

# DRAFT: When is a function oscillatory?

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

August 27, 2009

## Abstract

Numerical integration or “quadrature” of an “oscillatory” function  $f(x)$  is difficult in the sense that—if you don’t recognize that  $f(x)$  is oscillatory—some usually-good methods will be inaccurate, or fail to converge to a correct answer. On the other hand, if you can correctly determine that  $f(x)$  can be manipulated into one of several plausible oscillatory “templates” say  $g(x)\sin(mx)$  then there are special methods that are both fast and accurate.

A quick and accurate categorization of some level of oscillation is therefore handy as an adjunct to numerical integration, especially as a potential preliminary step to symbolically re-arranging an integrand to suit an oscillatory method.

## 1 Introduction

Our objective is to consider analyzing a function sufficiently as being oscillatory or not for purposes of choosing a numerical integration or “quadrature” program. Oscillatory functions present challenges for the usual numerical tools like Gaussian or Clenshaw-Curtis quadrature.

What characterizes such integrands is that any sampling process is too likely to choose an unrepresentative sample set of values to use as a basis for estimating the quadrature.

If it is the case, we should consider methods to apply to the integrand  $f(x)$  symbolically (by human or computer) to decompose it as a product, e.g.  $g(x)\sin(x)$  of a slowly-varying function times an oscillatory function. There are procedures for dealing more effectively with such quadrature going back at least as far as Filon’s 1928 paper [1].

## 2 What does oscillatory mean?

One might think that the function  $\sin(x)$  is oscillatory, but in fact it does not oscillate in  $[0, \pi/4]$ . It is monotonic. However most readers would agree that  $\sin(x)$  does oscillate quite a bit in  $[0, 1000\pi]$ , where there are 500 relative maxima and minima. Clearly we must consider a function *and a range*.

By our criterion,  $10000 + \sin x$  also oscillates, but is fairly easy to integrate, even over a large range. An alternative is to insist on *sign changes*, which could be implemented by a small change in our program.

Another way to quantify “oscillatory” might be by some kind of spectral analysis, but this somewhat begs the question: we don’t want to do so much work that it amounts to integration: we want to save time and limit our analysis work to a small-fraction multiple of the time it would take to do the integration, typically measured in terms of the number of evaluations of the integrand.

Articles and texts tend to make vague statements about “high frequency” or use some notation like  $\omega \gg 1$  to indicate oscillations. This does not lead to a useful automated procedure. Here we try to be more specific.

Here we say “ $n$ -oscillatory” to mean “more than  $n$  local extrema in the interval of interest.” If  $n$  is far less than the typical number of sample points for integration, then we can use a conventional quadrature

formula. If  $n$  is more, we should consider the use of an oscillatory method. Here is a rule of thumb: “Use an  $n$ -point Gauss formula only if there are fewer than  $\log_2 n$  extrema.” (e.g. 16 point formula if there are no more than 4 extrema).

Unfortunately it would likely be too much work generally to compute exactly the number of extrema of  $f$  in  $[a, b]$ , or correspondingly, the number of zeros of  $f'$ .

It is also presumably too much work to try to compute the quadrature and discover that the estimate of the error in the result remains high even as we increase the number of sample points.

### 3 Heuristics

In this section we defend a heuristic to estimate oscillation rate of a function  $f$  on an interval  $[a, b]$ .

We construct a sorted, indexed list  $\{i_k\}$ ,  $k = 1, \dots, 2n$  of  $2n$  pseudo-random numbers distributed uniformly in  $[a, b]$ . Consider first the odd numbered items,  $i_1, i_3, \dots$  and count the non-monotonic subsequences (NMS). We define an NMS as a 3-element subsequence, where the middle element is larger/smaller than both of the elements on either side of it. Next, consider the same characterization for the even elements. Then consider the changes for the list including all the elements. (Note: If we were looking for sign changes, the test would require that an element has a different sign from its neighbors.)

If the three counts (odd, even, all) are substantially the same, consider that the sequence is not oscillatory at level  $n$ . There are no more than about  $n$  extrema. If the three numbers differ, and in particular the third number is about the sum of the first two, then there are more than  $n$  extrema. If this information is sufficient, we are done. If more information is needed we can repeat the test for a larger  $n$  (We can, perhaps, use some of the same data, if we anticipate repeating the testing.)

There is a chance that this program will be fooled. Here are a few ways.

1. The oscillations may be restricted to a small portion of the range. For example  $f(x) := \sin(\exp(x))$  does not oscillate much between  $x = -\infty$  and  $x = 2$  but becomes frantic for larger  $x$ . If most of the sampling takes place on the slow side, it will not look oscillatory.
2. The random numbers may be well distributed but be poorly chosen and lie on some more-or-less monotonic subset of  $f(x)$ .

Here’s a sample program we call `ocelot` (Oscillates a Lot) in Macsyma.

```
(vars(f,a,b,n):=
/* count variations in sample of f on [a,b] at random 2*n points */
map(f,sort(a+makelist(random(float(b-a)),i,1,2*n))),

/* separate out even and odd sublists */

oddlist(k):=block([ans:[]],
for i:1 thru length(k) step 2 do ans:cons(k[i],ans),
reverse(ans)),
evenlist(k):= if k=[] then [] else oddlist(rest(k)),

/*monocount returns a count of non-monotonic subsequences */
monocount(k):=block([a,b,c,ll:k,ans:0],
for i:0 thru length(k)-3 do
(if (ll[i]<ll[i+1] and ll[i+1]>ll[i+2]) or
(ll[i]>ll[i+1] and ll[i+1]<ll[i+2]) then ans:ans+1,
ll:rest(ll)), ans),

ocelot(f,a,b,n):= block ([hh:vars(f,a,b,n)],
/* return list of 3 samplings from same data. Even, odd, and ALL points */
[monocount(evenlist(hh)), monocount(oddlist(hh)), monocount(hh)]))
```

`ocelot(sin,0,6000,10)`; returns [6,4,13] which suggests that  $\sin$  oscillates more than 13 times in [0,6000]; it has at least 6 extrema in [0,6000] detectable by sampling 10 points, and if we sample an additional 10 points we get twice that number. That suggests we have not detected the full extent of the variations: the test says there are probably more extrema.

We can test at a higher sampling rate: `ocelot(sin,0,6000,1000)`; returns [683, 702, 1162] which suggests that there are still more than 1000 extrema.

By contrast, `ocelot(sin,0, 6, 10)`; returns [2,2,2] which suggests it does not vary that much.

## 4 Variations

The same program could, by means of bisection of the range of the function being tested, determine heuristically *where* a function oscillates. That is, if it is determined that  $f$  oscillates on  $[a, b]$ , then one can test  $[a, (a+b)/2]$  and  $[(a+b)/2, b]$  separately to see if one of them is smoother. This can be done recursively, and, for purposes of integration, different quadrature techniques can be used in each section as appropriate. For example,  $\sin(\cosh(x))$  oscillates rather furiously for  $|x| > 5$ , but is always positive if  $|x| < 1.81$ , and has only three gentle extrema.

## 5 Conclusion

We propose a simple heuristic to determine, for a given  $n$  if a function  $f$  oscillates more than  $n$  times in a given interval. This could be used to choose a tactic for numerical integration of  $f$ . An alternative to this random-point testing might be to test at the points used for Gaussian integration, in which case the information gathered could, in the case of a non-oscillatory integrand, be used to find the quadrature. We prefer pseudo-random points to avoid the (perhaps minor) risk that any systematically prescribed set of sample points would result in a misleading conclusion given some deviously constructed function.

This program (`ocelot`) can be used to test each term in a (symbolic) product for oscillatory behavior, allowing the program to partition the oscillatory and non-oscillatory components for further treatment by one of the methods suggested in the references [2], [3]. It can also be used to detect, again heuristically, a range for which a given function is oscillatory.

## 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] R.J. Fateman, "Computer Algebra and Numerical Integration, Proc. SYMSAC-81 (ISSAC), August, 1981, 228-232.
- [5] K.O. Geddes, "Numerical Integration in a Symbolic Context" Proc. SYMSAC-86, (ISSAC) July, 1986, 185-191.