Electronic Transactions on Numerical Analysis.
Volume 39, pp. 403-413, 2012.
Copyright © 2012, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

INTEGRATING OSCILLATORY FUNCTIONS IN MATLAB, II[*]

L. F. SHAMPINE[†]

Abstract. In a previous study we developed a MATLAB program for the approximation of $\int_a^b f(x)\,e^{i\omega x}\,dx$ when ω is large. Here we study the more difficult task of approximating $\int_a^b f(x)\,e^{ig(x)}\,dx$ when $g(x)$ is large on $[a,b]$. We propose a fundamentally different approach to the task— backward error analysis. Other approaches require users to supply the location and nature of critical points of $g(x)$ and may require $g'(x)$. With this new approach, the program `quadgF` merely asks a user to define the problem, i.e., to supply $f(x)$, $g(x)$, $[a,b]$, and specify the desired accuracy. Though intended only for modest relative accuracy, `quadgF` is very easy to use and solves effectively a large class of problems. Of some independent interest is a vectorized MATLAB function for evaluating Fresnel sine and cosine integrals.

Key words. quadrature, oscillatory integrand, regular oscillation, irregular oscillation, backward error analysis, Filon, Fresnel integrals, MATLAB

AMS subject classifications. 65D30, 65D32, 65D07

1. Introduction.

In a previous study [11] we considered the approximation of

$$I(f,\omega x) = \int_a^b f(x)\,e^{i\omega x}\,dx \tag{1.1}$$

for real $f(x)$ on a finite, real interval $[a,b]$ when the real parameter ω is large in magnitude and presented an effective MATLAB [9] program called `osc` for approximating $I(f,\omega x)$. Here we investigate the more general problem

$$I(f,g) = \int_a^b f(x)\,e^{ig(x)}\,dx \tag{1.2}$$

for real $g(x)$ that is large on $[a,b]$. The integrand of (1.2) is often called an *irregular* oscillation to distinguish it from the *regular* oscillation of (1.1). Many authors write $g(x)$ as $\omega\,G(x)$ so as to facilitate an asymptotic analysis as $\omega\to\infty$. We prefer $g(x)$ because it leads to a simpler user interface for a new MATLAB program, `quadgF`, that we develop here for integrals of the form (1.2).

Regular oscillatory problems (1.1) are difficult for conventional quadrature methods when ω is large because the exponential factor oscillates rapidly. A conventional method must take many samples of the integrand to resolve this behavior. This is at best expensive. It is also dangerous because if the samples taken do not represent adequately the behavior of the integrand, the method might produce a poor approximation that is not recognized as being unsatisfactory because the error estimate is also poor. We provide an example of this in Section 5. Filon [5] was the first to propose an effective scheme for such problems. He approximates $f(x)$ on $[a,b]$ by dividing the interval into pieces of equal length and forming a continuous spline $S(x)$ by interpolating $f(x)$ at the ends and middle of each piece with a quadratic polynomial. An approximation to $I(f,\omega x)$ is then obtained by evaluating *analytically*

$$Q(f,\omega x) = \int_a^b S(x)\,e^{i\omega x}\,dx. \tag{1.3}$$

[*] Received February 29, 2012. Accepted August 10, 2012. Published online on November 6, 2012. Recommended by K. Burrage.

[†]Mathematics Department, Southern Methodist University, Dallas, TX (`lfshampine@aol.com`)

The FSER1 program of Chase and Fosdick [2] is based on this method. They implement it as an iterative, non-adaptive scheme that attempts to approximate (1.1) to a specified accuracy. Bakhvalov and Vasil'eva [1] suggest high order interpolation at Gaussian nodes. Piessens and Branders [10] use high order interpolation at Chebyshev nodes in their program AINOS. The high order and their adaptive implementation of the formula make this a strong program for the task. Iserles and Nørsett [7] show that smoother $S(x)$ have an advantage when ω is large. The `osc` [11] program is an adaptive implementation of a formula based on a smooth cubic spline.

It is *much* harder to deal with irregular oscillations because there may be critical (stationary) points where $g'(x) = 0$. Near such a point the exponential factor in the integrand of (1.2) is not oscillatory, changing the nature of the task. A critical point causes some methods proposed for irregular oscillations to fail. For instance, a number of authors, e.g., [4, 13], use the transformation $x = g^{-1}(y)$ to reduce an irregular oscillation to a regular oscillation, a transformation that breaks down at a critical point. Assuming that moments $\int_a^b x^k\, e^{i\,g(x)}\, dx$ are readily available for $k = 1, 2, \ldots$, Iserles and Nørsett [7] and others generalize the Filon method (1.3) to an irregular oscillation as

$$Q(f, g) = \int_a^b S(x)\, e^{i\,g(x)}\, dx.$$

Although applicable to problems with critical points, this is such a strong assumption about $g(x)$ that the approach is unsuitable for general-purpose software. A more practical way to generalize Filon's method for (1.2) is to approximate $g(x)$ by a spline $s(x)$ and then integrate analytically

$$Q(f, g) = \int_a^b S(x)\, e^{i\,s(x)}\, dx. \tag{1.4}$$

However, even for piecewise polynomial $s(x)$, the necessary integrals may not be readily available. Evans [3] suggests approximating both $f(x)$ and $g(x)$ with linear functions. Harris and Chen [6] are primarily interested in integrals involving two independent variables, but in the case of one independent variable, they also suggest linear functions. We have modified `osc` to use a linear spline for $s(x)$. Our experience has been good with this modification, but a higher degree polynomial provides a more plausible approximation of $g(x)$ near critical points. Evans [3] generalizes Filon's method for regular problems by approximating both $f(x)$ and $g(x)$ with quadratics. The necessary integrals can be expressed in terms of Fresnel integrals, and he goes on to explain how to evaluate these special functions. Evans has shown how to deal with a general quadratic approximating $g(x)$, but no one has shown how to deal with general polynomials of higher degree. Accordingly, our new program `quadgF` is based on piecewise-quadratic interpolants $S(x)$ and $s(x)$ in (1.4). Experiments with the approach reported by Evans [3] and extensive experiments with the software `quadgF` show this to be a good approach to integrals of the form (1.2) when modest accuracy is sufficient.

Previous theoretical work on irregular oscillations has assumed that $g(x)$ has either no critical points at all, or an end point is a critical point of known order. We propose a fundamentally different approach to the integration of irregular oscillations—backward error analysis. Just as with solving systems of linear algebraic equations, this new way of looking at the task separates difficulties associated with the problem from difficulties associated with the numerical method. A very important result is that we can solve problems in a meaningful way without having to ask users about the location and nature of critical points. An important practical matter is that the backward error can be estimated in a simple and reliable manner.

ETNA
Kent State University
http://etna.math.kent.edu

Furthermore, Evans [3] notes that quadratic interpolants that degenerate to linear polynomials or even to constants present numerical difficulties in the evaluation of (1.4). A backward error analysis provides a simple and reliable way to recognize and deal with this issue.

The new program is remarkably easy to use: $I(f, g)$ is approximated with default error tolerance by `Q = quadgF(fun,gfun,a,b)`. Here `fun` and `gfun` are procedures for evaluating $f(x)$ and $g(x)$, respectively. It is convenient in both theory and practice to approximate the complex integral and then obtain trigonometric integrals from the real and imaginary parts of the result,

$$I(f,g) = \int_a^b f(x)\,\cos(g(x))\,dx + i\int_a^b f(x)\,\sin(g(x))\,dx, \tag{1.5}$$

so `quadgF` always computes the complex integral.

MATLAB does not provide functions for the direct evaluation of the Fresnel sine and cosine integrals that are the foundation of the generalized Filon method we study here. We have translated a Fortran program for the task [14] to MATLAB and vectorized it. The resulting program, called `fresnel`, is of some independent interest.

2. Error analysis and estimation.

The idea of backward error analysis is to view the numerical approximation as the exact solution of a problem close to the given problem. In this view the task is to estimate and control the error in approximating the *problem*. Generalized Filon methods for regular oscillations can obviously be viewed in this way—the numerical approximation is the exact integral of the approximating problem (1.3). The backward error is a measure of the difference between $S(x)$ and $f(x)$. Most of the investigations of regular oscillations use (perhaps implicitly) the L^∞ norm to measure this error. It is well-known that a good approximation in this sense is furnished by interpolation at Chebyshev nodes, which is one reason why Piessens and Branders [10] do this in their program AINOS. However, an L^2 norm is equally plausible, and it is also well-known that a good approximation is then furnished by interpolation at Gaussian nodes, as done by Bakhvalov and Vasil'eva [1].

As it happens, the analysis of generalized Filon methods for a regular oscillation is quite special because a forward error analysis is to hand. A forward error analysis bounds the error of the numerical approximation itself. We did this in our discussion of the `osc` program [11], which approximates $f(x)$ with a smooth cubic spline $S(x)$. There are standard results about $\|f - S\|$ in L^∞ that we can use to bound the forward error with

$$|I(f,\omega x) - Q(f,\omega x)| \le \int_a^b |f(x) - S(x)|\,|e^{i\omega x}|\,dx \le \|f - S\|\,|b - a|.$$

We see that when approximating a regular oscillation with a generalized Filon method, controlling the forward error is essentially the same as controlling the backward error. It is easy to prove the same for the L^2 norm using the Cauchy-Schwarz inequality.

We have no need of a backward error analysis in the regular case, but it is very advantageous in the irregular case. The generalized Filon methods we consider for (1.2) approximate the integral with the exact integral of the approximating problem (1.4). Accordingly, we estimate and control the error of approximating $f(x)$ by $S(x)$ and $g(x)$ by $s(x)$. In this way we avoid entirely the issue of critical points. We prefer an L^2 norm because it describes better the behavior of the function and it is easy then to obtain a good estimate of the error.

Although any standard quadrature scheme might be used to estimate the integral norms of the backward error, we have developed a new formula tailored to the circumstances. The `quadgF` program is an adaptive implementation of the generalized Filon method that approximates both $f(x)$ and $g(x)$ with interpolating quadratics on subintervals $[x_m, x_{m+1}]$ of

$[a, b]$. To estimate the backward error, we must therefore approximate integrals of the form

$$\int_{x_m}^{x_{m+1}} (F(x) - Q(x))^2 \, dx = \int_{x_m}^{x_{m+1}} H(x) \, dx, \tag{2.1}$$

where $F(x)$ is either $f(x)$ or $g(x)$ and $Q(x)$ is a quadratic that interpolates $F(x)$ at both ends and the midpoint of the interval. We can obtain a good approximation with a small number of samples by taking into account the qualitative behavior of $H(x)$, namely that it vanishes along with its first derivative at both ends of the interval and the midpoint. The new scheme uses the samples formed in defining $Q(x)$ and two additional samples at points that are convenient for an adaptive implementation. If we let $h_m = x_{m+1} - x_m$, it is an easy matter to derive a quadrature formula based on interpolation to value and slope at $x_m, x_m + h_m/2, x_m + h_m$ and to value at $x_m + h_m/4, x_m + 3h_m/4$. By construction, this quadrature formula has degree of precision 7. When applied to $H(x)$ the formula has a very simple form because of the values known to be zero, namely

$$\int_{x_m}^{x_m+h_m} H(x) \, dx \approx \frac{256}{945} h_m \left[H(x_m + h_m/4) + H(x_m + 3h_m/4) \right]. \tag{2.2}$$

It is easy to vectorize the application of this formula to all pertinent subintervals, a matter explained more fully in Section 3. This is a simple and inexpensive way to compute a good estimate of the backward error.

Dealing with the scale of $f(x)$ and $g(x)$ is an important practical aspect of backward error analysis. The integral (1.2) and the generalized Filon method are linear in $f(x)$, so multiplying $f(x)$ by a constant is unimportant to both, but it does affect the way we measure backward error. Something similar is true of $g(x)$. If we write $g(x) = \omega G(x)$, the accuracy of the method depends on how well we approximate $G(x)$ because the generalized Filon approach deals analytically with the effects of large ω. However, the size of ω does play a role when we ask how well $s(x)$ approximates $g(x)$. Clearly we need somehow to scale our definition of the size of the backward error. With tolerance τ we can do this by testing whether

$$\|F(x) - Q(x)\| \leq \tau \, \|F(x)\| \tag{2.3}$$

for both $f(x)$ and $g(x)$. As explained in Section 3, we use a more conservative test of this kind in `quadgF` because it is more convenient for our adaptive implementation of the method.

3. Adaptive quadrature.

We provide an overview of our adaptive implementation of the generalized Filon method in this section and provide details of the most important algorithms in later sections. Several popular adaptive schemes, and in particular the scheme implemented in `osc`, are described in [11]. Our scheme for `quadgF` is more efficient because more of the scheme is vectorized. As usual with adaptive quadrature, the interval $[a, b]$ is partitioned into subintervals $[x_m, x_{m+1}]$ and the integral (1.2) is approximated by summing generalized Filon approximations

$$Q_m = \int_{x_m}^{x_{m+1}} S(x) \, e^{is(x)} \, dx \tag{3.1}$$

over the subintervals. Our scheme maintains a list of active subintervals for which we do not yet have a sufficiently accurate approximation to $f(x)$ and $g(x)$. `osc` processes intervals one at a time, but `quadgF` processes them simultaneously. Each iteration begins with a list of active subintervals and all the function values needed. The program then calculates quadratic approximations $S(x)$ to $f(x)$ and $s(x)$ to $g(x)$ on all of the active intervals and uses (2.2) to estimate the backward errors. If both approximations on a subinterval are sufficiently accurate, the approximate integral (3.1) is formed as described in Section 4.1. This value is added to a running total that approximates $I(f,g)$ and the subinterval is then of no further interest. If one of the approximations on $[x_m, x_{m+1}]$ is not sufficiently accurate, the subinterval is replaced in a new active list with the two subintervals $[x_m, 0.5(x_m + x_{m+1})]$ and $[0.5(x_m + x_{m+1}), x_{m+1}]$. When all subintervals have been considered, either the new active list is empty and we have an accurate approximation to $I(f,g)$, or we replace the old active list with the new one. In this we reuse many function values, but some additional function values are formed to complete preparations for the next iteration. It is worth remark that in our adaptive scheme, *all* the function evaluations made in a call to `quadgF` are used in forming the approximation returned for $I(f,g)$ or in the estimate of its backward error.

Key to our implementation is recognizing that an approximation is accurate enough that we can form (3.1) and then forget about that subinterval. In the notation of Section 2, we require that

$$\int_{x_m}^{x_{m+1}} (F(x) - Q(x))^2 \, dx \le \tau^2 \|F(x)\|^2 \left(\frac{x_{m+1} - x_m}{b - a} \right) \tag{3.2}$$

for both $f(x)$ and $g(x)$. Passing this local test on all subintervals of $[a,b]$ implies that the approximations pass the global test (2.3), so this is a somewhat more conservative way to control the error. For the test (3.2) we can obtain a reasonable estimate of the general size of $F(x)$ for "free" by applying Simpson's rule on each $[x_m, x_{m+1}]$ to approximate

$$\|F(x)\|^2 = \sum_{m=1}^{N-1} \int_{x_m}^{x_{m+1}} F^2(x) \, dx.$$

All the values of $F(x)$ needed for this are formed when $Q(x)$ is defined. The test (3.2) copes with the scales of the functions; but, as with any relative test, we must consider the possibility that a function vanishes. An integral (1.2) with $f(x) \equiv 0$ would not be the subject of numerical approximation. Accordingly, if `quadgF` computes an approximation of $\int_a^b f^2(x)\,dx$ that is zero, it returns with an error message: *F(X) vanishes at all X considered*. An irregular oscillation with $g(x) \equiv 0$ is a conventional integral and so of little interest in the present context. As with $f(x)$, `quadgF` returns with an error message if it appears that $\int_a^b g^2(x)\,dx$ is zero.

Any quadrature formula that uses only a finite number of samples of the integrand might fail if the samples are not representative. For this reason `quadgF` initializes with a relatively large number of samples and includes the following warning in its prolog: *Any quadrature program can be deceived if the initial samples do not reveal the behavior of the integrand.* `QUADGF` *initializes with samples of $f(x), g(x)$ at 129 equally spaced points in $[A, B]$. If $f(x)$ or $g(x)$ has sharp peaks or is highly oscillatory, it might be necessary to write the integral as the sum of integrals over subintervals of $[A, B]$ chosen so that the behavior of $f(x), g(x)$ is captured by the initial samples from those subintervals.*

Certain programming practices are very important to efficiency in MATLAB. Indeed, vectorization of functions is so important that all the quadrature programs of MATLAB *require* the functions for evaluating integrands to be vectorized. This means that the function for evaluating, say, $f(x)$, must be coded so that when called with a row vector `x`, it returns a row vector `fx` of corresponding function values. Using array operations and fast built-in functions, it is often the case that evaluating $f(x)$ at 129 points as in the initialization of `quadgF` has a cost comparable to evaluating the function at just a few points when it is not vectorized. An important difference between the schemes of `osc` and `quadgF` is that the latter evaluates $f(x)$ and $g(x)$ at all the necessary points for the entire active list in a single call. We need values of the functions not just at the ends and midpoint of each subinterval to form quadratic interpolants, but also at the two additional points for estimating the error. For the error estimate of (2.2) this means that we have 5 equally spaced points in the subinterval. When the active list is initialized with 32 subintervals of equal length, the subintervals are contiguous. We use this fact to reduce the number of function evaluations, leading to a total of 129 evaluations. If the approximation is not sufficiently accurate on a subinterval $[x_m, x_{m+1}]$, so that we split the interval in half for the new active list, the nature of our formulas means that we have available 6 of the 10 function values that we need for the two new subintervals. That is, all we have to do is form the additional values needed for the two error estimates.

We use a relatively large number of samples in the initialization of `quadgF` for reliability. This also makes the program fast because it is inexpensive to form all these values at the same time and often this is sufficiently many samples that there is no need for further refinement of the mesh. As a consequence, the program might well approximate $f(x)$ and $g(x)$ far better than required by the tolerance specified.

4. Evaluation of integrals.

Evans [3] explains in detail how he evaluates the basic integrals (3.1). Our scheme differs in some important respects. He works entirely with real integrals, and because software for evaluating Fresnel sine and cosine integrals was not available, he presented algorithms for the purpose. We work with complex quantities because this is more convenient and it is easily done in MATLAB. Evans notes that his scheme breaks down when a quadratic interpolant degenerates to a linear interpolant and also when it degenerates to a constant. He presents algorithms for the degenerate cases, but does not explain how to recognize an interpolant that is degenerate or nearly so. In Section 4.1 we show how backward error analysis can be used to recognize degeneracy. We also provide there details of efficient analytical evaluation of the integrals in the three cases. MATLAB does not include software for Fresnel integrals, so in Section 4.2 we consider how to evaluate them efficiently in this computing environment. The `fresnel` function presented there accepts a vector argument x and returns corresponding vectors of both the cosine and sine integrals.

4.1. Algorithms.

Vectorization and array operations are very important to efficiency in MATLAB. The `quadgF` program is more efficient in this regard than the `osc` program for the regular problem because we process at one time all the intervals where $f(x)$ and $g(x)$ can be approximated sufficiently well by quadratic interpolants. Although we describe the algorithms here for a single interval, they are easily vectorized. Indeed, this is little more than replacing a multiplication (`*`) with an array multiplication (`.*`), an exponentiation (`^`), with an array exponentiation, (`.^`), and the like. In this we also exploit the fact that all the standard functions like “`exp`” are vectorized. The fast built-in function `find` is used to identify all active subintervals with a given property. For example, error estimates are computed for all the subintervals at the same time using array operations. The subintervals where the approximations are sufficiently accurate are then identified with `find` for further processing.

We now consider in detail how to evaluate the approximation (3.1) when $S(x)$ is a quadratic that interpolates $f(x)$ at the ends of the interval and the midpoint $x_{m+1/2}$ and $s(x)$ is a quadratic interpolant to $g(x)$ at the same points. We write these interpolants as $s(t) = a_1 + a_2 t + a_3 t^2$ and $S(x) = b_1 + b_2 t + b_3 t^2$ for $t = x - x_m$. With $h = x_{m+1} - x_m$ and the notation $f_j = f(x_j)$, the integral is

$$Q_m = \int_0^h S(t)\, e^{is(t)}\, dt.$$

The coefficients of $S(t)$ are

$$b_1 = f_m, \quad b_2 = \frac{4f_{m+1/2} - 3f_m - f_{m+1}}{h}, \quad b_3 = \frac{2f_m - 4f_{m+1/2} + 2f_{m+1}}{h^2},$$

and similarly for $s(t)$.

The backward error approach provides a natural way to recognize degenerate and near-degenerate approximations. We first ask if we can use the constant approximation $s(t) \approx a_1$ on $[0, h]$ by testing whether

$$\|s(t) - a_1\| \leq 10^{-6}\, \|s(t)\|.$$

The norms here are calculated analytically. The constant in the test is an order of magnitude smaller than the smallest tolerance allowed in `quadgF`. If the test is passed, the integral is

$$Q_m = e^{ia_1} \sum_{j=1}^{3} b_j \int_0^h t^{j-1}\, dt = e^{ia_1} \sum_{j=1}^{3} b_j\, \frac{h^j}{j}.$$

If $s(t)$ cannot be approximated well by a constant, we ask if it can be approximated well by a straight line. Specifically, we test whether

$$\|s(t) - (a_1 + a_2 t)\| \leq 10^{-6}\, \|s(t)\|.$$

If so, a_2 is not extremely small and the integral is

$$Q_m = e^{ia_1} \sum_{j=1}^{3} b_j \int_0^h t^{j-1}\, e^{ia_2 t}\, dt = e^{ia_1} \sum_{j=1}^{3} b_j\, I_j.$$

Letting $\mu = e^{ia_2 h}$, the I_j are evaluated successively as

$$I_1 = \frac{1}{ia_2}\, [\mu - 1], \quad I_2 = \frac{1}{ia_2}\, [h\,\mu - I_1], \quad I_3 = \frac{1}{ia_2}\left[h^2\,\mu - 2I_2\right].$$

In the typical case, we have $s(t) = a_1 + a_2 t + a_3 t^2$ and we have verified that $s(t)$ is not approximated well by a constant nor by a straight line. Accordingly, a_3 is not extremely small. A key step in Evans' analysis is to complete the square,

$$\begin{gathered} s(t) = c_1 + c_3 (t + c_2)^2 \qquad \text{where} \\ c_1 = a_1 - \frac{a_2^2}{4a_3}, \qquad c_2 = \frac{a_2}{2a_3}, \qquad c_3 = a_3. \end{gathered}$$

We introduce the variable $y = t + c_2$ so that $s(y) = c_1 + c_3 y^2$. Correspondingly, we rewrite $S(t)$ as $S(y) = d_1 + d_2 y + d_3 y^2$. The change of variable leads to

$$Q_m = e^{ic_1} \sum_{j=1}^{3} d_j \left[\int_0^{h+c_2} y^{j-1}\, e^{ic_3 y^2}\, dy - \int_0^{c_2} y^{j-1}\, e^{ic_3 y^2}\, dy \right].$$

The integrals here have the form

$$\int_0^u y^{j-1}\, e^{i\, c_3 y^2}\, dy \tag{4.1}$$

for real u, real c_3, and positive integer j. They can be evaluated by first using `fresnel` to compute

$$\rho = \sqrt{0.5\pi / |c_3|}, \qquad v_{0c} = \rho C(|u| / \rho), \qquad v_{0s} = \rho S(|u| / \rho),$$
$$\int_0^u e^{ic_3 y^2}\, dy = \mathrm{sign}(u)\left[v_{0c} + i\,\mathrm{sign}(c_3)\, v_{0s}\right].$$

We can then compute

$$\tau = e^{i|c_3|u^2}, \qquad v_{1c} = 0.5\,\Im(\tau) / |w|, \qquad v_{1s} = \sin(0.5|c_3|u^2)^2 / |c_3|,$$
$$\int_0^u y\, e^{ic_3 y^2}\, dy = v_{1c} + i\,\mathrm{sign}(c_3)\, v_{1s}.$$

And finally

$$v_{2c} = |u|\, v_{1c} - 0.5 v_{0s} / |c_3|, \qquad v_{2s} = (v_{0c} - 0.5|u|\,\Re(\tau)) / |c_3|,$$
$$\int_0^u y^2\, e^{ic_3 y^2}\, dy = \mathrm{sign}(u)\left[v_{2c} + i\,\mathrm{sign}(c_3)\, v_{2s}\right].$$

4.2. Fresnel integrals. With the aid of the symbolic algebra program MuPAD [9] we find that the integrals (4.1) can be evaluated in terms of the incomplete gamma function,

$$\int_0^u y^{j-1}\, e^{ic_3 y^2}\, dy = \mathrm{sign}(u)^j \int_0^{|u|} y^{j-1}\, e^{ic_3 y^2}\, dy$$
$$= \frac{\mathrm{sign}(u)^j}{2(-ic_3)^{j/2}} \left[\Gamma\left(\frac{j}{2}\right) - \Gamma\left(\frac{j}{2}, -ic_3 u^2\right)\right].$$

The MATLAB function for evaluating the incomplete gamma function does not allow complex arguments, but the one that is accessed through the symbolic engine does. Though slower, it is fast enough to be practical and it is easy enough to use this function via the gateway `mfun`. Indeed, that function for $\Gamma(a, z)$ is vectorized with respect to z, which is quite helpful in our program. Evans [3] works in real arithmetic and evaluates the equivalent of (4.1) in terms of the Fresnel cosine integral $C(x)$ and sine integral $S(x)$. Because the functions were not widely available, he explained how to evaluate them. Using MuPAD we found the expressions for the integrals in terms of Fresnel integrals that we provide in Section 4.1. Fresnel integrals are not available in MATLAB itself, but they are available through the symbolic engine. The authors of *Computation of Special Functions* [14] provide Fortran programs for the functions they discuss. We translated their FCS.FOR program for Fresnel integrals to the MATLAB language *and vectorized it*. We have verified that the results of our function, `fresnel`, agree to nearly machine precision with results computed using the symbolic engine of MATLAB. As it happens, it is convenient to evaluate both integrals at the same time, so in a call `[C,S] = fresnel(x)`, the function accepts a vector argument x and returns corresponding vectors $C(x)$ and $S(x)$. This is helpful in the present application because we need values of both integrals at the same argument. Using `fresnel` in MATLAB is faster than using `mfun` and the symbolic engine. Just how much faster depends on the length of `x` and whether you want both integrals, but an order of magnitude is representative. Accordingly, the function `fresnel` is of some independent interest.

5. Illustrative computations.

The prolog of `quadgF` contains four examples that illustrate aspects of its user interface and capabilities. Reference values for the integrals are provided along with the examples. `quadgF` approximates (1.2) in the sense of backward error. More precisely, it evaluates analytically an approximation (1.4) with piecewise-quadratic functions $S(x)$ and $s(x)$ that approximate the $f(x)$ and $g(x)$ of (1.2) to a specified relative accuracy in a least-squares sense. Though a meaningful way of solving the problem, this is not the same as computing an approximation to (1.2) with given relative error. Nevertheless, we report here the conventional relative error of the approximations computed. In computing an approximation `Q` to (1.2), the full call list is

```
  Q = quadgF(fun,gfun,a,b,tol)
```

Here `fun` is used to define $f(x)$ and `gfun` is used to define $g(x)$. The interval $[a, b]$ is defined by `a` and `b`. The optional argument `tol` is the tolerance on the relative error in the approximation of $f(x)$ and $g(x)$. It has the default value 10^{-3}. Any value input for `tol` that is smaller than 10^{-9} will be increased to that value. If `quadgF` needs more than 512 active subintervals to obtain the specified accuracy, it returns with an error message.

As it happens, all the examples of the prolog are approximated using only *one* array function evaluation. Because the initial partition provides approximations to $f(x)$ and $g(x)$ that are at least as accurate as the specified tolerance, the program produces precisely the same results for tolerances $10^{-1}, 10^{-2}, \ldots, 10^{-5}$ for each of these four examples.

It is difficult to obtain consistent run times in MATLAB, especially when the times are small, as they are for these four examples. We computed an average run time by using `tic` and `toc` to measure the time elapsed in running the example 500 times. The times reported here were computed using MATLAB R2012a with 64-bit operating system on a PC with an Intel i3-2120 CPU @ 3.30 GHz.

As a convenience when approximating a regular oscillation (1.1), the user need only supply the real scalar ω as the argument `gfun`. With this and an anonymous function for $f(x)$, we can approximate

$$I = \int_0^1 \cosh(x)\, e^{i\,10^5 x}\, dx \tag{5.1}$$

with default relative error tolerance by

```
  Q = quadgF(@(x) cosh(x),1e5,0,1);
```

The first example of the prolog displays the result of this computation as

```
  Example 1: Re(I) = 5.51515e-007, Imag(I) = 2.54209e-005.
             Re(Q) = 5.51515e-007, Imag(Q) = 2.54209e-005.
```

The average run time was 0.0011s.

High order formulas and a vigorous use of vectorization make the `quadgk` program of MATLAB a very effective way to approximate conventional integrals. Indeed, if the oscillation of (1.1) or (1.2) is not too rapid, this program might be the fastest way to compute an approximation. However, as noted in Section 1, there is a real chance that a conventional program might fail when the oscillation is rapid. Indeed, with a pure relative error tolerance of 10^{-3} (which is much less demanding than its default), `quadgk` was not able to approximate (5.1).

The second example of the prolog is an interesting example of Evans [3],

$$\int_{100}^{200} (1 + \log(x))\, \cos(x\,\log(x))\, dx.$$

It shows how $g(x)$ is defined for irregular oscillations. Also, `quadgF` approximates the complex integral (1.5), so we must apply the `real` function to Q to approximate the given integrand. Like all the quadrature programs of MATLAB, `quadgF` requires that the functions for evaluating $f(x)$ and $g(x)$ be vectorized. The program

```
  fun = @(x) 1 + log(x);  gfun = @(x) x.*log(x);
  Q = quadgF(fun,gfun,100,200);  Q = real(Q);
```

produces a result with a relative error of 1.1×10^{-5}. The average run time was 0.0006s.

The third example of the prolog,

$$\int_2^0 e^x \sin(50\cosh(x))\,dx, \tag{5.2}$$

has $b < a$ and is to be solved with a tolerance of 10^{-4}. There is a first-order critical point at one end of the interval, but *we do not inform the program of these facts*. We use the `imag` function to get the result for the sine function. The integral is approximated with

```
  fun = @(x) exp(x);  gfun = @(x) 50*cosh(x);
  Q = quadgF(fun,gfun,2,0,1e-4); Q = imag(Q);
```

The approximation has a relative error of 1.3×10^{-3}. The average run time was 0.0019s. For this integral the initial partition suffices for tolerances down to 10^{-5}, so the approximations and run times do not depend on the tolerance in this range. Table 5.1 shows how the accuracy and run time depend on smaller tolerances.

TABLE 5.1
Performance of `quadgF` for the integral (5.2).

tolerance	1.0e-5	1.0e-6	1.0e-7	1.0e-8	1.0e-9
relative error	1.3e-3	6.4e-5	1.2e-6	1.4e-7	4.8e-9
average run time	1.9e-3	2.5e-3	2.8e-3	3.2e-3	3.9e-3

The last example of the prolog,

$$\int_0^{\pi} \cos(1000\sin(x) - 3x)\,dx,$$

has a first-order critical point at the interior point $\arccos(3/1000) \approx 1.5678$, but *we do not supply this information to the program*. There is no factor $f(x)$ in this integrand, but `quadgF` requires a vectorized function `fun`. This example shows how to deal with the matter. The integral is approximated with

```
  fun = @(x) ones(size(x));  gfun = @(x) 1000*sin(x) - 3*x;
  Q = quadgF(fun,gfun,0,pi); Q = real(Q);
```

The approximation has a relative error of 2.9×10^{-3}. The average run time was 0.0014s.

We have applied `quadgF` successfully to many test problems from the literature. For instance, Li et al. [8] provide three test problems that we have solved with `quadgF` using the default tolerance. The program used two array function evaluations to approximate

$$\int_0^1 e^{-\tan(x)} \frac{\sec(x)}{x + 0.1} e^{i\,100\tan(x)}\,dx$$

with relative error 1.9×10^{-4}. It used one array evaluation to approximate

$$\int_0^1 \frac{e^{10x}}{x + 0.1} e^{i\,200(x^2+x)}\,dx$$

with relative error 5.1×10^{-5}. It used three array evaluations to approximate

$$\int_1^2 \left[\cos(10x^2) + \frac{1}{x + 0.1}\right] e^{i\,(10^7+10^4x^2)^{1/2}}\,dx$$

with relative error 3.7×10^{-4}.

For positive integers p, integrals of the form $\int_0^1 e^{i\,\omega x^p}\,dx$ have a critical point of order $p-1$ at the left end point. Xiang [12] contrasts the performance of a method for the case $p=2$ of a first-order critical point with that for a very high order critical point with $p=10$. The generalized Filon method is exact for the case $p=2$, so we consider here only $p=10$. We changed the interval so as to have an interior critical point. This draws attention to methods which *require* critical points to be end points. Also, it tests whether a code can locate and deal with critical points. It would be natural to use $[-1,1]$, but a critical point at the middle of the interval might prove to be a special case for an adaptive quadrature scheme. To make this less likely, we use as test problem

$$\int_{-1/3}^{2/3} e^{i\,500\,x^{10}}\,dx \approx 0.8437719580 + 0.08517716473i.$$

With no information about the location and nature of the critical point, `quadgF` used only two array function evaluations to obtain an approximation with relative error 1.2×10^{-5}.

6. Conclusions.

Backward error analysis is an important new tool in both the theory and practice of approximating $\int_a^b f(x)\,e^{ig(x)}\,dx$ when $g(x)$ is large on $[a,b]$. Exploiting this approach, we have written a MATLAB program for the task called `quadgF` that does not ask users to supply the location and nature of critical points of $g(x)$, nor does it ask for $g'(x)$. The program is intended only for modest relative error because it is an adaptive implementation of a generalized Filon method of modest order, but it is very easy to use and solves effectively a large class of problems.

MATLAB does not provide functions for the direct evaluation of the Fresnel sine and cosine integrals that are the foundation of the generalized Filon method implemented in `quadgF`. For this purpose we translated the FCS.FOR program of [14] to MATLAB and vectorized it. The resulting program `fresnel` is of some independent interest.

REFERENCES

[1] N. S. BAKHVALOV AND L. G. VASIL'EVA, *Evaluation of the integrals of oscillating functions by interpolation at nodes of Gaussian quadratures*, U.S.S.R. Comput. Math. Math. Phys., 1 (1968), pp. 241–249.

[2] S. M. CHASE AND L. D. FOSDICK, *An algorithm for Filon quadrature*, Comm. ACM, 12 (1969), pp. 453–457.

[3] G. A. EVANS, *Two robust methods for irregular oscillatory integrals over a finite range*, Appl. Numer. Math., 14 (1994), pp. 383–395.

[4] G. A. EVANS AND J. R. WEBSTER, *A comparison of some methods for the evaluation of highly oscillatory integrals*, J. Comput. Appl. Math., 112 (1999), pp. 55–69.

[5] L. N. G. FILON, *On a quadrature formula for trigonometric integrals*, Proc. Roy. Soc. Edinburgh, 49 (1928), pp. 38–47.

[6] P. J. HARRIS AND K. CHEN, *An efficient method for evaluating the integral of a class of highly oscillatory functions*, J. Comput. Appl. Math., 230 (2009), pp. 433–442.

[7] A. ISERLES AND S. NØRSETT, *Efficient quadrature of highly-oscillatory integrals using derivatives*, Proc. R. Soc. Lond. Ser. A. Math. Phys. Eng. Sci.A, 461 (2005), pp. 1383–1399.

[8] J. LI, X. WANG, T. WANG, AND S. XIAO, *An improved Levin quadrature method for highly oscillatory integrals*, Appl. Numer. Math., 60 (2010), pp. 833–842.

[9] MATHWORKS, *MATLAB 7.13 (R2011b)*, Three Apple Hill Drive, Natick MA.

[10] R. PIESSENS AND M. BRANDERS, *Computation of oscillating integrals*, J. Comput. Appl. Math., 1 (1975), pp. 153–164.

[11] L. F. SHAMPINE, *Integrating oscillatory functions in* MATLAB, Int. J. Comput. Math., 88 (2011), pp. 2348–2358.

[12] S. XIANG, *Efficient quadrature for highly oscillatory integrals involving critical points*, J. Comput. Appl. Math., 206 (2007), pp. 688–698.

[13] ———, *On the Filon and Levin methods for highly oscillatory integral* $\int_a^b f(x)\,e^{i\omega g(x)}\,dx$, J. Comput. Appl. Math., 208 (2007), pp. 434–439.

[14] S. ZHANG AND J. M. JIN, *Computation of Special Functions*, Wiley, New York, 1996.