Derivatives, Finding Roots, Interpolating and Integration

Doing Numerical Derivatives

Although taking derivatives is a very easy operation for all of you, there are situations where it must be done numerically. For example, suppose you are given experimental data points, rather than a known mathematical function. Or, suppose you have a complicated mathematical function (a Bessel function, or the like) where you don't know the derivative formula or it is too difficult to work out (later when we study Mathematica we will se that it can symbolically work out many derivatives)

An algorithm is the specific procedure you follow to calculate a desired result, i.e., when you find the roots x_1 and x_2 of

$$ax^2 + bx + c = 0$$

by calculating

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

you are using an algorithm.

Using guesswork and intuition to put the equation in the form

$$(x - x_1)(x - x_2) = 0$$

would be a different algorithm. The word comes from a book by a great Persian mathematician of the 9th century, Abu al-Khwarizmi; we also get the word "**algebra**" from him.

Imagine that you have an experiment that produces some discrete values of the position of a particle. So there is some unknown function x(t) for which you have only numerical values at a discrete set of points. You would, however, like to calculate the speed of the particle v(t) - this means that you need to do a numerical derivative.

Suppose that we are interested in the derivative at x = 0, x'(0), of some function x(t) and also suppose that, as stated above. we only know the function x(t) on an equally spaced lattice of t-values.

$$x_n = x(t_n) \quad , \quad t_n = nh \quad , \quad n = 0,1,2,3.....$$

We begin by using a Taylor series to expand the function x(t) in the neighborhood of t = 0.

$$x(t) = x(0) + x'(0)t + \frac{x''(0)}{2!}t^2 + \frac{x'''(0)}{3!}t^3 + ...$$

We then have

$$x_{+1} = x(t_1) = x(h) = x(0) + x'(0)h + \frac{h^2}{2!}x''(0) + \frac{h^3}{3!}x'''(0) + O(h^4)$$

$$x_{+2} = x(t_2) = x(2h) = x(0) + 2x'(0)h + 2h^2x''(0) + \frac{4h^3}{3}x'''(0) + O(h^4)$$

$$x_{-1} = x(t_{-1}) = x(-h) = x(0) - x'(0)h + \frac{h^2}{2!}x''(0) - \frac{h^3}{3!}x'''(0) + O(h^4)$$

$$x_{-2} = x(t_{-2}) = x(-2h) = x(0) - 2x'(0)h + 2h^2x''(0) - \frac{4h^3}{3}x'''(0) + O(h^4)$$

where $O(h^4)$ means terms of order h^4 or higher (h small) have been neglected. We then have

$$x'(0) = \frac{x_1 - x_0}{h} + O(h) = \frac{x(h) - x(0)}{h} + O(h)$$

which is called a **2-point formula** for the derivative at t = 0. It is also called the **forward difference algorithm.**

Alternatively we have

$$x'(0) = \frac{x_0 - x_{-1}}{h} + O(h) = \frac{x(0) - x(-h)}{h} + O(h)$$

Using these two results we can derive a more accurate **3-point formula** or **central difference algorithm.**

$$\frac{1}{2}\left(\frac{x_1 - x_0}{h} + \frac{x_0 - x_{-1}}{h}\right) = x'(0)$$

$$\frac{x_1 - x_{-1}}{2h} = \frac{1}{2h}(x(0) + x'(0)h + \frac{h^2}{2!}x''(0)....- (x(0) - x'(0)h + \frac{h^2}{2!}x''(0) + ...))$$

$$= x'(0) + O(h^2)$$

or

$$x'(0) = \frac{x_1 - x_{-1}}{2h} + O(h^2)$$

In a similar manner, it is possible to improve on the 3-point formula by relating x'(0) to grid points futher removed from x = 0. The 5-point formula is

$$x'(0) = \frac{1}{12h}\left(x_{-2} - 8x_{-1} + 8x_1 - x_2\right) + O(h^4)$$

Formulas for higher derivatives can be constructed by taking appropriate combinations, for example,

$$x''(0) = \frac{1}{h^2}\left(x_1 - 2x_0 + x_{-1}\right) + O(h^2)$$

Finally, we give a table of higher-order derivatives (5-point formulas using central-difference formulas):

$$x'(0) = \frac{1}{12h}\left(x_{-2} - 8x_{-1} + 8x_1 - x_2\right)$$

$$x''(0) = \frac{1}{12h^2}\left(-x_{-2} + 16x_{-1} - 30x_0 + 16x_1 - x_2\right)$$

$$x'''(0) = \frac{1}{8h^3}\left(x_{-3} - 8x_{-2} + 13x_{-1} - 13x_1 + 8x_2 - x_3\right)$$

$$x''''(0) = \frac{1}{6h^4}\left(-x_{-3} + 12x_{-2} - 39x_{-1} + 56x_0 - 39x_1 + 12x_2 - x_3\right)$$

Derivatives at other values of t (other than t = 0) are obtained by translation:

$$x'(0) = \frac{x(h) - x(-h)}{2h} \Rightarrow x'(t) = \frac{x(t+h) - x(t-h)}{2h}$$

$$x''(0) = \frac{1}{h^2}\left(x(h) - 2x(0) + x(-h)\right) \Rightarrow x''(t) = \frac{1}{h^2}\left(x(t+h) - 2x(t) + x(t-h)\right)$$

Sample Code for Numerical derivatives

```
function y = fp(x)
y= sin(x)*exp(x);
end

% fivepoint derivatives
h=input('Enter stepsize - (0 < h < 0.1): ' );
x = 2;
d1 = (fp(x-2*h)-8*fp(x-h)+8*fp(x+h)-fp(x+2*h))/(12*h);
d2 = (-fp(x-2*h)+16*fp(x-h)-30*fp(x)+16.0*fp(x+h) ...
        -fp(x+2*h))/(12*h^2);
d3 = (fp(x-3*h)-8*fp(x-2*h)+13*fp(x-h)-13*fp(x+h) ...
        +8*fp(x+2*h)-fp(x+3*h))/(8*h^3);
d4 = (-fp(x-3*h)+12*fp(x-2*h)-39*fp(x-h)+56*fp(x) ...
        -39*fp(x+h)+12*fp(x+2*h)-fp(x+3*h))/(6*h^4);
[d1,d2,d3,d4]
```

```
>> fivepoint
Enter stepsize - (0 < h < 0.1): .5

ans = 3.6694   -6.1686  -19.8818  -27.0642

>> fivepoint
Enter stepsize - (0 < h < 0.1): .1

ans = 3.6440   -6.1499  -19.5880  -26.8757

>> fivepoint
Enter stepsize - (0 < h < 0.1): .01

ans = 3.6439   -6.1499  -19.5876  -26.8754
```

Clearly we are seeing convergence. The exact answers are:

```
      3.6439   -6.1499  -19.5876  -26.8754
```

Finding Zeroes or Roots

The solution of an arbitrary equation f(x) = 0 is commonly called the “root” or "zero" of the function f(x). More specifically, if f(x) is continuous on an interval [a, b] and the signs of f(a) and f(b) differ, then for some x in [a, b], say x = z, f(z) = 0 and z is the root of the function. There are a variety of iterative methods for finding roots of functions and we shall illustrate a few of them below:

Crude Home-In Method (CHIM)

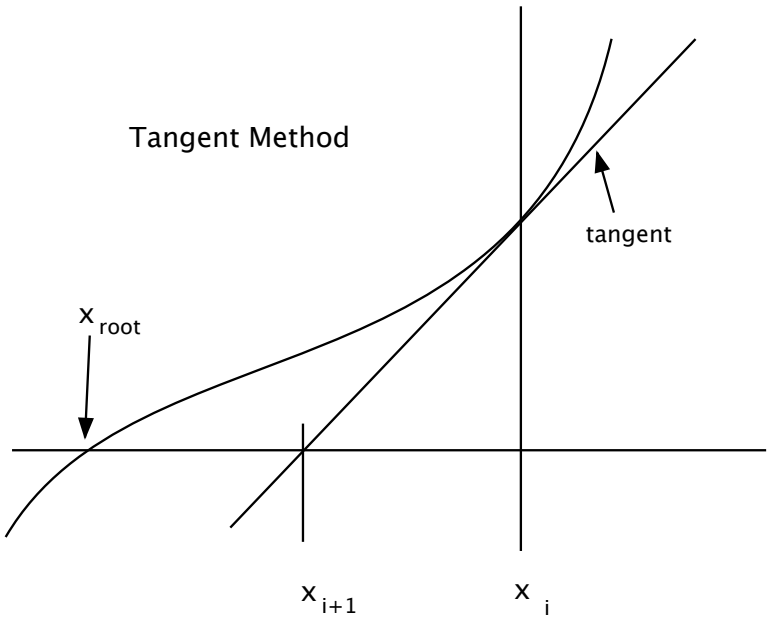
In this method we need to know an approximate value of the root. The procedure then is:

(1) pick x < xroot (guess it) and calculate f(x)
(2) let x → x + D, calculate f(x)
(3) continue this process until f(x) changes sign
(4) back up 1 step, decrease the step size, D → D/10
(5) return to step (2)
(6) decide when to stop

If we can compute(analytically) the derivative f’(x), then another technique is used.

Newton - Raphson Tangent Method (NRTM)

This method generates a sequence of values x_i converging to x_{root}. Using the diagram below NRTM uses



The NRTM uses the equation

$$y_{\text{tangent line}} = f(x_i) + f'(x_i)(x - x_i) \rightarrow y_{root} = 0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

or

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Any value x_0 can be used to start the process.

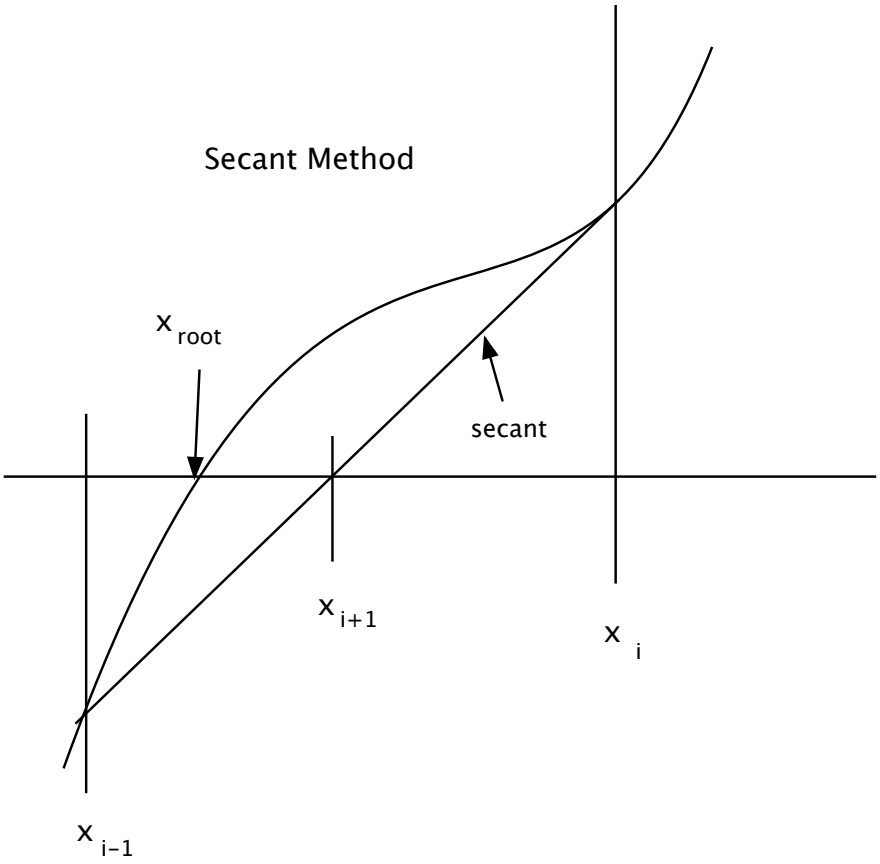
A second version does not require an analytic formula for the derivative.

Newton - Raphson Secant Method (NRSM)

This method generates a sequence of values x_i converging to x_{root}. Using the diagram below, NRSM uses

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$ (as an approximation for the slope in NRTM method)

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$



Any two values x_1 and x_2 can be used to start the process

When the function is badly behaved near its root, i.e., there is an inflection point or several nearby roots, the NR methods can fail to converge at all or converge to the wrong answer depending on your starting point. A safe procedure is to use the CHIM method to get close and then us one of the NR methods to get the exact result.

Sample Code for Finding Roots

```
function z=nuclear(x)
z=tan(0.1*x)-9.2*exp(-x);

% CHIM method
% 0,10,1,.001
xmin=input('Enter xmin : ' );
xmax=input('Enter xmax : ' );
step=input('Enter step : ' );
tol=input('Enter tol : ' );
```

```
while ((xmax-xmin) >= tol)
 x=xmin;
 ch0=sign(nuclear(x));
 ch=ch0;
 while (ch == ch0)
  x=x+step;
  ch0=ch;
  ch=sign(nuclear(x));
 end
 xmax=x;
 xmin=x-step;
 step=step/10;
end
[xmin,xmax,nuclear(xmin)]

function z=nuclear(x)
z=tan(0.1*x)-9.2*exp(-x);

function z=dnuclear(x)
z=0.1*(1.0/cos(0.1*x))^2+9.2*exp(-x);

% NRTM Method
% 0, 0.001
x1=input('Enter x1 : ' );
tol=input('Enter tol : ' );
xnew = x1;
xold = xnew-10*tol;
while (abs(xold-xnew) >= tol)
 xold=xnew;
 xnew=xold-nuclear(xold)/dnuclear(xold);
end
[xold,nuclear(xold)]

function z=nuclear(x)
z=tan(0.1*x)-9.2*exp(-x);

% NRSM Method
% 0,10,0.001
x0=input('Enter x0 : ' );
x1=input('Enter x1 : ' );
tol=input('Enter tol : ' );
xold = x1;
xoldold = x0;
while (abs(xoldold-xold) >= tol)
 xnew=xoldold-nuclear(xoldold)*(xold-xoldold)/ ...
      (nuclear(xold)-nuclear(xoldold));
```

```
 if (abs(xnew-xold) > abs(xnew-xoldold))
  xoldold =xoldold;
  xold=xnew;
 else
  xoldold=xold;
  xold=xnew;
 end
end
[xoldold,xold,nuclear(xold)]
```

Interpolating Data

Lagrange Interpolation

The method of Lagrange Interpolation can be used to approximate a function everywhere even if values of the function are only known at a finite set of points. We derive Lagrange's three-point interpolating polynomial $p(x)$ from a Taylor series by expressing the function at x_1 and x_2 in terms of the function and its derivatives:

$$f(x_1) = f(x) + (x_1 - x)f'(x) + ...$$
$$f(x_2) = f(x) + (x_2 - x)f'(x) + ...$$

We would like to truncate the series and retain only the first two terms. But in doing so, the equality would be destroyed. We can, however, introduce approximations to the function and its derivatives such that the equality is retained. Consider the relations

$$f(x_1) = p(x) + (x_1 - x)p'(x)$$
$$f(x_2) = p(x) + (x_2 - x)p'(x)$$

where we must then have

$$p(x_1) = f(x_1) \text{ and } p(x_2) = f(x_2)$$

Some algebra then gives

$$p(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_1 - x_2} f(x_2)$$

which is a linear function of x. This function is the equation of the line passing through the points $(x_1, f(x_1)), (x_2, f(x_2))$. Note how the weighting factors emphasize each term at different places. Higher approximations (non-linear) are obtained by keeping and approximating more terms in the Taylor series.

Thus, if we know the function at three points,

$$f(x_1) = f(x) + (x_1 - x)f'(x) + \frac{(x_1 - x)^2}{2} f''(x) + \ .....$$

$$f(x_2) = f(x) + (x_2 - x)f'(x) + \frac{(x_2 - x)^2}{2} f''(x) + \ .....$$

$$f(x_3) = f(x) + (x_3 - x)f'(x) + \frac{(x_3 - x)^2}{2} f''(x) + \ .....$$

Truncating these equations(as above) and solving we get

$$p(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3)$$

These results can be generalized to give a general n-point interpolating polynomial of order (n-1) as

$$p(x) = \sum_{j=1}^{n} C_{j,n}(x) f(x_j)$$

where

$$C_{j,n}(x) = \frac{(x - x_1)(x - x_2).....(x - x_{j-1})(x - x_{j+1}).....(x - x_n)}{(x_j - x_1)(x_j - x_2).....(x_j - x_{j-1})(x_j - x_{j+1}).....(x_j - x_n)}$$

where

$$C_{j,n}(x_i) = \left\{ \begin{matrix} 1 & \text{i=j} \\ 0 & \text{i} \neq \text{j} \end{matrix} \right\} = \delta_{ij}$$

Sample Code for Lagrange Interpolation

```
% interpolating
t = [94.0, 205.0, 371.0];
rho = [929.0, 902.0, 860.0];
x = 251.0;
p1 = ((x-t(2))*(x-t(3))*rho(1))/((t(1)-t(2))*(t(1)-t(3)));
p2 = ((x-t(1))*(x-t(3))*rho(2))/((t(2)-t(1))*(t(2)-t(3)));
p3 = ((x-t(1))*(x-t(2))*rho(3))/((t(3)-t(1))*(t(3)-t(2)));
dens=p1 + p2 + p3
```

Numerically Doing Integrals

Now we are interested in numerically calculating the definite integral of a function f(x) between two limits a < b. We define

$$N = \frac{b - a}{h}$$

such that N = an even integer and h = **step size.** Alternately, we can choose N(even) and compute the corresponding step size h.

Now we can write (by breaking up the range of integration into many smaller ranges)

$$\int_a^b f(x)dx = \int_a^{a+2h} f(x)dx + \int_{a+2h}^{a+4h} f(x)dx + ...+ \int_{b-4h}^{b-2h} f(x)dx + \int_{b-2h}^{b} f(x)dx$$

Therefore we need only calculate the integral

$$g(0) = \int_{-h}^{h} f(x)dx$$

and the general formula can be calculated from this result using the relation

$$g(q) = \int_{q-h}^{q+h} f(x)dx$$

which follows from translation and gives

$$\int_a^b f(x)dx = g(a+h) + g(a+3h) + g(a+5h) + ...+ g(b-3h) + g(b-h)$$

The basic idea here(this is the so-called **Newton-Cotes method**) is to approximate f(x) between -h and h by a function that can be integrated exactly so we can evaluate g(0) exactly and then use that exact result to generate an approximation for the entire integral.

For example, the simplest approximation is given by considering the intervals [-h,0] and [0,h] separately and assuming that the function f(x) is **constant** in each region of these intervals. In this case we have

$$g(0) = \int_{-h}^{h} f(x)dx = \int_{-h}^{0} f(x)dx + \int_{0}^{h} f(x)dx = \int_{-h}^{0} f(-h)dx + \int_{0}^{h} f(0)dx$$
$$= h\big(f(-h) + f(0)\big)$$

which is the well-known "**rectangle**" rule because we are summing rectangular areas. This gives for the entire integral

$$\int_a^b f(x)dx = g(a+h) + g(a+3h) + g(a+5h) + ...+ g(b-3h) + g(b-h)$$
$$= h\big(f(a) + f(a+h)\big) + h\big(f(a+2h) + f(a+3h)\big) + h\big(f(a+4h) + f(a+5h)\big) + ...$$
$$+h\big(f(b-4h) + f(b-3h)\big) + h\big(f(b-2h) + f(b-h)\big)$$

or

$$\int_a^b f(x)dx = h(f(a)+f(a+h)+f(a+2h)+f(a+3h)+f(a+4h)+f(a+5h)+.....$$
$$+f(b-4h)+f(b-3h)+f(b-2h)+f(b-h))$$

The next simplest approximation can be had by considering the intervals [-h,0] and [0,h] separately and assuming that the function f(x) is **linear** in each region of these intervals. The approximate integral is then

$$g(0)=\int_{-h}^{h} f(x)dx = \int_{-h}^{0}\left(f(-h)+\frac{f(0)-f(-h)}{h}(x+h)\right)dx + \int_{0}^{h}\left(f(0)+\frac{f(h)-f(0)}{h}x\right)dx$$
$$=\frac{h}{2}\left(f(h)+2f(0)+f(-h)\right)$$

which is the well-known "**trapezoidal**" rule because we are summing trapezoidal areas. This gives

$$\int_a^b f(x)dx = g(a+h)+g(a+3h)+g(a+5h)+.....+g(b-3h)+g(b-h)$$
$$=\frac{h}{2}\left(f(a+2h)+2f(a+h)+f(a)\right)+\frac{h}{2}\left(f(a+4h)+2f(a+3h)+f(a+2h)\right)+.....$$
$$+\frac{h}{2}\left(f(b-4h)+2f(b-3h)+f(b-2h)\right)+\frac{h}{2}\left(f(b-2h)+2f(b-h)+f(b)\right)$$

or

$$\int_a^b f(x)dx = \frac{h}{2}(f(a)+2f(a+h)+2f(a+2h)+2f(a+3h)+2f(a+4h)+.....$$
$$+2f(b-4h)+2f(b-3h)+2f(b-2h)+2f(b-h)+f(b))$$

An even better approximation can be had by realizing that a Taylor series can provide an improved interpolation of the function f(x)

We have

$$f(x)=f(0)+\frac{f(h)-f(-h)}{h}x+\frac{f(h)-2f(0)+f(-h)}{2h^2}x^2+O(x^3)$$

which gives

$$\int_{-h}^{h} f(x)dx = \frac{h}{3}(f(h)+4f(0)+f(-h))$$

This is "**Simpson's**" Rule. It is accurate to two orders higher than the trapezoid rule. It corresponds to approximating f(x) by a **parabola.**

Using this result we get

$$\int_a^b f(x)dx = \frac{h}{3}\big(f(a+2h)+4f(a+h)+f(a)\big)+\frac{h}{3}\big(f(a+4h)+4f(a+3h)+f(a+2h)\big)+.....$$
$$+\frac{h}{3}\big(f(b-4h)+4f(b-3h)+f(b-2h)\big)+\frac{h}{3}\big(f(b-2h)+4f(b-h)+f(b)\big)$$

or

$$\int_a^b f(x)dx = \frac{h}{3}(f(a)+4f(a+h)+2f(a+2h)+4f(a+3h)+2f(a+4h)+.....$$
$$+2f(b-4h)+4f(b-3h)+2f(b-2h)+4f(b-h)+f(b))$$

Remember, the number of intervals must be even(or the number points is odd).

Sample Codes to Implement Rectangle Rule

```
function y=tfofx(x)
% integration function
y= x.*exp(x);

% rectangle rule using a for loop
n=100;b=2;a=0;h=(b-a)/n;intrect=0;
for x=0:h:(n-1)*h
  intrect=intrect+tfofx(x)*h;
end
intrect

% rectangle rule - no loops (vectorized)
n=100;b=2;a=0;h=(b-a)/n;x=h*(0:n-1);
f=tfofx(x);
intrect=h*sum(f);
intrect
```

Vectorized Trapezoid and Simpson Rule(two wsys) Codes

```
% Numerical Integration
n=100;
b=2;
a=0;
h=(b-a)/n;
x=h*(0:n);
f=tfofx(x);
% trapezoid rule
wtrap=[1,2*ones(1,n-1),1];
inttrap=(h/2)*sum(wtrap.*f);
% simpson rule
w2=2*ones(1,n+1);
```

```
w2(1:2:n+1)=zeros(1,n/2+1);
wsimp=[1,2*ones(1,n-1),1]+w2;
intsimp=(h/3)*sum(wsimp.*f);
[inttrap,intsimp]

% Numerical Integration
n=1000;
b=2;
a=0;
h=(b-a)/n;
x=h*(0:n);
f=tfofx(x);
%trapezoid rule
wtrap=[1,2*ones(1,n-1),1];
inttrap=(h/2)*sum(wtrap.*f);
% simpson rule
wsimp=[1,reshape((ones(1,(n-2)/2)'*[4,2])',1,n-2),4,1];
intsimp=(h/3)*sum(wsimp.*f);
[inttrap,intsimp]
```