for i from 1 to totnodes[tlevel] do;
                    if typeinfo[tlevel,i]<>[`S`,0] then
                      ovrflo:=true;
                    fi;
                 od;
                 if ovrflo then
                    print(`WARNING: maxterms exceeded at level`,tlevel)
                 else
                    print(`Completely simplified at level: `,tlevel+1);
                 fi;
                 if tlevel-1>=0 then tnode:=totnodes[tlevel-1]+1 fi;
                 break;
              fi;
              if typeinfo[tlevel,tnode]<>[`S`,0] then dochop(tlevel,tnode) fi;
           od;
        od;
    end;

    dochop:=proc(level,node)
        local temp;
        if not(type(level,`integer`) and type(node,`integer`)) then
           ERROR(`Non-integer values for level and node passed to dochop`) fi;
        temp:=typeinfo[level,node];
        if temp=[`S`,0] then ERROR(`Invalid call of dochop`) fi;
        if op(1,temp)=`+` then dochopplus(level,node,op(2,temp))
        elif op(1,temp)=`*` then dochoptimes(level,node,op(2,temp))
        elif op(1,temp)=`**` or op(1,temp)=`^`
           then dochoppow(level,node,op(2,temp))
        else print(op(1,temp));
           ERROR(`Invalid typeinfo table: S + ** ^ * only are allowed.`)
        fi;
    end;

    numterms:=proc(f);
        if type(f,`+`) then RETURN(nops(f))
        elif f=0 then RETURN(0)
        else RETURN(1) fi;
    end;

    xterm:=proc(num,f); if type(f,`+`) then RETURN(op(num,f))
        else RETURN(f) fi;
    end;

    dochopplus:=proc(level,node,numops)
        local maxterms,i,j,icount,jcount,allops,cc,
           nbrterms,jmax,xtra,startop,totterms,newnumops,
           temp1,temp2,temp4,temp5,temp6,numtemps,lowrstartop,lowrallops;
        maxterms:=32000;
        temp1:=table(); temp2:=table();
        startop:=0;
        allops:=sum('op(2,typeinfo[level,i])',
            'i'=1..totnodes[level]);
        if node>1 then
           for i from 1 to node-1 do;
              startop:=startop+op(2,typeinfo[level,i])
           od; # find out the start of our operands.
```

```
fi;
totterms:=0;
lowrallops:=sum('op(2,typeinfo[level+1,i])', 'i'=1..startop);
for i from startop+1 to startop+numops do;
   if typeinfo[level+1,i]=[`S`,0] then
      oprands[level+1,i]:=seriestrunc(oprands[level+1,i],[eps=1],JMAX);
      totterms:=totterms+numterms(oprands[level+1,i])
   elif op(1,typeinfo[level+1,i])=`+` then
    for cc from 1 to op(2,typeinfo[level+1,i]) do;
       oprands[level+2,lowrallops+cc]:=
          seriestrunc(oprands[level+2,lowrallops+cc],[eps=1],JMAX);
    od;
    totterms:=totterms+sum('numterms(oprands[level+2,lowrallops+cc])',
         'cc'=1..op(2,typeinfo[level+1,i]));
      lowrallops:=lowrallops+op(2,typeinfo[level+1,i])
   else
      ERROR(`Logic error in count of terms.`)
   fi;
od;
if totterms>maxterms then
   temp4:=totterms/maxterms;
   if trunc(temp4)=temp4 then
      newnumops:=temp4;
   else newnumops:=trunc(temp4)+1
   fi;
   typeinfo[level,node]:=[`+`,newnumops];
   numtemps:=0; lowrstartop:=0;
   for i from 1 to numops do;
      if typeinfo[level+1,startop+i]=[`S`,0] then
         numtemps:=numtemps+1;
         temp1[numtemps]:=oprands[level+1,startop+i]
      elif op(1,typeinfo[level+1,startop+i])=`+` then
         for j from 1 to op(2,typeinfo[level+1,startop+i]) do;
            numtemps:=numtemps+1;
            temp1[numtemps]:=oprands[level+2,j+lowrstartop];
         od;
         lowrstartop:=lowrstartop+op(2,typeinfo[level+1,startop+i]);
      else
         ERROR(`Logic error in calculation of temps.`)
      fi;
   od;
   for i from 1 to numops do;
     oprands[level+1,startop+i]:=evaln(oprands[level+1,startop+i]);
   od;
   temp2[1]:=0;
   i:=1; j:=1; jmax:=1;
   do;
      nbrterms:=numterms(temp1[i]);
      for jcount from 1 to jmax do;
         nbrterms:=nbrterms+numterms(temp2[jcount])
      od;
      if nbrterms<maxterms then
         temp2[1]:=temp2[1]+temp1[i];
         i:=i+1;
         if i>numops then ERROR(`Logic error in dochopplus.`) fi;
         next;
      fi;
      if nbrterms>maxterms*jmax then
         temp5:=nbrterms/maxterms;
         if trunc(temp5)=temp5 then temp6:=jmax; jmax:=temp5
         else temp6:=jmax; jmax:=trunc(temp5)+1
         fi;
         for jcount from temp6+1 to jmax do;
            temp2[jcount]:=0;
         od;
```

```
            fi;
            for icount from 1 to numterms(temp1[i]) do;
               if numterms(temp2[j])>=maxterms then j:=j+1 fi;
               temp2[j]:=temp2[j]+xterm(icount,temp1[i]);
            od;
            i:=i+1;
            if i>numtemps then break fi;
         od;
         if jmax<=newnumops and jmax>1 then
            do;
               if temp2[jmax]=0 then jmax:=jmax-1 else break fi;
            od;
         fi;
         if jmax>1 and temp2[jmax]<>0 then
            typeinfo[level,node]:=['+',jmax];
            xtra:=jmax-numops; # xtra might be -ve
            if xtra>0 then
               for jcount from allops by -1 to startop+numops+1 do;
                  oprands[level+1,jcount+xtra]:=oprands[level+1,jcount]
               od;
            elif xtra<0 then
               for jcount from startop+numops+1 by 1 to allops do;
                  oprands[level+1,jcount+xtra]:=oprands[level+1,jcount]
               od;
# Erase trailing arguments after shifting down.
               for jcount from allops+xtra+1 to allops do;
                  oprands[level+1,jcount]:=evaln(oprands[level+1,jcount]);
               od;
            fi;
            totnodes[level+1]:=totnodes[level+1]+xtra;
            for jcount from 1 to jmax do;
               oprands[level+1,jcount+startop]:=temp2[jcount]
            od;
         elif jmax=1 then
# managed to simplify in maxterms without overspill
            typeinfo[level,node]:=['S',0];
            oprands[level,node]:=temp2[jmax];
# erase old operands
            xtra:=-numops;
            totnodes[level+1]:=totnodes[level+1]+xtra;
            for jcount from startop+numops+1 by 1 to allops do;
               oprands[level+1,jcount+xtra]:=oprands[level+1,jcount]
            od;
# Erase trailing arguments after shifting down.
            for jcount from allops+xtra+1 to allops do;
               oprands[level+1,jcount]:=evaln(oprands[level+1,jcount]);
            od;
         else ERROR('Logic error concerning terms.')
         fi;
      else
         typeinfo[level,node]:=['S',0];
#         print(level,node,i);
         oprands[level,node]:=sum(oprands[level+1,'i'],'i'=
            startop+1..startop+numops);
         for jcount from startop+1 to startop+numops do;
             oprands[level+1,jcount]:=evaln(oprands[level+1,jcount])
         od;
# erase old operands
         xtra:=-numops;
         totnodes[level+1]:=totnodes[level+1]+xtra;
         for jcount from startop+numops+1 by 1 to allops do;
            oprands[level+1,jcount+xtra]:=oprands[level+1,jcount]
         od;
# Erase trailing arguments after shifting down.
         for jcount from allops+xtra+1 to allops do;
```

```
            oprands[level+1,jcount]:=evaln(oprands[level+1,jcount]);
         od
      fi;
   end;

   dochoptimes:=proc(level,node,numops)
      local maxterms,i,j,k,imax,jmax,kmax,startop,totterms,allops,xtra,
         newnumops,temp1,temp2,temp3,temp4,temp5,temp6,cc,ic,numtemps,
         nbrterms,icount,icount2,jcount,kcount,lowrstartop,lowrallops;
      maxterms:=32000;
      temp1:=table(); temp2:=table(); temp3:=table();
      startop:=0;
      allops:=sum('op(2,typeinfo[level,i])',
       'i'=1..totnodes[level]);
      if node>1 then
         for i from 1 to node-1 do;
           startop:=startop+op(2,typeinfo[level,i])
         od; #find out the start of our operands.
      fi;
      totterms:=1;
      lowrallops:=sum('op(2,typeinfo[level+1,i])', 'i'=1..startop); #0;
      lowrstartop:=lowrallops;
      for i from startop+1 to startop+numops do;
         if typeinfo[level+1,i]=['S',0] then
            oprands[level+1,i]:=seriestrunc(oprands[level+1,i],[eps=1],JMAX);
            totterms:=totterms*numterms(oprands[level+1,i])
         elif op(1,typeinfo[level+1,i])='+' then
            for cc from 1 to op(2,typeinfo[level+1,i]) do;
               oprands[level+2,lowrallops+cc]:=
                  seriestrunc(oprands[level+2,lowrallops+cc],[eps=1],JMAX);
            od;
            totterms:=totterms*
               sum('numterms(oprands[level+2,lowrallops+cc])',
                  'cc'=1..op(2,typeinfo[level+1,i]) );
            lowrallops:=lowrallops+op(2,typeinfo[level+1,i])
         else
            ERROR(`Logic error in count of terms.`)
         fi;
      od;
      if totterms>maxterms then  # must have a '+' in it somewhere.
         temp4:=totterms/maxterms;
         if  trunc(temp4)=temp4 then
            newnumops:=temp4;
         else newnumops:=trunc(temp4)+1;
         fi;
         typeinfo[level,node]:=['+',newnumops];
   # newnumops must be >= old numops
         numtemps:=0; # lowrstartop:=0;
         for i from 1 to numops do;
            if typeinfo[level+1,startop+i]=['S',0] then
               numtemps:=numtemps+1;
               temp1[i,1]:=oprands[level+1,startop+i]
            elif op(1,typeinfo[level+1,startop+i])='+' then
               for j from 1 to op(2,typeinfo[level+1,startop+i]) do;
                  numtemps:=numtemps+1;
                  temp1[i,j]:=oprands[level+2,lowrstartop+j];
               od;
               lowrstartop:=lowrstartop+op(2,typeinfo[level+1,i]);
            else
               ERROR(`Logic error in calculation of temps.`)
            fi;
         od;
         for i from 1 to numops do;
            oprands[level+1,startop+i]:=evaln(oprands[level+1,startop+i]);
         od;
```

```
        temp2[1]:=1;
        i:=1; j:=1; k:=1; jmax:=1; kmax:=1;
        do;
           imax:=0;
           do;
              if assigned(temp1[i,imax+1]) then imax:=imax+1 else break fi;
           od;
           nbrterms:=sum('numterms(temp1[i,ic])','ic'=1..imax)*
              sum('numterms(temp2[jcount])','jcount'=1..jmax);
           if nbrterms<=maxterms then
              temp3[1]:=expand(temp2[1]*temp1[i,1]);
              temp2[1]:=temp3[1];
              temp3[1]:=0; # evaln(temp3[1]);
              i:=i+1;
              if i>numops then ERROR(`Logic error in dochoptimes`) fi;
              next;
           fi;
           if nbrterms>maxterms*kmax then
              temp5:=nbrterms/maxterms;
              if trunc(temp5)=temp5 then temp6:=kmax; kmax:=temp5
              else temp6:=kmax; kmax:=trunc(temp5)+1
              fi;
              for kcount from 1 to kmax do; # temp6+1 to kmax
                 temp3[kcount]:=0;
              od;
           fi;
           for jcount from 1 to numterms(temp2[j]) do;
              for icount2 from 1 to imax do;
                 for icount from 1 to numterms(temp1[i,icount2]) do;
                    if numterms(temp3[k])>=maxterms then k:=k+1 fi;
                    temp3[k]:=temp3[k]+
                     xterm(icount,temp1[i,icount2])*xterm(jcount,temp2[j]);
                 od;
              od;
           od;
           j:=j+1;
           if j>jmax then
              for kcount from 1 to kmax do;
                 temp2[kcount]:=temp3[kcount];
                 temp3[kcount]:=0;
              od;
              jmax:=kmax;
              if i=numops then # finished.
                 break;
              else j:=1; k:=1; i:=i+1; # jmax:=kmax
              fi;
           fi;
        od;
        if jmax<=newnumops and jmax>1 then
           do;
             if temp2[jmax]=0 then jmax:=jmax-1 else break fi;
           od;
        fi;
        if jmax>1 and temp2[jmax]<>0 then
           typeinfo[level,node]:=[`+`,jmax];
           xtra:=jmax-numops; # xtra might be -ve, jmax=kmax 2 before break
           if xtra>0 then
              for kcount from allops by -1 to startop+numops+1 do;
                 oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
              od;
           elif xtra<0 then
              for kcount from startop+numops+1 by 1 to allops do;
                 oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
              od;
     # Erase trailing arguments after shifting down.
```

```
            for kcount from allops+xtra+1 to allops do;
               oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
            od;
         fi;
         totnodes[level+1]:=totnodes[level+1]+xtra;
         for jcount from 1 to jmax do;
            oprands[level+1,jcount+startop]:=temp2[jcount]
         od;
      elif jmax=1 then
# managed to simplify in maxterms without overspill
         typeinfo[level,node]:=['S',0];
         oprands[level,node]:=temp2[jmax];
#erase old operands
         xtra:=-numops;
         totnodes[level+1]:=totnodes[level+1]+xtra;
         for kcount from startop+numops+1 by 1 to allops do;
            oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
         od;
# Erase trailing arguments after shifting down.
         for kcount from allops+xtra+1 to allops do;
            oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
         od;
      else ERROR('Logic error concerning terms.')
      fi;
   else
      typeinfo[level,node]:=['S',0];
      oprands[level,node]:=expand(product(oprands[level+1,'i'],'i'=
         startop+1..startop+numops));
      for kcount from startop+1 to startop+numops do;
          oprands[level+1,kcount]:=evaln(oprands[level+1,kcount])
      od;
#erase old operands
      xtra:=-numops;
      totnodes[level+1]:=totnodes[level+1]+xtra;
      for kcount from startop+numops+1 by 1 to allops do;
         oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
      od;
# Erase trailing arguments after shifting down.
      for kcount from allops+xtra+1 to allops do;
         oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
      od
   fi;
end;

dochoppow:=proc(level,node,numops)
  local maxterms,i,j,k,imax,jmax,kmax,startop,totterms,allops,lowrallops,
      newnumops,temp1,temp2,temp3,temp4,temp5,temp6,cc,ic,ndx,
      nbrterms,icount,icount2,jcount,kcount,xtra,lowrstartop;
   if numops<>2 then ERROR('Wrong number of operands for a power.') fi;
   maxterms:=32000;
   temp1:=table(); temp2:=table(); temp3:=table();
   startop:=0;
   allops:=sum('op(2,typeinfo[level,i])',
    'i'=1..totnodes[level]);
   if node>1 then
      for i from 1 to node-1 do;
        startop:=startop+op(2,typeinfo[level,i])
      od; #find out the start of our operands.
   fi;
   if not type(oprands[level+1,startop+numops],'integer') then
      ERROR('Second argument of power is not an integer.') fi;
   ndx:=oprands[level+1,startop+numops];
   if typeinfo[level+1,startop+1]=['S',0] then
      oprands[level+1,startop+1]:=
      seriestrunc(oprands[level+1,startop+1], [eps=1], JMAX);
```

```
   totterms:=numterms(oprands[level+1,startop+1])**ndx;
elif op(1,typeinfo[level+1,startop+1])='+' then
   lowrallops:=sum('op(2,typeinfo[level+1,i])',
       'i'=1..startop); #0;
   lowrstartop:=lowrallops;
   totterms:=
      sum('numterms(oprands[level+2,lowrallops+cc])',
         'cc'=1..op(2,typeinfo[level+1,startop+1]) )**ndx;
else
   ERROR('Logic error in count of terms.')
fi;
if totterms>maxterms then  # must have a '+' in it somewhere.
   temp4:=totterms/maxterms;
   if trunc(temp4)=temp4 then
      newnumops:=temp4;
   else newnumops:=trunc(temp4)+1
   fi;
   typeinfo[level,node]:=['+',newnumops];
   for i from 1 to ndx do;
      if typeinfo[level+1,startop+1]=['S',0] then
         temp1[i,1]:=oprands[level+1,startop+1];
      elif op(1,typeinfo[level+1,startop+1])='+' then
         for j from 1 to op(2,typeinfo[level+1,startop+1]) do;
            temp1[i,j]:=oprands[level+2,lowrstartop+j];
         od;
      else
         ERROR('Error in calculation of temps.')
      fi;
   od;
   oprands[level+1,startop+1]:=evaln(oprands[level+1,startop+1]);
   oprands[level+1,startop+numops]:=
      evaln(oprands[level+1,startop+numops]);
   temp2[1]:=1;
   i:=1; j:=1; k:=1; jmax:=1; kmax:=1;
   do;
      imax:=0;
      do;
         if assigned(temp1[i,imax+1]) then imax:=imax+1 else break fi;
      od;
      nbrterms:=sum('numterms(temp1[i,ic])','ic'=1..imax)*
         sum('numterms(temp2[jcount])','jcount'=1..jmax);
      if nbrterms<=maxterms then
         temp3[1]:=expand(temp2[1]*temp1[i,1]);
         temp2[1]:=temp3[1];
         temp3[1]:=0; # evaln(temp3[1]);
         i:=i+1;
         if i>ndx then
             print(numterms(temp2[1]),numterms(temp1[3,1]) );
             print(level, node, numops);
             print(numterms(temp1[2,1]), numterms(temp1[1,1]) );
             print(evalb(temp1[1,1]=temp1[2,1]));
             print(evalb(temp1[2,1]=temp1[3,1]));
             print(nbrterms,imax,j,k,jmax,kmax);
             print(i,ndx,'i, ndx, in dochoppow.');
             break; # done without overflow.
         fi;
         next; # finished.
      fi;
      if nbrterms>maxterms*kmax then
         temp5:=nbrterms/maxterms;
         if trunc(temp5)=temp5 then temp6:=kmax; kmax:=temp5
         else temp6:=kmax; kmax:=trunc(temp5)+1
         fi;
         for kcount from 1 to kmax do; # temp6+1 to jmax
            temp3[kcount]:=0;
```

```
            od;
         fi;
         for jcount from 1 to numterms(temp2[j]) do;
            for icount2 from 1 to imax do;
               for icount from 1 to numterms(temp1[i,icount2]) do;
                  if numterms(temp3[k])>=maxterms then k:=k+1 fi;
                  temp3[k]:=temp3[k]+
                   xterm(icount,temp1[i,icount2])*xterm(jcount,temp2[j]);
               od;
            od;
         od;
         j:=j+1;
         if j>jmax then
            for kcount from 1 to kmax do;
               temp2[kcount]:=temp3[kcount];
               temp3[kcount]:=0
            od;
            jmax:=kmax;
            if i=ndx then # finished
               break;
            else j:=1; k:=1; i:=i+1; # jmax:=kmax;
            fi;
         fi;
      od;
      if jmax<=newnumops and jmax>1 then
         do;
            if temp2[jmax]=0 then jmax:=jmax-1 else break fi;
         od;
      fi;
# may have several empty entries especially in powers.
# if started a new element but did not need it. =0 so discard
      if jmax>1 and temp2[jmax]<>0 then
         typeinfo[level,node]:=['+',jmax];
         xtra:=jmax-numops; # xtra might be -ve, jmax=kmax 2 before break
         if xtra>0 then
            for kcount from allops by -1 to startop+numops+1 do;
               oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
            od;
         elif xtra<0 then
            for kcount from startop+numops+1 by 1 to allops do;
               oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
            od;
# Erase trailing arguments after shifting down.
            for kcount from allops+xtra+1 to allops do;
               oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
            od;
         fi;
         totnodes[level+1]:=totnodes[level+1]+xtra;
         for jcount from 1 to jmax do;
            oprands[level+1,jcount+startop]:=temp2[jcount]
         od;
      elif jmax=1 then
# managed to simplify in maxterms without overspill
         typeinfo[level,node]:=['S',0];
         oprands[level,node]:=temp2[jmax];
# erase old operands.
         xtra:=-numops;
         totnodes[level+1]:=totnodes[level+1]+xtra;
         for kcount from startop+numops+1 by 1 to allops do;
            oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
         od;
# Erase trailing arguments after shifting down.
         for kcount from allops+xtra+1 to allops do;
            oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
         od;
```

```
      else ERROR('Logic error concerning terms.')
      fi;
   else
      typeinfo[level,node]:=['S',0];
      oprands[level,node]:=expand(oprands[level+1,startop+1]**
        oprands[level+1,startop+numops]);
      for kcount from startop+1 to startop+numops do;
          oprands[level+1,kcount]:=evaln(oprands[level+1,kcount])
      od;
# erase old operands.
      xtra:=-numops;
      totnodes[level+1]:=totnodes[level+1]+xtra;
      for kcount from startop+numops+1 by 1 to allops do;
         oprands[level+1,kcount+xtra]:=oprands[level+1,kcount]
      od;
# Erase trailing arguments after shifting down.
      for kcount from allops+xtra+1 to allops do;
         oprands[level+1,kcount]:=evaln(oprands[level+1,kcount]);
      od;
   fi;
end;
```

```
# ================================================================ #
#                                                                  #
# This is a function, ordindex, which puts the indices of the      #
# derivative functions DX and DT in ascending order. The argument  #
# must be a polynomial in which DX and DT functions are present and #
# have their usual varying number of arguments. In all cases of    #
# mixed derivatives, the order of the derivatives should be :      #
#                                                                  #
#                              DX(DT(....),..)                     #
#                                                                  #
# ================================================================ #
ordindex:=proc(f)
   local i,j,temp1,temp2,temp3,temp4;
   if not (hastype(f,`DX`) or hastype(f,`DT`) ) then RETURN(f)
   elif type(f,{`+`,`*`,`list`}) then RETURN(map(ordindex,f))
   elif type(f,`**`) then RETURN(subsop(1=ordindex(op(1,f),f)))
   elif type(f,`DX`) then
      temp1:=op(1,f); i:=nops(f);
      if type(temp1,`DT`) then
         temp2:=op(1,temp1); j:=nops(temp1);
         if type(temp2,`DX`) or type(temp2,`DT`) then
           ERROR(`Argument `,f,` to ordindex not simplified.`)
         fi;
         temp3:=DT(temp2,op(sort([op(2..j,temp1)],`<=`)));
      else temp3:=temp1
      fi;
      temp4:=DX(temp3,op(sort([op(2..i,f)],`<=`)));
      RETURN(temp4);
   elif type(f,`DT`) then
      temp1:=op(1,f); i:=nops(f);
      if type(temp1,`DX`) or type(temp1,`DT`) then
        ERROR(`Argument `,f,` to ordindex not simplified.`)
      fi;
      temp2:=DT(temp1,op(sort([op(2..i,f)],`<=`)));
      RETURN(temp2);
   else RETURN(f);
   fi;
end;
```

```
# ================================================================ #
#                                                                  #
# This procedure is used as an argument to sort when ordering lists #
# of DT, DX, or DTDX terms as produced by the mapmod procedure.    #
# dordbool should return true if its first argument precedes its   #
# second in the desired ordering. Factors to be taken into account #
# are the number of derivatives, their numerical size (ie. t1,t2   #
# etc.), and the root of their arguments. Note that the procedure  #
# ordindex will have been applied to put the derivative indices into #
# order and that the expression is assumed to have been simplified #
# using the relevant simplify procedures. Only the ordering DXDT is #
# valid.                                                           #
#                                                                  #
# ================================================================ #
dordbool:=proc(a,b)
   local res1;
   if type(a,`DT`) then
      if not type(b,`DT`) then ERROR(`Non-matching argument types`) fi;
      res1:=doordD1arg(a,b);
      RETURN(res1);
   elif type(a,`DX`) then
      if not type(b,`DX`) then ERROR(`Non-matching argument types`) fi;
      if not hastype(a,`DT`) then
         if hastype(b,`DT`) then ERROR(`Non-matching argument types`) fi;
         res1:=doordD1arg(a,b)
      else
         if not hastype(b,`DT`) then
            ERROR(`Non-matching argument types`) fi;
         res1:=doordDXDT(a,b)
      fi;
   else
      ERROR(`Invalid arguments.`)
   fi;
end;
doordD1arg:=proc(a,b)
   local i,root1,root2,alist1,alist2,temp1,templist,templist2;
   readlib(zip);
   root1:=op(1,a);
   root2:=op(1,b);
   alist1:=[op(2..nops(a),a)];
   alist2:=[op(2..nops(b),b)];
   if root1<>root2 then
      temp1:=sort([root1,root2], fordbool);
      if temp1=[root1,root2] then
         RETURN(true)
      else RETURN(false)
      fi;
   elif nops(alist1)<>nops(alist2) then
      if nops(alist1)<nops(alist2) then
         RETURN(true)
      else RETURN(false)
      fi;
   else
      templist:=zip(szip,alist1,alist2);
      templist2:=zip(lzip,alist1,alist2);
      for i from 1 to nops(alist1) do;
         if op(i,templist)<>op(i,templist2) then
            RETURN(false);
         fi;
      od;
      RETURN(true);
   fi;
end;
doordDXDT:=proc(a,b)
   local root1,root2,alist1,alist2,temp1,temp2;
```

```
    root1:=op(1,a);
    root2:=op(1,b);
    alist1:=[op(2..nops(a),a)];
    alist2:=[op(2..nops(b),b)];
    if root1=root2 then
       temp1:=DX('dummy',op(alist1));
       temp2:=DX('dummy',op(alist2));
       RETURN(doordD1arg(temp1,temp2))
    else
       RETURN(doordD1arg(root1,root2))
    fi;
end;
lzip:=proc(a,b) [a,b] end;
szip:=proc(a,b) sort([a,b]) end;
fordbool:=proc(a,b)
    local ord1,harm1,ord2,harm2,root1,root2;
    if not type(a,`indexed`) or not type(b,`indexed`) then
       ERROR(`Invalid arguments.`) fi;
    root1:=op(0,a);
    root2:=op(0,b);
    harm1:=op(2,a);
    harm2:=op(2,b);
    ord1:=op(1,a);
    ord2:=op(1,b);
    if root1<>root2 then
       if sort([root1,root2], lexorder)=[root1,root2] then
         RETURN(true)
       else RETURN(false)
       fi;
    elif ord1<>ord2 then
       if sort([ord1,ord2])=[ord1,ord2] then
          RETURN(true)
       else RETURN(false)
       fi;
    elif harm1<>harm2 then
       if sort([harm1,harm2])=[harm1,harm2] then
         RETURN(true)
       else RETURN(false)
       fi;
    else
       if a<>b then
           ERROR(`Logic error.`)
       else RETURN(true)
       fi;
    fi;
end;
```

```
# ================================================================== #
#                                                                    #
# This is a modified version of the program maptree. This version    #
# merely uses the recursive method of the previous incarnation to put#
# all occurences of DT or DX or DX(DT into three sets: dxvar, dtvar, #
# dxdtvar. These sets will later be used in the creation of the      #
# Euler-Lagrange equations.                                          #
#                                                                    #
# ================================================================== #
mapmod:=proc(f);
   if assigned(DXvar) or assigned(DTvar) or assigned(DXDTvar)
      or assigned(modlevel) then
         if prinlvl>4 then
            print(`Mapmod used previously`);
         fi;
      fi;
   DXvar:={};
   DTvar:={};
   DXDTvar:={};
   modlevel:=0;
   domapmod(f);
   RETURN();
end;

domapmod:=proc(f)
   local node; node:=0;
   modlevel:=modlevel+1;
   do;
      node:=node+1;
      if node>nops(f) then
         modlevel:=modlevel-1;
         break;    #Does node exist?    #
      fi;
      if not hastype(op(node,f),`DT`) and
           not hastype(op(node,f),`DX`) then
           next; # no point in going on.
      elif type(op(node,f),`DT`) then
         DTvar:=DTvar union {op(node,f)};
         next; # This branch is ok already. #
      elif type(op(node,f),`DX`) then
         if not hastype(op(node,f),`DT`) then
            DXvar:=DXvar union {op(node,f)}
         else
            DXDTvar:=DXDTvar union {op(node,f)}
         fi;
         next; # This branch is ok already. #
      elif type(op(node,f),{`+`,`*`,`**`,`star`}) then
         domapmod(op(node,f)); #Continue down this branch. #
      else
         ERROR(`Logic error in domapmod.`);
      fi;
   od;
   RETURN();
end;
```

```
# ================================================================== #
#                                                                    #
# This is a program which is designed to determine the equations of  #
# motion of a given lagrangian. The function assumes that the        #
# variable JMAX is the highest order of expansion; that hiharm will  #
# affect the expansion of the variables in the way given in SUMS     #
# MPL; that newnlist contains a list of the root of scalar           #
# variables; that newklist contains a similar list of vector         #
# variables; that the lagrangian is supplied as the argument. The    #
# procedure first finds the variables with which subsequent          #
# manipulations will concern themselves. It then finds the DX, DT    #
# and DX(DT derivatives in order that it may determine the relevant  #
# Euler-Lagrange equation to use. The sets scalvar, vecvar are used  #
# to store the variables found.                                      #
# Later on the program derives the correct Euler=Lagrange equation   #
# to use and then applies it to derive the equations of motion.      #
#                                                                    #
# ================================================================== #
equationsofmotion:=proc(L)
   local ind1,ind2,ind3;
   if not assigned(`JMAX`) or not assigned(`newnlist`) or not
      assigned(`newklist`) or not assigned(`hiharm`) then
      ERROR(`Necessary variables not assigned.`) fi;
   if assigned(`scalvar`) or assigned(`vecvar`) or
      assigned(`DTvar`) or assigned(`DXvar`) or
      assigned(DXDTvar) then
         if prinlvl>4 then
            print(`Procedure mapmod has been used before.`);
         fi;
         scalvar:={}; vecvar:={}; DTvar:={}; DXvar:={}; DXDTvar:={};
   fi;
   scalvar:={}; vecvar:={};
   for var in newnlist do;
      for ind1 from 0 to JMAX do;
         if hiharm then
            for ind2 from 0 to JMAX do;
               if has(L,var[ind1,ind2]) then
                  scalvar:=scalvar union {var[ind1,ind2]} fi;
            od;
         else
            ind2:=1;
            if has(L,var[ind1,ind2]) then
               scalvar:=scalvar union {var[ind1,ind2]} fi;
         fi;
      od;
   od;
   for var in newklist do;
      for ind1 from 0 to JMAX do;
         for ind3 from 1 to op(2,op(2,op(1,var))) do; # upper array bound.
            if ind3>10 then ERROR(`Ind3 too big.`) fi;
            if hiharm then
               for ind2 from 0 to JMAX do;
                  if has(L,cat(var,ind3)[ind1,ind2]) then
                     vecvar:=vecvar union {cat(var,ind3)[ind1,ind2]} fi;
               od;
            else
               ind2:=1;
               if has(L,cat(var,ind3)[ind1,ind2]) then
                  vecvar:=vecvar union {cat(var,ind3)[ind1,ind2]} fi;
            fi;
         od;
      od;
   od;
   mapmod(L);
```

```
    RETURN(NULL);
end;
```

```
# ============================================================== #
# procedure to return the derivative order of a given argument of #
# type DX, DT or DXDT. Note that type DTDX is not allowed and so  #
# all functions are assumed to ahve been simplified using         #
# simpdtdx.                                                       #
# ============================================================== #
derivord:=proc(f)
   local xtargs,xargs,targs;
   if not (type(f,`DX`) or type(f,`DT`)) then
      ERROR(`Argument must be of type DX, DT or DX(DT.`) fi;
   if type(f,`DT`) then
      if has(op(1,f),`DT`) or has(op(1,f),`DX`) then
         ERROR(`Arguments must be simplified before calling this fn.`)
      fi;
      targs:=nops(f)-1;
      RETURN(targs);
   elif type(f,`DX`) then
      if type(op(1,f),`DT`) and not (has(op(1,op(1,f)),`DX`)
         or has(op(1,op(1,f)),`DT`) ) then
         xargs:=nops(f)-1;
         targs:=nops(op(1,f))-1;
         xtargs:=xargs+targs;
         RETURN(xtargs);
      elif not (has(op(1,f),`DX`) or has(op(1,f),`DT`)) then
        xargs:=nops(f)-1;
        RETURN(xargs);
      else ERROR(`Argument must be simplified first.`)
      fi;
   else ERROR(`Logic error.`)
   fi;
end;
```

```
# ========================================================================== #
# This is a procedure to generate a term in an Euler-Lagrange equation.    #
# The lagrangian is given as L; newscalvar, newvecvar, newDXvar,           #
# newDTvar, newDXDTvar are all assumed to have been defined; the           #
# procedure uses the procedure derivord.mpl to genereate a number which    #
# givethe sign of the term. The argument, f, to the function is the term   #
# which has been located in the lagrangian and whose E-L term is to be     #
# found.                                                                   #
#       The procedure finds:-                                              #
#                                                                          #
#             derivord(arg)    derivord(arg)                               #
#       (-1)                * d              [del L/del arg]              #
#                             d(args of arg)                               #
#                                                                          #
# ========================================================================== #
makeELterm:=proc(f,L)
   local signpow,dLdarg,tout,toutT,toutX,toutXT;
   if not (type(f,'DX') or type(f,'DT')) then
      signpow:=0;
   else signpow:=derivord(f);
   fi;
   dLdarg:=frontend(diff,[L,star(f)],[{'+','*','**'},{}]);
   if type(f,'DT') then
      toutT:=DT(dLdarg,op(2..nops(f),f) );
      tout:=simplify(toutT,'DT'); # ordindex
   elif type(f,'DX') and not type(op(1,f),'DT') then
      toutX:=DX(dLdarg,op(2..nops(f),f) );
      tout:=simplify(toutX,'DX'); # ordindex
   elif type(f,'DX') and type(op(1,f),'DT') then
      toutXT:=DX(DT(dLdarg,op(2..nops(op(1,f)),op(1,f)) ),op(2..nops(f),f) );
      tout:=simplify(toutXT,'DX','DT'); # ordindex
   elif not (has(f,'DT') or has(f,'DX') ) then
      tout:=dLdarg; # No DX or DT so no differentiation: dL/darg**(-1)**0
   else ERROR('Logic error.');
   fi;
   RETURN((-1)**signpow*tout);
end;
```

```
#####################################################################
# This is a program which is designed to print out the results      #
# of the averaging program. If assumes that the equations are       #
# held in arrays whose names are held in the sets scalareqnset      #
# and vectoreqnset. After checking for the existence of these       #
# sets, the program goes on to print out the equations. Its         #
# main function is to determine the size of the arrays. If itis     #
# desirable, a pretty-printing engine could be written to print     #
# the expressions out in a more appealing manner.                   #
# Note also that the variable JMAX must have been defined.          #
#####################################################################
doprinteqns:=proc()
   local i,j,scalname,vecname;
   if (not assigned(scalareqnset) or not assigned(vectoreqnset)
      or not assigned(JMAX)) or (not type(scalareqnset,set)
      or not type(vectoreqnset,set)) then
      ERROR(`Procedure doprint called inappropriately.`)
   fi;
#   for levcount from 0 to JMAX do;
      if scalareqnset<>{} then
         print(`The equations of motion for the scalar variables are:-`);
         print();
         for scalname in scalareqnset do;
#           print(scalname,`TATEST1`);
           if not type(scalname,table) then ERROR(`Logic error in doprint.`) fi;
           for i from 3 by -1 to 0 do; # JMAX do;
              for j from 3 by -1 to 0 do; # JMAX do;
                  if assigned(scalname[i,j]) and scalname[i,j]<>0 then
     print(`The equation of motion for `,cat(substring(scalname,6..100),`[`,i,`,`,j,`]`),` is:-`);
                     print(scalname[i,j],`=0`);
                     print();
                  fi;
              od;
           od;
       od;
      fi;
      if vectoreqnset<>{} then
        print(`The equations of motion for the vector variables are:-`);
        print();
        for vecname in vectoreqnset do;
#           print(vecname,`TATEST2`);
           if not type(vecname,table) then ERROR(`Logic error in doprint.`) fi;
           for i from 3 by -1 to 0 do; # JMAX do;
              for j from 3 by -1 to 0 do; # JMAX do;
                  if assigned(vecname[i,j]) and vecname[i,j]<>0 then
     print(`The equation of motion for `,cat(substring(vecname,6..100),`[`,i,`,`,j,`]`),` is:-`);
                     print(vecname[i,j],`=0`);
                     print();
                  fi;
              od;
           od;
       od;
      fi;
   # od;
   RETURN(NULL):
end;
```

A *MAPLE* IMPLEMENTATION OF WHITHAM'S AVERAGED LAGRANGIAN TECHNIQUE.

T.D. Arbuckle, J.M. Arnold,
Department of Electronics and Electrical Engineering,
The University of Glasgow,
Glasgow, G12 8QQ,
Scotland,
U.K..

Abstract

Whitham's method of averaged lagrangians is a very powerful method for examining the behaviour of nonlinear systems which can be put in lagrangian form. Using this method, one can determine the equations which govern the evolution of the envelope of an almost periodic carrier provided that the rate of change of the carrier is much slower than that of the envelope - in effect, a two-timing approach. Additionally, the lagrangian formulation has the benefit of avoiding secularities which appear when using other methods.

In an extension of the method due to Kawahara, Whitham's procedure is combined with the method of multiple scales, thus producing a technique which may correctly model any slowly varying wavetrain in two (or more) dimensions and may be shown to give results which reduce to the nonlinear Schrodinger or Korteweg-de Vries equations in appropriate limits. It is this variant which has been implemented using the *MAPLE* computer algebra package.

The mechanics of the procedure require the introduction of a small parameter which is a measure of the differing scales of changes in the carrier and in the envelope. One then goes on to substitute power series in terms of this small parameter for all derivative operators and for all functions within the lagrangian whose extremals are to be found. Even for moderately simple lagrangians and fairly low orders, this quickly gives rise to thousands of terms which then require to be integrated (averaged) in order to obtain the necessary equations of motion. This renders the technique impractical for all but the most essential work.

The program that has been written almost exactly mirrors the algorithm used when carrying out the calculations by hand. After determining the validity of the input file- which may contain any suitable meaningful lagrangian together with some switches which affect levels of expansion and program output- the program then proceeds to produce the expanded form of the lagrangian; to simplify and expand it; and to average it by simulating integration. It then goes on to separate terms in order of the small parameter thus generating a sequence of lagrangians, one for each order. For each of these lagrangians, the program then determines the equations of motion which must be obeyed, saving the results in machine readable form for future manipulation. For simple lagrangians, useful results- which appear at or above the same order as nonlinear Schrodinger type equations- can be obtained for around one hour's CPU time on an Amdahl 5890 running under CMS. For more complicated problems, however, it is necessary to take advantage of the larger amount of contiguous memory available under Unix. The processing is then carried out on a Sequent Symmetry S81: the program runs more slowly, however, due to hardware limitations.

At present the program will handle scalar or vector lagrangians in real or complex quantities in two dimensions. The extension to three or more dimensions would be fairly simple. Currently, modelled systems must be conservative: there is no facility for correctly modelling loss. A further variant of the Whitham method due to Ostrovsky and Pelinovsky could be coded to allow for this although this would involve some rewriting of the program. Nevertheless, the program allows the calculation of results with a facility which would not have existed before the advent of symbolic manipulation packages. It is planned to utilise this by investigating a variety of lagrangians from a nonlinear optics context.

Acknowledgements.

The author would like to acknowledge the assistance of Dr. David Harper (University of London) in the writing of this software. His help and encouragement are greatly appreciated. The author would also like to thank the computing services at Glasgow, Manchester and Lancaster Universities.