ENGINEERING COMPUTATION Lecture 6

Computing derivatives and integrals

Stephen Roberts
Michaelmas Term

Topics covered in this lecture:

1. Forward, backward and central differences.
2. Revision of integration methods from Prelims
   a. Trapezium method
   b. Simpson's method

Estimating Derivatives

Derivatives- motivation

Engineers often need to calculate derivatives approximately, either from data or from functions for which simple analytic forms of the derivatives don't exist.

Examples:

- Motion simulation, such as in flight simulators solving $\ddot{\mathbf{x}} = \mathbf{Forces}$ equations..
- estimation of rates of change of measured signals.
- control systems, where e.g. velocity signals are wanted from position encoder data.
- Medical signal & image processing – analysis and visualisation

Most methods derive from the basic derivation of differentiation of a function *f*(*t*):

$$f' = \frac{\mathrm{d}f}{\mathrm{d}t} = \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t}.$$

Forward difference

If a function (or data) is sampled at discrete points at intervals of length *h*, so that

$f_n = f(nh)$, then the forward difference approximation to f' at the point *nh* is given by

$$f_n' \approx \frac{f_{n+1} - f_n}{h}.$$

How accurate is this approximation? Obviously it depends on the size of *h*.

Use the Taylor expansion of f_{n+1}:

$$\begin{aligned}\frac{f_{n+1} - f_n}{h} &= \frac{f(nh+h) - f(nh)}{h} \\ &= \frac{\left(f(nh) + hf'(nh) + \frac{h^2}{2} f''(nh) + \cdots\right) - f(nh)}{h} \\ &\approx f'(nh) + \frac{h}{2} f''(nh) = f'(nh) + O(h)\end{aligned}$$

Clearly the order of approximation is $O(h)$. Lets check this out with the derivative of $f(t)=e^t$ at the point $t = 0$.

Clearly, the exact answer is $f'(0)=1.00000$. Using a spreadheet, we get:

Forward difference formula		
h	***f'***	**error**
1.000E+00	1.718E+00	7.183E-01
1.000E-01	1.052E+00	5.171E-02
1.000E-02	1.005E+00	5.017E-03
1.000E-04	1.000E+00	5.000E-05
1.000E-05	1.000E+00	5.000E-06
1.000E-12	1.000E+00	8.890E-05
1.000E-16	0.000E+00	-1.000E+00

Note that the error does behave as $O(h)$ until very small h, when rounding errors cause problems.

Of course with data (say accurate to .± 1.0%) this could cause problems much earlier.

Much of the error comes from the asymmetry of the approximation, where the points used are on one side of the point where the derivative is sought.

Backward difference

This follows a similar line of argument but we step backwards from $f_n = f(nh)$ rather than forward. Thus the backward difference formula is

$$f_n' \approx \frac{f_n - f_{n-1}}{h}$$

The error behaves as before. Why is this more useful than the forward difference in practice?

Central differences

We can improve matters by taking a symmetric approximation:

$$f'(nh) \approx \frac{f_{n+1} - f_{n-1}}{2h} = \frac{f((n+1)h) - f((n-1)h)}{2h}$$

$$= \frac{\left(f(nh) + hf'(nh) + \frac{h^2}{2} f''(nh) + \frac{h^3}{6} f'''(nh)\cdots\right) - \left(f(nh) - hf'(nh) + \frac{h^2}{2} f''(nh) - \frac{h^3}{6} f'''(nh)\cdots\right)}{2h}$$

$$\approx f'(nh) + \frac{h^2}{6} f'''(nh) = f'(nh) + O(h^2)$$

Here the error is $O(h^2)$, clearly much improved in our example, although the penalty for making h too small remains:

Central difference formula		
h	***f'***	**error**
1.000E+00	1.175E+00	1.752E-01
1.000E-01	1.002E+00	1.668E-03
1.000E-02	1.000E+00	1.667E-05
1.000E-04	1.000E+00	1.667E-09
1.000E-05	1.000E+00	1.210E-11
1.000E-12	1.000E+00	3.339E-05
1.000E-16	5.551E-01	-4.449E-01

Forward difference formula		
h	***f'***	**error**
1.000E+00	1.718E+00	7.183E-01
1.000E-01	1.052E+00	5.171E-02
1.000E-02	1.005E+00	5.017E-03
1.000E-04	1.000E+00	5.000E-05
1.000E-05	1.000E+00	5.000E-06
1.000E-12	1.000E+00	8.890E-05
1.000E-16	0.000E+00	-1.000E+00

N-point Formulae

The central difference equation is an example of a **three-point formula** – it gets its name from the fact that it uses a 3x1 neighbourhood about a point.

$$f^{'}(nh) = \frac{f_{n+1} - f_{n-1}}{2h}$$

You can show that the extended **five-point formula**

$$\dot{f}_n \approx \frac{f_{n-2} - 8f_{n-1} + 8f_{n+1} - f_{n+2}}{12h}$$

is accurate to $O(h^4)$.

Formulae for Higher order derivatives

A forward difference approximation for the second derivative is:

$$f_n'' \approx \frac{f_{n+1}' - f_n'}{h} \approx \frac{1}{h}\left(\frac{f((n+2)h) - f((n+1)h)}{h} - \frac{f((n+1)h) - f((n)h)}{h}\right)$$

$$= \frac{f_{n+2} - 2f_{n+1} + f_n}{h^2} + O(h)$$

This is only accurate to order $O(h)$, so isn't much used. If the points used are shifted one to the left, then we get the more accurate central difference formula for the second derivative:

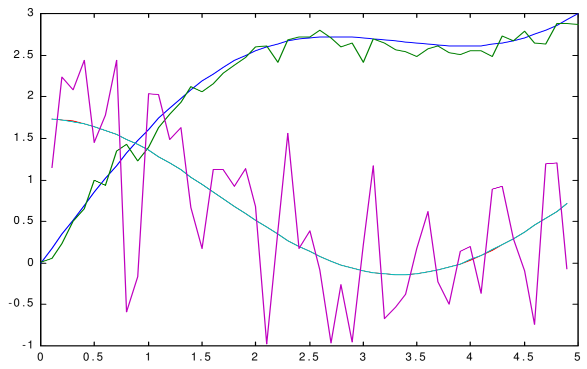
$$f_n'' \approx \frac{f_{n+1} - 2f_n + f_{n-1}}{h^2} + O(h^2)$$

Use Taylor series expansions to the term in $f^{(iv)}$ to show that the error is $O(h^2)$.

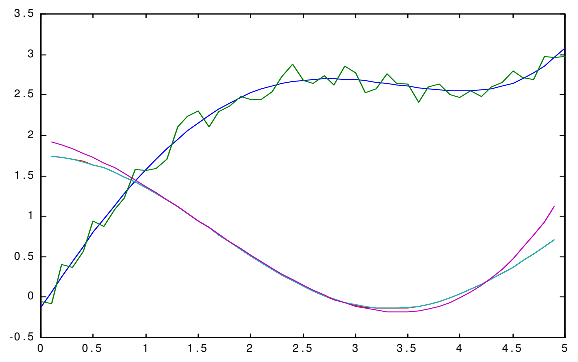
Differentiation and Noise

The numerical differentiation process amplifies any noise in the data. Consider using the central difference formula with $h = 0.1$ to find the derivative of $\sin x$ with little added noise, using a MATLAB m-file:

```
%   diff1.m
%plots the differential coefficient of noisy data.

h=0.1;   %set h
x = [0:h:5]'; %data range.
n=length(x);
x2 = x(2:n-1);     %trim x

fn = 0.8*x + sin(0.3*pi*x);      %function to differentiate.

dfn = 0.8 + 0.3*pi*cos(0.3*pi*x); %exact differential
dfn = dfn(2:n-1);%trim to n-2 points

fn2 = fn + 0.1*(randn(n,1)-0.5);   %add a little noise

df1=zeros((n-2),1);  %initialise non-noisy diff.
df2=zeros((n-2),1);  %initialise noisy diff.

for i=2:n-1
   df1(i-1)=(fn(i+1)-fn(i-1))/(2*h);  %non-noisy differential
   df2(i-1)=(fn2(i+1)-fn2(i-1))/(2*h);     %noisy differential
end

plot(x,fn,x,fn2,x2,dfn,x2,df1,x2,df2)   %plot functions and derivatives
```



Note that even though the noise on the functions is small, the noise on the derivative is horrendous.



Solution: fit a polynomial to the to the function, and differentiate that. fitting a 4th order poly to the noisy data gets a smooth differential coefficient, even if it is off at the ends.

Look at finite differences again in Lecture 7 and 8.

Estimating Integrals

Numerical Integration

You've done this in Prelims, so sit back and revise (or look at Kreyszig):

The object is to calculate integrals approximately, either from data or from functions for which simple analytic forms of the integrals don't exist.

Engineering examples: Calculation of the weight or volume of a body, fuel used from flow rate-time measurements, etc.

We want to compute $y(x) = \int_0^x f(x)\mathrm{d}x$ where *f*(*x*) is known on the regular array of points $x = nh,\ n = 1,2,3,\cdots$.

Generally this is performed by computing a discrete sum of the form,

$$y(x) = \sum_{i=0}^{n} a_i f(x_i)$$

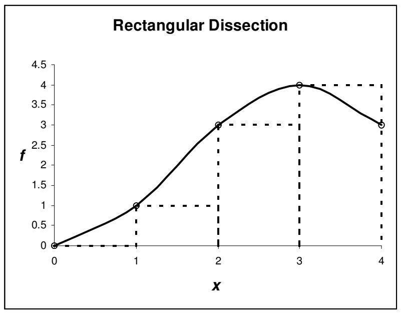
where the a's are weights over n+1 intervals. This process is sometimes called **numerical quadrature**.

Rectangular Dissection is the forward difference formula in disguise:

$$f(nh) = y'(nh) = \frac{y((n+1)h) - y(nh)}{h} + O(h)$$

Expanding the first term as a Taylor series, and remembering that $f(x) = y'(x)$ gives the area between two adjacent points

$$y((n+1)h) - y(nh) = hf(nh) + O(h^2)$$



However, in a given length *L,* there are *L/h* elements, a number which increases as *h* decreases.

Thus the **local, one step error of *O*(*h*^{2}) becomes a *Global error* *O*(*h*)** , which is not very good.

The Trapezium rule is better. The formula for one step is

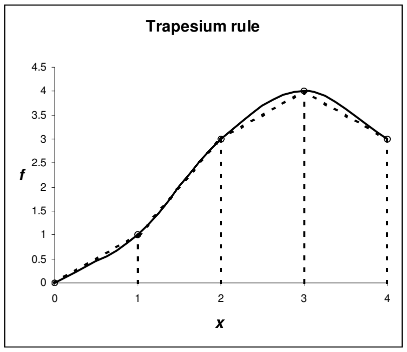
$$y((n+1)h)-y(nh)=\frac{h}{2}(f(nh)+f(n+1))+\varepsilon$$

where ε is the error at each step.

By expanding in Taylor series,

$$y_{n+1}=y_n+hy'_n+\frac{h^2}{2}y''_n+O(h^3)$$ and

$$f_{n+1}=y'_{n+1}=y'_n+hy''_n+\frac{h^2}{2}y'''_n+O(h^3)$$



You can show that the **local error** $\varepsilon=O(h^3)$**, and hence the global error is *O*(*h*^{2}),** which is much better.

Simpson's rule

Fit quadratics (i.e. parabolas) to the middle and end points of an interval $nh\le x\le(n+2)h$.
The formula is derived in Kreyszig (p. 960, 7th Ed.) :

$$y(2nh)-y(0)=\frac{h}{3}\{f(0)+4f(h)+2f(2h)+4f(3h)+2f(4h)+\cdots\cdots+f(2nh)\}$$

Note the alternating 2*f* and 4*f* terms.

Simpson's rule is **globally accurate to *O*(*h*^{4})**, and is so good that it is not usually necessary to go to more accurate methods.

Kreyszig list an *Algorithm* for integrating by Simpson's rule which could be adapted to MATLAB.

MATLAB has a trapezoidal rule integrator trapz and a Simpson's rule integrator quad, (short for quadrature)!