DRAFT: Integration of oscillatory integrals, a computer-algebra approach

Richard Fateman
Computer Science
University of California
Berkeley, CA, USA

November 30, 2012

Abstract

The numerical integration of oscillatory integrals is an important and well-studied area of mathematical inquiry. See [11] for a series of recent conferences on the topic, including applications. Examining some of the proffered methods from the position of a computer algebra system (CAS) provides several opportunities not available with purely numerical approaches. Given an explicit representation of the integrand in symbolic form, one can consider use use of *symbolic derivatives* of components as part of the processing. Additionally, one may utilize a framework for error control based on exact or arbitrarily-high-precision arithmetic. A CAS may provide for simple exposition upon which further elaborations may be built. By executing code in a symbolic system it is also possible to compute approximate answers as expressions in terms of symbolic parameters such as frequency, displaying the nature of dependencies as well as re-useable formulas for different values of parameters.

1 Introduction

Assume that we have a computer algebra system (CAS) at our disposal, and we need numerical or semi-numerical approximations to certain integrals that we can write in this form:

$$I \ = \ \int_a^b e^{i\omega f(x)} g(x) dx.$$

As the real function $f(x)$ varies for x in $[a, b]$, the complex exponential (alternatively the literature may use sines or cosines since $\exp(ix) \ = \ \cos(x) + i\sin(x)$) produces oscillations in the integrand. As is done commonly in this context, we multiply $f(x)$ by a real parameter ω to allow us to demonstrate how the situation changes with frequency. That is, given a particular $f(x)$, an exponent where $\omega = 100$ will have a higher frequency, and be "more oscillatory" than $\omega = 1$. In our subsequent considerations, the situation is that $f(x)$ and $g(x)$ are analytic and vary relatively modestly in the interval $[a, b]$. In some cases it becomes important that f be non-zero in $[a, b]$ or even monotonic. If it is not, the integral may be broken up into pieces where these assumptions hold. Sometimes it is computationally convenient to deal with the real and imaginary parts of the integral separately, but notationally, the complex exponential is more often used in the literature.

This type of integral is problematical for typical quadrature programs since almost any procedure which is based on sampling of the integrand will be subject to the unreliable numerical consequences resulting from computing values of the rapidly varying integrand. Other approaches targeted to oscillatory integrands can be quite successful on these, typically improving as the frequency increases. A readable survey of various approaches from the theoretical and numerical analysis perspective has been composed by Iserles [6].

There are apparently significant applications of this type of integral, for example in investigations of optics, [11]. There are also many papers and computer programs ranging back to work by Filon [4], addressing this issue. One recent computer algebra package by Andrew Moylan [10], has been incorporated into

Mathematica version 8.0's `NIntegrate`[1] command. `NIntegrate` is an impressive collection of many of the standard and recently-developed methods. Unfortunately as apparently is the case with any "automatic" quadrature packages, the user's particular problems may not fit well with the default choices made by the system, and some "poly-algorithm" that reacts to the difficulties may still waste time on a choice that is far from optimal in speed or accuracy. In such cases the user may be required to deal with the mundane aspects of trying to track many options, estimating errors, adjusting precision, etc. Especially in these cases, understanding some of the design ideas behind the user-available choices may be helpful.

In subsequent sections we review a few of the approaches specifically for oscillatory integrals that have occurred in the literature and view each from a simple tutorial perspective to see how these ideas fit in a computer algebra framework. Understanding simple programs may allow the reader to adjust them to particular situations. Rather than dealing with the full complexity of packages intended to account for all eventualities for all inputs, we hope to provide a readable exposition.

2 Filon's method, more or less

The idea from Filon [4] is to approximate $g(x)$ piecewise as a polynomial, each section looking like $g_0+g_1x+g_2x^2$.

Instead of integrating $g(x)$ using Simpson's Rule, consider that you can integrate $x^k\exp(i\omega f(x))$ for $k=0,1,2$. This gives you alternatives for the Simpson's Rule weights, and the integral can be written as the (piecewise) sum of the modified Simpson's Rule. If we were actually integrating a quadratic polynomial times an exponential, we could write it out:

$$\frac{e^{i\,\omega\,x}}{\omega^3}\left(\left(-g_2\,\omega^2\,x^2-g_1\,\omega^2\,x-g_0\,\omega^2+2\,g_2\right)\,i+2\,g_2\,\omega\,x+g_1\,\omega\right)$$

Of course you may not start with the coefficients in that polynomial but only its value at several points. Three points determine a quadratic, and thus the polynomial can be computed by interpolation. As an example, the interpolation algebra works out simply if you have values of g at the points $-1,\ 0,\ +1$ in which case

$$g_0=g\left(0\right),g_1=\frac{2\,g\left(1\right)-2\,g\left(-1\right)}{4},g_2=-\frac{-g\left(1\right)+2\,g\left(0\right)-g\left(-1\right)}{2}$$

There are advantages to alternative techniques in accuracy and speed if (more) points are not equally spaced, but chosen instead to facilitate Gaussian quadrature.

In any case, details are then encoded in numerical routines. The result is to entirely eliminate the problematic oscillatory term from the quadrature. Programs for Filon integration for $f(x)=x$ can be found in scientific subroutine libraries. One precaution of note included in the ACM library implementation is that it evaluates certain subexpressions by an alternative series calculation less subject to cancellation error than the most obvious formula. A notationally simpler (but perhaps slower) technique to avoid accumulation of this error which is available in a CAS, is to simply increase the floating-point precision temporarily. There are numerous tweaks possible on this idea.

The QUADPACK library routine QAWO[2] and related programs can also be used. These use a "modified Clenshaw-Curtis" quadrature scheme. The adaptation provides that the function $g(x)$ is approximated by a single Chebyshev series approximated over the whole range.

We provide a version of Filon's procedure using the Maxima computer algebra system to illustrate its brevity. It is based on a FORTRAN program by Chase and Fosdick[1][3]. This is the code for the imaginary part $\sin(\omega x)$; similar code for $\cos(\omega x)$ computes the real part; combined they compute the code for $\exp(i\omega x)$.

```
filons0(f,m,p):=  /*integrate f(x)*sin(m x) from 0 to 1, p points.
```

[1]For documentation, see `http://reference.wolfram.com/mathematica/tutorial/NIntegrateIntegrationRules.html`

[2]Available at http://www.netlib.org, and included in the CAS Maxima.

[3]We do not display lengthier "optimized" code which computes arrays of sines/cosines by differencing, and/or uses faster machine floats rather than higher-precision software floats, deals with cosines rather than sines, uses general endpoints, attempts to estimate error, etc.

```
  for notation, see CACM Algorithms: Chase & Fosdick Alg 353  */
block([h:bfloat(1/(2*p)),
       k:bfloat(m), theta,s2p,s2pm1,alf,bet,gam],

  theta:bfloat(k*h),
  s2p:
   block([sum:0],
    for i:0 thru p do sum:sum+f(2*h*i)*sin(2*k*h*i),
     sum-1/2*(f(1)*sin(k))),
  s2pm1:
   block([sum:0],
    for i:1 thru p do sum:sum+f(h*(2*i-1))*sin (k*h*(2*i-1)), sum),
  fpprec:2*fpprec, /*double the floating precision, precaution */
  alf: 1/theta + sin(2*theta)/2/theta^2-2*sin(theta)^2/theta^3,
  bet: 2*((1+cos(theta)^2)/theta^2 -sin(2*theta)/theta^3),
  gam: 4*(sin(theta)/theta^3-cos(theta)/theta^2),
  fpprec:fpprec/2,
 h*(alf*(f(0)-f(1))*cos(k) +bet*s2p+ gam*s2pm1))
```

3 What if the argument of sin/cos/exp is complicated?

The case $f(x) = mx$, simply a constant times x, is solved by a direct application of Filon's method or modified Clenshaw-Curtis. If the function $f(x)$ is more complicated than this, we can try to make a substitution (change of variables) [2] to convert $f(x)$ to this simple form. Let $y = f(x)$. Solve for $x = f^{(-1)}(y)$, compute dy and integrate $g(f^{(-1)}(y))e^{i\omega y}dy$ between transformed bounds.

Finding a suitable inverse function can be tricky[4]. Some computer algebra systems provide a command (e.g. Maxima's `changevar`).

Example:

$$\int_a^b e^{i\,w\,\sinh x}\,\cos x\;dx$$

becomes suitable for Filon quadrature after it is changed to

$$\int_{\sinh a}^{\sinh b} \frac{e^{i\,\omega\,y}\;\cos \operatorname{arcsinh}\,y}{\sqrt{y^2+1}}\;dy.$$

It is interesting to see how this change can be used to pre-process out the difficulty and just use an ordinary (non-oscillatory) adaptive quadrature program. Given a simple test of $a = -1,\ b = 1,\ \omega = 1000$, Mathematica 7's default numerical integration program ran on the original problem for 0.8 seconds before signalling failure. On the transformed result the same program produced a result in 0.05 seconds. In each case a result of approximately 1.69206e-4 was obtained, so the error message was itself erroneous or at least overly cautious. Quadpack's `QAWO`, called from Maxima, returns in about 0.001 seconds[5] with an answer 1.69206436906*66968*e-4, where we know from more accurate evaluation that the trailing digits in italics are in error. A Mathematica program oriented toward oscillatory integrands, `LevinIntegrate` by Andrew Moylan [10] obtained the same result in 0.05 seconds. (More about this method later.)

Note that the inverse function could in fact be represented by an approximation, and computed in a separate step, rather than seeing it substituted literally into the transformed integrand.

In passing we also note that there is sometimes a requirement to integrate $\sin(x)^n g(x)$ for some integer n. Since the power of $\sin(x)$ can be written as a sum of multiple angles, (e.g. $\sin^3 x\ =\ 1/4(3\sin x - \sin 3x)$) the task can be reduced to the previous case.

[4]Sufficient conditions, mathematically speaking: f is differentiable and $f^(-1)$ is continuous. Solution methods in computer algebra systems may not necessarily assure this second condition, or for that matter, may not find an explicit inverse.

[5]QAWO returns in 0.0002 sec if the function $\cos(\operatorname{asinh}(y))/\sqrt{y^2+1}$ is compiled.

4 Integration by Parts

In this section we follow the approach suggested by work by Iserles and Olver on using derivatives [6]; computer algebra systems provide an efficient route to setting up this kind of method. Also see (for example) Xiang's paper [12] for more examples.

Here's the idea. We can use integration by parts as follows:

$$\int e^{i\omega f(x)} g(x)dx \;=\; \int udv \;=\; uv - \int v\ du.$$

Let $u = g/(i\omega f')$, $dv = e^{i\omega f} \times i\omega f'$. Then $v = e^{i\omega f}$, and $du = (f'g' - gf'')/(-i\omega (f')^2)$

There are several points to note. First, we can conveniently integrate this dv, since it is an exponential times the derivative of its argument. In the case of a generalization to other oscillatory functions (see later), this will not be so simple. We consider the uv part as "solved" and that it requires no further attention other than ordinary evaluation. We can factor out $-1/(i\omega)$ from the remaining integral, $K = \int v\ du$. Notice now that K is itself of the same form as the original problem, so integration by parts can be used again. Because the remaining integral will have that coefficient out front of $1/\omega^2$, we can consider this as a relatively small "correction" to the uv. Furthermore the size of this correction drops faster if the oscillation parameter ω is larger. (That is, the approximation gets better for *more oscillatory* integrands.) Indeed, by repeating this operation (recursively, in our program) we can generate a series in $1/\omega$. Caution must be exercised though, since this is an asymptotic series and does not actually converge for *fixed* ω as we continue to add more terms.

Computationally, this expansion in series requires the production and evaluation of (perhaps many) derivatives of g, although ultimately these will be evaluated only at the endpoints of the definite integral, and there are tricks for doing this faster. The slightly odd display of du with the $i\omega$ in the denominator makes for a slightly better appearance of the formula/program below.

```
/*integrate by parts oscillatory integral, ibpoi */

ibpoi(f,g,x,w,L):= ibpoi1(f,g,x,w,L,0);  /* set count to 0 */

ibpoi1(f,g,x,w,L,c):= if c>=L then 0  else
 block([df:diff(f,x)],
  exp(%i*w*f)*g/(%i*w*df)
  - 1/(%i*w)*ibpoi1(f, (df*diff(g,x)-g*diff(df,x))/df^2,x,w,L,c+1))

/*our example */
define(v(x),ibpoi(sinh(x),cos(x),x,1000,1));
expand(v(1.0)-v(-1.0));
```

In this case, just one term yields 1.*702*e-4, which is correct to almost 2 decimal places. Carrying 3 terms provides 1.69206436*7290*e-4, good for 9 decimal places. Continuing for 6 terms, the result 1.69206436906*6987*e-4, is good for about 13 places. Ultimately, increasing the number of terms ceases to increase the accuracy in this asymptotic series for fixed ω. The final remainder term could be evaluated by some other means rather than simply omitted as we have done. Because ipboi is a symbolic program, we can leave the parameter ω symbolic instead of using the number 1000, and compute the resulting formula. For $v(x)$ with one step in the expansion:

$$-\frac{i\, e^{i\,\omega\,\sinh x}\,\cos x}{\omega\,\cosh x}.$$

Clearly this formula is not expensive to evaluate.

For further analysis, see Iserles, (Lemma 2.1) [6]. While additional intuition may be gleaned from examining these formulas, in practice just running the four-line program may be quite useful. The trick, in case very many terms are deemed useful, is to observe that one only needs $f'(a)$, $f'(b)$ $f''(a)$, $f''(b)$ etc. These are likely more rapidly calculated by computing two Taylor series for f, one centered at a and one centered at b, and similarly for g. Ordinary arithmetic on Taylor series (so long as they are expanded at the same point) is also provided in most computer algebra systems.

The Maxima program then looks like this:

```
(ibpoilims(f,g,x,w,L,a,b):= /* integrate from a to b */
block([keepfloat:true, ratprint:false],
 at(ibpoilims1(taylor(f,x,b,L),taylor(g,x,b,L),x,w,L,0),x=b)
 -at(ibpoilims1(taylor(f,x,a,L),taylor(g,x,a,L),x,w,L,0),x=a)),

ibpoilims1(f,g,x,w,L,c):= if c>=L then 0  else
 block([df:diff(f,x,1)],
  exp(%i*w*f)*g/(%i*w*df)
  - 1/(%i*w)*ibpoilims1(f,(df*diff(g,x,1)-g*diff(df,x,1))/df^2,x,w,L,c+1)))

/*our example */
ibpoilims(sinh(x),cos(x),x,1000.0,3,-1.0,1.0); /* integrate from -1 to 1 */
```

Another tack, similar in concept to the program above but quite different in execution would be to use "automatic differentiation" [6].

While the example given runs quite fast as given, note if we used the command `ibpoilims(sinh(x),cos(x), x,1000,3,-1,1)` using integer values rather than floats, the program produces a large symbolic expression including exact terms like sinh(1), which can later be evaluated to any desired precision. Furthermore, if we change the first line of `ibpoilims1` to return a non-zero function of x, say `ErrorTerm(x)` we can see how its value enters into the result.

Note that this idea as well as variable substitution (change of variables) essentially approaches the definite integration problem by modifying the related *indefinite* integral. Change of variables removes the oscillation, and integration by parts attenuates the contribution of the oscillation.

5 Elaborations and improvements: Levin

When the integrand is recognized as a case of a smooth function multiplied by a simple oscillation, Filon's method or modified Clenshaw-Curtis can be used directly, and oscillation presents no problem. If we intend for the computer system to use this procedure automatically as part of a general integration routine then we need to programmatically recognize that the integral is in fact appropriately oscillatory, and the routine must separate the pieces.

A major alternative can be based on an idea due to Levin [8], which works as follows: Again we compute the *indefinite* integral, a function of x

$$H(x) \;=\; \int e^{i\omega f(x)} g(x)dx.$$

Given H one can simply computes $H(b) - H(a)$. The problem is neatly solved if any of the infinite number of formulas for H can be found. H, in this case is a formula for an indefinite integral which is valid up to an arbitrary constant! All we need is some formula which can be evaluated (perhaps only approximately) at two points. In the course of the subtraction the arbitrary constant cancels. Finding some H may be time-consuming though it is something that computer algebra systems can sometimes do. If we can anticipated that it is too time-consuming, perhaps we won't bother to look! In the case of solving many problems with the same integrand but different endpoints or internal parameters, it may be well worth looking. A more likely case is that it is *not especially convenient or possible to find a closed form.* It is still possible to find a method to approximate H as closely as desired.

[6]http://www.autodiff.org

The Levin technique [8] proceeds by finding an approximation for H by observing that we are looking for (or approximating) a function $p(x)$ such that

$$e^{f(x)}p(x) \;\approx\; \int e^{f(x)}g(x)dx.$$

If we compute the derivative with respect to x of both sides and divide out by $e^{f(x)}$ we have a differential equation

$$p'(x) + f'(x)p(x) = g(x)$$

.

All we need to do is find p, and then $H(x) = e^{f(x)}p(x)$, leading to the approximate value of the desired integral, $H(b) - H(a)$.

Levin observes that under the assumptions that g and f' are at most slowly oscillatory, then among the solutions for p there is one that is not rapidly oscillating. Furthermore, we will find it by a collocation method. Using a computer algebra system instead of a purely numerical approach to finding $p(x)$ can provide additional information – like variation of values (and error) with respect to ω.

In reviewing options for solving the associated differential equation, one thought in using computer algebra was to try direct generation of a Taylor series for p, using one of several symbolic-iterative methods. We found that this is not a good idea because such a solution requires choosing an expansion point (we could select the midpoint of the interval) and producing a series whose coefficients are computed by matching derivatives at that point. Since the integrand is highly oscillatory, this does not provide a high quality result at any distance from the point of expansion, and as ω increases, the approximation worsens. Basically, we would find the wrong solution to the ODE.

Instead, as demonstrated by Levin, we can find an approximation to a less oscillatory solution by collocation.

This transforms the computation of finding p into a set of linear equations for finding a polynomial fit to the solution. Numerous variations on this technique have emerged to try to overcome problems with this numerical approach, since it appears that the generated equations may be an unstable system.

Let us return to a simple form where we have, for the moment absorbed $i\omega$ into f. We wish to compute or approximate p in:

$$p' + f'p = g$$

We proceed by assuming some expansion of p in the same (or other) basis functions but with coefficients to be found by collocation.

Given the basis $\{x^j\}$, and therefore assuming $p_k = \alpha_0 + \alpha_1 x + ... + \alpha_{k-1}x^{k-1}$, we have a corresponding $p' = \alpha_1 + ... + (k-1)\alpha_{k-1}x^{k-2}$.

Choose some set of collocation points, which might be chosen for simplicity of exposition (but not optimality) as equally-spaced from a to b. $a = r_0, r_1, ..., r_{k-1} = b$. We set up k linear equations by which one can determine the $\{\alpha_j\}, j = 0, ..., k-1$. This requires computing $f'(x_j)$ and $g(x_j)$ at those points, as part of the setup.

Once one has the values of $\{\alpha_j\}, j = 0, ..., k-1$, and hence an approximate functional form for p, we can use that to compute $p(a)$ and $p(b)$.

Here's a sample program in Maxima's language:

```
/* integrate (exp(i*w*f)*g,a,b) by Levin method, k points*/

LevinInt(f,g,a,b,omega,k):= block([varlist,p,pointlist,sol,ratprint:false],
   varlist: makelist(alpha[i],i,0,k-1),
   p: varlist . makelist(x^i,i,0,k-1),
   pointlist : makelist(a+ (b-a)*i/(k-1),i,0,k-1),
   define(eq(x), diff(p,x)+%i*omega*diff(f,x)*p-g),
   sol:linsolve(eqs:map(eq,pointlist),varlist),
   define(H(x), exp(%i*omega*f)*ev(p,sol)),
   rectform(H(b)-H(a)))
```

Using our standard example, `LevinInt(sinh(x),cos(x),-1.0,1.0,1000,8);` in which we have chosen collocation at only 8 points, produces a result of 1.69*178...*e-4. Using 23 subdivisions we get 1.6920643*3981*e-4.

This program is only a single example, and building a more-or-less automatic program would require attempting some measurement of likely error, measuring effort, increasing precision, and for that matter, checking that f and g are sufficiently smooth, and the formulation makes sense.

We have experimented with increasing the precision (above machine double-floats), with somewhat inconclusive results. In particular an interesting alternative to the numerical operation of the above program is to allow for the complete answer to be generated (symbolically including expressions like sinh(1/30)) and then proceed to simplify that [large] expression, and/or evaluate it using ever-higher numerical precision to see if that affects the result[7]. Other variation include using points and a set of orthogonal functions that provide better fit with fewer terms, for example Gauss-Lobatto points and Chebyshev polynomials.

6 Extension to other functions

Let us sketch the basic idea of extension to other functions as shown by Levin [9] where the exponential can be replaced by an element of a vector of oscillators which has the required form $\mathbf{W}' = \mathbf{A} \cdot \mathbf{W}$.

Levin's paper solves for (composed) functions for collocation[8]. Extensions to this idea include matching derivatives as well as values at grid points [7].

As a specific example, take the Bessel functions of the first kind, (J) which satisfy the differential equation

$$\begin{pmatrix} J_0(x)' \\ J_1(x)' \end{pmatrix} = \mathbf{A} \begin{pmatrix} J_0(x) \\ J_1(x) \end{pmatrix}$$

where

$$\mathbf{A} = \begin{pmatrix} 0 & -1 \\ 1 & -\frac{1}{x} \end{pmatrix}.$$

The product of any two oscillators satisfying the criterion with an invertible matrix also satisfies the criterion. For example, to add products of sin, cos, J_0 and J_1 to the possible oscillators, consider the matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 0 & 0 & 0 & 0 & -\frac{1}{x} & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -\frac{1}{x} \end{pmatrix}$$

corresponding to the vector of concatenated oscillators:

$$\mathbf{W}^T = (\sin(x), \cos(x), J_0(x), J_1(x), \sin(x)J_0(x), \cos(x)J_1(x), \cos(x)J_0(x), \sin(x)J_1(x)).$$

This can now be solved in the same way.

We reduce the integration problem to finding the (now vector) solutions for $\mathbf{p}$ of the differential equation

$$\mathbf{p}' + \mathbf{A}\mathbf{p} = \mathbf{g}.$$

[7]Methods for the exact solution of a linear system with symbolic coefficients are quite different from numerical methods, and can result in huge solution expressions.

[8]On the computer algebra front, Levin's 1996 paper expands on this and provides an example computed by Mathematica, but other than setting up the collocation matrix more or less conveniently, no symbolic features are used: the example is primarily using Mathematica to access a library routine for numerical solution of a set of linear equations. Moylan makes all this work more automatically [10].

We do this via collocation: by choosing a set of points, say $\{c_j\}$ and a proposed expansion of $\mathbf{p}$ in some orthogonal set (e.g. powers of x, or Chebyshev polynomials of degree r or less), evaluating and solving

$$\mathbf{p}'(c_j) + \mathbf{A}(c_J)\mathbf{p}(c_j) = \mathbf{g}(c_j).$$

Again, finding $\mathbf{p}$ allows us to integrate from a to b by computing $\mathbf{p}(b)\mathbf{W}(b) - \mathbf{p}(a)\mathbf{W}(a)$.

Arranging this computation satisfactorily demands attention to many details; for example subdivision of the range from a to b may be advisable since fitting a sequence of low-degree curves may be superior to trying to make a single polynomial fit. Separating sections at zeros of the non-oscillatory function has advantages. Certainly providing an estimate of the error, and well as methods to decrease the error at the cost of more computation, are expected from a full-service library routine such as the previously cited Mathematica's `NIntegrate`.

7 Revisiting integration by parts

Previously we took advantage of the fact that $e^{i\omega f(x)}f'(x)$ was easily integrated to develop an asymptotic series for an integral, and we could slip the $f'(x)$ term into the denominator of the other factor. Since this term has a factor of ω, it attenuates the remaining term.

We have the vector differential equation for $J_0(x)$ and $J_1(x)$ conveniently at hand. Our hypothesis once again is that there is some parameter ω representing the frequency, in this case such that the integral involves $V_0 = J_0(\omega x)$ and $V_1 = J_1(\omega x)$ for large ω, and when we take the derivative, a (large) multiplier of ω emerges.

Let $dv = Wdx$ or, rewriting it, $dv = (1/\omega)A^{-1}W'dx$. So $v = (A^{-1}/\omega)W$. Let $u = g$, and $du = dg$. Using integration by parts, the integral is $uv - \int vdu = (A^{-1}/\omega) * (g * f - \int W * g'dx)$. The integral on the right is of the same form as the initial one but divided by ω. If we repeat the process on this integral, we end up with the next remainder integral divided by $omega^2$ (and this can be repeated again).

Working out the details one can get a formula like this one [12] (Lemma 3.1): Assume that $W'(\omega x) = A(\omega, x)W(\omega x)$ where $A(\omega, x)$ is a non-singular $m \times m$ matrix. Let $B(\omega, x) = ((1/\omega)A(\omega, x))^{-1}$. Suppose also that $W(\omega x), B(\omega, x) \in C^\infty[a, b]$. Let

$$F_1(x) = F(x),$$
$$F_{k+1}(x) = (B^T(\omega, x)F_k(x))', \quad k = 1, 2, \dots .$$

Then for $\omega >> 0$ we can construct the following asymptotic formula (by repeated integration by parts):

$$\int_a^b F(x)W(\omega x)dx \;\approx\; \sum_{k=1}^{s}(-1)^{k+1}/\omega^k(F_m(b)B(\omega, x)W(\omega b) - F_m(a)B(\omega, a)W(\omega a)).$$

The proof, details, and numerical experiments are described by Xiang [12]. This can easily be written out in a computer algebra system.

8 CAS quadrature and error estimation

It is easily demonstrated that any deterministic mechanism for quadrature that is based on sampling can be stymied by an adversarial integrand-evaluation procedure to produce an arbitrarily incorrect result, and an under-estimated error. A CAS may have a different prospect in this regard, since it is possible in this context that the integrand is presented as a mathematical formula. This formula can be examined for various characteristics including continuity, differentiability, and location of poles and singularities.

A CAS may also support, in a natural way, interval calculations, and simple ideas can come to the fore: An obvious upper and lower bound can be obtained by even such a simple mechanism as the rectangular rule. If $M = \max_{a \le x \le b} f(x)$, and $m = \min_{a \le x \le b} f(x)$, then $|b - a|m < \int_a^b f(x)dx < |b - a|M$. For some of the techniques here, $|b - a|$ can be made small by subdivision, or $f(x)$ can be made slowly-varying.

It is also possible, in many cases, to bound the truncation error, that is, the error incurred in replacing a mathematical function by an approximation: say, replacing a function by its Taylor series, or Chebyshev series.

Finally, there will usually be a contribution to the error in a reported result caused by the finite-precision arithmetic used for evaluation. This can be alleviated in a CAS in several ways.

- The expressions being evaluated can be mechanically rearranged to minimize cancellation.
- The expression, or parts of it, can sometimes be evaluated *exactly* using rational arithmetic.
- The expression can be evaluated in arbitrary-precision arithmetic. That is, the same expression can be used but with a working precision of any specified number of bits, and thus some computational deficiency at precision n can be overwhelmed by recomputation at precision $2n$ or higher.

Overall, we have found that in studying particular formulations of quadrature, we can more easily experiment in this framework: given an integral, how best can we approximate it with one of these techniques? Which improves the value, a higher order procedure (more terms, smaller subdivisions, etc.) or more precise arithmetic? While a standard numerical language or system can generally allow for more terms, a CAS provides easy access for other arithmetics, in particular, arbitrary precision experiments, or perhaps interval arithmetic. These techniques can be honed on chosen examples whose exact values are known.

9 Conclusion

We have added a few nuances to previous discussions of numerical integration in a symbolic context ([3, 5]) which were primarily concerned with dealing with singularities in the integrand, stationary points in the oscillatory component, or with infinite integrals. We have by no means completely surveyed the literature on the numerical treatment of highly oscillatory problems, but have simply demonstrated the kinds of illustrations that rather short computer algebra programs can make toward implementing some of the methods. We have not, in this brief paper, attempted to provide a "one-stop" complete automated solution, nor have we included necessary code in our examples to account for checking of input and estimates of error. We have not covered a number of other approaches or variants (in particular, variations on the method of stationary phase) that seem either less accurate, more expensive, or more difficult to program than the methods here.

After we wrote an initial draft of this paper, version 8.0 of Mathematica was released. This version has an expanded integration facility, enlarging its already ambitious numerical integration suite of programs. It includes the Levin-like methods contributed by Andrew Moylan [10]. The resulting system is quite complex, although in our experimentation, capable of good results with many "automatic" settings. Even so, a better understanding of how these ideas work may be fruitful in guiding the program to use efficient methods for particular kinds of documented or new oscillatory integrals.

References

[1] S.M. Chase and L.D. Fosdick, "An algorithm for Filon Quadrature," *CACM 12 no 8* August 1969 453–457.

[2] G.A. Evans, "An alternative method for irregular oscillatory integrals over a finite range", *Internat. J. Comput. Math. 53* (1994) 185-193.

[3] R.J. Fateman, "Computer Algebra and Numerical Integration," Proc. SYMSAC'81 (ISSAC), August, 1981, 228–232.

[4] L.N.G. Filon, "On a quadrature formula for trigonometric integrals," *Proc. Roy Soc. Edinburgh 49* 1928-29 38.

[5] K.O. Geddes, "Numerical Integration in a Symbolic Context" Proc. SYMSAC-86, (ISSAC) July, 1986, 185–191.

[6] Arieh Iserles and Syvert P Norsett. "Efficient quadrature of highly oscillatory integrals using derivatives," *Proc. R. Soc. A 8* May 2005 vol. 461 no. 2057 1383—1399 .

[7] A. Iserles and S. P. Norsett, “On Quadrature Methods for Highly Oscillatory Integrals and Their Implementation” *BIT Numerical Mathematics 44*, Number 4, 755—772, DOI: 10.1007/s10543-004-5243-3

[8] David Levin. “Procedures for computing one- and two-dimensional integrals of functions with rapid irregular oscillations,” *Math. Comput. 38 (1982)* 531—538.

[9] David Levin, “Fast integration of rapidly oscillatory functions,” *J. Computational and Applied Math. 67 (1996)*, 95—101.

[10] David Levin,`http://sites.google.com/site/andrewjmoylan/levinintegrate`. Also see arxiv 0710.3140v1.pdf.

[11] Newton Institute Seminars on Highly Oscillatory Problems, `http://www.newton.ac.uk/webseminars/pg+ws/2007/hop/`

[12] Shuhuang Xiang, Weihua Gui, Pinghua Mo, “Numerical quadrature for Bessel transformations,” *Appl. Numer. Math. 58* (2008) 1247–1261.