Adaptive Quadrature

• We consider the integration problem

\[ I = \int_{a}^{b} f(x) \, dx =？ \]  \hspace{1cm} (1)

• Note that this is a special IVP of ODEs:

\[ I = F(b) =？ \text{ where } F'(x) = f(x), \quad F(a) = 0. \]  \hspace{1cm} (2)

Many of the ideas we discuss today will carry over to the solution of IVPs of ODEs, a subject we will discuss at length throughout this semester.

• Suppose we wish to write a Matlab method

\[
\text{function } R = \text{adaptq}(f, a, b, \tau)
\]

that returns a number R that (hopefully) satisfies

\[ |I - R| < \tau \]  \hspace{1cm} (3)

and that requires as little effort as possible for its computation.
The Basic Approach

• An outline of the basic approach is:

1. Move from \( a \) to \( b \) in steps. (Contrast this with an alternative adaptive quadrature approach, described below, where we maintain a stack of successively refined intervals.)

2. At each step approximate the integral over the current step, and the error in the approximation.

3. Accept the approximation if the error is sufficiently small, otherwise reject it and take a smaller step.

4. Based on the information created in the previous step, calculate a new stepsize.

5. Repeat items 2–4 until done.

• The obvious question is: How do we approximate the error? This question arises in many other contexts. The answer is usually the same: Calculate two different approximations, and compare them. Relate the difference to the error. If the difference is small enough use a specific one of the two approximations.

• The method described here uses the midpoint and trapezoidal rules for simplicity. Usually one will want basic methods of higher order.

• Let’s introduce some notation: We will come
up with a sequence of points:

\[ a = x_0 < x_1 < x_2 < \cdots < x_N = b. \]  \hspace{1cm} (4)

- The points are unevenly spaced, and their number \( N \) is unknown beforehand. We also use the notations

\[
I_n = \int_{x_{n-1}}^{x_n} f(x) \, dx, \quad h_n = x_n - x_{n-1}, \quad (5)
\]

\[
T_n = \frac{h_n (f(x_{n-1}) + f(x_n))}{2}, \quad (6)
\]

and

\[
M_n = h_n f \left( \frac{x_{n-1} + x_n}{2} \right). \quad (7)
\]

Thus \( T_n \) is the \textit{trapezoidal approximation}, and \( M_n \) is the \textit{midpoint approximation}.

- We know that

\[
I_n - T_n = -\frac{h_n^3}{12} f'' \left( \frac{x_{n-1} + x_n}{2} \right) + \text{H.O.T.} \quad \hspace{1cm} (8)
\]

and

\[
I_n - M_n = +\frac{h_n^3}{24} f'' \left( \frac{x_{n-1} + x_n}{2} \right) + \text{H.O.T.} \quad \hspace{1cm} (9)
\]

where “H.O.T.” means “higher order terms”. More precisely,

\[
\lim_{h \to 0} \frac{I_n - T_n}{-\frac{h_n^3}{12} f'' \left( \frac{x_{n-1} + x_n}{2} \right)} = 1, \quad (10)
\]
and similarly for the midpoint approximation.

- Query. Thus it appears that we can obtain twice as accurate a formula with half the numerical effort. Is this really true? Why not?

- We now come to the key issue in error estimation.

- The Asymptotic Assumption.

\[ \text{H.O.T.} = 0. \]  

(11)

- Note. Thus in building our algorithm we will ignore the higher order terms. This assumption is fundamental in most ODE and PDE solvers. Its vast simplifying power is responsible for our ability to build rather sophisticated algorithms. However, the assumption is also responsible for the fundamental shortcoming of all of these codes: the underlying analysis is only asymptotically correct, and in general one doesn’t know if one is sufficiently close to the asymptotic limit \((h = 0)\). This shortcoming pervades almost all numerical algorithms for solving ODEs and PDEs. In spite of this, modern numerical techniques, particularly for IVPs of ODEs, have been extremely successful.

- Example. To bring home the significance of the asymptotic assumption consider a function that is zero everywhere, except that it
has a single narrow spike enclosing a region with an area of $10^{20}$, say. If the spike is narrow enough and its location unknown, most codes will skip right across it. (In fact, if you want to get really extreme think of a spike that is located entirely between two neighboring floating point numbers. No numerical method will be able to deal with it. However, the asymptotic assumption and its implications are independent of the fact that we use a discrete number system.)

**Error Estimate**

- Suppose now that we have calculated $M_n$ and $T_n$ on some particular interval. Evaluating all derivatives at the midpoint of the current interval, we can estimate the error in $M_n$, say, as follows:

$$I_n - M_n \approx \frac{h_n^3}{24} f''$$

$$I_n - T_n \approx -\frac{h_n^3}{12} f''$$

subtracting $T_n - M_n \approx \frac{3h_n^3}{24} f'' = 3 (I_n - M_n)$

Thus $I_n - M_n \approx e_n := \frac{T_n - M_n}{3}$.

Note that this estimate of the error in the midpoint approximation is *computable*.

**Stepsize Policy**
We now turn to the question of whether or not to accept the midpoint approximation. We don’t know beforehand how many steps we are going to take. But we can relate the maximum allowable error on the current interval to the length of the interval, i.e., the value of \( h_n \). If we require that

\[
e_n \leq \frac{h_n}{b-a} \tau
\]  

(13)

then the overall error will be \( \leq \tau \) (subject to the validity of the asymptotic assumption).

**Selection of new Stepsize**

Whether or not we accept the last step, we will have to take another step with a new stepsize. For efficiency, the stepsize should be such that the inequality in (13) is satisfied as closely as possible by equality. Denote the last stepsize, i.e., \( h_n \), by \( h \), and the new stepsize by \( \hat{h} \). Then the next error, \( \hat{e}_n \), will be

\[
\hat{e}_n \approx \frac{\hat{h}^3}{24} f'' \approx \left( \frac{\hat{h}}{h} \right)^3 e_n \text{ which we would like to equal } \frac{\hat{h} \tau}{b-a}.
\]  

(14)

Solving for \( \hat{h} \) yields:

\[
\hat{h} = \sqrt[3]{\frac{3\tau h^3}{(b-a)(T_n-M_n)}}
\]  

(15)
• Note. If $T_n = M_n$ we divide by zero and obtain an infinite new step size. Of course, this just means that we choose as the new step-size the remaining distance to the end of the interval $[a, b]$.

• Note. Actually we would do well to use a slightly smaller stepsize than given in (14), since otherwise we run the risk of a marginal (and costly) rejection of the next step.

We now have all the ingredients needed for our adaptive quadrature routine. Following is an outline of the algorithm:
Initialize: $n = 1$, $\text{RES} = 0$, $x_{n-1} = a$, $h = $ small, but given

Do forever:

if $x_{n-1} = b$ then EXIT! 
we are done

$x_n := x_{n-1} + h$
new right endpoint

if $x_n > b$ then $h = b - x_{n-1}$, $x_n = b$. Don’t overshoot!

$M_n := hf \left( \frac{x_{n-1} + x_n}{2} \right)$
midpoint approximation

$T_n := \frac{h}{2} \left( f(x_{n-1}) + f(x_n) \right)$
trapezoidal approximation

$e_{\text{est}} := |\frac{T_n - M_n}{3}|$
estimated error

$e_{\text{des}} := |\frac{h \tau}{b - a}|$
desired error

$h := \sigma \sqrt{\frac{3 \tau h^3}{(b - a)(T_n - M_n)}}$
new stepsize

$= \sigma h \sqrt{\frac{e_{\text{des}}}{e_{\text{est}}}}$
alternative expression

if $e_{\text{est}} > e_{\text{des}}$ go to forever failed step

$n := n + 1$
update $n$

$\text{RES} := \text{RES} + M_n$
update RES step

$x_{n-1} := x_n$
update $x_{n-1}$

go to forever repeat
• Note. The initial value of $h$ should be very small, e.g., as small as possible without invalidating results by round-off errors. Only one step will be taken with the initial step-size so that efficiency in the first step is not a significant concern. Note that implicit in our error estimation is an estimate of the second derivative of $f$. In a serious implementation one would probably have the code define an initial stepsize using for example a variant of the numerical differentiation procedure described in Gill, Murray and Wright\textsuperscript{1}, section 8.6.2. Bother the user only if you can’t help it!

• Note. $\sigma$ is a safety factor, e.g., $\sigma = 0.9$.

• Note. The alternative expression for $h$ makes it clear that a stepsize is maintained or increased (subject to multiplication with the safety factor $\sigma$) if the last step was successful, and reduced otherwise.

• Query. What do you think about using the error estimate to improve the approximation? We face this question whenever we have an error estimate: How about subtracting it and getting a better answer? Ponder this!

**Another Approach**

• We focused on the approach described in the

previous section because it introduces many of the ideas that are used for the solution of initial value problems of ordinary differential equations. The approach described in this section is more effective for quadrature problems by exploiting their special structure\(^2\). Suppose we are given some specific basic quadrature rule. In this note, we'll use Simpson's Rule, but there are many other possibilities. As became apparent in the preceding section, the essence of error estimation is to compute two approximations, and to relate the error to the difference of those two approximations.

- Simpson's Rule is given by

\[
I_{a,a+2h} = \int_a^{a+2h} f(x)\,dx = S_{a,a+2h}(f) + E_{a,a+2h}
\]  
(16)

where

\[
S_{a,a+2h}(f) = \frac{h}{3} \left( f(a) + 4f(a+h) + f(a+2h) \right)
\]  
(17)

and

\[
E_{a,a+2h} = -\frac{h^5}{90} f^{(4)}.
\]  
(18)

As usual, we only know that the fourth derivative is evaluated somewhere in the interval \([a, a+2h]\). We ignore the point of evaluation, and assume that all fourth derivatives have\(^2\) Think about just what that structure is!
the same value locally. (This is the asymptotic assumption).

- We can also approximate by applying Simpson’s rule twice, with half the step-size. This gives

\[ I_{a,a+2h} = S_{a,a+h} + S_{a+h,a+2h} + \hat{E}_{a,a+2h} \]  

(19)

where

\[ \hat{E}_{a,a+2h} = E_{a,a+h} + E_{a+h,a+2h} = -2 \times \frac{h^5}{32 \times 90} f^{(4)} = \frac{1}{16} E_{a,a+2h}. \]  

(20)

- Subtracting (17) and (19), and using (20), and solving for \( \hat{E}_{a,a+2h} \) gives the computable estimate

\[ \hat{E}_{a,a+2h} = \frac{S_{a,a+h} + S_{a+h,a+2h} - S_{a,a+2h}}{15} \]

- Suppose we want the overall error to be less than some tolerance \( \tau \), as before:

\[ \left| \int_a^b f(x) \, dx - R \right| \leq \tau \]  

(21)

where \( R \) is the computed approximation.

- To implement this idea we maintain a stack\(^3\)

\(^3\) A stack is a data structure based on the last in—first out principle. You can take the top element from the stack, or add elements on top. Think of it as a stack of plates. You don’t want to pull out a plate from the bottom of the stack.
of data of the form
\[ <a, b> = (a, b, f(a), f((a + b)/2), f(b), S_{[a,b]}) \]  \hspace{1cm} (22)

- The algorithm proceeds as follows:

1. **Put** \(<a, b>\) **on the stack.** Set \(R = 0\)

2. **Until the stack is empty:**
   
   2.1 **Take the top element** \(<\alpha, \beta>\) from the stack.
   
   2.2 Let
   \[ \rho = \frac{\alpha + \beta}{2} \]  \hspace{1cm} (23)
   and compute \(<\alpha, \rho>\) and \(<\rho, \beta>\).
   Evaluate the function \(f\) only at the two new points \((\alpha + \rho)/2\) and \((\rho + \beta)/2\).

   2.3 If
   \[ \left| \frac{S_{\alpha,\rho} + S_{\rho,\beta} - S_{\alpha,\beta}}{15} \right| < \frac{(\beta - \alpha)\tau}{b - a} \]  \hspace{1cm} (24)
   then
   \[ R = R + S_{\alpha,\rho} + S_{\rho,\beta} \]  \hspace{1cm} (25)

   else
put \( \langle \alpha, \rho \rangle \) and \( \langle \rho, \beta \rangle \) on the stack.

- As before, you may want to include a safety factor \( \sigma \), and you may want to use your error estimate to improve your approximation.