Numerical Algorithms

Lecture 1 – Introduction
Function Fitting and Approximation

Recommended books:

Working text:

G. Wheatley ‘Applied Numerical Analysis – 7th Editions’, Pearson International Edition.

Additional text:

R.L Burden J.D. Faires ‘Numerical Analysis – 8th Edition’, Thomson International Student Edition.

Reference text:

F B Hildebrand ‘Introduction to Numerical Analysis – 2nd Edition’, Dover reprint.

Syllabus

This course syllabus is:

- Sources of error: round-off, order of algorithm.
- Function fitting and approximation: polynomial & trigonometric bases; Gram-Schmidt orthogonalisation.
- Convergence: Richardson’s method.
- Numerical quadrature: trapezium & Simpson’s rule, sampling methods.
- Numerical integration of ODEs: predictor-corrector methods, modified Euler, Runge-Kutta, adaptive step-size control, stiff systems.

- Discretization in both space and time.
- The solution of second order linear partial differential equations by difference methods.

Lecture 1 will cover function fitting and a general introduction to some of the problems dealt with when developing numerical algorithms.

Lecture 2 will cover numerical differentiation and integration

Lecture 3 will cover the numerical solution of ordinary differential equations

Lecture 4 will cover the finite difference method applied to the solution of partial differential equations.

1. Introduction (Gerald Chap 0, Burden and Faires Chap 1, Hildebrand Chap 1)

This course is a short introduction into the problems of fitting functions and solving certain types of differential equations. There are other problems that require careful numerical analysis, such as solving sets of linear and non-linear equations (see for example Burden and Faires chapters 2, 6 and 7); we will not be covering these issues in this 4-lecture course.

Solving a problem using a computer involves a number of concerns regarding the reliability and accuracy of the result. Of particular note are two issues:

- The effect of using a real computer, in particular the effect of finite precision.
- The behaviour of an algorithm independent of the precision of the computer being used, in particular how quickly it converges to the correct result (if at all) and the accuracy of the result given the nature of the computation being used.

There are two basic measures for an error at a single data value, these are:

Absolute error

If $\hat{x}$ is an approximation to a correct data value x, then $|\hat{x} - x|$ is the absolute error.

Relative error

If $\hat{x}$ is an approximation to a correct data value x, then $\frac{|\hat{x}-x|}{|x|}$ is the relative error provided that $x \neq 0$.

1.1. Computational error

Computers use floating point numbers to represent numbers such as 7.356. These numbers are represented to a finite precision determined by the number of bits within the mantissa of the representation being used (see for example the IEEE floating point standard http://en.wikipedia.org/wiki/IEEE_754-2008). In particular, Matlab uses the variable 'eps' to represent the smallest number that can be added to 1.0 so that the result is larger than 1.0 (eps = 2.220446049250313e-016). Here two example pieces of Matlab code

```
j=1.0;for i = 1:100000, j=j+eps;end;j-1.0

ans =

   2.220446049250313e-011
```

```
j=10.0;for i = 1:100000, j=j+eps;end;j-10.0

ans =

    0
```

In the code on the left, the variable 'j' is initialized as 1.0 and 'eps' is added 100000 times. The answer is bigger than 1.0. The code on the right is identical, but now 'j' is initialized as 10.0, the answer is 'exactly' 10.0. This error arises from the finite precision used in the computation.

1.2. Truncation errors and order of a computation

We start with Taylor's theorem.

If a function f is continuous and the $(n+1)^{th}$ derivative exists on an interval surrounding a point x_0, then the function $f(x_0+\delta x)$ is given by the expansion

$$f(x_0+\delta x) = f(x_0) + f'(x_0)\delta x + \frac{f''(x_0)}{2!}\delta x^2 + \cdots + \frac{f^{n+1}(\xi)}{n+1!}\delta x^{n+1}$$

The number ξ lies between x_0 and $x_0+\delta x$. Note that this is not an infinite series and there is no approximation involved! The last term is called

the remainder term. If the expansion converges for all n, then the series can be continued indefinitely to become an infinite series and we end up with a Taylor's series.

Note an alternative explicit form for the remainder (not involving the unknown ξ, but is in fact used to derive the above expression of the remainder term) is

$$\frac{1}{n!}\int_{x_0}^{x_0+\delta x}(x_0+\delta x-s)^n f^{n+1}(s)ds$$

Truncation error is the error determined purely by the algorithm and not by the precision of the computer used – it is the 'infinite precision' error of the algorithm. Approximating a function by the initial terms of its Taylor series always involves a truncation error. The truncation error of an algorithm is often estimated by performing a Taylor expansion of some expression and determining a bound on the remainder term. [For those who are interested, if the infinite Taylor series does not converge, a highly accurate series often exists near a particular value of x in the form of an 'asymptotic expansion'. See http://en.wikipedia.org/wiki/Asymptotic_expansion and references therein.]

1.2.1. Big 'O' notation

We do not know the value of the remainder, but we can place some bound on it, which is the largest value of the n+1th derivative within the interval x_0 and $x_0+\delta x$. If $\left|\frac{f^{n+1}(\xi)}{n+1!}\right| < K$ then the error between the function and the first n terms of the Taylor series is said to be of order $O(\delta x^n)$, this order determines the rate of convergence of the series

approximation as the order of the expansion is increased, or the rate at which the approximation increases in accuracy as δx is reduced.

When looking at various integration schemes, the 'global error' is often expressed in the form $O(h^n)$ where h is the step size in the algorithm. Such an expression determines the rate of increase of the accuracy of the algorithm as the step size, h, is decreased.

As an aside, big O notation is also used to describe *rates* of convergence of a sequence, which for a Taylor series is the identical expression. Let a series $\{a_n\}$ converges to a number A, and another sequence $\{b_n\}$ converge to zero. If a number K exists such that

$$|a_n - A| \leq K|b_n|$$

Then the series $\{a_n\}$ is said to converge to A with rate of convergence $O\{b_n\}$. A typical example is $b_n = 1/n^2$. (See Burden and Faires section 1.3, or http://en.wikipedia.org/wiki/Big_O_notation)

1.3. Other measures of size - Norms

The concept of absolute value or magnitude of a number can be extended to other objects, such as vectors, matrices and functions. In these cases this magnitude operator is called a 'norm'. Examples that you have seen already in the course are:

- The infinity norm of a vector = $\max[|x_i|]$
- The 2-norm of a vector (or Euclidian norm, or 'length') = $\sqrt{\sum x_i^2}$.

These concepts can be extended to functions

- The L_2 norm = $\sqrt{\left\{\int_a^b \left(f(x)\right)^2 w(x)dx\right\}}$, *w(x)* is a positive 'weighting function'
- The L_∞ norm = $\underset{a \leq x \leq b}{max} |f(x)|$

The above norms are used when discussing the fitting of curves to a set of points and when discussing optimization problems.

2. Polynomials and expansions (Gerald Chapter 3, Burden and Faires Chapter 3)

The Taylor series can be thought of a way of generating a polynomial that 'fits' a function. Much of numerical analysis is involved in identifying clever ways of generating polynomials that enable us to carry out numerical procedures, such as integration and differentiation, with great accuracy.

2.1. The Weierstrass approximation theorem and the uniqueness of a polynomial fit (our motivation)

The Weierstrass approximation theorem states that every continuous function on a bounded interval a≤x≤b can be approximated to arbitrary accuracy by a polynomial (we do not make a formal statement of the theorem or provide a proof!). The importance of the theorem is that it demonstrates the existence of a sequence of polynomials that can fit any continuous function to any degree of precision that is required, i.e. it demonstrates the importance of polynomials for numerical algorithms.

Another important theorem is that it is always possible to find an n^{th} degree polynomial that passes exactly through any n+1 points and *that this polynomial is unique*.

2.2. Lagrangian interpolation

Given the above motivation, we next ask how to find the n^{th} order polynomial passing through n+1 points.

The problem:

Given n+1 data points y_i, the sample points being at x_i (not necessarily evenly spaced) find the interpolating polynomial P(x) that passes through all n+1 points.

We know that such a polynomial exists and is unique. Thus if we find ANY solution to the problem above, then it is THE solution. Thus ALL methods that generate a solution are equivalent in so far is the solution is concerned.

The Lagrangian solution to the problem is to identify n+1 polynomials, each of order n, such that each is ZERO on a subset of n of the points and equal to the interpolating value at only ONE point. One such polynomial is generated for each of the n+1 points. The solution is the sum of the polynomials so found.

Consider point x_0. A polynomial that is zero on all other points x_1 to x_n has $(x-x_i)$, i = 1..n as its roots. Thus a component polynomial is of the form

$$P_0(x) = \lambda_0(x - x_1)(x - x_2) \dots (x - x_n)$$

This polynomial is n^{th} order and is zero on all points except point 0. We also wish this polynomial to pass through y_0, thus

$$P_0(x_0) = \lambda_0(x_0 - x_1)(x_0 - x_2) \dots (x_0 - x_n) = y_0$$

Solving for λ_0

$$\lambda_0 = \frac{y_0}{(x_0 - x_1)(x_0 - x_2) \dots (x_0 - x_n)}$$

The unique interpolating polynomial is the sum of all such polynomial components, i.e.

$$P(x) = \sum L_i(x) y_i$$
$$= \frac{(x - x_1)(x - x_2) \dots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \dots (x_0 - x_n)} y_0$$
$$+ \frac{(x - x_0)(x - x_2) \dots (x - x_n)}{(x_1 - x_0)(x_1 - x_2) \dots (x_1 - x_n)} y_1 + \cdots$$

The L_i are the unweighted components of the Lagrangian interpolating polynomial.

2.3. Divided differences (Gerald section 3.2, but I have found no good references for the proof – try Hildebrand Section 2.5.)

Consider the identity

$$f(x) = f(x_0) + (x - x_0) \frac{f(x) - f(x_0)}{(x - x_0)}$$

The quantity $\frac{f(x) - f(x_0)}{(x - x_0)}$ is called the first divided difference with respect to x and x_0 and is written $f[x_0, x]$. We thus have

$$f(x) = f(x_0) + (x - x_0) f[x_0, x]$$

Now add a point x_1 between x_0 and x. We can repeat the above, only now considering the first divided difference as a function of x

$$f[x_0, x] = f[x_0, x_1] + (x - x_1) \frac{f[x_1, x] - f[x_0, x_1]}{(x - x_1)}$$

The expression $\frac{f[x_1, x] - f[x_0, x_1]}{(x - x_1)}$ is called the second divided difference with respect to x, x_1 and x_0. We can keep on repeating the sequence of operations above generating higher divided differences of the form

$$f[x_i, x_{i+1}, \ldots x_{i+k}] = \frac{f[x_{i+1}, \ldots x_{i+k}] - f[x_i, x_{i+1}, \ldots x_{i+k-1}]}{x_{i+k} - x_i}$$

where x_{i+k} is our 'x' in the previous expressions.

The above is a polynomial expansion scheme similar to that used in the development of a Taylor's series, but using the values of a function at separate points rather than the derivatives at a fixed point (it is called Newton's fundamental formula).

If we substitute back the later difference formulae into the earlier formulae, we arrive at the expansion

$$f(x) = f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \cdots (x - x_0)(x - x_1) \ldots (x - x_n)f[x_0, x_1, x_2, \ldots x_n, x]$$

If f(x) is a polynomial (and if we are fitting n+1 points we ASSUME f(x) is a polynomial) then the last term will be of order n+1 and thus must be zero (as the polynomial is of order n). We thus have the unique polynomial passing through n+1 points being

$$f(x) = f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \cdots (x - x_0)(x - x_1) \ldots (x - x_{n-1})f[x_0, x_1, x_2, \ldots x_n]$$

2.4. Newton-Gregory forward interpolation polynomial

If the spacing between the samples is uniform, with $x_{i+1} - x_i = h$, there is a standard notation for the above polynomial with the substitution $(x - x_0)/h = s$ and $f_{i+1} - f_i = \Delta f_i$, $\Delta f_{i+1} - \Delta f_i = \Delta^2 f_i$, etc. and is

$$P(x_s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \cdots$$

The divided differences now become a set of differences called the Newton-Gregory forward interpolation polynomial (it is forwards as it starts from x_0 and works its way through the x_i.)

2.5. Error of Interpolation (Gerald 3.1)

We have assumed that the function we are fitting is a polynomial and have arrived at a recursive way of generating a series representation of using Newton's fundamental formula. Suppose f(x) was not a polynomial. Then the last term that involved 'x' would NOT be zero and is given by

$$E(x) = f(x) - P(x) = (x - x_0)(x - x_1) \dots (x - x_n) f[x_0, x_1, x_2, \dots x_n, x]$$

We now assume that x is not at one of the fitting points (otherwise the error is zero!) but is otherwise unknown. Now replace 'x' everywhere by 't' except in the final difference term; this leaves the 'x' in the difference expression (which is assumed fixed – I have indicated this by placing a 'hat' over the fixed x in the following). Then

$$W(t) = f(t) - P(t) = (t - x_0)(t - x_1) \dots (t - x_n) f[x_0, x_1, x_2, \dots x_n, \hat{x}]$$

The auxiliary function of 't' is zero for all the fitting points AND for $t = \hat{x}$. Thus W(t) has at least n+2 zeros. We assume that W(t) is continuous and differentiable n+1 times.

If W(t) is continuous and zero at n+2 points, then W(t) must reach alternating maxima and minima between each of these zeros (by continuity) and thus the derivative must have n+1 zeros lying in the n+1 intervals defined by the n+2 zeros of W(t). This argument may be

carried out repeatedly until we arrive at the fact that $W^{n+1}(t)$ must have at least one zero inside the region bounded by the smallest interval containing the set of fitting points and the point $\hat{x}$. Let this zero be at t=ξ. Then

$$0 = W^{n+1}(\xi) = f^{n+1}(\xi) - (n+1)!\, f[x_0, x_1, x_2, \ldots x_n, \hat{x}]$$

All the polynomials go to zero for the n+1th derivative. We thus have

$$f[x_0, x_1, x_2, \ldots x_n, \hat{x}] = \frac{f^{n+1}(\xi)}{(n+1)!}$$

The error term can thus be written in the form

$$E(x) = (x - x_1) \ldots (x - x_n) \frac{f^{n+1}(\xi)}{(n+1)!}$$

The value of ξ is unknown but lies somewhere within the region defined by the fitting points and the interpolation point.

2.6. Neville's method

One never actually computes the interpolating polynomial as the coefficients are very badly conditioned – for a large number of points they will be very unreliable when computed using a real computer. One thus always computes an interpolated value at the given point from scratch using a table of values, or tableau. An efficient way to compute the table was developed by Neville.

Suppose we wish to compute the value of a function at the point $\hat{x}$ given the values of the function at 4 other points. Define the *number* (not polynomial!)

$$P_{i,i+1,i+2,\ldots i+m} = \frac{(\hat{x} - x_{i+m}) P_{i,i+1,i+2,\ldots i+m-1} + (x_i - \hat{x}) P_{i+1,i+2,\ldots i+m-1}}{x_i - x_{i+m}}$$

The above recurrence relation is used to fill in a tableau leading to the answer P_{1234} as shown below

x_1: $y_1=P_1$

P_{12}

x_2: $y_2=P_2$ P_{123}

P_{23} P_{1234}

x_3: $y_3=P_3$ P_{234}

P_{34}

x_4: $y_4=P_4$

The tableau is computed one column at a time, each number being generated by its predecessors in the rows immediately above and below it and in the previous column, i.e. P_{12} comes from P_1 and P_2, etc.

2.7. Problems with polynomials

It has already been stated that one never computes the interpolating polynomial, only the interpolated values, the values being computed using a table of divided differences. The reason is the poor conditioning associated with the computation of the coefficients. We return to this later in the lecture when we consider the condition number of a particular matrix.

A second problem is that polynomials tend to oscillate between the interpolation points, particularly if one tries to fit a polynomial of high order to a set of points whose $n+1^{th}$ derivative is large (look at the error term expression). This problem is particularly acute if ones tries to fit a high order polynomial to a set of data with noisy measurements - one can then end up with pretty wild oscillations between the sample points because the algorithm is trying to fit a function with very large derivatives because of the noise.

To demonstrate the problem with high order polynomials two pieces of Matlab code are presented below. The first is a piece of code I wrote to implement Neville's method stored as polyinterp.m

```
% Compute an interpolated point using Neville's method. xValues are the
% sample points and yValues are the function values to interpolate.
% xhat is the point for which interpolation is required.  yhat is the
% interpolated function at xhat.

function [yhat] = polyinterp(xValues,yValues,xhat)

% assign a default value for the return
yhat = 0;

n = length(xValues);

if n ~= length(yValues)
   error('Vectors must be of same length')
end


% Dummy assignment to get correct column or vector orientation
nextValues = yValues;

% Work through the tableau of divided differences P_ijk.... row by row
% (the inner loop) and column by column (outer loop) nextValues contains
% the column from the tableau being worked on and yValues the previous
% column.
% The answer ends up in nextValues(1)
for m = 1:n-1
   for i = 1:n-m
      nextValues(i) = ((xhat-xValues(i+m))*yValues(i)+(xValues(i)-
xhat)*yValues(i+1));
      nextValues(i) = nextValues(i)/(xValues(i)-xValues(i+m));
   end
   % move to the next column
   yValues = nextValues;
end

% return the answer
yhat = yValues(1);
```

The next piece of code runs polyinterp.m to demonstrate the impact of noise on a high order interpolating polynomial. The fitting points are evenly spaced between -10 and 10 and fit a cosine function. The noise is uniformly distributed about 0 with amplitude 0.01 (1% noise).

```
% Demonstration of the effect of noise on high order interpolation

% A function to be interpolated
xreal = [-10:0.1:10];
yreal = cos(xreal/2);
```

```
% Data for the fitting
xSample = [-10:1.0:10];    %21 points means a 20th order polynomial!
ySample = cos(xSample/2);

% Add some zero mean noise to the sample points
ySample = ySample+0.01*(rand(1,length(ySample))-0.5);

% Compute the interpolated data from the sample points
n = length(xreal);
yinterp = yreal;
for i = 1:n
   yinterp(i)=polyinterp(xSample,ySample,xreal(i));
end

% Look at what happened
figure(1)
plot(xreal,yreal,xSample,ySample,'xr',xreal,yinterp,'.')
grid
title('Comparison of real (solid) and interpolated data (dotted), crosses
are the sample points')
zoom on
```

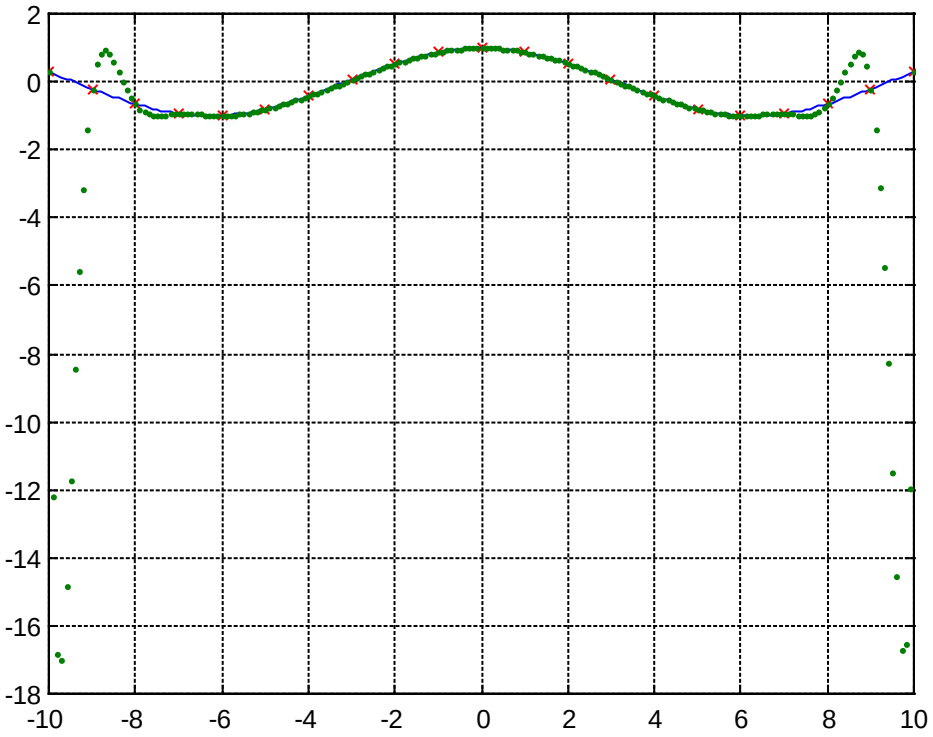


Figure 1 Sample points are crosses, the underlying function is solid, the interpolation is dotted

Figure 1 shows the result of the interpolation when noise is present. Note the wild oscillations near the end of the interpolation. If the noise source is commented out, then the interpolation shown in Figure 2 results. It is not a good idea to interpolate functions with high order polynomials!

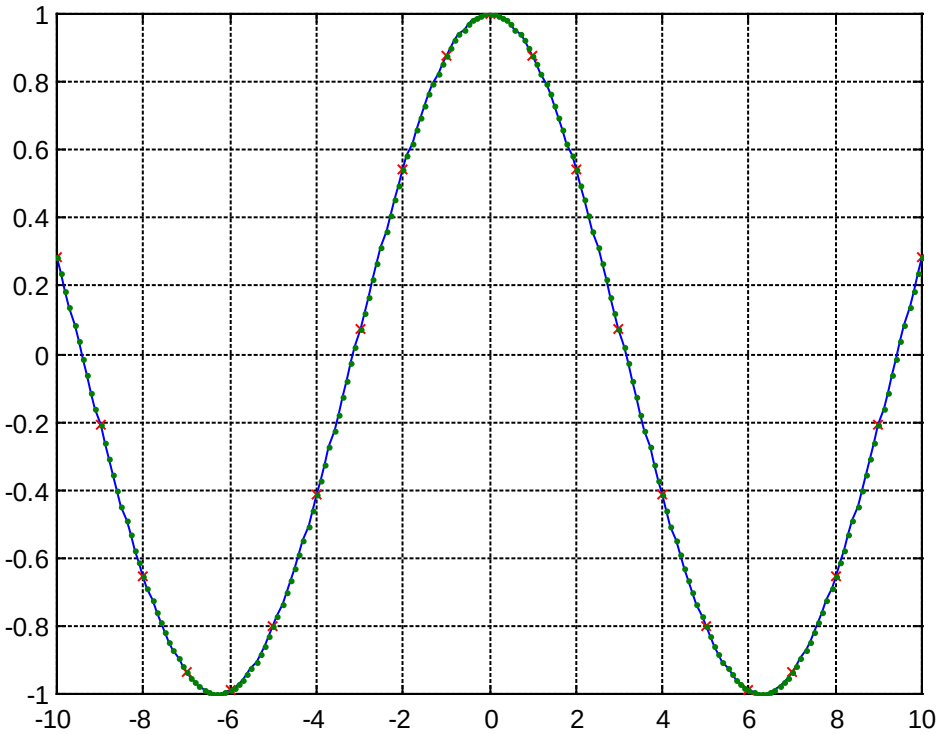


Figure 2 Sample points are crosses, function is solid, interpolation is dotted.

Exercise:

Copy out the function polyinterp.m and try various options for interpolating a function. Try the following:

- Comment out the noise and output the error function (yreal-yinterp) – where is it large?
- The samples are defined from -10:1:10. Change this to -10:2:10 (fewer samples). What is the effect on the error?
- Change the samples to -8:1:8. What is the effect at -10 and 10 on the interpolated function? Where is the interpolating polynomial good, where is it poor?

3. Bases and representations (Gerald Chapter 3, Burden and Faires Chapter 8)

You have covered vectors and matrices in linear algebra. We now consider vectors as 'objects' that live in a 'space' called a 'vector space'. The spaces are labeled by a dimension. For example, the set of vectors with 3 elements come from a 3-dimensional vector space. Every vector that lives in a space can be made up by weighted addition of other vectors that act as building blocks. A set of building blocks that can be used for building any vector in a space is called a basis for the space.

Example

The set of 3-D vectors can be made up from the 'standard basis' called e_i.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} a + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} b + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} c$$

The e_i are vectors with zeros in positions except the i^{th}, which is 1. In fact ANY set of 3 independent vectors can be used as a basis. A set of n independent vectors from an n-dimensional space is said to 'span' the space (we can get into all the nooks and corners of the space).

Imagine increasing the dimension of the space. We can no longer draw the vectors as arrows, so we represent the vectors as a set of points in a graph.

Example
Represent the vector [2,-1,4, 3, 2] (a 5-D vector) on a graph.

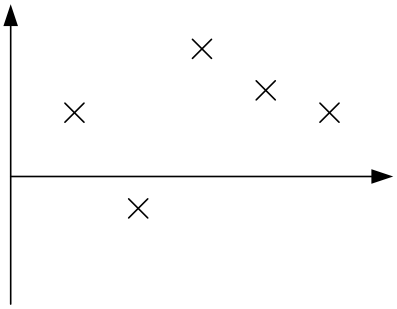


Figure 3 Representing a 5-D vector as a set of points

Imagine increasing the number of points without limit. We end up with functions as vectors having an 'infinite' number of points. We can build up sets of functions from other functions, i.e. a set of functions can act as a basis for an infinite dimensional space. You have seen this before when you covered Fourier series – the basis functions were sines and cosines.

3.1. Monads as a basis – building polynomials

A monad is a polynomial with a single term, such as 'x' or 'x^2'. Polynomials are built up out of monads by multiplying each monad by a number called a coefficient.

Example

$$3x^3 + 1.5x^2 - 0.1x + 2.5 = 3 \times [x^3] + 1.5 \times [x^2] - 0.1 \times [x^1] + 2.5 \times [x^0]$$

The example has built the vector on the left (a polynomial) by adding a set if weighted monads (the basis). Thus the set of monads can be thought of as a basis for all polynomials, and hence, by Weierstrass, as a basis for a sequence of polynomials approximating a function (and for functions suitably constrained to be smooth, as a basis for these functions as well).

3.2. Fitting a polynomial using monads – the Vandermonde matrix

Consider fitting a cubic polynomial to four points of a function using the ideas above. The function values are f_1 to f_4 at points x_1 to x_4. We get 4 equations in four unknowns, the unknowns being the coefficients, i.e.

$$f_1 = a \times [x_1^3] + b \times [x_1^2] + c \times [x_1] + d \times [1]$$
$$f_2 = a \times [x_2^3] + b \times [x_2^2] + c \times [x_2] + d \times [1]$$
$$f_3 = a \times [x_3^3] + b \times [x_3^2] + c \times [x_3] + d \times [1]$$
$$f_4 = a \times [x_4^3] + b \times [x_4^2] + c \times [x_4] + d \times [1]$$

The equations can be expressed as a matrix problem

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ x_3^3 & x_3^2 & x_3 & 1 \\ x_4^3 & x_4^2 & x_4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

The matrix is known as a Vandermonde matrix. The coefficients [a, b, c, d]^T may be obtained by solving this matrix problem. However, as the matrix becomes larger (the fitting polynomial becomes higher order) the condition number of the matrix increases rapidly. The matrix becomes effectively singular for numerical purposes and the fitting polynomial becomes highly unreliable (as well as oscillatory).

3.3. Revisiting polynomial interpolation

We have derived the Lagrangian interpolating polynomial as

$$P(x) = \sum L_i(x) y_i$$

$$L_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

As the Lagrangian polynomials have been constructed to interpolate any polynomial up to order n, then they must also interpolate x^j if $j \leq n$, i.e.

$$\sum L_i(x) x_i^j = x^j$$

In particular

$$\sum L_i(x_i) x_i^j = x_i^j$$

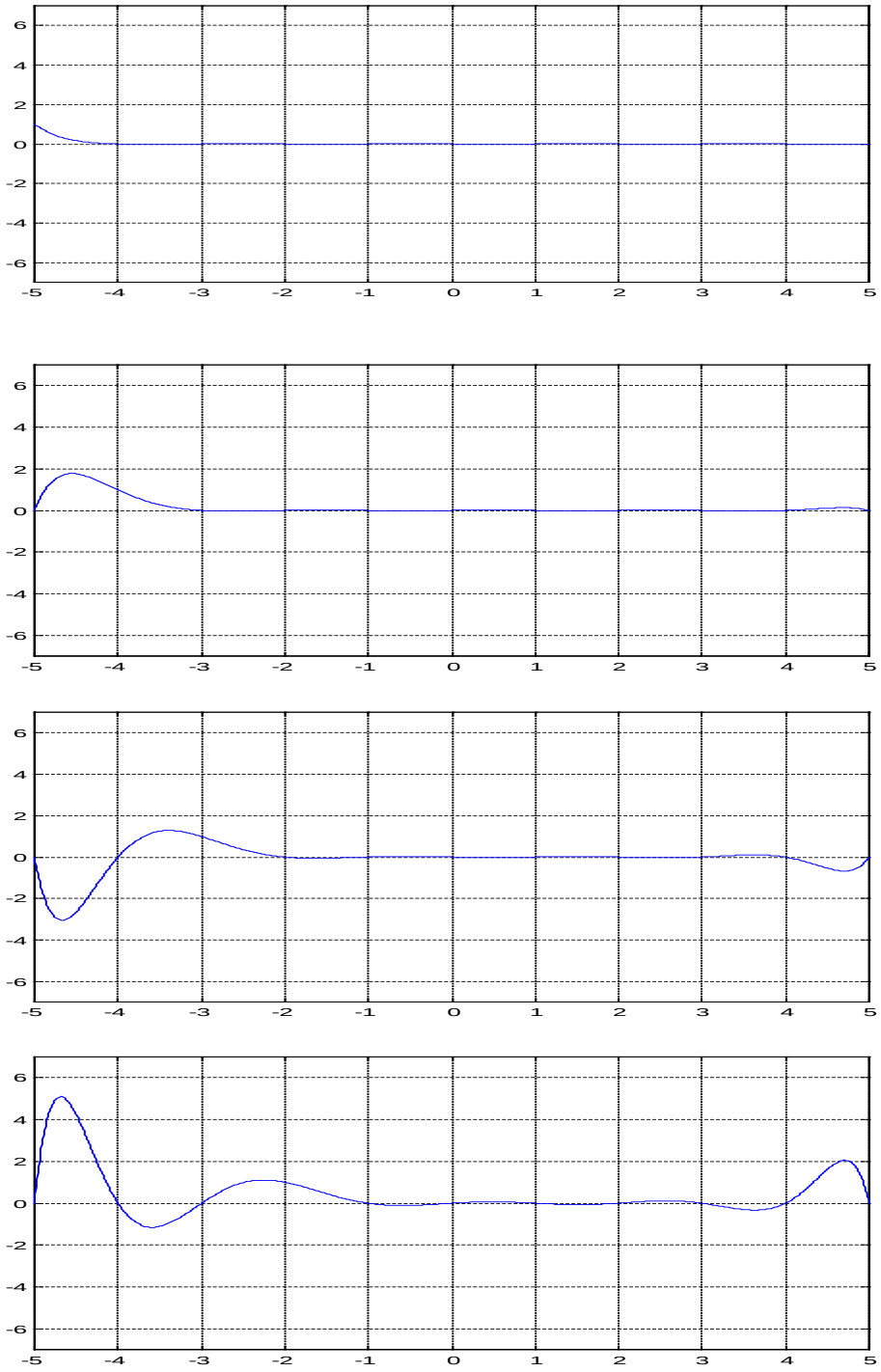
We have thus shown that the set of Lagrangian components from the interpolating polynomial form a basis for the set of nth order polynomials. If we call the matrix of coefficients of the L_i 'L', then the matrix L must be the inverse of the Vandermonde matrix V.

Let us now look at what can happen. The following Matlab code computes the first 6 Lagrangian interpolating polynomials on [-5,5] for

constant sampling of , i.e. for the points [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]. Only the left set of interpolating polynomials (and the centre) are computed as the right set is similar by symmetry. Any 11th order polynomial can be constructed by adding up a weighted sum of the components (and if we use the sampled values of a function we get the interpolating polynomial going through the sample points).

```
% Code to illustrate the behaviour of exact polynomial interpolation using
Lagrange

% Define 11 samples from -5 to 5 at equal steps
xsamp = [-5:5];
L = length(xsamp);

% Define the x axis for plotting and initialise the y
x = [-5:0.01:5];
y = x;
n = length(x);

% Prepare the figure

% s steps through the left half of the set of sample points (plus the
middle one)
for s = 1:L/2+1
   % den is the denominator of the Lagrange component generated by point s
   den = 1;
   for p = 1:L
      if p ~= s
         den = den*(xsamp(s) - xsamp(p));
      end
   end
   % Compute the set of points for this component
   for pt = 1:n
      % num is the numerator of the component generated by point s
      num = 1;
      for p = 1:L
         if p~=s
            num = num*(x(pt)-xsamp(p));
         end
      end
      y(pt) = num/den;
   end
   % Plot out the component assuming that f_s = 1
   figure(s)
   plot(x,y)
   ax = [min(x),max(x),-7,7];
   axis(ax);
   grid
   zoom on
end
```

The output from the code is shown below, the top figure being the component generated by y_{-5} and the bottom by y_0.



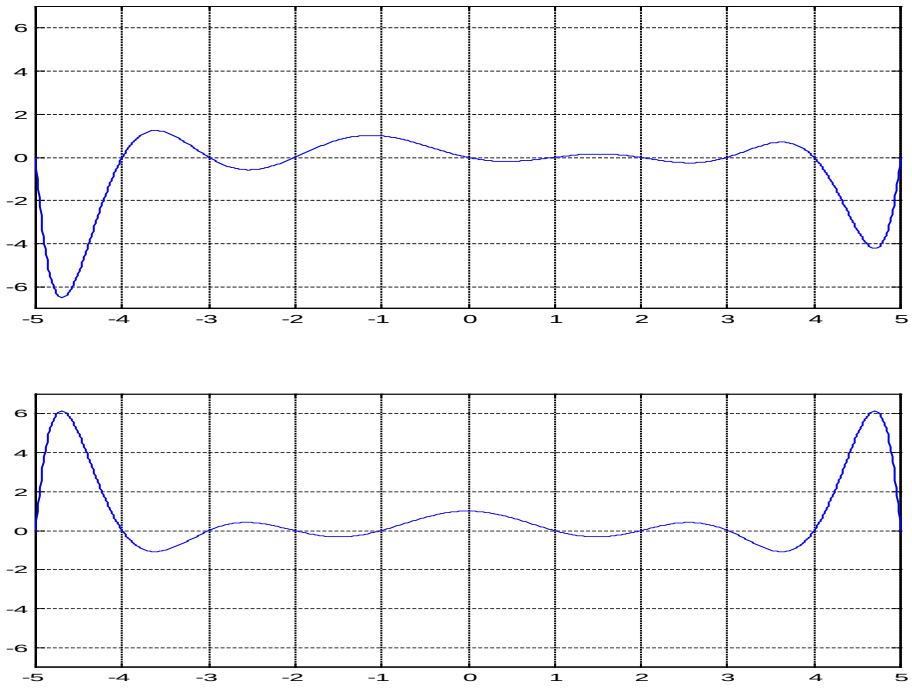


Figure 4 The first 6 Lagrangian interpolating polynomial comonents, top is at -5, last is the centre.

The results demonstrate that small changes to the function values, particularly near the middle of the interpolating region, cause large changes near to the edge of the interpolating region. This effect becomes more and more pronounced as the number of interpolation points increases and corresponds to the poorer and poorer condition number of the Vandermonde matrix.

Exercise

Copy out the Matlab code and change x to [-10:0.1:10]. What happens to the interpolating polynomials for x outside [-5,5]? Will you need to change the axis settings? What does this tell you about interpolating outside the set of sample points?

3.4. Least squares approximation

We have shown that exact fitting of a high order polynomial to a set of points is a very bad idea. There are other fitting functions that are less

badly behaved (such as a rational function approximation like a Padé series – see http://en.wikipedia.org/wiki/Pad%C3%A9_approximant). We will now consider using the L_2 norm described at the beginning of this lecture.

If we are given N+1 points there is a unique polynomial that goes through all the points with zero error. Suppose that we allow an error – we can then find a polynomial with lower order that almost goes through the points. Suppose we wish to fit a cubic polynomial to, say, 10 points. Let the values of the polynomial at the sample points be the vector P and the set of points the vector F. Then the least-squares problem is

Find the coefficients {a,b,c,d} so that the square error = $(F-P)^T(F-P)$ is a minimum.

The least squares error is just the sum of the squares of the errors for the set of points (10 in this case). But we know that the values of the polynomial at the set of sample points is VC, where V is the Vandermonde matrix with 4 columns (going through the coefficients) and 10 rows (going through the samples), C is the vector of coefficients. We thus need to solve:

$$\min\{(F-VC)^T(F-VC) = F^TF - 2C^TV^TF + C^TV^TVC\}$$

This is a function of the four coefficients $\{a,b,c,d\}^T$=C. We differentiate with respect to the elements of C (remembering that F is a constant vector) and set to zero to get

$$-V^TF + V^TVC = 0$$

Solving, we get

$$C = (V^TV)^{-1}V^TF$$

The vector of coefficients is called the least-squares solution.

Suppose the number of points in the vector F becomes very large, i.e. becomes infinite. We then adopt the L_2 norm as our measure of error. The problem then becomes

$$\text{minimize } \left\{\int_a^b (f(x)-p(x))^2 w(x)dx\right\}$$

If we let w(x) = 1 (uniform weighting) then the above becomes

$$\text{minimize } \left\{\int_a^b (f(x)-p(x))^2 dx\right\}$$

Let $p(x) = \sum_{i=0}^{i=n} a_i x^i$ be the nth order polynomial approximation to f, we then need to minimize

$$\int_a^b \left(f(x) - \sum_{i=0}^{i=n} a_i x^i\right)^2 dx = \langle e, e\rangle$$

We have introduced the notation <e,e> to denote the 'inner product' on the left (an extension of the idea of 'dot product' used for finite vectors). Now expand the terms inside the integral

$$\langle e, e\rangle = \int_a^b f(x)^2 - 2\sum_{i=0}^{i=n} a_i f(x) x^i + \sum_{j=0}^{j=n}\sum_{i=0}^{i=n} a_i a_j x^{i+j}\, dx$$

$$\langle e, e\rangle = \int_a^b f(x)^2 dx - 2\sum_{i=0}^{i=n}\left\{a_i \int_a^b f(x) x^i dx\right\} + \sum_{j=0}^{j=n}\sum_{i=0}^{i=n}\left\{a_i a_j \int_a^b x^{i+j} dx\right\}$$

We have taken the integral inside the summation. The above may now be written in matrix-vector form as

$$\langle e,e\rangle = \int_a^b f(x)^2 dx - 2\mathbf{a}^T \begin{bmatrix} \int_a^b f(x)dx \\ \int_a^b f(x)xdx \\ \vdots \end{bmatrix} + \mathbf{a}^T \begin{bmatrix} \int_a^b dx & \int_a^b xdx & \int_a^b x^2dx & \cdots \\ \int_a^b xdx & \int_a^b x^2dx & \int_a^b x^3dx & \cdots \\ \int_a^b x^2dx & \int_a^b x^3dx & \int_a^b x^4dx & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \mathbf{a}$$

The large matrix formed from the integrals of all powers of the form x^{i+j} is called a Hilbert matrix H. The vector **a** is the vector of coefficients. We now differentiate with respect to each coefficient in turn and equate to zero, leading to the matrix equation

$$\mathbf{a} = \text{inv}\left(\begin{bmatrix} \int_a^b dx & \int_a^b xdx & \int_a^b x^2dx & \cdots \\ \int_a^b xdx & \int_a^b x^2dx & \int_a^b x^3dx & \cdots \\ \int_a^b x^2dx & \int_a^b x^3dx & \int_a^b x^4dx & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \right) \begin{bmatrix} \int_a^b f(x)dx \\ \int_a^b f(x)xdx \\ \vdots \end{bmatrix}$$

The Hilbert matrix is again poorly conditioned and so it is not a good idea to fit a polynomial of high order to a function. It is also sometimes tricky to compute the vector of integrals of $f(x)x^i$ that appears on the right hand side.

3.5. Orthogonal bases

Part of the problem of fitting a polynomial to a function is that the Hilbert matrix involves the integral of products of monads x^{i+j}. This produces a

matrix that has a very poor condition number if the polynomial is of high order.

Adopting the 'inner product' notation from above, the Hilbert matrix is composed of terms of the form $\langle x^i, x^j \rangle$ and the vector of constants on the right hand side is made up of terms of the form $\langle x^i, f(x) \rangle$. The monads form a basis for the set of polynomials. The inner product we are using is the same as the dot product between vectors in a finite dimensional space. We are thus projecting the monads onto each other as well as projecting the function f(x) onto each monad. However, suppose we choose another basis for the space.

Choose a set of independent functions $g_i(x)$ so that $\langle g_i(x), g_j(x) \rangle = 0$ if $i \neq j$. then the 'Hilbert' matrix will be of the form

$$\begin{bmatrix} \int_a^b g_0^2(x)dx & \int_a^b g_0(x)g_1(x)dx & \int_a^b g_0(x)g_2(x)dx & \cdots \\ \int_a^b g_0(x)g_1(x)dx & \int_a^b g_1^2(x)dx & \int_a^b g_1(x)g_2(x)dx & \cdots \\ \int_a^b g_0(x)g_2(x)dx & \int_a^b g_1(x)g_2(x)dx & \int_a^b g_2^2(x)dx & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

But we have chosen the functions to be 'orthogonal', thus the 'Hilbert' matrix will be of the form

$$\begin{bmatrix} \int_a^b g_0^2(x)dx & 0 & 0 & \cdots \\ 0 & \int_a^b g_1^2(x)dx & 0 & \cdots \\ 0 & 0 & \int_a^b g_2^2(x)dx & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

This is a diagonal matrix! It is thus easy to invert. It is not only easy to invert, the coefficients of the representation of the function to be fitted are independent of each other – we can increase or reduce the order of expansion at will without re-computing all the other coefficients. This is not the case if the matrix is not diagonal.

You have seen the above property before in Fourier series. The basis functions for Fourier series are sines and cosines. The diagonal terms of the Fourier series are all 1/L, where 2L is the period of the periodic function being fitted – the basis functions are all orthogonal with respect to integration over the period, i.e. such a series can only be easily computed for a periodic function – if it isn't then the Hilbert matrix is non-diagonal and we have a big headache!

There are other orthogonal bases:

Chebyshev polynomials

These are defined as $T_n(x) = \cos(n\cos^{-1}x)$

They can be computed from the recurrence relation

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

The definition gives $T_0=1$, $T_1=x$, thence $T_2 = 2x^2-1$ and so on.

The inner product is defined with a weighting so that

$$\langle T_n(x), T_m(x)\rangle = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_n(x)T_m(x)dx$$

$$\langle T_n(x), T_m(x)\rangle = \begin{cases} \pi \text{ if } m = n = 0 \\ \frac{\pi}{2} \text{ if } m = n \neq 0 \\ 0 \text{ if } m \neq n \end{cases}$$

The Chebyshev polynomials are orthogonal over [-1,1].

Legendre polynomials

These are obtained by the solution of the differential equation

$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}L_n(x)\right]+n(n+1)L_n(x)=0$$

$L_0(x) = 1$, $L_1(x) = x$, then use Bonnet's recursion formula

$$(n+1)L_{n+1}(x)=(2n+1)L_n(x)-nL_{n-1}(x)$$

Legendre polynomials are orthogonal on [-1,1] with a unit weighting function and

$$\langle L_n(x),L_m(x)\rangle=\int_{-1}^{1}L_n(x)L_m(x)dx=\begin{cases}0 \text{ if } n\neq m\\ \frac{2}{2n+1} \text{ if } n=m\end{cases}$$

Given that Legendre polynomials are a basis for polynomials then any polynomial of order less than n can be made up from the sum of Legendre polynomials of order less than n, i.e.

$$P(x)=\sum_{i=0}^{k<n}a_ix^i=\sum_{i=0}^{k<n}\lambda_iL_i(x)$$

We thus have

$$\int_{-1}^{1}P(x)L_n(x)dx=\int_{-1}^{1}\sum_{i=0}^{k<n}\lambda_iL_i(x)\,L_n(x)dx=\sum_{i=0}^{k<n}\lambda_i\int_{-1}^{1}L_i(x)L_n(x)dx=0$$

Thus all polynomials of order less than n are orthogonal to the n^{th} Legendre polynomial.

Question:

Given the above, what are the diagonal terms of the 'Hilbert' matrix for Legendre polynomials?

3.6. Gram-Schmidt orthogonalisation

Given the value of a set of orthogonal functions for fitting a function, we ask if there is a systematic process for generating a set of such functions from a non-orthogonal set.

Suppose we have a set of functions $f_i(x)$ and an inner product that also defines an L_2 norm we wish to use, i.e. given an operator <f(x),g(x)>
Define

$$g_1(x) = \frac{f_1(x)}{\sqrt{\langle f_1(x), f_1(x)\rangle}}$$

The new function has unit norm as

$$\langle g_1(x), g_1(x)\rangle = \frac{\langle f_1(x), f_1(x)\rangle}{\langle f_1(x), f_1(x)\rangle} = 1$$

Now generate $g_2(x)$ as

$$\widehat{g_2}(x) = f_2(x) - \langle g_1(x), f_2(x)\rangle g_1(x)$$

$$g_2(x) = \frac{\widehat{g_2}(x)}{\sqrt{\langle \widehat{g_2}(x), \widehat{g_2}(x)\rangle}}$$

The function $\langle g_1(x), f_2(x)\rangle g_1(x)$ is the projection of f_2 onto g_1. We have thus subtracted the projection of f_2 onto g_1 away from f_2, essentially removing any component making up g_1 from f_2. We then normalize again. Now note that

$$\langle g_1(x), g_2(x)\rangle = \langle g_1(x), \frac{\widehat{g_2}(x)}{\sqrt{\langle \widehat{g_2}(x), \widehat{g_2}(x)\rangle}}\rangle$$

$$= \langle g_1(x), \frac{f_2(x) - \langle g_1(x), f_2(x)\rangle g_1(x)}{\sqrt{\langle \widehat{g_2}(x), \widehat{g_2}(x)\rangle}}\rangle$$

$$= \frac{\langle g_1(x), f_2(x)\rangle - \langle g_1(x), f_2(x)\rangle\langle g_1(x), g_1(x)\rangle}{\sqrt{\langle \widehat{g_2}(x), \widehat{g_2}(x)\rangle}}$$

$$\frac{\langle g_1(x), f_2(x)\rangle - \langle g_1(x), f_2(x)\rangle}{\sqrt{\langle \widehat{g_2}(x), \widehat{g_2}(x)\rangle}} = 0$$

The first two generated functions are thus orthogonal and have unit norm.

The next function is

$$\widehat{g_3}(x) = f_3(x) - \langle g_2(x), f_3(x)\rangle g_2(x) - \langle g_1(x), f_3(x)\rangle g_1(x)$$

$$g_3(x) = \frac{\widehat{g_3}(x)}{\sqrt{\langle \widehat{g_3}(x), \widehat{g_3}(x)\rangle}}$$

Carry on this process, the general term being

$$\widehat{g_n}(x) = f_n(x) - \sum_{i=1}^{n-1} \langle g_i(x), f_n(x)\rangle g_i(x)$$

$$g_n(x) = \frac{\widehat{g_n}(x)}{\sqrt{\langle \widehat{g_n}(x), \widehat{g_n}(x)\rangle}}$$

An alternative approach, which just generates and orthogonal sequence of functions (rather than orthonormal) is to miss out the normalization process. We thus define

$$g_1(x) = f_1(x)$$

$$g_2(x) = f_2(x) - \frac{\langle g_1(x), f_2(x)\rangle}{\langle g_1(x), g_1(x)\rangle} g_1(x)$$

$$g_3(x) = f_3(x) - \frac{\langle g_1(x), f_3(x)\rangle}{\langle g_1(x), g_1(x)\rangle} g_1(x) - \frac{\langle g_2(x), f_3(x)\rangle}{\langle g_2(x), g_2(x)\rangle} g_2(x)$$

We have used the fact that

$$\langle g(x), f(x)\rangle = \sqrt{\langle g(x), g(x)\rangle} \times \langle \hat{g}(x), f(x)\rangle$$

The term $\sqrt{\langle g(x), g(x)\rangle}$ is the norm of g(x), often written as $\|g(x)\|$.

The above is called the Gram-Schmidt orthogonalisation process and ensures that the inner product between any two independent generated functions is zero.

Example

Consider the set of monads {1, x, x^2, x^3,….} and the weighting function $\frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$. Generate a set of orthonormal functions with respect to this weighting function on the interval (-∞,∞).

$$\langle f_1, f_1\rangle = \int_{-\infty}^{\infty} 1 \times 1 \times \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx = 1$$

Thus $g_1(x) = 1$.

$$\langle g_1, f_2\rangle = \int_{-\infty}^{\infty} 1 \times x \times \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx = 0$$

$$\hat{g}_2 = f_2 - \langle g_1, f_2\rangle = x$$

$$\langle \hat{g}_2, \hat{g}_2\rangle = \int_{-\infty}^{\infty} x \times x \times \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx = 1$$

$$g_2 = \frac{\hat{g}_2}{\sqrt{\langle \hat{g}_2, \hat{g}_2\rangle}} = x$$

We can keep on repeating this process to generate the orthonormal polynomials

$$\frac{x^2 - 1}{\sqrt{2}}$$

$$\frac{x^3 - 3x}{\sqrt{6}}$$

$$\frac{x^4 - 6x^2 + 3}{\sqrt{24}}$$

And so on …The polynomials in the numerators are called Hermite polynomials, which satisfy the orthogonality relationship

$\int_{-\infty}^{\infty} H_n(x)H_n(x)e^{\frac{-x^2}{2}}dx = n!\sqrt{2\pi}.$

3.7. The equi-ripple property

The least-squares fitting of a function is based on minimising the L_2 norm. As expected of these things, there is also a solution to the minimum L_∞ norm problem of fitting a polynomial to a function over a bounded interval [a,b].

The minimum L_∞ norm solution means that we wish to minimize the maximum value of the magnitude of the error function between a polynomial and a function for a polynomial of order N between points a and b.

The solution arises from Chebyshev's theorem, which states that an optimum solution exists, is unique and has the property that the maxima and minima of the error function must have equal magnitude and must alternate in sign. Moreover, there are exactly N+2 such stationary points. This is known as the equi-ripple property (the error function must look like a wave whose peaks and troughs have the same deviation from zero). [In fact the statement holds true for a rational approximation to the function as well.]

A nice proof can be found at
http://oleg.moltenstudios.com/math/ChebyshevCriterion.pdf

Finding the equi-ripple solution involves a search (it is an optimization problem). It is sometimes stated that a Chebyshev series solution to the L_2 fitting problem, as described earlier, generates the L_∞ solution because the Chebyshev polynomials are equi-ripple. This is NOT necessarily true. Given a polynomial fit of a given order, we can reduce the order of the polynomial by one by subtracting a weighted Chebyshev polynomial to cancel the highest order term. The resulting increase in error has equal ripple and is thus the polynomial with lower order that best matches the original polynomial. This process is called 'economizing' and the polynomial of order 1 less than the original polynomial is called the economizing polynomial. However, the sum of two weighted Chebyshev polynomials does not necessarily have equi-ripple, i.e. an error made up of the sum of weighted Chebyshev polynomials is not necessarily equi-ripple. So beware!

Numerical Algorithms

Lecture 2

Numerical Integration and Differentiation

1. Differentiation (Gerald Chapter 5.1)

The working definition of a derivative that we shall use is

$$\frac{df}{dx} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x}$$

Lecture 1 described two sources of error in numerical computations, namely truncation error and errors generated by finite precision. The expression above has a truncation error, easily found by expanding the function in a Taylor's series

$$\frac{f(x+\delta x) - f(x)}{\delta x} = \frac{f(x) + \delta x \frac{df}{dx}(x) + 0.5\delta x^2 \frac{d^2 f}{dx^2}(\xi) - f(x)}{\delta x}$$

$$= \frac{df}{dx}(x) + 0.5\delta x \frac{d^2 f}{dx^2}(\xi)$$

The value of ξ lies between x and x+δx and the error is bounded by the maximum of the absolute value of the second derivative. The error is

thus O(δx). However, we have not taken into account the error in the numerical computation itself.

When we compute the function we generally hope to compute f such that

$$f(x)_{\text{compute}} = f(x)_{\text{correct}} \pm eps \times |f(x)|$$

The number '*eps*' is a property of the computer and its value can be found in Matlab by typing '*eps'*. Thus the bound on the actual error we get is

$$|E(\delta x)| \leq \max_{x \leq \xi \leq \delta x} \left| 0.5\delta x \frac{d^2 f}{dx^2}(\xi) \right| + 2 \times eps \times \left| \frac{f(x)}{\delta x} \right|$$

Making δx very small now looks like a bad idea! The error will reduce as δx becomes smaller but then suddenly start to become quite large. Consider the example given in Gerald, f(x) = e^xsin(x). The analytic derivative is e^x(sin(x)+cos(x))

Here is some code to test out our theory

```
% Comparison of numerical derivative with analytic
% the code halves the step size for every iteration.

% The function point
x = 2.0;
% The function and analytic derivative
fx = exp(x)*sin(x);
dfx = exp(x)*(sin(x)+cos(x));
% Assume that the max second derivative occurs at the start (an
approximation)
d2fx = 2*exp(x)*cos(x);

% Initial step size
h = 0.05;
% The errors
e = zeros(1,50);
% The approximate bound
bound = e;
for i = 1:50
   % The derivative
   dfhat = (exp(x+h)*sin(x+h)-fx)/h;
   % The error
   e(i) = dfx - dfhat;
   % Compute the approximate bound
```

```
    bound(i)=abs(0.5*h*d2fx)+2*eps*abs(fx/h);
    % The step is halved
    h = h/2;
end

figure(1)
semilogy([1:50],abs(e),'g',[1:50],bound,'r')
grid
zoom on
```

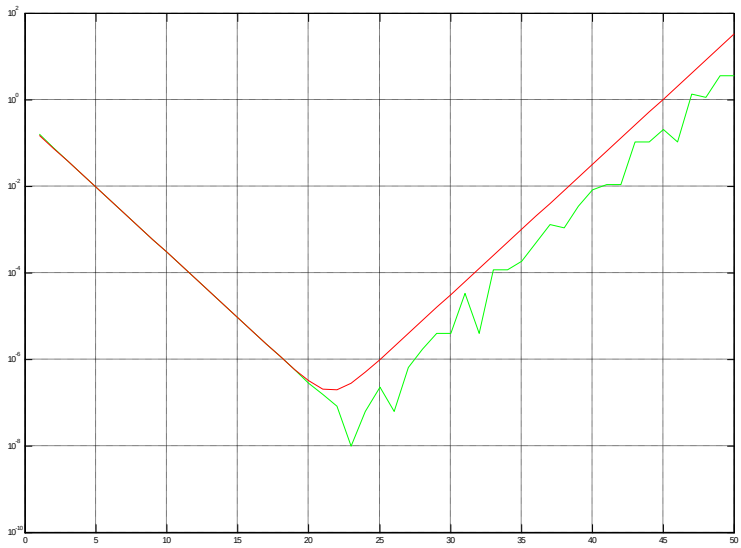


Figure 1 Red is the approximate bound and green is the computed value

The figure above plots out the log of the absolute error (in green) and the approximate bound (in red) as the step size is halved. Note that there is a point where reducing the step size results in a larger error.

The approximation above is called the forward difference approximation for the derivative. There is also the backward difference approximation given by

$$\frac{df}{dx}(x) \cong \frac{f(x) - f(x - \delta x)}{\delta x}$$

This approximation has the same error characteristics as the forward difference approximation.

We have demonstrated that extreme accuracy is not possible with the above approximations because of the finite precision of computers. The above numerical example was carried out with double precision – single precision computations are much worse.

A strategy for overcoming the problem of finite precision swamping computation accuracy before the truncation error is reduced to an acceptable level is to use a higher order approximation. Gerald, Section 5.1, goes in to considerable detail in the use of divided differences and the Newton-Gregory interpolation formula to achieve this aim. The treatment is straightforward, differentiate the divided difference formula with respect to x and then use the resulting table to interpolate. However, the error terms are hard to compute.

Rather than go through the machinery of the above, a more important point to note is the fact that the interpolating polynomial is least accurate towards the ends of the set of interpolation points. Thus if one needs to

differentiate a function, one should use a 'central support'. This means that the table should be of the form [-nh, -(n-1)h, …. 0, h, 2, …nh] where 'nh' should be taken to mean 'x_0+nh' and '0' means 'x_0'. The sample points are evenly space about the point of interest.

The most commonly used approximation to the derivative is the central difference

$$\frac{df}{dx}(x) \cong \frac{f(x+\delta x) - f(x-\delta x)}{2\delta x}$$

This approximation has two function evaluations either side of the point of interest. Expand both evaluations using Taylor series

$$\frac{df}{dx}(x)_{\text{estimate}}$$

$$= \frac{1}{2\delta x}\left\{\left[f(x) + \delta x\frac{df}{dx}(x) + \frac{1}{2}\delta x^2\frac{d^2f}{dx^2}(x) + \frac{1}{6}\delta x^3\frac{d^3f}{dx^3}(\xi_1)\right]\right.$$

$$\left. - \left[f(x) - \delta x\frac{df}{dx}(x) + \frac{1}{2}\delta x^2\frac{d^2f}{dx^2}(x) - \frac{1}{6}\delta x^3\frac{d^3f}{dx^3}(\xi_2)\right]\right\}$$

$$= \frac{df}{dx} + \frac{\delta x^2}{12}\left\{\frac{d^3f}{dx^3}(\xi_1) + \frac{d^3f}{dx^3}(\xi_2)\right\}$$

We have two unknowns ξ, one in the left region, one on the right. By the intermediate value theorem for derivatives (assuming continuity) there

must be a point somewhere between the two points that is equal to the mean, thus

$$\frac{\delta x^2}{12}\left\{\frac{d^3 f}{dx^3}(\xi_1)+\frac{d^3 f}{dx^3}(\xi_2)\right\}=\frac{\delta x^2}{6}\frac{d^3 f}{dx^3}(\xi)$$

The point ξ lies somewhere between x_0-δx and x_0+δx. We thus have an expression that is of O(h^2) (using the standard notation h = δx).

If a second derivative is required, then the central difference approximation is

$$\frac{d^2 f}{dx^2}(x)\cong\frac{f(x+\delta x)-2f(x)+f(x-\delta x)}{\delta x^2}$$

Again of the error is O(h^2). [Note: you need to go to the 4th order Taylor expansion to show this.]

2. Richardson extrapolation

Numerical differentiation always results in a truncation error (from Taylor) involving an unknown ξ, some constant and a power of the step size 'h'. However, if we assume that f(ξ) is a smooth function of 'h' then the truncation error can be considered to be a function of 'h' which may, itself, have a polynomial expansion. This is the idea behind Richardson extrapolation.

The algorithm involves computing an estimate of the derivative using, say central difference, for a range of step sizes and then using divided differences (Neville's algorithm from Lecture 1) to compute a final estimate of the 'step size zero' result.

Here is some code to demonstrate Richardson extrapolation

```
% A demonstration of Richardson extrapolation
```

```
% The function point
x = 2.0;
% The function and analytic derivative
fx = exp(x)*sin(x);
dfx = exp(x)*(sin(x)+cos(x));


% Generate 5 steps, starting at 0.05 to generate a 5-point polynomial in
'h'.
% 6th step is the richardson extrapolation.
samples = zeros(1,6);
estimates = zeros(1,6);
exact = dfx*ones(1,6);

h = 0.05;
for i = 1:5
   % Estimate the derivative using central difference
   dfest = (exp(x+h)*sin(x+h)-exp(x-h)*sin(x-h))/(2*h);
   samples(i)=h;
   estimates(i)=dfest;
   % reduce the step size
   h = h/2;
end

% compute the estimate via Neville's method at a step size of 0
estimates(6) = polyinterp(samples(1:5),estimates(1:5),0)

% Look at the results
figure(1)
semilogy(samples,abs(exact-estimates))
grid
zoom on
```

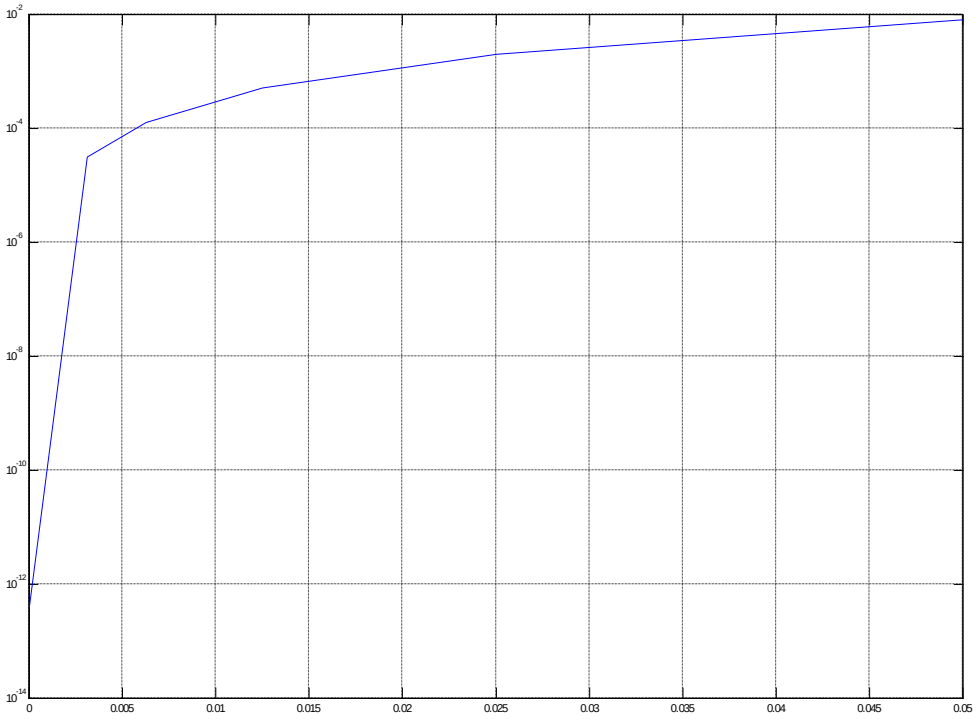


Figure 2 A log plot of the central difference error with step size (x axis), step size zero is extrapolated

The 'real' derivative was reported by Matlab to be 3.64391737678889, the Richardson extrapolated value was 3.64391737678926. Note that the error $O(10^{-13})$ was computed from estimates whose best error was $O(10^{-5})$.

3. Integration

We will be covering the numerical computation of the Riemann integral (see http://en.wikipedia.org/wiki/Riemann_integral) which is essentially defined as

$$\lim_{\delta x \to 0} \sum x_i \delta x$$

The integral is written as

$$\int_a^b f(x)dx = F(b) - F(a)$$

The integral may be thought of as the 'area under the curve' or as an anti-derivative via the relationship

$$\frac{d\int_a^x f(s)ds}{dx} = f(x)$$

The increment of integration is often written as 'h', the length of the integration interval is then 'nh'.

3.1. The Trapezoidal rule

The trapezoidal rule is based on the approximation that the integral is the sum of the areas of a set of strips, each strip being a trapezoid.

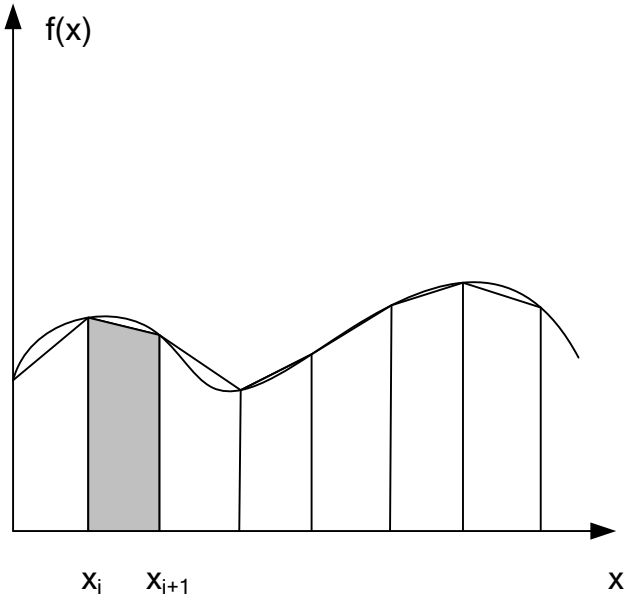


Figure 3 Integrating a function by the Trapezium rule

The area of a the shaded trapezium is

$$A = \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

If we add up the areas of all the trapezia, all heights except the first and last appear twice and we get

$$\int_a^b f(x)dx \cong \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + f(x_n)\right)$$

To obtain an error estimate, expand f(x) as a difference series and integrate over a single 'panel'

$$\int_{x_i}^{x_{i+1}} f(x)dx$$

$$= \int_{x_i}^{x_{i+1}} f(x_i) + \frac{(x - x_i)}{h}\Delta f(x_i) + \frac{(x - x_i)(x - x_{i+1})}{2}\frac{d^2 f}{dx^2}(\xi)dx$$

$$= h\int_0^1 f(x_i) + s\Delta f(x_i) + \frac{h^2}{2}s(s-1)\frac{d^2 f}{dx^2}(\xi)ds$$

The error term was derived in Lecture 1 for the divided difference – we have let s = $(x-x_i)/h$ in the divided difference formula. The first 2 terms give the area of the trapezium and the third term is the error.

Integrating, we get

$$\frac{h^3}{2}\left[\left(\frac{s^3}{3}-\frac{s^2}{2}\right)\frac{d^2f}{dx^2}(\xi)\right]_0^1 = -\frac{1}{12}h^3\frac{d^2f}{dx^2}(\xi)$$

The above is the error for integrating a single panel and is called the 'local error', which is $O(h^3)$.

The 'global error' is the total error for the whole integration. We next note that h = (b-a)/n, thus

$$E = -\frac{n}{12}h^3\frac{d^2f}{dx^2}(\xi) = \frac{a-b}{12}h^2\frac{d^2f}{dx^2}(\xi)$$

Thus the global error is $O(h^2)$.

3.1.1. Romberg integration – using extrapolation

Romberg integration is essentially the same as Richardson extrapolation. We perform integration using, say, the trapezium rule for h , h/2, h/4, etc. and then fit a polynomial to the integral. The extrapolated value can then be extracted by setting h=0, or one can solve for higher order accuracy as explained in Gerald. Note that if one halves the interval at each stage one can use the previous set of function evaluations for every other evaluation at the smaller step-size, thus saving computation.

3.2. Simpson's rule (Gerald 5.3: Note: Burden and Faires 4.3 is more complete)

There are 2 Simpson's rules. The 1/3 rule is for integrating in sections of 3 points (2 panels) and the 3/8 rule for integrating in sections of 4 points (3 panels). If there is an even number of panels, then the integral can be completed by using just the 1/3 rule. If there is an odd number of panels at least one of the integrals must use the 3/8 rule.

The 1/3 rule has a neat trick to show that the integral has a higher order error than expected by utilizing the properties of odd and even functions about the central point of the panel x_1. We start by finding the Taylor series approximation for f(x) about the central point

$$f(x) = f(x_1) + f'(x_1)(x - x_1) + \frac{f''(x_1)}{2}(x - x_1)^2 + \frac{f'''(x_1)}{6}(x - x_1)^3 + \frac{f''''(\xi(x_1))}{24}(x - x_1)^4$$

Note that for each value of 'x' we have an unknown $\xi(x)$.

We now integrate over the two panels about the central point

$$\int_{x_1-h}^{x_1+h} f(x)dx = \left[f(x_1)x + f'(x_1)\frac{(x - x_1)^2}{2} + \frac{f''(x_1)}{6}(x - x_1)^3 + \frac{f'''(x_1)}{24}(x - x_1)^4 \right]_{x_1-h}^{x_1+h} + \int_{x_1-h}^{x_1+h} \frac{f''''(\xi(x_1))}{24}(x - x_1)^4 dx$$

The mean-value theorem is used on the error term to obtain

$$\int_{x_1-h}^{x_1+h} \frac{f''''(\xi(x_1))}{24}(x - x_1)^4 dx = \frac{f''''(\xi_1)}{120}\left[(x - x_1)^5\right]_{x_1-h}^{x_1+h} = \frac{f''''(\xi_1)}{60}h^5$$

We have some unknown point ξ_1 between x_0 and x_2.

Now consider the terms left inside the definite integral, we get

$$2hf(x_1) + \frac{h^3}{3}f''(x_1)$$

But we have a central difference approximation for the second derivative

$$f''(x_1) = \frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2} - \frac{h^2}{12}f''''(\xi_2)$$

We can now gather terms to obtain

$$\int_{x_1-h}^{x_1+h} f(x)dx = \frac{h}{3}\{f(x_0) + 4f(x_1) + f(x_2)\} - \frac{h^5}{12}\left\{\frac{f''''(\xi_2)}{3} - \frac{f''''(\xi_1)}{5}\right\}$$

Thus the 1/3 rule is locally O(h^5). [To obtain the exact expression for the error as given in Gerald needs some more work – see Burden and Faires Section 4.3, exercise 24]. The global error is O(h^4) by the same argument as expressed in the section on the trapezium rule above.

The 3/8 rule is

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}\{f_0 + 3f_1 + 3f_2 + f_3\} - \frac{3}{80}h^5 f''''(\xi)$$

This expression is obtained in exactly the same way as that used to derive the Trapezium rule via the Newton-Gregory formula. Note that the order of error is the SAME as the 1/3 rule. The Newton-Gregory formula applied to the 2-panel problem would predict a local error O(h^4). It is the fact that the 2-panel problem is even in h and thus admits a central expansion that generates the O(h^5) estimate, one order higher than that expected (we cannot get the Newton-Gregory formula up to such a high order as we only have 3 points!)

3.3. Gaussian quadrature

So far we have derived expressions for an integral via a Taylor series and via the Newton-Gregory formula. We can generate another estimate by a third method, as developed by Gauss, in an integration method called Gaussian quadrature.

In the previous methods of integration we used a pre-determined constant increment ‘h’ between the sample points and derived weights for the sampled function points using Newton-Gregory or Taylor series approximations to the function over a panel. Consider a single panel running from -1 to 1. We wish to estimate the integral over the interval

[-1,1], but we are willing to have the points at which the function is computed as variables.

The standard example is

$$\int_{-1}^{1} f(x)dx = af(x_1) + bf(x_2)$$

There are four unknowns, {a, b, x_1, x_2}. We can thus, in principle, choose the four variables so that the integral is exact for all polynomials up to third order.

We solve for x^3, x^2, x and 1

$$\int_{-1}^{1} x^3 dx = 0 = ax_1^3 + bx_2^3$$

$$\int_{-1}^{1} x^2 dx = \frac{2}{3} = ax_1^2 + bx_2^2$$

$$\int_{-1}^{1} x dx = 0 = ax_1 + bx_2$$

$$\int_{-1}^{1} 1 dx = 2 = a + b$$

We end up with four non-linear equations that only have one real solution with different sample points

$$a = b = 1$$

$$x_1 = -\sqrt{\frac{1}{3}}$$

$$x_2 = \sqrt{\frac{1}{3}}$$

Thus our approximation is

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

For 2n terms we have

$$\int_{-1}^{1} f(x)dx \cong \sum \lambda_i f(x_i)$$

which is made to be exact for all polynomials up to (n-1)th order.

The general case.

Suppose we wish the Gaussian integral to be exact for all polynomials up to order 2n-1. Any polynomial up to order n-1 (note: this is NOT 2n-1) may be expressed as the weighted sum of Legendre polynomials (remember Legendre polynomials are an orthogonal basis covered in Lecture 1). Now consider some polynomial P(x) of order greater than or equal to n but less than 2n-1. Then we can express this polynomial as a product and remainder as

$$P(x) = Q(x)L_n(x) + R(x)$$

Q will have order n-1 or less, as will R.

But we showed in Lecture 1 that the Legendre polynomial of order n is orthogonal (over [-1,1] with unit weighting) to all polynomials of order less than n, thus

$$\int_{-1}^{1} Q(x)L_n(x) + R(x)dx = \int_{-1}^{1} R(x)dx$$

The remainder R(x) can itself be expressed as the weighted sum of Legendre polynomials of order less than n. Thus if the quadrature equation is exact on the first n-1 Legendre polynomials, it is exact on all

polynomials up to order 2n-1, PROVIDED that $P(x_i) = R(x_i)$ for each of the sample points. Let the x_i be the roots of $L_n(x)$, then

$$P(x_i) = Q(x_i)L_n(x_i) + R(x_i)$$

But $L_n(x_i) = 0$, thus

$$P(x_i) = R(x_i)$$

Thus the quadrature will be exact if we choose the roots of $L_n(x)$ as the sample points.

To determine the weights in the quadrature expression we take the roots of $L_n(x)$ as the sample points of a Lagrange interpolating polynomial (Lecture 1) and consider all polynomials of order less than n. Then each of these polynomials can be expressed using Lagrange interpolants as

$$P(x) = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)} P(x_i)$$

Thus

$$\int_{-1}^{1} P(x)dx = \sum_{i=1}^{n} \left\{ P(x_i) \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)} dx \right\}$$

is true for any such polynomial, including the Legendre polynomials up to order n-1. We have thus found an exact expression for all

polynomials up to order n-1, and thus, by orthogonality of the polynomial defined by the sample points for all polynomials up to order 2n-1.
The quadrature problem can now be solved:

- Choose the roots of $L_n(x)$ as the sample points x_i.
- The weights are given as $\lambda_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)} dx$
- The integral is $\int_{-1}^{1} f(x)dx \cong \sum \lambda_i f(x_i)$, which is exact for all polynomials up to order 2n-1.

The error estimate can be shown to be

$$E(x) = \frac{f^{2n}(\xi)}{(2n)!} \frac{2}{2n+1}$$

Exactly the same argument can be followed for any set of orthogonal polynomials with associated weighting, i.e. the Gaussian quadrature sample points and weights associated with making the integral $\int_a^b w(x)f(x)dx$ exact for a set of polynomials up to order 2n-1 (given a weighting function w(x)) are the zeros of the n^{th} order orthogonal function associated with the weighting function plus interval, the weights being given by a suitably modified set of expressions involving the Lagrange interpolant over the zeros of the basis function associated weights.

Numerical Algorithms

Lecture 3

Solving Ordinary Differential Equations

1. Background and motivation

Ordinary differential equations are equations that relate the total derivative(s) of dependent variable(s) to function(s) of the dependent variable(s) and an independent variable. Such an equation of the form

$$\frac{dy}{dt} = f(t, y)$$

is called a first order differential equation. If an equation has higher order, such as

$$\frac{d^2y}{dt^2} = -\frac{k}{m}y$$

(which is second order) it is first converted to a set of coupled first order equations by the substitution

$$\frac{dy}{dt} = v$$
$$\frac{dv}{dt} = -\frac{k}{m}y$$

The result is a vector of first order equations with a vector of dependent variables.

Differential equations arise naturally from the analysis of the physics of systems and their solution is central to many engineering tasks.
Ordinary differential equations arise in two general contexts:

- Initial value problems: The initial ‘position’, ‘velocity’ of an object is known, and some functional ‘forces’ act that may vary with time

and depend on position and/or velocity. The objective is to find the 'trajectory' that will be followed by the object.

- Boundary value problems: Something is known about the 'shape' of an object at its edges, such as its position and/or slope. The object's 'curvature' satisfies a differential equation. The aim is to find the resulting shape away from the boundaries.

The second type of problem is much more difficult to solve than the first. We concentrate on the solution of initial value problems in this lecture and touch on the boundary value problem at the end. Boundary value problems come to the fore in the next lecture on Partial Differential Equations.

For information, the initial value problem

$$\frac{dy}{dt} = f(t, y), \quad y(0) = y_0$$

has a unique solution within a convex region containing y_0 and t=0 if

$$\left|\frac{\partial f}{\partial y}(t, y)\right| \leq L$$

for all (t, y) inside the convex region. L is any positive fixed constant.

The above function f can be shown to satisfy a Lipschitz condition in y and L is called the Lipschitz constant. The Lipschitz condition in y for the function f is

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|$$

for all y_1, y_2 inside the convex region. For more details see Burden and Faires Section 5.1.

It is only worth trying to solve an equation if it satisfies the above condition, otherwise you will have big problems as the underlying

equation probably does not have a unique solution. It is likely that the underlying physical model is wrong, or the logic followed to arrive at the differential equation is flawed, or you are trying to solve a problem in a 'non-physical' part of the state space (P1 Modelling).

Example. Solve the differential equation $\frac{dy}{dt} = y^{\frac{2}{3}}$ with the initial condition $y_0 = 0$.

$\frac{\partial f}{\partial y} = \frac{2}{3\times\sqrt[3]{y}}$ is unbounded at y=0, thus $f(t,y) = y^{\frac{2}{3}}$ is not Lipschitz for the given boundary condition. Expect trouble! Using separation of variables

$$\int y^{-\frac{2}{3}} dy = \int dt$$

Thus

$$\left[3 \times \sqrt[3]{y}\right]_{y_0}^{y} = t \Rightarrow y = \left(\frac{t}{3}\right)^3$$

We have *A* solution. But y = 0 is also a solution (in fact this is the one that a numerical solution will give). Thus the solution is NOT unique.

2. Simple Euler integration (Gerald Section 6.2, Burden and Faires Section 5.2)

Simple Euler integration uses rectangular integration to solve the differential equation for a constant step size 'h'. It is based on the Taylor expansion

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2} y''(\xi)$$

The differential equation may be substituted into the Taylor expansion to get a recurrence relation between the 'next' y and the 'current' y

$$y_{k+1} = y_k + hf(t, y_k)$$

We are given y_0 (the initial condition) and we compute the next value of y by adding the area of the rectangle which is 'h' wide and '$f(t,y_k)$' high. Hence the name 'rectangular integration'.

We immediately see that solving differential equations is related to numerical integration – the Simple Euler Method should have local accuracy $O(h^2)$ and global accuracy O(h). But have we shown that the method will work?

3. Stability and convergence (Gerald Section 6.4, Burden and Faires Section 5.9)

The equation

$$y_{k+1} = y_k + hf(t, y_k)$$

is a Difference Equation. The next state depends on the previous state, i.e. there is feedback of the value of y. Such equations have dynamics of their own, for example they can be stable or unstable, this property may, or may not, be shared with the ordinary differential equation being solved. Moreover, at each stage of the iteration we have an error of $O(h^2)$ (the truncation error) – does the effect of these errors on future samples grow or shrink as the iteration proceeds? It would be problematic if a small error at the start of the solution grows to be a massive error at the end of the problem because of the solution method.

Note the difference between numerical integration and the solution of a differential equation. Numerical integration (Lecture 2) solves the

problem $\frac{dy}{dt} = f(t)$, whereas we wish to solve the problem $\frac{dy}{dt} = f(t, y)$ – the 'y' now appears on the right hand side as well as on the left.

The derivation of bounds on the error of solution is quite tricky and will not be covered in detail here (see Burden and Faires Section 5.2 for details). However, the general approach is to compare the correct solution

$$y_{k+1} = y_k + hf(kh, y_k) + \frac{h^2}{2} y''(\xi_k)$$

with the solution generated by the proposed method (Euler in this case)

$$w_{k+1} = w_k + hf(hk, w_k)$$

Subtracting, letting e = y-w, we get

$$e_{k+1} = e_k + h[f(kh, y_k) - f(hk, w_k)] + \frac{h^2}{2} y''(\xi_k)$$

But we have assumed f is Lipschitz in the second variable (for the equation to have a unique solution) thus, for some positive constant L,

$$|f(kh, y_k) - f(hk, w_k)| \leq L|y_k - w_k| = L|e_k|$$

Thus

$$|e_{k+1}| \leq (1 + hL)|e_k| + \frac{h^2}{2}|y''(\xi_k)|$$

It can be seen that the rate of error growth will be determined by the magnitude of 1+hL and the 'size' of the error by a bound on the second derivative.

To conclude, suppose we use Euler's Method to solve the STABLE differential equation

$$\frac{dy}{dt} = -2y, \quad y_0 = 1$$

The analytic solution is $y = e^{-2t}$. Euler's solution is

$$y_{k+1} = y_k - 2hy_k = (1 - 2h)y_k$$

Suppose h = 1.5. The difference equation 'solution' is

$$y_{k+1} = -2y_k$$

The 'solution' is [1, -2, 4, -8, …..] which is clearly unstable, i.e. wrong!

The messages to take away from this brief study are:

- Reducing the step size reduces the rate of growth of the bound on the global error as well as on the bound on the local truncation error.
- Too large a step size can lead to instability of the solution method, even if the underlying differential equation is stable.

4. The Runge-Kutta family (Gerald Section 6.3, Burden and Faires Section 5.4)

We seek alternative solution schemes that have better stability properties and have a slower growth in their error bounds.

Euler's method was derived using the Taylor expansion

$$y(t + h) = y(t) + hy'(t) + \frac{h^2}{2}y''(\xi)$$

We now extend the series and write out the derivatives in full to emphasize that we need TOTAL derivatives, i.e. y is a function of t

$$y(t + h) = y(t) + h\frac{dy}{dt}(t) + \frac{h^2}{2}\frac{d^2y}{dt^2}(t) + \frac{h^3}{6}\frac{d^3y}{dt^3}(\xi)$$

Substituting in the differential equation, we get

$$y(t+h) = y(t) + hf\big(t, y(t)\big) + \frac{h^2}{2}\frac{df\big(t, y(t)\big)}{dt} + \frac{h^3}{6}\frac{d^3y}{dt^3}(\xi)$$

$$y(t+h) = y(t) + h\left[f\big(t, y(t)\big) + \frac{h}{2}\frac{df\big(t, y(t)\big)}{dt}\right] + \frac{h^3}{6}\frac{d^3y}{dt^3}(\xi)$$

Thus, if we can find an expression for $f\big(t, y(t)\big) + \frac{h}{2}\frac{df\big(t,y(t)\big)}{dt}$ we will have a higher order solution. But the TOTAL derivative df/dt is a bit of a problem!

We first revisit Taylor's theorem and extend it to more than one variable before tackling the core problem.

4.1. Taylor's theorem in two variables

If the function f(t,y) is suitably continuous in t and y (possessing partial derivatives of sufficient order) then

$$f(t+\delta t, y+\delta y) = f(t,y) + \delta t\frac{\partial f}{\partial t}(t,y) + \delta y\frac{\partial f}{\partial y}(t,y) + \left[\frac{\delta t^2}{2}\frac{\partial^2 f}{\partial t^2}(\xi,\mu) + \delta t\delta y\frac{\partial^2 f}{\partial t\partial y}(\xi,\mu) + \frac{\delta y^2}{2}\frac{\partial^2 f}{\partial y^2}(\xi,\mu)\right]$$

The two unknowns, ξ and μ, lie somewhere inside the region defined by the steps in t and y. As with the Taylor expansion in one variable, the above can be extended to higher order derivatives.

Now reconsider the problematic derivatives

$$f\big(t,y(t)\big)+\frac{h}{2}\frac{df\big(t,y(t)\big)}{dt}=f\big(t,y(t)\big)+\frac{h}{2}\left[\frac{\partial f}{\partial t}+\frac{\partial f}{\partial y}\frac{dy}{dt}\right]$$
$$=f\big(t,y(t)\big)+\frac{h}{2}\left[\frac{\partial f}{\partial t}+\frac{\partial f}{\partial y}f\big(t,y(t)\big)\right]$$
$$=f\big(t,y(t)\big)+\left\{\frac{h}{2}\right\}^{*}\frac{\partial f}{\partial t}+\left[\frac{h}{2}f\big(t,y(t)\big)\right]^{*}\frac{\partial f}{\partial y}$$

But the above is the beginning of a Taylor expansion in two variables with increments highlighted with a '*', i.e. going back the higher order Taylor expansion

$$y(t+h)\cong y(t)+h\left[f\big(t,y(t)\big)+\frac{h}{2}\frac{df\big(t,y(t)\big)}{dt}\right]$$
$$\cong y(t)+hf\left(t+\frac{h}{2},y+\frac{h}{2}f\big(t,y(t)\big)\right)$$

The above is the called the Mid-Point method.

Higher order methods are derived by extending the Taylor series expansion in the single variable t and trying to match higher order total derivatives using the differential equation and the Taylor series expansion in two variables.

4.2. Modified Euler Method (Gerald Section 6.2, Burden and Faires Section 5.4)

The Modified Euler Method is an important example and is sometimes called a 'predictor–corrector' method.

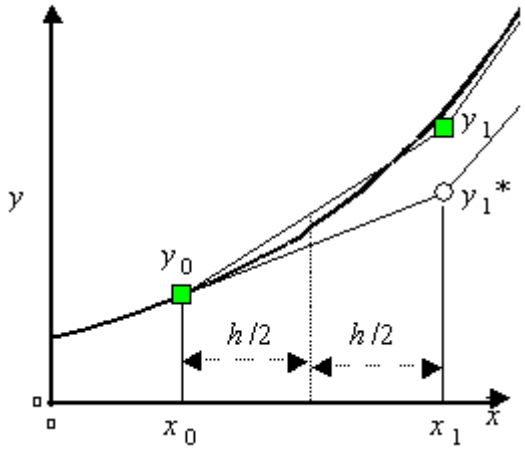


Figure 1 The Modified Euler Method

The method first predicts a value for y using the Euler method

$$y_{k+1}^{*} = y_k + hf(hk, y_k)$$

We then correct the prediction by using the average of the slope at the predicted point and the current slope

$$y_{k+1} = y_k + \frac{h}{2}[f(hk, y_k) + f(hk + h, y_{k+1}^{*})]$$

The method is illustrated in Figure 1 above – the green squares are the corrected values, the open circle is the predicted value. The black line is the 'correct' solution.

To estimate the error we expand the above

$$y_{k+1} = y_k + \frac{h}{2}\left[f(hk, y_k) + f\left(hk + h, y_k + hf(hk, y_k)\right)\right]$$

Then use a Taylor expansion in two variables

$$y_{k+1} = y_k + \frac{h}{2}\left[f(hk, y_k) + \left\{f(hk, y_k) + h\frac{\partial f}{\partial t} + hf(hk, y_k)\frac{\partial f}{\partial y} + O(h^2)\right\}\right]$$

$$= y_k + hf(hk, y_k) + \frac{h^2}{2}\left[\frac{\partial f}{\partial t} + f(hk, y_k)\frac{\partial f}{\partial y}\right] + O(h^3)$$

$$= y_k + hf(hk, y_k) + \frac{h^2}{2}\frac{df}{dt} + O(h^3)$$

$$= y_k + hf(hk, y_k) + \frac{h^2}{2}\frac{d^2y}{dt^2} + O(h^3)$$

We have arrived back at the Taylor expansion for y.

The Modified Euler Method is globally $O(h^2)$ and can be reduced to the same Taylor series expansion used to derive the mid-point method. They are essentially equivalent except for details in their error estimates.

4.3. Fourth order Runge-Kutta

Runge-Kutta methods are based on generating a high order Taylor series expansion by successive evaluations of the function in the differential equation at a carefully chosen set of points. We shall only consider what is called the 'explicit' form of the method. The general explicit method computes

$$y_{n+1} = y_n + h\sum_{i=1}^{q} w_i k_i$$

$$k_i = f\left(nh + \alpha_i h, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j\right)$$

The numbers w_i, α_i, β_{ij} are obtained by equating coefficients of Taylor series expansions and the derivations are very involved. The number 'q' is the number of function evaluations required by the method. One of

the most important methods is fourth order, as this is the highest order for which q is equal to the order of approximation. For example, the 5th order method requires 6 function evaluations (q=6) and the 6th order requires 7 evaluations (q=7). The 4th order method is usually written in the following tabular form

$$k_1 = hf(nh, y_n)$$

$$k_2 = hf\left(nh + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(nh + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = hf(nh + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

5. Multi-step predictor-corrector methods (Gerald Section 6.4, Burden and Faires Section 5.6)

The methods considered so far are called single-step methods.

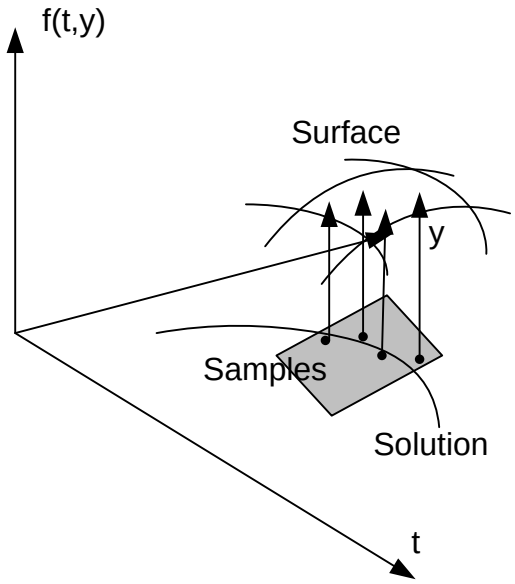


Figure 2 Illustration of the strategy followed for single-step methods

Figure 2 illustrates the strategy that is followed by the single-step methods. We have a differential equation relating the rate of change of y with respect to t. This differential equation has a solution that is a path shown as a solid line in the t-y plane. There is a function of t and y that gives the derivative of y with respect to t. This function defines a surface which itself has a local geometry in 3D (described by a Taylor series in 2 variables). We sample the local geometry by performing a sequence of function calls in a region about the current point (t,y) (shaded in grey). This geometric information is then used to compute the differentials (used in the Taylor series in 1 variable) of the curve by comparing

coefficients of the Taylor series in 1 variable with the surface Taylor series coefficients generated by the sampling.

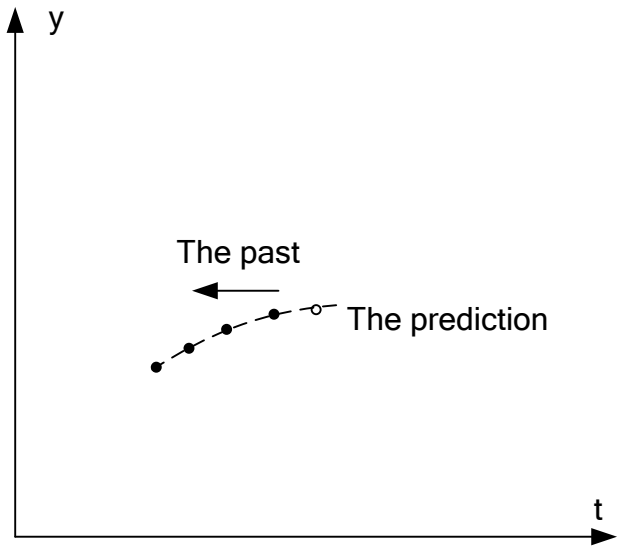


Figure 3 Illustration of a multi-step method

An alternative approach is to use the past solution points along the curve and use the difference equations derived in Lecture 1 to predict the next point in the solution and then correct this prediction given the differential equation. Figure 3 above illustrates this idea. No sampling of the function is performed and all information is contained in past estimates of points on the curve.

5.1. Predicting a solution

The differential equation can be written in the form

$$dy = f(t, y)dt$$

Thus

$$\int_{y_{nh}}^{y_{nh+h}} dy = y_{n+1} - y_n = \int_{nh}^{nh+h} f(t, y)\, dt$$

We can then demand that the right hand side is exact for all polynomials up to third order, say, resulting in a difference equation

$$y_{n+1} = y_n + \frac{h}{24}[55f(nh, y_n) - 59f(nh - h, y_{n-1}) + 37f(nh - 2h, y_{n-2}) - 9f(nh - 3h, y_{n-3})] + O(h^5)$$

5.2. Correcting a solution – the Adams-Mouton Method

The Adams-Moulton method takes the prediction based on the previous four points and then repeats the above, but now using the three previous points and the predicted point to correct the predicted value. The corrector is then

$$y_{n+1} = y_n + \frac{h}{24}[9f(nh + h, y^*_{n+1}) + 19f(nh, y_n) - 5f(nh - h, y_{n-1}) + f(nh - 2h, y_{n-2})] + O(h^5)$$

Note that multi-step methods require a start-up procedure, such as Runge-Kutta so that a 'past' can be computed. Such methods can also suffer from stability problems.

6. Step size control (Burden and Faires Section 5.5)

We have assumed throughout this lecture that the step size 'h' is given and is constant. This requires some knowledge of the solution as well. Requiring that the step size be constant can be highly inefficient if the solution is very smooth for some value of t but is less smooth at other values. It can also lead to stability problems in stiff systems (see below).

Most solvers have some form of adaptive step-size control, a common method being the Runge-Kutta-Fehlberg Method. This method computes both a 4th order and 5th order Runge-Kutta estimate for the

next point and compares the two solutions at each step. If the two methods differ by an amount smaller than some pre-defined limit, then the higher order estimate is accepted and the step size is increased for the next point. If the error too large, then the step size is reduced and the estimates recomputed.

7. Higher order equations

As stated at the beginning of this lecture, we first convert higher order differential equations into a set of coupled first order equations. We thus end up with a set of equations of the form

$$\frac{dy_i}{dt} = y_{i+1}$$

$$\frac{dy_p}{dt} = f\left(t, y_1, y_2 \ldots, y_p\right)$$

where the subscript above indicates the entry in a vector of 'p' unknowns. If Runge-Kutta is to be used for the solution, then we first compute all the k_1s, one for each equation, then the set of k_2s using all the k_1s that have been computed, etc.

7.1. Stiff equations – implicit methods (Gerald Section 6.6)

A higher order equation, say second order, has various components that make up the solution (this is particularly evident in linear differential equations where the solution is the superposition of exponentials). If the rates of change of the various components are very different, then the equation is said to be 'stiff', the problem has more than one characteristic 'scale'. The 'fast' component can drive the solution method unstable.

One way of overcoming the problems of stiff equations is to use implicit methods. An implicit method expresses the Taylor series approximation used to derive the difference equation about the future solution (which is unknown) essentially answering the question ‘What values of x and y in the future would be needed if one was to run the differential equation in reverse and end up at the current values?’. The resulting difference equation then has future values on the right hand side. The result is that

one needs to solve a set of equations before the correct 'next step' can be computed. The problem is only simple if the equations are linear.

Example (Gerald Section 6.6)

Solve

$$\frac{dx}{dt} = f(x,y) = 1195x - 1995y$$
$$\frac{dy}{dt} = g(x,y) = 1197x - 1997y$$

using implicit Euler and step size 0.1.

The difference equations for Taylor series expansions about the future point are

$$x_{n+1} = x_n + hf(x_{n+1}, y_{n+1}) = x_n + 119.5x_{n+1} - 199.5y_{n+1}$$
$$y_{n+1} = y_n + hy(x_{n+1}, y_{n+1}) = y_n + 119.7x_{n+1} - 199.7y_{n+1}$$

Move right hand side to left

$$\begin{bmatrix} 1 - 119.5 & 199.5 \\ -119.7 & 1 + 199.7 \end{bmatrix} \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

Thus

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} -118.5 & 199.5 \\ -119.7 & 200.7 \end{bmatrix}^{-1} \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

The non-implicit Euler solution is

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 120.5 & -199.5 \\ 119.7 & -198.7 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

which is unstable.

8. Boundary value problems – the shooting method. (Gerald Section 6.7)

The initial value problem defines the value of the unknown and all its relevant derivatives at t=0. A second order differential equation may not

have its boundary values defined at just t=0, the dependent variable may just have its value defined at t=0 and t=T.

If the differential equation is linear, then a particularly effective way of solving a boundary value problem is the Shooting Method.

Suppose we are given the value of y_0 and y_T. We choose two arbitrary initial conditions $dy/dt(0)_1$ and $dy/dt(0)_2$. We get two solutions of the differential equation $y_1(t)$ and $y_2(t)$, neither of which equals y_T when t=T. However, the solution of the equation with initial conditions $\lambda_1 dy/dt(0)_1+\lambda_2 dy/dt(0)_2$, $(\lambda_1+\lambda_2)y_0$ is $\lambda_1 y_1(t)+\lambda_2 y_2(t)$ (by linearity), thus we just need to solve

$$(\lambda_1 + \lambda_2)y_0 = y_0$$
$$\lambda_1 y_1(T) + \lambda_2 y_2(T) = y_T$$

If the equations are non-linear, then we have to solve the above iteratively.

Numerical Algorithms

Lecture 4

The finite difference method and partial differential equations

1. Background and motivation

Partial differential equations arise when the solution to be found is a shape rather than a simple path. Applications include:

- Determine the temperature distribution throughout a body.
- Predict the distribution of fluid flow, such as water or wind.
- Solving potential problems, such as electrostatic or electromagnetic problems.

Finite differencing is one way of solving the partial differential equations that arise from the physical models above. Finite differences are also another way to solve boundary value problems that were briefly discussed in the previous lecture. The finite element method is another key method for solving partial differential equations – but this method is taught elsewhere in the course and will not form part of this lecture.

The boundary conditions for partial differential equations are now functions rather than just single values (as was the case in ordinary differential equations) and involve more than one dependent variable. We will thus have to consider how to convert derivatives into differences in more than one variable, i.e. differences in time and space. The fact

that linked dependent variables are involved has implications for the stability of numerical solutions to partial differential equations.

2. Types of equation (Gerald Chapter 8)

The elementary theory of partial differential equations considers equations of the form

$$A\frac{\partial^2 y}{\partial x^2} + B\frac{\partial^2 y}{\partial x \partial t} + C\frac{\partial^2 y}{\partial t^2} + f(t,x,y) = 0$$

A, B and C may be functions. These equations are split into three types (based on conic sections):

- Elliptic equations: B^2-4AC<0
- Parabolic equations: B^2-4AC=0
- Hyperbolic equations:B^2-4AC>0

Laplace's equation $\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$ is a typical elliptic equation and describes how a potential must vary within some boundary, i.e. elliptic equations are usually associated with potential problems. A potential function is usually defined on the boundary and the problem is solved by an iterative process, often called 'Relaxation'. The solutions tend to be smooth functions (but allow discontinuities in the boundary conditions).

The diffusion equation $\frac{\partial^2 T}{\partial x^2} = \frac{c\rho}{k}\frac{\partial T}{\partial t}$ is a typical parabolic equation. These are usually initial value problems where we are given an initial distribution, such as a temperature or concentration, and the objective is to determine how the distribution evolves with time. The solutions become smoother as time progresses. When there is more than one spatial variable, the steady state solution can itself be a solution to an

elliptic equation (the time derivative is zero at infinite time) – hence providing a physical motivation for the name 'Relaxation' in the methods mentioned above ('Relaxation' is a term to describe how a physical system tends to a steady state after a long period of time, often associated with cooling).

The wave equation $c^2 \frac{\partial^2 \phi}{\partial x^2} = \frac{\partial^2 \phi}{\partial t^2}$ is a typical hyperbolic equation describing how a wave propagates by the exchange of energy between complementary phenomena, such as potential and kinetic energy or electric and magnetic field energy (pairs of variable that behave in the way are sometimes called 'Conjugate'). The solution can retain discontinuities as it propagates (unless there is dispersion – see your lecture notes on A1 waves).

Linear equations are those whose solutions to the homogeneous form of the equation (zero on the right hand side) satisfy the superposition principle, i.e. given if y_1 is a solution and y_2 is a solution then ay_1+by_2 is a solution. You have met this property when solving partial differential equations by the separation of variables in A1. If superposition does not hold, then the equation is said to be non-linear and can admit a vast range of behaviour, such as 'Solitons' and other exotica.

All the equations must have boundary values defined for there to be a solution. If the value of the unknown function is given on the boundary, then it is said there is a Dirichlet condition at that point. If the derivative is given along the boundary normal, it is said to be a Neumann condition. Real problems often have mixed types of boundary conditions. Demonstrating the existence of solutions, let alone the

uniqueness of solutions, is non-trivial and far more complex than for ordinary differential equations.

3. Finite differences and boundary value problems

Suppose one wishes to solve Poisson's equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \rho(x, y)$$

The solution is sought within some boundary with Dirichlet conditions.

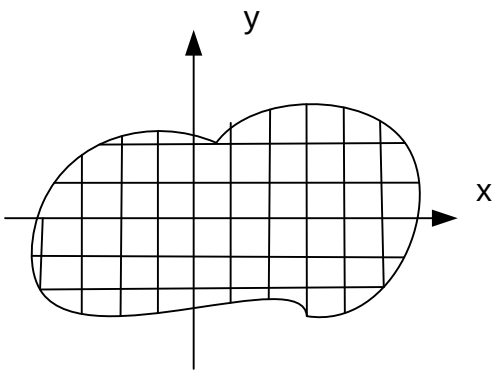


Figure 1 A gridded region in x and y

A possible region of interest is shown in figure 1 above. We impose a set of grid-lines on the region and then approximate the derivatives as

$$\frac{\partial^2 \phi}{\partial x^2} \cong \frac{\phi_{j+1,i} - 2\phi_{j,i} + \phi_{j-1,i}}{\delta x^2}$$

$$\frac{\partial^2 \phi}{\partial y^2} \cong \frac{\phi_{j,i+1} - 2\phi_{j,i} + \phi_{j,i-1}}{\delta y^2}$$

The differential equation then becomes a set of linear equations

$$\frac{\phi_{j+1,i} - 2\phi_{j,i} + \phi_{j-1,i}}{\delta x^2} + \frac{\phi_{j,i+1} - 2\phi_{j,i} + \phi_{j,i-1}}{\delta y^2} = \rho_{i,j}$$

We next write out the set of unknown ϕ as a long vector and convert the above equations into the matrix equation

$$\mathbf{A}\phi = \mathbf{b}$$

The boundary values are in fact known as the problem is Dirichlet. We thus move the known values of ϕ to the right hand side.

A Dirichlet example

Generate the finite difference equations that will solve Laplace's equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

On a regular 5 by 5 unit grid centred on x=0, y=0 with the boundary conditions ϕ=1 if x= -2 or x = 2 and ϕ=0 on y=-2 or y = 2 (and x ≠ 2 and x ≠ -2).

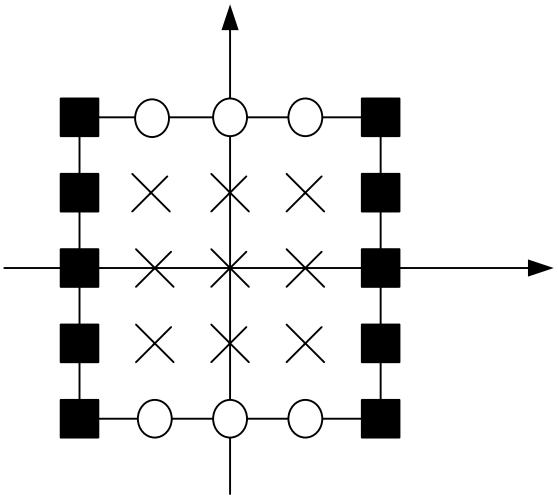


Figure 2 The region to be solved

Figure 2 illustrates the region for solution. The crosses are the points at which the solution is not known, the circles are where the boundary values are zero and the filled squares where the boundary values are 1.

The sampling is 1, thus $\delta x = \delta y = 1$. The difference equations are thus

$$\phi_{j+1,i} - 2\phi_{j,i} + \phi_{j-1,i} + \phi_{j,i+1} - 2\phi_{j,i} + \phi_{j,i-1} = 0$$

(Look at the indices in the equations and sketch on the diagram above which neighbouring sample points affect any given point). The values of i and j for the internal points run from -1 to +1 (as we know the values on

the boundary). We thus end up with 9 equations (for the 9 crosses in the middle of the region, starting at the top left)

$$\{\phi_{0,1} - 2\phi_{-1,1} + \phi_{-2,1}\} + \{\phi_{-1,2} - 2\phi_{-1,1} + \phi_{-1,0}\} = 0 \ (x = -1, y = 1)$$

$$\{\phi_{1,1} - 2\phi_{0,1} + \phi_{-1,1}\} + \{\phi_{0,2} - 2\phi_{0,1} + \phi_{0,0}\} = 0 \ (x = 0, y = 1)$$

$$\{\phi_{2,1} - 2\phi_{1,1} + \phi_{0,1}\} + \{\phi_{1,2} - 2\phi_{1,1} + \phi_{1,0}\} = 0 \ (x = 1, y = 1)$$

$$\{\phi_{0,0} - 2\phi_{-1,0} + \phi_{-2,0}\} + \{\phi_{-1,1} - 2\phi_{-1,0} + \phi_{-1,-1}\} = 0 \ (x = -1, y = 0)$$

$$\{\phi_{1,0} - 2\phi_{0,0} + \phi_{-1,0}\} + \{\phi_{0,1} - 2\phi_{0,0} + \phi_{0,-1}\} = 0 \ (x = 0, y = 0)$$

$$\{\phi_{2,0} - 2\phi_{1,0} + \phi_{0,0}\} + \{\phi_{1,1} - 2\phi_{1,0} + \phi_{1,-1}\} = 0 \ (x = 1, y = 0)$$

$$\{\phi_{0,-1} - 2\phi_{-1,-1} + \phi_{-2,-1}\} + \{\phi_{-1,0} - 2\phi_{-1,-1} + \phi_{-1,-2}\} = 0 \ (x = -1, y = -1)$$

$$\{\phi_{1,-1} - 2\phi_{0,-1} + \phi_{-1,-1}\} + \{\phi_{0,0} - 2\phi_{0,-1} + \phi_{0,-2}\} = 0 \ (x = 0, y = -1)$$

$$\{\phi_{2,-1} - 2\phi_{1-,1} + \phi_{0,-1}\} + \{\phi_{1,0} - 2\phi_{1,-1} + \phi_{1,-2}\} = 0 \ (x = 1, y = -1)$$

We then put in the boundary conditions (all those elements with a ‘2’ in them)

$$\{\phi_{0,1} - 2\phi_{-1,1} + 1\} + \{0 - 2\phi_{-1,1} + \phi_{-1,0}\} = 0$$

$$\{\phi_{1,1} - 2\phi_{0,1} + \phi_{-1,1}\} + \{0 - 2\phi_{0,1} + \phi_{0,0}\} = 0$$

$$\{1 - 2\phi_{1,1} + \phi_{0,1}\} + \{0 - 2\phi_{1,1} + \phi_{1,0}\} = 0$$

$$\{\phi_{0,0} - 2\phi_{-1,0} + 1\} + \{\phi_{-1,1} - 2\phi_{-1,0} + \phi_{-1,-1}\} = 0$$

$$\{\phi_{1,0} - 2\phi_{0,0} + \phi_{-1,0}\} + \{\phi_{0,1} - 2\phi_{0,0} + \phi_{0,-1}\} = 0$$

$$\{1 - 2\phi_{1,0} + \phi_{0,0}\} + \{\phi_{1,1} - 2\phi_{1,0} + \phi_{1,-1}\} = 0$$

$$\{\phi_{0,-1} - 2\phi_{-1,-1} + 1\} + \{\phi_{-1,0} - 2\phi_{-1,-1} + 0\} = 0$$

$$\{\phi_{1,-1} - 2\phi_{0,-1} + \phi_{-1,-1}\} + \{\phi_{0,0} - 2\phi_{0,-1} + 0\} = 0$$

$$\{1 - 2\phi_{1-,1} + \phi_{0,-1}\} + \{\phi_{1,0} - 2\phi_{1,-1} + 0\} = 0$$

Take the constants to the right hand side and gather terms

$$\phi_{0,1} - 4\phi_{-1,1} + \phi_{-1,0} = -1$$
$$\phi_{1,1} - 4\phi_{0,1} + \phi_{-1,1} + \phi_{0,0} = 0$$
$$-4\phi_{1,1} + \phi_{0,1} + \phi_{1,0} = -1$$
$$\phi_{0,0} - 4\phi_{-1,0} + \phi_{-1,1} + \phi_{-1,-1} = -1$$
$$\phi_{1,0} - 4\phi_{0,0} + \phi_{-1,0} + \phi_{0,1} + \phi_{0,-1} = 0$$
$$-4\phi_{1,0} + \phi_{0,0} + \phi_{1,1} + \phi_{1,-1} = -1$$
$$\phi_{0,-1} - 4\phi_{-1,-1} + \phi_{-1,0} = -1$$
$$\phi_{1,-1} - 4\phi_{0,-1} + \phi_{-1,-1} + \phi_{0,0} = 0$$
$$-4\phi_{1-,1} + \phi_{0,-1} + \phi_{1,0} = -1$$

Now write the equations in matrix form (I have used row-major order, i.e. generate the vector of unknowns by taking a row of the sampled region at a time)

$$\begin{bmatrix} -4 & 1 & & 1 & & & & & \\ 1 & -4 & 1 & & 1 & & & & \\ & 1 & -4 & & & 1 & & & \\ 1 & & & -4 & 1 & & 1 & & \\ & 1 & & 1 & -4 & 1 & & 1 & \\ & & 1 & & 1 & -4 & & & 1 \\ & & & 1 & & & -4 & 1 & \\ & & & & 1 & & 1 & -4 & 1 \\ & & & & & 1 & & 1 & -4 \end{bmatrix} \begin{bmatrix} \phi_{-1,1} \\ \phi_{0,1} \\ \phi_{1,1} \\ \phi_{-1,0} \\ \phi_{0,0} \\ \phi_{1,0} \\ \phi_{-1,-1} \\ \phi_{0,-1} \\ \phi_{1,-1} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -1 \\ -1 \\ 0 \\ -1 \\ -1 \\ 0 \\ -1 \end{bmatrix}$$

Note the banded structure of the matrix.

The equations can be solved in a single step as no function evaluations are required (the PDE has constant coefficients). But the method can lead to very large matrices that require iterative methods for solution.

If the problem is Neumann, then the rate of change of the variable along an outward facing normal to the boundary is given. One then defines an auxiliary set of grid points OUTSIDE the region and then approximate the slope along the boundary normal by the first difference between the points on the boundary and the appropriate auxiliary points (see Gerald example 8.6).

4. Initial value problems

Both parabolic and hyperbolic equations can be thought of as 'Initial Value Problems' (another name is 'Cauchy' problems). They both describe the time evolution of something. The elliptic equation above is 'static' in that its solution does not evolve with time.

4.1. Flux conservation problems

The finite difference approach is often used to solve partial differential equations where the underlying physics is based on flux conservation. A physical problem can often be posed in the vector form

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\partial \mathbf{F}(\mathbf{u})}{\partial x}$$

The vector function **F** (the conserved flux) may depend on spatial derivatives of **u** as well as **u** itself. An example is Maxwell's equations

where **u** has 6 elements made up from **E** and **H**. Such equations represent a 'flux', such as a fluid, that must be conserved as it 'flows'.

The reason for introducing flux conservation (apart from providing background information on important physical principles) is to consider the solution of the simple first order hyperbolic PDE

$$\frac{\partial \psi}{\partial t} = -v\frac{\partial \psi}{\partial x}$$

(sometimes called an 'Advection' equation, as it describes fluid flowing with velocity v while carrying some local change in the variable ψ, such as salt concentration).

One could consider trying to solve the equation above using simple finite differences using the scheme

$$\frac{\partial \psi}{\partial t} \cong \frac{\psi_j^{n+1} - \psi_j^n}{\delta t}$$

$$\frac{\partial \psi}{\partial x} \cong \frac{\psi_{j+1}^n - \psi_{j-1}^n}{2\delta x}$$

ψ_j^n represents the value of the dependent variable at time-step 'n' at spatial grid-point 'j'.

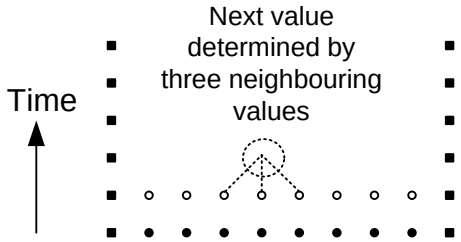


Figure 3 Illustration of the time evolution of the FTCS scheme

The solution process is illustrated in Figure 3 and is called a Forward Time Centred Space scheme (FTCS). The next value of the dependent variable is determined completely by a previous set of 3 values and the solution evolves with time along the time axis as shown. The squares represent the boundary conditions and the filled circles the initial conditions. The open circles are the previously computed solution points.

The problem with the FTCS scheme for this first order PDE (and the reason for including it here) is that it is unstable – it will not work. Care is needed when solving PDEs!

4.2. Stability Analysis

The very simple scheme we have adopted for the first order PDE above results in the difference equation

$$\frac{\psi_j^{n+1} - \psi_j^n}{\delta t} + v\left\{\frac{\psi_{j+1}^{n+1} - \psi_{j-1}^n}{2\delta x}\right\} = 0$$

The equation may be re-arranged to give

$$\psi_j^{n+1} = \psi_j^n - \frac{v\delta t}{2\delta x}\{\psi_{j+1}^{n+1} - \psi_{j-1}^n\}$$

We let $\frac{v\delta t}{2\delta x} = r$ leading to the equation

$$\psi_j^{n+1} = \psi_j^n - r\{\psi_{j+1}^{n+1} - \psi_{j-1}^n\}$$

We now consider the propagation of errors through the solutions mesh. Suppose that a row of the solution is in error, then all errors can only occur inside the boundary as the boundary is a given – we thus set the

boundary value errors to zero. We can represent the evolution of the set of errors at stage ‘n’ by successive applications of the matrix equation

$$\boldsymbol{\psi}^{n+1} = \begin{bmatrix} 1 & -r & & & & \\ r & 1 & -r & & & \\ & r & 1 & -r & & \\ & & \ddots & \ddots & \ddots & \\ & & & r & 1 & -r \\ & & & & r & 1 \end{bmatrix} \boldsymbol{\psi}^{n}$$

Let the matrix in the iteration be called ‘A’; it is of form:

$$\mathbf{A} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} + r \begin{bmatrix} & -1 & \\ 1 & & \ddots \\ & \ddots & \end{bmatrix}$$

The component on the left is the unit matrix and the component on the right is anti-symmetric. The eigenvalues of the matrix on the right must be either zero or purely imaginary (as it is anti-symmetric) thus the eigenvalues of the matrix A must be of the form

$$\lambda_j(\mathbf{A}) = 1 + ir\mu_j \Longrightarrow \left|\lambda_j(\mathbf{A})\right| = \sqrt{1 + \left(r\mu_j\right)^2}$$

As the anti-symmetric matrix is not zero, it must have a non-zero eigenvalue. Thus the eigenvalue of A of largest magnitude must have a

magnitude larger than 1. Thus the iterative scheme is unstable and any error introduced into the calculation will grow without bound.

4.3. The heat equation – an explicit method for a parabolic equation and an important lesson about stability.

The heat equation is a parabolic partial differential equation and is given by

$$\frac{\partial U}{\partial t} = \frac{k}{c\rho}\frac{\partial^2 U}{\partial x^2}$$

where c is the specific heat capacity, ρ is density, and k conductivity. This equation can be discretized in time and space using the difference approximations

$$\frac{\partial U}{\partial t} = \frac{U_j^{n+1} - U_j^n}{\delta t} + \mathrm{O}(\delta t)$$

$$\frac{\partial^2 U}{\partial x^2} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\delta x^2} + \mathrm{O}(\delta x^2)$$

If we call the numerical solution 'w' for the exact solution 'U' (as per Burden and Faires Section 12.2) we seek a solution to the difference equation

$$\frac{w_j^{n+1} - w_j^n}{\delta t} - \frac{k}{c\rho}\left\{\frac{w_{j+1}^n - 2w_j^n + w_{j-1}^n}{\delta x^2}\right\} = 0$$

Let $r = \frac{k\delta t}{c\rho\delta x^2}$ then the 'next' temperature is given by the equation

$$w_j^{n+1} = w_j^n + r\left\{w_{j+1}^n - 2w_j^n + w_{j-1}^n\right\}$$

Such an equation is called 'explicit' because the 'next' value is given explicitly in terms of the last values to be computed (refer again to Figure 3 above for the flux conservation introductory example). We again consider the propagation of errors via the iterative scheme above (the error on the boundary again being zero as they are known in advance),

so that the zero error boundary solution can be considered as the successive application of the matrix equation

$$\boldsymbol{\psi}^{n+1} = \begin{bmatrix} 1-2r & r & & & & & \\ r & 1-2r & r & & & & \\ & r & 1-2r & r & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & r & 1-2r & r & \\ & & & & r & 1-2r & \end{bmatrix} \boldsymbol{\psi}^{n}$$

where ψ is the vector of errors. We again call the iteration matrix 'A' and split into two halves

$$\mathbf{A} = (1-2r)\begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & 1 & \end{bmatrix} + r\begin{bmatrix} 0 & 1 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & \ddots & & & \\ & & \ddots & \ddots & 1 & & \\ & & & 1 & 0 & 1 & \\ & & & & 1 & 0 & \end{bmatrix}$$

We now have (1-2r) times a unit matrix plus r times a symmetric matrix. The eigenvalues are thus now all real (the previous example resulted in

1+imaginary) and so we are interested in the eigenvalues of a matrix with all 1's down the sub and super diagonals.

Note: I have not found a good reference for the following stability analysis.

Consider the matrix on the right. Let the matrix be n by n and consider the set of vectors k=1..n with the jth element given by

$$[\boldsymbol{v}_k]_j = \left[\sin\left(\frac{kj\pi}{n+1}\right)\right], j = 1..n$$

Multiply the right hand matrix by $\mathbf{v}_k$. The jth element of the product is given by

$$[\boldsymbol{A}\boldsymbol{v}_k]_j = \sin\left(\frac{k(j-1)\pi}{n+1}\right) + \sin\left(\frac{k(j+1)\pi}{n+1}\right) = 2\sin\left(\frac{kj\pi}{n+1}\right)\cos\left(\frac{k\pi}{n+1}\right)$$

$$= 2[\boldsymbol{v}_k]_j \cos\left(\frac{k\pi}{n+1}\right)$$

The above equation is not correct for the first and last row (we are missing a column on the left for the first row and a column on the right for the last row). But the $\sin\left(\frac{k(j-1)\pi}{n+1}\right)$ is zero for j=1, and thus drops out and hence gives the correct equation. The last row is for j=n, and thus $\sin\left(\frac{k(j+1)\pi}{n+1}\right)$ goes to zero (sin(kπ)=0) and drops out – so the equation works for the last row as well. Thus the vector $\mathbf{v}_k$ is an eigenvector

(remember: **Ax** = λ**x** is the defining equation for eigenvalues) and the eigenvalues of the matrix component on the right are

$$\lambda_k = 2\cos\left(\frac{k\pi}{n+1}\right)$$

Given the above, the eigenvalues of the matrix 'A' are made up from the two components, that from the scaled unit matrix and the off diagonal part, i.e.

$$\lambda_k(\mathbf{A}) = \{1-2r\} + r2\cos\left(\frac{k\pi}{n+1}\right) = 1 - 4r\left\{\sin\left(\frac{k\pi}{2(n+1)}\right)\right\}^2$$

For the errors to not grow without bound we need the eigenvalues of A to have a modulus less than 1, thus adopting 1 as the upper bound of $\sin^2 x$,

$$|1-4r| \leq 1 \Rightarrow 4r \leq 2 \Rightarrow r \leq \frac{1}{2}$$

It looks as though there exist conditions under which this solution scheme is stable.

The interesting point about the stability analysis above is that it requires

$$r = \frac{c\rho\delta t}{k\delta x^2} \leq \frac{1}{2} \Rightarrow \delta t \leq \frac{k\delta x^2}{2c\rho}$$

Suppose that you are not happy with accuracy of the solution to the heat equation at some critical position inside the boundary, so you decide to increase the number of spatial sample points (make δx smaller). You cannot do this without reducing the step size in time and satisfying the stability constraints above! The time-step is constrained to be below a

threshold set by the size of the spatial grid (sometimes referred to the cell diffusion time).

4.4. Richardson's method

In the above we have used the difference approximation

$$\frac{\partial U}{\partial t} = \frac{U_j^{n+1} - U_j^n}{\delta t} + O(\delta t)$$

This approximation can be improved on by using the central difference

$$\frac{\partial U}{\partial t} = \frac{U_j^{n+1} - U_j^{n-1}}{2\delta t} + O(\delta t^2)$$

The iterative equations are now

$$\frac{w_j^{n+1} - w_j^{n-1}}{2\delta t} - \frac{k}{c\rho}\left\{\frac{w_{j+1}^n - 2w_j^n + w_{j-1}^n}{\delta x^2}\right\} = 0$$

The stability analysis using the matrix method used previously is much more tricky and a technique called von Neumann analysis is more appropriate.

4.5. Crank-Nicholson method

Richardon's method achieves quadratic accuracy in time by using the central difference for approximating the time derivative. The Crank-

Nicholson method takes the backward difference but assumes it is the central difference at half the time step, i.e.

$$\left.\frac{\partial U}{\partial t}\right\}_{\delta t/2} = \frac{U_j^{n+1} - U_j^n}{2\,\delta t/2} + 0\left(\left(\frac{\delta t}{2}\right)^2\right)$$

We now need to find an approximation for the second partial spatial derivative at this half-way point (we take the mean)

$$\left.\frac{\partial^2 U}{\partial x^2}\right\}_{\delta t/2} \cong \frac{1}{2}\left\{\frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{\delta x^2} + \frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{\delta x^2}\right\}$$

This leads to the difference equation

$$\frac{w_j^{n+1} - w_j^n}{\delta t} - \frac{k}{2c\rho}\left\{\frac{w_{j+1}^n - 2w_j^n + w_{j-1}^n}{\delta x^2} + \frac{w_{j+1}^{n+1} - 2w_j^{n+1} + w_{j-1}^{n+1}}{\delta x^2}\right\} = 0$$

Again using $r = \frac{k\delta t}{c\rho\delta x^2}$ we get

$$-rw_{j+1}^{n+1} + (2+r)w_j^{n+1} - rw_{j-1}^{n+1} = rw_{j+1}^n + (2-r)w_j^n - rw_{j-1}^n$$

We now have three unknowns on the left hand side. We thus have to solve a set of simultaneous equations at each iteration – such methods are called ‘implicit’ methods as the next step is not given explicitly by the previous values (and hence each ‘next’ point is independent of all other

'next' points) but must be obtained by solution of a problem linking ALL the next values.

4.6. Hyperbolic second order equations

The typical example of a second order hyperbolic PDE is the wave equation

$$c^2 \frac{\partial^2 \phi}{\partial x^2} = \frac{\partial^2 \phi}{\partial t^2}$$

The central difference can be used to generate the difference equation

$$c^2 \left\{ \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{h^2} + O(h^2) \right\} = \frac{\phi_i^{n+1} - 2\phi_i^n + \phi_i^{n-1}}{k^2} + O(k^2)$$

If the error terms are ignored, the above equation leads to an explicit method (the 'next' value in time is expressed in terms of previous values)

$$\phi_i^{n+1} = 2\left(1 - \frac{c^2 k^2}{h^2}\right)\phi_i^n + \frac{c^2 k^2}{h^2}(\phi_{i+1}^n + \phi_{i-1}^n) - \phi_i^{n-1}$$

Note the last term ϕ_i^{n-1}. This term is a previous value and not the current value of the dependent variable. There is thus a difference from the parabolic equation, where only the initial values are required on the boundary. This problem is usually overcome by using the initial rate of change $g_i = \frac{\partial \phi_i^0}{\partial t}$, as well as the initial values of the dependent variable ϕ_i^0. There are many ways to perform this operation:

- Use a forward difference approximation to generate an initial estimate of ϕ_i^1 using $\phi_i^1 = \phi_i^0 + k\frac{\partial \phi_i^0}{\partial t} + \frac{k^2}{2}\frac{\partial^2 \phi}{\partial t^2}(x_i, \xi_i)$. The resulting approximation is $\phi_i^1 \cong \phi_i^0 + k g_i$. We can now iterate onwards for

n=2 onwards. This method is O(k) and is the one described in the slides.

- Use a backward difference approximation to generate a 'previous' value for the dependent variable as $\phi_i^{-1} \cong \phi_i^0 - kg_i$. We can now iterate from n=1 onwards. This method is also O(k).

The method in Burden and Faires is to use the PDE itself in a manner similar to that used to generate the Runge Kutta family of methods. We start with the Taylor series for the dependent variable

$$\phi_i^1 = \phi_i^0 + k\frac{\partial \phi_i^0}{\partial t} + \frac{k^2}{2}\frac{\partial^2 \phi}{\partial t^2}(x_i, 0) + \frac{k^3}{6}\frac{\partial^2 \phi}{\partial t^2}(x_i, \xi_i)$$

$$= \phi_i^0 + kg_i + \frac{k^2}{2}c^2\frac{\partial^2 \phi}{\partial x^2}(x_i, 0) + \frac{k^3}{6}\frac{\partial^3 \phi}{\partial t^3}(x_i, \xi_i)$$

The PDE has been used to substitute the second spatial derivative for the second time derivative, we thus have the central difference approximation

$$\frac{\partial^2 \phi}{\partial x^2}(x_i, 0) = \frac{\phi_{i+1}^0 - 2\phi_i^0 + \phi_{i-1}^0}{h^2} - \frac{h^2}{12}\frac{\partial^4 \phi}{\partial x^4}(\xi_i, 0)$$

Substituting into the difference equation

$$\phi_i^1 = \phi_i^0 + kg_i + \frac{c^2k^2}{2h^2}\{\phi_{i+1}^0 - 2\phi_i^0 + \phi_{i-1}^0\} + O(k^3 + h^2k^2)$$

The above gives a more accurate estimate of the first generated row of data ready for the general two-row difference equation. Further accuracy can be obtained by noting

$$\frac{\partial^3 \phi}{\partial t^3}(x_i, 0) = \frac{\partial}{\partial t}\frac{\partial^2 \phi}{\partial t^2}(x_i, 0) = \frac{\partial}{\partial t}c^2\frac{\partial^2 \phi}{\partial x^2}(x_i, 0) = c^2\frac{\partial^2}{\partial x^2}\frac{\partial \phi}{\partial t}(x_i, 0) = c^2\frac{\partial^2 g_i}{\partial x^2}$$

$$= c^2\left\{\frac{g_{i+1} - 2g_i + g_{i-1}}{h^2} - \frac{h^2}{12}\frac{d^4 y}{dx^4}g(\xi)\right\}$$

You are welcome to plug this approximation into the initialisation scheme!

4.6.1. Stability

Stability analysis of the method above is tricky (I have also not been able to find a good reference to the matrix method, I have taken the method used for the analysis of the parabolic equation in Burden and Faires to its logical conclusion below and is only included for information).

Let $\frac{c^2k^2}{h^2} = r$, then the difference equation (after the initial start-up) is

$$\phi_i^{n+1} = 2(1-r)\phi_i^n + r(\phi_{i+1}^n + \phi_{i-1}^n) - \phi_i^{n-1}$$

Leading to the error propagation equation

$$\boldsymbol{\psi}^{n+1} = \begin{bmatrix} 2-2r & r & & & & & \\ r & 2-2r & r & & & & \\ & r & 2-2r & r & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & r & 2-2r & r & \\ & & & & r & 2-2r & \end{bmatrix} \boldsymbol{\psi}^n - \boldsymbol{\psi}^{n-1}$$

where $\boldsymbol{\psi}$ is a vector of errors (following the same reasoning as in the parabolic equation case). The problem is that the 'next' error is dependent on the previous error AND the error previous to that. We

augment the state (as per Lecture 3 on higher order ordinary differential equations) and represent the above as

$$\begin{bmatrix} \boldsymbol{\psi}^{n+1} \\ \boldsymbol{\psi}^{n} \end{bmatrix} = \begin{bmatrix} \Gamma & \mathbf{-I} \\ \mathbf{I} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\psi}^{n} \\ \boldsymbol{\psi}^{n-1} \end{bmatrix}$$

where

$$\Gamma = \begin{bmatrix} 2-2r & r & & & & & \\ r & 2-2r & r & & & & \\ & r & 2-2r & r & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & r & 2-2r & r & \\ & & & & r & 2-2r & \end{bmatrix}$$

The capital 'I' entries are unit matrices and the iteration equation above is a 'block' matrix equation. The eigenvalues are the solutions of

$$\det \begin{bmatrix} \Gamma - \lambda & \mathbf{-I} \\ \mathbf{I} & -\lambda \end{bmatrix} = 0$$

To find the eigenvalues we express the above using the 'Schur form' for block determinants (see http://en.wikipedia.org/wiki/Determinant#Block_matrices) and obtain

$$\det \begin{bmatrix} \Gamma - \lambda & \mathbf{I} \\ \mathbf{I} & -\lambda \end{bmatrix} = \det[-\lambda] \det[\Gamma - \lambda - \lambda^{-1}] = 0 \ (\lambda \neq 0)$$

If we let $\mu = \lambda + \lambda^{-1}$, the admissible solutions are those for which

$$\det[\Gamma - \mu] = 0$$

The matrix Γ is made up from a scaled unit matrix (the scaling is 2-2r) plus an off-diagonal part. But we have seen the off-diagonal part before

(from the parabolic stability analysis, and it has eigenvalues $2r\cos\left(\frac{k\pi}{n+1}\right)$).

The eigenvalues are thus solutions to

$$\frac{\lambda^2+1}{\lambda} = \mu = 2\left\{1 - r + r\cos\left(\frac{k\pi}{n+1}\right)\right\}$$

Or, expanding the expression for λ

$$\lambda^2 - 2\left\{1 - r + r\cos\left(\frac{k\pi}{n+1}\right)\right\}\lambda + 1 = 0$$

The defining equation is of the form

$$\lambda^2 - 2(1-a)\lambda + 1 = 0$$

$$a = r - r\cos\left(\frac{k\pi}{n+1}\right) = r\left(1 - \cos\left(\frac{k\pi}{n+1}\right)\right)$$

Let us assume that $r < 1 \Rightarrow a < 2$ then the roots are

$$\lambda = (1-a) \pm i\sqrt{2a - a^2}$$

$$|\lambda|^2 = (1-a)^2 + 2a - a^2 = 1$$

And all the eigenvalues lie on a unit circle centred at the origin. An eigenvalue analysis of the method is thus inconclusive. The errors at any stage can be projected onto the complex eigenvectors and the resulting complex components will rotate in the complex plane as the iterations proceed – this predicts that the errors will be periodic as the method proceeds. Numerical rounding errors will accumulate and propagate as waves as the method progresses. As an aside, all norms

applied to the matrix will result in a value greater than 1.0, thus norm arguments cannot be used to prove stability.

If r>1 then it is possible for a>2 (for some k) and

$$\lambda = (1 - a) \pm \sqrt{a^2 - 2a}$$

At least one pair of eigenvalues will be real and $(1 - a) - \sqrt{a^2 - 2a} < -1$, i.e. the modulus of one of the eigenvalues will be greater than 1 and the method will be unstable.

$\frac{c^2 k^2}{h^2} \leq 1$ is the generally accepted condition for the 'stability' of the method (the analysis above is inconclusive and is probably why it does not appear in the books!). Note: Burden and Faires references the proof of stability in another book – Eissacson and H B Keller 'Analysis of numerical methods' which uses another method for the approximate analysis of the stability of PDE solution methods (von Neumann's Stability Analysis, see http://en.wikipedia.org/wiki/Von_Neumann_stability_analysis.)