ALGORITHM 682
Talbot's Method for the Laplace Inversion Problem

A. MURLI
University of Naples
and
M. RIZZARDI
University of Lecce

We describe a FORTRAN implementation, and some related problems, of Talbot's method which numerically solves the inversion problem of almost arbitrary Laplace transforms by means of special contour integration.

The basic idea is to take into account computer precision to derive a special contour where integration will be carried out.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.1.2 [**Numerical Analysis**]: Approximation—*nonlinear approximation*; G.1.4 [**Numerical Analysis**]: Quadrature and Numerical Differentiation—*equal interval integration*; G.1.9 [**Numerical Analysis**]: Integral Equations—*Fredholm equations*

General Terms: Algorithms

Additional Key Words and Phrases: Complex inversion formula, inverse Laplace transform, Laplace transform, numerical software, TALBOT, trapezoidal rule

1. INTRODUCTION

The Laplace transform of a function $f(t)$, defined on the interval $[0, \infty)$ and absolutely integrable on any finite interval $[0, a]$, is defined as follows

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt, \quad \mathrm{Re}(s) > \gamma_0$$

where γ_0 is the Laplace transform *abscissa of convergence*. The inverse Laplace problem is that of reconstructing $f(t)$ from known values of $F(s)$.

In this paper we describe Talbot's method for the numerical inversion of the Laplace transform. We provide a software implementation and give some of

Authors' address: Universitá di Napoli, Dipartimento di Matematica e Applicazioni, Via Mezzocannone, 16, cap. 80134, Napoli, Italy.

the results of numerical tests. In particular, in Section 2, a brief description of the underlying theory is given. Section 3 describes the software implementation, its input requirements, and some of the difficulties that a user may encounter. Moreover, we describe how we solved the problem of avoiding overflow in computing the numerical approximation to $f(t)$. In Section 4, some particular problems are discussed and, in Section 5, we present some of the results of numerical comparisons between our software and other available software items.

2. OUTLINE OF THE UNDERLYING THEORY

The Riemann inversion formula gives an integral representation for $f(t)$ in terms of $F(s)$:

$$f(t) = \frac{1}{2\pi i} \oint_{\mathrm{B}} e^{st} F(s)\, ds, \qquad t > 0, \quad i = \sqrt{-1} \tag{2.1}$$

where B is the Bromwich contour from $\gamma - i\infty$ to $\gamma + i\infty$, with $\gamma > \gamma_0$, parallel to the imaginary axis.

This contour is located to the right of all the singularities of $F(s)$. When $f(t)$ is to be calculated using numerical quadrature, it may be convenient to use a different contour. In general, one would like to move the contour to the left so as to reduce in magnitude the factor e^{st} in the integrand, but the contour must not be moved too close to singularities of $F(s)$, as to do so will result in peaks in the integrand function.

A prescription for finding a suitable contour (one which retains a proper balance between the two effects described above) has been provided by Talbot. This requires that the locations of the singularities of $F(s)$ be known, and that $F(s)$ satisfies

(a) $|F(s)| \to 0$ uniformly as $|s| \to \infty$ in $\mathrm{Re}(s) < \gamma_0$,
(b) for all singularities s_j, $|\mathrm{Im}(s_j)| < K$ and a numerical value of K is known. (2.2)

Talbot's contour, which is illustrated in Figure 1, is of the form

$$s(z) = \sigma + \lambda s_\nu(z), \qquad z \epsilon (-2\pi i, 2\pi i) \tag{2.3}$$

where

$$s_\nu(z) = \frac{z}{1 - e^{-z}} + z\,\frac{\nu - 1}{2}.$$

His prescription for determining the *geometrical parameters* λ, σ, ν and for determining the *accuracy parameter* n, i.e., the number of points required in the quadrature rule, is lengthy and is given in references [4] and [8].

The software provided here is in two parts: the first requires as input the locations of the singularities, the required accuracy, and machine characteristics and provides the value of λ, σ, ν, and n. The second carries out a numerical integration using n points, which lie on the contour and whose imaginary parts are equally spaced.

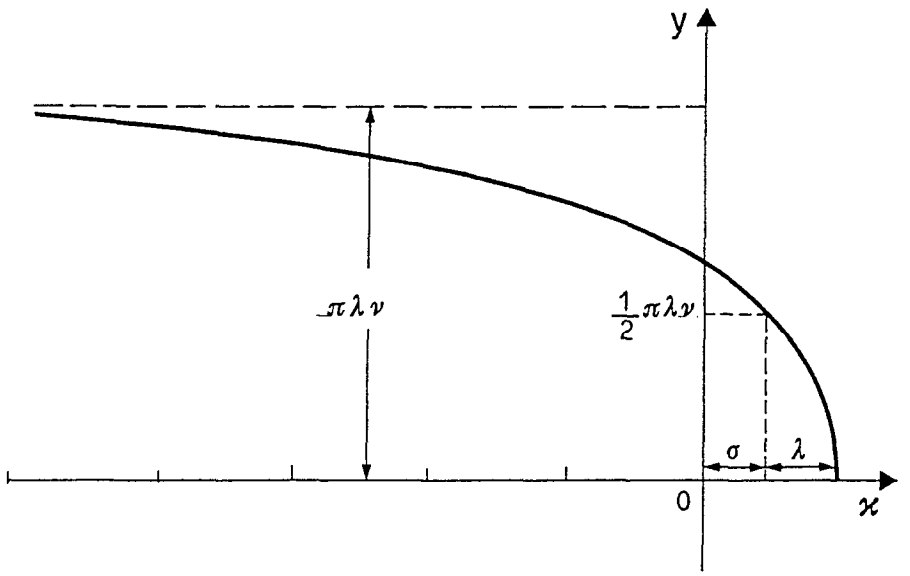

Fig. 1. Talbot's contour.

Specifically, put in (2.3) $z = 2i\theta$, then the contour is parameterized using

$$s(\theta) = \sigma + \lambda s_\nu(\theta), \qquad \theta \in (-\pi, \pi) \tag{2.4}$$

where

$$s_\nu(\theta) = \theta \cot \theta + i\nu\theta,$$

and the integral (2.1) takes the form

$$\begin{aligned} f(t) &= \frac{\lambda e^{\sigma t}}{2\pi i} \int_{-\pi}^{\pi} e^{\lambda t s_\nu(\theta)} F[\sigma + \lambda s_\nu(\theta)] s_\nu'(\theta)\, d\theta \\ s_\nu'(\theta) &= i\left\{\nu + \frac{\theta - \cos\theta \sin\theta}{\sin^2\theta}\right\}. \end{aligned} \tag{2.5}$$

Taking symmetry into account, the trapezoidal rule approximation gives

$$\tilde{f(t)} = \frac{\lambda e^{\sigma t}}{n} T_n(t) \tag{2.6}$$

where

$$T_n(t) = \sum_{j=0}^{n}{}'' e^{\lambda t s_\nu(\theta_j)} F[\sigma + \lambda s_\nu(\theta_j)] \frac{1}{i} s_\nu'(\theta_j)$$

$$\theta_j = j\frac{\pi}{n}.$$

The $\sum''$ notation usually denotes a sum where the first and last terms are halved; in this case, in the sum, the $j = n$ term is zero and the $j = 0$ term becomes

$$\frac{\nu}{2} e^{\lambda t} F(\sigma + \lambda).$$

3. SOFTWARE IMPLEMENTATION

The user is provided with two routines. These are called successively: the first (subroutine TAPAR) provides the geometrical parameters λ, σ, ν defining a Talbot's contour and the number n of function values that should be used in the subsequent quadrature routine. These parameters need to be computed for each value of t for which the Inverse transform is required. Inputs to the subroutine TAPAR are

—the value of t where the Inverse Laplace transform $f(t)$ is required ($t > 0$);
—the location and nature of some of the singularities of the Laplace transform $F(s)$;
—decimal machine-precision; and
—the required accuracy in the result.

The second routine (subroutine TSUM) requires t, λ, σ, ν, and n as input, and calculates the approximation (2.6) to the contour integral (2.5). More precisely, inputs to the subroutine TSUM are

—the value of t (as before);
—the Laplace transform, passed by an external complex function of a complex argument;
—geometrical parameters λ, σ, and ν; and
—the number n of abscissas.

The division into two subroutines allows the possibility, for expert users, to modify either the method's parameters or the quadrature rule at will. Our experience is that minor modifications to the contour can result in a large increase in the cost of integration. Nevertheless, users may

(a) increase or decrease σ, or
(b) change λ and ν in such a way that neither λ nor $\lambda\nu$ are decreased,

and remain confident that the new contour is equivalent to Talbot's. In addition, users may use Talbot's contour but, for additional safety, use a larger value of n for the number of abscissas.

Incidentally, we remark that in general the cost of running TSUM far exceeds that of running TAPAR, and for different values of t, TSUM cannot reuse previous function values whether or not the same contour is used.

Moreover, we chose complex arithmetic for the Laplace transform function—input parameter to the subroutine TSUM—because it is simpler, for nonexpert users to write a FORTRAN complex function as a copy of the corresponding mathematical function. But, as we read in [1], the collection of software for

complex functions in general is not very extensive, and the most important numerical libraries are only minimally cognizant of complex variable computation. A reason for this is that the FORTRAN double precision complex arithmetic is not yet standardized; we give an example of this fact in the sequel.

So if the Laplace transform function is furnished by some library or some existing software, users will probably have at their disposal a FORTRAN subroutine instead of the FORTRAN complex function required by TSUM. How can users use the Talbot method implementation? Suppose they have, for computing $F(s)$, a subroutine like the following:

```
SUBROUTINE LAPFUN (SREAL, SIMAG, FREAL, FIMAG)
```

where SREAL, SIMAG, FREAL, and FIMAG are respectively, real variables, for the real and imaginary parts of the argument of the function and the real and imaginary parts of the function itself.

To use TSUM, it is sufficient to provide the following interface function:

```
COMPLEX FUNCTION F(S)
COMPLEX S
REAL   SREAL, SIMAG, FREAL, FIMAG
  SREAL = REAL(S)
  SIMAG = IMAG(S)
  CALL LAPFUN(SREAL, SIMAG, FREAL, FIMAG)
  F = CMPLX(FREAL, FIMAG)
  RETURN
END
```

A similar interface function (but one that is simpler) is required if the Laplace transform function $F(s)$ is computed in complex arithmetic by a FORTRAN subroutine such as the following:

```
SUBROUTINE FUNLAP(SCMPLX, FCMPLX)
COMPLEX SCMPLX, FCMPLX
  ...
```

The software presented here is written in ANSI FORTRAN 77, using single precision arithmetic throughout. We have available fully tested versions using double precision arithmetic.

Double precision complex arithmetic is not standard in FORTRAN; we have tested our software successfully on several computers: at first a UNIVAC 1100/90 (FTN ASCII compiler) and a HEWLETT PACKARD 1000/F (FTN7X compiler), and then a CDC, a VAX, and an IBM-PC, each of which uses different double precision complex arithmetic. It is straightforward for users to modify these routines to use double precision arithmetic; however, they should take great care to locate and allow for these sort of effects.

A more insidious problem we encountered using the double complex arithmetic in FORTRAN 77 is concerned with different values furnished, in single and double precision, by some built-in functions with branch-points such as the square root. Testing our software on the HP 1000 machine, we found that the FORTRAN routine CSQRT(S), used in a Laplace transform test function, gives different results (for the same complex argument S, of course) in single and double precision, due to the fact that the complex square root is a two-valued

function. In single precision, it seems to work in the cut plane $|\arg s| < \pi$, while in double precision, in the cut plane $-3/2\pi < \arg s < \pi/2$. So, for example, if we compute CSQRT(S) where the argument of S lies in $(\pi/2, \pi)$, the complex single precision computation gives the correct result, with its argument lying in $(\pi/4, \pi/2)$, while the complex double precision computation gives as a result the *opposite* of the value returned in single precision. Naturally, this phenomenon may have a catastrophic effect on the numerical result also if, when *discovered*, it is easy to solve the problem: it suffices to multiply the wrong complex value by an appropriate rotation factor.

In order to assure complete portability, we provide only the single precision version of the software.

Besides TAPAR and TSUM, implementation of Talbot's method requires two further subroutines. One is INVPC (called by TAPAR). This inverts the equation of the contour (for $\nu = 1$):

$$s_1(z) = \frac{z}{1 - e^{-z}}$$

for any s located to the left of the contour. The subroutine INVPC implements a suitable real Newton process to solve the above complex nonlinear equation, and it only requires a few iterations (in most cases three to five) to satisfy a convergence criterion of 0.01 percent.

The other routine is R1MACH [3] (called by TSUM). This routine furnishes some environmental arithmetic parameters for several machines (by a simple manipulation of the FORTRAN code); in this case it is used to obtain the largest magnitude floating-point number to avoid an overflow in the final result.

An overflow problem in computing $f(t)$ may arise, for large t, when $F(s)$ has a singularity s_j such that

$$\mathrm{Re}(s_j) > 0.$$

The same problem occurs in the numerical approximation $\tilde{f}(t)$ given by (2.6), because of the factor

$$\frac{\lambda e^{\sigma t}}{n}, \qquad \sigma > 0.$$

The remedy is the following: since the algorithm at first computes the sum $T_n(t)$ in (2.6), whose terms are small when compared to the final result, in order to evaluate (2.6), we put

$$\tilde{f}(t) = \mathrm{sgn}[\tilde{f}(t)]e^{\ln|\tilde{f}(t)|}. \tag{3.1}$$

If M is the maximum representable floating-point number and

$$M = e^{\mu},$$

then $\tilde{f}(t)$ may produce overflow when

$$\sigma t + \ln\left[\frac{\lambda}{n} T_n(t)\right] > \mu.$$

In this case, an error indicator (output parameter) in the subroutine TSUM signals that an overflow might occur, and the routine, instead of $f(\tilde{t})$, returns $T_n(t)$; the true result can then be computed by (2.6).

Several different schemes have been tested for computing the sum $T_n(t)$ in the quadrature routine (TSUM). In our context, better results were obtained using the Reinsch form of the Goertzel algorithm [7] to evaluate the Chebyshev–Clenshaw sum that gives the Trapezoidal approximation to $f(t)$. This algorithm is implemented in the program.

4. MISCELLANEOUS

In this section we provide two examples in which Talbot's method does not work well, and give some suggestions to bypass the obstacles.

4.1 Essential Singularity

A particular problem arises for the function

$$F(s) = \frac{e^{-1/s}}{\sqrt{s}}, \qquad f(t) = \frac{\cos 2\sqrt{t}}{\sqrt{\pi t}}.$$

This Laplace transform has an essential singularity in 0, and its behavior near an essential singularity badly influences the results of the method, possibly leading to overflow in $F(s)$ for large t. A similar situation occurs using other methods.

Talbot suggests a remedy: to increase the value of λ so that the contour avoids this singularity by a wide margin. If c is the decimal machine precision, Talbot's modification consists of

$$\lambda = \frac{\omega - 1}{t} + \frac{1}{30}, \qquad \omega = 0.4(c + 1). \tag{4.1}$$

On the basis of many experimental results, we suggest that the modification

$$\lambda = \frac{\omega - 1}{t} + \frac{1}{\sqrt{\omega t}}, \qquad \omega \text{ as before} \tag{4.2}$$

should be used instead of (4.1), and that n should be increased to n', given by

$$n' = n \log_{10}(t) \qquad \text{when} \quad t > 10.$$

With the modification (4.2), the result is correct for t less than 10^5, while the modification (4.1) produces overflow when $t > 10^3$.

For larger t, the overflow problem may still remain, but it can be mitigated by increasing λ again. Our recommendations are based on limited numerical evidence.

4.2 Necessity of Condition (2.2a)

Condition (2.2a) is necessary for Talbot's method to work. For example, when applied to the function

$$F(s) = \frac{e^{-5s}}{s}, \qquad f(t) = H(t - 5)$$

where H is the *Heaviside step unit function*, Talbot's method gives unpredictable results (often overflow) for small values of t; however, when $t > 5$, its behavior returns to normal. If it is known that $F(s)$ is of the form

$$F(s) = e^{-as}G(s), \qquad a > 0$$

and $G(s)$ is known to satisfy Condition (2.2a), the method may still be applied to $F(s)$, but with the restriction $t > a$.

5. NUMERICAL RESULTS

Many tests have been performed on several Laplace transforms, including functions with only real singularities and functions with complex singularities (both polar and essential).

In discussing numerical results, we compare our implementation of Talbot's method with a modification of Talbot's method introduced by Piessens et al., in the book QUADPACK [5], and with DLAINV [6]. For brevity, we refer to our implementation as TALBOT-SOFTWARE and to Piessens' modification of Talbot's method as TALBOT-QUADPACK.

DLAINV implements a method based on a Fourier series expansion for approximating $f(t)$ and uses the ϵ-*algorithm* to accelerate the convergence; while TALBOT-QUADPACK and TALBOT-SOFTWARE implement methods based on the numerical quadrature in the complex plane, starting from the Riemann inversion formula. The difference between them consists of two different contours.

In Tables I and II we report, for several values of t, some numerical results related to the following tests functions:

$$F_1(s) = s^{-2} \qquad f_1(t) = t$$

$$F_2(s) = s^{-1}\ln s \qquad f_2(t) = -\eta - \ln t \qquad (\eta = \text{Euler's number})$$

$$F_3(s) = e^{-4\sqrt{s}} \qquad f_3(t) = \frac{2e^{-4/t}}{t\sqrt{\pi t}}$$

$$F_4(s) = \arctan\frac{1}{s} \qquad f_4(t) = \frac{\sin t}{t}$$

$$F_5(s) = \ln\frac{s^2+1}{s^2+4} \qquad f_5(t) = 2\,\frac{\cos 2t - \cos t}{t}$$

$$F_6(s) = \frac{s^2}{s^3+8} \qquad f_6(t) = \frac{e^{-2t} + 2e^{t}\cos(t\sqrt{3})}{3}$$

The first four functions are taken from [2] and the last two from [8]. All the computations were carried out on a UNIVAC 1100/90 computer. TALBOT-QUADPACK (single precision) was run with constants c_1 and c_2 equal to 5, requiring an accuracy in the result of 10^{-6}; DLAINV (double precision) was run requiring an accuracy in the result of 10^{-6}; TALBOT-SOFTWARE (single precision) was run requiring an accuracy in the result of 6 decimal digits.

Comparing the results of Tables I and II, we note first of all that the accuracy reached by TALBOT-SOFTWARE and TALBOT-QUADPACK is almost the same, while the accuracy furnished by DLAINV is worse, although DLAINV

Table I. Numerical Results for Transforms with Real Singularities. Required Accuracy = 10^{-6}.

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	+.100E+0	11	1.E-14	4005$^+$	3.E-7	155	1.E-8
1	+.100E+1	11	1.E-8	"	3.E-6	185	1.E-7
10	+.100E+2	11	1.E-15	"	3.E-5	485	8.E-8
100	+.100E+3	11	1.E-15	"	3.E-4	485	1.E-7
1000	+.100E+4	11	3.E-8	"	3.E-3	485	9.E-8

$$F_1(s) = 1/s^2 \qquad f_1(t) = t$$

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	+.172E+1	11	1.E-7	4005$^+$	1.E-2	155	2.E-7
1	-.577E+0	11	2.E-7	"	9.E-6	185	1.E-7
10	-.287E+1	11	4.E-8	"	4.E-3	185	7.E-8
100	-.518E+1	11	9.E-9	"	5.E+0	185	3.E-8
1000	-.748E+1	11	2.E-9	"	4.E-3	185	1.E-7

$$F_2(s) = \frac{\ln s}{s} \qquad f_2(t) = -\eta - \ln t \quad (\eta = \text{Euler's number})$$

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	+.151E-15	11	1.E-8	93	2.E-20	95	2.E-7
1	+.206E-1	11	3.E-7	4005$^+$	3.E-19	185	1.E-9
10	+.239E-1	11	5.E-9	"	2.E-12	185	4.E-9
100	+.108E-2	11	3.E-9	"	5.E-6	155	1.E-9
1000	+.355E-4	11	5.E-10	3597	7.E-10	215	3.E-10

$$F_3(s) = e^{-4\sqrt{s}} \qquad f_3(t) = \frac{2e^{-4/t}}{t\sqrt{\pi t}}$$

Note: $^+$maximum number of evaluations has been achieved.

works in double precision. This fact confirms our experience in testing several methods based on Fourier series: these methods require that a small range of t is fixed and suffer from convergence acceleration problems. However, they are the only applicable methods when the singularities of $F(s)$ do not satisfy the property (2.2b).

Second, comparing the number of function evaluations used by the three softwares, our implementation of Talbot's method is much more efficient because it uses, in general, many fewer function values: this always seems to be true for every t when $F(s)$ has only real singularities, and for moderate t when $F(s)$ has complex singularities.

Table II. Numerical Results for Transform with Complex Singularities. Required Accuracy = 10^{-6}.

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	+.998E+0	11	1.E-7	4005+	2.E-3	155	4.E-6
1	+.841E+0	11	1.E-7	"	2.E-3	215	4.E-7
10	-.544E-1	19	1.E-7	"	2.E-3	255	1.E-7
100	-.506E-2	70	9.E-7	"	2.E-3	485	4.E-8
1000	+.826E-3	525	5.E-8	"	8.E-4	755	7.E-8

$$F_4(s) = \arctan(1/s) \qquad f_4(t) = \frac{\sin t}{t}$$

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	-.298E+0	11	5.E-8	4005+	8.E-7	155	6.E-7
1	-.191E+1	11	1.E-8	"	1.E+0	185	4.E-8
10	+.249E+0	23	7.E-7	"	8.E-5	315	5.E-7
100	-.750E-2	112	1.E-7	"	8.E-9	905	1.E-6
1000	-.185E-2	2642	1.E-7	"	2.E-3	1755	6.E-6

$$F_5(s) = \ln \frac{s^2+1}{s^2+4} \qquad f_5(t) = 2(\cos 2t - \cos t)/t$$

		TALBOT		DLAINV		QUADPACK	
t	*Exact*	*No.Eval.*	*Error*	*No.Eval.*	*Error*	*No.Eval.*	*Error*
.1	+.998E+0	11	2.E-10	4005+	2.E-3	155	5.E-7
1	-.245E+0	11	4.E-7	"	5.E-3	185	3.E-8
10	+.612E+3	23	1.E-4	"	6.E-2	255	2.E-5
100*	-.163E+44	152	3.E-5	"	2.E-3	1185	4.E+2
1000*	+.659E+304	1112	8.E-6	***		765	2.E+2

$$F_6(s) = \frac{s^2}{s^3+8} \qquad f_6(t) = (e^{-2t} + 2e^t \cos t\sqrt{3})/3$$

Note: +maximum number of evaluations has been achieved. *Single precision overflow in the result. ***The routine has aborted, since an overflow occurs during its execution.

Finally, but equally importantly, we emphasize that our remedy to solve an overflow problem in the final result (an example is reported in Table II for F_6) allows us to get a correct numerical result, while in DLAINV the occurrence of an overflow suddenly aborts the run, and in TALBOT-QUADPACK the overflow is detected, but a partial numerical result is returned with a large relative error.

6. CONCLUDING REMARKS

In this paper we described a software implementation of Talbot's method for the inverse Laplace transform problem.

Numerical results confirm that this method is able to satisfy efficiently any accuracy requirement (of course bounded by computer precision), provided that the locations of singularities of $F(s)$ are known and the transform $F(s)$ can be found as a calculable analytic function of s.

The software has been designed in a modular manner; so that, on the one hand, it is easily used by a nonexpert user and, on the other hand, it allows great flexibility to the expert user.

ACKNOWLEDGMENT

We would like to acknowledge the help and encouragement given by James N. Lyness on a preliminary version of this paper. We also wish to thank the referee for his suggestions and for his work on our software.

REFERENCES

1. AMOS, D. E. Algorithm 644: A portable package for Bessel functions of a complex argument and nonnegative order. *ACM Trans. Math. Softw. 12*, 3 (1986).
2. DAVIES, B., AND MARTIN, B. Numerical inversion of the Laplace transform: A survey and comparison of methods. *J. Comput. Phys. 33* (1979).
3. FOX, P. A., HALL, A. D., AND SCHRYER, N. L. Algorithm 528: Framework for a portable library. *ACM Trans. Math. Softw. 4*, 1 (1978).
4. MURLI, A., AND RIZZARDI, M. Sull'implementazione del metodo di Talbot per la inversione numerica della Transformata di Laplace—Progetto Finalizzato Informatica del CNR, Rome, Rapporto SOFMAT 10.83, 1983. (In Italian.)
5. PIESSENS, R., DE DONCKER-KAPENGA, E., UBERHUBER, C. W., AND KAHANER, D. K. *QUADPACK. A Subroutine Package for Automatic Integration.* Springer Verlag, New York, 1983.
6. PIESSENS, R., AND HUYSMANS, R. Algorithm 619: Automatic numerical inversion of the Laplace transform. *ACM Trans. Math. Softw. 10*, 3 (1984).
7. STOER, J. *Introduzione all'Analisi Numerica*, vol. I. Zanichelli, 1976. Original Title: *Einfuhrung in die Numerische Mathematik.* Springer Verlag, 1972.
8. TALBOT, A. The accurate numerical inversion of Laplace transforms. *J. Inst. Math. Appl. 23* (1979).

Received May 1986; revised November 1987, February 1989; accepted June 1989