Topic 21: Numerical Differentiation and Integration

Numerical Differentiation

- The aim of this topic is to alert you to the issues involved in numerical differentiation and later in integration.

Differentiation

- The definition of the derivative of a function $f(x)$ is the limit as $h \to 0$ of
  \[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]
- This equation directly suggests how you would evaluate the derivative of a function numerically. Pick a small value of $h$ and evaluate your function.

  *** But... take care or you will get inaccurate results! ***

- There are two sources of error,
  - the discrete approximation introduced by the equation above for non-zero values of $h$, and
  - numerical roundoff error.

The discrete approximation error

- Using the Taylor series expansion we have
  \[ f(x + h) = f(x) + hf'(x) + \frac{1}{2} h^2 f''(x) + \frac{1}{6} h^3 f'''(x) + \ldots \]
- This implies
  \[ \frac{f(x + h) - f(x)}{h} = f' + \frac{1}{2} hf'' + \ldots \]
  error from non-zero $h$

- The error is a function of the magnitudes of 2nd and higher order derivatives
Numerical roundoff error

- There is likely to be roundoff error in representing h.
- For example: assume you are at x = 1000.0 and you have chosen h = 0.000001
- Neither x = 1000.0 nor x + h = 1000.000001 may be a number that has an exact representation in binary
- The effective error in h will be the error in evaluating the difference between 1000.0 and 1000.000001. This will be of the order

Roundoff error

- Under single precision EPS is $\sim 10^{-7}$, under double precision EPS is $\sim 10^{-16}$
- Thus the error in evaluating the difference between 1000.0 and 1000.000001 will be of the order $10^{-7}$ for single precision and $10^{-16}$ for double precision.
- The fractional error in h will be $\frac{10^{-4}}{10} = 10^{-5}$ for single precision !!!
- under double precision it will be $\frac{10^{-16}}{10} = 10^{-17}$, ok.
- Thus the errors you get are a function of the magnitude of x (where on the function you are evaluating the derivative), and of the magnitude of h.

Minimising roundoff error

- Choose h so that x+h and x differ by a value that is exactly representable by the computer.
- We can use the following steps:
  
  ```matlab
  temp = x + h;  % An inaccurate calculation of x + h
  h = temp - x;  % Adjust h to match the inaccurate calculation of x + h
  % so that it is now exact
  ```

  (this has to be done for every x that you evaluate the derivative at)
- Under most conditions, and for simple functions where the 2nd derivative is not too high, the error you will get in evaluating the derivative will be very roughly
  
  $$\frac{\text{EPS} \times |f(x)/h|}{2}$$

Making the derivative calculation symmetric

- We can make the expression for the derivative symmetric about the point of interest x
  
  $$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x - h)}{2h}$$

- This is what one might use to calculate a crude derivative of an array of values.
- In this case, in the absence of an x-scale, h = 1

  a = 2 4 5 3 3 4
  da = 3/2 -1/2 -2/2 -1/2
Integration

- Differentiation involves taking differences between function values, integration involves addition. Numerical integration is also known as quadrature.
- We have a function \( f(x) \) that is discretely sampled at locations \( x_0, x_1, x_2, \ldots \) to produce values \( f_0, f_1, f_2, \ldots \). In the simplest case the samples are a constant step size of \( h \) apart.

![Diagram of trapezoid rule](image)

The trapezoid rule

- The building blocks for integration algorithms are simple formulas that integrate a function over a small number of intervals.
- The simplest algorithm is the trapezoid rule, which integrates a function over a single interval. It is exact for functions that are polynomials up to degree one (i.e., straight lines).

\[
\int_{x_i}^{x_{i+1}} f(x) \, dx = h \left( \frac{1}{2} f_i + \frac{1}{2} f_{i+1} \right)
\]

Extend the trapezoid rule

- To integrate a function you repeatedly apply the trapezoid rule to successive intervals of your function, from \( x_1 \) to \( x_2 \), \( x_2 \) to \( x_3 \), \( x_3 \) to \( x_4 \), etc.
- If you apply the trapezoid rule \( N-1 \) times over \( N \) points you can combine the results to produce the formula

\[
\int_{x_1}^{x_N} f(x) \, dx = h \left( \frac{1}{2} f_1 + f_2 + f_3 + \ldots + f_{N-1} + \frac{1}{2} f_N \right)
\]

(Obviously very easy to implement in MATLAB)

- The trapezoid rule is a 2-point formula that is exact for polynomials up to degree 1. Formulas involving more points provide solutions that are exact for sampled polynomials of higher orders.
Simpson's rule: - a 3 point formula

\[ \int_a^b f(x)\,dx = h \left( \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right) \]

- This rule has the nice property that not only is it exact for sampled functions up to degree 2, it is, in fact, exact for sampled functions up to degree 3 due to a cancellation of error terms resulting from the left-right symmetry of the formula.
- (There are 4, 5, and greater point formulas but they are generally not worth the bother)
- If you apply Simpson's rule to successive non-overlapping pairs of intervals you get the formula

\[ \int_a^b f(x)\,dx = h \left( \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{2}{3} f_3 + \frac{4}{3} f_4 + \ldots + \frac{4}{3} f_{n-1} + \frac{1}{3} f_n \right) \]

How many intervals?

- How many intervals should one use? Of course one never knows the answer!
- The general strategy is to start with the integral evaluated using a coarse number of subdivisions, and then repeat the process with finer and finer subdivisions. The issue is when to stop.
- Assume the integral evaluated at some subdivision level will be \( Q \), and the integral evaluated at the previous subdivision will be \( Q_{\text{last}} \), if the difference between \( Q \) and \( Q_{\text{last}} \) is less than \( \text{EPS} \times Q \) then we are at the limits of machine precision and we know there is no point in going any further.

Efficient integration algorithms re-use the calculations made at course subdivision levels to generate the successive refined values.
- MATLAB has a function 'quad' that performs numerical integration.

\[ Q = \text{quad}('f', A, B) \]

approximates the integral of \( f(X) \) from \( A \) to \( B \) to within a relative error of \( 1 \times 10^{-3} \) using an adaptive recursive Simpson's rule. 'F' is a string containing the name of the function.

\[
>> Q = \text{quad('}\sin\text{',}0,2\text{*pi}) \\
\% \text{Integrate sine from } 0 \text{ to } 2\text{pi}
\]

\[
Q = 0
\]

Monte Carlo Integration

- There can be times when numerical integration can be problematic, especially if one is evaluating double or triple integrations, or the function one is working with is very expensive to evaluate.
- The solution may be to use a Monte Carlo algorithm.
- Monte Carlo algorithms always give an answer, but the answer is not always correct. The probability of correctness increases as the time the algorithm runs increases.
- Simple example - Estimate the value of \( \pi \) using a random number generator.

Diagram:
Estimating Pi

• How do we estimate this?
• Generate random points in the square

No of points within radius r of the centre
------------------------------------------
Total number of points

is our estimate of \( \frac{\text{area of circle}}{\text{area of square}} \)

\[ \pi = \frac{4 \text{area of circle}}{\text{area of square}} \]

The volume of the intersection of a torus with a box

• Three simultaneous conditions

\[ z^2 + (\sqrt{x^2 + y^2} - 3)^2 \leq 1 \]

- the torus, centred on the origin with major radius of 4 and minor radius of 2

\[ \begin{cases} x \geq 1 & \} - the two sides of the box that intersect the torus \\ y \geq -3 & \} \end{cases} \]

Advantages of using Monte Carlo

• This is an example of how a relatively simply defined problem can be very difficult to express in an analytical form. The integral that has to be evaluated is very awkward.
• But the implementation of a Monte Carlo algorithm is very simple. Generate points in 3-space and count the number of points that satisfy the three constraints above.
• The MATLAB function 'rand' generates uniformly distributed random numbers. To generate random 3D points just extract groups of 3 numbers from 'rand'.
• This kind of algorithm is an example of a solution technique that you cannot conceive of doing working with pencil and paper - They are solutions that arise from having a computing machine.

Reference


(This is a really great book)