Scientific Computing Notes

R. Wait

January 16, 2009

UPPSALA UNIVERSITET
Inst. för informationsteknologi
Avd. för teknisk databehandling

UPPSALA UNIVERSITY
Information Technology
Dept. of Scientific Computing

Contents

Chapter 1

Matrix Algebra

1.1 multiplication

For two matrices with *compatible dimensions* such as

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1k} & \cdots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ik} & \cdots & a_{iN} \\ \vdots & & \vdots & & \vdots \\ a_{M1} & \cdots & a_{Mk} & \cdots & a_{MN} \end{pmatrix} = \{a_{ik}\}_{1\le i\le M, 1\le k\le N}$$

with M rows and N columns

$$B = \begin{pmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1L} \\ \vdots & & \vdots & & \vdots \\ b_{k1} & \cdots & b_{kj} & \cdots & b_{kL} \\ \vdots & & \vdots & & \vdots \\ b_{N1} & \cdots & b_{Nj} & \cdots & b_{NL} \end{pmatrix} = \{b_{kj}\}_{1\le k\le N, 1\le j\le L}$$

with N rows and L columns, then matrix multiplication defines the product as the $M \times L$ matrix

$$C = AB = \{c_{ij}\}_{1\le i\le M, 1\le j\le L}$$

where

$$\begin{aligned} c_{ij} &= a_{i1}b_{1j} + \cdots + a_{ik}b_{kj} + \cdots + a_{iN}b_{Nj} \\ &\equiv \sum_{k=1}^{N} a_{ik}b_{kj} \end{aligned}$$

That is, a component of the product is the *inner product* of a row of A with a column of B In general $AB \neq BA$, if AB exists it is not necessarily true that BA exists, *e.g.* if $L \neq M$ above. The *unit matrix* or *identity matrix* I has ones on the diagonal and zeros elsewhere and for any $M \times N$ matrix A,

$$IA = AI = A$$

where the identity on the right is $M \times M$ and on the left is $N \times N$. A *permutation matrix* P has exactly one unit component in each row and each column, all other components are

zero. The matrix multiplication PA permutes the rows of A and AP permutes the columns of A.

For addition the matrices must have the same dimensions, thus if A and B are both $M \times N$ then

$$D_{(M\times N)} = A + B = \{d_{ij}\}_{1\le i\le M, 1\le j\le N}$$
$$d_{ij} = a_{ij} + b_{ij}$$

examples

$$A = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}, \; B = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{pmatrix} \text{ and } P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

then

$$AB = \begin{pmatrix} 9 & 14 & 19 & 24 \\ 15 & 22 & 29 & 36 \\ 21 & 30 & 39 & 48 \end{pmatrix}$$

and

$$AP = \begin{pmatrix} 4 & 1 \\ 5 & 2 \\ 6 & 3 \end{pmatrix}, \; PB = B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

1.2 transpose and inverse

The *transpose* of the $M \times N$ matrix A is

$$A^T = \begin{pmatrix} a_{11} & \cdots & a_{k1} & \cdots & a_{M1} \\ \vdots & & \vdots & & \vdots \\ a_{1i} & \cdots & a_{ki} & \cdots & a_{Mi} \\ \vdots & & \vdots & & \vdots \\ a_{1N} & \cdots & a_{kN} & \cdots & a_{MN} \end{pmatrix}$$

so if

$$A = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

then

$$A^T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

A square matrix A such that $A^T = A$ is said to be symmetric. The inverse of a *square* matrix A is denoted by A^{-1} and is such that

$$AA^{-1} = A^{-1}A = I$$

Not every square matrix has an inverse, if no inverse exists the matrix is said to be *singular*. Note on the notation: Vectors, denoted in bold such as

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

are always assumed to be *column vectors*, that is matrices of dimension $n \times 1$ where n is the length of the vector. So $\boldsymbol{x}^T$ is a row vector.

Exercise 1.1 *Verify that the permutation matrix P that interchanges rows i and j can be defined as*

$$P = I - (\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^T$$

where $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ are respectively the i-th and j-th unit vectors.

1.3 norms

The size of matrices and vectors are represented in terms of *norms* which are generalisations of the concept in three dimensions where the length of a vector $\boldsymbol{x} = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^T$ is denoted by $\|\boldsymbol{x}\|_2$ and is defined by

$$\|\boldsymbol{x}\|_2^2 = x_1^2 + x_2^2 + x_3^2$$

This is generalised for any $\boldsymbol{x} = \begin{pmatrix} x_1 & \cdots & x_N \end{pmatrix}^T$ as

$$\|\boldsymbol{x}\|_2 = \left[\sum_{i=1}^{N} x_i^2\right]^{\frac{1}{2}}$$

it is possible to define other norms the most common are

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{N} |x_i|$$

and

$$\|\boldsymbol{x}\|_\infty = \max\{|x_i|, i = 1, \cdots, N\}$$

thus if $\boldsymbol{x} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}^T$, $\|\boldsymbol{x}\|_1 = 6$, $\|\boldsymbol{x}\|_2 = 3.74$, $\|\boldsymbol{x}\|_\infty = 3$. All norms satisfy the same basic axioms:

1. $\|\boldsymbol{x}\| \geq 0$
2. $\|\boldsymbol{x}\| = 0 \iff \boldsymbol{x} = 0$
3. $\|\alpha\boldsymbol{x}\| = |\alpha| \, \|\boldsymbol{x}\|$ for all $\alpha \in \mathbb{R}$
4. $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ The triangle inequality

The norm of a matrix is defined in terms of the norm of a vector as

$$\|A\| = \max_{\|\boldsymbol{x}\| \neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

it can be shown that the simplest cases reduce to

$$\|A\|_1 = \max_{1 \leq i \leq N} \{\sum_{j=1}^{N} |a_{ji}|\},$$

the maximum column sum, and

$$\|A\|_\infty = \max_{1\le i\le N}\{\sum_{j=1}^{N}|a_{ij}|\},$$

the maximum row sum. Matrix norms satisfy an additional axiom

1. $\|AB\| \le \|A\|\,\|B\|$

The condition number, $\kappa(A)$, of a matrix defined by

$$\kappa(A) = \|A^{-1}\|\|A\|$$

is a useful indicator of the sensitivity of the problem to small changes in the data. If the condition number is large, we say that the problem is *ill conditioned* whereas if it is small the problem is *well conditioned*. Note that an accurate evaluation of the condition number requires knowledge of the *inverse* A^{-1} as this is invariably not available computer packages (such as *MATLAB*) make of *estimates* of the condition number.
Note on the notation: Components of vectors and matrices are denoted by subscripts *e.g.* x_i, a_{jk}. *etc.* elements of a sequence are denoted by superscripts *e.g.* $x^{(n)}$, the elements in the sequence could be vectors *e.g.* $\boldsymbol{x}^{(n)}$, then the components are $x_i^{(n)}$, *etc.*.

Exercise 1.2 `Matlab Exercise` *Construct a random* 5×5 *matrix* A *and compute* $\dfrac{\|A\boldsymbol{x}\|_1}{\|\boldsymbol{x}\|_1}$ *for 100 different random vectors* $\boldsymbol{x}^{(n)}$, $n = 1,\ldots,100$. *Compute* $\|A\|_1$ *and* $\max_n\left\{\dfrac{\|A\boldsymbol{x}^{(n)}\|_1}{\|\boldsymbol{x}^{(n)}\|_1}\right\}$ *are they close, explain the difference?*

1.4 Simultaneous Linear Equations

Systems of simultaneous linear algebraic equations are written as

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1.1}$$

where A is an $N\times N$ matrix with $\boldsymbol{x}$ and $\boldsymbol{b}$ column vectors of length N, or explicitly a component-wise definition is

$$\sum_{j=1}^{N} a_{ij}x_j = b_i \qquad i = 1,\cdots,N$$

If the matrix A is *regular* or (more commonly) *non-singular* then the system (1.1) has exactly one solution and the problem is said to be *well-posed*. If the matrix A is singular, then $A\boldsymbol{x} = \boldsymbol{b}$ may have *no solutions* or *an infinity of solutions* depending on the form of $\boldsymbol{b}$ *e.g.*

$$\begin{aligned} x + y &= a \\ 2x + 2y &= b \end{aligned}$$

has no solution if $b \ne 2a$ and has the solution $\begin{pmatrix} x \\ a-x \end{pmatrix}$ for any value of x if $b = 2a$. In either case the problem is said to be *ill-posed* In general, if the rows (or equivalently

the columns) of a matrix are linearly dependent then the matrix is singular. A sequence of vectors (either rows or columns) $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_i, \cdots, \boldsymbol{a}_N$ is linearly dependent if there exist a sequence of scalars (*i.e.* numbers) $\alpha_1, \cdots, \alpha_i, \cdots, \alpha_N$ such that

$$\alpha_1 \boldsymbol{a}_1 + \cdots + \alpha_i \boldsymbol{a}_i + \cdots + \alpha_N \boldsymbol{a}_N = 0$$

if we define the matrix A with columns $\boldsymbol{a}_i, i = 1, .., N$ this becomes

$$A\boldsymbol{x} = 0$$

where $\boldsymbol{x} = (\alpha_1, \cdots, \alpha_i, \cdots, \alpha_N)^T$.

If the vectors are not linearly dependent they are said to be *linearly independent*. The *rank* of a square matrix is the number of linearly independent rows (or columns). The theory is usually summarised as in theorem 1. A singular matrix is said to be *rank deficient*, a non-singular matrix is said to have *full rank*.

Theorem 1 *Let A be a square matrix of order N, the following statements are equivalent:*

1. *For any vector $\boldsymbol{b}$, the solution of $A\boldsymbol{x} = \boldsymbol{b}$ is unique.*
2. *If a solution of the system $A\boldsymbol{x} = \boldsymbol{b}$ exists it is unique.*
3. *For all $\boldsymbol{x}$, $A\boldsymbol{x} = 0 \Rightarrow \boldsymbol{x} = 0$.*
4. *The columns (rows) of A are linearly independent.*
5. *There is a matrix A^{-1} such that $A^{-1}A = AA^{-1} = I$.*
6. $\det(A) \neq 0$

The solution vector $\boldsymbol{x} \equiv \begin{pmatrix} x_1 \cdots x_i \cdots x_N \end{pmatrix}^T$ can be written as $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ but this formula should **never be used for numerical computation**.

Example 1

- $\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$ *rank* 1
- $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 3 & 6 & 9 \end{pmatrix}$ *rank* 2
- $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 7 \\ 5 & 7 & 10 \end{pmatrix}$ *rank* 2
- $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$ *rank* 1

If the matrix is not symmetric then the coefficients α_i linking the rows will not be the same as those linking the columns.

1.5 Eigenvalues and Eigenvectors

Given any square matrix A, the *eigenvalue problem* or *eigenproblem* is to find a vector $\boldsymbol{x}$ (an *eigenvector*), with at least one component not zero, and a scalar λ (an *eigenvalue*) such that

$$A\boldsymbol{x} = \lambda\boldsymbol{x} \tag{1.2}$$

Clearly $\boldsymbol{x} = 0$ is a trivial solution of (1.2) that we exclude. If $\boldsymbol{x}$ is a solution of (1.2) so is $\alpha\boldsymbol{x}$, any multiple of a eigenvector is an eigenvector, so eigenvectors are frequently normalised so $\boldsymbol{x}^*\boldsymbol{x} = 1$

It follows from (1.2) and theorem 1 that the eigenvalues λ of an $x \times N$ matrix A satisfy

$$\det(A - \lambda I) = 0 \tag{1.3}$$

this is a polynomial of degree N in λ, known as the *characteristic equation* of A. A polynomial of degree N has N roots, which may not be distinct and which may be complex. Thus a real matrix A may have multiple eigenvalues or complex eigenvalues. If a matrix has distinct eigenvalues then the eigenvectors are linearly independent, if there are multiple eigenvalues then the structure of the system of eigenvectors is more complicated. Since the determinant of A^T equals the determinant of A, they have the same eigenvalues

$$A^T\boldsymbol{y} = \lambda\boldsymbol{y} \qquad \text{or} \qquad \boldsymbol{y}^T A = \lambda\boldsymbol{y}^T \tag{1.4}$$

but different eigenvectors, thus if $\boldsymbol{x}$ is an eigenvector of A corresponding to the eigenvalue λ and $\boldsymbol{y}$ is an eigenvector of A^T corresponding to the eigenvalue $\tilde{\lambda}$ with $\lambda \neq \tilde{\lambda}$ then $y^T x = 0$. Thus the eigenvectors of a symmetric matrix are orthogonal. Assuming that there are N linearly independent eigenvectors $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ (for an $N \times N$ matrix A) then if $U = (\boldsymbol{x}_1 \dots \boldsymbol{x}_N)$ and $V = (\boldsymbol{y}_1 \dots \boldsymbol{y}_N)$ it follows that $UV^T = I$ (or $V^T = U^{-1}$) and $V^T AU = D$ where $D = \text{diag}(\lambda_1, \dots, \lambda_N)$, this can also be written as $UDV^T = A$.

Example 2

$$A = \begin{pmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{pmatrix} \qquad \begin{aligned} \lambda &= 11, \quad \boldsymbol{x} = (2\ 4\ 3)^T \\ \lambda &= -3, \quad \boldsymbol{x} = (0\ 2\ -3)^T \\ \lambda &= -2, \quad \boldsymbol{x} = (1\ 2\ -5)^T \end{aligned}$$

$$A^T = \begin{pmatrix} 2 & 10 & 3 \\ 3 & 3 & 6 \\ 2 & 4 & 1 \end{pmatrix} \qquad \begin{aligned} \lambda &= 11, \quad \boldsymbol{y} = (4\ 3\ 2)^T \\ \lambda &= -3, \quad \boldsymbol{y} = (2\ -1\ 0)^T \\ \lambda &= -2, \quad \boldsymbol{y} = (-9\ 3\ 2)^T \end{aligned}$$

1.5.1 Mechanical Vibration

Consider the motion of five equally spaced masses on a taut string. The system defining the *natural modes* of vibration can be written in matrix terms as

$$K\boldsymbol{x} = -\omega^2\boldsymbol{x}$$

or

$$(K + \omega^2 I)\boldsymbol{x} = 0$$

and this is an *eigenvalue problem*. A solution in this case consists of an *eigenvalue* (in this case ω^2 and hence positive, and a non-zero *eigenvector* $\boldsymbol{x}$. In structural terms, the matrix K is the *stiffness matrix*, ω is a *natural frequency* and $\boldsymbol{x}$ is the *mode* corresponding to that frequency.

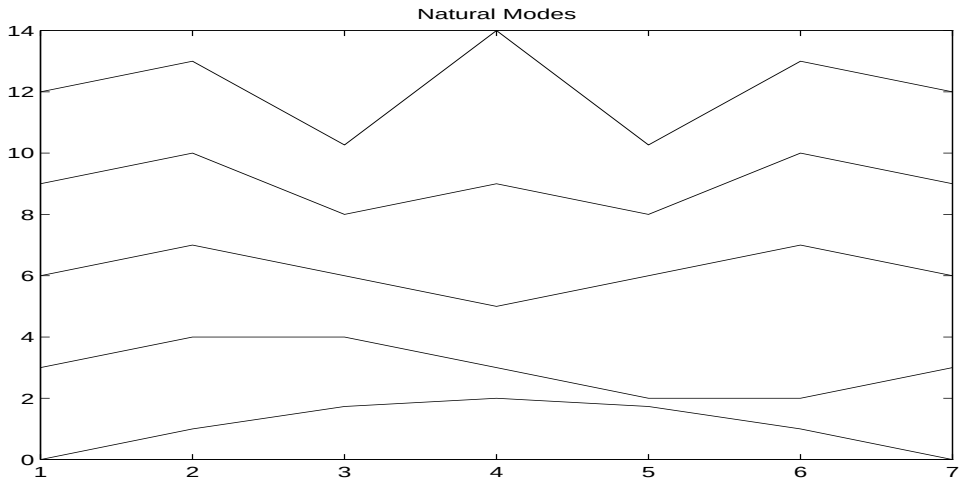


Figure 1.1: Natural modes of equally spaced vibrating masses

In terms of this simple example of five masses on a string, if all the masses are equal

$$K = \frac{T}{h}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}$$

where T is the tension and h is the spacing between the masses. For $\boldsymbol{x}_1 = \begin{pmatrix} 1 & -\sqrt{3} & 2 & -\sqrt{3} & 1 \end{pmatrix}^T$ then

$$K\boldsymbol{x}_1 = -(2+\sqrt{3})\frac{T}{h}\boldsymbol{x}_1 \qquad \sqrt{3} = 1.73, 2+\sqrt{3} = 3.73$$

for $\boldsymbol{x}_2 = \begin{pmatrix} 1 & -1 & 0 & -1 & 1 \end{pmatrix}^T$ then

$$K\boldsymbol{x}_2 = -3\frac{T}{h}\boldsymbol{x}_2$$

for $\boldsymbol{x}_3 = \begin{pmatrix} 1 & 0 & -1 & 0 & 1 \end{pmatrix}^T$ then

$$K\boldsymbol{x}_3 = -2\frac{T}{h}\boldsymbol{x}_3$$

for $\boldsymbol{x}_4 = \begin{pmatrix} 1 & 1 & 0 & -1 & -1 \end{pmatrix}^T$ then

$$K\boldsymbol{x}_4 = -\frac{T}{h}\boldsymbol{x}_4$$

for $\boldsymbol{x}_5 = \begin{pmatrix} 1 & \sqrt{3} & 2 & \sqrt{3} & 1 \end{pmatrix}^T$ then

$$K\boldsymbol{x}_5 = -(2-\sqrt{3})\frac{T}{h}\boldsymbol{x}_5 \qquad 2-\sqrt{3} = 0.27$$

Note the eigenvectors in figure (1.1) get "smoother" as the eigenvalues get smaller. At any given moment time=t, the position vector of the masses is $\boldsymbol{y} = \cos(\omega t)\boldsymbol{x}$. Usually in

mechanical vibration problems we wish to compute the *smallest* eigenvalue together with the corresponding eigenvalue, but in some problems (possibly electrical circuits) we wish to compute the largest eigenvalue.

Numerically, the largest eigenvalue is easier to compute so we consider that first.

1.5.2 Principal Component Analysis

This is usually viewed as a statistical technique, sometimes it is viewed as a technique in *Data Mining*. The idea is to characterise a set of observations as columns of a matrix and then identify the key features of the data in terms of the eigenvectors corresponding to the largest eigenvalues.

Eigenfaces Face recognition, the art of matching a face to a database of photographs, is an important security issue nowadays. An image of a face is a set of pixels, which are numerical values. The pixel values for each face form a vector, the vectors are put together as the columns of a matrix. The matrix is padded with zero columns to form a square matrix as there are usually many more pixels in a face than faces in the database. The eigenvectors can be interpreted as eigenfaces and then any individual face can be expressed as a linear combination of eigenfaces. Two images can then be compared in terms of the expansion in terms of eigenfaces.

Insect Taxonomy Small winged aphids *Alate Adelges* can be caught in a light trap and the point of interest of the study is to determine the number of distinct taxa (sub-populations) present in the particular habitat from which the trap was taking samples. Adelges are difficult to identify with any certainty by the conventional taxonomic keys and so principal component analysis is used to provide guidance on the number of distinct taxa present in the collection. It is possible to make a variety of measurements of the adelges, the variables chosen for this model were;

body length
body width
fore-wing length
hind-wing length
number of spiracles
length of antennal segment 1
length of antennal segment 2
length of antennal segment 3
length of antennal segment 4
length of antennal segment 5
number of antennal spines
leg length, tarsus
leg length, tibia
leg length femur
rostrum
ovipositor
number of ovipositor spines
anal fold (categorical data - *yes/no*)
number of fond-wing hooks

These 19 variables were chosen as being possible diagnostic characters and can be measured (or counted) by microscopic examination. The purpose of the model is to identify sub-groups in a sample, hence a diagrammatic representation that clusters the data is adequate if followed by a visual inspection for clusters. In an experiment N individuals are collected, the data is assembled into an $N \times 19$ data matrix A. The variability of each of the 19 sets of data with respect to each other are computed as the 19×19 covariance matrix $cov(A)$.

Covariance Matrix Given the raw data (n sets $\boldsymbol{a}_i$ of m parameter values, $n > m$) define

$$A = \begin{pmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_m \end{pmatrix}$$

and

$$E = \begin{pmatrix} \boldsymbol{a}_1 - \overline{a_1} & \boldsymbol{a}_2 - \overline{a_2} & \cdots & \boldsymbol{a}_m - \overline{a_m} \end{pmatrix}$$

then the $m \times m$ (sample) covariance matrix[1] can be defined as

$$cov(A) = \frac{1}{n-1} E^T E$$

If it is possible to identify just two factors that account for this variability then a scatter plot would split into distinct clusters if here were separate sub-populations. Thus if it is possible to identify two factors that account for "most" of the variability then it is possible to produce an "approximate" scatter plot which best represents the data, in the sense of being as widely dispersed as possible, by selecting vectors (in 19 dimensional space) $\boldsymbol{x}_i$ $i = 1, 2, \ldots, m$ that maximise the directional sum of squares

$$\boldsymbol{x}^T cov(A) \boldsymbol{x}$$

The vector $\boldsymbol{x}_1$ gives the maximum and $\boldsymbol{x}_2$ is the optimal subject to the restriction

$$\boldsymbol{x}_1^T \boldsymbol{x}_2 = 0$$

A scatter plot of the data projected onto $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ space is then the best two-dimensional representation of the data in the above sense. The directions $\boldsymbol{x}_i$ $i = 1, 2$ are the first two of the m eigenvectors of the matrix $cov(A)$ *i.e.*

$$cov(A)\boldsymbol{x}_i = \lambda_i \boldsymbol{x}_i$$

where the eigenvalues are ordered as

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m > 0$$

and the eigenvectors are orthogonal so that

$$\boldsymbol{x}_i^T \boldsymbol{x}_j = 0 \quad i \neq j$$

It can be shown that the total sum of squares is the sum of the eigenvalues hence

$$\frac{\lambda_1 + \lambda_2}{\sum \lambda_i}$$

[1] The analysis can alternatively be performed in terms of the correlation matrix. If the covariance matrix is denoted by C and the correlation matrix by K then $K_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} C_{jj}}}$

is the proportion of the variability illustrated in such a two-dimensional plot. Thus this quantity can be used as a validation of the model, a value of (say) less then 0.7 indicates there is a considerable body of information that is not modelled by this two-dimensional representation. In the case of the *Alate Adelges* data $\frac{\lambda_1+\lambda_2}{\sum \lambda_i} = 0.92$ and it is possible to identify 4 distinct clusters from the scatter plot.

Exercise 1.3 *Show that for any two vectors* $\boldsymbol{x}$, $\boldsymbol{y}$ *not necessarily the same length, that* $\boldsymbol{x}\boldsymbol{y}^T$ *is a matrix with rank one (a* rank-one matrix*) and that any rank-one matrix can be written in the form* $\boldsymbol{x}\boldsymbol{y}^T$.

Exercise 1.4 *Show that if* $UDV^T = A$ *then* $A^n = UD^nV^T$ *or* $A^T = \sum_{i=1}^{N} \lambda_i^n \boldsymbol{x}_i \boldsymbol{y}_i^T$.

Exercise 1.5 `Matlab Exercise` *Using the* Alate Adelges *data, compute the eigenvalues and eigenvectors of the covariance matrix and construct a two-dimensional scatter-plot (using* `plot` *and a three-dimensional scatter-plot (using* `plot3`*). Are the clusters easier to identify in 3-D? What is the proportion of the residual sum of squares attributed to* λ_3*?*

Chapter 2

Numerical Approximations

On any digital computer, numbers are stored to a finite precision in *floating point* form, typically as $2^e \times d$, d is known as the *mantissa* or *fraction* and e is the *exponent*. The round-off error is the difference between the exact answer (say) $x+y$ (the sum of two numbers) and the result of the evaluating the numerical value using floating point arithmetic, in this case denoted by $\mathrm{fl}(x+y)$. If there are t *bits* (*bi*nary digi*ts*) in the mantissa, then it is possible to state a bound on the round-off error incurred in a single arithmetic operation in the form

$$\mathrm{fl}(x+y) = (x+y)(1+e) \tag{2.1}$$

where

$$|e| \leq \epsilon_M = 2^{-t}$$

the constant ϵ_M is known as the machine precision. Alternatively we can write

$$|\mathrm{fl}(x+y) - (x+y)| \leq |(x+y)||e|$$

with similar bounds for the other arithmetic operations *viz* -, $\times$, /. Bounds of the form (2.1) lead to *backward error analysis*. The standard (IEEE) form of *single precision* arithmetic is to store numbers in the form

$$x = \pm 2^e \times d$$

where the e is stored as 8 binary digits (bits) and $d \equiv 0.d_1 \ldots d_{23}$ is stored as 23 bits, there is a single bit for the sign. This is a *binary floating point number*. In IEEE double precision e is 11 bits and d is 52 bits, and this is the form that *MATLAB* uses by default (64 bits including the sign bit).

Floating point numbers are *normalised* so that $d_1 \neq 0$ hence in base-β arithmetic, $x = \beta^e \times d$ then $1 > d \geq \beta^{-1}$ (an alternative normalisation would be to interpret $d = d_1.d_2 \ldots d_{23}$ then $\beta > d \geq 1$). If as a result of the computation a number is generated that is too large for the defined form of floating point representation then this is known *overflow* which is one form of *floating point exception* the unrepresentable value is replaced (in IEEE arithmetic) by the character string `Inf` and this is replicated throughout any subsequent computation. Alternatively, if the computed number is too small to be represented (e large and negative) this is termed *underflow* and the unrepresentable number replaced by zero. The result of the calculations `0/0` or `Inf/Inf` is `NaN` (not a number). For further information consult Wikepedia [6].

2.1 Rounding errors

The limit on the stored precision means that numbers have to be rounded to the fixed precision available. Thus for example in five digit decimal arithmetic ($t = 5$) $a = .x_1x_2x_3x_4x_5y_1y_2 \ldots\ (\times 10^k)$, with$x_1 \neq 0$ is rounded to $b = .x_1x_2x_3x_4z\ (\times 10^k)$. If we define $y = y_1.y_2y_3 \ldots$ then when a is rounded to b:

- $y \geq 5$ then (ignoring the complications with $x_5 = 9$ hence $z = 10$) $z = x_5 + 1$ and

$$b - a = (10 - y) \times 10^{-t-1}(\times 10^k)$$

- $y < 5$ then $z = x_5$ and

$$b - a = -y \times 10^{-t-1}(\times 10^k)$$

In both cases we have the *absolute error* $|b - a| \leq 5 \times 10^{-t-1}\ (\times 10^k) = \frac{1}{2} \times 10^{-t}\ (\times 10^k)$. Assuming that the numbers are normalised $x_1 \neq 0 \Rightarrow x_1 \geq 1$ the *relative error*

$$\frac{|b-a|}{|a|} \leq \frac{5 \times 10^{-t-1}\ (\times 10^k)}{0.x_1\ (\times 10^k)} \tag{2.2}$$

$$\leq \frac{5 \times 10^{-t-1}\ (\times 10^k)}{0.1\ (\times 10^k)} \tag{2.3}$$

$$= \frac{1}{2} \times 10^{-t+1} \tag{2.4}$$

With t-digit binary arithmetic the results are similar if $a = .x_1x_2x_3x_4x_5y_1y_2 \ldots\ (\times 2^k)$, with$x_1 \neq 0$ is rounded to $b = .x_1x_2x_3x_4z\ (\times 2^k)$.

- $y \geq 1 \Rightarrow y_1 = 1$ then (ignoring the complications of "carry digits" with $z = 2$) $z = x_5 + 1$ and

$$b - a = (2 - y) \times 2^{-t-1}(\times 2^k)$$

- $y < 1 \Rightarrow y_1 = 0$ then $z = x_5$ and

$$b - a = -y \times 2^{-t-1}(\times 2^k)$$

In both cases we have the *absolute error* $|b - a| \leq 2^{-t-1}\ (\times 2^k) = \frac{1}{2} \times 2^{-t}\ (\times 2^k)$. Assuming that the numbers are normalised $x_1 \neq 0 \Rightarrow x_1 = 1$ the *relative error*

$$\frac{|b-a|}{|a|} \leq \frac{2^{-t-1}\ (\times 2^k)}{0.x_1\ (\times 2^k)} \tag{2.5}$$

$$= \frac{2^{-t-1}\ (\times 2^k)}{\frac{1}{2}\ (\times 10^k)} \tag{2.6}$$

$$= 2^{-t} \tag{2.7}$$

$$= \frac{1}{2} \times 2^{-t+1} \tag{2.8}$$

From (2.4) and (2.8) it can be seen that the bound on the relative error is independent of the base.

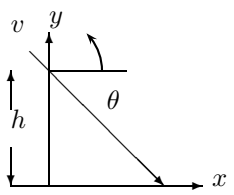


Figure 2.1: The dynamics of Table Tennis

2.2 loss of accuracy

A common reason for a serious loss of accuracy in numerical calculation is the subtraction of two nearly equal numbers.

Example 3 *At table tennis I can smash the ball at 47 ms (the world record is 47.2 ms). I claim that at that speed, gravity has no effect on the path of the ball. To verify this claim we can perform a simple calculation*[1]*.*

Using the notation in figure (2.1),

$$\begin{aligned} x = &\quad vt\cos(\theta) \\ y = &\quad h - vt\sin(\theta) - \tfrac{1}{2}gt^2 \end{aligned}$$

where $v = 47$ *ms,* $g = 981\ ms^{-2}$*,* $h = .15$ *m and* $\theta = arctan\left(\frac{15}{20}\right)$*. The ball hits the table when* $y = 0$*, ignoring* g *this gives*

$$\begin{aligned} vt\sin(\theta) &= h \\ x &= \tfrac{h}{\tan(\theta)} = 20\ cms \end{aligned}$$

incorporating g

$$0 = h - x\tan(\theta) - \frac{g}{2v^2\cos^2(\theta)}\,x^2$$

assuming that h*,* v *and* $\tan(\theta)$ *are measured to 4 figures*

$$.003469x^2 + .7500x - .1500 = 0$$

using the formula

$$x = \frac{-b \pm (b^2 - 4ac)^{1/2}}{2a}$$

[1]Example taken from OU course M351

to solve $ax^2 + bx + c = 0$ *we have, working to four figures, to find the positive root* x_1

$$\begin{aligned} x_1 &= \frac{-.7500+(.5625+.002081)^{1/2}}{.006938} \\ &= \frac{-.7500+(.5646)^{1/2}}{.006938} \\ &= \frac{-.7500+.7514}{.006938} \\ &= \frac{.0014}{.006938} \\ &= .2018 \ \textit{(metres)} \\ &> 20 \ \textit{cms} \end{aligned}$$

If this value were correct it would imply that gravity is acting upwards (to increase the distance travelled). The problem is the numerical computation of the difference $.7514 - .7500$ *both of which are correct to 4 figures but the difference only has 2 figures correct. In this example we should compute the large (negative) root* x_2 *as*

$$\begin{aligned} x_2 &= \frac{-.7500-(.5625+.002081)^{1/2}}{.006938} \\ &= \frac{-.7500-(.5646)^{1/2}}{.006938} \\ &= \frac{-.7500-.7514}{.006938} \\ &= \frac{-1.501}{.006938} \end{aligned}$$

and then compute x_1 *using* $x_1 x_2 = -\frac{c}{a}$ *as*

$$\begin{aligned} x_1 &= -\frac{c}{ax_2} \\ &= \frac{.1500}{.003469}\frac{.006938}{1.501} \\ &= \frac{.3000}{1.501} \\ &= .1999 \end{aligned}$$

i.e. *gravity reduces the distance travelled by 0.01cm.*

2.3 backward error analysis

Error analysis can be either forward or backward, in forward error analysis we try to predict the error made in the computation, in backward error analysis we show that the computed solution could have come from slight perturbations of the original data. So

- Forward analysis: bounds on computed solution of the exact problem
- Backward analysis: bounds on exact solution of a perturbed problem

An *ill-conditioned* problem contains a high degree of uncertainty, a stable algorithm applied to a well conditioned problem leads to an accurate result. The ill-conditioning in a problem is quantified by the *condition number*. The difference between ill conditioned and well conditioned systems is illustrated in figure (2.2), in both cases the dotted lines illustrate

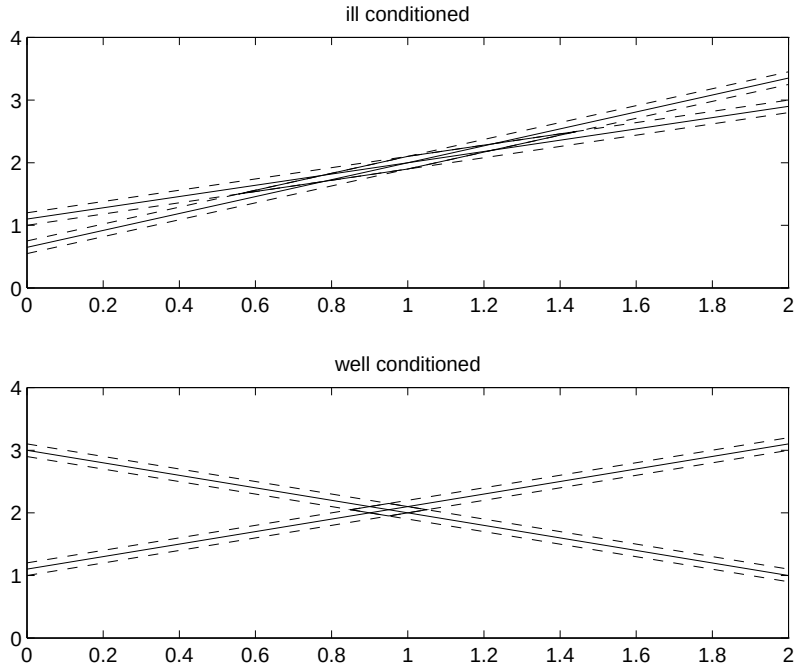


Figure 2.2: Conditioning of a pair of linear equations

$ax_1 + bx_2 = c + \epsilon$ with $|\epsilon| = 0.1$ and the central diamond contains all possible solutions with $|\epsilon| \leq 0.1$. The well conditioned problem is the pair of simultaneous equations

$$\begin{aligned} -x_1 + x_2 &= 1 \\ x_1 + x_2 &= 3 \end{aligned}$$

the ill conditioned system is

$$\begin{aligned} -x_1 + x_2 &= 1 \\ -1.2x_1 + 1.1x_2 &= 1 \end{aligned}$$

The components of the matrix in a linear system may not be known exactly, they may be measured to a finite precision or the results of numerical computation and hence subject to round off error. In either case the true problem

$$A\boldsymbol{x} = \boldsymbol{b}$$

is replaced by a *perturbed system*

$$(A + \delta A)\tilde{\boldsymbol{x}} = \boldsymbol{b} + \boldsymbol{\delta b}.$$

Assuming that A is nonsingular if $\|A^{-1}\delta A\| < 1$ then $A+\delta A$ is nonsingular and the perturbed system has a unique solution. Since $\|A^{-1}\delta A\| < \|A^{-1}\|\|\delta A\|$ a similar, but weaker, result can be stated if $\|A^{-1}\|\|\delta A\| < 1$. In particular, if $\boldsymbol{\delta b} = 0$ and $\frac{\|\delta A\|}{\|A\|}$ is small then the following inequality is *almost true*

$$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \kappa(A)\frac{\|\delta A\|}{\|A\|} \tag{2.9}$$

This suggests that the relative perturbations in A are scaled by the condition number to give the relative perturbations in $\boldsymbol{x}$. This is the standard use of the term condition number, that is the ratio (or a bound on the ratio) of the relative error output to the relative error in input.

Exercise 2.1 *Using the properties of norms, with* $\boldsymbol{\delta b} = 0$

1. $$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\tilde{\boldsymbol{x}}\|} \leq \|A^{-1}\delta A\|$$

2. $$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\|A^{-1}\delta A\|}{1 - \|A^{-1}\delta A\|}$$

3. $$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(A)\frac{\|\delta A\|}{\|A\|}}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}}$$

Chapter 3

Numerical Solution of simultaneous equations

3.1 Introduction

Assuming that $A\boldsymbol{x} = \boldsymbol{b}$ has a unique solution:

- Interchanging the rows of A (together with the rows of $\boldsymbol{b}$) does not change the solution.
- Replacing a row by a linear combination of it with another row does not alter the solution

If

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \tag{3.1}$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \tag{3.2}$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \tag{3.3}$$

for any α, β define a linear combination of (3.1) and (3.2) in the form

$$(\alpha a_{21} + \beta a_{11})x_1 + (\alpha a_{22} + \beta a_{12})x_2 + (\alpha a_{23} + \beta a_{13})x_3 = \alpha b_2 + \beta b_1 \tag{3.4}$$

then if $|\beta| \neq 0$ (3.1), (3.2), (3.3) and (3.1), (3.4), (3.3) are equivalent systems, if $|\alpha| \neq 0$ then (3.1), (3.2), (3.3) and (3.4), (3.2), (3.3) are equivalent systems.

Using these two rules *systematically* we can eliminate variables from the equations to derive a sequence of equivalent systems that are progressively easier to solve.

exmpl

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &= 2 \\ 2x_1 + 2x_2 - 2x_3 &= 2 \\ x_1 + 5x_2 - 8x_3 &= 6 \end{aligned} \tag{3.5}$$

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &= 2 \\ -2x_2 + 4x_3 &= -2 \\ 3x_2 - 5x_3 &= 4 \end{aligned} \tag{3.6}$$

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &= 2 \\ -2x_2 + 4x_3 &= -2 \\ x_3 &= 1 \end{aligned} \tag{3.7}$$

All have the solution

$$x_1 = -1, x_2 = 3, x_3 = 1$$

To transform (3.5) into (3.6) subtract multiples of row 1 from rows 2 and 3. To transform (3.6) into (3.7) subtract a multiple of row 2 from row 3. To obtain the solution of the *upper triangular* system (3.7), solve the third equation for x_3, substitute this value into the second equation and solve for x_2. Thus

$$\begin{aligned} x_3 &= 1 \\ (-2x_2 &= -2 - 4x_3 = -6) \\ x_2 &= 3 \\ (x_1 &= 2 - 2x_2 + 3x_3) \\ x_1 &= -1 \end{aligned}$$

In this example we have a sequence of linear systems

$$A^{(1)}\boldsymbol{x} = \boldsymbol{b}^{(1)} \qquad A^{(1)} \equiv A, \boldsymbol{b}^{(1)} \equiv \boldsymbol{b}$$

$$A^{(2)}\boldsymbol{x} = \boldsymbol{b}^{(2)}$$

$$A^{(3)}\boldsymbol{x} = \boldsymbol{b}^{(3)}$$

where, in this example,

$$A^{(1)} = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 2 & -2 \\ 1 & 5 & -8 \end{pmatrix} \qquad \boldsymbol{b}^{(1)} = \begin{pmatrix} 2 \\ 2 \\ 6 \end{pmatrix}$$

$$A^{(2)} = \begin{pmatrix} 1 & 2 & -3 \\ 0 & -2 & 4 \\ 0 & 3 & -5 \end{pmatrix} \qquad \boldsymbol{b}^{(2)} = \begin{pmatrix} 2 \\ -2 \\ 4 \end{pmatrix}$$

Note that

$$A^{(1)} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & 0 & 1 \end{pmatrix} A^{(2)} \qquad \text{and} \qquad A^{(2)} = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -1 & 0 & 1 \end{pmatrix} A^{(1)} \tag{3.8}$$

The gaps in the matrices linking $A^{(1)}$ and $A^{(2)}$ signify zeros, hence the matrices are *lower triangular*. The transformation from $A^{(1)}$ to $A^{(2)}$ is therefore equivalent to multiplication by a lower triangular matrix.

Continuing the example

$$A^{(3)} = \begin{pmatrix} 1 & 2 & -3 \\ 0 & -2 & 4 \\ 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{b}^{(3)} = \begin{pmatrix} 2 \\ -2 \\ 1 \end{pmatrix}$$

with

$$A^{(2)} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -\frac{3}{2} & 1 \end{pmatrix} A^{(3)} \qquad \text{and} \qquad A^{(3)} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & \frac{3}{2} & 1 \end{pmatrix} A^{(2)}$$

These matrices

$$L^{(1)} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & 0 & 1 \end{pmatrix} \quad \text{and} \quad L^{(2)} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -\frac{3}{2} & 1 \end{pmatrix}$$

are known as *elementary matrices* or *elementary transformations*. In both cases the coefficients in the lower triangular matrices are the multipliers used in the elimination. The final system has an upper triangular matrix $A^{(3)}$ where

$$A^{(3)} = L^{(2)}L^{(1)}A^{(1)} \quad \Rightarrow \quad A^{(1)} = {L^{(1)}}^{-1}{L^{(2)}}^{-1}A^{(3)}$$

$$\begin{aligned} A^{(1)} &= \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & 0 & 1 \end{pmatrix} A^{(2)} \\ &= \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -\frac{3}{2} & 1 \end{pmatrix} A^{(3)} \\ &= \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & -\frac{3}{2} & 1 \end{pmatrix} A^{(3)} \end{aligned}$$

Note $\begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -\frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & 0 & 1 \end{pmatrix} \neq \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & -\frac{3}{2} & 1 \end{pmatrix}$ and the order of the factors is important.

That is, a by-product of the elimination process to solve $A\boldsymbol{x} = \boldsymbol{b}$ has been to generate a lower triangular matrix which we denote by L, in terms of the elementary matrices

$$L = {L^{(1)}}^{-1}{L^{(2)}}^{-1}$$

is this example

$$L = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 1 & -\frac{3}{2} & 1 \end{pmatrix}$$

and an upper triangular matrix, usually denoted by U but sometimes by R such that $A = LU$, in this example

$$U = \begin{pmatrix} 1 & 2 & -3 \\ & -2 & 4 \\ & & 1 \end{pmatrix}$$

Exercise 3.1

1. *Verify that the elementary matrix $L^{(j)}$, that eliminates column j, can be written as*

$$L^{(j)} = I - \boldsymbol{v}_j \boldsymbol{e}_j^T$$

where $\boldsymbol{e}_j$ is the j-th unit vector and $\boldsymbol{v}_j$ is a vector such that $\boldsymbol{v}_j^T \boldsymbol{e}_k = 0$ for all $k \leq j$.

2. *Prove that ${L^{(j)}}^{-1} = I + \boldsymbol{v}_j \boldsymbol{e}_j^T$.*

3. *Prove that if $i \leq j$ then*

$$L^{(i)} L^{(j)} = I - \boldsymbol{v}_i \boldsymbol{e}_i^T - \boldsymbol{v}_j \boldsymbol{e}_j^T$$

and hence that if $U = L^{(n-1)} \cdots L^{(2)} L^{(1)} A$ then

$$L = I + \sum_{l=1}^{n-1} \boldsymbol{v}_l \boldsymbol{e}_l^T.$$

3.2 elimination algorithm

The basic form of Gauss Elimination is given in figure 3.1. At the k-th stage, we say that the

```
do k = 1 : N − 1
    do i = k + 1 : N
        m_ik = a_ik / a_kk
        do j = k + 1 : N
            a_ij = a_ij − m_ik a_kj
        enddo j
        b_i = b_i − m_ik b_k
    enddo i
enddo k
```

Figure 3.1: Gauss Elimination

term $a_{kk}^{(k)}$ is the *pivot*. The *computational cost* can be determined by the number of `flops` = "floating point operations" needed for the computation

$$\begin{aligned} \text{1 operation} &= \text{1 multiplication} \\ &= \text{1 addition} \\ &= \text{1 division} \end{aligned}$$

Ignoring the operations on the vector $\boldsymbol{b}$, the computational cost, in terms of all arithmetic

operations, is

$$\sum_{k=1}^{N}\sum_{i=k+1}^{N}\left((1 \text{ division}) + \sum_{j=k+1}^{N}(1 \text{ multiplication} + \ 1 \text{ addition})\right) \tag{3.9}$$

there are many more multiplications and additions than there are divisions, so those terms dominate. There are usually (almost) the same number of multiplications as additions. From (3.9), the number of multiplications (or additions) is

$$\begin{aligned}\sum_{k=1}^{N-1}\sum_{i=k+1}^{N}(N-k) &= \sum_{k=1}^{N-1}(N-k)^2\\ &= \sum_{k=1}^{N-1}k^2\\ &= \tfrac{N(N-1)(2N-1)}{6}\\ &= \tfrac{N^3}{3}-\tfrac{N^2}{2}+\tfrac{N}{6}\end{aligned}$$

For large N the $\frac{N^3}{3}$ term dominates and we say that the computational cost (*work* or *computer time*) is proportional to N^3 which is written as $\mathcal{O}(N^3)$. The additional cost of the operations involving components of $\boldsymbol{b}$ can be written as

$$\sum_{i=1}^{N-1}\left(\sum_{j=i+1}^{N}(1 \text{ multiplication} + \ 1 \text{ addition})\right)$$

so in terms of multiplications or additions we have

$$\begin{aligned}\sum_{i=1}^{N-1}(N-i) &= \tfrac{N(N-1)}{2}\\ &= \tfrac{N^2}{2}-\tfrac{N}{2}\\ &= \mathcal{O}(N^2)\end{aligned}$$

The *backward substitution* can be written as in figure 3.2. The computational cost is

```
do  i = N : 1
    do  j = i + 1 : N
        b_i = b_i − a_ij x_j
    enddo j
    x_i = b_i / a_ii
enddo i
```

Figure 3.2: Backward Substitution

$$\sum_{i=N-1}^{1}\left((1 \text{ division}) + \sum_{j=i+1}^{N}(1 \text{ multiplication} + \ 1 \text{ addition})\right)$$

in terms of multiplications or additions, ignoring divisions, we have

$$\begin{aligned}\sum_{i=N-1}^{1}(N-i) &= \tfrac{N(N-1)}{2} \\ &= \tfrac{N^2}{2} - \tfrac{N}{2} \\ &= \mathcal{O}(N^2)\end{aligned}$$

Hence the work in the elimination dominates.

3.3 alternative forms

The elimination algorithms are not unique. The order of the loops in the algorithm can be changed so, for example, back-substitution could be reordered to access the matrix (in the innermost loop) by columns, *i.e.* the order of the loops could be reversed so the the outer loop involves the subscript j, as in figure 3.3.

```
do  j = N : 1
    x_j := b_j / a_jj
    do  i = j − 1 : 1
        b_i := b_i − a_ij x_j
    enddo i
enddo j
```

Figure 3.3: Backward Substitution by columns

An alternative form of elimination is to scale the pivot row first to give a unit diagonal. This alternative form as shown in figure 3.4 can also be written in terms of elementary matrix operations and the computational cost is unaltered. The elimination algorithm has three nested loops so there are 6 permutations of the subscripts (assuming that the form of the innermost loop is always $a_{ij} = a_{ij} \cdots$.

```
do k = 1 : N
    do j = k + 1 : N
        a_kj = a_kj / a_kk
    enddo j
    b_k = b_k / a_kk
    do j = k + 1 : N
        do i = k + 1 : N
            a_ij = a_ij − a_ik a_kj
        enddo i
        b_j = b_j − a_jk b_k
    enddo j
enddo k
```

Figure 3.4: Alternative form of GE

Exercise 3.2

1. *Verify that using the revised elimination, the elimination of column k can be written as*
$$A^{(k+1)} = L^{(k)} A^{(k)}$$
where
$$L^{(k)} = I - \boldsymbol{v}_k \boldsymbol{e}_k^T$$
where $\boldsymbol{e}_k$ is the k-th unit vector and $\boldsymbol{v}_k$ is a vector such that $\boldsymbol{v}_k^T \boldsymbol{e}_j = 0$ for all $j < k$.
2. *What is the form of $L^{(k)}$?.*
3. *Verify that this elimination corresponds to a factorisation $A = LU$ where $diag(U) = I$ and $diag(L) \neq I$*

3.4 augmented matrix notation*

We can define the $N \times (N+1)$ matrix

$$\left(\begin{array}{c|c} A & \boldsymbol{b} \end{array} \right) = \begin{pmatrix} a_{11} & \cdots & a_{1k} & \cdots & a_{1N} & b_1 \\ \vdots & & \vdots & & \vdots & \vdots \\ a_{i1} & \cdots & a_{ik} & \cdots & a_{iN} & b_i \\ \vdots & & \vdots & & \vdots & \vdots \\ a_{N1} & \cdots & a_{Nk} & \cdots & a_{NN} & b_N \end{pmatrix}$$

within which we clearly use the notation $a_{i,N+1} = b_i \;\; i = 1, \cdots, N$ and the elimination can be written as in figure 3.5.

i.e. no special treatment for the right hand side vector $\boldsymbol{b}$.

```
do k = 1 : N − 1
    do i = k + 1 : N
        m_ik = a_ik / a_kk
        do j = k + 1 : N + 1
            a_ij = a_ij − m_ik a_kj
        enddo j
    enddo i
enddo k
```

Figure 3.5: Augmented Matrix form of GE

3.5 Crout Factorisation

If we can write $A = LU$, the product of a lower triangular matrix L and and upper triangular matrix U then we can solve $A\boldsymbol{x} = \boldsymbol{b}$ *i.e.* $L(U\boldsymbol{x}) = \boldsymbol{b}$ as $L\boldsymbol{y} = \boldsymbol{b}$ then $U\boldsymbol{x} = \boldsymbol{y}$. If the matrix U has a unit diagonal while the diagonal of L varies, This is known as *Crout Factorisation*. If it is assumed that the components of the matrix A, can be overwritten by the corresponding components of L ($a_{ij} := l_{ij} \quad i > j$) and U ($a_{ij} := u_{ij} \quad i \leq j$) . The symbol := is interpreted as *is over written by* and is equivalent to an assignment statement in a computer code. This replacement could also be written as $A := (L - I) + U$, where the matrix $L - I$ has a zero diagonal, it is not necessary to store the fixed unit diagonal of the lower triangular matrix L.

3.6 Cholesky Factorisation

If the matrix A is symmetric then it is possible to construct a symmetric factorisation. If the matrix A is symmetric and positive definite *i.e.* $\boldsymbol{x}^T A\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq 0$ then there exists a factorisation of the form $A = LL^T$ this is known as a Cholesky Factorisation. If A is symmetric and nonsingular, but not necessarily positive definite, then it is possible to construct a factorisation of the form $A = LDL^T$ where $\mathrm{diag}(L) = I$ and D is a diagonal matrix. As with the LU factorisations, there exist inner-product forms, outer-product forms, row oriented forms, column oriented forms,*etc.*.

3.7 Computing an inverse*

If Gauss elimination is applied to a sequence of problems, *viz*

$$\begin{aligned} A\boldsymbol{x}_1 &= \boldsymbol{e}_1 \\ &\vdots \\ A\boldsymbol{x}_i &= \boldsymbol{e}_i \\ &\vdots \\ A\boldsymbol{x}_N &= \boldsymbol{e}_N \end{aligned}$$

where the right hand sides are the columns of the unit matrix *i.e.*

$$I = \left(\begin{array}{c|c|c|c} \boldsymbol{e}_1 & \cdots & \boldsymbol{e}_i & \cdots & \boldsymbol{e}_N \end{array} \right)$$

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \boldsymbol{e}_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow i\text{th position} \quad \cdots \quad \boldsymbol{e}_N = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

The solution vectors are then the columns of the inverse *i.e.*

$$A^{-1} = \left(\begin{array}{c|c|c|c} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_i & \cdots & \boldsymbol{x}_N \end{array} \right)$$

3.8 elimination with partial pivoting

In the elimination at the kth stage, we divide by the *pivot* a_{kk}. Clearly if $a_{kk} = 0$ the algorithm breaks down such as when

$$A = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 4 & 1 \\ -2 & 2 & 2 \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} 2 \\ -6 \\ 0 \end{pmatrix}$$

In order to avoid a zero pivot at the kth stage, we search the pivot column (column k) below the diagonal to find the component of maximum modulus (say a_{pk} then, if $p \neq k$ interchange rows p and k). At the kth stage we have a reduced matrix of the form

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1k} & \cdots & a_{1N} \\ & \ddots & \vdots & & \vdots \\ & & a_{kk} & \cdots & a_{kN} \\ & & \vdots & & \vdots \\ & & a_{pk} & \cdots & a_{pN} \\ & & \vdots & & \vdots \\ & & a_{Nk} & \cdots & a_{NN} \end{pmatrix}$$

Interchanging rows does not alter the solution of $A\boldsymbol{x} = \boldsymbol{b}$ and the search for the

$$\max \{|a_{ik}|, i = k, \cdots, N\}$$

does not require a great computational effort. A general purpose elimination algorithm should always include *partial pivoting*.

Note that the *pivot ratios* $a_{ik}, i > k$, which are also the components of the lower triangular factor L, all satisfy $|a_{ik}| \leq 1$. Provided that the row interchanges on the right hand side are carried out during the elimination (*i.e.* forward substitution) the backward substitution is unaffected by the pivoting. Gauss Elimination with pivoting can be written as a sequence of elementary matrix operations so that at the k-th stage:

$$A^{(k+1)} = L^{(k)} P^{(k)} A^{(k)}$$

```
do  k = 1 : N − 1
    find p such that |a_pk| = max_{k≤i≤N} |a_ik|
    if   p ≠ k then
         interchange a_pj with a_kj, j = k, ··· , N
         interchange b_p with b_k
    end if
    do  i = k + 1 : N
         a_ik := a_ik/a_kk
         do  j = k + 1 : N
              a_ij := a_ij − a_ik a_kj
         enddo j
         b_i := b_i − a_ik b_k
    enddo i
enddo k
```

Figure 3.6: elimination with partial pivoting

. The result of partial pivoting is to compute a factorisation such that $PA = LU$ where P is a *permutation matrix*.

Exercise 3.3

1. *Verify that elimination can be written as*

$$L^{(N-1)}P^{(N-1)}\dots L^{(1)}P^{(1)}A = U.$$

and that

$$\begin{aligned} L^{(N-1)}P^{(N-1)}\dots L^{(1)}P^{(1)} &= \tilde{L}^{(N-1)}\dots\tilde{L}^{(1)}P^{(N-1)}\dots P^{(1)} \\ &= \tilde{L}P \end{aligned}$$

where $\tilde{L}^{(j)}$ are elementary transformations.

2. *Hence show that $\tilde{L}PA = U$ so $PA = LU$ with $L = \tilde{L}^{-1}$. Verify that L is lower triangular.*

3.9 pivoting for accuracy

The main purpose of partial pivoting (row interchanges) is not to avoid the breakdown of the method because of a zero pivot but to avoid unnecessary loss of accuracy because of small pivots and large pivot ratios.

Example 4 *Using three digit arithmetic to solve* $A\boldsymbol{x} = \boldsymbol{b}$ *with*

$$A = \begin{pmatrix} -1.41 & 2 & 0 \\ 1 & -1.41 & 1 \\ 0 & 2 & -1.41 \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

without pivoting

$$L = \begin{pmatrix} 1 & & \\ -.709 & 1 & \\ 0 & 200 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} -1.41 & 2 & 0 \\ & .0100 & 1 \\ & & -201 \end{pmatrix}$$

which gives the "solution" $\boldsymbol{x} = \begin{pmatrix} .709 & 1.00 & 1.70 \end{pmatrix}^T$. *If partial pivoting is applied, then at the second stage we interchange rows 2 and 3 to give*

$$L = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ -.709 & .00500 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} -1.41 & 2 & 0 \\ & 2 & -1.41 \\ & & 1.01 \end{pmatrix}$$

which gives the solution $\boldsymbol{x} = \begin{pmatrix} 1.69 & 1.69 & 1.69 \end{pmatrix}^T$ *which is correct, rounded to 3 figures.*

Another view of the same phenomenon observes that, using 3 figure arithmetic, the matrices

$$A = \begin{pmatrix} -1.41 & 2 & 0 \\ 1 & -1.41 & 1 \\ 0 & 2 & -1.41 \end{pmatrix} \quad \textit{and} \quad A + \delta A = \begin{pmatrix} -1.41 & 2 & 0 \\ 1 & -1.41 & 1 \\ 0 & 2 & -1 \end{pmatrix}$$

lead to the same LU factors.

Chapter 4

Nonlinear Equations

Taylor's Series

In one dimension,

$$\begin{aligned} f(x+\delta) &= f(x) + \tfrac{1}{1!}\delta f'(x) + \tfrac{1}{2!}\delta^2 f''(x) + \tfrac{1}{3!}\delta^3 f'''(x) + \cdots \\ &\quad + \tfrac{1}{n!}\delta^n f^{(n)}(x) + \cdots \\ &\quad + \tfrac{1}{(n+1)!}\delta^{n+1} f^{(n+1)}(x+\theta\delta), \end{aligned}$$

for some $0 < \theta < 1$, *i.e.* the point $\xi = x + \theta\delta$ is an unknown point in the open interval $(x, x+\delta)$ (assuming $\delta > 0$). [1] In $\mathbb{R}^2$

$$\begin{aligned} f(x+\delta_x, y+\delta_y) &= f(x,y) + \tfrac{1}{1!}\left(\delta_x \tfrac{\partial}{\partial x} + \delta_y \tfrac{\partial}{\partial y}\right) f(x,y) + \cdots \\ &\quad + \tfrac{1}{2!}\left(\delta_x^2 \tfrac{\partial^2}{\partial x^2} + 2\delta_x\delta_y \tfrac{\partial^2}{\partial x \partial y} + \delta_y^2 \tfrac{\partial^2}{\partial y^2}\right) f(x,y) + \cdots \\ &\quad + \tfrac{1}{n!}\left(\delta_x^n \tfrac{\partial^n}{\partial x^n} + \cdots + \delta_y^n \tfrac{\partial^n}{\partial y^n}\right) f(x,y) + \cdots \\ &\quad + \tfrac{1}{(n+1)!}\left(\delta_x^n \tfrac{\partial^n}{\partial x^n} + \cdots + \delta_y^n \tfrac{\partial^n}{\partial y^n}\right) f(\xi,\eta), \end{aligned}$$

the point (ξ, η) is in the open set $(x, x+\delta_x) \times (y, y+\delta_y)$ (assuming $\delta_x, \delta_y > 0$).

[1] Note there are a number of different notations for derivatives, Newton (4 January 1643 31 March 1727) was responsible for the $\dot{f}(x)$ notation and the $f'(x)$ notation is due to Lagrange (January 25, 1736 April 10, 1813), while Leibniz (July 1 1646 November 14, 1716) was responsible for the $\frac{dy}{dx}$. In $\mathbb{R}^n$ there are even more notational variations, mostly of a more recent origin. Assume that $u = u(x_1, \ldots, x_n)$ then the partial derivatives w.r.t. x_i (assuming x_j, $j \neq i$ are constant) can be denoted by $\frac{\partial u}{\partial x_i}$, u_{x_i} or $\partial_{x_i} u$ and the differential operator can be denoted by $\frac{\partial}{\partial x_i}$ or ∂_{x_i}. A second derivative can be $\frac{\partial^2 u}{\partial x^2}$, u_{xx} or $\partial_{xx} u$. The operator notation is often extended to the *multi-index* notation, where for $x = (x_1, \ldots, x_n)$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $x_i \in \mathbb{R}$ and α_i non-negative integers,

$$\partial^\alpha = \frac{\partial^{\alpha_1}}{\partial {x_1}^{\alpha_1}} \cdots \frac{\partial^{\alpha_n}}{\partial {x_n}^{\alpha_n}}$$

4.1 Simple Iteration

The problem is to solve $f(x) = 0$, where f can be any nonlinear function, we denote the desired (unknown) solution by $x = x^*$. To solve $f(x) = 0$ rewrite the equation in the form

$$x = G(x)$$

Then iterate as

$$x^{(n+1)} = G(x^{(n)}) \qquad n = 0, 1, \ldots$$

assuming we have a starting value x_0. This process is an example of *iteration*, any numerical iteration reqires:

- An initial approximation, x_0
- An update formula $x^{(n+1)} = G(x^{(n)})$
- A terminating condition, *e.g.* given a small parameter ϵ (=10^{-5} say) stop when $|x^{(n+1)} - x^{(n)}| \leq \epsilon$

Example 5 *Given the equation $x - e^{-x} = 0$ we could use*

$$x = e^{-x}$$

or

$$x = 0.36x + 0.64e^{-x}$$

or

$$x = x \; - \; \frac{x - e^{-x}}{1 + e^{-x}} \qquad \textit{A special form known as Newton-Raphson (see later)}$$

in each case the equation $x = G(x)$ has the same solution as $f(x) = 0$. In each case the iteration is started with $x_0 = 1.2$ and the iteration is terminated when $|x^{(n+1)} - x^{(n)}| < 10^{-14}$. The results are given in table 4.1. Clearly some iterations work faster than others

If we take the final solution of the fastest iteration as the correct result x^*, then the errors in the iterates $e^{(n)} = x^{(n)} - x^*$, are as shown in 4.2, it can be seen that the errors in the first column (using $x = e^{-x}$) reduce by a factor -0.57, in the second column (using $x = 0.36x + 0.64e^{-x}$) by a factor -0.003 and in the third column the errors are reducing faster than linear.

4.1.1 analysis of convergence

From the *Mean Value Theorem* (M.V.T.)

$$G(x) - G(y) = (x - y)\; G'(\xi)$$

where ξ is a particular (but unknown) point between x and y so

$$G(x^{(n)}) - G(x^{n-1}) = (x^{(n)} - x^{(n-1)})\; G'(\xi_n)$$

where ξ_n is a particular (but unknown) point between $x^{(n)}$ and x^{n-1}, but as $f(x^*) = 0$, it follows that $x^* = G(x^*)$ so

$$\begin{aligned} x^{(n+1)} - x^* &= G(x^{(n)}) - G(x^*) \\ &= (x^{(n)} - x^*)\; G'(\eta_n) \end{aligned} \tag{4.1}$$

n	$x = e^{-x}$	$x = 0.36x + 0.64e^{-x}$	$x = x - \frac{x-e^{-x}}{1+e^{-x}}$
0	1.20000000000000	1.20000000000000	1.20000000000000
1	0.30119421191220	0.62476429562381	0.50924547630216
2	0.73993405478361	0.56756321480851	0.56652610854838
3	0.47714537993889	0.56714207451609	0.56714322147124
4	0.62055230687888	0.56714329402333	0.56714329040978
5	0.53764740921430	0.56714329039905	0.56714329040978
6	0.58412083447184	0.56714329040982	
7	0.55759586544701	0.56714329040978	
8	0.57258397937492	0.56714329040978	
9	0.56406601899719		
10	0.56889123231067		
	...		
57	0.56714329040978		
58	0.56714329040979		

Table 4.1: Iterates

where in general $\xi_n \neq \eta_n$ and are unknown. Assume that

$$|G'(x)| \leq m \ (< 1) \quad \text{for all } x$$

i.e. define m such that

$$m = \max_x |G'(x)| \tag{4.2}$$

then the estimate of the convergence will be a good one if $|G'| \approx m$, that is if the derivative is approximately constant. If it is not possible to find a value $m < 1$ then we cannot prove that the sequence of approximate values $x^{(n)}$, $n = 0, 1, \cdots$ converges to the solution x^*. Combining the results (4.2) and (4.1)

$$|x^{(n+1)} - x^*| \leq m|x^{(n)} - x^*| \tag{4.3}$$

If we define the *error* in $x^{(n)}$, the n-th iterate, as

$$e^{(n)} = x^{(n)} - x^*$$

n	$x = e^{-x}$	$x = 0.36x + 0.64e^{-x}$	$x = x - \frac{x-e^{-x}}{1+e^{-x}}$
0	6.32856709e-01	6.32856709e-01	6.32856709e-01
1	-2.65949078e-01	5.76210052e-02	-5.78978141e-02
2	1.72790764e-01	4.19924398e-04	-6.17181861e-04
3	-8.99979104e-02	-1.21589369e-06	-6.89385396e-08
4	5.34090164e-02	3.61354668e-09	-8.88178419e-16
5	-2.94958811e-02	-1.07384101e-11	
6	1.69775440e-02	3.18634008e-14	
7	-9.54742496e-03	-2.22044604e-16	
8	5.44068896e-03		
9	-3.07727141e-03		
10	1.74794190e-03		

Table 4.2: Errors

then we can write (4.3) as

$$|e^{(n+1)}| \leq m|e^{(n)}| \tag{4.4}$$

This assumes exact arithmetic, if we include rounding errors, we must define the actual error $\tilde{e}^{(n)}$ in terms of $\tilde{x}^{(n)}$, the computed value of the n-th iterate, this leads to

$$|\tilde{x}^{(n+1)} - x^*| \leq m|\tilde{x}^{(n)} - x^*| + |E_n|$$

or

$$|\tilde{e}^{(n+1)}| \leq m|\tilde{e}^{(n)}| + |E_n|$$

Either way the error is only reducing if $m < 1$, the closer m is to 0 the faster the convergence. In particular

$$\begin{aligned} |x^{(2)} - x^{(1)}| &\leq m|x^{(1)} - x^{(0)}| \\ |x^{(3)} - x^{(2)}| &\leq m|x^{(2)} - x^{(1)}| \leq m^2|x^{(1)} - x^{(0)}| \\ |x^{(n)} - x^{n-1}| &\leq m^{n-1}|x^{(0)} - x^{(0)}| \end{aligned}$$

so

$$\begin{aligned} |x^{(n)} - x^{(0)}| &\leq |x^{(n)} - x^{n-1}| + \ldots + |x^{(2)} - x^{(1)}| + |x^{(1)} - x^{(0)}| \\ &\leq m^{n-1}|x^{(1)} - x^{(0)}| + \ldots + m|x^{(1)} - x^{(0)}| + |x^{(1)} - x^{(0)}| \\ &= (m^{p-1} + \ldots + m + 1)|x^{(1)} - x^{(0)}| \end{aligned} \tag{4.5}$$

It is possible to sum the power series for

$$m^{n-1} + \cdots + m + 1 = \frac{1 - m^n}{1 - m}$$

and so

$$|x^{(n)} - x^{(0)}| \leq \frac{1 - m^n}{1 - m}\ |x^{(1)} - x^{(0)}| \tag{4.6}$$

Let $n \to \infty$ then $m^n \to 0$, if $0 < m < 1$ the iteration process converges,

$$1 + m + m^2 + \cdots = \frac{1}{1 - m}$$

and it follows that as $n \to \infty$ and $x^{(n)} \to x^*$ from (4.6)

$$|x^* - x^{(0)}| \leq \frac{1}{1 - m}\ |x^{(1)} - x^{(0)}|$$

The purpose of this analysis is to estimate the unknown *errors* $e^{(n)}$ in terms of the known *displacements* (also known as *corrections*), defined as $\delta^{(n)} = x^{(n+1)} - x^{(n)}$, so in general

$$|x^* - x^{(n)}| \leq \frac{1}{1 - m}\ |x^{(n+1)} - x^{(n)}| \tag{4.7}$$

$$|e^{(n)}| \leq \frac{1}{1 - m}\ |\delta^{(n)}| \tag{4.8}$$

and

$$|x^* - x^{(n+1)}| \leq \frac{m}{1 - m}\ |x^{(n+1)} - x^{(n)}| \tag{4.9}$$

$$|e^{(n+1)}| \leq \frac{m}{1 - m}\ |\delta^{(n)}| \tag{4.10}$$

(with exact arithmetic). We can use any of either (4.7) or (4.9) to *estimate the unknown errors* $e^{(n)}$ *in terms of the known displacements* $\delta^{(n)}$. Note that with (4.7) we are using the latest value $x^{(n+1)}$ to estimate the error in the previous iterate $x^{(n)}$. If the rounding errors are included and if

$$|E_n|,\ |E^{(n+1)}|, \textit{etc.} < E$$

the absolute error bound by an analogous algebra is

$$|x^* - \tilde{x}^{(n)}| \leq \frac{1}{1-m}\ |\tilde{x}^{(n+1)} - \tilde{x}^{(n)}|\ +\ \frac{E}{1-m}$$

If $|G'| \ll 1$ the convergence is fast and if $|G'| \approx 1$ the convergence is very slow.

Example 6 *(***continued***)* *For the iteration* $x - e^{-x} = 0$*, the solution is* $x^* \approx 0.567$*, then in the simple iteration*

$$x^{(n+1)} = e^{-x^{(n)}}$$

$$G(x) = e^{-x} \qquad \textit{and} \qquad G'(x) = -e^{-x}$$

so

$$|G'(x^*)| = x^* \approx .567$$

and the error is (approximately) halved at each iteration. If we take

$$x = x + \theta(x - e^{-x})$$

and choose a constant θ *such that* $G' \approx 0$ *then*

$$\begin{aligned} G' &\approx 1 + \theta(1 + e^{-x^*}) \\ &\approx 1 + \theta(1 + 0.567) \\ &\approx 0 \end{aligned}$$

so

$$\theta = -\ \frac{1}{1.567} \approx -0.64$$

so

$$\begin{aligned} G(x) &= x - 0.64(x - e^{-x}) \\ &= 0.36x + 0.64e^{-x} \end{aligned}$$

If we allow a variable θ *such that* $G'(x) = 0$ *then* $\theta = -\frac{1}{f'} = -\frac{1}{1+e^{-x}}$ *and we have a new method, known as the Newton-Raphson method, for which the convergence to the solution is much faster.*

4.1.2 Order of convergence

If $\lim_{n\to\infty} x^{(n)} = x^*$ and there exists a constant $K \neq 0$ such that

$$\lim_{n\to\infty}\ \frac{|e^{(n+1)}|}{|e^{(n)}|^p}\ =\ K$$

then we say that the *order of convergence* is p,

- $p = 1$ linear convergence (first order) and we must have $K < 1$
- $p = 2$ quadratic convergence, no restriction on the value of K
- $p = 3$ cubic convergence

K is known as the *asymptotic error constant*. From (4.4) it follows that the simple iteration has linear convergence and the value of the asymptotic error constant is m.

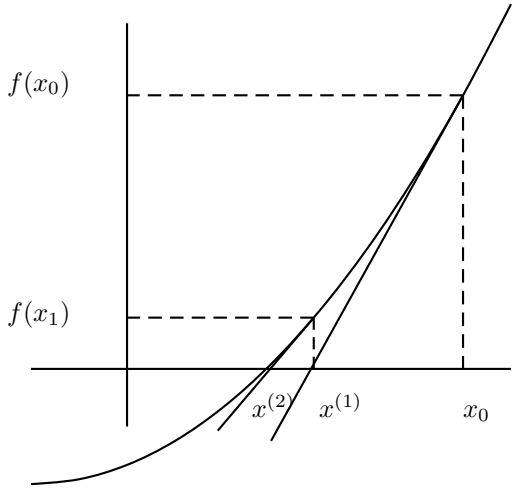


Figure 4.1: The Newton-Raphson Method

4.2 Newton-Raphson

The tangent to the curve $y = f(x)$ at any point $x = X$ is

$$y = f(X) + (x - X)f'(X)$$

this is obtained by truncating the Taylor series

$$f(x) = f(X) + (x - X)f'(X) + \dots$$

We can approximate the root $x = x^*$ where $f(x) = 0$ by the root of the tangent line, that is where

$$0 = y = f(X) + (x - X)f'(X)$$

which is

$$x = X - \frac{f(X)}{f'(X)}$$

We can use this process to generate a sequence $\{x^{(n)}\}$ as

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} \qquad n = 0, 1, 2, \dots$$

assuming we have a starting value x_0

We compute the sequence in the form, given x_0

$$\left.\begin{array}{rcl} \delta^{(n)} & = & -\dfrac{f(x^{(n)})}{f'(x^{(n)})} \\ x^{(n+1)} & = & x^{(n)} + \delta^{(n)} \end{array}\right\} \qquad n = 0, 1, 2, \dots$$

Exercise 4.1 *In molecular dynamics simulations in $\mathbb{R}^3$, the strengths of the interactions depend on the inverse of the distance between particles, thus as the simulation proceeds, there are a great many computations of the form $d^{-1/2}$. The value of d can be determined quickly, but on some modern processors used in parallel machines the accurate inverse square root is comparatively slow, although it is possible to determine an approximate solution. Construct a Newton-Raphson iteration for finding the inverse square root. Use you algorithm to determine $130^{-1/2}$ given the solution is near 0.1!*

4.3 Error Analysis

Write

$$\tilde{e}^{(n)} = \tilde{x}^{(n)} - x^*$$

as the error in the computed n-th iterate, write E_n as the error in computing the update

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$

so

$$\tilde{x}^{(n+1)} = \tilde{x}^{(n)} - \frac{f'(\tilde{x}^{(n)})}{f(\tilde{x}^{(n)})} + E_n$$

From Taylor's series

$$\begin{aligned} f(\tilde{x}^{(n)}) &= f(x^*) + (\tilde{x}^{(n)} - x^*)f'(x^*) + \tfrac{1}{2}(\tilde{x}^{(n)} - x^*)^2 f''(x^*) + \dots \\ &= 0 + \tilde{e}^{(n)} f'(x^*) + {\tilde{e}^{(n)}}^2 \tfrac{1}{2} f''(x^*) + \dots \\ f'(\tilde{x}^{(n)}) &= f'(x^*) + (\tilde{x}^{(n)} - x^*)f''(x^*) + \tfrac{1}{2}(\tilde{x}^{(n)} - x^*)^2 f'''(x^*) + \dots \end{aligned}$$

If $\tilde{x}^{(n)}$ is close to x^* then $f'(\tilde{x}^{(n)}) \approx f'(x^*)$ so (provided that $f'(x^*) \neq 0$)

$$\begin{aligned} x^{(n+1)} &= x^{(n)} - \frac{e^{(n)} f'(x^*) + {e^{(n)}}^2 \frac{1}{2} f''(x^*) + \dots}{f'(x^*) + e^{(n)} f''(x^*) + \dots} \\ e^{(n+1)} &= e^{(n)} - \frac{e^{(n)} f'(x^*) + {e^{(n)}}^2 \frac{1}{2} f''(x^*) + \dots}{f'(x^*) + e^{(n)} f''(x^*) + \dots} \\ &\approx \frac{e^{(n)} \left(f'(x^*) + e^{(n)} f''(x^*)\right) - \left(e^{(n)} f'(x^*) + {e^{(n)}}^2 \frac{1}{2} f''(x^*)\right)}{f'(x^*) + e^{(n)} f''(x^*)} \\ &= {e^{(n)}}^2 \tfrac{1}{2} \tfrac{f''(x^*)}{f'(x^*)} + \mathcal{O}({e^{(n)}}^3) \end{aligned}$$

also

$$\begin{aligned} \tilde{x}^{(n+1)} &= \tilde{x}^{(n)} - \frac{e^{(n)} f'(x^*) + {e^{(n)}}^2 \frac{1}{2} f''(x^*) + \dots}{f'(x^*) + e^{(n)} f''(x^*) + \dots} + E_n \\ \tilde{e}^{(n+1)} &= \left(\frac{1}{2}\frac{f''(x^*)}{f'(x^*)}\right) {\tilde{e}^{(n)}}^2 + E_n + \mathcal{O}({\tilde{e}^{(n)}}^3) \\ &\approx K {\tilde{e}^{(n)}}^2 + E_n \end{aligned} \tag{4.11}$$

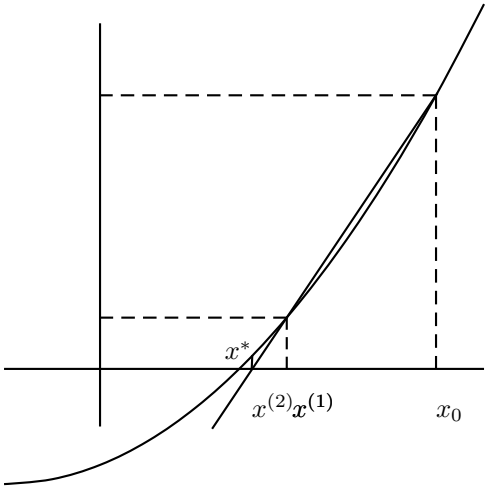


Figure 4.2: The Secant Method

Assuming that that rounding error E_n is much smaller that the truncation error we get

$$\tilde{e}^{(n+1)} \approx K\tilde{e}^{(n)^2}$$

where $K = \left(\frac{1}{2}\frac{f''(x^*)}{f'(x^*)}\right)$ is a constant and we say that the Newton-Raphson method *converges quadratically*. Note that if $\tilde{e}^{(n)} \approx 10^s$ then $\tilde{e}^{(n+1)} \approx 10^{2s}$ so that the number of digits correct (approximately) doubles at each iteration. We do not know the true solution x^* so it is necessary to estimate the errors using the differences between the iterates. If we assume that the iteration converges then from (4.11) it follows that $|e^{(n+1)}| \ll |e^{(n)}|$ and so

$$\begin{aligned} |e^{(n)}| = |x^{(n)} - x^*| &= |x^{(n)} - x^{(n+1)} + x^{(n+1)} - x^*| \\ &< |x^{(n)} - x^{(n+1)}| + |x^{(n+1)} - x^*| \\ &\approx |x^{(n)} - x^{(n+1)}| = |\delta^{(n)}| \end{aligned} \tag{4.12}$$

Note in figure(4.1), where $f', f'' > 0$ it follows from (4.11) that $e^{(n)} > 0$. Note in (4.12) the error in $x^{(n)}$ is estimated using the more accurate approximation $x^{(n+1)}$.

4.4 The Secant Method

The Newton-Raphson method required derivatives and the function was approximated by a tangent. In the secant method the function is *interpolated* locally using two function values. So

$$p(x) = f(x^{(n)}) + (x - x^{(n)})\frac{f(x^{(n)}) - f(x^{n-1})}{x^{(n)} - x^{n-1}}$$

i.e. we use the replacement

$$f'(x^{(n)}) \approx \frac{f(x^{(n)}) - f(x^{n-1})}{x^{(n)} - x^{n-1}}$$

The value of $x^{(n+1)}$ is then the solution of $p(x) = 0$ so

$$x^{(n+1)} = x^{(n)} - f(x^{(n)}) \ \frac{x^{(n)} - x^{n-1}}{f(x^{(n)}) - f(x^{n-1})}$$

computed as, given x_0 and x_1,

$$\left.\begin{array}{rcl} \delta^{(n)} & = & -f(x^{(n)}) \ \dfrac{x^{(n)} - x^{n-1}}{f(x^{(n)}) - f(x^{n-1})} \\ x^{(n+1)} & = & x^{(n)} + \delta^{(n)} \end{array}\right\} \qquad n = 1, 2, \ldots$$

If we *assume that the method converges* and that near the solution $f'(x^{(n)}) \approx f'(x^*)$ then we can generate estimates for the error $e^{(n)} = x^{(n)} - x^*$ we can rewrite the secant update as

$$\begin{aligned} x^{(n+1)} &= x^{(n)} - f(x^{(n)}) \ \frac{x^{(n)} - x^{n-1}}{f(x^{(n)}) - f(x^{n-1})} \\ &= \frac{x^{n-1} f(x^{(n)}) - x^{(n)} f(x^{n-1})}{f(x^{(n)}) - f(x^{n-1})} \end{aligned}$$

then

$$\begin{aligned} e^{(n+1)} &= x^{(n+1)} - x^* \\ &= \frac{x^{n-1} f(x^{(n)}) - x^{(n)} f(x^{n-1})}{f(x^{(n)}) - f(x^{n-1})} - x^* \ \frac{f(x^{(n)}) - f(x^{n-1})}{f(x^{(n)}) - f(x^{n-1})} \\ &= \frac{e^{n-1} f(x^{(n)}) - e^{(n)} f(x^{n-1})}{f(x^{(n)}) - f(x^{n-1})} \end{aligned}$$

but

$$\begin{aligned} f(x^{(n)}) &= f(x^*) + e^{(n)} f'(x^*) + \frac{{e^{(n)}}^2}{2} f''(x^*) + \ldots \\ f(x^{n-1}) &= f(x^*) + e^{n-1} f'(x^*) + \frac{{e^{n-1}}^2}{2} f''(x^*) + \ldots \end{aligned}$$

where $f(x^*) = 0$ so

$$\begin{aligned} e^{(n+1)} &= \frac{(e^{n-1} {e^{(n)}}^2 - e^{(n)} {e^{n-1}}^2) \frac{1}{2} f''(x^*) + \ldots}{(e^{(n)} - e^{n-1}) f'(x^*) + \ldots} \\ &\approx e^{(n)} e^{n-1} \tfrac{1}{2} \ \tfrac{f''(x^*)}{f'(x^*)} \end{aligned}$$

For Newton-Raphson we had

$$\begin{aligned} e^{(n+1)} &= e^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} \\ &\approx e^{(n)} - \frac{f(x^*) + e^{(n)} f'(x^*) + \frac{{e^{(n)}}^2}{2} f''(x^*)}{f'(x^*)} \\ &\approx {e^{(n)}}^2 \ \tfrac{1}{2} \ \tfrac{f''(x^*)}{f'(x^*)} \end{aligned}$$

If we assume that the secant method is order then then

$$e^{(n+1)} \approx K e^{(n)p} \tag{4.13}$$

but we have

$$e^{(n+1)} \approx k e^{(n)} e^{n-1} \tag{4.14}$$

from (4.13)

$$e^{(n+1)} \approx K \left(K e^{n-1p} \right)^p$$

and from (4.14)

$$e^{(n+1)} \approx k \left(K e^{n-1p} \right) e^{n-1}$$

comparing coefficients gives

$$p^2 = p + 1$$

and

$$K^{p+1} = kK$$

so

$$p = \frac{1 + \sqrt{5}}{2}$$

Again we can estimate the unknown errors by writing

$$\begin{aligned} |e^{(n)}| = |x^{(n)} - x^*| &= |x^{(n)} - x^{(n+1)} + x^{(n+1)} - x^*| \\ &< |x^{(n)} - x^{(n+1)}| + |x^{(n+1)} - x^*| \\ &\approx |x^{(n)} - x^{(n+1)}| = |\delta^{(n)}| \end{aligned}$$

assuming once again that $|e^{(n+1)}| \ll |e^{(n)}|$ and using the value of $x^{(n+1)}$ in the estimate of the error in $x^{(n)}$.

4.5 Nonlinear Systems

To solve the coupled pair of equations

$$\begin{aligned} f(x, y) &= 0 \\ h(x, y) &= 0 \end{aligned} \tag{4.15}$$

where we denote the desired solution by $x = x^*$, $y = y^*$.

4.5.1 Simple Iteration

The system (4.15) must be rewritten in the form

$$\begin{aligned} x &= F(x, y) \\ y &= H(x, y) \end{aligned} \tag{4.16}$$

Then given a pair of initial approximations $x^{(0)}$ and $y^{(0)}$ we iterate using (4.16) as

$$\begin{aligned} x^{(n+1)} &= F(x^{(n)}, y^{(n)}) \\ y^{(n+1)} &= H(x^{(n)}, y^{(n)}) \end{aligned} \qquad n = 0, 1, \cdots \tag{4.17}$$

In matrix notation if we write

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{and} \quad \boldsymbol{G}(\boldsymbol{x}) = \begin{pmatrix} F(x,y) \\ H(x,y) \end{pmatrix}$$

we have the iteration is

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{G}(\boldsymbol{x}^{(n)}) \tag{4.18}$$

For a single equation $x^{(n+1)} = G(x^{(n)})$ the iteration converges if $|\frac{dG}{dx}| < 1$, we need an equivalent condition for the iteration (4.18). The Taylor series in two variables gives:

$$F(x,y) = F(x^*,y^*) + (x-x^*)\frac{\partial}{\partial x}F(x^*,y^*) + (y-y^*)\frac{\partial}{\partial y}F(x^*,y^*) + \cdots \tag{4.19}$$

since $x^* = F(x^*,y^*)$ it follows setting $x = x^{(n)}$ and $y = y^{(n)}$ in (4.17) using (4.19) that

$$\begin{aligned} x^{(n+1)} &= x^* + (x^{(n)}-x^*)\frac{\partial}{\partial x}F(x^*,y^*) + (y^{(n)}-y^*)\frac{\partial}{\partial y}F(x^*,y^*) + \cdots \\ x^{(n+1)} - x^* &\approx (x^{(n)}-x^*)\frac{\partial}{\partial x}F(x^*,y^*) + (y^{(n)}-y^*)\frac{\partial}{\partial y}F(x^*,y^*) \end{aligned} \tag{4.20}$$

similarly

$$y^{(n+1)} - y^* \approx (x^{(n)}-x^*)\frac{\partial}{\partial x}H(x^*,y^*) + (y^{(n)}-y^*)\frac{\partial}{\partial y}H(x^*,y^*) \tag{4.21}$$

Writing (4.20) and (4.21) gives

$$\begin{pmatrix} x^{(n+1)} - x^* \\ y^{(n+1)} - y^* \end{pmatrix} \approx \begin{pmatrix} \frac{\partial}{\partial x}F(x^*,y^*) & \frac{\partial}{\partial y}F(x^*,y^*) \\ \frac{\partial}{\partial x}H(x^*,y^*) & \frac{\partial}{\partial y}H(x^*,y^*) \end{pmatrix} \begin{pmatrix} x^{(n)} - x^* \\ y^{(n)} - y^* \end{pmatrix} \tag{4.22}$$

this is equivalent to the single variable error equation

$$x^{(n+1)} - x^* \approx \frac{d}{dx}G(x^*)(x^{(n)} - x^*) \tag{4.23}$$

In order to provide an error bound (4.23) was replaced by

$$|x^{(n+1)} - x^*| \leq m|x^{(n)} - x^*| \tag{4.24}$$

where $m = \max_x |\frac{dG}{dx}|$. To construct an analogous bound based on (4.22) it is necessary to use matrix norms so that if the matrix error equation is

$$\boldsymbol{e}^{(n+1)} = A\boldsymbol{e}^{(n)}$$

then

$$\|\boldsymbol{e}^{(n+1)}\| \leq \|A\| \, \|\boldsymbol{e}^{(n)}\|$$

From (4.22), the matrices are

$$A = \begin{pmatrix} \frac{\partial}{\partial x}F(x,y) & \frac{\partial}{\partial y}F(x,y) \\ \frac{\partial}{\partial x}H(x,y) & \frac{\partial}{\partial y}H(x,y) \end{pmatrix}$$

with

$$\boldsymbol{e}^{(n+1)} = \begin{pmatrix} x^{(n+1)} - x^* \\ y^{(n+1)} - y^* \end{pmatrix} \quad \text{and} \quad \boldsymbol{e}^{(n)} = \begin{pmatrix} x^{(n)} - x^* \\ y^{(n)} - y^* \end{pmatrix}$$

It then follows that the method converges if

$$m = \max_{x,y} \|A\| \ < 1$$

where x and y are evaluated "near" the solution $x = x^*,\ y = y^*$. If we define the displacement as

$$\boldsymbol{\delta}^{(n)} = \begin{pmatrix} x^{(n+1)} - x^{(n)} \\ y^{(n+1)} - y^{(n)} \end{pmatrix}$$

we can estimate the error norm $\|\boldsymbol{e}^{(n)}\|$ in terms of the displacement norm $\|\boldsymbol{\delta}^{(n)}\|$ as for (4.7).

4.5.2 Newton-Raphson Method

To find the root of a single equation $f(x) = 0$, the Newton-Raphson method replaced the function $f(x)$ by a linear approximation and at each iteration found the root of the tangent to the curve $y = f(x)$. For a pair of equations the Taylor series, at $x = X$, $y = Y$ are

$$\begin{aligned} f(x,y) &= f(X,Y) + (x-X)\tfrac{\partial}{\partial x} f(X,Y) + (y-Y)\tfrac{\partial}{\partial y} f(X,Y) + \cdots \\ h(x,y) &= h(X,Y) + (x-X)\tfrac{\partial}{\partial x} h(X,Y) + (y-Y)\tfrac{\partial}{\partial y} h(X,Y) + \cdots \end{aligned}$$

We can approximate the root (x^*, y^*) where $f(x,y) = 0$ and $h(x,y) = 0$ by the root of the local linear approximations (tangents), that is where

$$\begin{aligned} 0 &= f(X,Y) + (x-X)\tfrac{\partial}{\partial x} f(X,Y) + (y-Y)\tfrac{\partial}{\partial y} f(X,Y) \\ 0 &= h(X,Y) + (x-X)\tfrac{\partial}{\partial x} h(X,Y) + (y-Y)\tfrac{\partial}{\partial y} h(X,Y) \end{aligned}$$

which can be written in matrix notation as

$$0 = \boldsymbol{f}(\boldsymbol{X}) + J(\boldsymbol{X})(\boldsymbol{x} - \boldsymbol{X})$$

where $\boldsymbol{x} = (x,y)^T$, $\boldsymbol{X} = (X,Y)^T$, $\boldsymbol{f}(\boldsymbol{X}) = (f(X,Y), h(X,Y))^T$ and

$$J(\boldsymbol{X}) = \begin{pmatrix} \frac{\partial}{\partial x} f(X,Y) & \frac{\partial}{\partial y} f(X,Y) \\ \\ \frac{\partial}{\partial x} h(X,Y) & \frac{\partial}{\partial y} h(X,Y) \end{pmatrix}$$

The matrix of partial derivatives, denoted by $J(\boldsymbol{X})$ is known as the *Jacobian Matrix*. The improved approximation to the solution can then be *written as*

$$\boldsymbol{x} = \boldsymbol{X} - J(\boldsymbol{X})^{-1} \boldsymbol{f}(\boldsymbol{X})$$

So if $\boldsymbol{X} = \boldsymbol{x}^{(0)}$, the initial approximation, then the improved approximation $\boldsymbol{x}^{(1)}$ is given by

$$J(\boldsymbol{x}^{(0)})(\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}) = -\boldsymbol{f}(\boldsymbol{x}^{(0)})$$

or

$$\begin{aligned} J(\boldsymbol{x}^{(0)})\boldsymbol{\delta}^{(0)} &= -\boldsymbol{f}(\boldsymbol{x}^{(0)}) \quad \text{Solve for } \boldsymbol{\delta}^{(0)} \\ \\ \boldsymbol{x}^{(1)} &= \boldsymbol{x}^{(0)} + \boldsymbol{\delta}^{(0)} \end{aligned}$$

so in general the Newton-Raphson method can be written as

$$\left.\begin{aligned} J(\boldsymbol{x}^{(n)})\boldsymbol{\delta}^{(n)} &= -\boldsymbol{f}(\boldsymbol{x}^{(n)}) \quad \text{Solve for } \boldsymbol{\delta}^{(n)} \\ \\ \boldsymbol{x}^{(n+1)} &= \boldsymbol{x}^{(n)} + \boldsymbol{\delta}^{(n)} \end{aligned}\right\} \qquad n = 0, 1, 2, \ldots$$

where $\boldsymbol{x}^{(n)} = \begin{pmatrix} x^{(n)} \\ y^{(n)} \end{pmatrix}$, $\boldsymbol{f}(\boldsymbol{x}^{(n)}) = \begin{pmatrix} f(x^{(n)}, y^{(n)}) \\ h(x^{(n)}, y^{(n)}) \end{pmatrix}$, *etc.*. This is the solution of a system of simultaneous linear equations and so the solution should be by an efficient technique such as Gauss Elimination and not be explicit matrix inversion. Newton-Raphson converges quadratically for systems so that

$$\|e^{(n+1)}\| = \mathcal{O}\left(\|e^{(n)}\|^2\right)$$

Chapter 5

Numerical Solution of Ordinary Differential Equations

5.1 Introduction

A typical differential equation has infinitely many solutions, we apply a numerical method to find the solution of a *well posed problem* with a *unique solution*. In figure 5.1 for example a few of the solution curves for the simple differential equation $\frac{dy}{dt} = t + y$ are illustrated. We frequently use the notation $y' \equiv \frac{dy}{dt}$. The first type of problem we consider is an *initial*

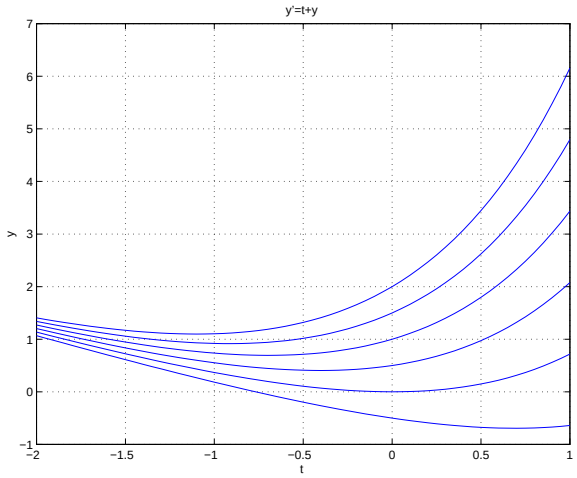


Figure 5.1: Solutions of $\frac{dy}{dt} = t + y$

value problem that consists of a differential equation together with additional conditions that restrict the solution set to a single function. For example:

- *First order* problems:

$$\frac{dy}{dt} = F(t, y) \qquad t \geq a \qquad y(a) = c$$

$y(a) = c$ is the initial condition. Hence in figure 5.1 we might specify the initial condition $y(0) = 1$, then there is exactly one solution curve.

- *Second order* problems:

$$\frac{d^2w}{dt^2} = f(t, w, w') \qquad t \geq a \qquad \begin{cases} w(a) = c \\ w'(a) = d \end{cases} \tag{5.1}$$

usually we write $y'' \equiv \frac{d^2y}{dt^2}$

- *Systems of first order* problems:

$$\left.\begin{aligned} \frac{dy}{dt} &= f(t, w, z) \\ \frac{dz}{dt} &= g(t, w, z) \end{aligned}\right\} \qquad t \geq a \qquad \begin{cases} w(a) = c \\ z(a) = d \end{cases} \tag{5.2}$$

We shall not consider *partial differential equations* such as

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

in this chapter.

Theorem 1 *Given to differential equation*

$$y' = F(t, y), \qquad t > t_0$$

such that $y(t_0) = y_0$
Assume that the function F *is* Lipschitz continuous*, that is there exists a constant* $L > 0$ *such that*

$$|F(t, y) - F(t, \tilde{y})| \leq L|y - \tilde{y}|$$

then the two solutions y *and* $\tilde{y}$ *subject to the initial conditions* $y(t_0) = y_0$ *and* $\tilde{y}(t_0) = \tilde{y}_0$ *satisfy*

$$|y(t) - \tilde{y}(t)| \leq e^{L(t - t_0)}|y_0 - \tilde{y}_0|$$

and it can be shown that a unique solution exists.

5.2 Numerical methods for 1st order systems

All the methods discussed will be defined for *scalar problems* of the form

$$y' = F(t, y) \qquad t \geq a \qquad y(a) = c$$

but can be applied to 1st order systems written in matrix notation as

$$\boldsymbol{y}' = \boldsymbol{F}(t, \boldsymbol{y}) \qquad t \geq a \qquad \boldsymbol{y}(a) = \boldsymbol{c}$$

where for a system written as (5.2),

$$\boldsymbol{y} = \begin{pmatrix} w \\ z \end{pmatrix} \quad \boldsymbol{y}' = \begin{pmatrix} w' \\ z' \end{pmatrix} \quad \boldsymbol{F}(t, \boldsymbol{y}) = \begin{pmatrix} f(t, w, z) \\ g(t, w, z) \end{pmatrix} \quad \boldsymbol{c} = \begin{pmatrix} c \\ d \end{pmatrix}$$

Higher order equations such as (5.1), can be written as first order systems. For example if we define $z = y'$ (so that $y'' = z'$) then

$$\boldsymbol{y} = \begin{pmatrix} w \\ w' \end{pmatrix} \quad \boldsymbol{y}' = \begin{pmatrix} w' \\ w'' \end{pmatrix} \quad \boldsymbol{F}(x, \boldsymbol{y}) = \begin{pmatrix} z \\ f(t, w, z) \end{pmatrix} \quad \boldsymbol{c} = \begin{pmatrix} c \\ d \end{pmatrix}$$

So a numerical method defined for a scalar 1st order equation can be used for a 1st order system and hence for higher order problems (and even systems of higher order problems).

The only methods considered for initial value problems are *step-by-step* methods. The original problem in terms of a continuous function $y(t)$ for all $t \geq a$, is replaced by a numerical problem in terms of a sequence of *discrete values* Y_n at the mesh points $a = t_0 < t_1 < t_2 < \ldots$ such that $Y_n \approx y(t_n)$. To begin with, assume that $t_{n+1} - t_n = h$, known as the *step length* is constant.

5.3 Forward, backward and central differences

A forward difference is an expression of the form

$$D_+ f(x) = f(x + h) - f(x).$$

A backward difference arises when h is replaced by $-h$:

$$D_- f(x) = f(x) - f(x - h).$$

Finally, the central difference is the average of the forward and backward differences. It is given by

$$D_0 f(x) = \frac{D_+ f(x) + D_- f(x)}{2} = \frac{f(x + h) - f(x - h)}{2}.$$

The derivative of a function f at a point x is defined by the limit

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

Assuming that f is continuously differentiable, it is possible to estimate the *truncation error* using *Taylor's series*[1]

$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f^{(2)}(a)}{2!}(x - a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x - a)^n + R_n$$

The Lagrange form of the remainder term states that there exists a number ξ between a and x such that

$$R_n = \frac{f^{(n+1)}(\xi)}{(n + 1)!}(x - a)^{n+1}.$$

Then

$$\frac{D_+ f(x)}{h} - f'(x) = \mathcal{O}(h)$$

$$\frac{D_- f(x)}{h} - f'(x) = \mathcal{O}(h)$$

[1] Brook Taylor (August 18, 1685 - November 30, 1731), although the result was first discovered in 1671 by James Gregory

and

$$\frac{D_0 f(x)}{h} - f'(x) = \mathcal{O}(h^2)$$

Similarly the central difference approximation of the second derivative of f is

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

and

$$f''(x) - \frac{D_+ D_- f(x)}{h^2} = \mathcal{O}(h^2)$$

5.4 Euler's Method

The simplest method is to replace $y'(t)$ by a *forward divided difference*, thus

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}$$

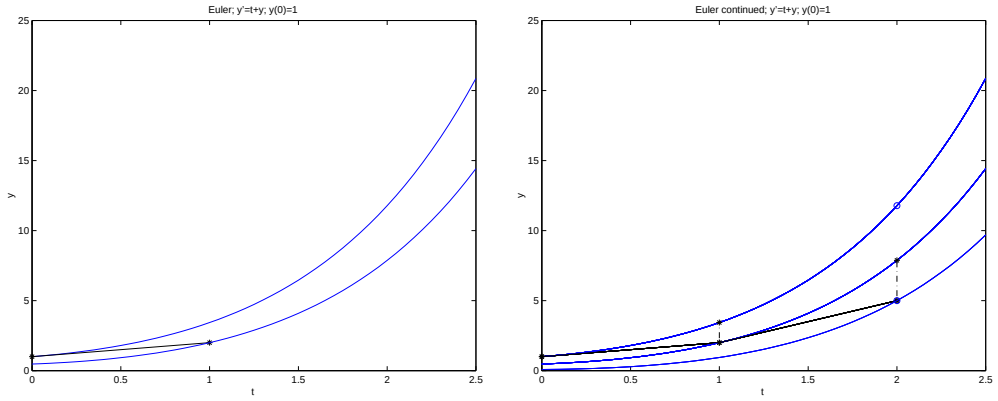
so if $t = t_n$, then $t + h = t_{n+1}$, $y(x) \approx Y_n$, $y(t+h) \approx Y_{n+1}$ and $y'(t) = F(t, y) \approx F(t_n, Y_n)$ which all leads to

$$\frac{1}{h}(Y_{n+1} - Y_n) = F(t_n, Y_n)$$

or

$$\begin{array}{rcll} Y_{n+1} & = & Y_n + hF(t_n, Y_n) & \\ t_{n+1} & = & t_n + h & \end{array} \qquad n = 0, 1, 2, \ldots$$

with $t_0 = a$ and $Y_0 = c$. Euler's method approximates the gradient $y'(t)$ by a tangent to the solution curve through $y = Y_n$ at $t = t_n$ as in figure 5.2(a), this leads to a point $y = Y_{n+1}$ at $t = t_{n+1}$ that will, in general, be on a different solution curve. The next step of Euler's method uses the tangent of this second solution curve to generate a point $y = Y_{n+2}$ at $t = t_{n+2}$. At step n there is a *local error* which is the difference between the solution curves through $y = Y_n$ at $t = t_n$ and through $y = Y_{n+1}$ evaluated at $t = t_{n+1}$. The *global error* is the combined effect of all the local errors as illustrated in figure 5.2(b).



(a) One step (b) Two steps

Figure 5.2: Euler's Method

5.5 error

To solve $y' = t + y$ subject to $y(0) = 1$ by *Euler's Method*

$$Y_{n+1} = Y_n \ + \ h\ F(t_n, Y_n) \qquad n = 0, 1, \ldots$$

where $t_n = nh$ and for this problem $F(t, y) = t + y$. The exact solution is $y(t) = 2e^t - t - 1$. The numerical solution with $h = 0.1$ is:

n	t_n	Y_n	$F(t_n, Y_n)$	$y(t_n)$
0	0.0	1.0	1.0	1.0
1	0.1	1.1	1.2	1.1103
2	0.2	1.2200	1.4200	1.2428
3	0.3	1.3620	1.6620	1.3997
4	0.4	1.5282	1.9282	1.5836
5	0.5	1.7210	2.2210	1.7974
6	0.6	1.9431	2.5431	2.0442
7	0.7	2.1974	2.8974	2.3275
8	0.8	2.4872	3.2872	2.6511

Estimate of $y(0.8)$ using the numerical solution $Y(0.8)$ with different values of h:

h	n	$Y(t)$	Error
0.1	8	2.48717762	0.16390424
0.01	80	2.63343043	0.01765142
0.001	800	2.64930297	0.00177889
0.0001	8000	2.65090383	0.00017803

We introduce the alternative notation $Y(t)$ for the numerical solution at the fixed value of t (using steps of size h) in order to emphasise the dependence on the value of t rather than the value of n the number of steps needed. In the table above, and in other examples, we consider the solution for a fixed value of t and let h get smaller. As nh is fixed if t is fixed, this means that n must increase as h decreases, but when discussing the error it is the value of h that is important.

It is possible to prove that, for Euler's method, the *global* error, $y(t) - Y(t)$, varies linearly as h, for all problems, not just for simple linear problems as in the example above. Expand $y(t_{n+1})$ by Taylor's series about t_n then

$$y(t_{n+1}) = y(t_n + h) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \ldots$$

where $y'(t_n) = F(t_n, y(t_n))$, so

$$y(t_{n+1}) = y(t_n) + hF(t_n, y(t_n)) + \frac{h^2}{2}y''(t_n) + \ldots$$

and

$$Y_{n+1} = Y_n \ + \ h\ F(t_n, Y_n) \tag{5.3}$$

To determine the *local* truncation error (l.t.e.) let $Y_n = y(t_n) = y_n$ then

$$\begin{array}{lcl} y(t_{n+1}) & = & y_n + hF(t_n, y_n) + \frac{h^2}{2}y''(t_n) + \mathcal{O}(h^3) \\ Y_{n+1} & = & y_n + hF(t_n, y_n) \end{array}$$

and so

$$y(t_{n+1}) - Y_{n+1} = \frac{h^2}{2} y''(t_n) + \mathcal{O}(h^3)$$

It is possible to truncate the power series (using the mean value theorem) and write it with a finite remainder term so that, for example,

$$y(t_{n+1}) = y(t_n) + hF(t_n, y(t_n)) + \frac{h^2}{2} y''(\xi) \tag{5.4}$$

where ξ is a particular (but unknown) point between t_{n+1} and t_n. To *estimate* the *global error* we have to assume that the solution $y(t)$ is continuous and smooth so we can assume that for some (sufficiently large) value M

$$|y''(\xi)| \ \leq \ M \qquad \text{for all } \xi$$

and that for some (sufficiently large) value L

$$|\frac{\partial}{\partial y} \ f(x,y)| \ \leq \ L \qquad \text{for all } x, y$$

The global error is defined as

$$e_n = y(t_n) - Y_n$$

and so from (5.3) and (5.4)

$$e_{n+1} = e_n + h\left(F(t_n, y(t_n)) - F(t_n, Y_n)\right) + \frac{h^2}{2} y''(\xi)$$

and then by the mean value theorem

$$e_{n+1} = e_n + h e_n \frac{\partial}{\partial y} \ f(x,\eta) + \frac{h^2}{2} y''(\xi)$$

where η is another particular (but unknown) point this time between the values of $y(t_n)$ and y_n. Hence taking absolute values and applying the smoothness assumptions

$$\begin{aligned} |e_{n+1}| \quad &\leq \quad |e_n| + hL|e_n| + \tfrac{h^2}{2} \ M \\ &= \quad (1 + hL)|e_n| + \tfrac{h^2}{2} \ M \end{aligned}$$

Using the binomial theorem we can sum the terms and, assuming that $e_0 = y(t_0) - Y_0 = 0$, we get

$$|e_{n+1}| \leq \frac{hM}{2L} \ e^{L(t_{n+1} - t_0)}$$

so that in general

$$|Y(t) - y(t)| \leq \frac{hM}{2L} \ e^{L(t - t_0)}$$

hence the error in computing to a fixed point t varies linearly with h (if the solution is sufficiently smooth). This bound involving a Lipschitz constant L is similar to the stability result in the proof of a unique solution to the initial value problem (see theorem 1).

The error is a power series in h and the leading term is h^2 so the l.t.e. is $\mathcal{O}(h^2)$ and the method is *first order* accurate. If l.t.e. is $\mathcal{O}(h^p)$ then the method is said to be accurate to order $p - 1$. The l.t.e. is the additional new error made at each step assuming that there is no error in the solution so far.

Definition 2 *Truncation Error: The truncation error is defined by substituting the solution of the differential equation $y' = F(t, y)$ into the difference equation $Y_{n+1} = F_k(t_n, Y_n)$ then the approximation is* accurate of order q *if*

$$\begin{cases} |y(t_{n+1}) - F_k(t_n, y(t_n))| & \leq & d(t_n)h^{q+1}, \quad t_n = nh, \\ |y(t_j) - Y_j| & \leq & c_j h^q. \end{cases} \tag{5.5}$$

If $q > 0$ the approximation is said to be consistent.

5.6 Stability

Example 7 *Instability: Given*

$$y'(t) = F(t, y(t))$$

it follows that

$$\int_a^b y'(t)dt = \int_a^b F(t, y(t))dt$$

so

$$y(b) - y(a) = \int_a^b F(t, y(t))dt$$

We can then construct new numerical methods for ode's using quadrature formulae! For example, as we only evaluate at integer multiples of h, using the Mid-Point Rule for evaluating an integral over two step-lengths ($b = t_{n+1}$, $a = t_{n-1}$) would give:

$$y(t_{n+1}) - y(t_{n-1}) \approx 2hF(t_n, y(t_n))$$

leads to the formula

$$Y_{n+1} = Y_{n-1} \ + \ 2h \ F(t_n, Y_n) \qquad t_n = nh; \ n = 1, 2 \ldots$$

This formula is usually known as the Leapfrog Method

- Euler's Method*: l.t.e. is $\mathcal{O}(h^2)$*
- The Leapfrog Method*: l.t.e. is $\mathcal{O}(h^3)$*

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hF(t_n, y(t_n)) + \tfrac{h^2}{2}y''(t_n) + \tfrac{h^3}{6}y'''(t_n) + \cdots \\ y(t_{n-1}) &= y(t_n) - hF(t_n, y(t_n)) + \tfrac{h^2}{2}y''(t_n) - \tfrac{h^3}{6}y'''(t_n) + \cdots \\ y(t_{n+1}) &= y(t_{n-1}) + 2hF(t_n, y(t_n)) + \tfrac{h^3}{3}y'''(t_n) + \ldots \\ \textit{l.t.e.} \qquad & y(t_{n+1}) - Y_{n+1} = \mathcal{O}(h^3) \end{aligned}$$

To solve $y' = -2y + 1$ subject to $y(0) = 1$ by where $t_n = nh$ and for this problem $F(t, y) = -2y + 1$. The exact solution is $y(t) = 0.5(e^{-2t} + 1)$ with the given initial condition. As can be seem from figure 5.3, all solutions to the differential must decay, because the solution is a negative exponential. The numerical solution with the Leapfrog Method is compared

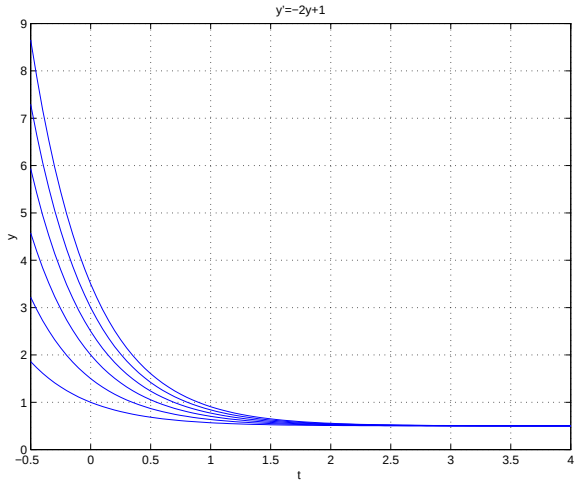


Figure 5.3: Solutions of $\frac{dy}{dt} = -2y + 1$

below with the results using the Euler method, Leapfrog is a 2-step method so it needs two initial conditions and we take the second condition from the exact solution in this example $Y_1 = y(h)$.

t_n	$y(t_n)$	*Euler*	*Leapfrog*
0	1.00000	1.00000	1.00000
0.03125	0.9697065	0.96875	0.9697065
0.0625	0.9412484	0.9394531	0.9412866
0.09375	0.9145145	0.9119873	0.9145456
0.125	0.8894003	0.8862380	0.8894684
0.15625	0.8658078	0.8620982	0.8658621
0.1875	0.8436446	0.8394670	0.8437357
0.21875	0.8228242	0.8182503	0.8228951
0.25	0.8032653	0.7983597	0.8033738
0.28125	0.7848914	0.7797122	0.7849734
0.3125	0.7676307	0.7622302	0.7677521
0.34375	0.7514157	0.7458408	0.7515044

The solutions seems to be developing satisfactorily, but if the computation is continued:

t_n	$y(t_n)$	*Euler*	*Leapfrog*
3.75	0.50027	0.50021	0.517434
3.78125	0.50025	0.50020	0.481999
3.8125	0.500234	0.50019	0.519684
3.84375	0.50022	0.50017	0.479539
3.875	0.50021	0.500167	0.52224
3.90625	0.50020	0.500156	0.47675
3.9375	0.50019	0.500147	0.52515
3.97825	0.500178	0.500138	0.47362
4.0	0.500168	0.500129	0.52844

The solution using the Leapfrog Method is clearly wrong and getting worse, the behaviour of the solutions is illustrated in figure 5.4. If the time step h *is reduced then the solution may*

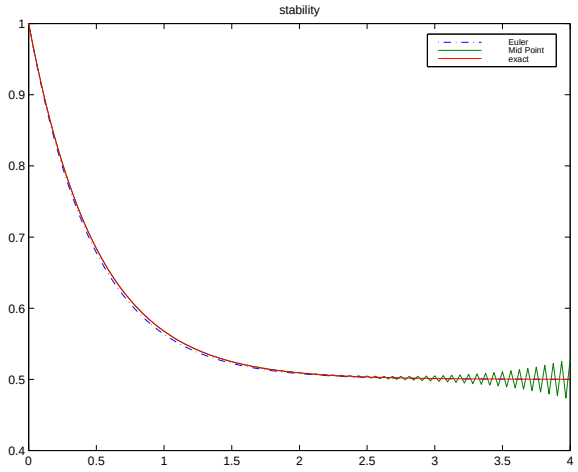


Figure 5.4: Instability of Leapfrog Method

appear better at first, but eventually the solution using Leapfrog Method will become unstable.

5.6.1 Conditional Instability

Another example is to solve $y' = -10y$ subject to $y(0) = 1$ by Euler's Method, so $F(t_n, Y_n) = -10Y_n$ and

$$Y_{n+1} = (1 \ - \ 10h)Y_n \qquad n = 0, 1, \ldots$$

with $Y_0 = 1$ this gives

$$Y_n = (1 \ - \ 10h)^n$$

and the solution is

h	0.5	0.25	0.2	0.1	0.05
Y_0	1	1	1	1	1
Y_1	-4	-1.5	-1	0	0.5
Y_2	16	2.25	1	0	0.25
Y_3	-64	-3.375	-1	0	0.125

The approximation to $y(1) = 4.5\ 10^{-05}$ for different values of h is

h	0.5	0.25	0.2	0.1	0.05
n	2	4	5	10	20
$Y(1)$	16	5.0625	-1	0	$9.8\ 10^{-04}$

The true solution is $y(t) = e^{-10t}$ but the numerical solution is only decreasing for $h \leq 0.1$!

5.6.2 Analysis of Stability

Theorem 3 *Convergence: Let y be a solution of the differential equation for which (5.5) holds, and Y be a solution of the difference approximation for which stability holds then*

$$\|y(t) - Y(t)\| = \mathcal{O}(h)^q$$

and $q > 0$ the method is convergent.

To study the stability of a numerical method, apply the method to the test equation

$$y' = -\lambda y$$

where λ is a positive constant. All solutions of this equation are decreasing exponential functions of the form

$$y(t) = Ae^{-\lambda t}$$

where A is a constant. A good numerical method should also generate solutions that are decreasing functions. A method is said to be *stable* if the solutions decrease *for all possible initial conditions*. Applying Euler's method to the test equation ($F(t, y) = -\lambda y$) we get

$$\begin{aligned} Y_{n+1} &= Y_n + h\ F(t_n, Y_n) \\ &= Y_n - h\ \lambda Y_n \\ &= (1 - \lambda h)\ Y_n \end{aligned}$$

If $|1 - \lambda h| < 1$ then $Y_{n+1} < Y_n$ and the solution is decreasing, so the method is stable if $|1 - \lambda h| < 1$ that is if

$$h\lambda < 2.$$

As λ is fixed this condition is a restriction on the step length h and is usually written as

$$h < \frac{2}{\lambda}. \tag{5.6}$$

The more rapid the decay of the solution (the greater the value of λ), the smaller the step length h. As stability imposes are condition on the step length, we say that the method is *conditionally stable*. To apply the 2-step Leapfrog Method to the test equation ($y' = -\lambda y,\ \lambda > 0$) we assume solution has form $Y_n = a^n Y_0$

$$\begin{aligned} Y_{n+1} &= Y_{n-1} + 2h\ f(t_n, Y_n) \\ a^{n+1} Y_0 &= a^{n-1} Y_0 - 2h\ \lambda a^n Y_0 \\ a^2 &= 1 - \lambda h\ a \end{aligned}$$

Solutions of $a^2 + 2\lambda h a - 1 = 0$ are $a = -\lambda h \pm \sqrt{1 + \lambda^2 h^2}$. The Leapfrog Method rule always generates an increasing solution $|a| > 1$ and is said to be unconditionally *unstable* (and therefore useless).

5.6.3 Implicit Methods

The condition for stability (5.6) can place a severe restriction of the step-size h if the λ is large, even though the solution might be smooth and the l.t.e. is very small. In order to remove this restriction on the step-length, it is necessary to condition *implicit methods*. for example the *Backward Euler Method*

$$Y_{n+1} = Y_n + hF(t_{n+1}, Y_{n+1})$$

has the same local truncation error as the (Forward) Euler Method, but in general it defines a nonlinear equation in Y_{n+1} which can only be solved by iteration. Thus the work at each time-step is much greater than for the *explicit* forward Euler method. The only reason for

considering this more complicated version is because of its stability properties. Applying Backward Euler to the test equation ($f(t,y) = -\lambda y$) we get

$$\begin{aligned} Y_{n+1} &= Y_n + h\, F(t_{n+1}, Y_{n+1}) \\ &= Y_n - h\, \lambda Y_{n+1} \\ (1+\lambda h)\, Y_{n+1} &= Y_n \\ Y_{n+1} &= \frac{1}{1+\lambda h}\, Y_n \end{aligned}$$

As $|\frac{1}{1+\lambda h}| < 1$ for all $\lambda > 0$ we say the method is unconditionally stable The Backward Euler solution to $y' = -10y$ starting from $Y_0 = y(0) = 1$

h	0.5	0.25	0.2	0.1	0.05
Y_1	0.16667	0.28571	0.33333	0.5	0.66667
Y_2	0.02778	0.08163	0.11111	0.25	0.44444
Y_3	0.00462	0.02332	0.03704	0.125	0.29629
Y_4	0.00077	0.00666	0.01235	0.0625	0.19753

The approximation to $y(1) = 4.5\ 10^{-05}$ is

h	0.5	0.25	0.2	0.1	0.05
n	2	4	5	10	20
$Y(1)$	0.02778	0.00666	0.00411	0.00098	0.00030

Approximating the integral

$$y(b) - y(a) = \int_a^b F(t, y(t))dt$$

using the Trapezoidal (Trapezium) Rule for numerical integration

$$y(t_{n+1}) - y(t_n) \approx \frac{h}{2}\left(F(t_n, y(t_n)) + F(t_{n+1}, y(t_{n+1}))\right)$$

leads to the formula

$$Y_{n+1} = Y_n + \frac{h}{2}\left(F(t_n, Y_n) + F(t_{n+1}, Y_{n+1})\right) \qquad n = 0, 1, \ldots \tag{5.7}$$

This formula is also known as the Trapezium Rule. If we apply this to the test equation we get

$$Y_{n+1} = Y_n + \frac{h}{2}\left(-\lambda Y_n - \lambda Y_{n+1}\right)$$

so

$$(1 + \lambda\frac{h}{2})Y_{n+1} = (1 - \lambda\frac{h}{2})Y_n$$

so

$$Y_{n+1} = \frac{(1 - \lambda\frac{h}{2})}{(1 + \lambda\frac{h}{2})}\, Y_n$$

and for all $\lambda\frac{h}{2} > 0$,

$$\left|\frac{(1-\lambda\frac{h}{2})}{(1+\lambda\frac{h}{2})}\right| < 1$$

so the method is *unconditionally stable*. To solve for Y_{n+1}at each step of the trapezium method (5.7), requires the solution of a nonlinear equation using iteration, *i.e.* an initial approximation and an iteration formula are needed. One implementation that uses Euler's method to generate the initial approximation $\tilde{Y}_{n+1}$ and then uses a single step of a iterative update in the form $Y_{n+1} = G(\tilde{Y}_{n+1})$ is known as the *modified Euler method* (or *Heun's method*),

$$\begin{aligned} \tilde{Y}_{n+1} &= Y_n + h\, F(t_n, Y_n) \\ Y_{n+1} &= Y_n + \tfrac{h}{2}\left(F(t_n, Y_n) + F(t_{n+1}, \tilde{Y}_{n+1})\right) \end{aligned}$$

It is illustrated in figure 5.5. The method can also be written in the standard notation for

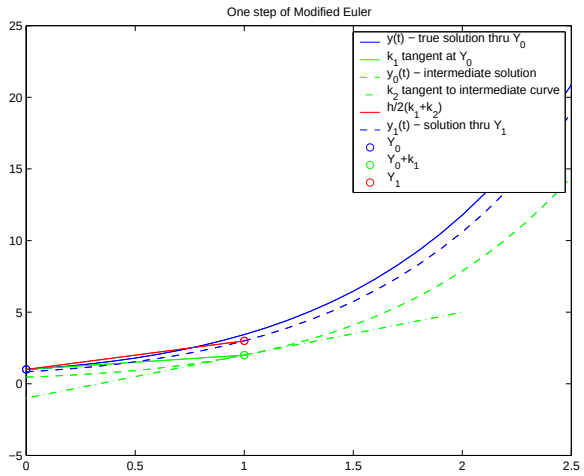


Figure 5.5: The Modified Euler Method

Runge-Kutta methods as a two-stage Runge-Kutta method

$$\begin{aligned} k_1 &= F(t_n, Y_n) \\ k_2 &= F(t_n + h, Y_n + hk_1) \\ Y_{n+1} &= Y_n + \tfrac{h}{2}\,(k_1 + k_2) \end{aligned}$$

Applying this method to the test equation gives:

$$\begin{aligned} \tilde{Y}_{n+1} &= Y_n - h\,\lambda Y_n \\ &= (1-\lambda h)\, Y_n \\ Y_{n+1} &= Y_n + \tfrac{h}{2}\left(-h\,\lambda Y_n - h\,\lambda\tilde{Y}_{n+1})\right) \\ &= Y_n + \tfrac{h}{2}\,(-h\,\lambda Y_n - h\,\lambda(1-\lambda h)\, Y_n)) \\ &= \left(1 - h\lambda + \frac{(h\lambda)^2}{2}\right) Y_n \end{aligned}$$

Then $\left|1 - h\lambda + \dfrac{(h\lambda)^2}{2}\right| < 1$ if

$$h < \frac{2}{\lambda}.$$

So Heun's method is conditionally stable, unconditionally stability is only attained if the iteration to solve for Y_{n+1} is continued to convergence. Note that

$$y(t_n + h) = e^{-h\lambda} y(t_n)$$

where

$$e^{-h\lambda} = 1 - h\lambda + \frac{(h\lambda)^2}{2} - \frac{(h\lambda)^3}{6} + \dots$$

and

$$Y_{n+1} = G\ Y_n$$

where

$$G = 1 - h\lambda + \frac{(h\lambda)^2}{2}$$

so the local truncation error is $\mathcal{O}(h^3)$ and the method is said to be *second order accurate*.

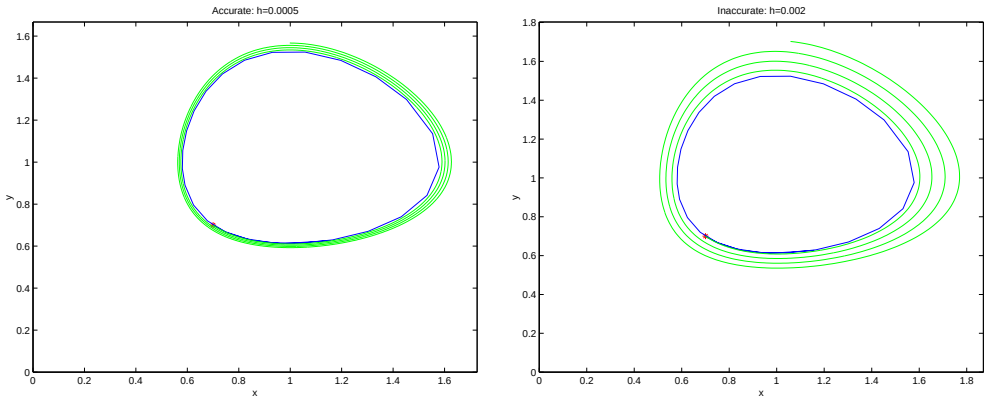
5.7 Systems of ode's

Systems of equations can be classified as either *stiff* or *non-stiff*. Stiff systems are those for which the limiting factor in the numerical solution is stability, whereas for non-stiff system for "reasonable" step sizes the limiting factor is truncation error. An example of a *Non-stiff system* is a model of two interacting species $y(t)$ (predator) + $x(t)$ (prey) modelled as

$$\left.\begin{aligned} y' &= -c(y - xy) \\ x' &= a(x - xy) \end{aligned}\right\} \quad \text{Lotka - Volterra Equations}$$

The example shown in figure 5.6 uses $a = 12$ and $c = 10$ as the rates of growth of the species and the population variation should follow the blue curve. In 5.6(a) the numerical solution (the green curve) uses the step size is $h = 0.0005$ in 5.6(b) the step size is increased by a factor 4 and clearly the error is increased in proportion.

In a non-stiff system the different components all have similar time-scales, but in a stiff



(a) $h = 0.0005$ (b) $h = 0.002$

Figure 5.6: Non-Stiff System: Error is truncation error

system there is a large difference in the time scales. Consider two linear systems of the form $\boldsymbol{y}' = A\boldsymbol{y}$ with

$$A_1 = \begin{pmatrix} -2 & 1 \\ 1 & -2 \end{pmatrix}$$

and

$$A_2 = \begin{pmatrix} -2 & 99 \\ 1 & -100 \end{pmatrix}$$

The eigenvalues of A_1 are $\lambda = -1, -3$ the eigenvalues for A_2 are $\lambda = -1, -100$. If $A = A_1$ the solution has two components e^{-t} and e^{-2t} whereas for $A = A_2$ the solution has two components e^{-t} and e^{-100t} for which the first is relatively slowly varying while it is the second component that determines the condition for stability, even if this is only a very small part of the solution. If, for example the Euler method is applied to the system $\boldsymbol{y}' = A\boldsymbol{y}$ then

$$\begin{aligned} \boldsymbol{Y}_{n+1} &= \boldsymbol{Y}_n - hA\boldsymbol{Y}_n \\ &= (I - Ah)\boldsymbol{Y}_n \end{aligned}$$

if the method is stable then $\|\boldsymbol{Y}_{n+1}\| < \|\boldsymbol{Y}_n\|$ and hence is stable if $\|I - Ah\| < 1$. Hence it is stable if the eigenvalues of $I - Ah$ are less the one in absolute value, that is if $h < \frac{2}{\lambda}$ for all eigenvalues λ of the matrix A. For stiff systems it is necessary to either or very small time steps or to use implicit methods.

5.8 Automatic Error Control

Automatic error control involves the automatic adjustment of the step size. It attempts to keep the additional local error generated at each step within predefined limits (both upper and lower bounds are needed). In Heun's method for example, each step provides *two* approximations to $y(t_{n+1})$, $\tilde{Y}_{n+1}$ and Y_{n+1}. It is assumed that Y_{n+1} is *much more accurate* than $\tilde{Y}_{n+1}$. So

$$|Y_{n+1} - y(t_{n+1})| \ll |\tilde{Y}_{n+1} - y(t_{n+1})|$$

and so:

$$\begin{aligned} \tilde{Y}_{n+1} - y(t_{n+1}) &= (\tilde{Y}_{n+1} - Y_{n+1}) + (Y_{n+1} - y(t_{n+1})) \\ &\approx \tilde{Y}_{n+1} - Y_{n+1} \end{aligned}$$

Thus we can estimate the error in the *least accurate* solution but *we use* the more accurate solution. With a predefined *local error tolerance* τ we check the additional error at each step:

- If $|\tilde{Y}_{n+1} - Y_{n+1}| > \tau$ reduce h and recompute Y_{n+1}.
- If $|\tilde{Y}_{n+1} - Y_{n+1}| << \tau$ accept Y_{n+1} but increase h before computing Y_{n+2}
- Otherwise accept Y_{n+1} and continue computation of Y_{n+2} with an unchanged value of h.

The *Runge-Kutta* method in the Matlab routine `ode45` uses this form of automatic step control. The formulae which are very accurate are set out below, but they are rather complicated and are taken from the `DOPRI5` method of *Dorman and Prince*. The basic

method is a *7 stage, 5th order method*:

$$\begin{aligned}
k_1 &= F(t_n, Y_n) && \approx y'(t_n)\\
k_2 &= F\left(t_n + \tfrac{1}{5}h, Y_n + \tfrac{1}{5}hk_1\right) && \approx y'(t_n + \tfrac{1}{5}h)\\
k_3 &= F\left(t_n + \tfrac{3}{10}h, Y_n + h(\tfrac{3}{40}k_1 + \tfrac{9}{40}k_2)\right) && \approx y'(t_n + \tfrac{3}{10}h)\\
k_4 &= F\left(t_n + \tfrac{4}{5}h, Y_n + h(\tfrac{44}{45}k_1 - \tfrac{56}{15}k_2 + \tfrac{32}{9}k_3)\right) && \approx y'(t_n + \tfrac{4}{5}h)\\
k_5 &= F\left(t_n + \tfrac{8}{9}h, Y_n + h(\tfrac{19372}{6561}k_1 - \tfrac{25360}{2187}k_2 + \tfrac{64448}{6561}k_3 - \tfrac{212}{729}k_4)\right) && \approx y'(t_n + \tfrac{8}{9}h)\\
k_6 &= F\left(t_n + h, Y_n + h(\tfrac{9017}{3168}k_1 - \tfrac{355}{33}k_2 + \tfrac{46732}{5247}k_3 + \tfrac{49}{176}k_4 - \tfrac{5103}{18656}k_5)\right) && \approx y'(t_n + h)\\
k_7 &= F\left(t_n + h, Y_n + h(\tfrac{35}{384}k_1 + \tfrac{500}{1113}k_3 + \tfrac{125}{192}k_4 - \tfrac{2187}{6784}k_5 + \tfrac{11}{84}k_6)\right) && \approx y'(t_n + h)
\end{aligned}$$

then

$$Y_{n+1} = Y_n + h\left(\frac{5179}{57600}k_1 + \frac{7571}{16695}k_3 - \frac{393}{640}k_4 - \frac{92097}{339200}k_5 + \frac{187}{2100}k_6 + \frac{1}{40}k_7\right) \tag{5.8}$$

This is a *seven stage method*, the local truncation error is $\mathcal{O}(h^6)$ hence it is a *fifth order method*. **But** within this method there is the six stage, fourth order method:

$$\tilde{Y}_{n+1} = Y_n + h\left(\frac{35}{384}k_1 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6\right) \tag{5.9}$$

hence we use $\tilde{Y}_{n+1} - Y_{n+1}$ as an estimate of the local truncation error $\tilde{Y}_{n+1} - y(t_{n+1})$ in the fourth order method. Thus we estimate the error in $\tilde{Y}_{n+1}$ by

$$\begin{aligned}
\tilde{Y}_{n+1} - y(t_{n+1}) &\approx (\tilde{Y}_{n+1} - Y_{n+1})\\
&= h\left(\tfrac{71}{57600}k_1 - \tfrac{71}{16695}k_3 + \tfrac{71}{1920}k_4 - \tfrac{17253}{339200}k_5 + \tfrac{22}{525}k_6 - \tfrac{1}{40}k_7\right)
\end{aligned}$$

This type of error control using two coupled methods is used in practical ode codes and has replaced the use of extrapolation in which each step was computed twice once with a step size h and once with a step size $\frac{h}{2}$.

The "classical" fourth order four stage Runge-Kutta method that appears in almost all textbooks and *was* extremely popular for pre-electronic computer calculation because the coefficients are simple, is now rarely used in practical computation as the only form of error control possible is extrapolation.

The Classical Runge-Kutta

$$Y_{n+1} = Y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned}
k_1 &= F(t_n, Y_n) && \approx y'(t_n)\\
k_2 &= F(t_n + \tfrac{1}{2}h, Y_n + \tfrac{h}{2}k_1) && \approx y'(t_n + \tfrac{1}{2}h)\\
k_3 &= F(t_n + \tfrac{1}{2}h, Y_n + \tfrac{h}{2}k_2) && \approx y'(t_n + \tfrac{1}{2}h)\\
k_4 &= F(t_n + h, Y_n + hk_3) && \approx y'(t_n + h)
\end{aligned}$$

5.9 Boundary value problems

The initial value problems discussed so far all had additional data specified at the beginning (say) $x = 0$ only. In a boundary value problem some data is specified at the beginning and some at the end such as

$$\begin{array}{c} y''(x) = f(x, y, y') \qquad a < x < b \\ y(a) = \alpha \quad y(b) = \beta \end{array}$$

(An initial value problem would have included $y'(a)$ but not $y(b)$.) There are two established numerical methods for solving problems of this type

- **Shooting Methods**, in which the problem is converted to a first order system and solved an an IVP, iterating on the unknown value of the the missing initial value $y'(a)$
- **Simultaneous difference equations**, in which the differential equation is replaced by difference equations on a fixed grid and resultant banded system is solved as a system of simultaneous equations.

5.9.1 Shooting

Solve as an IVP and use the mismatch at $x = b$ to correct the estimate of $y'(a)$. The error which can be written as

$$Y(b) - \beta$$

can be interpreted as a function of the initial value $\gamma = y'(a)$ so we implicitly define a function $F(\gamma)$ such that

$$F(\gamma) = Y(b) - \beta$$

and then we wish to solve $F(\gamma) = 0$ We use the secant method, as this avoids the difficulties associated with the definition of $\frac{dF}{d\gamma}$ needed in Newton-Raphson. Note that every evaluation of the function $F(\gamma)$ requires the solution of an IVP.

5.9.2 Band Matrix Method

Define a fixed mesh $x_n = a + nh$, $n = 0, 1, \ldots, N$ on the interval $[a, b]$, $x_N = b = a + Nh$ so $h = \frac{b-a}{N}$, with mesh values $y_n \approx y(x_n)$. Then

$$\begin{aligned} y'(x_n) &= \frac{D_0 y(x_n)}{h} + \mathcal{O}(h^2) \\ &\approx \frac{y_{n+1} - y_{n-1}}{2h} \end{aligned}$$

and

$$\begin{aligned} y''(x_n) &= \frac{D_+ D_- y(x_n)}{h^2} + \mathcal{O}(h^2) \\ &\approx \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} \end{aligned}$$

Definition 4 *Truncation Error The truncation error is defined by substituting the solution of the differential equation $L(u(x)) = g(x)$ into the difference equation $L_h(U(x_m)) = g_m$ then*

$$\left\{ \begin{array}{rcl} |L_h(u(x_m)) - g_m| & \leq & \mathcal{O}(h^q), \quad x_m = mh, \\ |u(x) - U(x)| & \leq & C(h^q). \end{array} \right. \tag{5.10}$$

The approximation is said to be accurate of order q, if $q > 0$ the method is consistent. [2]

Example 8 *Given*

$$y'' + (1+x^2)y' + e^x y + \sin x = 0 \quad 0 < x < 1$$
$$y(0) = 1 \quad y(1) = 0$$

we have

$$\frac{y_{n+1}-2y_n+y_{n-1}}{h^2} \; + \; (1+x_n^2)\frac{y_{n+1}-y_{n-1}}{2h} \; + \; e^{x_n}y_n + \sin x_n \; = \; 0$$

$$i = 1, 2, \ldots, N-1$$
$$y_0 = 1 \quad y_N = 0$$

Multiplying through by h^2 leads to a linear system that can be written as

$$A\boldsymbol{y} \; + \; B\boldsymbol{y} \; + \; C\boldsymbol{y} \; + \; \boldsymbol{f} \; = \; 0 \tag{5.11}$$

where

$$A = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \qquad C = \begin{pmatrix} e^{x_1} & & \\ & \ddots & \\ & & e^{x_{N-1}} \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & \frac{1+x_1^2}{2}h & & & \\ -\frac{1+x_2^2}{2}h & 0 & \frac{1+x_2^2}{2}h & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1+x_{N-2}^2}{2}h & 0 & \frac{1+x_{N-2}^2}{2}h \\ & & & -\frac{1+x_{N-1}^2}{2}h & 0 \end{pmatrix} \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_{N-1} \end{pmatrix}$$

$$\boldsymbol{f} = \begin{pmatrix} \sin(x_1)h^2 + y_0 + \frac{1+x_1^2}{2}hy_0 \\ \vdots \\ \sin(x_{N-1})h^2 + y_N - \frac{1+x_{N-1}^2}{2}hy_N \end{pmatrix}$$

Consider the same equation but with derivative boundary conditions:

$$y'' + (1+x^2)y' + e^x y + \sin x = 0 \quad 0 \le x \le 1$$
$$y'(0) = 1 \quad y'(1) = 0$$

then define a fixed mesh $x_n = a + (n - \frac{1}{2})h$, $n = 0, 1, \ldots, N$ on the interval $[a, b]$, $x_0 = a - \frac{h}{2}$ and $x_N = b + \frac{h}{2}$ hence x_0 and x_N are outside the interval $[a, b]$ and ther is no mesh point on the boundary so $h = \frac{b-a}{N-1}$, with mesh values $y_n \approx y(x_n)$. Then as $x_0 = a - \frac{h}{2}$, $x_1 = a + \frac{h}{2}$, $x_N = b + \frac{h}{2}$ and $x_{N-1} = b - \frac{h}{2}$ e

$$\begin{aligned} y'(a) &= \frac{y(x_1) - y(x_0)}{h} \; + \; \mathcal{O}(h^2) \\ y'(b) &= \frac{y(x_N) - y(x_{N-1})}{h} \; + \; \mathcal{O}(h^2) \end{aligned}$$

[2] This definition is analogous to the definition 2 for the truncation error in IVPs as the definition of the numerical scheme has been modified.

so for the numerical solution

$$y'(0) = \frac{y_1 - y_0}{h} \quad y_0 = y_1 - hy'(0) \tag{5.12}$$

and

$$y'(1) = \frac{y_N - y_{N-1}}{h} \quad y_N = y_{N-1} + hy'(1) \tag{5.13}$$

then the finite difference equations (5.11) are valid with the substitutions (5.12) and (5.13). In which case the matrices become

$$A = \begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix}$$

$$B = \begin{pmatrix} -\frac{1+x_1^2}{2}h & \frac{1+x_1^2}{2}h & & & \\ -\frac{1+x_2^2}{2}h & 0 & \frac{1+x_2^2}{2}h & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1+x_{N-2}^2}{2}h & 0 & \frac{1+x_{N-2}^2}{2}h \\ & & & -\frac{1+x_{N-1}^2}{2}h & \frac{1+x_{N-1}^2}{2}h \end{pmatrix}$$

$$\boldsymbol{f} = \begin{pmatrix} \sin(x_1)h^2 - hy'(0) - \frac{1+x_1^2}{2}h^2y'(0) \\ \vdots \\ \sin(x_{N-1})h^2 - hy'(1) + \frac{1+x_{N-1}^2}{2}h^2y'(1) \end{pmatrix}$$

Chapter 6

Finite Difference Methods for PDEs

6.1 Applications

- The transport equation:

  Assume a drop of ink in a pipe full of water moving a velocity c the concentration is denoted by $u(x,t)$. The amount of ink in a fixed length x of pipe at time t is $\int_0^x u(z,t)dz$ at a later time $t+h$ the ink has moved, with velocity c so the same ink is now $\int_{ch}^{x+ch} u(z,t+h)dz$ thus

  $$\int_0^x u(z,t)dz = \int_{ch}^{x+ch} u(z,t+h)dz$$

  differentiating w.r.t. x

  $$u(x,t) = u(x+ch,t+h)$$

  differentiating w.r.t. h at $h=0$

  $$0 = cu_x(x,t) + u_t(x,t)$$

- The wave equation:

  Consider the motion of a string of length l, fixed at the endpoints. From Newton's law of motion, the force of the tension is balanced by the acceleration, so considering two neighbouring points x_0 and x_1

  $$Tu_x|_{x_0}^{x_1} = \int_{x_0}^{x_1} \rho u_{tt} dx$$

  differentiate w.r.t x

  $$(Tu_x)_x = \rho u_{tt} \quad \rightarrow \quad u_{tt} = c^2 u_{xx}$$

- Diffusion (Heat Conduction):

  Given a region $\Omega \in \mathbb{R}^3$, the amount of heat at time t is denoted by $H(t)$ then

  $$H(t) = \int_\Omega c\rho u \, d\Omega$$

  where

– c is the specific heat

– ρ is the density

– u is the temperature

– k is the conductivity

then

$$\begin{aligned}\tfrac{dH}{dt} &= \textstyle\int_\Omega c\rho\partial_t u \, d\Omega \\ &= \text{heat flux out of } \Omega \\ &= \textstyle\int_\Gamma k\partial_n u \, ds \quad (\partial_n u = \boldsymbol{n}\cdot\nabla u \text{ the outward normal derivative}) \\ &= \textstyle\int_\Omega \nabla\cdot(k\nabla u) d\Omega \quad \text{The divergence theorem}\end{aligned}$$

so

$$c\rho\partial_t u = \nabla\cdot(k\nabla u)$$

Theorem 5 *Divergence Theorem: Let* $\Omega\mathbb{R}^n$ *be a bounded domain, with a piecewise smooth boundary* Γ. *If* $\boldsymbol{w}$ *is a smooth vector field defined on an open set containing* $\overline{\Omega} = \Omega\cup\Gamma$ *then*

$$\int_\Omega \nabla\cdot\ \boldsymbol{w}\ d\Omega \ = \ \int_\Gamma \boldsymbol{w}\cdot\boldsymbol{n}\ ds$$

- Laplace's Equation: Consider small displacements of a membrane, whose energy difference from no displacement is approximated by

$$I(\varphi) = \frac{1}{2}\int_\Omega \nabla^2\varphi \, d\Omega,$$

The functional I is to be minimised among all trial functions φ that assume prescribed values on the boundary Γ of Ω. If u is the minimising function and v is an arbitrary smooth function that vanishes on Γ, then the first variation of $I[u+v]$ must vanish:

$$\frac{d}{d\epsilon}I(u+\epsilon v)|_{\epsilon=0} = \int_\Omega \nabla u\cdot\nabla v \, d\Omega = 0.$$

Provided that u is sufficiently differentiable, it is possible to apply the divergence theorem to obtain

$$\int_\Omega \nabla\cdot(v\nabla u)\, d\Omega = \int_\Omega (\nabla u\cdot\nabla v + v\nabla\cdot\nabla u)\ d\Omega + \int_\Gamma v\partial_n u ds,$$

where s is arc-length along Γ. Since v vanishes on Γ and the first variation vanishes, the result is

$$\int_\Omega v\nabla u\cdot\nabla u \, d\Omega = 0$$

for all smooth functions v that vanish on Γ. It then follows that

$$\nabla^2 u = 0 \ \text{ in } \Omega,$$

assumption that the minimising function u has two derivatives.

6.2 Initial and Boundary Conditions

In order to obtain *exactly one solution*, it it is necessary to impose certain assumptions on the problem. Assumptions that can be justified by the physical meaning of the PDE are of two types:

- Initial Conditions: In the case of the wave, it is reasonable to specify the shape of the wave at $t = 0$ thus
$$u(x, 0) = g(x) \quad \text{given}$$
for the wave equation it is also necessary to specify the velocity at $t = 0$ (does the wave start from rest?) so
$$\partial_t u(x, 0) = f(x) \quad \text{is also known}$$

- Boundary Conditions: For the heat equation (stationary or not) it is necessary to specify what happens on the boundary Γ, there are three well known boundary conditions
  - $u|_\Gamma$ *Dirichlet boundary condition*, where the surface temperature is specified
  - $\partial_n u|_\Gamma$ *Neumann boundary condition*, where the heat flux out of the body is specified
  - $(\partial_n u + au)|_\Gamma$ where a is given, *Robin boundary condition*, the heat flux depends on the surface temperature (the body loses heat by conduction)

Definition 6 *: If the boundary condition is constantly equal to zero, then the condition is said to be* homogeneous*. If not then it is* inhomogeneous*.*

The heat conduction problem for which the heat distribution is known for $t = 0$ (an initial condition) and for which the surface temperature is specified (a boundary condition) is an example of an *initial-boundary value problem* or I.-B.V.P.

6.2.1 Equations of Second Order

Consider $\mathbb{R}^2$
$$0 = a_{11}u_{xx} + 2a_{12}u_{xy} + a_{22}u_{yy} + a_1u_x + a_2u_y + a_0u$$
note $u_{xy} = u_{yx}$ hence $a_{12} + a_{21} = 2a_{12}$. Then the second order part of the differential operator is
$$a_{11}\partial^2_{xx} + 2a_{12}\partial^2_{xy} + a_{22}\partial^2_{yy}$$
w.l.o.g. assume $a_{11} = 1$ then
$$\partial^2_{xx} + 2a_{12}\partial^2_{xy} + a_{22}\partial^2_{yy} = (\partial_x + a_{12}\partial_y)^2 + (-a^2_{12} + a_{22})\partial^2_{yy}$$
consider 3 cases depending on the sign of the second coefficient!

- Hyperbolic Equations: $(a_{11}a_{22} - a^2_{12}) < 0$
- Parabolic Equations: $(a_{11}a_{22} - a^2_{12}) = 0$
- Elliptic Equations: $(a_{11}a_{22} - a^2_{12}) > 0$

The three different types lead to 3 different types of well posed problems. For Hyperbolic and Parabolic problems, one of the variables is usually time so the variable y is replaced by t. Hyperbolic equations such as $u_{tt} - c^2u_{xx} = 0$ have many properties in common with the transport equation $u_t = au_x$.

6.3 Numerical Methods

As for o.d.e.'s it is possible to replace the derivatives by finite differences. Given an equation such as $u_t = au_x$, a mesh length h and a time-step k define a fixed mesh $x_n = mh$, $m = 0, 1, \ldots$ and $t_n = nk$, $n = 0, 1, \ldots$, with mesh values $U_m^n \approx u(x_m, t_n)$. As before we also use the alternative notation $U(x,t)$ for the numerical solution.

Definition 7 *Truncation Error The truncation error is defined by substituting the solution of the differential equation $\mathcal{L}u(x,t) = f(x,t)$ into the difference equation $\mathcal{L}_{h,k}(U(x_m,t_n)) = f_{m,n}$ then*

$$\begin{cases} |\mathcal{L}_{h,k}u(x_m,t_n) - f_{m,n}| & \leq & \mathcal{O}(h^{q_1} + k^{q_2}), \quad x_m = mh, t_n = nk, \\ |u(x,t) - U(x,t)| & \leq & C(h^{q_1} + k^{q_2}). \end{cases} \tag{6.1}$$

The approximation is said to be accurate of order (q_1, q_2), if $q_1, q_2 > 0$ the method is consistent.

An alternative definition is that the truncation error is defined as $\mathcal{L}_{hk}v - \mathcal{L}v \to 0$, for any function v, and the scheme is consistent if $\mathcal{L}_{hk}v - \mathcal{L}v \to 0$ as $h, k \to 0$, assuming that $f_{mn} = f(x_m, t_n)$.

Theorem 8 *Lax-Richtmeyer Equivalence Theorem: Given a properly posed linear initial-value problem and a linear finite-difference approximation to it that satisfies the consistency condition, stability is a necessary and sufficient condition for convergence*

Example 9 *Leapfrog: Consider the hyperbolic equation $u_t = au_x$, the leapfrog scheme is*

$$U(x, t+k) = U(x, t-k) + raD_0U(x,t)$$

where $r = k/h$ (see figure 6.1(b)). The order is $(2,2)$.

Example 10 *Lax-Friedrichs: The Lax-Friedrichs[1] scheme is*

$$U(x, t+k) = \frac{1}{2}(U(x+h,t) + U(x-h,t)) + raD_0U(x,t)$$

see figure 6.1(a).

Example 11 *Lax-Wendroff Scheme: Consider the equation $u_t = au_x$ the Lax-Wendroff scheme is*

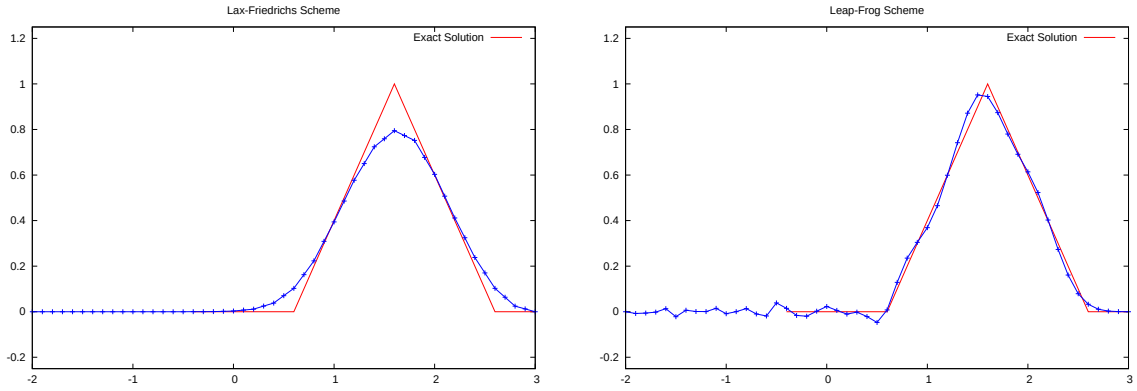
$$U(x, t+k) = \left(1 + raD_0 + \frac{a^2r^2}{2}D_+D_-\right)U(x,t)$$

if $u_t = au_x + f$ then with $x = mh$, $t = nk$ and $r = k/h$, with

$$g_{m,n} = \frac{1}{2}\left(f(x,t+k) + f(x,t)\right) - \frac{ak}{4h}\left(f(x+h,t) - f(x,t)\right)$$

the method is order (2,2), if $g_{m,n} = f(x,y)$ the scheme is only order (2,1).

[1] Peter David Lax (born May 1, 1926, Budapest), his PhD supervisor was Kurt O. Friedrichs



(a) Lax-Friedrichs scheme

(b) Leap-Frog Scheme

Figure 6.1: Solution of $u_t - u_x = 0$ at $t = 0.8$ with $r = 0.8$

The solution of the problem

$$\left.\begin{array}{rcl} u_t + au_x & = & 0, \quad t > 0 \\ u(x,0) & = & f(x) \end{array}\right\} \quad -\infty < x < \infty \tag{6.2}$$

satisfies

$$\left.\begin{array}{rcl} \hat{u}_t + i\xi a\hat{u} & = & 0 \\ \hat{u}(\xi,0) & = & \hat{f} \end{array}\right\} \quad \Rightarrow \quad \hat{u} = e^{-i\xi at}\hat{f}$$

so in particular

$$\hat{u}(\xi, t+k) = e^{-i\xi ak}\hat{u}(\xi,t)$$

In general a difference solution satisfies

$$\hat{U}(\xi,t) = g(\xi k)^n \hat{f}(\xi)$$

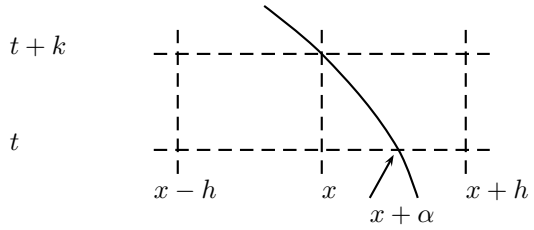
so

$$\begin{array}{rcl} \hat{U}(\xi,t+k) & = & g(\xi ah)\hat{U}(\xi,t) \\ & = & |g(\xi ah)|e^{-i\xi\alpha(\xi h)k}\hat{U}(\xi,t) \end{array}$$

If $\alpha = a$ for all frequencies ξ then all frequencies are propagated with the correct speed.

6.3.1 Courant Friedrichs Lewy (CFL) Condition

Assume that a first order hyperbolic differential equation has been approximated by a difference equation so that $U(x,t+k)$ is computed in terms of $U(x,t)$, $U(x+h,t)$ and $U(x-h,t)$ in addition assume that the characteristic through $(x,t+k)$ passes through the point $(x+\alpha,t)$.



Then the *domain of dependence* of the difference solution must include the domain of dependence of the differential equation, if not, the solution U *cannot converge* to u as $h, k \to 0$. Thus the point $(x+\alpha, t)$ must lie between $U(x+h,t)$ and $U(x-h,t)$. Given the differential equation $u_t = au_x$ the characteristics are $x + at =$ constant so the characteristic through $(x, t+k)$ passes through $(x+ak, t)$ so the *CFL condition* is

$$|ak| \leq h \quad \Rightarrow \quad |a|r \leq 1$$

where $r = \frac{k}{h}$. This shows that for $a > 0$ the forward difference scheme

$$U(x,t+k) = (1-ar)U(x,t) + arU(x+h,t)$$

will converge but

$$U(x,t+k) = (1+ar)U(x,t) - arU(x-h,t)$$

will converge if $a < 0$. In each case the scheme that will converge is known as an *upwind scheme*.

6.3.2 Stability

The CFL condition is concerned with convergence but it is closely associated with stability problems for the difference schemes. Again consider the first order equation

$$u_t = au_x \tag{6.3}$$

The coefficient a is the propagation velocity and is assumed constant. Then discretising the time derivative using a forward difference and expanding by Taylor's series (centred on x, t) gives

$$u_t = \frac{1}{k}\left(u(x,t+k) - u(x,t)\right) - \frac{k}{2}u_{tt} + \mathcal{O}(k^2) \tag{6.4}$$

using a central difference for the space derivative gives

$$u_x = \frac{1}{2h}\left(u(x+h,t) - u(x-h,t)\right) + \mathcal{O}(h^2) \tag{6.5}$$

Substituting into (6.3) gives

$$U(x,t+k) = U(x,t) + \frac{ak}{2h}\left(U(x+h,t) - U(x-h,t)\right) \tag{6.6}$$

this is first order accurate (in time), but substituting back with (6.4), shows that the difference equation (6.5) is a $\mathcal{O}(k^2+h^2)$ replacement (*i.e.* order (2,2)) of

$$u_t - au_x = -\frac{k}{2}u_{tt} \tag{6.7}$$

from (6.3) this is equivalent to

$$u_t - au_x = -\frac{k}{2}a^2 u_{xx} \tag{6.8}$$

as the diffusion coefficient in (6.8) is negative this means that the solution is unstable. If the centred difference (6.5) is replaced by either a forward difference which, expanding by Taylor's series gives

$$u_x = \frac{1}{h}(u(x+h,t) - u(x,t)) - \frac{h}{2}u_{xx} + \mathcal{O}(h^2) \tag{6.9}$$

or backward difference which, expanding by Taylor's series about (x,t), gives

$$u_x = \frac{1}{h}(u(x,t) - u(x-h,t)) + \frac{h}{2}u_{xx} + \mathcal{O}(h^2) \tag{6.10}$$

These give respectively

$$U(x,t+k) = U(x,t) + a\frac{k}{h}\left(U(x+h,t) - U(x,t)\right) \tag{6.11}$$

and

$$U(x,t+k) = U(x,t) + a\frac{k}{h}\left(U(x,t) - U(x-h,t)\right) \tag{6.12}$$

as second order replacements of respectively

$$u_t - au_x = \frac{a}{2}\left(h - ak\right)u_{xx} \tag{6.13}$$

and

$$u_t - au_x = -\frac{a}{2}\left(h + ak\right)u_{xx} \tag{6.14}$$

Thus for stability, with positive diffusion, the forward difference (6.11) is needed when a is positive and the backward difference (6.12) when a is negative (*i.e.* upwind or upstream differences), in addition to the CFL condition $|a|k \leq h$.

6.3.3 Stability Analysis

There are two standard ways of investigating the stability of finite difference approximations

- The matrix method where the scheme can be written as

  $$\boldsymbol{U}(t+k) = A\boldsymbol{U}(t) + \boldsymbol{b}$$

  and the stability depends on the eigenvalues of A.

- The Fourier Method (a.k.a. The von Neumann method[2]) is easier to apply, but from the von Neumann condition in general it only provides a necessary condition for stability.

[2] John von Neumann first used the method during world war II

In either case, assume that the finite difference equations are solved for two different initial conditions $f_1(x)$ and $f_2(x)$ to give two different solutions U_1 and U_2 the scheme is stable if the difference $w = U_1 - U_2$ remains bounded. The difference w satisfies the same finite difference equations as the solutions U_1 and U_2 so

$$\boldsymbol{w}(t+k) = A\boldsymbol{w}(t)$$

and it remains bounded if $\|A\| \leq 1$. In the Fourier method it is assumed that the difference can be expressed as a Fourier series, that is the sum of components such as

$$w(x,t) = \kappa^{t/k} e^{i\xi x}, \qquad x = mh,\ t = nk$$

for different frequencies ξ and the scheme is stable if it only permits solutions with $|\kappa| \leq 1$. Substituting into the difference scheme leads to an equation involving κ and ξh known as the *characteristic equation* (which has no connection with characteristics).

Definition 12 *Dissipation: Dissipation is the process whereby the solution looses energy and hence the solution decays ($|\kappa| < 1$). If all solutions satisfy*

$$|\kappa| \leq 1 - \delta|\xi h|^p, \quad 0 \leq |\xi h| \leq \pi$$

for some constant δ then p is called the order of dissipation.

Definition 13 *Dispersion: Dispersion is the phenomenon whereby different frequencies are propagated at different speeds. The quantity $\alpha(\xi h)$ is known as the* phase speed*.*

Example 14

- *Leapfrog: The characteristic equation is*

$$(\kappa^2 - 1) - 2\kappa ira \sin(\xi h) = 0$$

or

$$\kappa = ira \sin(\xi h) \pm \sqrt{1 - r^2 a^2 \sin^2(\xi h)}$$

so $|\kappa| = 1$ for $ra \leq 1$ and so the approximation is not dissipative.

- *Lax-Friedrichs: The characteristic equation is*

$$\kappa = \cos(\xi h) + ira \sin(\xi h)$$

so $|\kappa| \leq 1$ for $ra \leq 1$ it is not dissipative as $|\kappa| = 1$ when $\xi h = \pi$.

Finite Volume Methods

Consider the equation

$$u_t + u_x = 0,$$

define

$$\overline{u}(t) = \frac{1}{h}\int_x^{x+h} u(x,t)\ dt$$

then

$$\int_x^{x+h}\int_t^{t+k} (u_t + u_x)\ dtdx = \int_x^{x+h}\int_t^{t+k} u_t\ dtdx\ +\ \int_t^{t+k}\int_x^{x+h} u_x\ dxdt$$

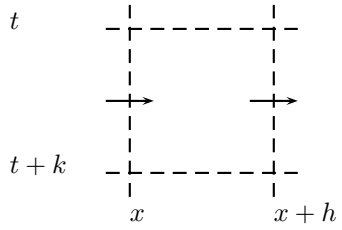
and so

$$\int_x^{x+h} (u(x,t+k) - u(x,t))\, dx \;+\; \int_t^{t+k} (u(x+h,t) - u(x,t))\, dt \;=\; 0$$

which can be written as

$$\overline{u}(t+k) = \overline{u}(t) - \int_t^{t+k} u(x+h,t)\, dt + \int_t^{t+k} u(x,t)\, dt$$

the integrals on the right represent flow across the boundaries of the cell (the finite volume).



6.4 Parabolic Equations

Consider the initial boundary value problem

$$\left\{ \begin{array}{rcl} u_t & = & u_{xx} \quad 0 < x < l,\ t > 0 \\ u(x,0) & = & f(x) \quad 0 < x < l \\ u(0,t) = g_0, & & u(l,t) = g_1 \quad t > 0 \end{array} \right. \tag{6.15}$$

Parabolic equations typically model heat conduction or diffusion processes, so if there is no input into the model, the typical solution exhibits an exponential decay. The proof is not restricted to one space dimension so rewrite (6.15) as

$$\left\{ \begin{array}{rcl} u_t & = & \nabla^2 u \quad \text{in } \Omega; t > 0 \\ u & = & f \quad \text{in } \Omega; t = 0 \\ u & = & g \quad \text{on } \Gamma; t > 0 \end{array} \right. \tag{6.16}$$

The space $\mathcal{L}_2(\Omega)$ is defined with the inner product

$$(u,v) = \int_\Omega uv\ d\Omega$$

and hence the norm

$$\|u\| = (u,u)^{1/2} = \left(\int_\Omega u^2\ d\Omega \right)^{1/2}.$$

Theorem 15 *Stability: Given that u is the solution of (6.16), it follows that if $g = 0$ then*

$$\frac{d}{dt}\|u\| \;<\; 0$$

and hence for any $t \geq 0$, $\|u(.,t)\| \leq \|f\|$.

Proof
For any fixed $t > 0$, using the divergence theorem

$$\begin{aligned} u_t &= \nabla^2 u \quad \text{in } \Omega, \\ u_t u &= u\nabla^2 u \quad \text{in } \Omega, \\ \int_\Omega u_t u \; d\Omega &= \textstyle\int_\Omega u\nabla^2 u \; d\Omega, \\ \int_\Omega u_t u \; d\Omega &= -\textstyle\int_\Omega |\nabla u|^2 \; d\Omega, \\ \tfrac{1}{2}\tfrac{d}{dt}\int_\Omega u^2 \; d\Omega &= -\textstyle\int_\Omega |\nabla u|^2 \; d\Omega \quad < 0 \end{aligned}$$

Hence $\|u(.,t)\|$ is a decreasing function of t so

$$\|u(.,t)\| < \|u(.,0)\| = \|f\|$$

Q.E.D.

6.4.1 Crank-Nicolson

A simple *explicit* method for the diffusion equation in (6.15) can be written as

$$\begin{aligned} U(x,t+k) &= U(x,t) + rD_+D_-U(x,t) \\ &= rU(x-h,t) + (1-2r)U(x,y) + rU(x+h,t) \end{aligned}$$

where now $r = \frac{k}{h^2}$. The Crank-Nicolson method for (6.15) is *implicit* and can be written as

$$U(x,t+k) = U(x,t) + \tfrac{r}{2}D_+D_- \left(U(x,t+k) + U(x,t)\right)$$

$$\begin{aligned} -\tfrac{r}{2}U(x-h,t+k) + (1+r)U(x,t+k) - \tfrac{r}{2}U(x+h,t+k) \\ = \tfrac{r}{2}U(x-h,t) + (1-r)U(x,t) + \tfrac{r}{2}U(x+h,t) \end{aligned}$$

Substituting w for U in the explicit scheme gives

$$\begin{aligned} w(x,t+k) &= rw(x-h,t) + (1-2r)w(x,t) + rw(x+h,t) \\ \kappa^{n+1}e^{i\xi x} &= r\kappa^n e^{i\xi(x-h)} + (1-2r)\kappa^n e^{i\xi x} + r\kappa^n e^{i\xi(x+h)} \\ \kappa &= re^{-i\xi h} + (1-2r) + re^{i\xi h} \\ &= 1 + 2r\left(\cos(\xi h) - 1\right) \\ &= 1 - 4r\sin^2\left(\tfrac{\xi h}{2}\right) \end{aligned}$$

hence the method is stable for all frequencies such that

$$-1 \le 1 - 4r\sin^2\left(\frac{\xi h}{2}\right)$$

and is stable for all frequencies if

$$r \leq \frac{1}{2}$$

In order to apply the matrix method it is necessary to write out the full set of difference equations, including the boundary conditions, in the form $\boldsymbol{U}(t+k) = A\boldsymbol{U}(t)+\boldsymbol{b}$, for problem (6.15) it follows that the explicit scheme with the mesh length $h = l/m$ leads to the $(m-1)\times(m-1)$ matrix

$$A = \begin{pmatrix} 1-2r & r & & & \\ r & 1-2r & r & & \\ & \ddots & \ddots & \ddots & \\ & & r & 1-2r & r \\ & & & r & 1-2r \end{pmatrix}$$

with

$$\boldsymbol{b} = \begin{pmatrix} rU(0,t) \\ 0 \\ \vdots \\ 0 \\ rU(mh,t) \end{pmatrix} \qquad \boldsymbol{U}(t) = \begin{pmatrix} U(h,t) \\ U(2h,t) \\ \vdots \\ U((m-2)h,t) \\ U((m-1)h,t) \end{pmatrix}$$

where $U(0,t)$ and $U(mh,t) \equiv U(l,t)$ are defined by the boundary conditions in (6.15). It is known that the eigenvalues of the $(m-1)\times(m-1)$ matrix

$$A = \begin{pmatrix} a & b & & & \\ c & a & b & & \\ & \ddots & \ddots & \ddots & \\ & & c & a & b \\ & & & c & a \end{pmatrix}$$

are

$$\lambda_s = a + 2\sqrt{bc}\cos\left(\frac{s\pi}{m}\right)$$

where a, b and c may be real or complex. With $h = \frac{l}{m}$ by writing $\xi = \frac{s\pi}{l}$ the condition $\lambda \leq 1$ gives the same condition as for the Fourier method which is necessary and sufficient in this case. The result for more general matrices such as

$$A = \begin{pmatrix} a & b & & & \\ c & a & b & & \\ & \ddots & \ddots & \ddots & \\ & & c & a & b \\ & & & c & -\beta+a \end{pmatrix}$$

which has eigenvalues

$$\lambda_s = a + 2\sqrt{bc}\cos\left(\frac{2s\pi}{2m-1}\right), \quad s = 1,\ldots,m-1$$

when $\beta = \sqrt{bc}$ are given in [8]. Matrices of this form arise when Neumann boundary conditions are included. For Crank-Nicolson the equation is

$$A_1\boldsymbol{U}(t+k) = A_0\boldsymbol{U}(t) + \boldsymbol{b}$$

where

$$A_0 = \begin{pmatrix} 1-r & \frac{r}{2} & & & \\ \frac{r}{2} & 1-r & \frac{r}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{r}{2} & 1-r & \frac{r}{2} \\ & & & \frac{r}{2} & 1-r \end{pmatrix} \quad \text{and} \quad A_1 = \begin{pmatrix} 1+r & -\frac{r}{2} & & & \\ -\frac{r}{2} & 1+r & -\frac{r}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{r}{2} & 1+r & -\frac{r}{2} \\ & & & -\frac{r}{2} & 1+r \end{pmatrix}$$

and $A = A_1^{-1}A_0$ as A_0 and A_1 have the same eigenvectors, the eigenvalues of A are

$$\lambda_s = \frac{1 - 2r\sin^2\left(\frac{\xi h}{2}\right)}{1 + 2r\sin^2\left(\frac{\xi h}{2}\right)}$$

which clearly satisfies $\lambda_s \leq 1$ for all ξ and all $r > 0$. So the stability of the Crank-Nicolson method is unrestricted. For other boundary conditions it may be necessary to bound the eigenvalues using:

Theorem 16 *Gershgorin's first theorem: The largest of the moduli of the eigenvalues cannot exceed the largest sum of the moduli of the elements in any row or column.*

Theorem 17 *Gershgorin's circle theorem: Let P_s be the sum of the moduli of the elements along the s-th row, excluding the diagonal a_{ss}. Then each eigenvalue lies inside or on at least one of the circles $|\lambda - a_{ss}| = P_s$.*

6.5 Elliptic Equations

Any numerical solution must preserve the property that the solution depends continuously on the boundary data and cannot have maxima or minima at interior points (*i.e.* Dirichlet's Principle). At the present time, in most practical computation finite element methods are preferred to finite differences. Currently work is focused primarily on solution methods, either fast direct solvers or efficient iterative methods.

Consider solving numerically the heat conduction problem

$$\begin{cases} u_{xx} + u_{yy} &= 0, \quad \forall x, y \in (0, l), \\ u(x, 0) &= g_0(x), \quad \forall x \in [0, 1] \\ u(x, 1) &= g_1(x), \quad \forall x \in [0, 1] \\ u_x(0, y) = & 0 \quad = u_x(1, y) \quad \forall y \in [0, 1]. \end{cases} \tag{6.17}$$

This corresponds to computing a temperature distribution on the unit square, with two edges thermally insulated and with prescribed temperatures on the others. The difference replacements are similar to those for two-point boundary value problems in section 5.9.2. A second order accurate finite difference approximation to the p.d.e. is

$$2(1+\alpha)U(x,y) - U(x-h_1,y) - U(x+h_1,y) - \alpha(U(x,y-h_2) + U(x,y+h_2)) = 0 \tag{6.18}$$

where $\alpha = \left(\frac{h_1}{h_2}\right)^2$. If $h_1 = h_2 = h$ this reduces to the *5-point Laplacian*

$$4U(x,y) - U(x-h,y) - U(x+h,y) - U(x,y-h) - U(x,y+h) = 0$$

To discretise the problem, use a uniform mesh of $(m_1+2)\times(m_2+2)$ nodes defined by:

$$\begin{array}{lll} x_j = (j-\frac{1}{2})h_1, & h_1 = 1/m_1, & j = 0,\ldots,m_1+1, \\ y_k = kh_2, & h_2 = 1/(m_2+1), & k = 0,\ldots,m_2+1. \end{array}$$

Note that the mesh is centred at the boundaries with Neumann boundary conditions ($x =$

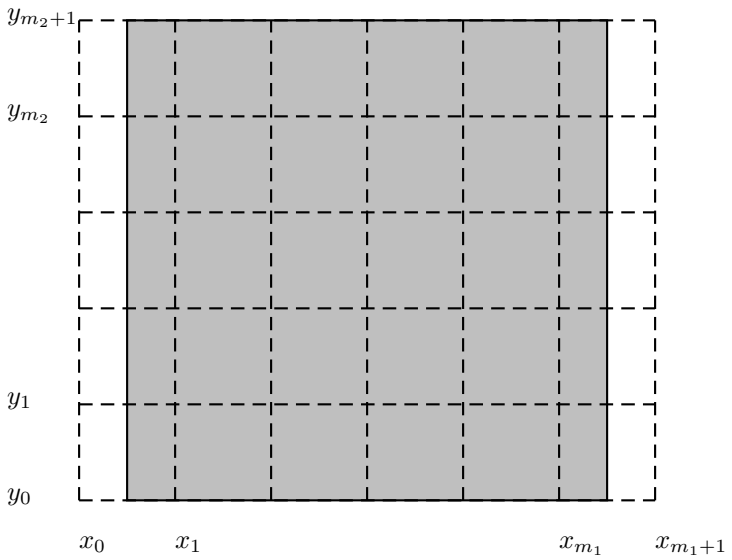


Figure 6.2: Grid for Poisson Problem

$0, 1$). Order the nodal values row-wise as

$$\boldsymbol{U}(y) = \begin{pmatrix} U(\frac{h_1}{2}, y) \\ \vdots \\ U(1-\frac{h_1}{2}, y) \end{pmatrix}, \qquad \boldsymbol{U} = \begin{pmatrix} U(h_2) \\ \vdots \\ U(1-h_2) \end{pmatrix}$$

The discrete solution satisfies finite difference equations that can be written in the form

$$A\boldsymbol{U} = \boldsymbol{b} \tag{6.19}$$

where

$$\boldsymbol{b}(0) = \begin{pmatrix} g_0(x_1) \\ \vdots \\ g_0(x_{m_1}) \end{pmatrix}, \qquad \boldsymbol{b}(1) = \begin{pmatrix} g_1(x_1) \\ \vdots \\ g_1(x_{m_1}) \end{pmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}(0) \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{b}(1) \end{pmatrix}$$

and the coefficient matrix is a block-tridiagonal $(m_1 m_2) \times (m_1 m_2)$ matrix with m_2 block rows

$$A = \begin{pmatrix} \tilde{A} & T & & & \\ T & \tilde{A} & T & & \\ & \ddots & \ddots & \ddots & \\ & & T & \tilde{A} & T \\ & & & T & \tilde{A} \end{pmatrix}$$

where for $\tilde{A}$ is the $m_1 \times m_1$ tridiagonal matrix

$$\tilde{A} = \begin{pmatrix} 2\alpha+1 & -1 & & & \\ -1 & 2\alpha+2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2\alpha+2 & -1 \\ & & & -1 & 2\alpha+1 \end{pmatrix}$$

and $T = -\alpha I$.

Chapter 7

Finite Element Methods

7.1 Introduction

The finite-element method was originally introduced in the 1950's as a method to calculate elastic deformations in solids. Later the method has been developed and generalised for all kinds of partial differential equations. It is the dominating technique for solid-mechanics problems such as estimating stresses and strains in elastic material under prescribed loads. CAD (Computer Aided Design) systems typically provides finite-element solvers in a highly integrated fashion. The engineer can typically with a few clicks on the computer screen estimate the deformations and stresses of, say, a machine part during the design. Finite-element methods are also commonly applied to other areas, such as calculations of electromagnetic fields and fluid flows.

In order to provide a brief introduction to the ideas, this note concentrates on a standard model problem for elliptic boundary-value problems, the Poisson problem. Only homogeneous Dirichlet boundary conditions are covered here.

7.2 FEM for the Poisson Problem in Two Space Dimensions

We consider the boundary-value problem

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \Gamma, \end{aligned} \tag{7.1}$$

where Ω is an open, bounded and connected domain in the plane, and Γ is its boundary. The Laplacian Δ is the sum of second derivatives

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Letting u represent a temperature field, equation (7.1) models steady heat conduction in a homogeneous, isotropic material, such as a metal, in which the temperature is held at zero on the boundary. The function f can be used to model heat sources such as electric heaters embedded in the material.

7.3 Green's Formula

.

We need some definitions and formulae from vector calculus. For any differentiable function v from $\mathbb{R}^m$ to $\mathbb{R}$, the *gradient* is the vector

$$\nabla v = \left(\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2}, \ldots, \frac{\partial v}{\partial x_m} \right),$$

and for any differentiable vector-valued function $\boldsymbol{w} = (w_1, w_2, \ldots, w_m)$ from $\mathbb{R}^m$ to $\mathbb{R}^m$, the *divergence* is

$$\nabla \cdot \boldsymbol{w} = \sum_{i=1}^{m} \frac{\partial w_i}{\partial x_i}.$$

By the product rule of differentiation, the formula

$$\nabla \cdot (v \nabla u) = \nabla v \cdot \nabla u + v \Delta u, \tag{7.2}$$

that is,

$$\sum_{i=1}^{m} \frac{\partial}{\partial x_i} \left(v \frac{\partial u}{\partial x_i} \right) = \sum_{i=1}^{m} \frac{\partial v}{\partial x_i} \frac{\partial u}{\partial x_i} + \sum_{i=1}^{m} v \frac{\partial^2 u}{\partial x_i^2},$$

holds for differentiable functions v and twice differentiable functions u.

Also recall the *divergence theorem* (or Gauss' theorem) which identifies the integral of a vector-field divergence over a domain with the integral of the normal component of the field along the boundaries:

$$\int_\Omega \nabla \cdot \boldsymbol{w} \, d\Omega = \int_\Gamma \boldsymbol{n} \cdot \boldsymbol{w} \, ds, \tag{7.3}$$

Here, $\boldsymbol{n}$ denotes the outward-directed unit normal on Γ. Identity (7.3) holds for functions $\boldsymbol{w}$ and boundaries Γ that are sufficiently smooth.

Combining the divergence theorem (7.3) with formula (7.2) yields *Green's formula*

$$\int_\Gamma v \frac{\partial u}{\partial n} \, ds = \int_\Omega \nabla v \cdot \nabla u \, d\Omega + \int_\Omega v \Delta u \, d\Omega, \tag{7.4}$$

where

$$\frac{\partial u}{\partial n} = \boldsymbol{n} \cdot \nabla u = \sum_{i=1}^{m} n_i \frac{\partial u}{\partial x_i},$$

denotes the directional derivative of u in the normal direction. Green's formula is a generalisation to higher dimensions of the integration-by-parts formula

$$\int_0^1 u' v' \, dx = u'(1)v(1) - u'(0)v(0) - \int_0^1 u'' v \, dx.$$

7.4 The Variational Form

A *classical solution* to the Poisson problem (7.1) is a smooth function u satisfying equation (7.1). The precise requirements for u to be a classical solution is that it should be twice continuously differentiable, and its first and second derivatives should be functions that can

be continuously extended up to the boundary. This assures that Green's formula (7.4) can be applied on u. Let v be a smooth function from $\overline{\Omega} = \Omega \cup \Gamma$ to $\mathbb{R}$ such that $v(x) = 0$ for each $x \in \Gamma$. Multiply both sides of equation (7.1) with v, integrate over Ω, and apply Green's formula (7.4) to obtain

$$\begin{array}{ll} \int_\Omega f v \, d\Omega & = -\int_\Omega v \Delta u \, d\Omega \\ & = -\int_\Gamma v \frac{\partial u}{\partial n} \, ds + \int_\Omega \nabla v \cdot \nabla u \, d\Omega = \int_\Omega \nabla v \cdot \nabla u \, d\Omega, \end{array} \tag{7.5}$$

where the fact that v vanishes on the boundary has been used in the last equality. From expression (7.5) immediately follows

Theorem 18 *If u is a classical solution to the Poisson problem (7.1), then u satisfies*

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega = \int_\Omega f v \, d\Omega, \tag{7.6}$$

for each smooth function v vanishing on the boundary.

Equation (7.6) is called the *variational form* of the Poisson equation. Theorem 18 refers to the original problem (7.1), but the variational form can be used to *define* a function u without reference to the differential equation. For this purpose, we introduce the *function space*

$$V = \left\{ v : \int_\Omega |\nabla v|^2 \, d\Omega < +\infty \text{ and } v|_\Gamma = 0 \right\} , \tag{7.7}$$

where

$$|\nabla v|^2 = \left(\frac{\partial v}{\partial x_1} \right)^2 + \left(\frac{\partial v}{\partial x_2} \right)^2 .$$

The condition

$$\int_\Omega |\nabla v|^2 \, d\Omega < +\infty$$

corresponds in many applications to demanding that the *energy* should be bounded, for instance when the Poisson equation is used to model steady heat conduction. Note that V is a *linear space*, that is, if v, $w \in V$, then $\alpha v + \beta w \in V$ for each α, $\beta \in \mathbb{R}$. The space V is a so-called Sobolev space, and is often denoted $H_0^1(\Omega)$ in the literature.

The variational problem, now formulated without reference to the differential equation (7.1) is

$$\begin{array}{l} \text{Find } u \in V \text{ such that} \\ \int_\Omega \nabla u \cdot \nabla v \, d\Omega = \int_\Omega f v \, d\Omega \qquad \forall v \in V. \end{array} \tag{7.8}$$

Solutions to variational problem (7.8) are called *weak solutions* of the partial differential equation (7.1). From Theorem 18 follows that classical solutions are weak solutions. As the label "weak" suggests, there are weak solutions that are not classical solutions. However, one can show that weak solutions are classical solutions provided that the function f and the boundary Γ are sufficiently smooth.

7.5 The Minimisation Problem

The variational form above is all that is needed to define a finite-element discretisation. However, a classical solution to the particular problem that we consider, equation (7.1),

also satisfies a certain *minimisation problem*, that is, the classical solution minimises the quadratic form

$$F(v) = \frac{1}{2}\int_\Omega |\nabla v|^2 \, d\Omega - \int_\Omega fv \, d\Omega.$$

Similarly as was done for the variational problem, we can also consider the problem of minimising F within the function space V without reference to classical solutions, that is, consider the problem:

$$\begin{array}{c} \text{find } u \in V \text{ such that} \\ F(u) \leq F(v) \qquad \forall v \in V. \end{array} \tag{7.9}$$

In fact, the variational problem (7.8) and the minimisation problem (7.9) are equivalent:

Theorem 19 *The element $u \in V$ minimises F if and only if it is a solution to the variational problem (7.8)*

Remark 1 *The proof below may appear long, but in essence it is really no more complicated than showing that the parabola $F(x) = \frac{1}{2}x^2 - xf$ has its minimum at $x = f$*

Proof
For any u, $v \in V$, we have

$$\begin{array}{rl} F(u+v) & = \frac{1}{2}\int_\Omega |\nabla u + \nabla v|^2 \, d\Omega - \int_\Omega f(u+v) \, d\Omega \\ & = \frac{1}{2}\int_\Omega \left[|\nabla u|^2 + 2\nabla u \cdot \nabla v + |\nabla v|^2\right] \, d\Omega - \int_\Omega f(u+v) \, d\Omega. \end{array} \tag{7.10}$$

(i) Assume that $u \in V$ is a solution to the variational problem (7.8). Then expression (7.10) reduces to

$$\begin{array}{rl} F(u+v) & = \frac{1}{2}\int_\Omega |\nabla u|^2 \, d\Omega - \int_\Omega fu \, d\Omega + \frac{1}{2}\int_\Omega |\nabla v|^2 \, d\Omega \\ & = F(u) + \underbrace{\frac{1}{2}\int_\Omega |\nabla v|^2 \, d\Omega}_{\geq 0} \geq F(u) \end{array} \tag{7.11}$$

for any $v \in V$, which shows that u minimises F.

(ii) Now assume that $u \in V$ minimises F. For any $t \in \mathbb{R}$ and $v \in V$, we define the function $f(t) = F(u + tv)$, that is, by perturbing F away from its minimum. Expression (7.10) yields that

$$\begin{array}{rl} f(t) & = F(u+tv) \\ & = F(u) + t\left(\int_\Omega \nabla u \cdot \nabla v \, d\Omega - \int_\Omega fv \, d\Omega\right) + \frac{t^2}{2}\int_\Omega |\nabla v|^2 \, d\Omega, \end{array} \tag{7.12}$$

that is, f is a second-order polynomial in t with a minimum when the derivative vanishes (since the leading term is non-negative). We also know that the minimum is attained for $t = 0$ since u minimises F. Setting $f'(0) = 0$ yields that

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega - \int_\Omega fv \, d\Omega = 0, \tag{7.13}$$

for any $v \in V$, that is, u is a solution to the variational problem (7.8).

Q.E.D.

Remark 2 *Variational forms can be defined for practically all elliptic boundary-value problems, but minimisation forms cannot always be defined, for instance when the differential equation contains first-derivative terms.*

Remark 3 *In mechanics application the variational form (7.8) is called* the principle of virtual work*, and the minimisation problem (7.9) is called* the principle of minimum potential energy.

Remark 4 *The terminology used here, "variational" for (7.8) and "minimisation" for (7.9), is convenient for our purpose, but is not the only existing. Quite commonly the minimisation problem is called a variational form. In fact, the notion of variational forms was first attached to minimisations of "functionals" like F in the* calculus of variations.

7.6 Meshing and Finite-Element Approximation

We introduce a *triangulation* of the domain Ω, that is, Ω will be subdivided into non-overlapping triangles as illustrated in figures 7.1 and 7.3. The triangular corners are called the *nodes* of the triangulation. The *boundary nodes* are the nodes which are located on the boundary, and the *internal nodes* are the nodes which are not boundary nodes. A valid triangulation should not contain "hanging nodes", that is, no node should be located at another triangles side, as in figure 7.2. The "fineness" of the triangulation is characterised by a parameter $h > 0$, the largest length of any of the triangular sides, for instance.

Now define V_h as the space of all functions that are *continuous* on $\overline{\Omega}$, *linear* on each triangle, and *vanishing* on the boundary Γ. The graph of such a function is a surface composed of triangular-shaped planes, as illustrated in figure 7.4.

This space is constructed so that $V_h \subset V$, and we define the *finite-element discretisation* of the Poisson problem (7.1) as

$$\begin{array}{l} \text{Find } u_h \in V_h \text{ such that} \\ \int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \qquad \forall v_h \in V_h. \end{array} \tag{7.14}$$

Note that the discretisation is obtained simply by replacing V with the subspace V_h in the variational form (7.8). This kind of procedure is also called a *Galerkin approximation*.

7.7 The Algebraic Problem

A function in the above defined space V_h is uniquely defined by its values at the *internal* nodes (we already know that the function is zero at the boundary nodes). To see this, it is enough to note that the planar surface of u_h on each triangle is uniquely defined by the values of u_h at the triangular corners. Let N be the number of internal nodes. Using the *basis functions* $\{\phi_j(\boldsymbol{x})\}_{j=1}^N \subset V_h$, each function $u_h \in V_h$ can be written

$$u_h(\boldsymbol{x}) = \sum_{j=1}^{N} u_j \phi_j(\boldsymbol{x}), \tag{7.15}$$

where u_j is the value of u_h at note j, and $\phi_j(\boldsymbol{x})$ is the "tent" function depicted in figure 7.5. The function ϕ_j is zero everywhere, except that it raises as a "tent" around node j, that is, $\phi_j \in V$ such that

$$\phi_j(\boldsymbol{x}_k) = \begin{cases} 1 & \text{if } k = j, \\ 0 & \text{otherwise,} \end{cases}$$

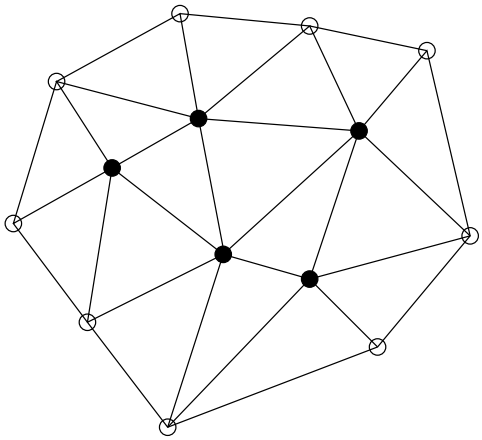


Figure 7.1: A valid triangulation. Internal nodes are marked by solid dots and boundary nodes by circles.

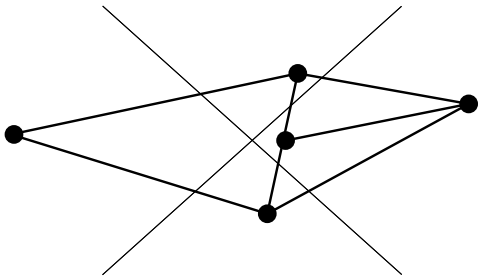


Figure 7.2: Not a valid triangulation: contains hanging nodes.

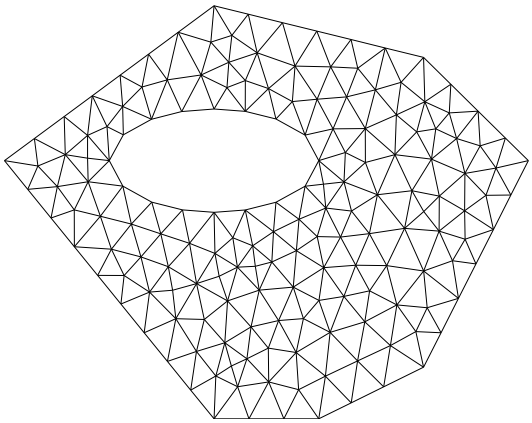


Figure 7.3: A more complicated triangulated domain (note that the domain may contain holes!)

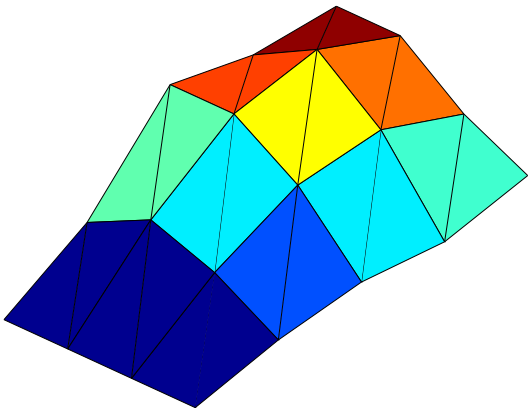


Figure 7.4: The functions in V_h are continuous and linear on each triangle. (The boundary nodes are not included in this picture.)

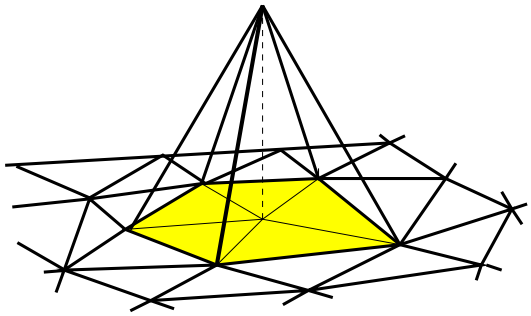


Figure 7.5: The basis function $\phi_j(\boldsymbol{x})$ is equal to one at node j and zero at all other nodes.

where $\boldsymbol{x}_k$ is the coordinate of node k. Substituting the expansion (7.15) into equation (7.14) yields that

$$\sum_{j=1}^{N} u_j \int_\Omega \nabla\phi_j \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \qquad \forall v_h \in V_h.$$

Since equation (7.7) should hold for each $v_h \in V_h$, it must in particular hold for $v_h = \phi_i$, $i = 1, \ldots, N$, which means that

$$\sum_{j=1}^{N} u_j \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\Omega = \int_\Omega f\phi_i \, d\Omega \qquad i = 1, \ldots, N. \tag{7.16}$$

Problem (7.16) is a system of linear equation in the coefficients u_j, $j = 1, \ldots, N$, that is,

$$A\boldsymbol{u} = \boldsymbol{b}, \tag{7.17}$$

where the matrix A has components

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, d\Omega,$$

and

$$\boldsymbol{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} \int_\Omega f\phi_1 \, d\Omega \\ \vdots \\ \int_\Omega f\phi_N \, d\Omega \end{pmatrix}.$$

With a terminology borrowed from solid mechanics, the matrix A is called the *stiffness matrix* and the vector $\boldsymbol{b}$ the *load vector*. This terminology is used also for cases, like heat conduction, when the PDE we are discretising has nothing to do with mechanics!

We conclude that a numerical approximation of the Poisson problem with a finite-element method involves setting up and solving the linear system (7.17).

7.8 An Example

Let the domain Ω be the unit square, and consider the *structured mesh* of figure 7.6. There are J internal nodes in both directions and the sides of each triangle are $h = 1/(J+1)$. There is a total of $J^2 = N$ internal nodes, assumed to be numbered in the row-wise direction

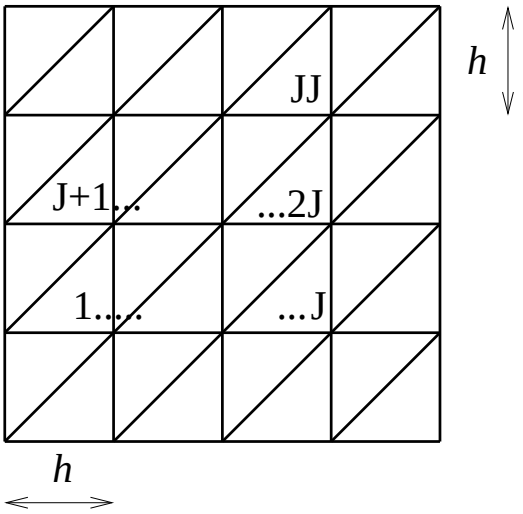


Figure 7.6: A structured meshing of the unit square.

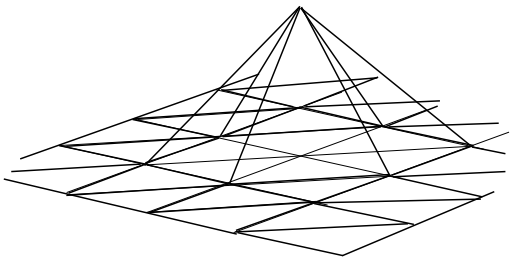


Figure 7.7: A basis function associated with the mesh in figure 7.6

as indicated in figure 7.6. The basis functions ϕ_i have the shape indicated in figure 7.7. The *support* of each basis function, that is, the nonzero region of the function, consists of the 6 neighbouring triangles that surround node i. Note that this means that most of the stiffness matrix elements

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, d\Omega$$

are zero. For instance, $A_{i,i+2} = 0$ since there is no overlap in the support for the functions ϕ_i and ϕ_{i+2}; see figure 7.8. In fact, A_{ij} can be nonzero only when i and j are associated with *nearest-neighbouring* nodes (figure 7.9).

To calculate the stiffness-matrix elements, we need to know the gradients of the basis functions,

$$\nabla\phi_i = \left(\frac{\partial\phi_i}{\partial x}, \frac{\partial\phi_i}{\partial y} \right).$$

The gradient is constant at each triangle since ϕ_i is composed of planar surfaces. Letting the x and y directions be oriented in the horizontal and vertical directions, respectively, the values of the gradient at the support of the basis function are indicated in figure 7.10. Note that the basis function is equal to one at the filled dot and equal to zero at the open dots, which means that the gradient can simply be read off as the slope of the "tent" function along the sides of the triangles. With the aid of the gradients given in figure 7.10, we can compute the diagonal elements in the stiffness matrix,

$$\begin{aligned} A_{ii} &= \textstyle\int_\Omega \nabla\phi_i \cdot \nabla\phi_i \, d\Omega = \sum_{i=1}^{6} \int_{T_k} \nabla\phi_i \cdot \nabla\phi_i \, d\Omega \\ &= \tfrac{1}{h^2}|T_1| + 2\tfrac{1}{h^2}|T_2| + \tfrac{1}{h^2}|T_3| + \tfrac{1}{h^2}|T_4| + 2\tfrac{1}{h^2}|T_5| + \tfrac{1}{h^2}|T_6| \\ &= 8\tfrac{1}{h^2}\tfrac{h^2}{2} = 4. \end{aligned}$$

To compute $A_{i,i+1}$, note that $\nabla\phi_i \cdot \nabla\phi_{i+1} \neq 0$ only in two triangles (figure 7.11), thus

$$\begin{array}{lllll} \text{on } T_1: & \nabla\phi_i = \left(-\frac{1}{h}, \frac{1}{h}\right) & \nabla\phi_{i+1} & = & \left(\frac{1}{h}, 0\right) \\ \text{on } T_2: & \nabla\phi_i = \left(-\frac{1}{h}, 0\right) & \nabla\phi_{i+1} & = & \left(\frac{1}{h}, -\frac{1}{h}\right) \end{array}$$

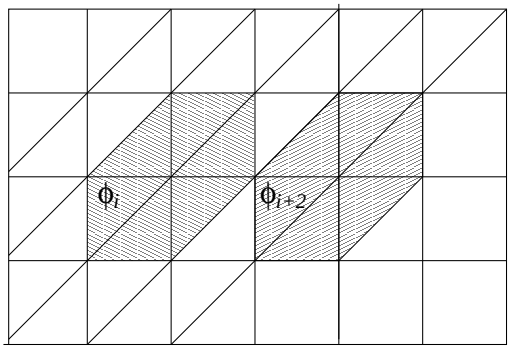


Figure 7.8: There is no overlap in the support for basis functions ϕ_i and ϕ_{i+2}.

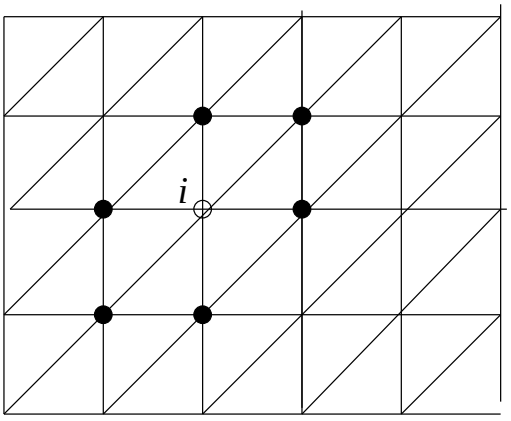


Figure 7.9: The nearest neighbours to node i are the six nodes marked with black dots. Thus, A_{ij} can be nonzero only when j corresponds to one of the black dots.

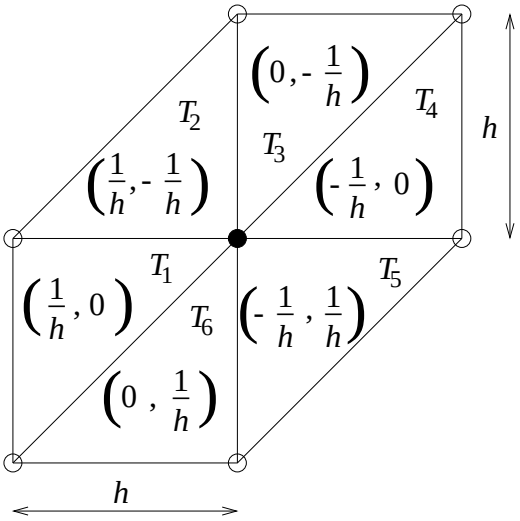


Figure 7.10: The gradient of basis function ϕ_i is piecewise constant on each triangle. The x- and y-coordinates are given as the pair $(\cdot, \cdot)$ at each triangle of the support of the function.

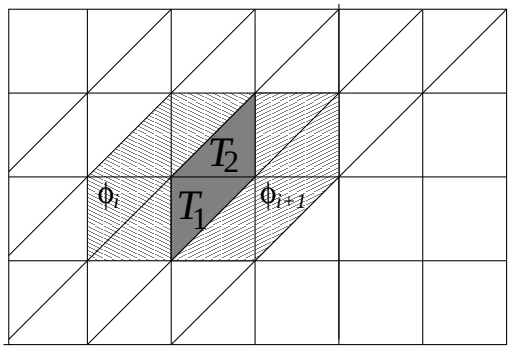


Figure 7.11: The overlap in the support of basis functions ϕ_i and ϕ_{i+1} are the triangles T_1 and T_2.

and thus

$$\begin{aligned} A_{i,i+1} &= \int_\Omega \nabla\phi_i \cdot \nabla\phi_{i+1}\, d\Omega = \sum_{k=1}^{2} \int_{T_k} \nabla\phi_i \cdot \nabla\phi_{i+1}\, d\Omega \\ &= -\tfrac{1}{h^2}|T_1| - \tfrac{1}{h^2}|T_2| = -\tfrac{2}{h^2}\tfrac{h^2}{2} = -1. \end{aligned}$$

Similar calculations yield that

$$A_{i,i-1} = A_{i,i+J} = A_{i,i-J} = -1, \qquad A_{i,i+J+1} = A_{i,i-J-1} = 0.$$

Also note that the matrix A is *symmetric*: $A_{ij} = A_{ji}$. Altogether, we obtain the *block triangular structure* (empty space means zeros!)

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix}$$

where T and I are the J-by-J matrices

$$T = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1. \end{pmatrix},$$

Thus, the ith row of the matrix-vector product $A\boldsymbol{u}$ will be

$$4u_i - u_{i+1} - u_{i-1} - u_{i+J} - u_{i-J}. \tag{7.18}$$

Node $i+1$ and $i-1$ is located to the right and left, respectively, of node i, whereas nodes $i+J$ and $i-J$ are above and below node i. Thus, expression (7.18) is precisely the classical five-point, finite-difference formula. We reach the remarkable conclusion that the finite-element discretisation of the Laplace operator using continuous, piecewise-linear functions on the structured mesh of figure 7.6 reduces to a standard finite-difference formula for the Laplacian. Note, however, that this does not hold in general; finite-element discretisations are not always easy to interpret as a finite-difference method.

7.9 Properties of the Stiffness Matrix

Consider the stiffness matrix A with components

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j\, d\Omega,$$

which was obtained by discretising the Poisson problem (7.1). This matrix has some very particular properties, which will be discussed in this section: it is *symmetric*, *positive definite*, *sparse*, and *ill conditioned*. All these properties, except the sparsity, reflects the nature of the boundary-value problem (7.1). Some or all of these properties may change if the equation or the boundary conditions are altered. For instance, if an additional term containing first derivatives of u is added to equation (7.1), the stiffness matrix will no longer be symmetric.

The sparsity is a consequence of the fact that the chosen piecewise-linear approximations allow a compact basis, the "tent" functions of figure 7.5.

The symmetry of the matrix is immediate,

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, d\Omega = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, d\Omega = A_{ji}.$$

Moreover, the matrix is *sparse*, since $A_{ij} = 0$ whenever i and j are not nearest neighbours. The number of neighbours to each point does not increase when the mesh is made finer, as long as the mesh refinements are made in a sensible way, see the discussion in section 7.10. Thus, the number of nonzero elements on each row does not increase with the order of the stiffness matrix, that is, the matrix in a sense becomes sparser and sparser with increasing matrix order.

Recall that a real matrix A is *positive definite* if $\boldsymbol{v}^T A\boldsymbol{v} > 0$ whenever $\boldsymbol{v} \neq 0$.

Theorem 20 *The stiffness matrix is positive definite.*

Proof
Let $v_h \in V_h$. Expanding v_h in the "tent" basis functions yields

$$v_h = \sum_{i=1}^{N} v_i \phi_i(\boldsymbol{x}).$$

Setting

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_N)^T,$$

yields that

$$\begin{aligned} \boldsymbol{v}^T A\boldsymbol{v} &= \textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} v_i \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, d\Omega v_j \\ &= \int_\Omega \underbrace{\sum_{i=1}^{N} \nabla(v_i\phi_i)}_{=\nabla v_h} \cdot \underbrace{\sum_{j=1}^{N} \nabla(v_j\phi_j)}_{=\nabla v_h} \, d\Omega = \int_\Omega |\nabla v_h|^2 \, d\Omega \geq 0, \end{aligned} \tag{7.19}$$

with equality if and only if $\nabla v_h = 0$, that is, if v_h is constant. However, since v_h is zero on the boundary (by definition of V_h), it follows that the constant must be zero. Thus expression (7.19) is zero only if $v_h \equiv 0$, that is, when $\boldsymbol{v} = \boldsymbol{0}$. Q.E.D.

One important consequence of Theorem 20 is that equation (7.17) has a unique solution. This follows from the fact that positive-definite matrices are nonsingular: For a *singular* matrix A, there would be nonzero vector $\boldsymbol{v}$ so that $A\boldsymbol{v} = \boldsymbol{0}$, and thus $\boldsymbol{v}^T A\boldsymbol{v} = 0$. Thus, singular matrices cannot be positive definite, and positive-definite matrices must therefore be nonsingular.

The condition number of the stiffness matrix depends strongly on h. In fact, if the quotient between the size of the smallest and largest triangle in the mesh is kept bounded as the mesh is refined, one can show that the condition number grows like $\text{cond}(A) = \mathcal{O}(h^{-2})$. The stiffness matrix is thus ill conditioned for fine meshes.

7.10 Accuracy

We have shown how to define a finite-element approximation of the Poisson problem (7.1), that this yields the linear system (7.17), and that this has a unique solution. The question

how good the finite-element solution is as an approximation of the original problem will be discussed in this section.

For finite-difference discretisations, accuracy questions are usually addressed indirectly by applying the Lax–Richtmyer Theorem. The crucial steps is then to derive truncation errors and to check stability. If the method is consistent, that is, if the truncation error vanishes as the mesh is refined, the method is convergent if and only if it is stable. For finite-element discretisations, this approach is hardly ever used, since it is possible to study the error in the discretisation directly. The easiest and most natural way is to work with *integral norms* of the difference between the weak solution u of problem (7.8) and the finite-element solution u_h of problem (7.14). The $L^2(\Omega)$ norm of a function,

$$|v|_{L^2(\Omega)} = \left(\int_\Omega v^2 \, d\Omega \right)^{1/2},$$

is the analogue for functions of the vector 2-norm. The perhaps most important norm for solutions of the Poisson problem is the *energy norm*

$$|v|_V = \left(\int_\Omega |\nabla v|^2 \, d\Omega \right)^{1/2}, \tag{7.20}$$

that is, the $L^2(\Omega)$-norm of the first derivatives; recall that weak solutions were defined among functions with bounded energy norm (the space of functions defined by (7.7)). The importance of the energy norm is that the finite-element solution is *optimal* in the energy norm. That is, no other function in V_h yields a smaller error in energy norm:

Theorem 21 *Let u be the solution to variational problem (7.8) and u_h the finite-element solution (7.14). Then*

$$|u - u_h|_V \leq |u - v_h|_V \qquad \forall v_h \in V_h, \tag{7.21}$$

Proof
By equation (7.14), the finite-element solution u_h satisfies

$$\int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \qquad \forall v_h \in V_h. \tag{7.22}$$

From equation (7.8) follows that the weak solution u satisfies

$$\int_\Omega \nabla u \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \qquad \forall v_h \in V_h, \tag{7.23}$$

since $V_h \subset V$. Subtracting equations (7.22) and (7.23) yields that

$$\int_\Omega \nabla (u - u_h) \cdot \nabla v_h \, d\Omega = 0 \qquad \forall v_h \in V_h. \tag{7.24}$$

Let v_h be an arbitrary element of V_h. Then

$$\begin{aligned}
|u - u_h|_V^2 &= \textstyle\int_\Omega |\nabla(u - u_h)|^2 \, d\Omega = \int_\Omega [\nabla(u - u_h)] \cdot [\nabla(u - u_h)] \, d\Omega \\
&= \textstyle\int_\Omega \nabla(u - u_h) \cdot \nabla u \, d\Omega - \underbrace{\int_\Omega \nabla(u - u_h) \cdot \nabla u_h \, d\Omega}_{= 0 \text{ by (7.24)}} \\
&= \textstyle\int_\Omega \nabla(u - u_h) \cdot \nabla u \, d\Omega - \underbrace{\int_\Omega \nabla(u - v_h) \cdot \nabla v_h \, d\Omega}_{= 0 \text{ by (7.24)}} \\
&= \textstyle\int_\Omega \nabla(u - u_h) \cdot \nabla(u - v_h) \, d\Omega \leq |u - u_h|_V |u - v_h|_V,
\end{aligned} \tag{7.25}$$

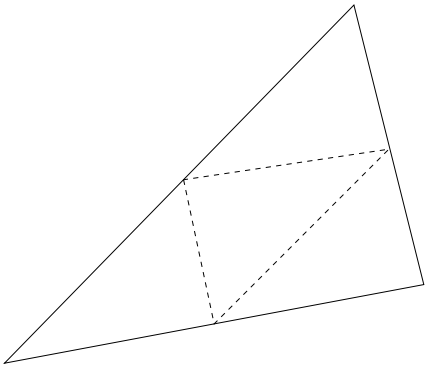


Figure 7.12: A strategy to maintain mesh quality is to subdivide each triangle into four new triangles by joining the edge midpoints.

where the last inequality follows from the Cauchy–Schwarz inequality. Dividing through with $|u - u_h|_V$ yields the conclusion. Q.E.D.

The optimality property (7.21) does not hold for all elliptic boundary-value problems. For the finite-element solution to be optimal, it is necessary that the variational problem yields a *symmetric* stiffness matrix.

The next step in an analysis of the error is a pure approximation problem. Typically, one considers the *interpolant*, that is, a piecewise-linear function agreeing with u at the node points; note that the interpolant is an element of V_h. The difference between the interpolant and u can be estimated by a type of Taylor expansion. From Theorem 21 follows that the error in the finite-element solution smaller or equal to the error in the interpolant. The precise magnitude of this error depends of course on how fine the mesh is, but it also depends on the *quality* of the mesh. Loosely speaking, one should avoid very thin triangles.

Altogether, estimating the interpolation error and utilising Theorem 21, it can be shown that the error in the finite-element solution is of *second order*, that is,

$$|u_h - u|_{L^2(\Omega)} = O(h^2). \tag{7.26}$$

Note that the norm above is not the energy norm; the error is of *first order* if measured in the energy norm. For estimate (7.26) to hold, assumptions have to be made on the mesh quality and on the smoothness of the solution to the variational problem (7.8). Following conditions are sufficient.

(i) (Mesh quality.) The smallest angle of any of the triangles is bounded below as the mesh is refined. This means that no triangle successively can become infinitely thin.

(ii) (Smoothness.) The boundary of Ω is smooth. Alternatively, the boundary is polygonal and the domain is convex. (If Ω is not polygonal to start with, it is typically approximated with a succession of polygonal domains Ω_h such that $\Omega_h \to \Omega$ as $h \to 0$).

The mesh quality condition above is maintained if the triangles, as the mesh is refined, are subdivided into four triangles in the way indicated in figure 7.12. Refining each triangle in the mesh in this way reduces all triangular sides with a factor 1/2. The error will thus be reduced with a factor 1/4 (for problems on convex domains at least).

Higher accuracy can thus be obtained through refinement of the mesh ("h method"). This should preferably be done *adaptively*, in the parts of the domain where it is needed, to prevent the size of the stiffness matrix to become too large. There are automatic methods for this. Higher accuracy can also be obtained through higher order on the polynomials on each triangle ("p method"). For instance, the error in the sense (7.26) can be improved to *third order* if V_h consists of continuous functions that are *quadratic* on each element.

7.11 Alternative Elements

Quadrilaterals, that is, a geometric figure obtained by connecting four points in the plane by straight lines, can be used to partition the domain instead of triangles, see figure 7.13. In this case will the approximating space V_h contain globally continuous functions who vary linearly along the edges of each quadrilateral. However, the functions will no longer be linear *within* the elements. In the special case when the quadrilaterals are rectangles oriented in the coordinate directions, a function $v_h \in V_h$ will be *bilinear*, that is, of the form

$$v_h(x, y) = a + bx + cy + dxy$$

on each element. The nodal values of v_h (the values of v_h at the four corners of the rectangle) uniquely determine the four coefficients above.

Quadrilaterals and, in particular, rectangular elements yields a regular structure that may give high solution accuracy and allow efficient solutions of the associated linear systems. It is, however, harder to generate such meshes automatically on complicated geometries compared to triangular meshes.

For three space dimensions, triangular and quadrilateral meshes generalise to *tetrahedral* and *hexahedral* meshes (figure 7.14) with advantages and limitations as for corresponding meshes in two space dimensions.

For higher order equations, such as the Euler-Bernoulli model of a bending beam

$$y''''(x) = -f(x)$$

or the biharmonic equation

$$\Delta^2 u = 0$$

it is necessary to use *smoother* elements for which the derivatives are continuous continuous across element boundaries. So, for example, in the one-dimensional beam bending problem it is possible to define $v_h \in V_h$ as *Hermite* piecewise cubic polynomials:

$$\left.\begin{array}{lcl} \phi_0(s) & = & (s-1)^2(2s+1) \\ \phi_1(s) & = & s^2(3-2s) \\ \varphi_0(s) & = & (s-1)^2 s \\ \varphi_1(s) & = & s^2(s-1) \end{array}\right\}$$

to define a cubic polynomial, $p(s)$, $s \in [0,1]$ that interpolates $v(0)$, $\frac{dv(0)}{ds}$, $v(1)$ and $\frac{dv(1)}{ds}$ can be written as

$$p(s) = v(0)\phi_0(s) + v(1)\phi_1(s) + \left(\frac{dv(0)}{ds}\right)\varphi_0(s) + \left(\frac{dv(1)}{ds}\right)\varphi_1(s).$$

The nodal values of v_h are the values of the solution at the nodes and the values of the derivative at the node.

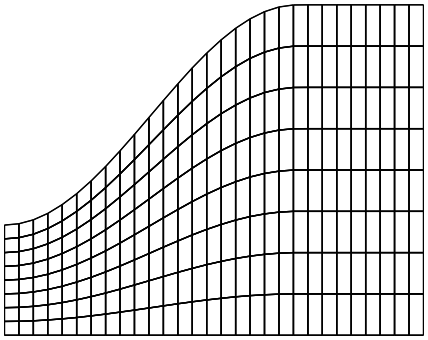


Figure 7.13: A quadrilateral mesh.

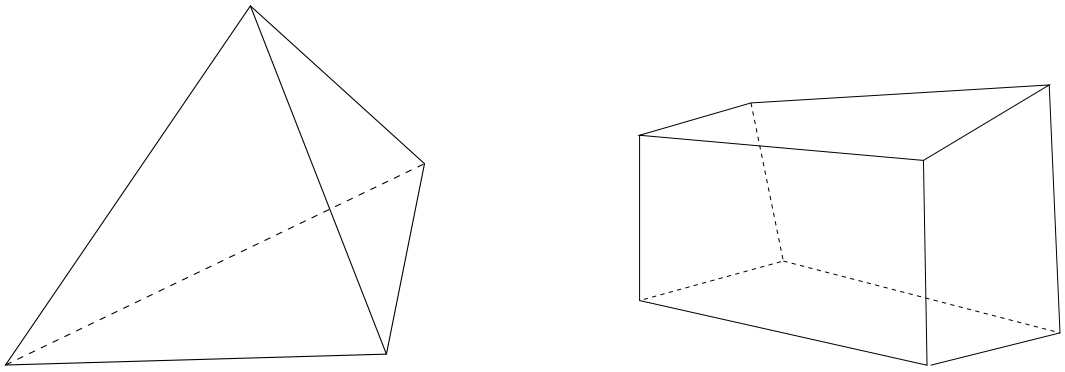


Figure 7.14: Meshes in three space dimensions can be composed of non-overlapping tetrahedra (left) or hexahedra (right).

Chapter 8

Iterative Solution of Simultaneous Equations

8.1 Banded Systems

Matrices that arise in practice from applications such as Finite Element Analysis are typically **very large**, a matrix of size 10^6 is not uncommon, but the coefficient matrix may have no more than 100 nonzeros in any row, *i.e.* over 99.9% of the matrix is zero. In such cases Guass Elimination involes a lot of wasted arithmetic *i.e.* $0 - 0 \times 0$. Such matrices typically have properties (Symmetric, Positive Definite, Sparse) that make them readily solvable, if the appropriate methods are used.

Definition 22

- *Matrices with a large number of zero elements are* Sparse Matrices
- *Band Matrices are particularly simple sparse matrices, with all the non-zeros near the main diagonal,*

$$\max_{(i)}\{i - \min_{j}\{a_{ij} \neq 0\}\}$$

is the semi-bandwidth*,* i.e. *the maximum distance along a row from the diagonal to a non-zero component.*

- *Gauss Elimination of Sparse matrices leads to fill-in, that is components that are zero in A are not zero in $L + U$*

In general when solving banded systems, by Gauss Elimination, zeros within the band are *filled in*, but no computation is necessary outside the band of non-zeros, so for matrices with a small and constant bandwidths (*e.g.* figure 8.1) for which no pivoting is required (*e.g.* symmetric, positive definite matrices), in particular tridiagonal matrices (semi-bandwidth = 1), banded GE is highly efficient. Any form of pivoting interchanges the rows and hence modifies the structure of the matrix, in particular it changes the value of the semi-bandwidth. A banded GE algorithm for an $n \times N$ matrix positive definite matrix (for which no pivoting is required) with semi-bandwidth M can be written as in figure 8.2. So the computational cost is $\mathcal{O}(NM^2)$ and not $\mathcal{O}(N^3)$.

8.2 Jacobi iteration

Example 9 *To solve*

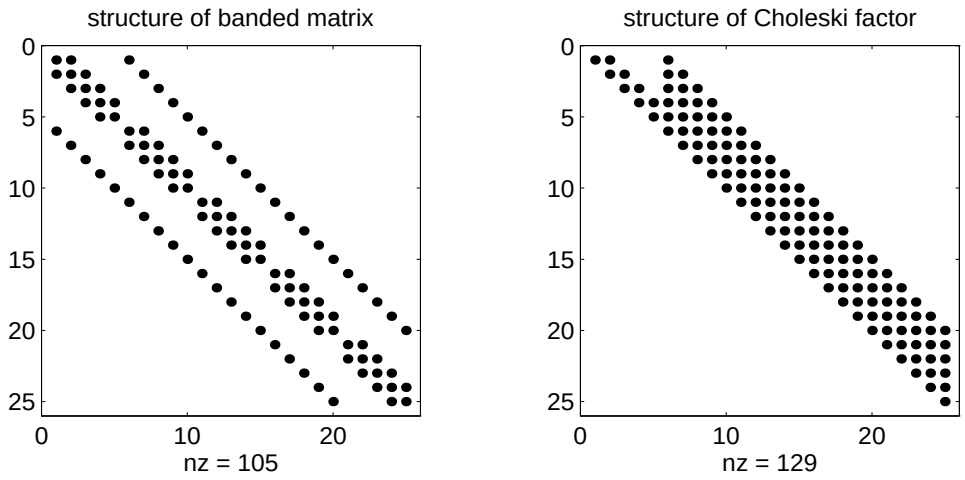


Figure 8.1: Non-zeros in a small sparse matrix, from a 2D discretisation problem

do $k = 1 : N - 1$
  do $i = k + 1 : \mathbf{min}\{k + M, N\}$
    $m_{ik} = a_{ik}/a_{kk}$
    do $j = k + 1 : \mathbf{min}\{k + M, N\}$
      $a_{ij} = a_{ij} - m_{ik}a_{kj}$
    enddo j
    $b_i = b_i - m_{ik}b_k$
  enddo i
enddo k

Figure 8.2: Banded Elimination

$$
\begin{array}{rcrcrcrcl}
10 & x_1 & - & x_2 & + & 2 & x_3 & & = 6 \\
- & x_1 & + & 11 & x_2 & - & x_3 & + & 3 & x_4 = 25 \\
2 & x_1 & - & x_2 & + & 10 & x_3 & - & x_4 = -11 \\
 & & & 3x_2 & - & x_3 & + & 8 & x_4 = 15
\end{array}
$$

(the solution is $\boldsymbol{x} = \begin{pmatrix} 1 & 2 & -1 & 1 \end{pmatrix}^T$*) write the equations as*

$$
\begin{aligned}
x_1 &= \tfrac{1}{10}(6 + x_2 - 2x_3) \\
x_2 &= \tfrac{1}{11}(25 + x_1 + x_3 - 3x_4) \\
x_3 &= \tfrac{1}{10}(-11 - 2x_1 + x_2 + x_4) \\
x_4 &= \tfrac{1}{8}(15 - 3x_2 + x_3)
\end{aligned}
$$

Starting from $\boldsymbol{x}^{(0)} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T$ *we generate a sequence of approximate solutions* $\boldsymbol{x}^{(n)}, n = 0, 1, 2, \cdots$ *such that*

$$\begin{array}{rcl} x_1^{(n+1)} & = & \frac{1}{10}(6 + x_2^{(n)} - 2x_3^{(n)}) \\ x_2^{(n+1)} & = & \frac{1}{11}(25 + x_1^{(n)} + x_3^{(n)} - 3x_4^{(n)}) \\ x_3^{(n+1)} & = & \frac{1}{10}(-11 - 2x_1^{(n)} + x_2^{(n)} + x_4^{(n)}) \\ x_4^{(n+1)} & = & \frac{1}{8}(15 - 3x_2^{(n)} + x_3^{(n)}) \end{array} \tag{8.1}$$

Hence

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},\ \boldsymbol{x}^{(1)} = \begin{pmatrix} 0.6 \\ 2.2727 \\ -1.1000 \\ 1.875 \end{pmatrix},\ \boldsymbol{x}^{(2)} = \begin{pmatrix} 1.0473 \\ 1.17159 \\ -0.8052 \\ 0.8852 \end{pmatrix},\ \boldsymbol{x}^{(3)} = \begin{pmatrix} 0.9326 \\ 2.0533 \\ -1.0493 \\ 1.1309 \end{pmatrix},\ \cdots$$

$$\cdots,\ \boldsymbol{x}^{(8)} = \begin{pmatrix} 1.0006 \\ 1.9987 \\ -0.9990 \\ 0.9989 \end{pmatrix},\ \boldsymbol{x}^{(9)} = \begin{pmatrix} .9997 \\ 2.0004 \\ -1.0004 \\ 1.0006 \end{pmatrix}$$

The method in the example is known as *Jacobi* iteration, in order to solve $A\boldsymbol{x} = \boldsymbol{b}$ we partition the matrix as

$$A \ = \ D \ + \ (A - D)$$

where the matrix D contains only the diagonal components of A, hence

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{iN} \\ \vdots & & \vdots & & \vdots \\ a_{N1} & \cdots & a_{Ni} & \cdots & a_{NN} \end{pmatrix},\ D = \begin{pmatrix} a_{11} & & & & \\ & \ddots & & & \\ & & a_{ii} & & \\ & & & \ddots & \\ & & & & a_{NN} \end{pmatrix}$$

and

$$A - D = \begin{pmatrix} 0 & \cdots & a_{1i} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i1} & \cdots & 0 & \cdots & a_{iN} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{Ni} & \cdots & 0 \end{pmatrix}$$

We can then write

$$A\boldsymbol{x} = \boldsymbol{b}$$

as

$$D\boldsymbol{x} = (D - A)\boldsymbol{x} + \boldsymbol{b}.$$

Since D is a diagonal matrix with components $a_{ii},\ i = 1, \cdots, N$, it follows that D^{-1} is a diagonal matrix with components $\frac{1}{a_{ii}},\ i = 1, \cdots, N$ and hence the rearrangement in the example corresponds to

$$\boldsymbol{x} = D^{-1}\left((D - A)\boldsymbol{x} + \boldsymbol{b}\right)$$

and the iteration can be written as

$$\boldsymbol{x}^{(n+1)} = D^{-1}\left((D - A)\boldsymbol{x}^{(n)} + \boldsymbol{b}\right)$$

In general iterative schemes can be written as

$$\boldsymbol{x}^{(n+1)} = M\boldsymbol{x}^{(n)} + \boldsymbol{g}$$

for Jacobi we have $M \equiv D^{-1}(D - A)$ and $\boldsymbol{g} \equiv D^{-1}\boldsymbol{b}$. Note there are two Jacobi Iterative methods the second method is for computing eigenvalues.

Frequently the matrix A is partitioned as $A = D + L + U$ where L and U denote the lower and upper triangle terms respectively, this partition is not to be confused with the LU factorisation defined earlier, here with

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{iN} \\ \vdots & & \vdots & & \vdots \\ a_{N1} & \cdots & a_{Ni} & \cdots & a_{NN} \end{pmatrix}$$

then

$$L = \begin{pmatrix} 0 & & & & \\ \vdots & \ddots & & & \\ a_{i1} & \cdots & 0 & & \\ \vdots & & \vdots & \ddots & \\ a_{N1} & \cdots & a_{Ni} & \cdots & 0 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 0 & \cdots & a_{1i} & \cdots & a_{1N} \\ & \ddots & \vdots & & \vdots \\ & & 0 & \cdots & a_{iN} \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix}$$

Then

$$\boldsymbol{x}^{(n+1)} = D^{-1}\left(-(L+U)\boldsymbol{x}^{(n)} + \boldsymbol{b}\right)$$

As with the power method, we generate a sequence of approximate solution and we need to know when to stop the sequence, say when the components of of the difference $\boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)}$ are sufficiently small *e.g.*

$$|x_i^{(n+1)} - x_i^{(n)}| \leq 0.001\,|x_i^{(n)}| \qquad i = 1, 2, \cdots, N$$

Iterative methods do not always work, *i.e.* the sequence $\boldsymbol{x}^{(n)},\ n = 0, 1, \cdots$ does not always converge to the solution of $A\boldsymbol{x} = \boldsymbol{b}$, the sequence can diverge (it cannot converge to the wrong answer) and so an additional termination condition such as $n \geq 100$ is essential. If all the eigenvalues of the matrix M are less than 1 in modulus, the method will converge. In the case of the Jacobi iteration a more simple interpretation is that if A is *diagonally dominant* the method works, *i.e.* if

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}|.$$

In the example 9

$$\begin{array}{rl} 10 & \geq 1 + 2 + 0 \\ 11 & \geq 1 + 1 + 3 \\ 10 & \geq 2 + 1 + 1 \\ 8 & \geq 0 + 3 + 1 \end{array}$$

or

$$M = \begin{pmatrix} 0 & \frac{1}{10} & -\frac{1}{5} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{1}{5} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{pmatrix}$$

which has eigenvalues -.4264, .1860, -.1040 and .3445 (found using Matlab). This explains why the error in each component in the above example oscillates in sign (the dominant eigenvalue is negative).

Note that it is the matrix components that determine whether the iteration converges, the right hand side vector $\boldsymbol{b}$ does not affect the convergence.

Example 10 *bad example*

*If we had written the equations of the example 9 in a different order (*i.e. *interchanging equations 2 and 3) and solved them as*

$$\begin{aligned} x_1 &= \tfrac{1}{10}(6 + x_2 - 2x_3) \\ x_2 &= 11 + 2x_1 + 10x_3 - x_4 \\ x_3 &= -25 - x_1 + 11x_2 + 3x_4 \\ x_4 &= \tfrac{1}{8}(15 - 3x_2 + x_3) \end{aligned}$$

we would have obtained

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ \boldsymbol{x}^{(1)} = \begin{pmatrix} 0.6 \\ 11 \\ -25 \\ 1.875 \end{pmatrix}, \ \boldsymbol{x}^{(2)} = \begin{pmatrix} 6.7 \\ -239.6 \\ 101.0 \\ -5.37 \end{pmatrix}, \ \boldsymbol{x}^{(3)} = \begin{pmatrix} -43.57 \\ 1040.0 \\ -2684.2 \\ 104.4 \end{pmatrix}$$

8.3 Gauss-Seidel iteration

In the Jacobi iteration (8.1) we updated all the values using only $x_i^{(n)}, \ i = 1, 2, \cdots, N$ *i.e.* we did not make use of (say) the new value of x_1 when computing the new value of x_2 and so on. We could use the iteration

$$\begin{aligned} x_1^{(n+1)} &= \tfrac{1}{10}(6 + x_2^{(n)} - 2x_3^{(n)}) \\ x_2^{(n+1)} &= \tfrac{1}{11}(25 + x_1^{(n+1)} + x_3^{(n)} - 3x_4^{(n)}) \\ x_3^{(n+1)} &= \tfrac{1}{10}(-11 - 2x_1^{(n+1)} + x_2^{(n+1)} + x_4^{(n)}) \\ x_4^{(n+1)} &= \tfrac{1}{8}(15 - 3x_2^{(n+1)} + x_3^{(n+1)}) \end{aligned} \tag{8.2}$$

then we have

$$\boldsymbol{x}^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ \boldsymbol{x}^{(1)} = \begin{pmatrix} 0.6 \\ 2.3227 \\ -.9873 \\ .8789 \end{pmatrix}, \ \boldsymbol{x}^{(2)} = \begin{pmatrix} 1.030 \\ 2.037 \\ -1.014 \\ 0.9844 \end{pmatrix},$$

$$\boldsymbol{x}^{(3)} = \begin{pmatrix} 1.0065 \\ 2.0036 \\ -1.0025 \\ .9983 \end{pmatrix}, \ \boldsymbol{x}^{(4)} = \begin{pmatrix} 1.0009 \\ 2.0003 \\ -1.0003 \\ .9999 \end{pmatrix}$$

This method is known as *Gauss-Seidel* iteration and, when it works, usually converges faster than the Jacobi iteration. In terms of the matrix splitting $A = L + D + U$ the Gauss-Seidel iteration can be written as:

$$\boldsymbol{x}^{(n+1)} = D^{-1}\left(-L\boldsymbol{x}^{(n+1)} - U\boldsymbol{x}^{(n)} + \boldsymbol{b}\right)$$

or

$$\boldsymbol{x}^{(n+1)} = (D+L)^{-1}\left(-U\boldsymbol{x}^{(n)} + \boldsymbol{b}\right)$$

In general the speed of convergence depends on the eigenvalues of the iteration matrix M since if

$$\boldsymbol{x}^{(n+1)} \;=\; M\boldsymbol{x}^{(n)} \;+\; \boldsymbol{g} \tag{8.3}$$

is the iteration, then the exact solution satisfies

$$\boldsymbol{x} \;=\; M\boldsymbol{x} \;+\; \boldsymbol{g}$$

and the error $\boldsymbol{e}^{(n)} \equiv \boldsymbol{x}^{(n)} - \boldsymbol{x}$ satisfies

$$\boldsymbol{e}^{(n+1)} = M\boldsymbol{e}^{(n)} \tag{8.4}$$

which is the formula used in the power method. Again for a bad ordering (with the largest eigenvalue of the iteration matrix M being greater than 1 in modulus), the error cannot decrease and the iteration diverges.

8.4 Successive Overrelaxation:SOR

In the Gauss-Seidel iteration (8.2) we could use the iteration in the form

$$\begin{array}{rcl}
x_1^{(n+1)} &=& x_1^{(n)} + \frac{1}{10}(6 - 10x_1^{(n)} + x_2^{(n)} - 2x_3^{(n)}) \\
x_2^{(n+1)} &=& x_2^{(n)} + \frac{1}{11}(25 + x_1^{(n+1)} - 11x_2^{(n)} + x_3^{(n)} - 3x_4^{(n)}) \\
x_3^{(n+1)} &=& x_3^{(n)} + \frac{1}{10}(-11 - 2x_1^{(n+1)} + x_2^{(n+1)} - 10x_3^{(n)} + x_4^{(n)}) \\
x_4^{(n+1)} &=& x_4^{(n)} + \frac{1}{8}(15 - 3x_2^{(n+1)} + x_3^{(n+1)} - 8x_4^{(n)})
\end{array} \tag{8.5}$$

This is known as the *correction form* and can be written as

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + D^{-1}\boldsymbol{r}$$

where the *residual* $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$ is evaluated component-by-component using the most recent values so

$$r_i = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n+1)} - \sum_{j=i}^{N} a_{ij}x_j^{(n)}$$

The SOR can be written as

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \omega D^{-1}\boldsymbol{r}$$

where ω is the relaxation parameter, $\omega > 1$ is over relaxation. In term of the partition of A the iteration can be written as

$$\boldsymbol{x}^{(n+1)} = (D + \omega L)^{-1}\left((1-\omega)D\boldsymbol{x}^{(n)} - \omega U\boldsymbol{x}^{(n)} + \omega\boldsymbol{b}\right)$$

The Jacobi iteration can also be written in correction form as

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + D^{-1}\boldsymbol{r}^{(n)}$$

8.5 Extrapolation

Given the form of the error equation (8.4) for linear iterative methods, eventually the error vector $\boldsymbol{e}^{(n)}$ will approximate an eigenvector of the iteration matrix M, but more importantly, the error equation (8.4) can be approximated by

$$\boldsymbol{e}^{(n+1)} \approx \lambda \boldsymbol{e}^{(n)} \tag{8.6}$$

hence

$$\boldsymbol{x}^{(n)} = \boldsymbol{e}^{(n)} + \boldsymbol{x}$$

and

$$\begin{aligned} \boldsymbol{x}^{(n+1)} = & \quad \boldsymbol{e}^{(n+1)} + \boldsymbol{x} \\ \approx & \quad \lambda \boldsymbol{e}^{(n)} + \boldsymbol{x} \end{aligned}$$

so eliminating $\boldsymbol{e}^{(n)}$ gives

$$\boldsymbol{x} \approx \frac{1}{1-\lambda}(\boldsymbol{x}^{(n+1)} - \lambda \boldsymbol{x}^{(n)}) \tag{8.7}$$

The formula (8.7) can be used to be compute a more accurate approximate solution iff we can estimate the value of λ. But we can replace n by $n-1$ in (8.7) thus

$$\boldsymbol{x} \approx \frac{1}{1-\lambda}(\boldsymbol{x}^{(n)} - \lambda \boldsymbol{x}^{(n-1)})$$

and hence

$$\boldsymbol{x}^{(n+1)} - \lambda \boldsymbol{x}^{(n)} \approx \boldsymbol{x}^{(n)} - \lambda \boldsymbol{x}^{(n-1)}$$

so

$$\boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)} \approx \lambda(\boldsymbol{x}^{(n)} - \boldsymbol{x}^{(n-1)})$$

and defining $\boldsymbol{\delta}^{(n)} \equiv \boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)}$ gives the approximation

$$\tilde{\lambda} = \frac{{\boldsymbol{\delta}^{(n)}}^T \boldsymbol{\delta}^{(n-1)}}{{\boldsymbol{\delta}^{(n-1)}}^T \boldsymbol{\delta}^{(n-1)}} \tag{8.8}$$

substituting in (8.7) leads to an improved approximation

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^{(n+1)} - \frac{\tilde{\lambda}}{1-\tilde{\lambda}} \boldsymbol{\delta}^{(n)} \tag{8.9}$$

Chapter 9

Large Linear Systems

9.1 Theory of Gradient Methods

For a more general sparse matrix, that does not have a narrow regular band structure GE is not efficient and it is necessary to use iteration. Unfortunately, simple iteration such as Jacobi or Gauss-Seidel converge rapidly for matrices that are strongly diagonally dominant ($|a_{ii}| \gg \sum_{j \neq i} |a_{ij}|$) and matrices from (say) finite element analysis are not strongly diagonally dominant, so alternatives methods that convergence faster are required. To cut down on the subscripts/superscripts in this chapter we adopt the following simplifying notation: $\boldsymbol{x} \equiv \boldsymbol{x}^{(i)}$ is the current approximation and $\boldsymbol{x}_+ \equiv \boldsymbol{x}^{(i+1)}$ is the next iterate, *etc.*., and the exact solution is $\boldsymbol{x}^*$.

At each iteration, given a *search direction* $\boldsymbol{p}$, then the approximate solution is updated as

$$\boldsymbol{x}_+ = \boldsymbol{x} \ + \ \alpha \boldsymbol{p} \tag{9.1}$$

where α is computed by a local minimisation that will be explained below. The residual $\boldsymbol{r} = \boldsymbol{b} \ - \ A\boldsymbol{x}$ is also updated rather than computed explicitly and so from (9.1) multiplying by A

$$A\boldsymbol{x}_+ = A\boldsymbol{x} \ + \ \alpha A\boldsymbol{p}$$

and it follows that

$$\boldsymbol{r}_+ = \boldsymbol{r} \ - \ \alpha A\boldsymbol{p} \tag{9.2}$$

If the matrix A is symmetric and positive definite then the quadratic form $\boldsymbol{x}^T A\boldsymbol{x}$ can be used to define a norm for the vector $\boldsymbol{x}$ as

$$\|\boldsymbol{x}\|_A = \left(\boldsymbol{x}^T A\boldsymbol{x}\right)^{1/2}.$$

When A is symmetric positive definite then A^{-1} is also symmetric positive definite and hence the quadratic form $\boldsymbol{x}^T A^{-1}\boldsymbol{x}$ also defines a norm. For *conjugate gradients*, the at each iteration the parameter α is computed by local minimisation of the residual in terms of such a norm, *i.e.*

$$\min_{\alpha} \ \boldsymbol{r}_+^T A^{-1}\boldsymbol{r}_+ = \min_{\alpha} \ (\boldsymbol{b} \ - \ A\boldsymbol{x}_+)^T A^{-1}(\boldsymbol{b} \ - \ A\boldsymbol{x}_+)$$

where as $A = A^T$ and so $\boldsymbol{x}^T A\boldsymbol{y} = \boldsymbol{y}^T A\boldsymbol{x}$ and from the update formula (9.2)

$$\begin{aligned} \boldsymbol{r}_+^T A^{-1}\boldsymbol{r}_+ \ &= (\boldsymbol{r} \ - \ \alpha A\boldsymbol{p})^T A^{-1}(\boldsymbol{r} \ - \ \alpha A\boldsymbol{p}) \\ &= \boldsymbol{r}^T A^{-1}\boldsymbol{r} \ - \ 2\alpha \boldsymbol{p}^T \boldsymbol{r} \ + \ \alpha^2 \boldsymbol{p}^T A\boldsymbol{p} \end{aligned}$$

the gradient is zero at minimum hence differentiating w.r.t. α gives

$$\alpha = \frac{(\boldsymbol{b} \ - \ A\boldsymbol{x})^T \boldsymbol{p}}{\boldsymbol{p}^T A \boldsymbol{p}} = \frac{\boldsymbol{r}^T \boldsymbol{p}}{\boldsymbol{p}^T A \boldsymbol{p}} \tag{9.3}$$

The *error* can be written in terms of the *residual* as $\boldsymbol{e} = \boldsymbol{x}^* - \boldsymbol{x} = A^{-1}\boldsymbol{r}$ and the norm to be minimised can be rewritten as

$$\boldsymbol{r}_+^T A^{-1} \boldsymbol{r}_+ = \boldsymbol{e}_+^T A \boldsymbol{e}_+$$

Thus conjugate gradients requires

$$\min_{\alpha} \ \boldsymbol{e}_+^T A \boldsymbol{e}_+.$$

For any symmetric positive definite matrix B other methods are derived with

$$\min_{\alpha} \ \boldsymbol{e}_+^T B \boldsymbol{e}_+$$

i.e. minimising any error *norm*

$$\|\boldsymbol{e}\|_B = \left(\boldsymbol{e}^T B \boldsymbol{e}\right)^{1/2}.$$

9.1.1 Computing the Search Direction

At each iteration of conjugate gradients we must compute the value of α (9.3), update the approximation (9.1), update the residual (9.2) but first it is necessary to compute the new search direction.

At each step define the matrix P in which the columns are the search directions, so $P_+ = \begin{bmatrix} P & \boldsymbol{p}_+ \end{bmatrix}$. Similarly define R in which the columns are the residuals, so $R_+ = \begin{bmatrix} R & \boldsymbol{r} \end{bmatrix}$. Then span$\{P\}$ as the space spanned by the search directions. If we require that $\boldsymbol{x}_+$ is also optimal in the whole space spanned by the search directions so far, *i.e.* a global minimum not just a local minimum, then writing $\boldsymbol{x}$ as a linear combination of all the search directions gives

$$\boldsymbol{x}_+ = P\boldsymbol{a} \ + \ \alpha \boldsymbol{p}$$

where the coefficients $\boldsymbol{a}$ and α solve the minimisation problem

$$\min_{\boldsymbol{a},\,\alpha} \boldsymbol{r}_+^T A^{-1} \boldsymbol{r}_+ = \min_{\boldsymbol{a},\,\alpha} \ (\boldsymbol{b} \ - \ AP\boldsymbol{a} \ - \ \alpha A\boldsymbol{p})^T A^{-1} (\boldsymbol{b} \ - \ AP\boldsymbol{a} \ - \ \alpha A\boldsymbol{p})$$

then expanding the quadratic form,

$$\begin{aligned} \boldsymbol{r}_+^T A^{-1} \boldsymbol{r}_+ = \ & (\boldsymbol{b} \ - \ AP\boldsymbol{a} \ - \ \alpha A\boldsymbol{p})^T A^{-1} (\boldsymbol{b} \ - \ AP\boldsymbol{a} \ - \ \alpha A\boldsymbol{p}) \\ = \ & \boldsymbol{b}^T A^{-1} \boldsymbol{b} \ - \ 2\alpha \boldsymbol{p}^T \boldsymbol{b} \ + \ \alpha^2 \boldsymbol{p}^T A \boldsymbol{p} \\ & - \ 2\boldsymbol{a}^T P^T \boldsymbol{b} \ + \ \boldsymbol{a}^T P^T A P \boldsymbol{a} \\ & + \ 2\alpha \boldsymbol{a}^T P^T A \boldsymbol{p} \end{aligned}$$

The two minimisations, w.r.t. α and w.r.t. $\boldsymbol{a}$ are uncoupled iff

$$\boldsymbol{a}^T P^T A \boldsymbol{p} \alpha = 0$$

as α and $\boldsymbol{a}$ are arbitrary this condition becomes

$$P^T A\boldsymbol{p} = 0 \tag{9.4}$$

that is, the search directions are *conjugate* (or A-*orthogonal*),

$$\boldsymbol{p}^T A\boldsymbol{q} = 0 \qquad \text{for any } \boldsymbol{p} \neq \boldsymbol{q}$$

The matrix A is symmetric positive definite so we can interpret $(\boldsymbol{x}^T A\boldsymbol{x})^{1/2}$ as a norm and hence $p^T Aq$ as an inner product.

9.1.2 Convergence

The minimisation w.r.t. α leads to (9.3) and the other minimisation

$$\min_{\boldsymbol{a}} \ (\boldsymbol{b} \ - \ AP\boldsymbol{a})^T A^{-1}(\boldsymbol{b} \ - \ AP\boldsymbol{a})$$

after differentiating leads to

$$P^T(\boldsymbol{b} \ - \ AP\boldsymbol{a}) = P^T\boldsymbol{r} = 0. \tag{9.5}$$

If in addition,

$$P_+^T\boldsymbol{r} = \boldsymbol{p}_+^T\boldsymbol{b} - \boldsymbol{p}_+^T AP\boldsymbol{a} = \boldsymbol{p}_+^T\boldsymbol{b} = 0$$

then $\boldsymbol{b} \in \text{span}\{AP\}$ and

$$\begin{aligned} & \boldsymbol{x}^* = A^{-1}\boldsymbol{b} \in \text{span}\{P\} \\ \Rightarrow \quad & \boldsymbol{x}^* = \boldsymbol{x} = PA\boldsymbol{a} \\ \Rightarrow \quad & \boldsymbol{r} = 0 \end{aligned}$$

so that either the iteration has converged or $\boldsymbol{p}^T\boldsymbol{r} \neq 0$.

Given that $\boldsymbol{p}^{(1)} = \boldsymbol{r}^{(0)} = \boldsymbol{b}$ it can be proved by induction from (9.4) and (9.5) that

$$\text{span}\{R\} = \text{span}\{P_+\}.$$

Thus there exists a nonsingular triangular matrix S corresponding to the change in basis $R = P_+S$, so from (9.5),

$$P^T\boldsymbol{r}_+ = 0 = R^T\boldsymbol{r}_+ \tag{9.6}$$

and the *residuals are orthogonal*, *i.e.* $\boldsymbol{r}^T\boldsymbol{s} = 0$.

9.1.3 Recurrence Relation for the search direction

We assume that the update for $\boldsymbol{p}$ can be written in the form

$$\boldsymbol{p}_+ = \boldsymbol{r}_+ \ + \ \beta\boldsymbol{p} \tag{9.7}$$

The recurrence (9.2) for the residual is

$$\boldsymbol{r}_+ = \boldsymbol{r} \ - \ \alpha A\boldsymbol{p}$$

Reducing subscript values in (9.7) gives

$$\boldsymbol{r} = \boldsymbol{p} \ - \ \beta_-\boldsymbol{p}_-$$

then substituting for $\boldsymbol{r}$

$$\boldsymbol{r}_+ = \boldsymbol{p} \; - \; \beta_- \boldsymbol{p}_- \; - \; \alpha A \boldsymbol{p}$$

and substituting for $\boldsymbol{r}_+$ using the original (9.7)

$$\boldsymbol{p}_+ = \; - \; \alpha A \boldsymbol{p} \; + \; (1+\beta)\boldsymbol{p} \; - \; \beta_- \boldsymbol{p}_-,$$

this is a *3-term recurrence relation* for updating the search direction and it is sometimes used as an alternative to (9.7) in the computation. From (9.7) the conjugacy condition $\boldsymbol{p}_+^T A \boldsymbol{p} = 0$ gives

$$\beta = - \frac{\boldsymbol{r}_+^T A \boldsymbol{p}}{\boldsymbol{p}^T A \boldsymbol{p}} \tag{9.8}$$

The update formulae (9.2) and (9.8) involve 3 different inner products so the efficiency of the method can be improved using the orthogonality of the residuals $\boldsymbol{r}_+^T \boldsymbol{r} = 0$ with

$$\boldsymbol{r}_+ = \boldsymbol{r} \; - \; \alpha A \boldsymbol{p}$$

gives

$$\boldsymbol{r}_+^T \boldsymbol{r}_+ = \; - \; \alpha \boldsymbol{r}_+^T A \boldsymbol{p}.$$

Similarly $\boldsymbol{r}_+^T \boldsymbol{p} = 0$ and so from (9.7)

$$\boldsymbol{r}_+^T \boldsymbol{p}_+ = \boldsymbol{r}_+^T \boldsymbol{r}_+$$

and from (9.2)

$$\boldsymbol{r}^T \boldsymbol{r} = \alpha \boldsymbol{p}^T A \boldsymbol{p}.$$

So (9.8) becomes

$$\beta = - \frac{\boldsymbol{r}_+^T \boldsymbol{r}_+}{\boldsymbol{r}^T \boldsymbol{r}} \tag{9.9}$$

with (9.2) becoming

$$\alpha = \frac{\boldsymbol{r}^T \boldsymbol{r}}{\boldsymbol{p}^T A \boldsymbol{p}}. \tag{9.10}$$

Hence with these modifications to the computations of the scalars **the Basic CG Algorithm** becomes as shown in figure 9.1.

9.2 Preconditioning

The key to rapid convergence of iterative methods is the preconditioning, that is in order to solve $A\boldsymbol{x} = \boldsymbol{b}$, then the iteration is applied to the system

$$\tilde{A}\boldsymbol{x} = \tilde{\boldsymbol{b}}$$

where $\tilde{A} = M^{-1}A$ and $\tilde{\boldsymbol{b}} = M^{-1}\boldsymbol{b}$. Alternatively with $M = M_L M_R$ solve

$$M_L^{-1} A M_R^{-1} \boldsymbol{y} = M_L^{-1} \boldsymbol{b}.$$

The two solutions are connected by $M_R \boldsymbol{x} = \boldsymbol{y}$. If the matrix $M_L^{-1} A M_R^{-1}$ is to be symmetric, when A is symmetric, then $M_L = M_R^T$.
The matrix $M = M_L M_R$ is an approximation to A

$\boldsymbol{x}^{(0)} = 0;$ $\quad \boldsymbol{r}^{(0)} = \boldsymbol{p}^{(0)} = \boldsymbol{b}$

$i = 0;$ $\quad \rho^{(0)} = \boldsymbol{r}^{(0)^T}\boldsymbol{r}^{(0)}$

while not converged do

$$\begin{aligned}
\boldsymbol{v}^{(i)} &= A\boldsymbol{p}^{(i)} \\
\alpha &= \rho^{(i)}/\boldsymbol{p}^{(i)^T}\boldsymbol{v}^{(i)} \\
\boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + \alpha\boldsymbol{p}^{(i)} \\
\boldsymbol{r}^{(i+1)} &= \boldsymbol{r}^{(i)} - \alpha\boldsymbol{v}^{(i)} \\
\rho^{(i+1)} &= \boldsymbol{r}^{(i+1)^T}\boldsymbol{r}^{(i+1)} \\
\beta &= \rho^{(i+1)}/\rho^{(i)} \\
\boldsymbol{p}^{(i+1)} &= \boldsymbol{r}^{(i+1)} + \beta\boldsymbol{p}^{(i)} \\
i &= i+1
\end{aligned}$$

enddo

Figure 9.1: Basic Conjugate Gradients

9.2.1 Preconditioned Conjugate Gradients (PCG)

The algorithm for PCG requires one linear solve

$$M_L M_R \boldsymbol{z} = \boldsymbol{r}$$

per iteration, the conjugacy conditions become

$$\begin{aligned}
\boldsymbol{r}^{(j)^T} M_R^{-1} M_L^{-1} \boldsymbol{r}^{(i)} &= 0 \\
\boldsymbol{p}^{(j)^T} M_R^{-1} A M_L^{-1} \boldsymbol{p}^{(i)} &= 0
\end{aligned}$$

The derivation is straightforward with the substitutions

$$\begin{aligned}
\boldsymbol{r} &\rightarrow M_L\boldsymbol{r} \\
\boldsymbol{v} &\rightarrow M_L\boldsymbol{v} \\
\boldsymbol{x} &\rightarrow M_R^{-1}\boldsymbol{x} \\
\boldsymbol{p} &\rightarrow M_R^{-1}\boldsymbol{p}
\end{aligned}$$

The PCG Algorithm can be written as in figure 9.2. The choice of a good preconditioner can have a dramatic effect upon the rate of convergence. Popular choices are

$\boldsymbol{x}^{(0)} = 0;$ $\quad \boldsymbol{r}^{(0)} = \boldsymbol{b}$

$i = 0;$ $\quad \boldsymbol{z}^{(0)} = \boldsymbol{p}^{(0)} = M_R^{-1} M_L^{-1} \boldsymbol{b} \qquad \rho^{(0)} = \boldsymbol{r}^{(0)^T} \boldsymbol{z}^{(0)}$

while not converged do

$$
\begin{aligned}
\boldsymbol{v}^{(i)} &= A\boldsymbol{p}^{(i)} \\
\alpha &= \rho^{(i)} / \boldsymbol{p}^{(i)^T} \boldsymbol{v}^{(i)} \\
\boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + \alpha \boldsymbol{p}^{(i)} \\
\boldsymbol{r}^{(i+1)} &= \boldsymbol{r}^{(i)} - \alpha \boldsymbol{v}^{(i)} \\
\boldsymbol{z}^{(i+1)} &= M_R^{-1} M_L^{-1} \boldsymbol{r}^{(i+1)} \qquad \textit{ie} \text{ Solve } M_L M_R \boldsymbol{z}^{(i+1)} = \boldsymbol{r}^{(i+1)} \\
\rho^{(i+1)} &= \boldsymbol{r}^{(i+1)^T} \boldsymbol{z}^{(i+1)} \\
\beta &= \rho^{(i+1)} / \rho^{(i)} \\
\boldsymbol{p}^{(i+1)} &= \boldsymbol{z}^{(i+1)} + \beta \boldsymbol{p}^{(i)} \\
i &= i + 1
\end{aligned}
$$

enddo

Figure 9.2: Preconditioned Conjugate Gradients

- Diagonal Preconditioning

$$M = \text{diag}(A) \text{ so } M_L = M_R = M^{1/2}$$

- Incomplete Cholesky Factorisation
  - By position: *e.g.* ICCG(0), the *position* of the nonzeros in M is governed by the *position* of the nonzeros in A.
  - By value: Components of M are nonzero if component of L in $A = LL^T$ is large enough

Chapter 10

Monte Carlo Methods

Introduction

The problem is to compute an approximation to

$$I(f) = \int_{\Omega} f(x)dx$$

where the region $\Omega = I^d$, the unit hypercube in $\mathbb{R}^d$. The classical Monte Carlo method is to evaluate the function f at n points that are chosen at random in Ω so that

$$I(f) \approx I_n(f) = \frac{|\Omega|}{n} \sum_{k=1}^{n} f(x_k)$$

Monte Carlo methods have become popular in the financial calculations, for example collateralized mortgage obligations, involving very high dimensional integrals *e.g.* 30 year mortgages implies $d = 360$.

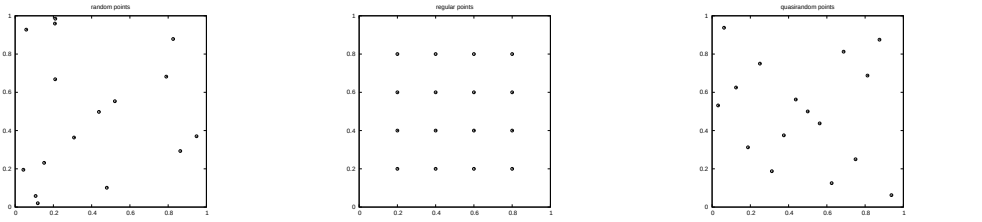
Low Discrepancy Sequences

The problem with random numbers is - they are random - they can cluster in some regions and leave gaps in other regions (see figure 10.1(a)). However a completely regular pattern such as figure 10.1(b) has serious drawbacks if depends strongly on one variable, say $f(x, y) \approx f(x)$, as there are essentially only 4 distinct points at which evaluate the function. A structured, but nonuniform, pattern gives a better coverage (see figure 10.1(b)). Matlab versions of the quasirandom number generator, [1], [5], are available from the website [2].

Pseudo Random Number Generators

The numbers used in figure 10.1(a) are not random. Computers generate *pseudorandom numbers* which are generated by a deterministic algorithm, but which appear to be random, *i.e.* there is no identifiable sequence in either the short term or long term and the numers are uniformily distributed over the given range. The most common algorithm is a *l*inear congruence generator,

$$X_{n+1} = (aX_n + c) \text{mod} m \tag{10.1}$$



(a) Psuedorandom Ordering (b) Regular Ordering (c) Quasirandom Ordering

Figure 10.1: Distributing Points

(this is read as X_{n+1} is cogruent to $aX_n + c$ modulo m) where a, $0 \leq a < m$ is the *multiplier*, c, $0 \leq c < m$ is the *increment* and X_0 is the *seed*. For example $a = 11$, $b = 0$ and $m = 101$ with $X_0 = 17$ gives

- $X_1 = 11 \times 17 \text{mod} 101 = 187 \text{mod} 101 = 86$
- $X_2 = 11 \times 86 \text{mod} 101 = 946 \text{mod} 101 = 37$
- The series is $17,\ 86,\ 37,\ 3,\ 33,\ 60,\ 54,\ \ldots$

The period is m if c an m are relatively prime and $a - 1$ is divisible by all the prime factors of m. The generator is extremely sensitive to the choice of m, a and c. The choice of a = 16807, c = 0, m = 2147483647 is a very good set of parameters for this generator. These parameters were published in [3]. This generator often known as the minimal standard random number generator. For more information on this and the more recent *Mersenne Twister* type of generator consult Wikipedia [7]. Often a sequence is normalised to lie in the range $[0, 1]$, in the sequence above this is by dividing by 101 to give

$$0.16832,\ 0.85148,\ 0.36634,\ 0.02970,\ 0.32673,\ 0.59406,\ 0.53465,\ \ldots$$

Example 11 *To compute the area of a quadrant of the unit circle* $\int_0^1 \sqrt{1 - x^2} dx$.

- *Hit-or-miss Monte Carlo*
  Construct a sequence of n cordinate pairs $0 \leq x, y \leq 1$ *compute the number k that lie inside the circle, this gives the ratio* k/n *as an approximation of the area of the quadrant to the area of the unit square. In two examples with* $n = 1000$, $k = 809$ *with* $n = 10000$, $k = 7847$ *with* $\pi/4 = 0.78540$.
- *Crude Monte Carlo*
  Construct sequence of x values evaluate the height $y = \sqrt{1 - x^2}$ *estimate quadrant as a rectangle with the mean height* $\overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$, *with the same sequences of x-values as above with* $n = 1000$, $\overline{y} = 0.79177$ *and with* $n = 10000$ $\overline{y} = 0.78522$.

The pseudo random numbers generated by the normalised (10.1) correspond to a *uniform distribution* on [0,1]. If any other distribution is required (*e.g.* exponential or normal) then a simple transformation is required.

Theorem 23 The Central Limit Theorem
As the the sample size increases, the distribution of the sample mean tends to a normal distribution even if the original population is not normally distributed.

In the example above, the sample size in N and the sample mean is n/N, the sampled values are either 0 or 1. The mean of n numbers $x_1, x_2, \ldots, x_n$ is the *sample mean* $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. The *population mean* is denoted by μ. The *sample variance* is $s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \overline{x}^2$, the *population variance* is denoted by σ^2. From the central limit theorem it follows that for large n, the sample mean is from a normal distribution:

- The mean of the distribution of sample means for samples of size n is the population mean μ.
- The variance of the distribution of sample means for samples of size n is σ^2/n where σ^2 is the population variance

It is then possible to use properties of the normal distribution to compute *confidence intervals* for the approximate value. For a normal distribution it is possible to find an interval about the mean such that (say) 95% of all values lie within the interval, his interval is known as the 95% *confidence interval*. It can be shown that

$$P(\mu - 1.96\sigma \leq X \leq \mu + 1.96\sigma) = 0.95$$

different confidence intervals:

90	95	99	99.9
1.645	1.960	2.326	3.291

The confidence interval is estimated using the sample mean and variance as

$$\left[\overline{x} - \frac{1.96s}{\sqrt{n}}, \overline{x} + \frac{1.96s}{\sqrt{n}}\right]$$

Example 11 continued *To compute the area of a quadrant of the unit circle* $\int_0^1 \sqrt{1-x^2}dx$.

- *Hit-or-miss Monte Carlo*
  With $n = 1000$, $s^2 = 0.15452$ *so* $s = 0.39309$ *and the 95% confidence interval is [0.78464, 0.83336],* $n = 10000$, $s^2 = 0.16895$ *so* $s = 0.41103$ *and the 95% confidence interval is [0.77664, 0.79276].*
- *Crude Monte Carlo*
  With $n = 1000$, $s^2 = 0.22517$ *so* $s = 0.050701$ *and the 95% confidence interval is [0.77782, 0.80573],* $n = 10000$, $s^2 = 0.050064$ *so* $s = 0.22375$ *and the 95% confidence interval is [0.78083, 0.7896].*

Reducing the Variance

In the example it was clear that a significant increase the size of the sample and hence increase the computation does not necessarily significantly decrease the variance and hence reduce the confidence interval. There are two standard techniques for reducing the variance.

Extracting the regular part
Write the integrand as $f = g + h$ where g can be integrated analytically and where h exhibits little variation.

Example 11 continued *To compute the area of a quadrant of the unit circle* $\int_0^1 \sqrt{1-x^2}dx$. *Write* $\sqrt{1-x^2} = 1 - x^2 + (\sqrt{1-x^2} - 1 + x^2) = g + h$, *then* $\int_0^1 1 - x^2 dx = \frac{2}{3}$ *and using the same sequence of x values as before, with* $N = 1000$ $\overline{h} + \int g = 0.78084$, $s^2 = 0.0074034$ *and the confidence interval is [0.7755, 0.78617], with* $N = 10000$ $\overline{h} + \int g = 0.78525$, $s^2 = 0.0078776$ *and the confidence interval is [0.78351, 0.78699].*

Antithetic Variables
The variance can be reduced by choosing two related sets of random numbers, computing two estimates of the integral and then combining them in a particular way. These sets of random number are x_i and $1 - x_i$. The variable $1 - x_i$ is the *antithetic variable* to x_i. So $\int f dx = \int (f_1 + f_2) dx$ with $\int f_1 dx$ estimated using x_i and $\int f_2 dx$ estimated using $1 - x_i$ so $f_1(x) = \frac{1}{2} f(x)$ and $f_2(x) = \frac{1}{2} f(1-x)$ and for each x_i compute $f_3(x_i) = \frac{1}{2} f(x_i) + \frac{1}{2} f(1 - x_i)$ and the corresponding sample mean and sample variance.

Example 11 continued *To compute the area of a quadrant of the unit circle* $\int_0^1 \sqrt{1-x^2} dx$. *Write* $f_1 = \sqrt{1-x^2}$ *and* $f_2 = \sqrt{1-(1-x)^2}$, *using the same sequence of x values as before, with* $N = 1000$ $\overline{f_1} + \overline{f_2} = 0.78715$, $s^2 = 0.0068569$ *and the confidence interval is [0.78202,, 0.79228], with* $N = 10000$ $\overline{f_1} + \overline{f_2} = 0.78492$, $s^2 = 0.0068149$ *and the confidence interval is [0.78331, 0.78654].*

Low Discrepancy Sequences

Given a set X of points $\boldsymbol{x}^k,\ k = 1, \ldots, N$ in the hypercube $I^d \subset \mathbb{R}^d$.

Definition 24 *Counting function: For any* $G \subset I^d$ *define the counting function* $S_N(G)$ *as the number of points of* X *in* G.

For any $\boldsymbol{x} = (x_1, \cdots, x_d)^T \in I^d$, define

$$G_{\boldsymbol{x}} = [0, x_1) \times \cdots \times [0, x_d)$$

Definition 25 *Discrepancy: The discrepancy of set* X *is*

$$D^*(X) = \sup_{\boldsymbol{x} \in I^d} |S_N(G_{\boldsymbol{x}}) - N x_1 \cdots x_d|$$

This discrepancy measures how much the clusters and the gaps differ from the "mean" density of points.

Definition 26 *Primitive Polynomial:*

$$P \equiv x^d + a_1 x^{d-1} \cdots + a_{d-1} x^1 + 1$$

with $a_k \in \{0, 1\}$ *and* P *is irreducible (mod 2)*

Example 12
1. $(1 + x)(1 + x) = 1 + 2x + x^2 = (1 + x^2) mod 2$ *so* $1 + x^2$ *is reducible*
2. $(1 + x)(1 + x + x^2) = 1 + 2x + 2x^2 + x^3 = (1 + x^3) mod 2$ *so* $1 + x^3$ *is reducible*
3. $1 + x$, $1 + x + x^2$, $1 + x + x^3$ *and* $1 + x^2 + x^3$ *are all irreducible.*

Definition 27 *XOR: The operator* $\oplus$ *is the bitwise exclusive-or (bitwise addition modulo 2).*

Example 13

$$\begin{aligned} a = 13 &= 1101_2 \\ b = 9 &= 1001_2 \\ a \oplus b &= 0100_2 = 4 \end{aligned}$$

Assume that m_i is an odd integer $0 < m_i < 2^i$, and define $v_i = m_i/2^i$ then $\{m_i\}$ or equivalently $\{v_i\}$ are define by the recurrence

$$v_i = a_1 v_{i-1} \oplus a_2 v_{i-2} \cdots \oplus a_{d-1} v_{i-d+1} \oplus v_{i-d} \oplus v_{i-d}/2^d$$
$$m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \cdots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d}$$
then
$$x^n = b_1 v_1 \oplus a_b v_2 \cdots$$

where $n = (\cdots b_3 b_2 b_1)_2$. Full details can be found in [1] and for example [4].

Example 14 *Take polynomial:* $x^3 + x + 1$ *and the initial values Then*

i	1	2	3
m_i	1	3	7
v_i	0.1	0.11	0.111

Table 10.1: Direction numbers

$$\begin{aligned}
m_i &= 4m_{i-2} \oplus 8m_{i-3} \oplus m_{i-3} \\
m_4 &= 12 \oplus 8 \oplus 1 \\
&= 1100 \oplus 1000 \oplus 0001 \\
&= 0101 \\
&= 5 \\
m_5 &= 28 \oplus 24 \oplus 3 \\
&= 11100 \oplus 11000 \oplus 00011 \\
&= 00111 \\
&= 7 \\
m_6 &= 20 \oplus 56 \oplus 7 \\
&= 010100 \oplus 111000 \oplus 000111 \\
&= 101011 \\
&= 43
\end{aligned}$$

Then

$$\begin{aligned}
n = 1 \quad & x^1 = v_1 = 0.1_2 \\
n = 10_2 \quad & x^2 = v_2 = 0.11_2 \\
n = 11_2 \quad & x^3 = v_1 \oplus v_2
\end{aligned}$$

Bibliography

[1] Paul Bratley and Bennett L. Fox, *Algorithm 659: implementing Sobol's quasirandom sequence generator*, ACM Transactions on Mathematical Software (TOMS) **14** (1988), 88–100.

[2] John Burkardt, *SOBOL: The Sobol Quasirandom Sequence*, `http://people.scs.fsu.edu/~burkardt/m_src/sobol/sobol.html`, [Online; accessed 31-Oct-2007].

[3] S.K. Park and K.W. Miller, *Random number generators: Good ones are hard to find*, Comm. of the ACM **31** (1988), 1192–1201.

[4] S. Paskov, *Computing high dimensional integrals with applications to finance*, Tech. Report CUCS-023-94, Department of Computer Science, Columbia University, NY, 1994.

[5] I.M. Sobol', *On the systematic search in a hypercube*, SIAM J. Numer. Anal. **16** (1979), 790–793.

[6] Wikipedia, *Floating point*, `http://en.wikipedia.org/wiki/Floating_point`, [Online; accessed 19-September-2008].

[7] ———, *Mersenne twister*, `http://en.wikipedia.org/wiki/Mersenne_Twister`, [Online; accessed 22-July-2008].

[8] Wen-Chyuan Yueh, *Eigenvalues of several tridiagonal matrices*, Applied Mathematics E-Notes (2005), 66–74, [Online; accessed 16-Oct-2008].