DRAFT: Numerical Integration for Oscillatory Integrands: a Computer Algebra Perspective

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

August 27, 2009

Abstract

How can a computer algebra system (CAS) help in getting a good numerical approximation to an oscillatory quadrature problem? Our assumption here is that we are primarily interested in the rapid and accurate numerical estimation of an integral, and that, even though we are using a CAS, an exact expression cannot be obtained for theoretical or practical reasons.

1 Introduction

Here we consider the numerical integration (quadrature) of functions $g(x)$ that are challenging for the usual tools like Gaussian quadrature or Clenshaw-Curtis quadrature because $g(x)$ oscillates too much. For such integrands, any sampling process is likely to choose an unrepresentative sample set of values to use as a basis for estimating the quadrature. A good conventional integration procedure *may* notice that there is such a problem if a sequence of approximations, each computed by increasing the number of samples, does not appear to converge. Such a procedure ordinarily will not offer an alternative method, though a kind of poly-algorithm based in a CAS may attempt a different method.

In particular, if the integrand $g(x)$ can be analyzed symbolically (by human or computer) so as to be decomposed to a product $f(x)\sin(x)$ of a slowly-varying function and an oscillatory function, there are ways of dealing more effectively with quadrature: there is a substantial literature, going back at least as far as Filon's 1928 paper [1].

What can we say now that hasn't already been said? In particular, a CAS has tools that might allow us to (a) notice: by looking at the symbolic form of an integrand, that it is likely to be oscillatory, and separate out $f(x)$, the "slower" term(s); and (b) perform the required arithmetic to arbitrarily high floating-point precision so that (heuristically) the computation can result in an answer with any desired goal accuracy.

2 What is Filon quadrature?

First of all let us review Filon's observations. Consider common forms for integrals that oscillate prominently. Say such an integral can be expressed as $\int g(x)dx = \int f(x)\sin(mx)dx$. Instead of integrating by approximating the integrand $g(x)$, let us approximate $f(x)$ by, say, a linear or quadratic polynomial, on each of p panels.

Then, knowing exact formulas for integrating $x^n\sin(mx)$ and summing over all the panels, we can do a better job, compared to ordinarily quadrature methods which must approximate all of $g(x)$ with a single relatively low-degree polynomial. This latter method is especially prone to inaccuracy if the oscillation in $g(x)$ is rapid (that is, $\sin(mx)$ has many changes in sign within the range of the integral).

There are a number of subtleties in implementation of Filon quadrature using fixed-precision floats. Chase and Fosdick [2] describe some concerns and techniques to overcome them, noting in particular that

some subexpressions are subject to dramatic cancellation. Some of the issues evaporate if a few intermediate calculations can be done in much higher precision, in particular evaluating the constants α, β, γ below. The code from Chase and Fosdick is available in ACM Algorithm 353. Since this material is easily available on the internet, we use similar variable names without apology.

An executable version of the code in Maxima/Macsyma is given below.

The goal is to compute $S = \int_0^1 f(x)\sin(mx)dx$ where we are given f, m, and p, the number of panels.

```
filons0(f,m,p):=  /*integrate f(x)*sin(m x) from 0 to 1,
  for notation, see Chase & Fosdick Alg 353  */
block([h:bfloat(1/(2*p)),
       k:bfloat(m), theta,s2p,s2pm1,alf,bet,gam],
  theta: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),
  block([fpprec:2*fpprec], /*temporarily double the floating precision */
    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)),
 h*(alf*(f(0)-f(1))*cos(k) +bet*s2p+ gam*s2pm1))
```

We can try, as an example,

```
filons0 (lambda([x],bfloat(x^2)),10,30)
```

which is an exact formula subject, however, to numerical error because it is using floating-point evaluation.

It is similarly exact if we use just one panel instead of 30 as

```
filons0 (lambda([x],bfloat(x^2)),10,1)
```

The formula is identical to exact integration for polynomials of degree 4 or less. For higher degree polynomials the fit is not exact: increasing the number of panels decreases the error as we are approximating $f(x)$ in smaller sections by parabolas. Doubling the number of panels gets us about one more hexidecimal digit of correct answer. Integrating some $f(x)$ which is not particularly well-fit with moderate degree polynomial sections may require many panels. For example, integrating $x^6\sin(10\pi x)$ from 0 to 1 with $p = 30$ gives an answer -.030875*1930*, where italics indicate incorrect digits. With $p = 3000$ the answer is better: -.0308752065344*97927* yet the correct answer is closer to -.030875206534498074666. These calculations were done in 100-decimal digit arithmetic.

There are several variations and optimizations that come to mind. An Appendix includes using cos rather than sin, changing the limits of integration, and removing some redundant calculations. A version using a high-performance multiple-precision package (MPFR) can be written in Lisp and loaded into Macsyma/Maxima, for a substantial performance gain, especially if very high precision is specified. The logic would also be executed faster, but this is probably not a major part of the cost.

3 What about less regular oscillatory functions?

Recently, researchers have been successful in approaching the problem for oscillations of the form $\sin(h(x))$ for more general $h(x)$ or for other functions such as Bessel functions, or products of trigonometric and Bessel functions. Chung, Evans, Webster [3] give a general framework for the task and show how to construct a

method that deals with any oscillatory kernel, subject to its satisfying a known linear ordinary differential equation, and expansion in some suitable basis functions. One of their examples is

$$\int_{0.5}^{1} e^x J_0(m\cos(x))$$

for values of m in the range [1,1000]. They were able to compute results to 12 correct decimal digits for different m with only 16 function evaluations of four quantities related to the associated ODE. A spot check with Mathematica 6.0 uses some 275 function evaluations (for $m = 100$) to attain similar accuracy. Even without this general framework, it is possible to simply compute using a CAS integrals of the form $x^n J_0(x)\cos(x)$. Given symbolic solutions (in hypergeometric functions) for some x^n, say $\{1,\ x,\ x^2\}$, we can approximate any other function by a polynomial, and use the fruits of the formula for numerical approximation. It appears that Mathematica version 6.0 includes numerical methods for oscillatory integrands including trigonometric and Bessel functions, though the methods are not specified.

It is perhaps worth noting that because a CAS may be presented by its faithful adherents as more of a universal solvent for mathematical problems, it provides a tempting challenge for some people. Thus if the CAS solves a problem for a given size or complexity parameter (say $m = 100$) then one can try it for $m = 1000$ or 10,000 or more, until it breaks. By contrast, the tradition in describing improved quadrature methods in the numerical community seems to be much kinder: assume that one can find the answer (slowly and perhaps to low accuracy) using a standard method. Can one find an alternative algorithm (faster and to high accuracy.)

Evans [4] provides a somewhat more explicit formulation for generating integration rules for some chosen set of basis functions. For example, he derives a formula for

$$\int_a^b J_n(q(x))F(x)dx = q(x)/q'(x)(f'(x)J_n(x) - J_n'(x)f(x))|_a^b|.$$

The function $f(x)$ on the right is the solution to

$$F(x) = (q/q')f'' + (1 - qq''/(q')^2)f' + q'(q - n^2/q)f.$$

Two conditions, $q \neq 0$ and $q' \neq 0$ in $[a, b]$, are required in this formula. These can be enforced by extracting any such objectionable sub-intervals from the task and using an ordinary quadrature program in these regions. Note that locally $q = 0$ or $q' = 0$ means that the factor is (locally) non-oscillatory. A formula for $q(x) = x$ with $F(x) = x^n$ would provide a basis for approximating $F(x)$ by polynomials, and solving the differential equation is possible but provides a clumsy result in general. For specific Bessel order, especially J_0 and simple enough argument, say $q = x$, integrating $x^k * J_\nu(x)$ is tolerable. In the section on change of variables below, simplifying the argument to a single J_ν is generally possible.

4 How Can Computer Algebra Systems Help?

We can approach the problem using tools from computer algebra systems.

We are not breaking new ground in the purely numerical approach, although "merely" doing very-high precision calculations, as supported by a CAS, can sometimes help. What we are looking for are tools along the lines of symbolic-numeric ideas, as shown by Geddes [7] or our earlier work [6].

These tools can, generically, rearrange expressions, compute derivatives, solve equations, make changes of variables of integrals, or (sometimes) compute symbolic closed forms of certain integrals.

In the case of oscillatory integrands, a CAS may be able to detect that an expression is oscillatory in form, and decompose the integrand into a product as required by Filon quadrature. Determining that an integrand is oscillatory could be done after the failure of routines to converge, or a special test could be applied to try to anticipate the situation [5]. Given the class of expressions that it is possible to construct in a CAS, no algorithm can guarantee determination and separation of an integrand into two parts[1]. On the other hand, a simple algorithm may be effective for many inputs.

[1]Indeed, well-known decidability results tell us that no algorithm can always determine if an expression is identically zero. This kind of decidability result does not prevent CAS from usually working pretty effectively.

For a more specific example, involving Bessel functions J_α, consider this:
An asymptotic approximation to $J_\alpha(x)$ is

$$J^*_\alpha(x) := \sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{x}}\cos\left(\frac{\pi\alpha}{2} - x + \frac{\pi}{4}\right).$$

This expansion becomes more accurate for $x >> |\alpha^2 - 1/4|$, but is not useful near $x = 0$ (at which point the formula has a division by zero!). To integrate $f(x) \times J_\alpha(x)$ over (say) $[1, 100]$, we can integrate $f(x) \times (J_\alpha(x) - J^*_\alpha(x))$, which is a relatively small multiple of $f(x)$ wiggling with low amplitude. Then we add the integral of $f(x) \times J^*_\alpha(x)$, which is likely to be hundreds of times larger: although it wiggles, it can be done by Filon integration to high accuracy!

In reality we could compute $J^*_\alpha(x)$ by

$$s = \sin(\alpha\pi/2)/\sqrt{\pi}, \quad c = \cos(\alpha\pi/2)/\sqrt{\pi}, \quad J^(*)_\alpha = ((s + c)\sin(x) + (c - s)\cos(x))/\sqrt{x}.$$

Our example suggested integration on $[1, 100]$, deliberately avoiding 0. For small x, $J_n(x)$ can be approximated by $(x/2)^n/n! - (x/2)^{n+2}/(n + 1)! + \cdots$, a Taylor series that gets better as $x \to 0$. Using this we can divide up a range of integration including or approaching zero into pieces, although exactly where to make the switch is tricky. (A third formula might be of interest for intermediate values, too. Good ways of approximating Bessel functions has been a popular pastime for a century; an approximation which provides a formula that is easy to symbolically integrate, or is especially easy to numerically approximate are of most value here.)

(Note also that if we are willing to approximate $J_n(x)$ by this Taylor series near $x = 0$ we could also consider expanding the whole integrand $f(x)J_n(x)$ as a Taylor series at $x = 0$ instead.

More generally, we may not be faced *a priori* with a case laid out as a product of different types of terms. To be effective, we must be able to separate out slowly-varying functions in a product that includes faster oscillatory functions. We must be able to combine the product of oscillatory functions into a single function. These determinations must be done relative to a given interval as well as with respect to other items in the product. That is, $\sin 1000x$ seems to vary rapidly, but not on the interval $[0, 0.001]$. Defining a heuristic procedure may be the best we can do in general: deciding whether a component varies rapidly may not be easier than finding an integral, and in fact trying to integrate a term in a product may provide an indication of oscillatory behavior, but we discuss some heuristics for this task elsewhere [5] .

One potentially helpful CAS technique is the ability to expand almost any expression with suitable analytic properties in a Taylor/Laurent series, or using other basis, perhaps using exponentials or logs, with helpful integrability properties (considered numerically or symbolically). This kind of technique is well-known in deriving asymptotic series approximations to integrals (Laplace's method, or steepest descent method).

We have not seen an implementation of the method invoked by Chang *et al* [3], but this too could probably be done most easily in a CAS framework. The prospect here is to essentially provide a newly-derived quadrature formula for an integrand that has been symbolically decomposed. A number of ticklish issues may need to be resolved for such a formula to be derived entirely automatically, although if the formulas are derived "off-line" a collection could be precomputed in advance of need.

5 Transformations

Consider again an integrand that looks like $f(x)\sin x$. Variations on this theme are possible, e.g. $f(x)\sin mx\cos nx$, but recall that

$$\sin a\,\cos b = \frac{\sin(b + a)}{2} - \frac{\sin(b - a)}{2}$$

so that we can consider an integrand to have only a single trigonometric factor. (In reducing an integral to computing $\int udx - \int vdx$ we should be cautious that we have not introduced massive cancellation). We can also easily change the variable of integration so that $\sin(ax+b)$ is rescaled to $\sin(t)$. The algorithm originally in Chase [2] uses $a = m\pi$ and $b = 0$. A more elaborate change of variable may be appropriate, as discussed below.

Now given an integrand of $f(x)\sin mx$ or $f(x)\cos mx$, we can alter the Filon approach and approximate $f(x)$ as a polynomial, or a polynomial times an exponential over each panel separately. Since we can exactly integrate $x^n \exp(ax)\sin(mx)$, we can approximate the integral even with all the wiggles. Or as has been proposed in the literature, we could simplify the setup (but reduce the accuracy) by approximating $f(x)$ with a uniform expression for the whole integral, (Say as an expansion in orthonormal polynomials).

Filon integration is essentially using Simpson's rule, per panel. This does not provide a plausible method if there is a singularity or near-singularity in f.

Here are some ideas which we do not pursue here:

- We could use an alternative, say Gaussian integration, on each panel, or if we assume the singularities are at the endpoints, restrict this technique to the end panels.
- We can expand $f(x)$ in a Taylor (etc) series about the midpoint of each panel.
- If we have deduced the location of a (near) singularity, Filon-style integration will not be especially helpful in that region and some local adaptive method may help. In particular, our assumption that the function $f(x)$ varies more slowly than the oscillatory factor is false near a singularity of f, and so this method is not especially useful: regardless of how much the sin/cos flails, it is relatively constant in the face of a singularity. This assumes the frequency of the oscillation does not also have a singularity as in $\sin(1/x)$.
- Other functions which can be viewed as oscillatory, but which have known exact formulas (when multiplied by a set of basis functions like x^n) can be treated similarly. Bessel functions are one such example. (Trivially, we can also see handle complex exponentials by expansion in sin/cos).
- Transformations of infinite integrals: one or both endpoints at ∞.

6 Change of Variable

We can rescale the integral of $f(x)sin(g(x))$ with a substitution $t = g(x), \quad x = r(t), a' = g(a), \quad b' = g(b)$. Then

$$\int_a^b f(x)\cos(g(x))dx = \int_{a'}^{b'} f(r(t))\cos(t)r'(t)dt.$$

This requires that we can solve the equation $t = g(x)$ for x in an appropriate interval for x, but is otherwise quite general. If there are many roots of $g(x) - t$, the choice of a solution may affect the stability of the numerical evaluation.

For example, Macsyma easily computes, using the `changevar` command,

$$\int_{\text{low}}^{\text{hi}} f\left(x\right)\,\sin\left(m\,e^{x}+b\right)\,dx = \int_{e^{\text{low}}\,m+b}^{e^{\text{hi}}\,m+b} \frac{\sin t\, f\left(\log\left(\frac{t-b}{m}\right)\right)}{t-b}\,dt.$$

This seems to be a rather general approach, though it depends on a somewhat more versatile version of `solve`, namely this program must determine how many of the possibly numerous solutions to an equation $t = g(x)$ are between the integral limits, and we must propose ways of dividing the integral into ranges.

This substitution does not allow us to use Filon integration for (say) Bessel functions; for that we can use some version of the method in Chung *et al* [3].

7 Shifting

We do not know if this technique is generally made available in math libraries, but it occurred to us to use a partial-sum method for improving accuracy: Just as a series acceleration can be applied to a summation, this method can be used for integrals with a periodic component. Consider integrating $f(x) * J_1(x)$ on $[a, b]$ over some large range and note that $J_1(x)$ is nearly equal to $J_1(x + 3\pi)$ for large x. Thus if we make suitable arrangement for matching pieces near a and b, we can use as an integrand $g(x) := f(x)J_1(x) +$

$f(x+3\pi)J_1(x+3\pi)$. Then g is a function that, assuming $f(x)$ is relatively smooth, "wiggles" with decreased amplitude compared to $f(x)J_1(x)$, and thus presents less of a barrier to accurate quadrature. The method can be repeated, since $g(x) + g(x + 3\pi)$ has even smaller wiggles

The actual integrand after this second iteration would be $f(x)J_n(x) + 2f(x + 3\pi)J_n(x + 3\pi) + f(x + 6\pi)J_n(x + 6\pi)$.

A further examination of this would have to be done in relation to series acceleration formulas, at least one of which (ACM Algorithm 639) involves infinite oscillating tails and an Euler acceleration.

8 Notes on Implementation

We wrote a version of the Filon code based on the description in Chase and Fosdick [2] in the Macsyma language. There are some parts of the algorithm that require some care because of numerical cancellation: in particular Chase and Fosdick describe computing the constants alpha, beta, gamma (see the code) by Taylor series, in some circumstances, to avoid troubling cancellation. One simple alternative is to compute them as we do, to much higher precision than the rest of the calculation.

A trick that we also can use in speeding up the computation: since we need $\sin(h)$, $\sin(2h)$, $\sin(3h)$ ... and similarly for $\cos(nh)$, consider this;

```
(sh:sin(h), ch:cos(h), kill(ss,cc),
ss[0]:0,
ss[n]:= ch*ss[n-1]+sh*cc[n-1],
cc[0]:1,
cc[n]:=ch*cc[n-1]-sh*ss[n-1])
```

`ss[n]` is $\sin(nh)$. `cc[n]` is $\cos(nh)$.

Using this technique there is a potential issue of accumulated round-off error for large n. How large?

```
If h=0.5d0, ss[1000] = sin(500.0d0)=  - 0.46777180532248d0
If h=0.1d0  ss[1000] =  - 0.50636564110978d0  which is very close to
        sin(100.0d0) =  - 0.50636564110976d0  which is even closer to
    bfloat(sin(100)) =  - 0.506365641109758793660d0
```

9 Conclusion

A CAS can easily be used to implement a method for quadrature of certain oscillatory forms, or can call an implementation of quadrature using a traditional numerical language. The role of a CAS is potentially different if high-precision arithmetic is useful, or if a combined symbolic and numeric approach is beneficial. More significantly, if the nature of such an integrand can be determined automatically by a CAS, the user of such a system may be handed a rather rapid and accurate result without knowing much about quadrature programs. In our experience one of the hindrances in using quadrature libraries is the burden of figuring out which routine to use and how; our intention here has been to point out (or remind) the reader of the roles that can be played by a CAS.

References

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

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

[3] K.C. Chung, G.A. Evans, J.R. Webster, "A method to generate generalized quadrature rules for oscillatory integrals," *Applied Numer. Math. 34 (2000)* 85—93.

[4] G.A. Evans, "How Integration by Parts Leads to Generalised Quadrature Methods," *Intern. J. Computer Math.,* 2003, vol 80(1), 75—81.

[5] R.J. Fateman, "When is a function oscillatory?" draft, June, 2007.

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

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

10 Appendix: Some more programs

```
filonsab(f,m,p,a,b):=  /*integrate f(x)*sin(m*x) from a to b,
                           a<b with p panels,
         transform to
(b-a) * integrate(f((b-a)*t+a)*sin(m*((b-a)*t + a)),t,0,1) =
(b-a) * integrate(g(t)*sin(r*t + a),t,0,1)  where r=m*(b-a), and g(t):= f((b-a)*t+a) =
(b-a) * integrate(g(t)*(cos(a)*sin(r*t)+sin(a)*cos(r*t),t,0,1) =
(b-a)*(cos(a)*filons00(g,r,p)+sin(a)*filonc00(g,r,p)) */

 block([g: lambda([x],f(x)),r:m*(b-a)],
   (b-a)*(cos(a)*filons00(g,r,p)+sin(a)*filonc00(g,r,p)))$

Try  filonab (lambda([x],bfloat(x^2)),10,30,3,4)
vs integrate(x^2*sin(10*x),x,3,4).
This should be the same for polynomials of degree 4 or less.
```

Here is the cosine formula from 0 to 1.

```
filonc00(f,m,p):=  /*integrate f(x)*cos(m pi x) from 0 to 1, fewer sin/cos.  */
block([h:bfloat(1/(2*p)),
       k:m*%pi, theta,c2p,c2pm1,alf,bet,gam, s2t,st,ct,ti, ti2, ti3],
 local(ss,cc),
 (sh:sin(k*h), ch:cos(k*h),
  ss[0]:0,
  ss[n]:=  ch*ss[n-1]+sh*cc[n-1],
  cc[0]:1,
  cc[n]:=  ch*cc[n-1]-sh*ss[n-1]),
  theta:bfloat(k*h),
  c2p:
   block([sum:0],
    for i:0 thru p do sum:sum+f(2*h*i)* cc[2*i],
     sum-1/2*(f(1)*cos(k))),
  c2pm1:
   block([sum:0],
    for i:1 thru p do sum:sum+f(h*(2*i-1))* cc[2*i-1], sum),
  fpprec:2*fpprec, /*double the floating precision, precaution */
   s2t:sin(2*theta),
   ct:cos(theta),
   st:sin(theta),
   ti: 1/theta,
```

```
  ti2: ti^2,
  ti3: ti*ti2,
 alf: ti+s2t*ti2/2- 2*st^2*ti3,
 bet: 2*((ct^2+1)*ti2-s2t*ti3),
 gam: 4*(st*ti3-ct*ti2),
 fpprec:fpprec/2,
h*(alf*f(1)*sin(k) +bet*c2p+ gam*c2pm1))$
```