ON A TECHNIQUE FOR THE NUMERICAL INVERSION OF THE LAPLACE AND MELLIN TRANSFORMS

by

T. S. HARRIS

1963

****

ProQuest Number: 13834211

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 13834211

Published by ProQuest LLC (2019). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

CONTENTS

Acknowledgement

Section

1. Introduction

2. General Problem of Inverting an Integral Transform

3. Numerical Inversion of the Laplace Transform

4. Numerical Procedure and the results of applying the method to the transforms of some known functions

5. Advantages of the M G Method

6. A Possible Algorithm for the Mellin Transform

Appendix 1 Solution of Inverse Problems Using the Laplace Transform

Appendix 2 ALGOL Program Developed

References

Acknowledgement

I would like to thank the members of the Glasgow University Computing Department for their helpful advice and "machine time" needed in the development of the ALGOL program, and also to thank Professor I.N. Sneddon for his encouragement and inspiration.

Further, I would like to thank the Science Research Council for providing the Advanced Course Studentship which I held while doing the work for this thesis.

Introduction

An important part of the usefulness of the Laplace transform (and other integral transforms) lies in its power to reduce the transcendence level of a functional equation involving such transcendental operators as differentiation and convolution. For example, if y is the solution of a linear ordinary differential equation with constant coefficients, then the Laplace transform Y of y satisfies a linear algebraic equation rather than a differential equation. Again, a partial differential equation in n independent variables is reduced to one in n-1 independent variables. However, the price paid for this powerful technique is the problem of inverting the transformed solution when its explicit form has been found. Extensive tables of known transform pairs do exist but these are for the cases where the complex inversion formula

$$f(t) = L^{-1}\{F(s);\ s \to t\} = \frac{1}{2\pi} \int_{c-i\infty}^{c+i\infty} F(s).e^{st}\, ds$$

is more or less tractable.

Faced with a mathematical problem of great complexity arising from an applied science we can either simplify the mathematical model until an analytic (or approximate analytic) solution can be obtained or else attempt to analyse numerically the original model in all its complexity. In the past the former was certainly the easier and sometimes successful, if risky, procedure.

But nowadays the electronic computer has opened up the second, again not entirely risk-free, path. This is the situation facing the user of the Laplace transform.

However, once it is decided to embark upon a numerical solution of the problem in hand, it may be better to return to the original equations and solve these directly rather than invert the Laplace transformed solution numerically. In some cases the Laplace transform approach is found in practice to be definitely superior to the classical numerical techniques. For some examples of particular applications see appendix 1. Another point in favour of the Laplace transform approach is that the computer program written to invert the transform numerically, can be applied in a routine way to several types of problem that would need several different classical algorithms, and this should make numerical and computing work less arduous for the occasional computor.

Accepting this defence of the Laplace transform method followed by numerical inversion, this dissertation endeavours to discuss and expound a particular numerical inversion algorithm that has been found successful in practice, together with a possible application to an inverse problem in viscoelastic theory and also a possible treatment of the Mellin transform on the same lines.

The General Problem of inverting an Integral Transform

The problem of inverting an integral transform K

$$Kf(s) \equiv \int_a^b k(s,t) \cdot f(t) \cdot dt = g(s)$$

is the problem of solving a Fredholm integral equation of the first kind. Whereas there is an extensive literature on Fredholm equations of the second kind, there is not much on those of the first kind: see D.L. Phillips 1 . An important characteristic of an integral operator such as K is that its inverse is unbounded so that the problem: given g, find $f = K^{-1}g$, is "ill-posed". Small changes in g can yield very large changes in f. To see this, suppose f is the solution of

$$\int_a^b k(s,t) \cdot f(t) \cdot dt = g(s) \quad (a \leq s \leq b) \qquad (2.1)$$

and add to f the function $f_m(t) = \sin(mt)$.

Then by the Riemann-Lebesgue lemma, for any integrable kernel

$$g_m(s) = \int_a^b k(s,t) \sin(mt) \, dt \to 0, \text{ as } m \to \infty,$$

uniformly on $[a,b]$ when $k(s,t)$ is continuous in s on the compact set $[a,b]$. Hence by choosing m large enough, f and $f + f_m$ will be mapped by K to points Kf, $K(f + f_m)$ in the range space which are arbitrarily close (in the uniform metric).

Also we would expect $g_m \to 0$ as $m \to \infty$ faster for flat smooth kernels than for sharply peaked kernels. (If $k(s,t) = \delta(s - t)$ then $g_m = f_m \not\to 0$).

Hence we conclude that success in solving equation (2.1) by any method depends largely on the accuracy of $g(s)$ and the shape of $k(s,t)$.

We now discuss two possible methods of solution of equation (2.1).

1. We approximate the operator K (operating on L_2 (a,b) by a matrix operator K_n (operating on R^n, the space of n-tuples of reals) leaving the problem of solving

$$K_n f_n = g_n \qquad (2.2)$$

where f_n and g_n are discrete approximations to the unknown and known functions repectively. The matrix K_n is obtained by approximating the integral by a discrete quadrature rule. The error of this approximation will decrease as n increases. Having got K_n, then providing $\det(K_n) \neq 0$ the solution of (2.2) would be straight forward on an 'ideal' computer working in infinite length arithmetic. In practice it is found that the solution is hampered by ill conditioning of the matrix K_n which grows with increasing n; this reflects the unstable nature of the original operator K^{-1} and we now have an ill-posed problem in the space R^n. Using this method we can expect the error to first decrease with increasing n, as the quadrature formula becomes more exact, and then increase as the ill-conditioning of K_n swamps this improvement.

2. The second method falls in the category of projection methods. The Fourier expansion of the unknown function f in terms of a complete orthonormal set of functions $\{Q_n\}$ is

$$f = \sum_{n=0}^{\infty} (f, Q_n) \cdot Q_n$$

Now we can truncate the Fourier development, or in geometric language consider the projection of f on the subspace spanned by

$$\{ Q_0, \cdots, Q_n \}$$

$$f \simeq \sum_{m=0}^{n} (f, Q_m) \cdot Q_m$$

and with appropriate conditions on the smoothness of f we can bound the error of this approximation. We now substitute this approximation to f in the integral equation

$$\int_a^b k(s,t) \sum_{m=0}^{n} (f, Q_m) \cdot Q_m(t) \cdot dt = g(s)$$

or

$$\sum_{m=0}^{n} \left[\int_a^b k(s,t) \cdot Q_m(t) \cdot dt \right] \cdot (f, Q_m) = g(s)$$

If the integrals $\int_a^b k(s,t)\, Q_m(t)\, dt$ $(m = 0, \ldots, n)$ can be evaluated then Fourier coefficients (f, Q_m) $(m = 0, \ldots, n)$ of f can be found by solving a system of linear equations possibly with the difficulties of ill-conditioning.

Numerical Inversion of the Laplace transform

The two methods outlined above are exemplified by the algorithms given recently by R.E. Bellman, R.E. Kalaba, and J. Lockett [2] 1966 (method 1) and by M.K. Miller and W.T. Guy, Jr., [3] 1966 (method 2). We discuss briefly the Bellman, Kalaba and Lockett method (abbreviated BKL) and then go on to develop the Miller and Guy (MG) method in full.

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt$$

The BKL method begins by changing the variable of integration: $x = e^{-t}$ to obtain a finite interval of integration

$$F(s) = \int_0^1 x^{s-1} f(-\log x)\, dx = \int_0^1 x^{s-1} g(x)\, dx$$

The approximation K_n to K is now obtained using Gauss- Legendre quadrature (but with the shifted Legendre Polynomials P_n^* defined on $[0,1]$) to get

$$\sum_{i=1}^{n} w_i x_i^{s-1} g(x_i) = F(s)$$

where $\{x_i\}$ are the n roots of $P_n^*(x)$ and $\{w_i\}$ are the associated weights. Letting s assume n different values say $s = 1,2, \ldots ,n$ yields a linear system

$$\sum_{i=1}^{n} w_i x_i^{k} g(x_i) = F(k+1) \quad (k = 0, \ldots ,n-1.)$$

After some manipulation using the Lagrange interpolation formula and some properties of the Legendre polynomial P_n^* we get an explicit relation for the elements of the inverse of the matrix $(w_i x_i^k)$ in terms of the x_i and the coefficients of P_n^*, both of which can be calculated to any required degree of accuracy. Thus the inverse matrices K_n^{-1} can be computed to any required degree of accuracy and stored on magnetic tape beforehand. Then when the vector F is given, the vector f is computed by a straight forward matrix product routine in a very short time:-

$$f = K_n^{-1} F.$$

We are however, still left with the problem: small n $\Rightarrow$ few values of approximation to f; large n $\Rightarrow$ badly ill conditioned K_n. And, although K_n^{-1} is known to any desired accuracy, if F is got empirically from measured values $F + \varepsilon$ where ε is a "noise" vector of random errors, then $\|K_n^{-1}F - K_n^{-1}(F + \varepsilon)\|$ could be very large. But if we have reason to expect f to be "smooth" then this can be taken account of as an extra side condition in solving

$$K_n f = F.$$

BKL do this for the general problem Ax = b, with A ill-conditioned, by minimizing functionals over x such as

$$(Ax - b,\ Ax - b) + \lambda D_n(x)$$

where (.,.) is an inner product and

$$D_n(x) = (x_1 - x_2)^2 + (x_2 - x_3)^2 + \quad + (x_{n-1} - x_n)^2$$

which decreases with increasing "smoothness".

They achieve this minimization with aid of dynamic programming. With λ small, x is then a good approximation to the solution. A similar approach to the selection of the, in some sense, smoothest x from among those which A maps into an ε-neighbourhood of b is given by D.L. Phillips [1] and S. Twomey [4] ; they have achieved encouraging results.

Also on the same topic are papers by A.N. Tihonov [5] &[6] where an error functional of integral form is minimized by the classical method of the Calculus of Variations, reducing the problem to an Euler equation which can be solved numerically by finite differences.

We now describe in full, a projection method based on the paper by Miller and Guy [3] .

The Laplace transform of f(t) is defined by the integral

$$F(s) = \int_0^\infty \exp(-st) \cdot f(t) \cdot dt \quad \left(\mathrm{Re}(s) \geq c > 0.\right) \tag{3.1}$$

We assume below that the integral in (3.1) exists for $\mathrm{Re}(s) > 0$. The variable of integration may be changed by the substitution

$$x = 2 \exp(-\delta t) - 1 \tag{3.2}$$

where δ is a positive real parameter. It follows that

$$\exp(-st) = \left(\frac{1+x}{2}\right)^{s/\delta}$$

and

$$t = -\frac{1}{\delta} \log\left(\frac{1+x}{2}\right)$$

We define g over $(-1,1)$ by

$$g(x) = f\left\{ - \left(\frac{1}{\delta}\right) . \log\left[\frac{1 + x}{2}\right]\right\} = f(t)$$

In order to extend the domain of definition for g, define $g(1)$ and $g(-1)$ by

$$g(1) = \lim_{x \to 1-} g(x) \quad \text{and} \quad g(-1) = \lim_{x \to -1+} g(x)$$

Essentially, these definitions require that $f(0) = \lim_{t \to 0+} f(t)$ and $f(\infty) = \lim_{t \to \infty} f(t)$ be finite. If f is continuous then g is also a continuous function. Substituting (3.2) into (3.1) we get

$$F(s) = 2\frac{1}{\delta} \int_{-1}^{+1} \left(\frac{1 + x}{2}\right)^{\frac{s}{\delta} - 1} . g(x) . dx \qquad (3.3)$$

Now assume that g can be expanded over $[-1, +1]$ in an infinite series of orthogonal polynomials.

A special caseof the Jacobi polynomial of degree n is defined by (see [7] Chapter 22)

$$P_n^{(0,\beta)}(x) = \frac{(-1)^n}{2^n n!} (1 + x)^{-\beta} \frac{d^n}{dx^n} \left[(1 - x)^n (1 + x)^{n+\beta}\right], \quad \beta > -1.$$

where the parameter α which appears in the general definition is zero.

If g can be expanded over $[-1,1]$ in terms of these polynomials then

$$g(x) = \sum_{n=0}^{\infty} C_n P_n^{(0,\beta)}(x) \tag{3.4}$$

substituting (3.4) in (3.3) we get

$$F(s) = \frac{1}{2\delta} \int_{-1}^{+1} \left(\frac{1+x}{2}\right)^{\frac{s}{\delta}-1} \left[\sum_{n=0}^{\infty} C_n P_n^{(0,\beta)}(x) \right] dx$$

Now put $s = (\beta + 1 + k)\delta$, where $k \geq 0$ is an integer to get

$$\delta F[(\beta + 1 + k)\delta] = 2^{-(\beta + k + 1)} \int_{-1}^{+1} (1+x)^{\beta+k} \left[\sum_{n=0}^{\infty} C_n P_n^{(0,\beta)}(x) \right] dx \tag{3.5}$$

Now the factor $(1 + x)^k$ which appears in (3.5) may be expressed as a finite linear combination of the Jacobi polynomials

$$P_0^{(0,\beta)}, \ldots, P_k^{(0,\beta)}.$$

Thus

$$(1 + x)^k = a_0 P_0^{(0,\beta)}(x) + a_1 P_1^{(0,\beta)}(x) + \ldots + a_k P_k^{(0,\beta)}(x) \tag{3.6}$$

For $0 \leq m \leq k$ a typical coefficient a_m is a function of k and β. In order to evaluate a_m, multiply both sides of (3.6) by

$$(1 + x)^{\beta} P_m^{(0,\beta)}(x) \quad \text{and integrate over } [-1,1].$$

$$\int_{-1}^{+1} (1+x)^{k} (1+x)^{\beta} P_m^{(0,\beta)}(x) \, dx = a_m \frac{2^{\beta+1}}{(2m+\beta+1)} \tag{3.7}$$

The factor $2^{\beta+1}/(2m+\beta+1)$ on the right is the normalization term for the Jacobi polynomials. Denote this by h_m. Now the Jacobi polynomial $P_m^{(0,\beta)}(x)$ can be expressed in the form

$$P_m^{(0,\beta)}(x) = b_0 + b_1(1+x) + \ldots + b_m(1+x)^m \tag{3.8}$$

Substituting $P_m^{(0,\beta)}(x)$ in this form into (3.7) yields

$$a_m h_m = \int_{-1}^{+1} (1+x)^{k+\beta} \left[b_0 + b_1 (1+x) + \ldots + b_m (1+x)^m \right] dx$$

$$= b_0 \frac{2^{k+\beta+1}}{(k+\beta+1)} + b_1 \frac{2^{k+\beta+2}}{(k+\beta+2)} + \ldots + b_m \frac{2^{k+\beta+m+1}}{(k+\beta+m+1)}$$

$$= 2^{k+\beta+1} \left[\frac{b_0}{k+\beta+1} + \frac{2b_1}{(k+\beta+2)} + \ldots + \frac{2^m b_m}{(k+\beta+m+1)} \right] \tag{3.9}$$

If the unknown a_m is considered as a function of the parameter k then we may write

$$a_m h_m = \frac{2^{k+\beta+1} \cdot \Omega_m(k)}{[k+(\beta+1)][k+(\beta+2)] \ldots [k+(\beta+m+1)]} \tag{3.10}$$

Where $Q_m(k)$ is a polynomial in the symbol "k" of degree m. The explicit expression for $Q_m(k)$ may be determined by the use of (3.6) and (3.7). In (3.7) let k = m - 1. Then, because of the orthogonality property of the Jacobi polynomials

$$\int_{-1}^{+1} (1 + x)^{m-1}(1 + x)^{\beta} P_m^{(0,\beta)}(x) \cdot dx = 0.$$

Thus one of the roots of $Q_m(k)$ must be k = m - 1. A similar procedure shows that the remaining roots of $Q_m(k)$ are k = m - 2, m - 3, ... , 1, 0. So $Q_m(k)$ is now known up to a constant multiplicative factor and may be written as

$$Q_m(k) = A\,[k-(m-1)]\cdot[k-(m-2)] \ \ldots \cdot k$$

and A is a constant to be determined. From the definition of $Q_m(k)$ in (3.10) from (3.9) we have, multiplying through by $(k+\beta+1)$ ... $(k+\beta+m+1)$,

$$Q_m(k) = (k+\beta+1) \ldots (k+\beta+m+1)\left[\frac{b_0}{k+\beta+1} + \ldots + \frac{2^m b_m}{k+\beta+m+1}\right]$$

and the coefficient of of k^m on the left is A and on the right is $b_0 + 2b_1 + 2^2 b_2 + \ldots + 2^m b_m$. But from (3.8)and the property $P_m^{(0,\beta)}(1) = 1$ for m = 0,1,2, ... we have

$$P_m^{(0,\beta)}(1) \equiv b_0 + 2b_1 + \ldots 2^m b_m = 1$$

Hence A = 1 for m = 0,1,2, ...

Thus from (3.9)

$$a_m = 2^k(2m + \beta + 1)\,\frac{k(k-1)\cdots(k-(m-1))}{(k+\beta+1)(k+\beta+2)\cdots(k+\beta+m+1)} \qquad (3.11)$$

For $k = 0$ and $m = 0$ the right side of (3.11) is replaced by 1.

Substituting (3.6) into (3.5) gives

$$F[(\beta + k + 1)\delta] = \frac{2^{-(\beta+k+1)}}{\delta} \cdot \int_{-1}^{+1} (1+x)^{\beta} \sum_{m=0}^{k} a_m P_m^{(0,\beta)}(x) \cdot$$

$$\cdot \left[\sum_{n=0}^{\infty} C_n P_n^{(0,\beta)}(x) \right] dx \qquad (3.12)$$

for $k = 0,1, \ldots$ where a_m is given by (3.11).

Integrating termwise in (3.12) gives $k + 1$ non-zero terms because of the orthogonality property of the Jacobi polynomials. Substituting for a_m we now get

$$\delta F[(\beta + 1 + k)\delta] = \frac{C_0}{(\beta + 1 + k)} + \sum_{m=1}^{k} \frac{k(k-1)\cdots[k-(m-1)]}{(k+\beta+1)(k+\beta+2)\cdots(k+\beta+1+m)} \cdot C_m$$

(3.13)

Now allowing k to take the values $0,1,2,\ldots$ we get the following system of equations:

$$\delta F\left[(\beta+1)\delta\right] = \frac{C_0}{(\beta+1)}$$

$$\delta F\left[(\beta+2)\delta\right] = \frac{C_0}{(\beta+2)} + \frac{C_1}{(\beta+2)(\beta+3)} \qquad (3.14)$$

$$\delta F\left[(\beta+3)\delta\right] = \frac{C_0}{(\beta+3)} + \frac{2C_1}{(\beta+3)(\beta+4)} + \frac{2C_2}{(\beta+3)(\beta+4)(\beta+5)}$$

$$\delta F\left[(\beta+4)\delta\right] = \frac{C_0}{(\beta+4)} + \frac{3C_1}{(\beta+4)(\beta+5)} + \frac{3\cdot 2C_2}{(\beta+4)(\beta+5)(\beta+6)}$$

$$+ \frac{3!\ .\ C_3}{(\beta+4)(\beta+5)(\beta+6)(\beta+7)}$$

If we know F on the positive real axis then this system can be solved for $C_0, C_1, \ldots$ very easily by successive back substitution: determining C_0 then with the knowledge of C_0 determining C_1 and so on.

If $C_0, C_1 \ldots, C_N$ are calculated then $g(x)$ may be approximated by

$$g(x) \simeq \sum_{n=0}^{N} C_n P_n^{(0,\beta)}(x)$$

since $x = 2\exp(-\delta t) - 1$ we can express the approximation to $f(t)$ directly

$$f(t) \simeq \sum_{n=0}^{N} C_n P_n^{(0,\beta)}\left[2\exp(-\delta t) - 1\right]$$

Now if we had an infinite-length computer (a perfect computer) then as we calculated more and more coefficients our approximation to f would get better and better and, although, logically, we could never know how good this approximation was in practice, for the sort of functions met in applications this would not matter. Not having such a computer we are limited to a finite number of the coefficients, after which rounding error builds up. With a computer working in 12 decimal floating point arithmetic approximately the first 10 or 12 coefficients are significant.

The points on the positive real axis at which F(s) is evaluated depend on the real parameters β and δ which we can vary within the constraints $\beta > -1$, $\delta > 0$.

Theoretically, again with a perfect computer, the Fourier-Jacobi expansions derived from an exactly known F for different values of β and δ would all yield the same function f. With real computers and, sometimes, an approximately known F we will generate as many different truncated approximations to f as values (β, δ) we take. Our task is now to choose the optimum (β, δ) pair.

From the Tauberian results

$$\lim_{s \to 0} sF(s) = f(\infty)$$

$$\lim_{s \to \infty} sF(s) = f(0)$$

(see [8] page 243 (37.5), (37.6) with $\gamma = 1$)

We see that for large t, f is determinedby the behaviour of F for small s on the positive real axis, and vice versa. Thus one guide

to the best choice of β and δ is the particular t-domain of f in which we are interested: If we are interested in f for small values of t we choose β and δ large, and so on.

Another important principle to follow in the choice of β and δ has already been expounded in connection with the BKL method, namely if we have reason to believe that f is smooth then we should choose β and δ to yield the smoothest function.

Once a pair (β, δ) is chosen, the ALGOL program determines the set of coefficients $\{C_0 \ldots C_n\}$ denoted $\{C_i\}_{\beta,\delta}$

$$\text{Now} \quad g(x) = \sum_{i=0}^{n} C_i P_i^{(0,\beta)}(x) + \sum_{i=n+1}^{\infty} C_i P_i^{(0,\beta)}(x)$$

and the Chebyshev norm of the truncation error is

$$\|g - \sum_{0}^{n} C_i P_i^{(0,\beta)}\|_T = \sup_{x\in[-1,1]} \left| g(x) - \sum_{0}^{n} C_i P_i^{(0,\beta)}(x) \right|$$

$$= \sup_{x\in[-1,1]} \left| \sum_{n+1}^{\infty} C_i P_i^{(0,\beta)}(x) \right| =$$

$$= \left\| \sum_{n+1} C_i P_i^{(0,\beta)} \right\| \qquad (3.15)$$

Let us define $\mathcal{E}_n(g;\beta,\delta) = \|g - \sum_{i=0}^{n} C_i P_i^{(0,\beta)}\|_T$.

Theoretically (3.15) allows us to minimize $\mathcal{E}_n(g;\beta,\delta)$ over some

allowed region of the β , δ -plane without any knowledge of g: we merely have to minimize

$$\left\| \sum_{n+1}^{\infty} c_i P_i^{(0,\beta)} \right\| \text{ over } \{ c_i \}_{\beta,\delta} .$$

In practice we can minimize

$$\left\| \sum_{n+1}^{n+k} c_i P_i^{(0,\beta)} \right\| \quad \text{for some finite } k.$$

The fact that this truncation and minimization procedure tends to cut down the "high-frequency" component of the Fourier-Jacobi development, fits in with our intention to choose the smoothest of the available approximations to the unknown f.

Numerical procedure and the results of applying the method to the transforms of some known functions

The basic program written to carry out the MG method of inversion of a given transform F has the structure indicated in Fig.1.

The sub-routine 3 evaluates the error estimate by finding

$$\max_{\{x_i\}} \left| \sum_{n+1}^{n+k} C_i \, P_i^{(0,\beta)}(x) \right| \qquad \text{over at first 8 values } \{x_{1i}\}$$

of x in the interval of interest, then over these and the interlacing equidistant $\{x_{2i}\}$ (giving 16 $\{x_i\}$ in total), then over the 16 $\{x_{3i}\}$ interlacing these points and so on until the successive maxima are within a given tolerance of one another.

The method of finding the optimum (β, δ) is quite unsophisticated. Good results can be had by making the grid of sample points very fine but then usually a vast amount of machine time is spent in looking

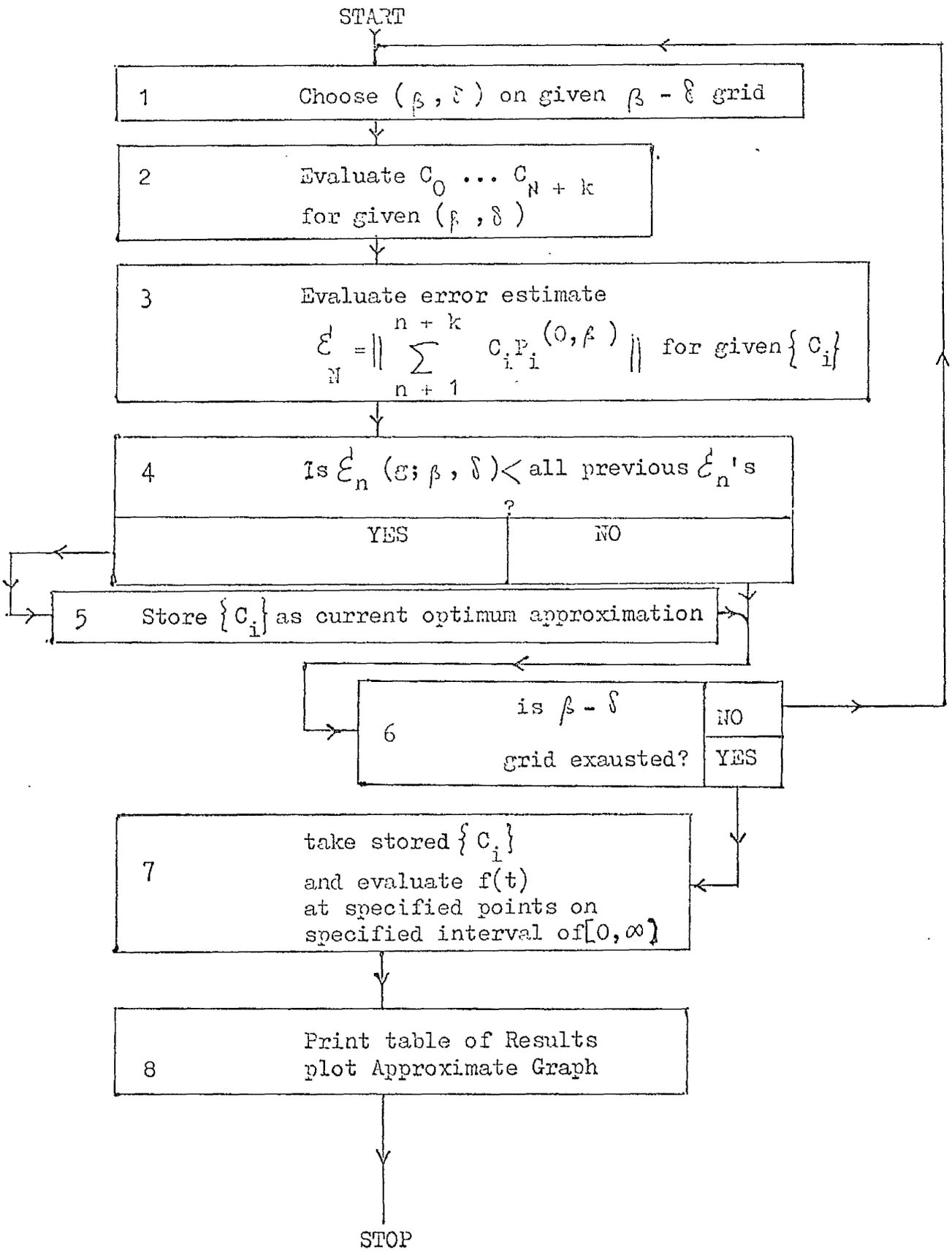

START
1
Choose (β , δ) on given β - δ grid
2
Evaluate C0 ... CN + k
for given (β , δ)
3
Evaluate error estimate
$\mathcal{E}_N = \| \sum_{n+1}^{n+k} C_i P_i(0, \beta) \|$ for given { Ci }
4
Is $\mathcal{E}_n(\sigma; \beta, \delta) <$ all previous $\mathcal{E}_n$'s ?
YES
NO
5
Store { Ci } as current optimum approximation
6
is β - δ grid exausted?
NO
YES
7
take stored { Ci }
and evaluate f(t)
at specified points on
specified interval of [0, ∞)
8
Print table of Results
plot Approximate Graph
STOP

FIG.1.

at areas of the $\beta - \delta$ plane that are far from the actual optimum. The problem we are faced with is the common one of finding the minimum of a multi-variable function of complicated or analytically unknown form. In our case the 2-variable function $\mathcal{E}'_n(g; \beta, \delta)$ can be evaluated at an individual point but it is not known analytically. To evaluate $\mathcal{E}'_n(g; \beta, \delta)$ at a large number of points is costly in computing time. However, with the examples of f (and thus g) considered, $\mathcal{E}'_n(g; \beta, \delta)$ seemed reasonably well behaved suggesting the use of a more sophisticated minimization procedure which could converge on the optimum (β, δ) much more quickly. One such procedure is the "Method of Steepest Descent" ([9] Ch.2) but this requires the evaluation of the gradient of $\mathcal{E}'_n(g; \beta, \delta)$ with respect to β, δ and this computation would both consume time and add error. A more promising possibility is to use one of the several proposed methods of function minimization without evaluating derivatives ([10], [11], [12]). The hope is that one of the methods would speed up and improve the accuracy of the program.

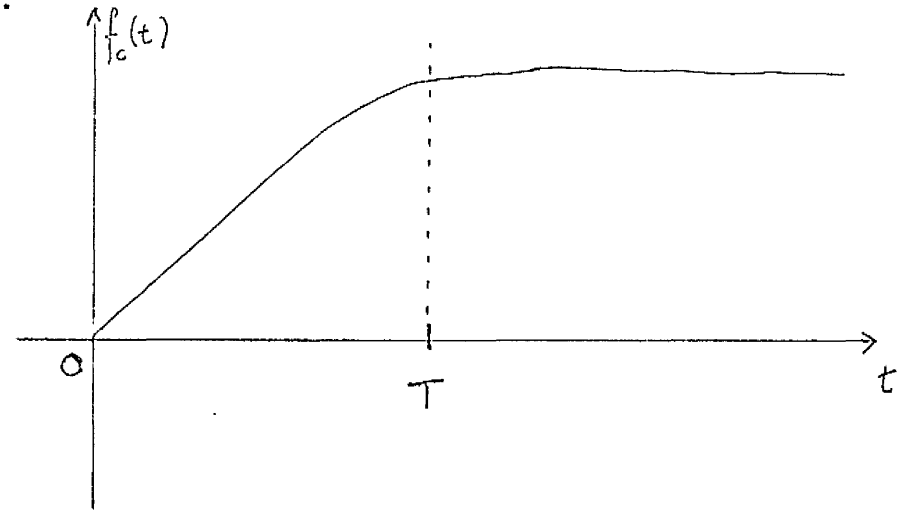
The program as written uses the basic program outlined above, to evaluate the approximation

$$\sum_{i=0}^{n} C_i P_i^{(0,\beta)}(x)$$

with optimum values of (β, δ) for values of n over a specified range.

The question of which n is the best one is a difficult one. For the functions e^{-t} and $J_0(t)$, with Laplace transforms $\frac{1}{s+1}$ and $\frac{1}{\sqrt{s^2+1}}$ respectively, the optimum value of n was around 4 for e^{-t} and around 10 for $J_0(t)$. A rough guide may be: small n for smooth f and large n for less smooth f. By large is meant large, but not so large that rounding-off error causes the higher coefficients to start growing. In fact another guide for optimizing with respect to n is, possibly, to stop when the highest coefficients begin to grow again after their initial decrease.

The best numerical results obtained for the test functions e^{-t} and $J_0(t)$ were for example, 8 decimal place accuracy over $[0,5]$ for e^{-t} and 4 decimalplace accuracy for $J_0(t)$ over the same range. These results are much better than those obtained by other published results as pointed out by Miller and Guy. In their paper [3] they give graphs of their results using this method plotted together with the results obtained by other methods. Another point made by Miller and Guy is that although the theory of the method is developed for functions f(t) finite at 0 and ∞, tests with $F(s) = \frac{1}{s^2}$, which has the inverse transform $f(t) = t$, yield the following type of result:



The algorithm achieves the result $f_c(t) = t$ for $t \in [0,T]$ but thereafter the computed function $f_c(t)$ settles down to a constant value T. Thus, approximately

$$f_c(t) = t - (t - T) \cdot H(t - T)$$

and $$L[f_c;\ t \geqslant s] = \frac{1}{s^2} - \frac{\exp(-Ts)}{s^2}$$

So for large T $$\frac{1}{s^2} \simeq \frac{1}{s^2} - \frac{\exp(-Ts)}{s^2}$$

and in the finite length computer arithmetic these two functions will be indistinguishable. But the algorithm has selected the one which is finite at ∞ . Another case in which, while the theory does not strictly apply, the algorithm gives good results is the Dirac delta function which has the Laplace transform $F(s) = 1$. Of course what the algorithm produces is an approximation to the behaviour of this generalized function. The more coefficients C_i that are calculated, the higher the peak at $t = 0$ becomes. Also this is a good example of a case when our policy of minimizing the "high frequency" component of the Fourier-Jacobi expansion is not a good one, any approximation to $\delta(t)$ being far from smooth at the origin. Nevertheless, once we have got some results from a first application of the algorithm we can use this rough description of the unknown f to govern our further application of the algorithm.

Advantages of the MG method

One immediate advantage would seem to be the superior accuracy of this method to that of other published methods (see [13], [14] and the bibliography in [14]) although an exhaustive comparitive study has not, as far as we know, been undertaken. A major advantage of the MG over the BKL method is the fact that the MG method gives the approximation to f(t) effectively as a polynomial which can be evaluated at any required value of t without any difficulty. The BKL method, on the other hand, only gives a set of points and since they are logorithms of the zeros of the shifted legendre polynomials they are awkwardly situated and irregularly spaced. Also in the MG method there is no need to store any matrices to perform the inversion.

A Possible Algorithm for the Mellin Transform

The essential feature of the MG method is the observation that once the defining integral has been put in the form

$$F(s) = \frac{1}{2\delta} \cdot \int_{-1}^{+1} \left(\frac{1+x}{2} \right)^{\frac{s}{\delta}-1} g(x)\, dx$$

a factor $(1+x)^{\beta}$ can be got in the integrand by choosing s appropriately and that this is a weighting function for the Jacobi polynomials $P_n^{(0,\beta)}(x)$. Now, another integral transform with important applications is the Mellin transform.

$$F(p) = \int_0^{\infty} x^{p-1} f(x)\, dx \qquad (6.1)$$

or putting $x = e^{-t}$ we get the two-side Laplace transform

$$F(p) = \int_{-\infty}^{+\infty} e^{-pt} f(e^{-t})\, dt \qquad (6.2)$$

Looking at (6.1), the factor x^{p-1} suggests the use of the Generalized Laguerre polynomials $L_n^{(\alpha)}(x)$, which are orthogonal on $[0, \infty)$ with weight function $w(x)$.(ψ) Put $f(x) = e^{-x} g(x)$ and assume that $g(x)$ can be expanded in terms of the generalized Laguerre polynomials

$$g(x) = \sum_{n=0}^{\infty} C_n(x) \cdot L_n^{(\alpha)}(x) \qquad (\alpha > -1) \qquad (6.3)$$

(ψ) $w(x) = e^{-x} x^{\alpha}$ $(\alpha > -1)$ see [7] section 22.2

where $C_n(\alpha)$ is the nth Fourier-Laguerre coefficient depending on the choice of α.

Rodrigues' Formula for the generalized Laguerre polynomials is ([7] 22.11)

$$L_n^{(\alpha)}(x) = \frac{1}{n!\, e^{-x}\, x^{\alpha}} \frac{d^n}{dx^n} \left\{ e^{-x}\, x^{\alpha}\, x^n \right\} \tag{6.4}$$

Substituting (6.3) in (6.1) and then (6.4) in to the result

$$F(p) = \int_0^{\infty} x^{p-1}\, e^{-x} \sum_{n=0}^{\infty} C_n(\alpha)\, L_n^{(\alpha)}(x)\, dx$$

$$= \int_0^{\infty} x^{p-1}\, e^{-x} \sum_{n=0}^{\infty} C_n(\,)\, \frac{1}{n!}\, e^{x}\, x^{-\alpha} \frac{d^n}{dx^n} \left(e^{-x} x^{\alpha+n}\right) dx \tag{6.5}$$

and now the exponential factor e^{-x}, which remains in the partial sums after simplification, ensures the validity of the interchange of the order of integration and summation since $\exp\left(-\frac{x}{2}\right)$ dominates (ultimately) $x^{\alpha}\, e^{-x}$ and the Lebesgue dominated convergence theorem gives the result

$$F(p) = \sum_{n=0}^{\infty} \frac{1}{n!}\, C_n(\alpha) \int_0^{\infty} x^{p-1-\alpha} \frac{d^n}{dx^n} \left(e^{-x}\, x^{\alpha+n}\right) dx \tag{6.6}$$

Put $p = \alpha + 1 + k$ where k is a positive integer i.e. choosing k and α such that $\alpha = p - (k+1)$ and $k < p$.

Then (6.6) gives

$$F(\alpha + k + 1) = \sum_{n=0}^{\infty} \frac{1}{n!} C_n(\alpha) \int_0^{\infty} x^k \frac{d^n}{dx^n} \left(e^{-x} x^{p-(k+1)+n}\right) dx$$

Now define $I_{k,n} = \int_0^{\infty} x^k \frac{d^n}{dx^n} \left(e^{-x} x^{p+n-k-1}\right) dx$

$$= \left[x^k \left(e^{-x} x^{p+n-k-1}\right)^{(n-1)} \right]_0^{\infty}$$

$$- k \int_0^{\infty} x^{k-1} \left(e^{-x} x^{p+n-k-1}\right)^{(n-1)} dx \qquad (6.7)$$

The first term in (6.7) is zero so

$$I_{k,n} = -k \int_0^{\infty} x^{k-1} \left(e^{-x} x^{p+n-k-1}\right)^{(n-1)} dx$$

$$I_{k,n} = -k \; I_{k-1,\, n-1}$$

For $n > k$ $I_{0,n-k} = \int_0^{\infty} \frac{d^{n-k}}{dx^{n-k}} \left(e^{-x} x^{p+n-k-1}\right) dx$

$$= \left[\frac{d^{n-k-1}}{dx^{n-k-1}} \left(e^{-x} x^{p+n-k-1}\right) \right]_0^{\infty}$$

$$= \left[(-1)^{n-k-1} \; e^{-x} \sum_{r=0}^{n-k-1} \binom{n-k-1}{r} \; (-1)^r \mu \ldots (\mu-r+1) \; x^{\mu-r} \right]_0^\infty \qquad (6.9)$$

where $\mu = p+n-k-1$, by Leibniz' formula

But $\mu - (n-k-1) = p > 0$ so the value at $x = 0$ of the expression inside (6.9) is zero and the exponential e^{-x} ensures that it is zero at ∞ as well.

Hence $I_{0,n-k} = 0 \qquad (n > k)$

and thus $I_{n,k} = 0$ for $n > k$ by (6.8)

For $n \leq k$ $\quad I_{k-n,0} = \int_0^\infty x^{k-n} \; e^{-x} \; x^{p+n-k-1} \; dx \qquad (6.10)$

$$= \int_0^\infty e^{-x} \; x^{p-1} \; dx$$

$$I_{k-n,0} = \Gamma(p) \qquad (6.11)$$

and for $n \leq k$ $\quad I_{k,n} = (-1)^n \; \frac{k!}{(k-n)!} \; I_{k,0} \qquad (6.12)$

which follows from the recurrence relation (6.8).

So by (6.10) and (6.11)

$$\left.\begin{aligned} I_{k,n} &= (-1)^n \; \frac{k!}{(k-n)!} \; \Gamma(p) \qquad (n \leq k) \\ &= 0 \qquad\qquad\qquad\quad (n > k) \end{aligned}\right\} \qquad (6.13)$$

Hence recalling the definition of ϕ

$$F(\alpha+k+1) = \sum_{n=0}^{\infty} \frac{1}{n!} C_n(\alpha) I_{k,n}$$

$$= \sum_{n=0}^{k} \frac{1}{n!} C_n(\alpha) \cdot (-1)^n \frac{k!}{(k-n)!} \Gamma(\phi).$$

$$\text{i.e.} \quad \frac{F(\alpha+k+1)}{\Gamma(\alpha+k+1)} = \sum_{n=0}^{k} (-1)^n \binom{k}{n} C_n(\alpha) \tag{6.14}$$

Put $G_k = F(\alpha+k+1) / \Gamma(\alpha+k+1)$ and $C_n = C_n(\alpha)$; then letting k assume the values $0,1,2,\ldots$ (6.14) gives a linear system of equations for the coefficients C_n

$$\begin{aligned} G_0 &= C_0 \\ G_1 &= C_0 - C_1 \\ G_2 &= C_0 - 2C_1 + C_2 \\ G_3 &= C_0 - 3C_1 + 3C_2 - C_3 \\ &\cdot \quad \cdot \quad \cdot \\ &\cdot \quad \cdot \quad \cdot \end{aligned} \tag{6.15}$$

Solving (6.15) we get

$$\begin{aligned} C_0 &= G_0 \\ C_1 &= G_0 - G_1 \\ C_2 &= G_0 - 2G_1 + G_2 \\ C_3 &= G_0 - 3G_1 + 3G_2 - G_3 \end{aligned} \tag{6.16}$$

which suggests the general solution $C_k = \sum_{j=1}^{k} (-1)^j \binom{k}{j} G_j$ (6.17)

We show this by induction

Suppose (6.17) holds for r = 0,1,2,... k-1, k

Now by (6.14)

$$G_{k+1} = \sum_{r=0}^{k+1} (-1)^r \binom{k+1}{r} C_r$$

so $(-1)^{k+1} C_{k+1} = G_{k+1} - \sum_{r=0}^{k} (-1)^r \binom{k+1}{r} C_r$

$$C_{k+1} = (-1)^{k+1} G_{k+1} - (-1)^{k+1}\left[\sum_{r=0}^{k} (-1)^r \binom{k+1}{r} \sum_{j=0}^{r} (-1)^j \binom{r}{j} G_j \right]$$

$$C_{k+1} = (-1)^{k+1} G_{k+1} -(-1)^{k+1}\left[\sum_{r=0}^{k} \sum_{j=0}^{k}{}' (-1) \binom{r}{j} G_j (-1)^r \binom{k+1}{r} \right]$$

where $\sum'$ indicates that the summand depending on j is zero for $j > r$.

$$C_{k+1} = (-1)^{k+1} G_{k+1} -(-1)^{k+1}\left[\sum_{j=0}^{k} (-1)^j G_j \sum_{r=j}^{k} (-1)^r \binom{k+1}{r} \binom{r}{j} \right] \quad (6.18)$$

$$\binom{k+1}{r}\binom{r}{j} = \frac{(k+1)!}{r!(k+1-r)!}\,\frac{r!}{j(r-j)!} = \frac{(k+1)!}{j!(k+1-j)!}\,\frac{(k+1-j)!}{(r-j)!\,(k+1-j-(r-j))!}$$

$$= \binom{k+1}{j}\binom{k+1-j}{r-j} = \binom{k+1}{j}\cdot\binom{k+1-j}{k+1-r} \tag{6.19}$$

Using (6.19) in (6.18)

$$C_{k+1} = (-1)^{k+1} G_{k+1} - \sum_{j=0}^{k} (-1)^j G_j \binom{k+1}{j} \sum_{r=j}^{k} \binom{k+1-j}{k+1-r} (-1)^{k+1-r}$$

$$\text{But} \sum_{r=j}^{k+1} \binom{k+1-j}{k+1-r} (-1)^{k+1-r} = \sum_{\rho=k+1-j}^{\rho=0} \binom{k+1-j}{\rho} (-1)^{\rho} \tag{6.20}$$

where $\rho = k+1-r$. But the expression on the right of (6.20) is the binomial expansion of $(1-1)^{k+1-j} = 0$.

$$\text{Hence} \sum_{r=j}^{k} \binom{k+1-j}{k+1-r} (-1)^{k+1-r} = -\binom{k+1-j}{0} (-1)^{\rho} = -1$$

$$\text{so } C_{k+1} = (-1)^{k+1} G_{k+1} + \sum_{j=0}^{k} \binom{k+1}{j} (-1)^j G_j$$

$$\text{that is } C_{k+1} = \sum_{j=0}^{k+1} \binom{k+1}{j} (-1)^j G_j \tag{6.21}$$

and this demonstrates, by induction, the truth of (6.17).

Thus we have an algorithm for the Mellin Transform which is similar to Miller & Guy's

Fourier-Jacobi algorithm for the Laplace transform. We would hope that the parameter α could be chosen to select a smooth solution in much the same way as we chose (β, δ). One slight advantage of this Fourier-Laguerre algorithm is that we have an explicit analytic expression (6.21) for the coefficients $C_n(\alpha)$ which may speed up the computation. On the other hand we have only one parameter α to vary. This algorithm has yet to be given a practical test.

Appendix 1

Solution of Inverse Problems Using the Laplace Transform

Many problems in mathematics are what might be called direct problems in the sense that once we have set up a mathematical model we are interested in seeking this model's implications and in applied mathematics, interpreting these in the physical domain. But many important problems need a different approach: a certain physical situation may be approximately representable by a whole range of models varying with parameters they incorporate. The problem now, is, given some data (which would correspond to the predictions of these models), to decide which is the (in some sense) best model to fit this data. This is an inverse problem.

For example we may have a system the behaviour of which can be described by a transfer function $k(t)$ which determines the output function $u(t)$ of the system given an input function $f(t)$, in the relation

$$u(t) = \int_0^t k(t-x) \; f(x) \; dx \qquad (A1)$$

Using the Laplace transform we can write

$$Lk(s) = \frac{Lu(s)}{Lf(s)} \qquad (A2)$$

The calculation of Lu(s)/Lf(s) involves two straight forward quadratures on $[0, \infty)$. Thus a subroutine to determine Lk(s) from u(t) and f(t) can be written, inserted in the Laplace transform inversion algorithm and the Fourier-Jacobi expansion of an approximation to k(t) obtained. See [2] Ch. 3. Sections 21-24 for a discussion on estimation of system constants.

It is possible that this approach may be useful in determining the functions $\chi(t)$, $\varphi(t)$ which relate stress and strain in 1-dimensional viscoelastic materials through the relations

$$\epsilon(t) = \int_{-\infty}^{t} \chi(t-\tau) \cdot \frac{d\sigma(\tau)}{d\tau} \cdot d\tau$$

$$\sigma(t) = \int_{-\infty}^{t} \varphi(t-\tau) \cdot \frac{d\epsilon(\tau)}{d\tau} \cdot d\tau \qquad \text{(A3)}$$

If it is assumed that $\sigma = \epsilon = 0$ for $t < 0$

Then equations (A3) reduce to two convolution equations

$$\epsilon(t) = \int_{0}^{t} \chi(t-\tau) \quad \frac{d\sigma(\tau)}{d\tau} \; d\tau$$

$$\sigma(t) = \int_{0}^{t} \varphi(t-\tau) \; \frac{d\epsilon(\tau)}{d\tau} \cdot d\tau$$

transforming:

$$\bar{\epsilon}(s) = \bar{\chi}(s) \cdot \left[s\bar{\sigma}(s) - \sigma(0+) \right] = \bar{\chi}(s).s\,\bar{\sigma}(s)$$

$$\bar{\sigma}(s) = \bar{\varphi}(s) \cdot \left[s\bar{\epsilon}(s) - \epsilon(0+) \right] = \bar{\varphi}(s).s\,\bar{\epsilon}(s)$$

so $$\bar{\chi}(s) = \frac{1}{s} \cdot \frac{\bar{\epsilon}(s)}{\bar{\sigma}(s)}$$

$$\bar{\varphi}(s) = \frac{1}{s} \cdot \frac{\bar{\sigma}(s)}{\bar{\epsilon}(s)}$$

and we can determine $\chi(t)$, $\varphi(t)$, in much the same way as the transfer function $k(t)$ in (A1), providing we know $\epsilon(t)$ and $\sigma(t)$ on $[0,\infty)$ Another method to determine χ and φ using the Laplace transform is given in [16] .

REFERENCES

[1] D.L. PHILLIPS, "A technique for the Numerical Solution of certain Integral Equations of the First Kind".
J. Assoc. Computing Machinery Vol.9. 1962 pp. 84-97.

[2] R.E. BELLMAN, R.E. KALABA, & J. LOCKETT, "Numerical Inversion of the Laplace Transform: Applications to Biology, Economics Engineering and Physics". American Elsevier. New York 1966

[3] M.K. MILLER & W.T. GUY, JR. "Numerical Inversion of the Laplace Transform by use of Jacobi Polynomials".
SIAM. J. Numer. Anal. Vol3, No.4 1966 pp 624-635.

[4] S. TWOMEY "On the Numerical Solution of Fredholm Integral Equations of the First Kind by the Inversion of the Linear System Produced by Quadrature"
J.Assoc. Computing Machinery, Vol.10. 1963. pp 97-101

[5] & [6] A.N. TIHONOV "Solution of Incorrectly Formulated Problems and the Regularization Method" &, "Regularization of Incorrectly Posed Problems".
Soviet Maths. Vol.4 1964. pp 1035-1038, 1624-1627.

[7] HANDBOOK OF MATHEMATICAL FUNCTIONS edited by M. Abramowitz & I.A. Stegun. National Bureau of Standards 1964 and Dover 1965.

[8] I.N. SNEDDON "Functional Analysis" In "Handbuch der Physik" Band II Springer, Berlin, 1955.

[9] T.L. SAATY & J. BRAM "Nonlinear Mathematics", Mcgraw - Hill, New York 1964.

[10] M.J.D. POWELL "Minimization of Functions of Several Variables" Ch.8 in "Numerical Analysis: An Introduction". Edited by J. Walsh, Academic Press, London 1966.

[11] M.J.D. POWELL "An efficient method for finding the minimum of a function of several variables without calculating derivatives." Comp.J. Vol.7 pp. 155-162.

[12] R. HOOKE & T.A. JEEVES " "Direct Search" Solution of Numerical and statistical Problems". J.Assoc. Computing Machinery Vol.8 pp. 212-229.

[13] A. PAPOULIS "A New Method of Inversion of the Laplace Transform." Quart. Appl. Math. Vol. 14. 1957. pp 405-414.

[14] H.V. NORDEN "Numerical Inversion of the Laplace Transform" Acad. Abo. Ser B Vol. 22. 1961 pp 3-31.

[15] D.S. BERRY & S.C. HUNTER "The Propagation of Dynamic Stresses in Visco Elastic Rods" Mech. Phys. Solids Vol.4. 1956 pp. 72-95.

[16] E. H. LEE & C. J. MARTIN. "A Technique for the Formulation of Meaningful Viscoelastic Constitutive Equations."
Avco Corporation, Lowell, Massachusetts.

```
→ESTABLISH DD004JR00KP4+K080007APSP;
JACOBI INVERSN EXPN;
O/PL,8→
begin library A0, A6, A12; integer n, n1, n2, N, ENLO, ENUP, M, KK, f1, f2; KK:= 0;  open(20); N:= read(20);
begin array C[0:N];  real b, d, dlo, dd, dup, blo, db, bup, TL, TU; boolean more data;

      real procedure F(s); value s; real s;
            begin F:= 1/s end F;



      procedure JCOEFFSP(F, delta, beta, print, N, C); value delta, beta, N; real procedure F;
            array C; real delta, beta; integer N; boolean print; comment JCOEFFS calculates the Jacobi
            polynomial coefficients C[0] to C[N];
            begin integer f1, j, k; real q, p;
            real procedure SIGMA(K); value K; integer K; comment sum from 0 to K - 1;
                  begin integer m; real s, q; s:= 0;  q:= K + beta + 1;
                  for m:= 0 step 1 until K - 1 do
                  s:= ((q + m + 1)/(K - m))×(C[m] + s);
                  SIGMA:= s
                  end SIGMA;
            f1:= format([nd]);
            if print then write text(70, [[5c24s]coefficients*c(k)[5c]]);
            q:= beta + 1;  C[0]:= delta×F(q×delta)×q;
            if print then begin write text(70,[k*=**0[15s]]); output(70,C[0]); end;
            for k:= 1 step 1 until N do
            begin q:= k + beta + 1; p:= 1; for j:= 1 step 1 until k do begin p:= p×((q + j)/j) end;
                  C[k]:= (p×q×delta×F(q×delta)) - SIGMA(k);
                  if print then begin write text(70,[k*=*]); write(70, f1, k); space(70,15); output(70, C[k])
                              end;
```

```
      end
      end JCOEFFSP;


procedure JACPOL(beta, N, x, P); value beta, N, x; real beta, x; integer N; array P;
      comment values P(0, beta, n, x) stored in array P for n = 0 to N;
      begin integer n; real A, B, C;  P[0]:= 1; P[1]:= ((beta + 2)xx - beta)/2;
      for n:= 2 step 1 until N do
      begin   A:= 2xnx(n + beta)x(2xn + beta - 2);
            B:= (2xn + beta - 1)x((2xn + beta)x(2xn + beta - 2)xx - (beta↑2));
            C:= 2x(n - 1)x(n + beta - 1)x(2xn + beta);
            P[n]:= (B x P[n - 1] - C x P[n - 2])/A
      end
      end JACPOL;


procedure JEXPN(N, C, delta, beta, tl, tu, M, V); value N, C, tl, tu, M;
      integer N, M; real delta, beta, tl, tu;  array C, V; comment values of approximation to f(t)
      are calculated for t = tl,.....,tu in M equal steps and stored in V[1:M, 2] ;
      begin integer i, j;  real dt, x, s;  array P[0:N];
      dt:= (tu - tl)/M;
      V[1,1]:= tl;  for i:= 2 step 1 until M do V[i,1]:= V[i - 1, 1] + dt;
      for i:= 1 step 1 until M do
      begin  x:= (2xexp(-(deltaxV[i,1]))) - 1;
            JACPOL(beta, N, x, P);  s:= 0;
            for j:= 0 step 1 until N do s:= s + (C[j]xP[j]);
            V[i,2]:= s
      end
      end JEXPN;
```

```
procedure PLOT(D,V, K, M); value D, V, K, M; integer D, K, M; array V; comment D is device
      number for graph output, V holds abscissae and ordinates K is plot number
      and M is number of abscissae;
      begin integer i, j, Y, Y1,   f1, f2, f3; real del, l, u;
      f1:= format([-d.dd␣nd]); f2:= format([-d.ddddsddddsdd␣nd]); f3:= format([nddccc]);
      write text(70, [[pc]plot**number*:*]); write(70, f3, K);
      write text(70, [[30s]x[59s]y[2c]]);
      for i:= 1 step 1 until M do
      begin write text(70, [[20s]]); write(70, f2, V[i,1]);
            write text(70, [[40s]]); write(70, f2, V[i,2]); write text(70, [[c]])
      end;
      l:=u:= V[1,2];
      for i:= 2 step 1 until M do
      begin if V[i,2] < l then l:= V[i,2];  if V[i,2] > u then u:= V[i,2]
      end;
      del:= (u-l)/89;  Y1:= entier((l/del) + 0.5);  write text(D, [[p4s]x[4s]]);
      space(D, 91);  write text(D, [[4s]y[c9s]]);
      for j:= 0 step 1 until 90 do write text(D, [+]); write text(D, [[c]]);
      for i:= 1 step 1 until M do
      begin Y:= entier((V[i,2]/del) + 0.5) - Y1; write(D, f1, V[i,1]);  write text(D,[+]);
            if Y > 0 then begin space(D, Y - 1); write text(D,[+]) end;
            space(D, 89 - Y); write text(D, [+]);
            write(D, f1, V[i,2]); write text(D,[[c]])
      end;
      write text(D, [[9s]]); for j:= 0 step 1 until 90 do write text(D,[+])
      end PLOT procedure;


procedure SUM(N1, N2, x, S);  value N1, N2, x; integer N1, N2; real x, S;
      comment  S = C[N1]P[N1] + ... + C[N2]P[N2] evaluated at x for values
      of beta and C[n] current at call of procedure;
      begin integer i;   array P[0:N2];
            JACPOL(b, N2, x, P);   S:= 0;
            for i:= N1 step 1 until N2 do S:= S + C[i]×P[i];
      end SUM;
```

```
procedure CHEBYNORM(N1, N2, t1, t2, E); value N1, N2, t1, t2; integer N1, N2;
      real t1, t2, E; comment  Chebyshev norm of truncation error on [t1,t2]
      estimated from max(t in [t1,t2]) of abs(C[N1]P[N1]+ ... +C[N2]P[N2]) for
      values of beta and delta current at call of procedure and stored in E;
      begin integer count, i, m, p; real M, S, A, B, D, DD, max;
            max:= 0;  A:= 2×exp(-d×t2) - 1;   B:= 2×exp(-d×t1) - 1;
            D:= B - A;     DD:= 0.125×D;
            JCOEFFSP(F, d, b, false, N2, C);
            for i:= 0 step 1 until 8 do
            begin SUM(N1, N2, A + i×DD, S);   S:= abs(S);
                  if S > max then max:= S
            end;
            p:= 8;   count:= 0;     M:= 1.01×max;      E:= max;
      L11:  DD:= 0.5×DD;   m:= p;   p:= 2×p;
            for i:= 1 step 1 until m do
            begin  SUM(N1, N2, A + (2×i - 1)×DD, S);   S:= abs(S);
                  if S > max then
                  begin if S < M then
                        begin if count < 3 then count:= count + 1
                              else begin  E:= S;  count:= 0;  goto L22 end;
                        end;
                         max:= S;     M:= 1.01×max
                  end;
            end;
            if p < 100 then   goto L11;
      L22:  end CHEBYNORM;
```

```
procedure FINDMIN(n, t1, t2, dlo, dd, dup, blo, db, bup, L, U, optdelta, optbeta, OPTC, MINERROR);
    value n, t1, t2, dlo, dd, dup, blo, db, bup, L, U; integer n, L, U;
    real t1, t2, dlo, dd, dup, blo, db, bup, optdelta, optbeta, MINERROR;
    array OPTC; comment truncation error E for nth order approximation is minimized
    (discrete approximation) over dlo < delta < dup, blo < beta < bup and optimum
    delta, beta, C, and E stored in optdelta, optbeta, OPTC, MINERROR;
    begin integer f, i, j, k; real E, m;    m:= 20;
        f:= format([sd.d#nd]);
        for i:= 0, i+1 while dlo + ixdd ≤ dup + dd do
        begin  d:= dlo + ixdd;
            for j:= 0, j+1 while blo + jxdb ≤ bup + db do
            begin b:= blo + jxdb;   CHEBYNORM(n - L, n + U, t1, t2, E);
                write(70, f, E);
                if E < m then
                begin m:= E;  optdelta:= d; optbeta:= b;
                    for k:= 0 step 1 until n do OPTC[k] := C[k];
                end;
            end;
            newline(70, 1);
        end;
        MINERROR := m;
    end FINDMIN;
```

```
      open(70); write text(70, [T.S.HARRIS]);
      f1:= format([nddc]);       f2:= format([nd]);
data: KK:= KK + 1; write text(70, [[p]result**]); write(70, f1, KK); newline(70, 5);
      write text(70, [N**=**]); write(70, f1, N);
      dlo:= read(20);  write text(70, [dlo*=**]); output(70, dlo);
      dd:= read(20); write text(70, [dd**=**]); output(70, dd);
      dup:= read(20); write text(70, [dup*=**]); output(70, dup);
      blo:= read(20); write text(70, [blo*=**]); output(70, blo);
      db:= read(20); write text(70, [db**=**]); output(70, db);
      bup:= read(20); write text(70, [bup*=**]); output(70, bup);
      M:= read(20); write text(70, [M**=**]); write(70, f1, M);
      n1:= read(20); write text(70, [n1*=**]); write(70, f1, n1);
      n2:= read(20); write text(70, [n2*=**]); write(70, f1, n2);
      ENLO:= read(20); write text(70, [ENLO*=**]); write(70, f1, ENLO);
      ENUP:= read(20); write text(70, [ENUP*=**]); write(70, f1, ENUP);
      TL:= read(20); write text(70, [TL*=**]); output(70, TL);
      TU:= read(20); write text(70, [TU*=**]); output(70, TU);



      begin array V[1:M,1:2], OPTC[0:N]; integer k; real optdelta, optbeta, MINERROR;
            for n:= n1 step 1 until n2 do
            begin write text(70, [[p3c] n*=**]);  write(70, f1, n);  newline(70, 3);
                  FINDMIN(n,TL,TU,dlo,dd,dup,blo,db,bup,ENLO,ENUP,optdelta,optbeta,OPTC,MINERROR);
                  write text(70, [[3c]optdelta*=**]);  output(70, optdelta);
                  write text(70, [optbeta**=**]);  output(70, optbeta);
                  write text(70, [[3c25s]OPTC(k)[3c]]);
                  for k:= 0 step 1 until n do
                  begin  write text(70, [k**=*]); write(70, f2, k); space(70, 10); output(70, OPTC[k])
                  end;
                  write text(70, [[2c] MINERROR*=**]);  output(70, MINERROR);
                  JEXPN(n, OPTC, optdelta, optbeta, TL, TU, M, V);
                  PLOT(70, V, n, M);
            end;
      end;
      more data:= read boolean(20);  if more data then goto data else close(20); close(70);
end program;
end
```