Some additional topics in numerical integration

This set of notes gives some information about

-- adaptive integration

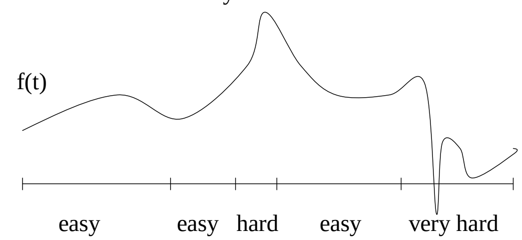
-- handling singularities

-- multidimensional integration

-- Matlab's integration software

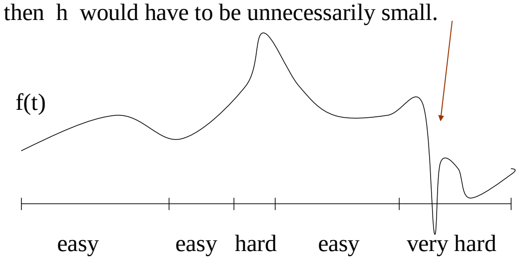
Adaptive integration

The idea: Often, if we integrate a function, we'll find that some intervals are "easy" and some are "hard".



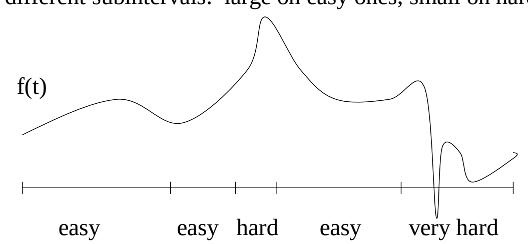
Non-Adaptive integration

If we always took equally spaced intervals of length h, then h would have to be unnecessarily small.



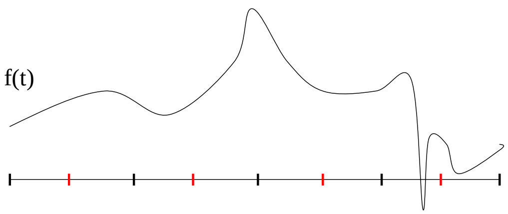
Adaptive integration

Instead, we would like to use a different value of h on different subintervals: large on easy ones, small on hard ones.



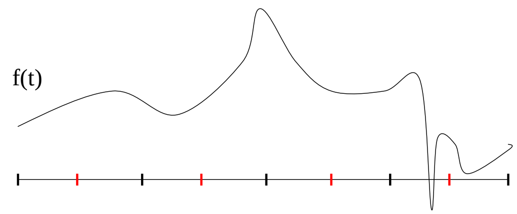
Adaptive integration

Idea: use an initial mesh (black points),
and then a mesh with more points (red and black points).



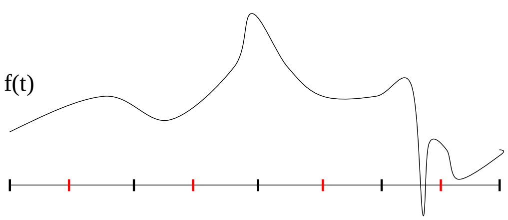
What we get:

This gives us two estimates of the integral, Q and Q.



Better yet:

It also gives us an estimate of the error, Q - Q, in the **less accurate formula** Q .



Important note

Q and Q can be any of our favorite methods.

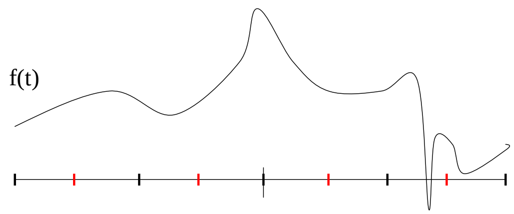
Usually, they are either

-- Romberg formulas (perhaps two trapezoidal rules, or trapezoidal and Simpson).

-- Gauss formulas.

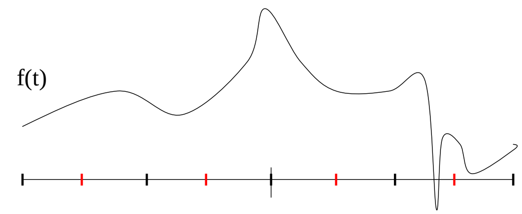
Are we finished?

If the error estimate is less than our tolerance, then we can quit. Otherwise, we subdivide the interval into two pieces, and repeat the procedure on each one.



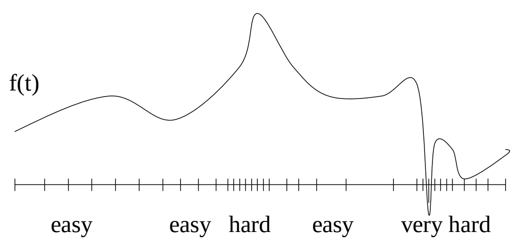
Setting the error tolerance

Each subinterval gets 1/2 the error tolerance, so if the full integral estimate needs to be within .001 of the true value, then Each subestimate needs to be within .0005 of its true value.



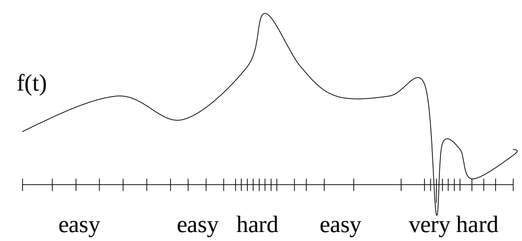
Upon convergence

Eventually, each subinterval achieves success. Some use a lot of points, some only a few.



The final answer

We add up the answers and the error estimates, and that is the information the user receives.



Recursion

The algorithm has a neat recursive implementation:

```
[est_integral,est_error] = Adapt(a,b,f,tol)

  Apply the basic formula to obtain estimate Q.
  Apply the improved formula to obtain estimate Q.
  If the error estimate |Q-Q| < tol ,
     return [Q, |Q-Q| ]
  else
     [I1, e1] = Adapt(a, (a+b)/2, f, tol/2)
     [I2, e2] = Adapt((a+b)/2, b, f, tol/2)
     return [I1+I2, e1+e2]
  end
```

Note our sin here.

Banking

We may be doing more work than necessary in that implementation.

Suppose that the first half interval is easy, so easy that e1 is much smaller than tol/2.

Then we can **bank** the extra tolerance, and give it to the second interval, asking for a less accurate answer.

The new tolerance would be tol-e1.

Adaptive integration with banking

The algorithm has a neat recursive implementation:

```
[est_integral,est_error] = Adapt(a,b,f,tol)

  Apply the basic formula to obtain estimate Q.
  Apply the improved formula to obtain estimate Q.
  If the error estimate |Q-Q| < tol ,
     return [Q, |Q-Q| ]
  else
     [I1, e1] = Adapt(a, (a+b)/2, f, tol/2)
     [I2, e2] = Adapt((a+b)/2, b, f, tol-e1)
     return [I1+I2, e1+e2]
  end
```


Complication 1: Confession of a lie

The recursive implementation isn't really as neat as we claimed.

We just did a lot of work (i.e., function evaluations) to get Q and Q, but now we throw it all away in our calls

```
[I1, e1] = Adapt(a, (a+b)/2, f, tol/2)
[I2, e2] = Adapt((a+b)/2, b, f, tol-e1)
```

To make this practical, we might want to pass this information Down so that the information gained from the function evaluations can be reused.

Complication 2: relative error tolerances

I don’t know a simple way to discuss the algorithm if the user wants a relative error tolerance instead of an absolute one.

In this case, it seems to be unavoidable that we may need to go back and reconsider intervals that we thought were finished.

Iterative implementation

Because of banking and relative error, the algorithm is usually implemented iteratively rather than recursively.

We keep a list of the current intervals, initially [a,b].

Until our total error estimate satisfies the tolerance,

We choose one of the longest intervals [c,d] with a big error est.

We get Q and Q for each subinterval
[c,(c+d)/2] and [(c+d)/2,d] and put them on the list,
along with bookkeeping information.

Panic button

There is also an error exit built in if there get to be too many subintervals.

And there is usually a provision to prevent any subinterval from becoming too short.

Fooling adaptive routines

Fact: Given any deterministic adaptive integration routine, you can construct a function that will fool it.

For example, you can construct a function with integral 1 for which the routine will return an estimated integral of 0, and a very small error estimate.

Randomization

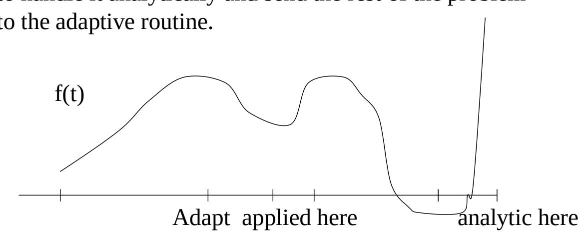
To prevent this embarrassment, adaptive routines usually include some randomization, breaking the interval [c,d] at (c+d)/2 + a small random number.

There are still functions that fool it, but they are not so easy to construct!

Singularities

Singularities

Adaptive routines handle singularities rather well, but if you know you have one, it is preferable, if possible, to handle it analytically and send the rest of the problem to the adaptive routine.



Multidimensional integration

In principle, all of our ideas still work.

Multidimensional integration

We can:

-- fit multidimensional polynomials $p(x_1,x_2,\ldots, x_k)$ and integrate.

-- fit polynomials $p_1(x_1)\, p_2(x_2)\ldots\, p_k(x_k)$ and integrate.
(This is called using a **product formula**.)

-- we can use multidimensional adaptive methods.

-- we can use our favorite one-dimensional method.
Unquiz: how?

But if the number of dimensions is high…

…these methods become hopelessly expensive.

The only practical algorithms…

…for high dimensional problems are based on statistical sampling.

The simplest of these: Monte Carlo

Idea:

To estimate the integral of some function $f(x)$, over some k-dimensional region S,

we generate n points, uniformly distributed in S, and estimate the integral as

the average function value among the n points

times the area of S.

Error estimation for Monte Carlo

The expected value of the estimate is I(f), the true integral.

The standard deviation of the estimate is

$$\text{area}(S)\ N^{-1/2}\ s(f)\ ,$$

where s(f) is a constant **independent of the dimension!**

If the expected value were normally distributed, this would mean that 19 times out of 20, the error would be less than

$$2\ \text{area}(S)\ N^{-1/2}\ s(f)\ .$$

Matlab's integration algorithms

quad Adaptive use of Simpson's rule
Useful when function is not very smooth.

quad8 Adaptive use of closed Newton-Cotes rules;
Listed as "obsolete"

quadl Adaptive use of "Lobatto" rule –
more precisely:
Gauss (chooses points to minimize the max of the polynomial in the error function)
Lobatto (uses endpoints)
Kronrod (reuses old points)
Useful on smooth functions – lots of small derivatives.

References

Adaptive integration: See the book Moler, published by SIAM, 2006.

Singularities: See Section 3.7 of book by Stoer and Bulirsch.

Monte Carlo integration: The methods are much more sophisticated than we hinted at here.

A starting place is, "Monte Carlo and quasi-Monte Carlo methods," Russel E. Caflisch, Acta Numerica 7 (1998) 1-49.