ELSEVIER

Performance Evaluation 31 (1998) 229–243

PERFORMANCE EVALUATION An International Journal

Numerical inversion of multidimensional Laplace transforms by the Laguerre method

Joseph Abate [a], Gagan L. Choudhury [b,1], Ward Whitt [c,*]

[a] 900 Hammond Road, Ridgewood, NJ 07450-2908, USA

[b] AT&T Laboratories, Room 1L-238, Holmdel, NJ 07733-3030, USA

[c] AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932-0971, USA

Received 15 July 1995; received in revised form 29 November 1996

Abstract

Numerical transform inversion can be useful to solve stochastic models arising in the performance evaluation of telecommunications and computer systems. We contribute to this technique in this paper by extending our recently developed variant of the Laguerre method for numerically inverting Laplace transforms to multidimensional Laplace transforms. An important application of multidimensional inversion is to calculate time-dependent performance measures of stochastic systems. Key features of our new algorithm are: (1) an efficient FFT-based extension of our previously developed variant of the Fourier-series method to calculate the coefficients of the multidimensional Laguerre generating function, and (2) systematic methods for scaling to accelerate convergence of infinite series, using Wynn's ϵ-algorithm and exploiting geometric decay rates of Laguerre coefficients. These features greatly speed up the algorithm while controlling errors. We illustrate the effectiveness of our algorithm through numerical examples. For many problems, hundreds of function evaluations can be computed in just a few seconds. © 1998 Published by Elsevier Science B.V.

Keywords: Numerical transform inversion; Laplace transforms; Multidimensional Laplace transforms; Laguerre polynomials; Weeks' algorithm; Fast Fourier transform; Accelerated summation; Wynn's ϵ-algorithm

1. Introduction

In this paper we develop an effective algorithm for numerically inverting multidimensional Laplace transforms by the Laguerre method. This paper is a sequel to our previous paper [1] in which we developed a new variant of the Laguerre method for numerically inverting one-dimensional Laplace transforms. Other one-dimensional variants of the Laguerre method are the original (1966) Weeks [18] algorithm and ACM Algorithm 662 in Garbow et al. [9,10]. Another variant of the Laguerre method

* Corresponding author. Tel.: +1 908 582 6484; fax: +1 908 582 2379; e-mail: wow@research.att.com.

[1] E-mail: gagan@buckaroo.att.com.

for multidimensional Laplace transforms has recently been proposed by Moorthy [11] (which came to our attention while this paper was under review). The general approach here is the same as in [11] and as in previous one-dimensional algorithms such as [1,9,10,18], but there are important differences in implementation.

We are interested in multidimensional transform inversion because it allows us to calculate quantities of interest in many important stochastic models arising in the performance analysis of telecommunications and computer systems. Examples include time-dependent performance of stationary and non-stationary systems [5] and joint distributions in polling models [6]. Our algorithm here provides an alternative to the Fourier-series algorithm for inverting multidimensional Laplace transforms developed in [4]. (Another variant of the Fourier-series method for multidimensional Laplace transforms recently has been presented by Moorthy [12].) As in the one-dimensional case, our experience is that the Fourier-series method tends to be more robust (i.e., works for a larger class of functions without special tuning), but the Laguerre method can be very fast for well-behaved functions, especially when function values are sought for a large number of arguments.

For simplicity, we consider only the bivariate case, but the algorithm extends directly to n-dimensional functions. Thus, our goal is to calculate values of a real-valued function f defined on the positive quadrant of the plane, $\mathbb{R}^2_+ \equiv [0,\infty) \times [0,\infty)$, by numerically inverting its Laplace transform

$$\hat{f}(s_1, s_2) = \int_0^\infty \int_0^\infty \mathrm{e}^{-(s_1 t_1 + s_2 t_2)} f(t_1, t_2)\,\mathrm{d}t_1\,\mathrm{d}t_2, \tag{1}$$

which we assume is well-defined, e.g., convergent and thus analytic for $\mathrm{Re}(s_1) > 0$ and $\mathrm{Re}(s_2) > 0$; e.g., see [8] or [17].

The basis for our inversion algorithm is the classical Laguerre-series representation of f, which we review in Section 2. We review how to compute the Laguerre functions in Section 3. In Section 4 we develop an efficient algorithm to compute the Laguerre coefficients. It is based on the multidimensional generating function inversion algorithm developed in [4], but greatly speeds it up through several modifications, including a fast-Fourier-transform (FFT) implementation. In Section 5 we develop scaling and summation acceleration techniques, extending those in [1], to speed up the convergence of the Laguerre series. In Section 6 we give numerical examples from queueing theory illustrating the algorithm. In particular, we calculate the complementary cumulative distribution function (tail probability) of the time-dependent workload (virtual waiting time) in the transient M/G/1 queue with various service-time distributions. Finally, in Section 7 we summarize the algorithm.

2. Laguerre-series representation

As indicated in Section 1, our goal is to compute values of a bivariate function f from its two-dimensional Laplace transform $\hat{f}$ in (1). To do so, we exploit a connection between the Laplace transform $\hat{f}$ and the generating function of the coefficients of the Laguerre-series representation of f.

For two dimensions, the classical *Laguerre-series representation* takes the form

$$f(t_1, t_2) = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q_{n_1,n_2} l_{n_1}(t_1) l_{n_2}(t_2), \quad t_1 \geq 0 \text{ and } t_2 \geq 0, \tag{2}$$

where

$$l_n(t) = \mathrm{e}^{-t/2} L_n(t), \quad t \geq 0, \tag{3}$$

$$L_n(t) = \sum_{k=0}^{n} \binom{n}{k} \frac{(-t)^k}{k!}, \quad t \geq 0, \tag{4}$$

and

$$Q(z_1, z_2) \equiv \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q_{n_1,n_2} z_1^{n_1} z_2^{n_2} = (1-z_1)^{-1}(1-z_2)^{-1} \hat{f}\left(\frac{1+z_1}{2(1-z_1)}, \frac{1+z_2}{2(1-z_2)}\right), \tag{5}$$

with $L_n(t)$ in (4) being the *Laguerre polynomials*, $l_n(t)$ in (3) the associated *Laguerre functions*, q_{n_1,n_2} in (2) the *Laguerre coefficients* and $Q(z_1, z_2)$ in (5) the *Laguerre generating function*.

The key connection between the Laguerre-series representation and the Laplace transform $\hat{f}$ is of course (5). The Laguerre-series representation of f can serve as a basis for inverting its Laplace transform $\hat{f}$ in (1) because the Laguerre generating function Q in (5) is expressed directly in terms of the Laplace transform $\hat{f}$. This occurs because the nth Laguerre function has the Laplace transform

$$\hat{l}_n(s) \equiv \int_0^{\infty} \mathrm{e}^{-st} l_n(t)\,\mathrm{d}t = 2(2s-1)^n/(2s+1)^{n+1}. \tag{6}$$

By (1), (2) and (6), the Laplace transform can be expressed as

$$\hat{f}(s_1, s_2) = 4 \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q_{n_1,n_2} \frac{(2s_1-1)^{n_1}}{(2s_1+1)^{n_1+1}} \frac{(2s_2-1)^{n_2}}{(2s_2+1)^{n_2+1}}. \tag{7}$$

By using the conformal mapping

$$(z_1, z_2) = (T(s_1), T(s_2)), \qquad (s_1, s_2) = (T^{-1}(z_1), T^{-1}(z_2)) \tag{8}$$

with

$$z = T(s) = \frac{2s-1}{2s+1}, \qquad s = T^{-1}(z) = \frac{1+z}{2(1-z)}. \tag{9}$$

we obtain (5) from (7).

We can now summarize the algorithm. By (5), the Laplace transform $\hat{f}$ enables us to obtain the Laguerre generating function. We then invert the generating function to obtain the Laguerre coefficients q_{n_1,n_2}. The Laguerre coefficients plus the Laguerre functions l_n in (3) enable us to compute the desired function values $f(t_1, t_2)$ via (2).

The bivariate Laguerre-series representation was considered by Sumita and Kijima [10], but they did not present an algorithm for computing the Laguerre coefficients q_{n_1,n_2} from the Laplace transform, which is the major contribution of this paper.

Formula (2) implies that the mathematical basis for the inversion algorithm is the theory of orthogonal polynomials. The *product Laguerre functions* $l_{n_1}(t_1) l_{n_2}(t_2)$ form an orthonormal basis for the Hilbert space $L_2(\mathbb{R}_+^2, \mathbb{R})$ of square integrable real-valued functions on $\mathbb{R}_+^2$, with the inner product

$$\langle f_1, f_2\rangle \equiv \int_0^\infty \int_0^\infty f_1(t_1, t_2) f_2(t_1, t_2)\, \mathrm{d}t_1\, \mathrm{d}t_2, \tag{10}$$

so that (2) is valid in the sense of convergence in $L_2(\mathbb{R}^2, \mathbb{R})$ for any f in $L_2(\mathbb{R}^2_+, \mathbb{R})$ with

$$q_{n_1,n_2} = \int_0^\infty \int_0^\infty f(t_1, t_2) l_{n_1}(t_1) l_{n_2}(t_2)\, \mathrm{d}t_1\, \mathrm{d}t_2; \tag{11}$$

see [14,16]. If $q^{(i)}_{n_1,n_2}$ are the Laguerre coefficients associated with f_i, then the inner product can be expressed as

$$\langle f_1, f_2\rangle = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q^{(1)}_{n_1,n_2} q^{(2)}_{n_1,n_2} \tag{12}$$

and the squared L_2 norm as

$$\|f\|_2^2 = \langle f, f\rangle = \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q^2_{n_1,n_2} < \infty. \tag{13}$$

Modifications of the Laguerre-series representation also hold for a large class of non-square integrable functions by virtue of scaling; see Section 5. The Laguerre-series representation also extends directly to complex-valued functions.

3. Calculating the Laguerre functions

In order to calculate $f(t_1, t_2)$ via the Laguerre-series representation in (2), we need to calculate the Laguerre functions $l_n(t)$ and the Laguerre coefficients q_{n_1,n_2}. We indicate how to calculate $l_n(t)$ in this section and q_{n_1,n_2} in the next section.

The Laguerre functions $l_{n_1}(t_1)$ and $l_{n_2}(t_2)$ are computed just as in the one-dimensional case. Specifically, the following recursion is used:

$$l_n(t) = \left(\frac{2n-1-t}{n}\right) l_{n-1}(t) - \left(\frac{n-1}{n}\right) l_{n-2}(t), \tag{14}$$

starting with $l_0(t) = \mathrm{e}^{-t/2}$ and $l_1(t) = (1-t)\,\mathrm{e}^{-t/2}$.

A requirement for directly obtaining an effective algorithm using (2) is that the summands $q_{n_1,n_2} l_{n_1}(t_1)$ $l_{n_2}(t_2)$ must decay fast as either n_1 gets large or n_2 gets large. (Note that it is not enough to have fast decay only when both n_1 and n_2 get large.) As noted in [1, Section 3], $|l_n(t)| \le 1$ for all n and t, but $l_n(t)$ approaches 0 slowly in an oscillating manner as $n \to \infty$, i.e.,

$$l_n(t) = \frac{1}{\sqrt{\pi}(nt)^{1/4}} \cos(2\sqrt{nt} - (\pi/4)) \quad \text{as } n \to \infty. \tag{15}$$

Thus, it is crucial to have the Laguerre coefficients q_{n_1,n_2} well-behaved. We discuss their calculation next.

4. Algorithm to compute the Laguerre coefficients

Since the decay of $l_n(t)$ with n is very slow, in order to have an effective algorithm, q_{n_1,n_2} must decay fast as either n_1 gets large or n_2 gets large. However, in many cases q_{n_1,n_2} obtained from $Q(z_1, z_2)$ in (5) may *not* actually decay fast with both n_1 and n_2. This difficulty is addressed in Section 5. In some cases the slow convergence may be handled by appropriately scaling $Q(z_1, z_2)$. In other cases a summation acceleration technique applied to the double infinite sum (2) greatly improves the accuracy. A combination of scaling and summation acceleration should handle most of these problems.

Computation of q_{n_1,n_2} requires the double inversion of the bivariate generating function $Q(z_1, z_2)$ given in (5). This we do by applying the Fourier-series based inversion algorithm given in [4, Section 3].

4.1. Fourier-series algorithm

Through a slight modification of Eq. (3.5) in [4], we get the approximation

$$q_{n_1,n_2} \approx \bar{q}_{n_1,n_2} \equiv \frac{1}{m_1 r_1^{n_1}} \{\mathrm{Re}[\hat{Q}(r_1, n_2)] + (-1)^{n_1}\mathrm{Re}[\hat{Q}(-r_1, n_2)]\} + \sum_{k=1}^{(m_1/2)-1} \mathrm{Re}[\exp(-2\pi i k n_1/m_1)\hat{Q}(r_1\,\mathrm{e}^{2\pi i k/m_1}, n_2)], \tag{16}$$

where $\mathrm{i} = \sqrt{-1}$,

$$\hat{Q}(z_1, n_2) = \frac{1}{m_2 r_2^{n_2}} \sum_{k=-m_2/2}^{(m_2/2)-1} \exp(-2\pi i k n_2/m_2) Q(z_1, r_2\,\mathrm{e}^{2\pi i k/m_2}) \tag{17}$$

and the resulting aliasing error is given by

$$e_a \equiv \bar{q}_{n,n_2} - q_{n,n_2} = \sum_{\substack{j=0 \\ j+k>0}}^{\infty} \sum_{k=0}^{\infty} q_{n_1+jm_1,n_2+km_2} r_1^{jm_1} r_2^{km_2}. \tag{18}$$

If $|q_{n_1+jm_1,n_2+km_2}| \le C$ for each j, k combination appearing in (18) and if we choose $r_1 = 10^{-A_1/m_1}$ and $r_2 = 10^{-A_2/m_2}$, then it can be shown that the aliasing error is bounded by

$$|e_a| \le \frac{C(10^{-A_1} + 10^{-A_2})}{(1 - 10^{-A_1})(1 - 10^{-A_2})} \approx C(10^{-A_1} + 10^{-A_2}). \tag{19}$$

The aliasing error may be effectively controlled by choosing A_1 and A_2 large, provided that C is not large. Typically, $A_1 = 11$ and $A_2 = 13$ are sufficient for good accuracy.

In [4] we chose $m_1 = 2l_1n_1$ and $m_2 = 2l_2n_2$, where l_i is a roundoff error control parameter. (To be consistent with [4], we retain the same notation; these are not the Laguerre functions $l_n(t)$ in (3).) The roundoff error may be reduced by increasing the parameters l_1 and l_2. Typically, $l_2 = 2$ and $l_1 = 1$ or 2 are sufficient for good accuracy. The above choice with m_j changing with n_j for $j = 1$ and 2 is appropriate if we need inversion at only a few points. However, in the current context, q_{n_1,n_2} has to be computed for all n_1, n_2 in the range $0 \le n_1 \le N_1 - 1$ and $0 \le n_2 \le N_2 - 1$, i.e., at a total of N_1N_2 points, where N_1 and N_2 have to be sufficiently large so that accurate evaluation of the double infinite sum in (2) is possible using q_{n_1,n_2} only in the above range, using either truncation or a summation acceleration technique to be described in Section 5.

In the current context, we obtain a better algorithm by fixing m_1 and m_2, i.e., by making m_i independent of n_i. Specifically, we choose $m_1 = 2l_1N_1$ and $m_2 = 2l_2N_2$ for all n_1, n_2 in the range $0 \le n_1 \le N_1 - 1$ and $0 \le n_2 \le N_2 - 1$. At first sight, this choice might seem inefficient, since by looking at (16) and (17), we see that with $m_j = 2l_jn_j$, the total number of summations performed is

$$\sum_{n_1=1}^{N_1-1}\sum_{n_2=1}^{N_2-1} 2l_2n_2(l_1n_1+1) \approx \frac{l_1l_2N_1^2N_2^2}{2} \quad \text{for large } N_1, N_2.$$

By contrast, the number of summations performed with the choice $m_j = 2l_jN_j$ is

$$\sum_{n_1=1}^{N_1-1}\sum_{n_2=1}^{N_2-1} 2l_2N_2(l_1N_1+1) \approx 2l_1l_2N_1^2N_2^2 \quad \text{for large } N_1, N_2,$$

which is four times as much as in the first case. A similar difference exists in the number of multiplications as well. However, the main computational advantage comes from the fact that, with constant m_1 and m_2, $Q(z_1, z_2)$ needs to be computed at the same set of points for all choices of n_1, n_2. If we compute the $Q(z_1, z_2)$ values once and store them for later use, then great computational saving results. Specifically, with the choice $m_j = 2l_jn_j$, the number of times $Q(z_1, z_2)$ needs to be computed is

$$\sum_{n_1=1}^{N_1-1}\sum_{n_2=1}^{N_2-1} 2l_2n_2(l_1n_1+1) \simeq \frac{l_1l_2N_1^2N_2^2}{2}.$$

By contrast, with constant m_1 and m_2, the number of times $Q(z_1, z_2)$ needs to be computed is $2l_2N_2$ $(l_1N_1 + 1) \approx 2l_1l_2N_1N_2$. This is a substantial savings for large N_1 and N_2. However, we also need a storage of $2l_1l_2N_1N_2$ complex quantities. With today's computers, this is usually not a problem, even with $N_1 = N_2 = 128$ and $l_1 = l_2 = 2$.

Besides the great savings in the computation of $Q(z_1, z_2)$, further savings comes from an efficient $(2l_1N_1 \times 2l_2N_2)$-term bivariate FFT implementation for evaluating the double sum given by (16) and (17), which we describe later. It is well known (see, e.g., [13]) that the computational complexity of such an algorithm is about $4l_1l_2N_1N_2\log_2(2l_1N_1)\log_2(2l_2N_2)$, which for large N_1 and N_2 will be substantially less than the computational complexity of the $m_j = 2n_jl_j$ algorithm, given by $l_1l_2N_1^2N_2^2/2$. However, the FFT implementation increases the required storage further to $4l_1l_2N_1N_2$. We summarize the performance of the proposed FFT-based algorithm for computing q_{n_1,n_2} and compare it to the algorithm from [4] in Table 1.

Table 1
A comparison of the FFT-based algorithm and the direct Fourier-series algorithm from [4] for calculating the Laguerre coefficients q_{n_1,n_2} from $Q(z_1, z_2)$

Performance measure	Algorithm in [4]	The new FFT-based algorithm
Number of times $Q(z_1, z_2)$ needs to be computed	$\approx l_1l_2N_1^2N_2^2/2$	$\approx 2l_1l_2N_1N_2$
Computational complexity of other computations	$\approx l_1l_2N_1^2N_2^2/2$	$\approx 4l_1l_2N_1N_2\log(2l_1N_1)\log(2l_2n_2)$
Storage	None	$\approx 4l_1N_1l_2N_2$

An interesting point to note is that in the algorithm in [4] q_{n_1,n_2} is not computable if either $n_1 = 0$ or $n_2 = 0$, so that we have to use other techniques in those cases. However, with $m_j = 2l_jN_j$ no such alternate algorithm is needed when one or both n_j's are 0.

4.2. Error analysis of the new algorithm

The basic equations (16)–(19) still hold with the understanding that $m_j = 2l_jN_j$ instead of $2l_jn_j$ for $j = 1$ and 2. In the aliasing error expression (19), note that q_{n_1,n_2} appears only for either $n_1 > N_1$ or $n_2 > N_2$. However, q_{n_1,n_2} has to be small when either $n_1 > N_1$ or $n_2 > N_2$ in order for us to be able to compute the double infinite sum in (2) with only N_1 and N_2 terms for the two indices. This implies that the quantity C in (19) should be small and the aliasing error will be controlled pretty tightly. In fact, the C in the new algorithm will typically be much smaller than the C in the algorithm in [3] and this is another minor advantage of the new algorithm.

Next, let us turn to the roundoff error. Define $l'_j = m_j/2n_j = l_jN_j/n_j > l_j$. For the new algorithm, the parameter l'_j will control the roundoff error. Since $l'_j > l_j$, the new algorithm should have less roundoff error than the algorithm in [4]. Of course, the absolute roundoff error is lower bounded by the machine precision.

4.3. FFT-based algorithm

For efficient FFT implementation, we assume that N_1, l_1, m_1, N_2, l_2 and m_2 are all nonnegative powers of 2, with $m_i = 2l_iN_i$ as before. For example, we may choose $N_1 = 128, l_1 = 1, N_2 = 64, l_2 = 2$. This would give $m_1 = m_2 = 256$. At first rewrite (16) and (17) as follows:

$$\bar{q}_{n_1,n_2} = \frac{1}{m_1r_1^{n_1}} \sum_{k=0}^{m_1-1} \exp(-2\pi i k n_1/m_1)\hat{Q}(r_1 \mathrm{e}^{2\pi i k/m_1}, n_2), \tag{20}$$

$$\hat{Q}(z_1, n_2) = \frac{1}{m_2r_2^{n_2}} \sum_{k=0}^{m_2-1} \exp(-2\pi i k n_2/m_2)Q(z_1, r_2 \mathrm{e}^{2\pi i k/m_2}). \tag{21}$$

Now define the $(m_1 \times m_2)$-dimensional sequences $\{a_{n_1,n_2}\}$ and $\{b_{n_1,n_2}\}$ as follows, allowing n_1 to range from 0 to $m_1 - 1$ and n_2 to range from 0 to $m_2 - 1$:

$$a_{n_1,n_2} = \bar{q}_{n_1,n_2} r_1^{n_1} r_2^{n_2}, \tag{22}$$
$$b_{n_1,n_2} = Q(r_1 \mathrm{e}^{2\pi i n_1/m_1}, r_2 \mathrm{e}^{2\pi i n_2/m_2}). \tag{23}$$

Note that a_{n_1,n_2} and $\bar{q}_{n_1,n_2}$ are only defined in the range $0 \le n_1 \le N_1 - 1, 0 \le n_2 \le N_2 - 1$. We extend the definition over the larger range $0 \le n_1 \le m_1 - 1, 0 \le n_2 \le m_2 - 1$ by the inverse discrete Fourier transform (IDFT) relation

$$a_{n_1,n_2} = \frac{1}{m_1m_2} \sum_{j=0}^{m_1-1} \sum_{k=0}^{m_2-1} \exp\left(-\frac{2\pi i j n_1}{m_1} - \frac{2\pi i k n_2}{m_2}\right) b_{j,k}, \tag{24}$$

which follows from (20) and (21).

Eq. (24) implies that $\{b_{n_1,n_2}\}$ is the two-dimensional DFT of $\{a_{n_1,n_2}\}$ and conversely $\{a_{n_1,n_2}\}$ is the two-dimensional IDFT of $\{b_{n_1,n_2}\}$. We at first compute $\{b_{n_1,n_2}\}$ using (24) and store them. Next we compute a_{n_1,n_2} using any standard two-dimensional FFT algorithm. Specifically we used iterative one-dimensional "decimation in frequency" algorithms as in [8], which take the form

$$a_{n_1,n_2} = \frac{1}{m_1} \sum_{j=0}^{m_1-1} \exp(-2\pi \mathrm{i} j n_1/m_1) C_{j,n_2}, \tag{25}$$

$$C_{j,n_2} = \frac{1}{m_2} \sum_{k=0}^{m_2-1} \exp(-2\pi \mathrm{i} k n_2/m_2) b_{j,k}. \tag{26}$$

Once the a_{n_1,n_2} are obtained, $\bar{q}_{n_1,n_2}$ are obtained using Eq. (22).

Remark 4.1. It is interesting to compare our algorithm to a more naive direct FFT-based approach in which it is assumed that $q_{n_1,n_2} = 0$ for $n_1 \geq N_1$ or $n_2 \geq N_2$. Then q_{n_1,n_2} would be a finite length sequence and its DFT will be given by $\{Q(\mathrm{e}^{2\pi \mathrm{i} n_1/N_1}, \mathrm{e}^{2\pi \mathrm{i} n_2/N_2})\colon 0 \leq n_1 \leq N_1 - 1, 0 \leq n_2 \leq N_2 - 1\}$. The desired Laguerre coefficients q_{n_1,n_2} could be recovered from this sequence by a two-dimensional $(N_1 \times N_2)$-term FFT computation. This is equivalent to our algorithm with $r_1 = r_2 = 1$ and $m_1 = N_1, m_2 = N_2$. If q_{n_1,n_2} were indeed 0 for $n_1 \geq N_1$ or $n_2 \geq N_2$, then this procedure would be correct. Indeed, it would also be faster than our algorithm, since it works on a smaller array. However, if q_{n_1,n_2} does not vanish whenever either $n_1 \geq N_1$ or $n_2 \geq N_2$, then large aliasing errors would be introduced. In contrast, our algorithm is effective even if $|q_{n_1,n_2}| > 0$ for $n_1 \geq N_1$ or $n_2 \geq N_2$. (We only need it to be an O(1) quantity, which is a much milder requirement, since we explicitly control the aliasing error using $r_1, r_2 < 1$ and then also control the roundoff error by requiring $(n_1/N_1) = 2l_1 \geq 2$ and $(m_2/N_2) = 2l_2 \geq 2$.)

4.4. Accurate computation of very small Laguerre coefficients

In [1] we noted for the one-dimensional case that, if we need to compute $f(t)$ for large t, then it is important to compute the Laguerre coefficients q_n accurately even when $|q_n|$ is small and below the machine precision (say 10^{-14}). We showed in [1] that it is possible to do that if q_n has an asymptotic decay rate that is geometric or faster. (For slower than geometric decay rates, this problem is typically not present since $|q_n|$ is unlikely to get very small unless n is very large.) Here we extend the same procedure to two dimensions.

The basic approach is to invert the scaled generating function $\hat{Q}(z_1, z_2) = Q(\alpha_1 z_1, \alpha_2 z_2)$ with inverse function $\hat{q}_{n_1,n_2}$, which remains large compared to machine precision even for large n_1 and n_2, and hence may be computed accurately. Next the original sequence is recovered as

$$q_{n_1,n_2} = \hat{q}_{n_1,n_2} \alpha_1^{-n_1} \alpha_2^{-n_2}. \tag{27}$$

In [1], the scale parameter α was dynamically determined based on the recent-most computations of q_n, but in order to apply our new FFT-based algorithm we have to use static α_1 and α_2 in the current context. From (27) it is clear that a good static choice for α_1 and α_2 would be the inverses of the asymptotic geometric decay rates of q_{n_1,n_2} with respect to n_1 and n_2, respectively. For this purpose, define the one-dimensional generating functions

$$\bar{Q}(z_1, n_2) = \sum_{n_1=0}^{\infty} q_{n_1,n_2} z_1^{n_1}, \tag{28}$$

$$\hat{Q}(n_1, z_2) = \sum_{n_2=0}^{\infty} q_{n_1,n_2} z_2^{n_2}. \tag{29}$$

The generating functions $\bar{Q}(z_1, n_2)$ and $\hat{Q}(n_1, z_2)$ may be obtained by one-dimensional inversion of $Q(z_1, z_2)$. Let $\bar{Q}(1, n_2)$ and $\hat{Q}(n_1, 1)$ have geometric decay rates as follows:

$$\bar{Q}(1, n_2) = a_2\beta_2^{n_2} + o(\beta_2^{n_2}) \quad \text{as } n_2 \to \infty, \tag{30}$$

$$\hat{Q}(n_1, 1) = a_1\beta_1^{n_1} + o(\beta_1^{n_1}) \quad \text{as } n_1 \to \infty. \tag{31}$$

We suggest using $\alpha_1 = 1/\beta_1$ and $\alpha_2 = 1/\beta_2$ as static scale factors. These decay rates can be found using inversion, as in [3]. For further discussion of scaling to compute very small function values, see [7].

5. Scaling and summation acceleration

If q_{n_1,n_2} does not decay fast with both n_1 and n_2, then often it is possible to speed up convergence by working with the scaled function

$$h(t_1, t_2) \equiv h(t_1, t_2; b_1, b_2, \sigma_1, \sigma_2) = \mathrm{e}^{-(\sigma_1 t_1 + \sigma_2 t_2)} f(t_1/b_1, t_2/b_2) \tag{32}$$

for positive real numbers b_1, b_2, σ_1 and σ_2. Clearly, h has Laplace transform

$$\hat{h}(s_1, s_2; b_1, b_2, \sigma_1, \sigma_2) = b_1 b_2 \hat{f}((s_1 + \sigma_1)b_1, (s_2 + \sigma_2)b_2). \tag{33}$$

If we can calculate h by numerically inverting $\hat{h}$, then we can recover f from h by setting

$$f(t_1, t_2) = \mathrm{e}^{\sigma_1 b_1 t + \sigma_2 b_2 t} h(b_1 t_1, b_2 t_2; b_1, b_2, \sigma_1, \sigma_2). \tag{34}$$

The Laguerre generating function associated with h is

$$Q_h(z_1, z_2) = b_1 b_2 \hat{f}\left(\frac{b_1(1+z_1)}{2(1-z_1)} + b_1\sigma_1, \frac{b_2(1+z_2)}{2(1-z_2)} + b_2\sigma_2\right) \Big/ (1-z_1)(1-z_2). \tag{35}$$

Hence, if $q_{n_1,n_2}^{(h)}$ are the coefficients of $Q_h(z_1, z_2)$ in (35), then we calculate f by

$$f(t_1, t_2) = \mathrm{e}^{\sigma_1 b_1 t_1 + \sigma_2 b_2 t_2} \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} q_{n_1 n_2}^{(h)} l_{n_1}(b_1 t_1) l_{n_2}(b_2 t_2) \tag{36}$$

for l_{n_i} in (3).

The reason for scaling is to ensure faster convergence of $q_{n_1,n_2}^{(h)}$ with n_1 and/or n_2 compared to q_{n_1,n_2}. One example is when $\hat{f}(s_1, s_2)$ has a singularity either at $s_1 = 0$ or at $s_2 = 0$. From (5) it is clear that that will cause a singularity of $Q(z_1, z_2)$ at either $z_1 = -1$ or at $z_2 = -1$ and cause slow convergence of q_{n_1,n_2} with either n_1 or n_2. However, from (36) it is clear that by choosing $\sigma_1 > 0$ or $\sigma_2 > 0$ the corresponding singularity will be moved outside of the unit circle, resulting in faster convergence of $q_{n_1,n_2}^{(h)}$ with both n_1 and n_2.

A major application of two-dimensional inversion is the computation of time-dependent probabilities. In that setting usually the probabilities would not go to zero as time approaches infinity. If t_1 represents time, then $\hat{f}(s_1, s_2)$ would have singularity at $s_1 = 0$, $Q(z_1, z_2)$ would have singularity at $z_1 = -1$, causing slow convergence of q_{n_1,n_2} with n_1, but $q^{(h)}_{n_1,n_2}$ would have fast convergence with $\sigma_1 > 0$.

Unfortunately, however, the scale factors do not solve all problems of slow convergence. Specifically, similarly to what was shown in [1], if $\hat{f}(s_1, s_2)$ has a singularity either at $s_1 = -\infty$ or at $s_2 = -\infty$, then that would cause a singularity of $Q(z_1, z_2)$ either at $z_1 = +1$ or at $z_2 = +1$ (as is clear from (5)). However, unlike the previous case, it is clear from (36) that whatever finite $\sigma_1, \sigma_2, b_1, b_2$ we might use, $Q_h(z_1, z_2)$ would still have a singularity at $z_1 = +1$ or $z_2 = +1$.

If scaling alone cannot speed up convergence, then a second technique is to use a summation acceleration technique instead of pure truncation. For that purpose, we first rewrite (37) as

$$f(t_1, t_2) = e^{\sigma_1 b_1 t_1} \sum_{n_1=0}^{\infty} \bar{q}_n(n_1, t_2) l_{n_1}(b_1 t_1), \tag{37}$$

$$\bar{q}_n(n_1, t_2) = e^{\sigma_2 b_2 t_2} \sum_{n_2=0}^{\infty} q^{(h)}_{n_1 n_2} l_{n_2}(b_2 t_2). \tag{38}$$

Note that (38) and (39) are in the form of one-dimensional Laguerre-series representations. So, just as in [1], we can apply Wynn's ϵ-algorithm to one or both equations (38) and (39); see [1, Section 4]. The ϵ-algorithm is defined by the recursion

$$\epsilon^n_{k+1} = \epsilon^{n+1}_{k-1} + (\epsilon^{n+1}_k - \epsilon^n_k)^{-1}, \tag{39}$$

where $\epsilon^n_{-1} = 0$ and $\epsilon^n_0 = s_n$, s_n is the nth partial sum in (38) or (39); see [20,21] or [19, p. 138]. The final approximation is ϵ^n_{2m}. The ϵ-algorithm is particularly suitable if q_{n_1,n_2} has a slower than geometric decay rate with n_1 or n_2. Indeed, as in [1], a combination of scaling and ϵ-algorithm can often remarkably improve accuracy.

Finally, if q_{n_1,n_2} does have a geometric decay rate with n_1 or n_2, then, as pointed out in [1], it is usually possible to attain better accuracy than with the ϵ-algorithm by assuming pure geometric decay beyond the point of truncation and using closed-form Laguerre-series sums of pure geometric functions. The technique in two dimensions is similar to that in one dimension in [1] and so is not repeated here.

6. Queueing examples

As in [4, Section 6.3], we illustrate our multidimensional inversion algorithm by calculating the *complementary cumulative distribution function* (ccdf, i.e., the probability of the interval (t, ∞)) of the time-dependent workload (or virtual waiting time) in an M/G/1 queue. Let the arrival rate be λ and the service-time cdf be $H(t)$ with Laplace–Stieltjes transform

$$\hat{h}(s) = \int_0^{\infty} e^{-st} \, dH(t). \tag{40}$$

Let there be i_0 customers in the system at time 0 and let the customer in service be just beginning service at time 0.

Let $W(t)$ be the workload at time t. Then $f(t_1, t_2) = P(W(t_1) > t_2)$ and its Laplace transform is

$$\hat{f}(s_1, s_2) = \frac{1}{s_2}\left[\frac{1}{s_1} - \frac{\hat{h}(s_2)^{i_0} - s_2\hat{P}_{i_0 0}(s_1)}{s_1 - s_2 + \lambda - \lambda\hat{h}(s_2)}\right], \tag{41}$$

where

$$\hat{P}_{i_0,0}(s) = \frac{\hat{G}(s)^{i_0}}{s + \lambda - \lambda\hat{G}(s)} \tag{42}$$

and

$$\hat{G}(s) = \hat{h}(s + \lambda - \lambda\hat{G}(s)) . \tag{43}$$

Note that $\hat{G}(s)$ is the Laplace–Stieltjes transform of the busy-period cdf, while $\hat{P}_{i_0,0}(s)$ is the Laplace transform of the emptiness function $P_{i_0,0}(t)$; i.e., $P_{i_0,0}(t)$ is the probability that the system is empty at time t given that it started out with i_0 customers at time 0.

Usually the most time consuming part of the algorithm is the calculation of $\hat{G}(s)$, which is done recursively [2]. However, note that $\hat{G}(s)$ is needed only for $2l_1N_1$ distinct values. If we compute these values once and store them for later use, then great computational savings are obtained.

Note that the function $f(t_1, t_2)$ decays to 0 as $t_2 \to \infty$ for each fixed t_1, but not as $t_1 \to \infty$ for each fixed t_2. Hence it is important to use the scaling $\sigma_1 > 0$, but the scaling variable σ_2 needs to be positive only for certain service-time distributions.

Example 6.1. We first consider an exponential service-time distribution with mean 1, so that $\hat{h}(s) = (1 + s)^{-1}$. We also let $\lambda = 0.7$ and $i_0 = 1$. Since the traffic intensity is $\rho = \lambda = 0.7 < 1$, the model is stable so that $P(W(t_1) > t_2)$ converges to a proper steady-state ccdf as $t_1 \to \infty$. For our computations with the Laguerre algorithm, we use the scaling parameters $\sigma_1 = 0.2$, $\sigma_2 = 0$, and $b_1 = b_2 = 1$. For this example, we use $N_1 = 64$ and $N_2 = 32$ with simple truncation in Eq. (39) and the third-order epsilon algorithm in Eq. (38).

We compare the Laguerre algorithm with exact results for $f(t_1, t_2)$ for several argument pairs (t_1, t_2) in Table 2. The exact results were obtained in three different ways. For $t_1 = 0$, the exact results were obtained by noting that the workload is just the service time of the single initial customer having an exponential distribution, i.e.,

$$f(0, t_2) = P(W(0) > t_2) = \mathrm{e}^{-t_2}. \tag{44}$$

For $t_1 > 0$ and $t_2 = 0$, the exact results were obtained by inverting the one-dimensional transform $\hat{P}_{10}(s)$ of the emptiness function; i.e.,

$$f(t_1, 0) = P(W(t_1) > 0) = 1 - P_{10}(t_1). \tag{45}$$

Finally, for $t_1 > 0$ nd $t_2 > 0$, the exact results were obtained by inverting the two-dimensional transform $\hat{f}(s_1, s_2)$ using the two-dimensional Fourier-series method [4]. As in [4], for each application of the Fourier-series method, different values of the roundoff control parameters (l_1, l_2) were used to provide an accuracy check.

Since the results in Table 2 are very accurate, we see that there is no need for large N_1 or N_2. We did use the epsilon algorithm in the time dimension, but we observed that accuracy suffers only slightly without

Table 2
A comparison of the Laguerre algorithm with exact results for the workload ccdf in the transient M/M/1 queue with $\rho = 0.7$ in Example 6.1

t_1	t_2	Exact	Laguerre algorithm ($\sigma_1 = 0.2, \sigma_2 = 0, b_1 = b_2 = 1$)
0	0	1.0000000	1.0000000
0	5	6.7379470D−03	6.7379470D−03
0	10	4.5399930D−05	4.5399774D−05
5	0	6.1864223D−01	6.1864223D−01
5	5	6.1113935D−02	6.1113935D−02
5	10	4.1009696D−03	4.1009696D−03
10	0	6.5395600D−01	6.5395600D−01
10	5	9.1511168D−02	9.1511168D−02
10	10	9.7185771D−03	9.7185771D−03

Table 3
A comparison of the Laguerre algorithm with exact results for the workload ccdf in the transient M/M/1 queue with $\rho = 2.0$ in Example 6.2

t_1	t_2	Exact	Laguerre algorithm ($\sigma_1 = 0.2, \sigma_2 = 0, b_1 = b_2 = 1$)
0	0	1.0000000	1.000000
0	20	2.0611536D−09	1.9103449D−09
10	0	9.9436312D−01	9.9436312D−01
10	20	9.2662196D−02	9.2662196D−02
10	40	1.5626542D−04	1.5626546D−04
20	0	9.9954627D−01	9.9954622D−01
20	20	5.4237295D−01	5.4237295D−01
20	40	2.6159632D−02	2.6159632D−02

it. However, the accuracy suffers greatly if we set $\sigma_1 = 0$, since q_{n_1,n_2} decays very slowly with n_1 in that case.

For this example with nine points, the Laguerre algorithm took only 3 s on a SUN workstation and was already faster than the Fourier-series method. Furthermore, when we increased the number of points to 100, the Laguerre method still took only about 5 s while the computation time of the Fourier-series method went up by a factor of 10. We also tried the algorithm in [4] unchanged for computing q_{n_1,n_2} and it took several minutes of computation time.

Example 6.2. Here we consider an unstable (in the queueing sense) case of the previous example by changing λ to 2.0 while keeping the service time exponentially distributed with mean 1. We use the same initial conditions as before, with $i_0 = 1$. Also we increase the upper limits of t_1, t_2. The results are shown in Table 3. The Laguerre method again produced highly accurate results with the same parameter settings and essentially the same computation time.

Example 6.3. Now we change the service-time distribution to a gamma distribution with mean 1 and squared coefficient of variation 2. The accuracy immediately suffered, because in this case $\hat{f}(s_1, s_2)$ has

Table 4
A comparison of the Laguerre algorithm with exact results for the workload ccdf in the transient M/G/1 queue having a gamma service-time distribution with mean 1 and SCV 2 and arrival rate $\rho = 0.7$ in Example 6.3

t_1	t_2	Exact	Laguerre algorithm ($\sigma_1 = 0.2, \sigma_2 = 0, b_1 = 1, b_2 = 5$)
0	0	1.0000000D0 0	0.9696784D0
0	5	2.5347319D−02	2.5347463D−02
0	10	1.5654023D−03	1.5656101D−03
5	0	5.8817561D−01	5.8123322D−01
5	5	1.0482486D−01	1.0482486D−01
5	10	1.7541753D−02	1.7541750D−02
10	0	6.2735311D−01	6.2729503D−01
10	5	1.4833812D−01	1.4833806D−01
10	10	3.2697995D−02	3.2697984D−02

a singularity at $s_2 = -\infty$; see [1] for a detailed explanation. To improve accuracy, we increase both N_1 and N_2 by factors of 2 (which increased computation time by about 5–6 times) used the epsilon algorithm in both dimensions and increased b_2 to 5. All these steps improved accuracy to a level that should be satisfactory for most applications, but the final accuracy is still less than that of Example 6.1, as can be seen from Table 4.

Remark 6.1. It appears that if the transform does not have a singularity at $s_i = -\infty$ for $i = 1$ or 2, then the Laguerre method with our efficient two-dimensional FFT-based implementation would clearly be the method of choice. Otherwise, the Fourier-series method, which is more robust, would be preferable. In case the transform is badly behaved, instead of trying to fix the Laguerre method, as we do in Example 6.3, a better approach might be to change the transform. In Example 6.3, if we replace the gamma service-time distribution by a distribution with a rational Laplace transform that matches the first several moments (e.g., the H_2 distribution can match the first three moments), then the Laguerre method would behave just as well as in Example 6.1.

7. Summary of the algorithm

As with the one-dimensional algorithm in [1, Section 10], we conclude this paper by summarizing the algorithm. We describe the FFT variant, using the enhancements described in Sections 4.1–4.3. Since the further refinements here parallel those in [1], we refer to [1, Section 10], for a summary of further refinements.

Basic FFT-based algorithm

Step 1: Compute and store the approximate Laguerre coefficients $\bar{q}_{n_1,n_2}$ *for* $0 \le n_1 \le N_1 - 1$ *and* $0 \le n_2 \le N_2 - 1$. First, for $i = 1, 2$ specify the parameter N_i as powers of 2, e.g., $N_1 = N_2 = 128$. Then specify the roundoff-error-control parameters l_i (also as powers of 2), e.g., $l_1 = 1$ and $l_2 = 2$. Then let $m_i = 2l_i N_i$. Choose the parameters A_i to control the aliasing error in (18) and (19), e.g., $A_1 = 11$ and $A_2 = 13$. Then let $r_i = 10^{-A_i/m_i}$ for $i = 1, 2$. Successively compute and store $\{b_{n_1,n_2}\}$, $\{a_{n_1,n_2}\}$

and $\{\bar{q}_{n_1,n_2}\}$ via (23), (25)–(26) and (22), where $Q(z_1, z_2)$ is obtained from the given Laplace transform $\hat{f}(s_1, s_2)$ via (5). (The FFT is used in (25)–(26).)

Step 2: Compute and store the Laguerre function values $l_n(t)$ for $0 \leq n \leq \max\{N_1 - 1, N_2 - 1\}$ *for each required* t. It is convenient to let the set of argument pairs (t_1, t_2) be a product set $T \times T$. Then $l_n(t)$ is needed for each $t \in T$. For each t, the recursion (14) is used.

Step 3: Compute the desired function values $f(t_1, t_2)$ *from* (2).

Step 4: Make an accuracy check. To verify accuracy, repeat the computation with a different pair of roundoff-error-control parameters (l_1, l_2); e.g., if (l_1, l_2) was $(1, 2)$, then repeat the calculation with $(l_1, l_2) = (2, 2)$.

References

[1] J. Abate, G.L. Choudhury and W. Whitt, On the Laguerre method for numerically inverting Laplace transforms, *Informs J. Comput.* **8** (1996) 413–427.

[2] J. Abate and W. Whitt, Solving probability transform functional equations for numerical inversion, *Oper. Res. Lett.* **12** (1992) 275–281.

[3] G.L. Choudhury and D.M. Lucantoni, Numerical computation of the moments of a probability distribution from its transform, *Oper. Res.* **44** (1996) 368–381.

[4] G.L. Choudhury, D.M. Lucantoni and W. Whitt, Multidimensional transform inversion with applications to the transient M/G/1 queue, *Ann. Appl. Probab.* **4** (1994) 719–740.

[5] G.L. Choudhury, D.M. Lucantoni and W. Whitt, Numerical solution of $\mathrm{M}_t/\mathrm{G}_t/1$ queues, *Oper. Res.* **45** (1997), to appear.

[6] G.L. Choudhury and W. Whitt, Computing distributions and moments in polling models by numerical transform inversion, *Performance Evaluation* **25** (1996) 267–292.

[7] G.L. Choudhury and W. Whitt, Probabilistic scaling for the numerical inversion of non-probability transforms, *Informs. J. Comput.* **9** (1997), to appear.

[8] V.A. Ditkin and A.P. Prudnikov, *Operational Calculus in Two Variables and Its Applications*, 2nd ed., Academic Press, New York (1962).

[9] B.S. Garbow, G. Giunta, J.N. Lyness and A. Murli, Software for an implementation of Weeks' method for the inverse Laplace transform problem, *ACM Trans. Math. Software* **14** (1988) 163–170.

[10] B.S. Garbow, G. Giunta, J.N. Lyness and A. Murli, Algorithm 662: A FORTRAN software package for numerical inversion of the Laplace transform based on Weeks' method, *ACM Trans. Math. Software* **14** (1988) 171–176.

[11] M.V. Moorthy, Inversion of the multi-dimensional Laplace transform – expansion by Laguerre series, *Z. Angew. Math. Phys.* **46** (1995) 793–806.

[12] M.V. Moorthy, Numerical inversion of two-dimensional Laplace transforms – Fourier series representation, *Appl. Numer. Math.* **17** (1995) 119–127.

[13] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliff, NJ (1975).

[14] W. Rudin, *Real and Complex Analysis*, McGraw-Hill, New York (1966).

[15] U. Sumita and M. Kijima, The bivariate Laguerre transform and its applications: Numerical exploration of bivariate processes, *Adv. in Appl. Probab.* **17** (1985) 683–708.

[16] G. Szegö, *Orthogonal Polynomials*, 4th ed., Amer. Math. Soc. Colloq. Publ., Vol. 23, AMS, Providence, RI (1975).

[17] B. Van der Pol and H. Bremmer, *Operational Calculus*, Cambridge University Press, Cambridge (1955) (reprinted in 1987 by Chelsea Press, New York).

[18] W.T. Weeks, Numerical inversion of Laplace transforms using Laguerre functions, *J. ACM* **13** (1966) 419–426.

[19] J. Wimp, *Sequence Transformations and Their Applications*, Academic Press, New York (1981).

[20] P. Wynn, On a device for computing the $e_m(S_n)$ transformation, *Math. Tables Aids Comput.* **10** (1956) 91–96.

[21] P. Wynn, On the convergence and stability of the epsilon algorithm, *SIAM J. Numer. Anal.* **3** (1966) 91–122.

Ward Whitt received the A.B. degree in Mathematics from Dartmouth College, Hanover, NH, USA, in 1964 and the Ph.D. degree in Operations Research from Cornell University, Ithaca, NY, USA, in 1969. He was on the faculty of Stanford University and Yale University before joining AT&T Laboratories in 1977. He is currently a member of the Network Mathematics Research Department in AT&T Labs-Research in Florham Park, NJ, USA. His research has focused on probability theory, queueing models, performance analysis and numerical transform inversion.

Gagan L. Choudhury received the B.Tech. degree in Radio Physics and Electronics from the University of Calcutta, India in 1979 and the MS and Ph.D. degrees in Electrical Engineering from the State University of New York (SUNY) at Stony Brook in 1981 and 1982, respectively. Currently he is a Technical Manager at the Teletraffic Theory and System Performance Department in AT&T Laboratories, Holmdel, New Jersey, USA. His main research interest is in the development of multi-dimensional numerical transform inversion algorithms and their application to the performance analysis of telecommunication and computer systems.

Joseph Abate received the B.S. degree from the City College of New York in 1961, and the Ph.D. degree in Mathematical Physics from New York University in 1967. From 1966 to 1970 he was employed by Computer Applications Corporation where he specialized in evaluating the performance of real-time computer systems. In 1971, he joined AT&T Bell Laboratories and retired in 1990. For most of his career, he worked on the design and analysis of transaction processing systems used in support of telephone company operations. For many years he has had a great passion for the use of Laplace transforms in queueing problems.

Currently, he is a contractor to Lucent Technologies in Liberty Corner, New Jersey.