CHAPTER 3

Numerical Differentiation and Integration

Numerical Methods in the Hydrological Sciences
Special Publication Series 57

doi:10.1029/057SP08

3. Numerical Differentiation and Integration

There are many situations in the hydrological sciences in which the need to perform a numerical differentiation or integration arises. Examples include integration of functions that are difficult or impossible to solve analytically and differentiation or integration of data having an unknown functional form. Numerical differentiation is also central to the development of numerical techniques to solve differential equations.

3.1. Finite differences

The definition of a derivative is

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

In numerical differentiation, instead of taking the limit as Δx approaches zero, Δx is allowed to have some small but finite value. The simplest form of a finite difference approximation of a derivative follows from the definition above:

$$f'(x) \cong \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Thought of geometrically, this estimate of a derivative is the slope of a linear approximation to the function f over the interval Δx (Figure 3-1). How well a linear approximation works depends on the shape of the function f and the size of the interval Δx.

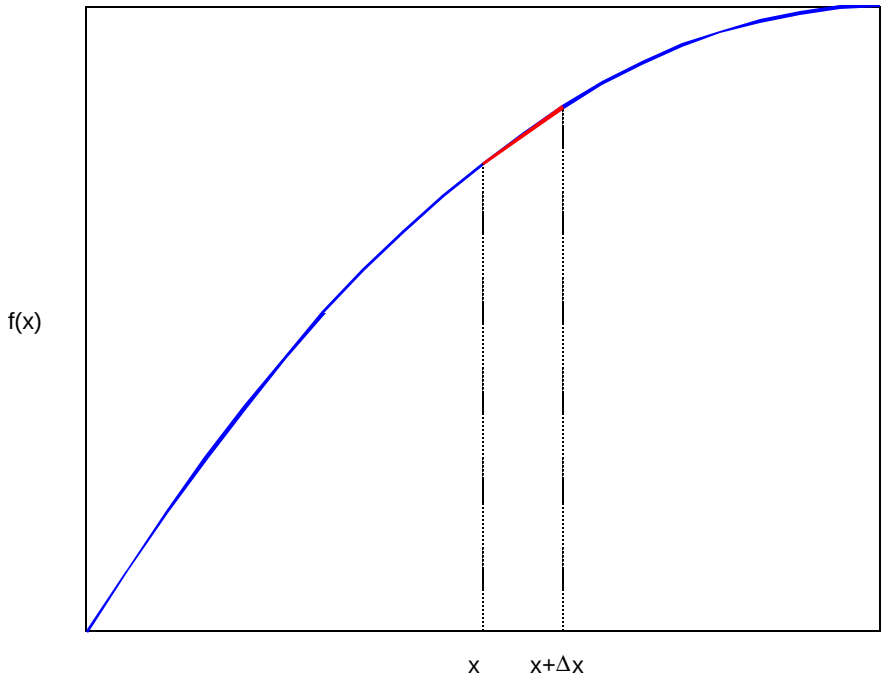

Figure 3.1. Finite difference approximation to a first derivative.

3.2. Taylor series approach

It is important that we have a way of determining the error associated with a finite difference approximation to a derivative. One straightforward way of determining the error is to use a Taylor series expansion to express the value of a function $f(x+\Delta x)$ in terms of the function and its derivatives at a nearby point x:

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)}{2}(\Delta x)^2 + \ldots + \frac{f^{(n)}(x)}{n!}(\Delta x)^n + R_{n+1} \qquad (3.1)$$

where the remainder, R_{n+1} is equal to

$$R_{n+1} = \frac{f^{(n+1)}(x)}{(n+1)!}(\Delta x)^{n+1} \qquad \text{for} \quad x < x < x+\Delta x$$

The error produced by truncating the Taylor series after the $f^{(n)}$-th term is given by R_{n+1} and is said to be of order $(\Delta x)^{n+1}$ or $O((\Delta x)^{n+1})$. Although in general we don't know the value of $f^{(n+1)}(x)$, it is evident that the smaller Δx, the smaller the error [but see Box 3.1 for additional information].

If we truncate the Taylor series after the first-derivative term,

$$f(x \pm \Delta x) = f(x) \pm f'(x)\Delta x + O(\Delta x)^2$$

and rearrange the expression to give an equation for $f'(x)$, we obtain

$$f'(x) = [f(x+\Delta x) - f(x)]/\Delta x + O(\Delta x) \qquad (3.2)$$

or

$$f'(x) = [f(x) - f(x-\Delta x)]/\Delta x + O(\Delta x) \qquad (3.3)$$

The first of these is called a *forward difference* and the second, a *backward difference* (Figure 3.2). Both expressions have a truncation error of $O(\Delta x)$.

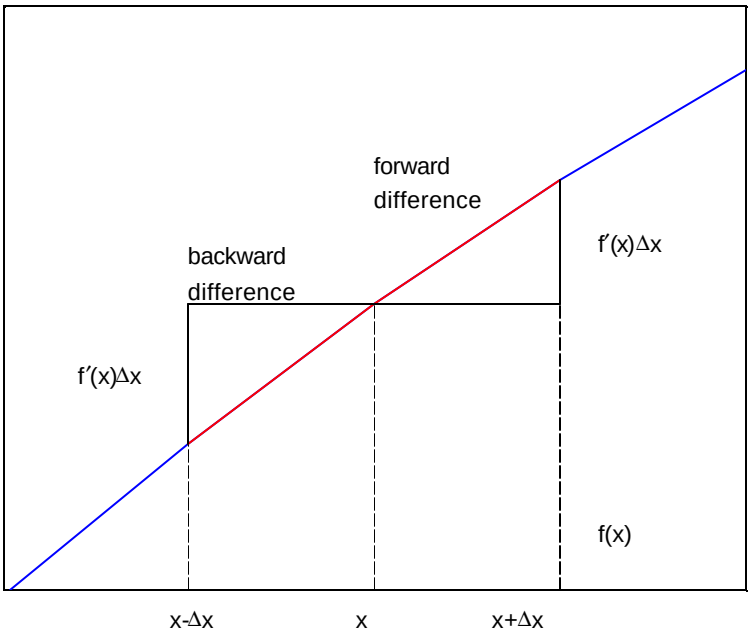

Figure 3.2. Forward and backward difference approximations to a first derivative.

An expression for $f'(x)$ with a smaller error (i.e., a higher order approximation) can be obtained using the first three terms of Taylor series (up to f'' in Eq. 3.1) for $f(x+\Delta x)$ and $f(x-\Delta x)$ and subtracting to cancel the f'' terms.

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + O((\Delta x)^3)$$

$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + O((\Delta x)^3)$$

Subtracting these leaves,

$$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x f'(x) + O((\Delta x)^3)$$

Rearranging to give an expression for f' gives

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2(\Delta x)} + O((\Delta x)^2) \qquad (3.4)$$

This is referred to as a *central difference* approximation. The error in this case is of order $(\Delta x)^2$, compared to an error of $O(\Delta x)$ for forward or backward differencing. The higher order approximation will generally yield a more accurate estimate of the derivative [see Box 3.1]. It can be applied everywhere except at the boundaries where a forward or backward estimate is necessary.

An $O((\Delta x)^2)$ approximation to $f''(x)$, the second derivative of $f(x)$, can be obtained by adding the $O((\Delta x)^3)$ expressions for $f(x+\Delta x)$ and $f(x-\Delta x)$, producing

$$f''(x) = \frac{f(x-\Delta x) - 2f(x) + f(x+\Delta x)}{(\Delta x)^2} + O((\Delta x)^2) \tag{3.5}$$

3.3. Differentiation using interpolating polynomials

Another way to derive expressions for numerical differentiation is to use interpolating polynomials. The resulting $O(\Delta x)$ and $O((\Delta x)^2)$ approximations to f' are the same, but the approach provides some insight into these approximations and into how to generate expressions for higher order derivatives.

An n-th degree polynomial $P_n(x)$ can be fit through any n+1 points in such a way that the polynomial is equal to the known function values $f(x)$ at the n+1 points $x_1, x_2, \ldots x_{n+1}$. The derivative of this polynomial then provides an approximation to the derivative of the function $f(x)$. Consider the 2nd-order polynomial,

$$P_2(x) = a_i + (x - x_i)a_{i+1} + (x - x_i)(x - x_{i+1})a_{i+2}$$

We assume that the values of $f(x)$ are known at 3 values of x. Our aim is to choose the values a_i so that $P_2(x) = f(x)$ at each x_i. The resulting polynomial $P_2(x)$ will provide a smooth interpolation between the known values of $f(x)$.

To find the coefficients of $P_2(x)$, we evaluate the function at $x = x_i$, $x = x_{i+1}$, and $x = x_{i+2}$. For $x = x_i$, this just gives $P_2(x_i) = a_i = f_i$ (all other terms are 0). For $x = x_{i+1}$, $P_2(x_{i+1}) = a_i + (x_{i+1}-x_i)\, a_{i+1} = f_{i+1}$. It is not hard to determine that this is satisfied if $a_{i+1} = (f_{i+1} - f_i)/(x_{i+1}-x_i)$. This ratio is referred to as the first divided difference $f[x_i, x_{i+1}]$. By extension, $a_{i+2} = f[x_i, x_{i+1}, x_{i+2}]$, the second divided difference, which is given by

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} = \frac{1}{x_{i+2} - x_i}\left(\frac{f_{i+2} - f_{i+1}}{x_{i+2} - x_{i+1}} - \frac{f_{i+1} - f_i}{x_{i+1} - x_i}\right)$$

If the points are evenly spaced, the expression for $P_2(x)$ can be simplified. In that case, we can let $\Delta x = x_{i+1} - x_i$ and denote $f_{i+1} - f_i$ as Δf_i. With these modifications, the expression for $P_2(x)$ becomes

$$P_2(x) = f_i + \frac{(x - x_i)}{\Delta x}\Delta f_i + \frac{(x - x_i)(x - x_{i+1})}{2(\Delta x)^2}\Delta^2 f_i$$

where $\Delta^2 f_i = \Delta(\Delta f_i) = \Delta(f_{i+1} - f_i) = \Delta f_{i+1} - \Delta f_i = f_{i+2} - 2f_{i+1} + f_i$. Letting $s = (x - x_i)/\Delta x$, the equation takes a still simpler form,

$$P_2(x) = f_i + s\Delta f_i + \frac{s(s-1)}{2}\Delta^2 f_i$$

If $P_n(x)$ is a good approximation to $f(x)$ over some range of x, then $P_n'(x)$ should approximate $f'(x)$ in that range. Thus we can write, $f'(x) = P_n'(x) + error$. For example, computing the derivative of $P_2(x)$, assuming evenly spaced points and noting that $df/dx = df/ds\,(ds/dx) = (1/\Delta x)\,df/ds$, gives

$$f'(x) = \frac{1}{\Delta x}\left(\Delta f_i + (2s-1)\frac{\Delta^2 f_i}{2} \right) + error\ of\ O((\Delta x)^2)$$

If we restrict ourselves to evaluating the derivative at one of the given points, say $x=x_{i+1}$, then s=l and the derivative reduces to

$$f'(x_{i+1}) = \frac{1}{\Delta x}\left(\Delta f_i + \frac{\Delta^2 f_i}{2} \right) + O((\Delta x)^2)$$
$$= \frac{1}{\Delta x}\left[(f_{i+1} - f_i) + \frac{f_{i+2} - 2f_{i+1} + f_i}{2} \right] + O((\Delta x)^2)$$
$$= \frac{f_{i+2} - f_i}{2\Delta x} + O((\Delta x)^2)$$

This is the same result we obtained using the Taylor series.

The table below summarizes some of the common formulas for computing numerical derivatives.

Table 3.1. Formulas for numerical differentiation

First derivatives	$f'(x_0) = \frac{f_1 - f_0}{\Delta x} + O(\Delta x)$	1st order, forward* difference
	$f'(x_0) = \frac{f_1 - f_{-1}}{2\Delta x} + O((\Delta x)^2)$	2nd order, central difference
	$f'(x_0) = \frac{-f_2 + 4f_1 - 3f_0}{2\Delta x} + O((\Delta x)^2)$	2nd order, forward* difference
Second derivatives	$f''(x_0) = \frac{f_1 - 2f_0 + f_{-1}}{(\Delta x)^2} + O((\Delta x)^2)$	2nd order, central difference
	$f''(x_0) = \frac{-f_3 + 4f_2 - 5f_1 + 2f_0}{(\Delta x)^2} + O((\Delta x)^2)$	2nd order, forward§ difference
Third derivatives	$f'''(x_0) = \frac{f_2 - 2f_1 + 2f_{-1} - f_{-2}}{2(\Delta x)^3} + O((\Delta x)^2)$	2nd order, central difference
	$f'''(x_0) = \frac{-3f_4 + 14f_3 - 24f_2 + 12f_1 - 5f_0}{(\Delta x)^3} + O((\Delta x)^2)$	2nd order, forward* difference

* To obtain the backward difference approximation for odd-order derivatives, multiply the forward difference equation by –1 and make all non-zero subscripts negative; e.g., the 1st-order backward difference approximation to the first derivative is $f'(x_0) = (f_0 - f_{-1})/\Delta x$

§ To obtain the backward difference approximation for even-order derivatives, make all non-zero subscripts negative.

3.4. *MATLAB* methods for finding derivatives

MATLAB has no built-in derivative functions except `diff`, which differences a vector; `diff(f)` is equivalent to Δf. A simple forward-difference estimate of the derivative is given by `diff(f)./diff(x)`, where f are the function values at the n points x. Note that if x has length n, `diff` returns a vector of length n-1. `diff` can be nested, so that $\Delta^2 f$ = `diff(diff(f))`. A centered difference estimate of a derivative can be obtained using:

```
x=x(:)'; %makes x a row vector to begin
xd=[x-h;x+h];
yd=f(xd); %define a function f or substitute the function for f
dfdx=diff(yd)./diff(xd);
```

MATLAB's symbolic math toolbox offers another option with `diff`. If we set `f='eqn'`, and `x` is the independent variable in `f`, then

```
dfdx=diff(f,'x')
```

returns the analytical expression for the derivative which can then be evaluated at any desired values of x.

For equally spaced points, the following expression provides an estimate for $f'(x_i)$ where x_i are values of x where f is known (see Gerald and Wheatley, 1999, or other texts on numerical analysis for details):

$$f'(x_i) = \frac{1}{\Delta x}\left[\Delta f_i - \frac{\Delta^2 f_i}{2} + \frac{\Delta^3 f_i}{3} - \ldots \frac{\Delta^n f_i}{n}\right]_{x=x_i}$$

This can be implemented in *MATLAB* to examine the improvement in the estimation of $f'(x)$ with additional terms and/or smaller values of Δx. The following m-file `deriv.m` does this for the function $f=\sin(x)$ and $0<x<2\pi$. The results are compared in Figure 3.3.

```
%deriv.m -- difference formulae for derivatives

dx=0.1;
x=0:dx:2.*pi;
f=sin(x);
n=length(x);

%first difference
df1=diff(f)./dx;
mean(abs(cos(x(1:n-1))-df1))
plot(x(1:n-1),cos(x(1:n-1))-df1,'-b')
hold on;

%second difference
df2=(diff(f(1:n-1))-diff(diff(f))./2)./dx;
mean(abs(cos(x(1:n-2))-df2))
plot(x(1:n-2),cos(x(1:n-2))-df2,'--r')

%third difference
df3=(diff(f(1:n-2))-diff(diff(f(1:n-1)))./2+diff(diff(diff(f)))./3)./dx;
mean(abs(cos(x(1:n-3))-df3))
plot(x(1:n-3),cos(x(1:n-3))-df3,'-g')
```

```
%smaller dx
dx2=0.01; x2=0:dx2:2.*pi;
f2=sin(x2); n2=length(x2);
df1=diff(f2)./dx2;  %first difference
plot(x2(1:n2-1),cos(x2(1:n2-1))-df1,'--c')
mean(abs(cos(x2(1:n2-1))-df1))

hold off;
legend('1st diff','2nd diff','3rd diff','1st diff, 0.1*dx')
xlabel ('x'); ylabel ('error in derivative estimate')
```

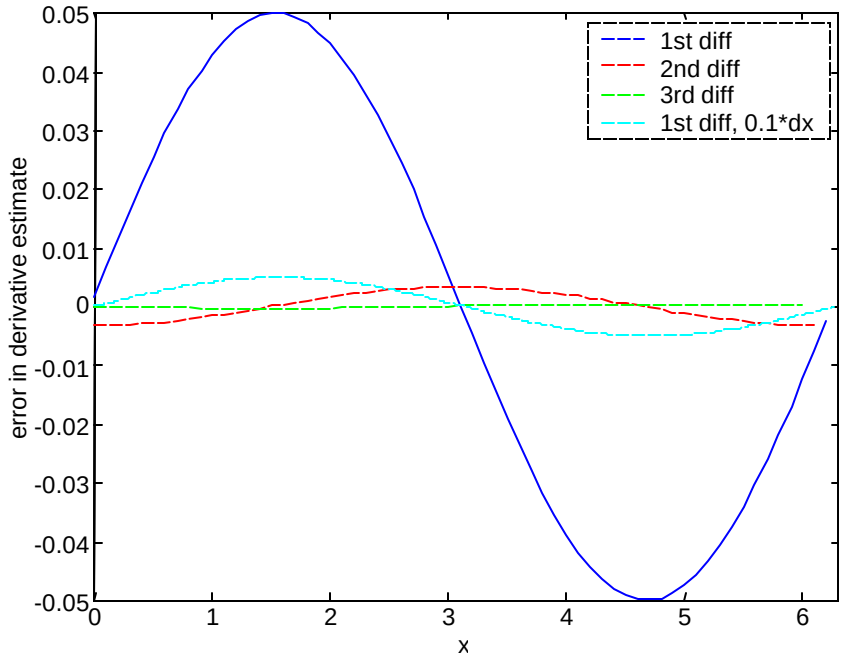

Figure 3.3. Accuracy of finite difference estimates of the derivative of sin(x).

3.5. Numerical integration

We will consider three methods of numerical integration: the trapezoidal rule, Simpson's rule(s), and Gaussian quadrature. *MATLAB* has functions `trapz`, `quad`, and `quadl` that perform numerical integration. In addition, the symbolic math function `int` finds integrals.

3.6. Trapezoidal rule

The simplest approach to numerically integrating a function f over the interval $[a,b]$ is to divide the interval into n subdivisions of equal width, $\Delta x=(b-a)/n$ and approximate f in each interval either by the value of f at the midpoint of the interval, which gives the rectangular rule, or by the average of the function over the interval, $\frac{1}{2}(f_{i+1} + f_i)$, which gives the *trapezoidal rule*. The rectangular rule approximates the function over each subinterval as a constant, while the trapezoidal rule makes a linear approximation to the function (Figure 3.4). Over each subinterval, the trapezoidal rule gives

$$\int_{x_i}^{x_{i+1}} f(x)dx \cong \frac{f(x_i)+f(x_{i+1})}{2}\Delta x = \frac{\Delta x}{2}(f_i + f_{i+1})$$

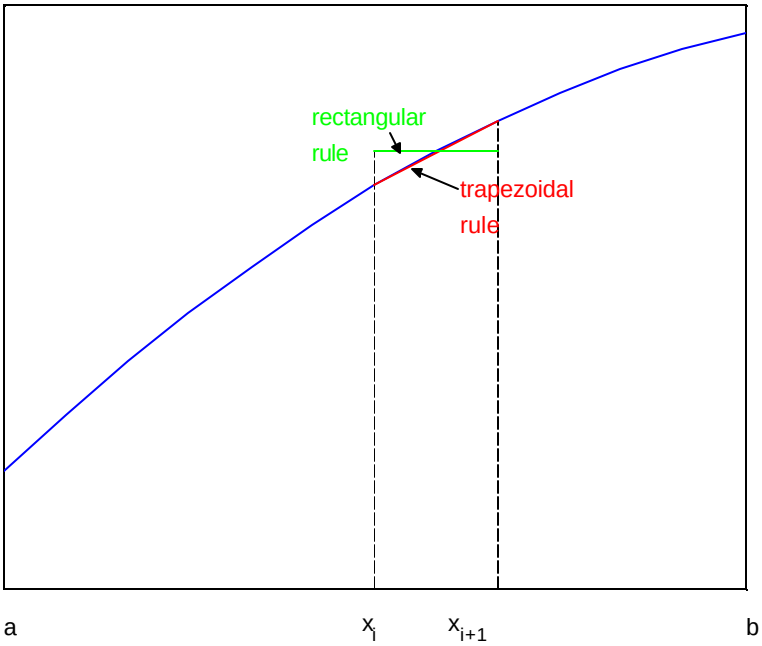

Figure 3.4. Trapezoidal and rectangular approximations of an integral.

If the interval [*a*,*b*] is subdivided into *n* subintervals of size Δx, then, over the whole interval, the trapezoidal rule gives

$$\int_a^b f(x)dx = \frac{\Delta x}{2}(f_a + 2f_2 + 2f_3 + \ldots + 2f_n + f_b)$$

This is called the composite trapezoidal rule. However, it is not necessary for the subintervals to be equally spaced when applying the trapezoidal rule. This and its simplicity make it a valuable technique.

As we would expect, the errors associated with the trapezoidal rule depend on the step size. We need to consider two errors in this case. The first is the *local error* for each step, which is $O((\Delta x)^3)$. Generally, the trapezoidal rule is applied over an interval comprising *n* equal steps. The total error, or *global error* is given by the sum of the local errors, and can be shown to be $O((\Delta x)^2)$ [Box 3.1].

The trapezoidal rule is simple, but uses only a linear approximation between successive points to estimate the integral. We could expect better accuracy if we instead approximated the function over two adjacent subintervals of equal width using a quadratic. This can be done with interpolating polynomials.

First, consider the interpolating polynomial $P_1(x) = f_i + s\Delta f_i$. If we integrate $P_1(x)$ between x_0 and x_1, noting that $dx = \Delta x\, ds$, we get

$$\int_{x_0}^{x_1} f(x)dx \cong \Delta x \int_{s=0}^{s=1} (f_0 + s\Delta f_0)ds = \Delta x f_0 s\,|_0^1 + \Delta x \Delta f_0 \frac{s^2}{2}|_0^1 = \Delta x(f_0 + \frac{1}{2}\Delta f_0)$$

$$= \frac{\Delta x}{2}[2f_0 + (f_1 - f_0)] = \frac{\Delta x}{2}(f_0 + f_1)$$

This is the trapezoidal rule. The same approach can be used to develop higher order methods such as Simpson's rules.

3.7. Simpson's rules

Following the same procedure as above with the polynomial $P_2(x)$ gives us

$$\int_{x_0}^{x_2} f(x)dx \cong \Delta x \int_0^2 (f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0)ds = \Delta x(2f_0 + 2\Delta f_0 + \frac{1}{3}\Delta^2 f_0)$$

$$= \frac{\Delta x}{3}(f_0 + 4f_1 + f_2)$$

The local error term is $O((\Delta x)^5)$. If we did the same thing using a cubic approximation over three adjacent subintervals, we would obtain

$$\int_{x_0}^{x_3} f(x)dx = \int_{x_0}^{x_3} P_3(x)dx = \frac{3\Delta x}{8}(f_0 + 3f_1 + 3f_2 + f_3)$$

with a local error $O((\Delta x)^5)$; this is Simpson's 3/8 rule. Notice that adding the extra point into the formula does not increase the order of accuracy of the approximation.

The first of these equations (based on a quadratic) is called Simpson's 1/3 rule. The corresponding composite formula for the integral over the interval $[a,b]$ is

$$\int_a^b f(x)dx = \frac{\Delta x}{3}(f_a + 4f_1 + 2f_2 + 4f_3 + 2f_4 + ... + 4f_{n-1} + f_b)$$

with a global error term of $O((\Delta x)^4)$. Because the method uses pairs of panels, the number of panels (subintervals) must be even. If the number of panels is uneven, another rule, e.g. Simpson's 3/8 rule, could be used at one end and the 1/3 rule over the remaining even number of panels. Alternatively, the size of the panels can be adjusted to accommodate an even number of panels.

The trapezoidal rule and Simpson's rules are examples of the general family of Newton-Cotes integration formulas. The general form is

$$\int_a^b f(x)dx \cong \int_a^b P_n(x)dx$$

with an error of $O((\Delta x)^{n+1})$. For interpolating polynomials of order 1, 2, and 3, the Newton-Cotes formulas give the trapezoidal rule, Simpson's 1/3 rule, and Simpson's 3/8 rule, respectively. If the degree of the interpolating polynomial is too high, errors due to round-off

and local irregularities can cause a problem [Box 3.1]. This is why usually only the lower-degree Newton-Cotes formulas are used.

3.8. Gaussian quadrature

Another common method for numerical integration is Gaussian quadrature. Although this method is most easily derived using the method of undetermined coefficients (e.g., see Gerald and Wheatley, 1999), we can gain an appreciation for its origin by recognizing that the methods we've described so far all have the form

$$\int_a^b f(x) \cong \sum_{i=1}^{n} w_i \, f(x_i)$$

where w_i are weights assigned to values of $f(x_i)$ in the integration formulas. The Newton-Cotes methods are based on evenly spaced values of x. However, we could let the x's be free parameters in our attempt to fit a polynomial through the function. This requires that $f(x)$ is known explicitly so it can be evaluated at any desired value of x. Using this approach, we can improve the accuracy of, e.g., a two-point method over that attainable with the trapezoidal rule. With two points, x_1 and x_2, and two weights, a and b, we can fit exactly polynomials of order 0, 1, 2, and 3. To simplify the calculations, we will evaluate the integrals over the interval [-1 1]. A transformation can be used to change these limits to those for any other bounded region.

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) + w_2 \, f(x_2)$$

We require this formula to be exact for polynomials of order three or less, including $f(x)=x^3$, $f(x)=x^2$, $f(x)=x$, and $f(x)=1$. Substituting these into the equation above gives,

$$\int_{-1}^{1} x^3 dt = 0 = w_1 x_1^{\,3} + w_2 x_2^{\,3}$$

$$\int_{-1}^{1} x^2 dx = 2/3 = w_1 x_1^{\,2} + w_2 x_2^{\,2}$$

$$\int_{-1}^{1} x dx = 0 = w_1 x_1 + w_2 x_2$$

$$\int_{-1}^{1} dx = 2 = w_1 + w_2$$

We now have four equations for the four unknown parameters, w_1, w_2, x_1, and x_2. Solving these gives $w_1 = w_2 = 1$, and $x_2 = -x_1 = \sqrt{1/3} = 0.5773$. Substituting these values back into $\int f = w_1 x_1 + w_2 x_2$ results in

$$\int_{-1}^{1} f(x)dx \cong f\left(\frac{-1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

This sum gives the exact integral of any cubic over the interval from -1 to 1. If the limits are $[a,b]$ rather than [-1,1], then it is necessary to use the linear transformation $t = [(b-a)x + b+a]/2$, $dt = [(b-a)/2]dx$, which gives

$$\int_a^b f(t)dt = \frac{(b-a)}{2}\int_{-1}^{1} f\left(\frac{(b-a)\,x+b+a}{2}\right)dx$$

Gaussian quadrature can be extended to include more than two points. The general expression has the form

$$\int_{-1}^{1} f(x)dx \cong \sum_{i=1}^{n} w_i f_i(x_i)$$

and is exact for functions that are polynomials of degree 2*n*-1 or less. A method for determining the weights w_i and the x_i's uses Legendre polynomials (see Gerald and Wheatley, 1999, or other texts on numerical analysis for details).

3.9. *MATLAB* methods

As noted previously, *MATLAB* has built-in functions to integrate using the trapezoidal rule and variants of Simpson's rule and another higher-order approximation.

trapz: `z=trapz(x,y)` or `z=trapz(y)` computes the integral of *y* with respect to *x* using trapezoidal integration. `trapz(y)` assumes unit spacing between data points. For other spacings, multiply *z* by the actual interval width. `trapz(x,y)` can be used for unequally spaced grids.

quad: `a=quad('fname',a,b,[tol],[trace])` approximates the integral of a function over the interval [*a*,*b*] using quadrature. The default tolerance is 1E-3. The function `fname` must return a vector of output values when given a vector of input values. `quad` uses a "recursive adaptive, Simpson's rule"; `quadl` uses high order recursive, adaptive Lobatto quadrature. Neither can integrate over singularities (essentially, places where the function is undefined).

3.10. Problems

1. Use forward, backward, and central difference approximations to numerically differentiate $\sqrt{x}$ over the range $0<x<2$. Compare the answers to the exact solutions at x=0.2, 0.5, 2 for Δx=0.05 and 0.02. How are the errors affected by the method and step size?

2. The Gaussian error function *erf*(x) is

$$erf(x) = \frac{2}{\sqrt{p}} \int_0^x e^{-t^2} dt$$

   a) Write an m-file implementing Simpson's 1/3 rule.

   b) Use the m-file to evaluate *erf*(x) between 0 and 5 (inclusive) with Δx = 0.1.

   c) How small does Δx have to be for the answer to be correct to 4 decimal places (error<0.00005) at x = 1? Note that *MATLAB* has a built-in function `erf` that you can use to check the accuracy of your answer. [*MATLAB*'s `erf` function also uses a numerical solution, but it is accurate to 1E-16.]

   d) Compare your answer to the results of `trapz` for the same Δx and `quad` or `quadl` for the same level of error.

3. The velocity distribution at the centerline of a steady, uniform channel flow is approximately given by the equation,

$$u(z) = \frac{u_*}{k}\left(\ln(z) - \ln(z_0)\right)$$

   where the shear velocity $u_* = \sqrt{ghS}$, S is channel slope, $k = 0.41$, z is level above the bottom, h is flow depth, and z_0 is the level close to the bed where the velocity u = 0. Use this velocity equation to generate values of velocity at 0.1h intervals for a flow with h = 0.7 m, $S = 2.5\times10^{-4}$, $z_0 = 1\times10^{-4}$ m. We will consider these to be our "data".

   a) Depth-averaged velocity $< u >$ is defined as $h^{-1}\int_0^h u(z)dz$. For the logarithmic velocity profile indicated above, the exact integral $< u >= u_* k^{-1}[\ln(0.367h) - \ln(z_0)]$. For wide, rectangular channels, this is a good approximation to the mean velocity in the channel. Numerically integrate the velocity "data" generated above to estimate the depth-averaged velocity using `trapz`. Compare the results to the exact integral.

   b) The exact derivative of the velocity profile is $u_*/(kz)$. Differentiate the profile "data" using centered differences and compare to the exact value.

   c) Add some noise to the data to represent measurement/instrument error. This can be done using the *MATLAB* command `un=u+a*randn(size(z))` where `a` is the amplitude of the noise and `z` is the vector of vertical positions; let `a` = 0.1. Now,

repeat the integration and differentiation of parts a) and b) to see how much the addition of this noise affects the results.

d) The most efficient way to get a good numerical estimate of $\int u dz$ for a logarithmic velocity profile is to perform the integration on a logarithmically spaced grid for which the grid spacing close to the bottom is smaller than that near the surface. Compare the results of integrating the velocity profile (using `trapz`) over 20 equally spaced points and 20 logarithmically spaced points to the exact solution.

3.11. References

Gerald C.F. and P.O. Wheatley, *Applied Numerical Analysis*, 319 pp., Addison Wesley, Reading, MA, 1999.

Box 3.1. Errors in numerical methods

There are two principal sources of error in numerical computation. One is due to the fact that an *approximation* is being made (e.g., a derivative is being approximated by a finite difference). These errors are called *truncation errors*. The second is due to the fact that computers have limited precision, i.e., they can store only a finite number of decimal places. These errors are called *roundoff errors*.

Truncation errors arise when a function is approximated using a finite number of terms in an infinite series. For example, truncated Taylor series are the basis of finite difference approximations to derivatives (Chapter 3.2). The error in a finite difference approximation to a derivative is a direct result of the number of terms retained in the Taylor series (i.e., where the series is truncated). Truncation error is also present in other numerical approximations. In numerical integration, for example, when each increment of area under a curve is calculated using a polynomial approximation to the true function (Chapters 3.6-3.7), truncation errors arise that are related to the order of the approximating polynomial. For example an n^{th}-order polynomial approximation to a function results in an error in the integral over an increment Δx of $O(\Delta x)^{n+2}$ *(local* error*)*. When the integrals over each increment are summed to approximate the integral over some domain $a \le x \le b$, the local errors sum to give a *global* error of $O(\Delta x)^{n+1}$. Truncation errors decrease as step size (Δx) is decreased – the finite difference approximation to a derivative is better (has lower truncation error) when Δx is "small" relative to when Δx is "large."

Roundoff errors stem from the fact that computers have a maximum number of digits that can be used to express a number. This means that the machine value given to fractional numbers without finite digit representations, for example, 1/3 = 0.33333333…, will be rounded or chopped at the precision of the computer. It also means that there is a limit to how large or small a number a computer can represent in floating point form. For many computations, the small changes in values resulting from roundoff are insignificant. However, roundoff errors can become important. For example, subtraction of two nearly identical numbers (as occurs when computing finite differences with very small values of Δx) can lead to relatively large roundoff error depending on the number of significant digits retained in the calculation. Interestingly, this means that approximations to derivatives will be improved by reducing Δx to some level because truncation errors are reduced, but that further decreases in Δx will make the estimate *worse* because roundoff error becomes large and dominates for very small values of Δx. Roundoff error can also complicate some logical operations that depend on establishing equality between two values if one or both are the result of computations that involved chopping or rounding.

Finally, in the hydrological sciences, measured data are often used in a calculation. For example, one might want to find the derivative of water velocity with respect to height above a streambed using numerical differentiation of data measured using a flowmeter. Such data are subject to *measurement errors*, which are then inserted into any numerical computation in which they are used.