MATH0033: Numerical Methods

David Hewett

2020-2021

Contents

1 Introduction **4**

1.1 Course outline 5

1.2 Suggested reading 5

1.3 Assessment 6

1.4 Acknowledgements 6

2 Fundamental concepts **7**

2.1 Norms 7

2.2 Errors and convergence 9

2.3 Matrix norms 10

2.4 Spectral radius and condition number 11

2.5 Asymptotic notation 14

2.6 The mean value theorem and Taylor expansions 14

2.7 Floating point arithmetic 15

3 Nonlinear equations **17**

3.1 Examples and motivation 18

3.2 Bisection method 19

3.3 Fixed point methods 21

3.4 Newton's method 26

3.5 The secant method 29

3.6 The chord method . . . 30
3.7 Systems of nonlinear equations . . . 31
3.7.1 Simultaneous iteration . . . 32
3.7.2 Newton's method for systems . . . 34
3.8 Stopping criteria for fixed point iterations . . . 37

4 Linear systems **39**
4.1 Example and motivation . . . 39
4.2 Introduction . . . 41
4.3 Direct solvers . . . 42
4.4 Iterative methods for linear systems . . . 45
4.5 Basic stationary method and preconditioning . . . 47
4.6 Stationary Richardson method . . . 48
4.7 The Jacobi method and the Gauss-Seidel method . . . 49
4.8 Non-stationary methods . . . 51
4.8.1 The gradient method . . . 52
4.8.2 The conjugate gradient method . . . 55
4.9 Stopping criteria for iterative methods . . . 60
4.10 Computational cost . . . 61

5 Ordinary differential equations **66**
5.1 Example and motivation . . . 66
5.2 Initial value problems - the Cauchy problem . . . 67
5.3 Numerical discretization . . . 69
5.4 Discretizations from quadrature . . . 71
5.5 One-step methods . . . 72
5.5.1 Truncation error . . . 73
5.5.2 Zero stability . . . 74

5.5.3 Convergence 77
5.5.4 Absolute stability 78
5.6 Runge-Kutta methods 82
5.7 Systems of ordinary differential equations 88
5.8 Applications 89
5.9 Boundary value problems 92
5.9.1 Finite difference approximation 92
5.9.2 Shooting methods 95

Chapter 1

Introduction

Many phenomena in engineering and the physical and biological sciences can be described using mathematical models. Frequently the resulting models cannot be solved analytically, in which case a common approach is to use a "numerical" or "computational" method to find an approximate solution. *Numerical analysis* is the area of mathematics concerned with the design and analysis of such computational algorithms.

The aim of this course is to introduce the basic ideas underpinning numerical analysis, study a series of numerical methods to solve different problems, and carry out a rigorous mathematical analysis of their accuracy and stability. Such analysis is important to ensure that the methods accurately capture the desired solution. Failure to apply computational algorithms correctly can be costly, as is illustrated here:

`http://www.ima.umn.edu/~arnold/disasters/`

The overarching theme of the course is the solution of large-scale differential equations. Such problems require the solution of several subproblems and in this course we introduce numerical methods for the most important building blocks:

- solution methods for nonlinear equations and systems;
- solution methods for large linear systems;
- solution methods for ordinary differential equations.

For each method we typically ask two questions:

1. Under what circumstances is the numerical solution a good approximation of the true solution?
2. How much better does the approximation become if we are able to devote more computational resources to its calculation?

To answer these questions we will draw on standard tools from analysis including the mean value theorem, Taylor's theorem and the contraction mapping theorem.

The course assumes only knowledge of basic analysis and linear algebra. However, familiarity with the basic notions of numerical analysis, as covered for example in MATH0058 (formerly MATH7601), is of course helpful (but not indispensable), namely:

- interpolation of functions
- numerical integration
- direct solution of linear systems by factorization

Some basic knowledge of programming will also be required.

1.1 Course outline

These lecture notes consist of five chapters:

1. Introduction
2. Fundamental concepts
3. Nonlinear equations
4. Linear systems
5. Ordinary differential equations - initial and boundary value problems

1.2 Suggested reading

The suggested textbooks for the course are:

- Süli, Endre; Mayers, David F. *An Introduction to Numerical Analysis.* Cambridge University Press, Cambridge, 2003. x+433 pp. ISBN: 0-521-81026-4, 0-521-00794-1

which covers the material on nonlinear equations and ODEs, and

- Quarteroni, Alfio; Sacco, Riccardo; Saleri, Fausto. *Numerical Mathematics.* Second edition. Springer, 2007. xviii + 657 pp. ISBN: 3-540-34658-9, 978-3-540-34658-6

which covers the material on linear systems.

The computational exercises and examples use the Matlab programming language. Many basic online tutorials are available, but the following textbook may also be helpful:

- Driscoll, Tobin A. *Learning MATLAB.* SIAM, 2009.
  110 pp. ISBN: 0-898-71683-7, 978-0-898-71683-2

1.3 Assessment

The assessment will consist of:

- Homework (20%)

  For each of chapters 3-5, there will be a set of theoretical exercises and a set of computational exercises, a subset of which are to be handed in and assessed.

  The theoretical and the computational exercises carry equal weight (10% each in total).

- Unseen exam (80%)

1.4 Acknowledgements

These notes are based on a set of notes kindly supplied by Prof. Erik Burman.

Chapter 2

Fundamental concepts

2.1 Norms

In this course we shall consider numerical methods for various problems that each produce a sequence of increasingly good approximations $\mathbf{x}_n$, $n \in \mathbb{N}$, converging to the (unknown) true solution $\mathbf{x}$. The index n is usually related to the amount of computational effort involved in calculating $\mathbf{x}_n$. As numerical analysts our aim is to prove precise statements about the performance of these methods, for example determining the rate at which $\mathbf{x}_n$ converges to $\mathbf{x}$ as $n \to \infty$.

Typically, the solutions $\mathbf{x}$ and $\mathbf{x}_n$ lie in a certain vector space[1]. Common examples include finite-dimensional vector spaces such as $\mathbb{R}$ or $\mathbb{R}^n$ ($n \geq 2$), and infinite-dimensional function spaces such as $C([a,b])$, the space of all continuous functions $f : [a,b] \to \mathbb{R}$. To measure the size of elements of a vector space, measure distances between elements (for example, to quantify the accuracy to which $\mathbf{x}_n$ approximates $\mathbf{x}$), and define convergence of sequences and series, we use a norm.

Definition 2.1.1 (Norms and normed vector spaces)**.** *Let V be a vector space over $\mathbb{R}$. A function $\|\cdot\| : V \to \mathbb{R}$ is a* norm *on V if*

1. *(i) $\|\mathbf{v}\| \geq 0$, $\forall \mathbf{v} \in V$ and (ii) $\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = \mathbf{0}$;*
2. *$\|\alpha\mathbf{v}\| = |\alpha|\|\mathbf{v}\|$ $\forall \alpha \in \mathbb{R}$ and $\forall \mathbf{v} \in V$;*
3. *$\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$, $\forall \mathbf{v}, \mathbf{w} \in V$.*

We then say that $(V, \|\cdot\|)$ is a normed vector space.

[1]For simplicity we consider only real vector spaces in this course. But the concepts we consider generalise easily to complex vector spaces, with only minor modifications.

The standard norm on $\mathbb{R}$ is the absolute value function, $\|x\| = |x|$ for $x \in \mathbb{R}$.

Examples of norms on $\mathbb{R}^n$ are given by the p-norm

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \text{ for } 1 \leq p < \infty, \tag{2.1}$$

where x_i denotes the ith component of the vector $\mathbf{x} \in \mathbb{R}^n$. Taking $p = 2$ in (2.1) leads to the classical Euclidean norm

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} = \sqrt{(\mathbf{x}, \mathbf{x})},$$

where[2] the inner product $(\mathbf{x}, \mathbf{y})$ is defined by $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} = \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$. In this case the Cauchy-Schwarz inequality holds:

$$|(\mathbf{x}, \mathbf{y})| \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2.$$

Observe that $p = \infty$ is excluded in the definition of the p-norm above. The *infinity-norm* or *max-norm* is defined separately by

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Each of the above norms measures the "length" of a vector $\mathbf{x} \in \mathbb{R}^n$ in a different way. (Exercise: sketch the unit ball $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| < 1\}$ for the cases $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_\infty$.)

One can define norms on the infinite-dimensional space $C([a, b])$ in an analogous way. For instance, the *infinity-norm* or *max-norm* is defined for $f \in C([a, b])$ by

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|.$$

Here the maximum is well-defined because a continuous function on a bounded interval is bounded and attains its bounds.

Definition 2.1.2 (Norm equivalence)**.** *Two norms $\|\cdot\|$ and $\|\cdot\|'$ on a vector space V are said to be* equivalent *if there exist constants $0 < c < C$ such that*

$$c\|\mathbf{v}\| \leq \|\mathbf{v}\|' \leq C\|\mathbf{v}\| \qquad \forall \mathbf{v} \in V. \tag{2.2}$$

On a finite-dimensional vector space (such as $\mathbb{R}^n$) all norms are equivalent. But this result does not in general extend to infinite-dimensional spaces such as $C([a, b])$.

[2]It is a general result that whenever one has an inner product $(\cdot, \cdot)$ defined on vector space V, one can generate a norm on V by the formula $\|\mathbf{v}\| = \sqrt{(\mathbf{v}, \mathbf{v})}$, and that the Cauchy-Schwarz inequality holds.

2.2 Errors and convergence

If $\tilde{\mathbf{x}} \in V$ is an approximation to $\mathbf{x} \in V$ we can consider the *absolute error*

$$E_{\text{abs}} = \|\tilde{\mathbf{x}} - \mathbf{x}\|$$

and the *relative error*

$$E_{\text{rel}} = \frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|}.$$

Note that if $V = \mathbb{R}$ then $-\log_{10}(E_{\text{rel}})$ indicates to how many decimal digits the approximate and exact solutions agree.

Convergence of sequences and series in a normed vector space is defined in the obvious way. For instance, we say a sequence $(\mathbf{x}_n)_{n=1}^{\infty} \subset V$ converges to $\mathbf{x} \in V$ with respect to the norm $\|\cdot\|$ if $\|\mathbf{x}_n - \mathbf{x}\| \to 0$ as $n \to \infty$ (as a sequence of real numbers)[3]. Note that if a sequence converges with respect to a given norm, then it also converges with respect to any equivalent norm.

To determine whether a numerical method is likely to be useful in a practical application, it is important to know how fast $\mathbf{x}_n$ converges to $\mathbf{x}$ as $n \to \infty$.

Definition 2.2.1. *For a sequence $(\mathbf{x}_n)$ that converges to $\mathbf{x}$ as $n \to \infty$, we say that the convergence is* linear *if there exists a constant $0 < C < 1$ such that, for n sufficiently large,*

$$\|\mathbf{x}_{n+1} - \mathbf{x}\| \leq C\|\mathbf{x}_n - \mathbf{x}\|.$$

We say that the convergence is quadratic *if there exists a constant $C > 0$ such that, for n sufficiently large,*

$$\|\mathbf{x}_{n+1} - \mathbf{x}\| \leq C\|\mathbf{x}_n - \mathbf{x}\|^2.$$

In general, we say the convergence is of order $p > 1$ *(not necessarily an integer) if there exists a constant $C > 0$ such that, for n sufficiently large,*

$$\|\mathbf{x}_{n+1} - \mathbf{x}\| \leq C\|\mathbf{x}_n - \mathbf{x}\|^p.$$

(Exercise: Explain why the condition $C < 1$ is only needed for the case $p = 1$.)

If a positive quantity $E(n)$ depending on a parameter $n \in \mathbb{N}$ (for example the error $\|\mathbf{x}_n - \mathbf{x}\|$ in a numerical approximation) behaves like $E(n) \approx Cn^{\alpha}$ for some $\alpha \in \mathbb{R}$, then

$$\log E(n) \approx \log C + \alpha \log n,$$

so plotting $\log E(n)$ against $\log n$ (e.g. using Matlab's `loglog` command) will produce a straight line with slope α. If $E(n)$ behaves like $E(n) \approx Ca^n$ for some $a > 0$, then

$$\log E(n) \approx \log C + n \log a,$$

so plotting $\log E(n)$ against n (e.g. using Matlab's `semilogy` command) will produce a straight line with slope $\log a$.

[3]Recall that a sequence $(c_n)_{n=1}^{\infty} \subset \mathbb{R}$ converges to $c \in \mathbb{R}$ as $n \to \infty$ if for every $\epsilon > 0$ there exists $N \in \mathbb{N}$ such that $n > N$ implies $|c_n - c| < \epsilon$.

2.3 Matrix norms

Definition 2.3.1 (Matrix norm)**.** *By a* matrix norm *we mean a norm on the vector space* $\mathbb{R}^{m\times n}$*. According to Definition 2.1.2 a function* $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ *is a matrix norm if*

1. *(i)* $\|A\| \geq 0$, $\forall A \in \mathbb{R}^{m\times n}$ *and (ii)* $\|A\| = 0$ *if and only if* $A = 0$*;*
2. $\|\alpha A\| = |\alpha|\|A\|$ $\forall \alpha \in \mathbb{R}$ *and* $\forall A \in \mathbb{R}^{m\times n}$*;*
3. $\|A + B\| \leq \|A\| + \|B\|$, $\forall A, B \in \mathbb{R}^{m\times n}$*.*

As for vectors, we say that a sequence of matrices $A^{(k)}$, $k = 0, 1, 2, \ldots$, converges to a matrix A, and write $\lim_{k\to\infty} A^{(k)} = A$, if $\lim_{k\to\infty} \|A^{(k)} - A\|_M = 0$ for some matrix norm $\|\cdot\|_M$. Note that since all norms on $\mathbb{R}^{m\times n}$ are equivalent, it doesn't matter which matrix norm we take.

One well-known example of a matrix norm is the *Frobenius norm*

$$\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} \qquad \text{for } A \in \mathbb{R}^{m\times n},$$

which is simply the $p = 2$ norm considered above, in the setting $V = \mathbb{R}^{m\times n}$, i.e. we are viewing the matrix as an element of Euclidean space.

But we are going to be more concerned with a different class of matrix norms, namely those defined by reference to the action of the linear map associated with matrix multiplication. For simplicity we focus on the case of square matrices $A \in \mathbb{R}^{n\times n}$, for which the linear map $T_A\mathbf{x} = A\mathbf{x}$ maps $\mathbb{R}^n$ to $\mathbb{R}^n$. Then, given a vector norm $\|\cdot\|_V$ on $\mathbb{R}^n$, we can define an *induced* (or *subordinate*) matrix norm on $\mathbb{R}^{n\times n}$ by[4]

$$\|A\|_M = \sup_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_V}{\|\mathbf{x}\|_V}, \qquad \text{for } A \in \mathbb{R}^{n\times n}. \tag{2.3}$$

Lemma 2.3.2. *Let* $\|\cdot\|_M$ *be the matrix norm* (2.3) *induced by a vector norm* $\|\cdot\|_V$*. Then*

- $\|\cdot\|_M$ *is a norm on* $\mathbb{R}^{n\times n}$*;*
- $\|A\mathbf{x}\|_V \leq \|A\|_M\|\mathbf{x}\|_V$ *(compatibility);*
- $\|I\|_M = 1$*;*
- $\|AB\|_M \leq \|A\|_M\|B\|_M$*, for all* $A, B \in \mathbb{R}^{n\times n}$*;*

Proof. The proof is left as an exercise. □

[4]For those familiar with the theory of bounded linear operators, the subordinate matrix norm $\|A\|_M$ is nothing other than the "operator norm" of the bounded linear operator $T_A : \mathbb{R}^n \to \mathbb{R}^n$, $T_A\mathbf{x} = A\mathbf{x}$.

For later use we note that the matrix norm $\|\cdot\|_\infty$ on $\mathbb{R}^{n\times n}$ induced by the supremum norm $\|\cdot\|_\infty$ on $\mathbb{R}^n$ has the following explicit form (exercise: prove this)

$$\|A\|_\infty = \max_{i\in\{1,\dots,n\}} \sum_{j=1}^{n} |a_{ij}|, \qquad A \in \mathbb{R}^{n\times n}. \tag{2.4}$$

2.4 Spectral radius and condition number

We will need the above norms in order to quantify the effect of perturbations on the solutions of linear systems obtained using different numerical methods. Two concepts will be particularly important in our analysis: the *spectral radius* of a square matrix, and its *condition number*. Both properties are related to the eigenvalues of the matrix. We denote the set of eigenvalues of a matrix $A \in \mathbb{R}^{n\times n}$ by

$$\sigma(A) = \{\lambda \in \mathbb{C} : A\mathbf{v} = \lambda\mathbf{v} \text{ for some } \mathbf{0} \neq \mathbf{v} \in \mathbb{R}^n\}.$$

Using this set we may immediately define the spectral radius of A.

Definition 2.4.1 (Spectral radius of a matrix)**.** *The* spectral radius $\rho(A)$ *of a matrix A is defined to be the largest eigenvalue of A in absolute value, i.e.*

$$\rho(A) = \max_{\lambda\in\sigma(A)} |\lambda|.$$

It should be noted that the spectral radius is a matrix "semi-norm", but *not* a matrix norm, in the sense that it satisfies all the properties of Definition 2.3.1 except for property $1(ii)$. It can, however, be viewed in a certain sense as a surrogate for a matrix norm, because of the following lemma, which we state without proof.

Lemma 2.4.2. (Relationship between spectral radius and compatible matrix norms.)

- *Let $\|\cdot\|_M$ be the matrix norm on $\mathbb{R}^{n\times n}$ induced by some vector norm $\|\cdot\|_V$ on $\mathbb{R}^n$. Then*
$$\rho(A) \leq \|A\|_M \quad \forall A \in \mathbb{R}^{n\times n}.$$
- *Given $A \in \mathbb{R}^{n\times n}$, for every $\epsilon > 0$ there exists a matrix norm $\|\cdot\|_{M,A,\epsilon}$ induced by some vector norm $\|\cdot\|_{V,A,\epsilon}$ (both depending on A and ϵ) such that*
$$\|A\|_{M,A,\epsilon} \leq \rho(A) + \epsilon.$$

This lemma gives an alternative characterisation of the spectral radius as

$$\rho(A) = \inf_{\|\cdot\|_M} \|A\|_M,$$

where the infimum is taken over the set of all matrix norms induced by some vector norm on $\mathbb{R}^n$.

For symmetric matrices and the Euclidean norm the situation is simpler.

Lemma 2.4.3. *If $A \in \mathbb{R}^{n\times n}$ is symmetric then the matrix norm $\|\cdot\|_2$ induced by the Euclidean norm $\|\cdot\|_2$ on $\mathbb{R}^n$ satisfies*

$$\|A\|_2 = \rho(A).$$

One can also relate the spectral radius to convergence of matrix powers. This will be important when we consider iterative methods for solving linear systems. In the following lemma, A^k means the product of A with itself k times, e.g. $A^2 = AA$, $A^3 = AAA$ etc., and the convergence of a sequence of matrices is understood as in the previous section, i.e. $\lim_{k\to\infty} A^k = 0$ means that $\lim_{k\to\infty} \|A^k\|_M = 0$ for some, and hence every, matrix norm $\|\cdot\|_M$.

Lemma 2.4.4. *Let $A \in \mathbb{R}^{n\times n}$. Then*

$$\lim_{k\to\infty} A^k = 0 \Leftrightarrow \rho(A) < 1. \tag{2.5}$$

Proof. For the forward implication, assume that $\lim_{k\to\infty} A^k = 0$ and that $\lambda \in \sigma(A)$ is an eigenvalue of A. Let $\mathbf{0} \neq \mathbf{v}$ be an eigenvector associated to λ, so that $A\mathbf{v} = \lambda\mathbf{v}$. Then $A^k\mathbf{v} = \lambda^k\mathbf{v}$ for each $k = 1, 2, \ldots$ and by Lemma 2.3.2 we have for any vector norm and its induced matrix norm that

$$|\lambda|^k\|\mathbf{v}\|_V = \|A^k\mathbf{v}\|_V \leq \|A^k\|_M\|\mathbf{v}\|_V \to 0 \text{ as } k \to \infty.$$

Since $\mathbf{v} \neq \mathbf{0}$, this implies that $|\lambda| < 1$, and hence that $\rho(A) < 1$.

For the reverse implication, assume that $\rho(A) < 1$. Then $\exists\epsilon > 0$ such that $\rho(A) < 1 - \epsilon$. (For example one can take $\epsilon = (1 - \rho(A))/2$.) By the second part of Lemma 2.4.2 there exists an induced matrix norm $\|\cdot\|_M$ such that $\|A\|_M \leq \rho(A) + \epsilon < 1$. But then using Lemma 2.3.2 we see that $\|A^k\|_M \leq (\|A\|_M)^k \to 0$ as $k \to \infty$, which proves that $\lim_{k\to\infty} A^k = 0$. □

We now turn to the definition of condition number.

Definition 2.4.5 (Condition number of a matrix)**.** *Let $\|\cdot\|_M$ be the matrix norm induced by a vector norm $\|\cdot\|_V$, as in (2.3). The* condition number *$K_M(A)$ of a matrix $A \in \mathbb{R}^{n\times n}$ with respect to $\|\cdot\|_M$ is defined by*

$$K_M(A) = \|A\|_M\|A^{-1}\|_M.$$

Observe that $K_M(A) \geq 1$ since $1 = \|I\|_M = \|AA^{-1}\|_M \leq \|A\|_M\|A^{-1}\|_M = K_M(A)$ and that $K_M(A^{-1}) = K_M(A)$. Note that the definition of the condition number requires the matrix to be invertible. For singular matrices the convention is that $K_M(A) = \infty$. One may show that $K_M(A)^{-1}$ measures the distance (in the $\|\cdot\|_M$ norm) from A to the closest singular matrix. If the p-norm is used in the definition of the condition number one may use the notation $K_p(A)$.

In the special case $p = 2$ there exist convenient characterizations of the condition number. For instance, suppose that A is symmetric positive definite (SPD), meaning that A is symmetric

and $\mathbf{v}^T A\mathbf{v} > 0$ for all $\mathbf{0} \neq \mathbf{v} \in \mathbb{R}^n$. An equivalent characterisation of SPD is that A is symmetric and has only positive eigenvalues, i.e. $\sigma(A) \subset (0, \infty)$. Then

$$K_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{2.6}$$

where $0 < \lambda_{\min} \leq \lambda_{\max}$ are the smallest and largest eigenvalues of A.

As we will see, the condition number of a matrix A provides an indication of how easy it is to numerically solve the linear system

$$A\mathbf{x} = \mathbf{b} \tag{2.7}$$

for $\mathbf{x} \in \mathbb{R}^n$, given $\mathbf{b} \in \mathbb{R}^n$. Matrices with $K_M(A) \approx 1$ are said to be *well-conditioned*, and those with $K_M(A) \gg 1$ are said to be *ill-conditioned*.

One particular situation in which the condition number plays a key role is in estimating the accuracy of a numerical solution by computing the residual. When solving the linear system (2.7) numerically, whichever method we use we will in general obtain a perturbed solution $\tilde{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$ satisfying

$$A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b},$$

for some perturbations $\delta\mathbf{x}$ and $\delta\mathbf{b}$. We would like to understand how the accuracy of the numerical solution (i.e. the size of $\delta\mathbf{x}$) relates to conditioning. Let's attempt to bound the size of the error $\delta\mathbf{x}$. Note that, by linearity, $\delta\mathbf{x} = A^{-1}\delta\mathbf{b}$. Then, dropping subscripts $_V$ and $_M$ on norms and condition numbers (to avoid clutter), since

$$\|\delta\mathbf{x}\| = \|A^{-1}\delta\mathbf{b}\| \leq \|A^{-1}\|\|\delta\mathbf{b}\|$$

and

$$\|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\|\|\mathbf{x}\|,$$

we have that

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = K(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Similarly we can get a lower bound on the error by observing that

$$\|\delta\mathbf{b}\| = \|A\delta\mathbf{x}\| \leq \|A\|\|\delta\mathbf{x}\|$$

and

$$\|\mathbf{x}\| = \|A^{-1}\mathbf{b}\| \leq \|A^{-1}\|\|\mathbf{b}\|,$$

which imply that

$$\frac{1}{K(A)}\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \leq \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}.$$

Combining these estimates we find that

$$\frac{1}{K(A)}\frac{\|A\tilde{\mathbf{x}} - \mathbf{b}\|}{\|\mathbf{b}\|} \leq \frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(A)\frac{\|A\tilde{\mathbf{x}} - \mathbf{b}\|}{\|\mathbf{b}\|}. \tag{2.8}$$

Hence if $K(A)$ is moderate, then the relative solution error $\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|}$ is guaranteed to be of a similar order to the relative residual $\frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} = \frac{\|A\tilde{\mathbf{x}}-\mathbf{b}\|}{\|\mathbf{b}\|}$. However, if $K(A)$ is large, then the relative residual will not necessarily give a good estimate of the relative solution error.

2.5 Asymptotic notation

We will often need to compare the growth or decay of different functions as their arguments tend to a certain limit. The following notation is very useful in this regard.

Definition 2.5.1 ("Big O" formalism)**.** *Let f and g be two functions defined on $\mathbb{R}$, and let $x_0 \in \mathbb{R}$. One writes*

$$f(x) = O(g(x)) \text{ as } x \to x_0$$

if there exist constants $\delta > 0$ and $C > 0$ such that

$$|f(x)| \leq C|g(x)| \quad \text{for } |x - x_0| < \delta.$$

Definition 2.5.2 ("Little o" formalism)**.** *Let f and g be two functions defined on $\mathbb{R}$, and let $x_0 \in \mathbb{R}$. One writes*

$$f(x) = o(g(x)) \text{ as } x \to x_0$$

if

$$\lim_{x \to x_0} \frac{f(x)}{g(x)} = 0.$$

Note that the above definitions can be generalised in an obvious way to the case where $x \to \pm\infty$.

2.6 The mean value theorem and Taylor expansions

A fundamental tool in our analysis of numerical methods will be Taylor expansion. The starting point is the mean value theorem.

Theorem 2.6.1 (Mean value theorem)**.** *Let $a, b \in \mathbb{R}$ with $a < b$, and let $f : (a, b) \to \mathbb{R}$ be differentiable*[5] *on (a, b). Then, given $x, x_0 \in (a, b)$ with $x \neq x_0$,*

$$f(x) = f(x_0) + f'(\xi)(x - x_0), \tag{2.9}$$

for some ξ between x_0 and x.

[5]Recall that $f : (a, b) \to \mathbb{R}$ is differentiable at a point $x_* \in (a, b)$ if the limit $\lim_{h \to 0}(f(x_* + h) - f(x_*))/h$ exists and is finite; this limit is then defined to be the derivative $f'(x_*)$. Differentiability on (a, b) means differentiability at every point $x_* \in (a, b)$. The standard example of a non-differentiable function is the absolute value $|x|$, which is not differentiable at $x_* = 0$ (we get a different limit for positive and negative h).

This result generalises to higher orders of differentiability.[6]

Theorem 2.6.2 (Taylor's theorem)**.** *Let $a, b \in \mathbb{R}$ with $a < b$, and let $f : (a, b) \to \mathbb{R}$ be k-times differentiable on (a, b), for some $k \in \mathbb{N}$. Then, given $x, x_0 \in (a, b)$ with $x \neq x_0$,*

$$f(x) = \sum_{j=0}^{k} \frac{f^{(j)}(x_0)}{j!}(x - x_0)^j + R_k(x; x_0), \tag{2.10}$$

where the remainder satisfies

$$R_k(x; x_0) = o(|x - x_0|^k) \quad \text{as } x \to x_0.$$

If f is $k+1$-times differentiable on (a, b) then the remainder can be expressed as

$$R_k(x; x_0) = \frac{f^{k+1}(\xi)}{k+1!}(x - x_0)^{k+1}, \tag{2.11}$$

for some ξ between x_0 and x.

2.7 Floating point arithmetic

When implementing numerical algorithms on a digital computer it is advisable to know something about how the computer stores and manipulates numbers. A computer has finite memory and cannot represent the full (uncountably infinite) set of real numbers. Instead it works with a finite set of *floating point* numbers of the form

$$f = \pm .d_1 d_2 \dots d_t \times \beta^e,$$

for some base $\beta \in \mathbb{N}$, precision $t \in \mathbb{N}$, mantissa $d_1 \dots d_t$ ($d_i \in \mathbb{N}_0$ with $0 \leq d_i < \beta$ and $d_1 \neq 0$), and exponent $e \in \mathbb{N}$, which lies in some range $L \leq e \leq U$. The numbers representable by this system satisfy

$$0 < \beta^{L-1} \leq |f| \leq \beta^U(1 - \beta^{-t}).$$

So there exists a largest floating point number, and a smallest non-zero floating point number. (Note that zero is included separately.) For an arbitrary real number x lying in the representable range, the relative error in the floating point representation $\mathrm{fl}(x)$ of x satisfies

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \leq \frac{1}{2}\beta^{1-t} =: eps;$$

[6] A function $f : (a, b) \to \mathbb{R}$ is twice differentiable if f is differentiable (in the sense of the previous footnote) and the derivative f' is also differentiable. We then define the second derivative $f'' := (f')'$. Analogously, f is k-times differentiable for some $k \in \mathbb{N}$ if the derivatives f', $f'' := (f')'$, $f''' := (f'')'$, $\dots f^{(k)} := (f^{(k-1)})'$ all exist.

eps is called the *unit round-off*. It has the property that $1 + eps$ is the next larger floating point number after 1. That is, there are no floating point numbers between 1 and $1 + eps$. If a numerical approximation achieves a relative error of size *eps* (or close to it) then one says the approximation is "correct to machine precision"; one cannot in general hope to do any better than this.

By default most modern PCs operate with *double precision arithmetic*, for which

$$\beta = 2, \quad L = -1022, \quad U = 1023, \quad t = 52.$$

This particular system is often referred to as "64-bit arithmetic", because each floating point number requires 64 "bits" (either a zero or a one) of storage: 1 for $\pm$, 52 for the mantissa, and 11 for the exponent (note that $1024 = 2^{10}$). The approximate range of representable numbers is

$$0 < 10^{-308} \leq |f| \leq 10^{308},$$

and the unit round-off is

$$eps = 2^{-52} \approx 2 \times 10^{-16}.$$

Whenever the computer carries out an arithmetic operations such as $+, -, \times, \div$ on a pair of floating point numbers, the result returned has to be rounded to the nearest floating point number. The "rounding errors" so introduced can have serious implications for the accuracy and stability of numerical methods. As a trivial example, we consider the operation of subtracting two real numbers that are very close to one another. This kind of operation may lead to a loss of significant digits in numerical computations if one is not careful. In Matlab one may illustrate this by taking

```
>> x = 1.e-15; ((1+x)-1)/x

   ans=1.1102
```

Obviously the exact solution to this operation is 1. The loss of significant digits has caused an error of 11%! (Exercise: what happens if we take x smaller than the unit round-off *eps*?)

Chapter 3

Nonlinear equations

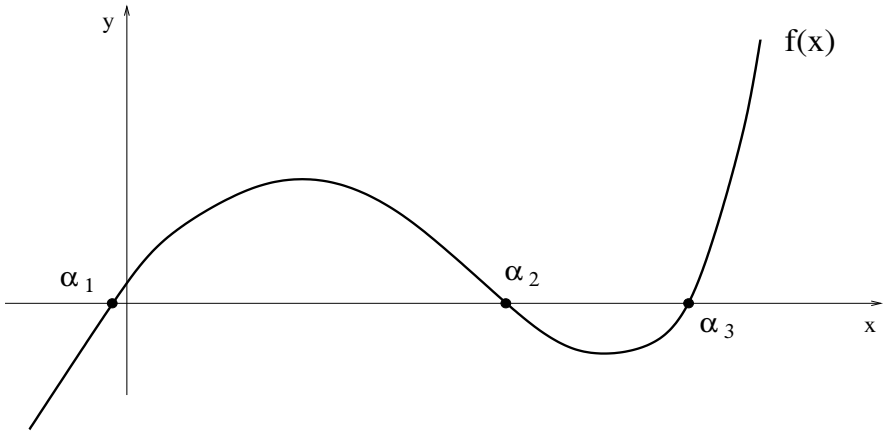
A common problem in scientific computing is to find the zeros of a function $f : \mathbb{R} \to \mathbb{R}$, i.e. the roots of the equation

$$f(x) = 0. \tag{3.1}$$

Note that any equation of the form $h(x) = g(x)$ can be written in the form (3.1) by setting $f(x) = h(x) - g(x)$. So (3.1) covers all equations involving functions of a single real variable.

For some special cases it is possible to write down explicit formulas for the roots of (3.1). But if f is a polynomial of order larger than or equal to five, then the roots cannot in general be expressed in this way. This implies that there is certainly no general explicit solution to the equation $f(x) = 0$ valid for an arbitrary continuous real valued function f.

In this chapter we will discuss iterative methods for the solution of (3.1).



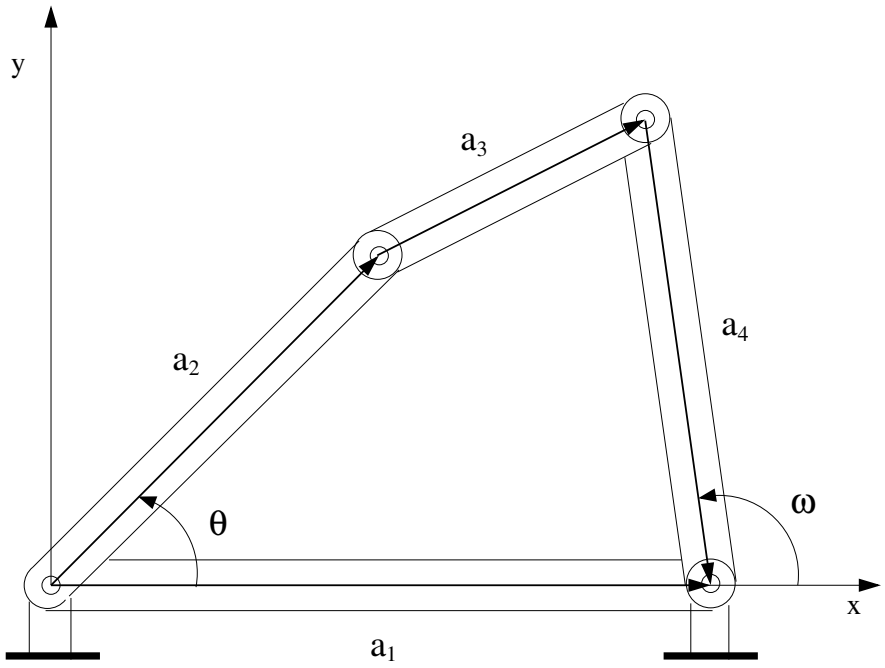


Figure 3.1: The four joined beams

3.1 Examples and motivation

Example 3.1.1 (Average interest rate)**.** *Suppose we wish to compute the average interest rate i on an investment over n years. $v = £1000$ has been invested every year and after 5 years the resulting sum is $p = £6000$.*

We know that p, v, i and the number of years n are related by the formula

$$p = v\sum_{k=1}^{n}(1+i)^k = v\frac{1+i}{i}\left[(1+i)^n - 1\right].$$

We may therefore write the problem as: find i such that

$$f(i) = p - v\frac{1+i}{i}\left[(1+i)^n - 1\right] = 0. \tag{3.2}$$

It follows that we must solve a nonlinear equation for which there is no analytical solution.

Example 3.1.2 (Configuration of beams)**.** *Consider the mechanical system of four interconnected beams shown in Figure 3.1. Given ω and the lengths a_i, $i = 1, \ldots, 4$ of the beams suppose that we wish to determine the angle θ between $\mathbf{a}_1$ and $\mathbf{a}_2$. Using the identity*

$$\mathbf{a}_1 - \mathbf{a}_2 - \mathbf{a}_3 - \mathbf{a}_4 = 0,$$

and observing that $\mathbf{a}_1$ is parallel to the x-axis, we find the following relation between ω and θ:

$$\frac{a_1}{a_2}cos(\omega) - \frac{a_1}{a_4}cos(\theta) - cos(\omega - \theta) = -\frac{a_1^2 + a_2^2 - a_3^2 + a_4^2}{2a_2a_4}. \tag{3.3}$$

The equation (3.3) can be solved analytically only for certain values of ω. In the general case the solution must be approximated numerically.

3.2 Bisection method

For continuous [1] functions f the following results of real analysis will be useful.

Theorem 3.2.1 (Intermediate value theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be continuous on $[a, b]$. Then $f(x)$ is bounded on $[a, b]$ and achieves its bounds. If y is any number such that*

$$\min_{x\in[a,b]} f(x) \leq y \leq \max_{x\in[a,b]} f(x),$$

then there exists $\alpha \in [a, b]$ such that $f(\alpha) = y$.

An immediate consequence of the intermediate value theorem is

Theorem 3.2.2. *Let $f : [a, b] \to \mathbb{R}$ be continuous on $[a, b]$, and assume that $f(a)f(b) \leq 0$. Then there exists $\alpha \in [a, b]$ such that $f(\alpha) = 0$.*

Proof. If $f(a)f(b) = 0$ then either $f(a) = 0$ or $f(b) = 0$, which yields $\alpha = a$ or $\alpha = b$ respectively. Otherwise we must have $f(a)f(b) < 0$, in which case

$$\min_{x\in[a,b]} f(x) < 0 < \max_{x\in[a,b]} f(x),$$

and the claim follows by the intermediate value theorem. □

Note that the root α may not be unique; there may be multiple roots in the interval $[a, b]$. To guarantee uniqueness one needs to impose additional constraints on f, e.g. requiring that it is differentiable on $[a, b]$ and that $f'(x) \neq 0$ for any $x \in [a, b]$.

These theoretical results naturally suggest a method for root finding for continuous functions f, called the *bisection method*. The idea is to use Theorem 3.2.2 repeatedly, looking for smaller and smaller intervals on which $f(x)$ changes sign, creating a sequence $x^0, x^1, x^2, \ldots$ converging to a root α. Suppose we have a function f satisfying the conditions of Theorem 3.2.2. Then the bisection algorithm takes the following form:

1. Set $k = 0$.
2. If $f(a)f(b) = 0$ then one of a or b must be a root, and we are done.

[1] Recall that a function $f : [a, b] \to \mathbb{R}$ is continuous at $x_* \in [a, b]$ if for every $\epsilon > 0$ there exists $\delta > 0$ such that $|x - x_*| < \delta$ implies $|f(x) - f(x_*)| < \epsilon$. Equivalently, if for every sequence $(x_n) \subset [a, b]$ converging to x_*, the sequence $(f(x_n))$ converges to $f(x_*)$. And continuity on $[a, b]$ means continuity at each point $x_* \in [a, b]$.

3. Set $x^k = \frac{a+b}{2}$.
4. If $f(x^k) = 0$ then x^k is a root and we are done.
5. If $f(x^k) \neq 0$ then either:
   - $f(x^k)f(a) > 0$, in which case $\alpha \in (x^k, b)$, so redefine $a = x^k$, increment k to $k+1$ and return to step 3.
   - $f(x^k)f(a) < 0$, in which case $\alpha \in (a, x^k)$, so redefine $b = x^k$, increment k to $k+1$ and return to step 3.

Using this type of division of the interval, or bisection, we create a sequence $x^0, x^1, x^2, \ldots$ which is guaranteed to converge to a root α (when one exists). In fact it is easy to quantify the rate of convergence, as we can write down an *a priori* error bound.

Theorem 3.2.3. *Let f satisfy the assumptions of Theorem 3.2.2. Then the iterates $x^0, x^1, x^2, \ldots$ of the bisection method converge to a root $\alpha \in [a, b]$ and for all k it holds that*

$$|x^k - \alpha| \leq \frac{b-a}{2^{k+1}}.$$

Proof. The error estimate (and hence convergence) is easily proven by observing that for each iteration the length of the interval is divided by two and that by construction a root α is always in the current interval. □

This is a major advantage of the bisection method: once an interval such that $f(a)f(b) \leq 0$ has been found, the algorithm is guaranteed to converge to a root. Another advantage is that the algorithm only uses values of the function f, and does not require (potentially costly) evaluations of derivatives of f.

The main disadvantages are that convergence is slow, and that not all roots of f can be found. (A root α can only be found using bisection if f changes sign on the interval $(\alpha - \delta, \alpha + \delta)$ for all δ sufficiently small). Also, the method does not generalise in a practical way to higher space dimensions.

Example 3.2.4. *Suppose we wish to find the root of the function $f(x) = sin(2x) - 1 + x$. Figure 3.2 shows a plot of the function f in the interval $[-3, 3]$ obtained using the following Matlab commands:*

```
>> f = @(x)sin(2*x)-1+x;
>> x=[-3:0.1:3];
>> plot(x,f(x))
>> grid on
```

Applying the method of bisection in the interval $[-1, 1]$ with a tolerance of 10^{-8}

```
>> [zero,res,niter]=bisection(f,-1,1,1e-8,1000)
```

gives the value $\alpha = 0.35228846$ after 27 iterations.

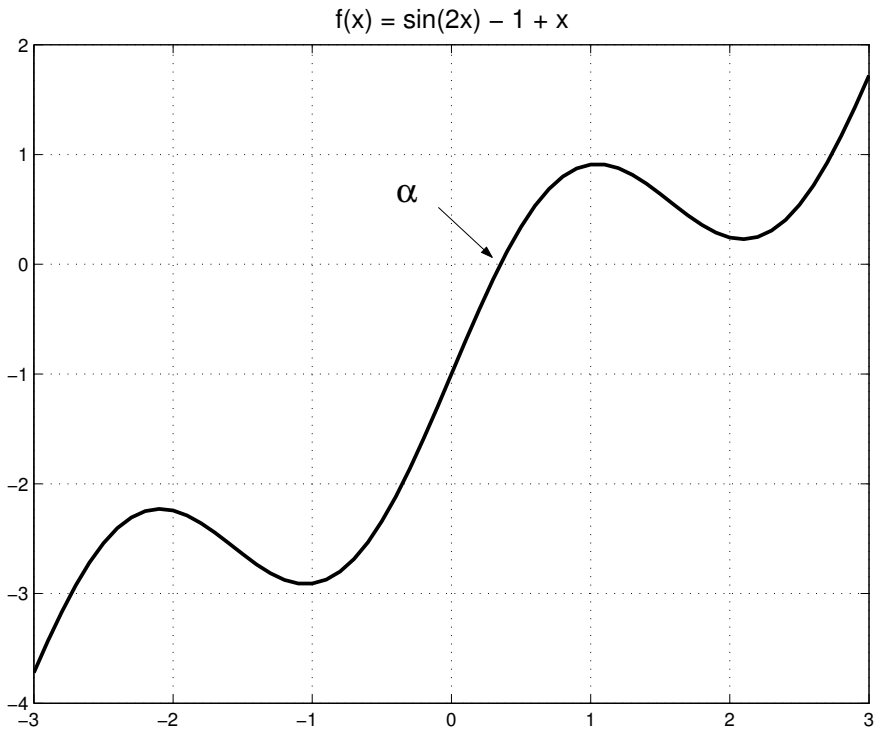


Figure 3.2: Graph of f from Example 3.2.4.

3.3 Fixed point methods

A general method to find the roots of a nonlinear equation $f(x) = 0$ is obtained by rewriting it in the form $\phi(x) = x$, for some function $\phi : [a, b] \to \mathbb{R}$ satisfying the "consistency" condition

$$\phi(\alpha) = \alpha \qquad \text{if and only if} \qquad f(\alpha) = 0. \tag{3.4}$$

The point α is called a *fixed point* of the function ϕ. It follows that to find the zeros of f we can determine the *fixed points* of ϕ. Note that one can always find a ϕ satisfying (3.4), because in particular one can take $\phi(x) = x - \beta f(x)$ for any $0 \neq \beta \in \mathbb{R}$. But other choices of ϕ may prove advantageous.

Example 3.3.1 (3.2.4, continued). *Once again consider $f(x) = \sin(2x) - 1 + x = 0$. We can recast the problem in fixed point form in (at least) two different ways:*

$$x = \phi_1(x) = 1 - \sin(2x)$$
$$x = \phi_2(x) = \frac{1}{2}\arcsin(1 - x), \qquad 0 \leq x \leq 2.$$

Figure 3.3 shows the functions ϕ_1, ϕ_2 as well as the line $y = x$. Note that ϕ_1 is of the simple form $\phi(x) = x - \beta f(x)$ mentioned above, with $\beta = 1$. But ϕ_2 is not of this form.

The following theoretical result about fixed points is a simple consequence of Theorem 3.2.2. For an illustration of the setting of the theorem consider the interval $[0, 1]$ in Figure 3.3.

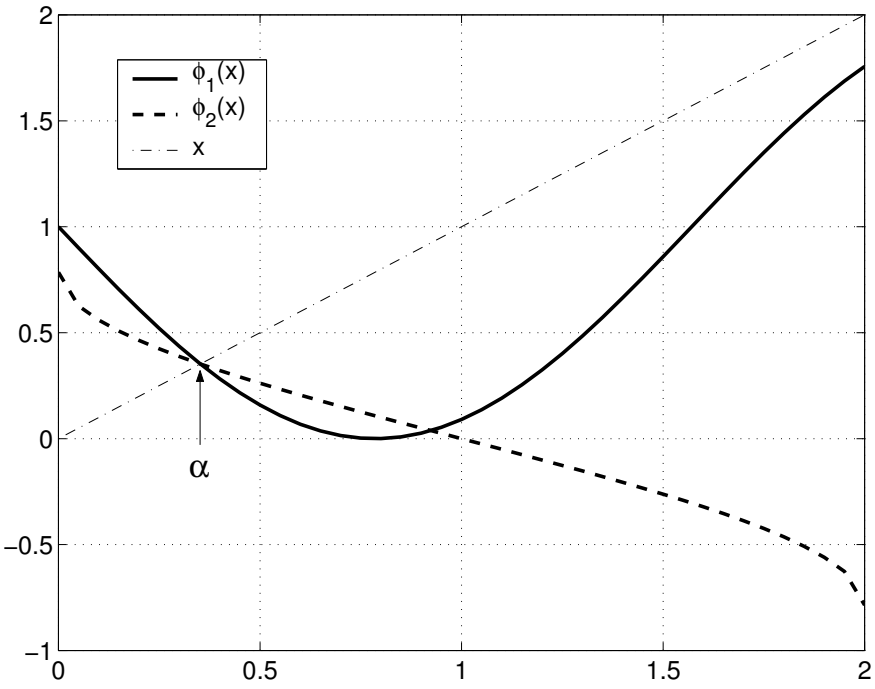


Figure 3.3: Functions ϕ_1 and ϕ_2 from Example 3.3.1.

Theorem 3.3.2 (Brouwer's fixed point theorem)**.** *Suppose that $\phi : [a, b] \to \mathbb{R}$ is continuous on $[a, b]$ and satisfies $\phi(x) \in [a, b]$ for all $x \in [a, b]$. Then there exists $\alpha \in [a, b]$ such that $\alpha = \phi(\alpha)$, i.e. α is a fixed point of ϕ.*

Proof. Let $f(x) = x - \phi(x)$. Then, since $\phi(x) \in [a, b]$ for $x \in [a, b]$, we have $f(a) = a - \phi(a) \leq 0$ and $f(b) = b - \phi(b) \geq 0$. Hence $f(a)f(b) \leq 0$, and then the existence of $\alpha \in [a, b]$ such that $f(\alpha) = 0$ and consequently $\alpha = \phi(\alpha)$ follows from Theorem 3.2.2. □

The importance of the Brouwer fixed point theorem is that it extends to higher dimensions (see Theorem 3.7.3 below). But the proof in the higher-dimensional case uses tools from topology that are beyond the scope of this course.

The fixed point formulation immediately suggests an iterative method for finding a fixed point, defined by the sequence

$$x^{k+1} = \phi(x^k), \quad k \geq 0. \tag{3.5}$$

If the resulting sequence converges to some α and ϕ is continuous then α is a fixed point. We formalise this in a lemma.

Lemma 3.3.3. *Let $\phi : \mathbb{R} \to \mathbb{R}$, let $x^0 \in \mathbb{R}$, and define a sequence $(x^k)_{k=0}^{\infty}$ by*

$$x^{k+1} = \phi(x^k), \quad k \geq 0.$$

If $x^k \to \alpha$ for $k \to \infty$ and ϕ is continuous in a neighbourhood of α then $\alpha = \phi(\alpha)$.

Proof. Using the definition of the sequence we may write

$$\alpha = \lim_{k\to\infty} x^{k+1} = \lim_{k\to\infty} \phi(x^k) = \phi\big(\lim_{k\to\infty} x^k\big) = \phi(\alpha).$$

Here the assumption that ϕ is continuous in a neighbourhood of α is what allows us to move the limit inside the argument of the function in the third equality. □

But under what conditions does the sequence obtained using (3.5) converge? This is not true for every function ϕ, but requires some additional assumptions.

Definition 3.3.4. *We say that the function $\phi : [a,b] \to \mathbb{R}$ is a strict contraction on the interval $[a,b]$ if there exists a constant $0 < \Lambda < 1$ such that*

$$|\phi(x) - \phi(x')| \le \Lambda |x - x'|, \quad \forall x, x' \in [a,b]. \tag{3.6}$$

Note that any strict contraction is necessarily a Lipschitz continuous function on $[a,b]$. A useful sufficient condition for a function to be a strict contraction is provided by the following lemma.

Lemma 3.3.5. *Let $\phi : [a,b] \to \mathbb{R}$ be differentiable on $[a,b]$, and suppose there exists a constant $0 < \Lambda < 1$ such that $|\phi'(x)| \le \Lambda$ for all $x \in [a,b]$. Then ϕ is a strict contraction on $[a,b]$.*

Proof. The mean value theorem implies that for any $x, x' \in [a,b]$ there exists ξ between x and x' such that

$$\phi(x) - \phi(x') = \phi'(\xi)(x - x').$$

The bound (3.6) then follows by taking absolute values and noting that $|\phi'(\xi)| \le \Lambda < 1$. □

Theorem 3.3.6 (Contraction mapping theorem)**.** *Let $\phi : [a,b] \to \mathbb{R}$ be a strict contraction on $[a,b]$ with contraction constant $0 < \Lambda < 1$. (In particular, by Lemma 3.3.5 this holds if ϕ is differentiable on $[a,b]$ and there exists a constant $0 < \Lambda < 1$ such that $|\phi'(x)| \le \Lambda$ for all $x \in [a,b]$.) Suppose also that $\phi(x) \in [a,b]$ for all $x \in [a,b]$. Then*

(i) ϕ admits a unique fixed point α in $[a,b]$;

(ii) $\forall x^0 \in [a,b]$ the sequence (x^k) defined by $x^{k+1} = \phi(x^k)$, $k \ge 0$, converges to α as $k \to \infty$;

(iii) The iterates satisfy the error estimate

$$|x^{k+1} - \alpha| \le \Lambda |x^k - \alpha|, \quad \forall k \ge 0. \tag{3.7}$$

Furthermore, if ϕ is continuously differentiable [2] on $[a,b]$ and $x^k \ne \alpha$ for any k then

$$\lim_{k\to\infty} \frac{x^{k+1} - \alpha}{x^k - \alpha} = \phi'(\alpha).$$

[2] A function is continuously differentiable on $[a,b]$ if f is differentiable on $[a,b]$ and the derivative f' is a continuous function.

Proof. As noted above, the assumption that ϕ is a strict contraction implies that ϕ is continuous. Hence the existence of a fixed point α follows from the Brouwer fixed point theorem (Theorem 3.3.2). (Note that one can also prove the existence of a fixed point directly from the strict contraction property, by showing that the sequence (x^k) is Cauchy. We leave this as an exercise.) To show uniqueness, suppose that $\alpha_1 \neq \alpha_2$ are two distinct fixed points. Then since ϕ is a strict contraction there exists $0 < \Lambda < 1$ such that

$$|\alpha_1 - \alpha_2| = |\phi(\alpha_1) - \phi(\alpha_2)| \leq \Lambda |\alpha_1 - \alpha_2| < |\alpha_1 - \alpha_2|.$$

But this is a contradiction, and hence the fixed point must be unique.

The convergence result and error estimate follow by similar arguments. Indeed, for each $k \geq 0$,

$$|x^{k+1} - \alpha| = |\phi(x^k) - \phi(\alpha)| \leq \Lambda |x^k - \alpha|,$$

and by induction it follows that $|x^k - \alpha| \leq \Lambda^k |x^0 - \alpha|$, so that $x^k \to \alpha$ as $k \to \infty$.

Finally, in the case when ϕ is continuously differentiable, the mean value theorem implies that, for each $k \geq 0$,

$$x^{k+1} - \alpha = \phi(x^k) - \phi(\alpha) = \phi'(\xi^k)(x^k - \alpha),$$

for some ξ^k between α and x^k. By the continuity of ϕ' and the convergence of x^k to α it follows that

$$\frac{x^{k+1} - \alpha}{x^k - \alpha} \to \phi'(\alpha) \qquad \text{as } k \to \infty.$$

□

Remark 3.3.7. *The condition "$\phi(x) \in [a, b]$ for all $x \in [a, b]$" in the hypotheses of the CMT is crucial and cannot in general be omitted. So don't forget to check it!*

The CMT provides sufficient conditions under which the fixed point iteration converges linearly, in the sense of Definition 2.2.1, for *any* choice of initial guess $x^0 \in [a, b]$. But it requires "global" assumptions on the behaviour of ϕ on the whole interval $[a, b]$. The following theorem provides a "local" convergence result which only requires a "local" assumption about ϕ.

Theorem 3.3.8 (linear convergence in a neighbourhood of α)**.** *Let $\phi : [a, b] \to \mathbb{R}$ be continuously differentiable and let $\alpha \in (a, b)$ be a fixed point of ϕ such that $|\phi'(\alpha)| < 1$. Then there exists $\delta > 0$ such that for all initial guesses $x^0 \in [\alpha - \delta, \alpha + \delta]$ the sequence (x^k) defined by $x^{k+1} = \phi(x^k)$ converges linearly to α as $k \to \infty$, with*

$$\lim_{k \to \infty} \frac{x^{k+1} - \alpha}{x^k - \alpha} = \phi'(\alpha).$$

Proof. Since $|\phi'(\alpha)| < 1$ and ϕ' is continuous, there exists $\delta > 0$ such that $|\phi'(x)| \leq \Lambda$ for all $x \in [\alpha - \delta, \alpha + \delta]$, where $0 < \Lambda = (1 + |\phi'(\alpha)|)/2 < 1$. Furthermore, ϕ maps $[\alpha - \delta, \alpha + \delta]$ to

$[\alpha-\delta, \alpha+\delta]$ because α is a fixed point of ϕ (exercise: use the MVT to check this!). Applying Theorem 3.3.6 on the interval $[\alpha-\delta, \alpha+\delta]$ we conclude that the sequence x^k converges to α as $k \to \infty$ for any initial guess $x^0 \in [\alpha-\delta, \alpha+\delta]$.

□

In Figures 3.4 and 3.5 we show in some examples how the value of $|\phi'(\alpha)|$ influences the convergence of the fixed point iteration.

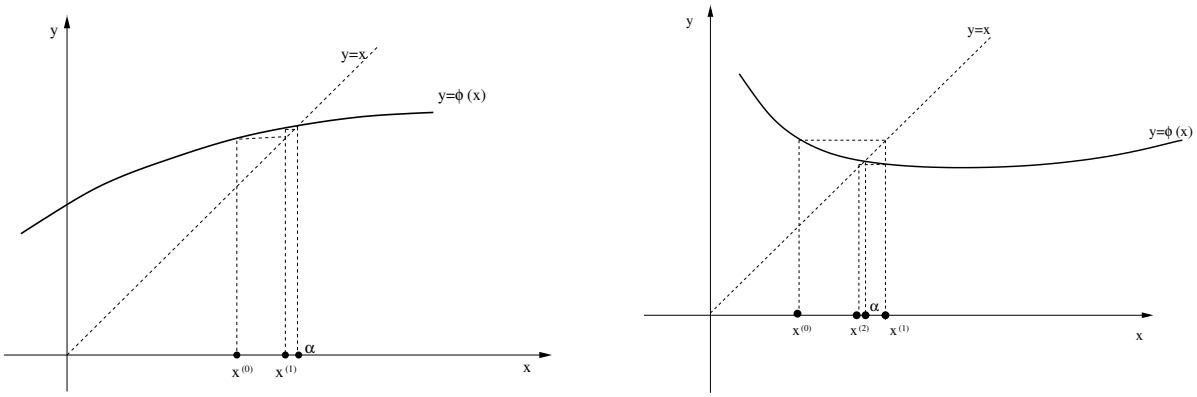


Figure 3.4: Fixed point iterations; convergent case. Left $0 < \phi'(\alpha) < 1$, right $-1 < \phi'(\alpha) < 0$.

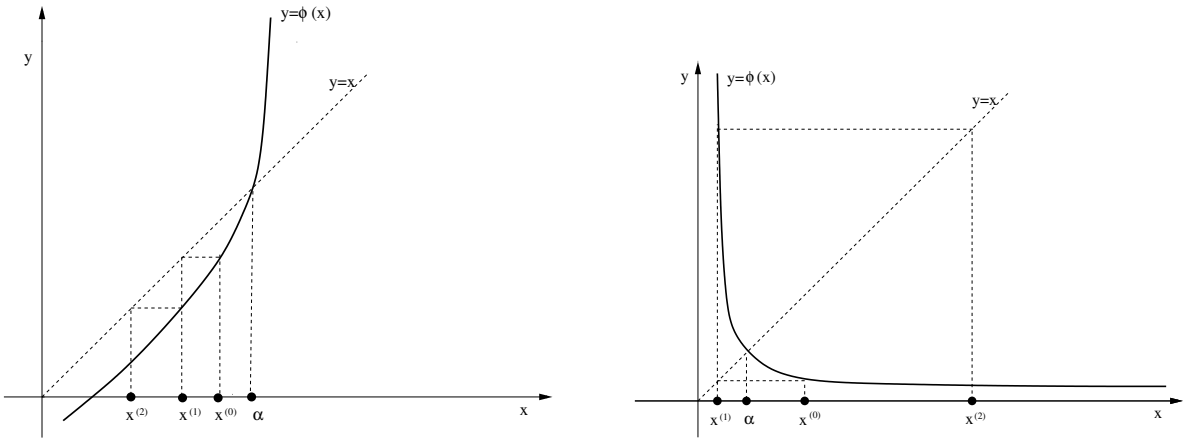


Figure 3.5: Fixed point iterations; divergent case. Left $\phi'(\alpha) > 1$, right $\phi'(\alpha) < -1$.

Example 3.3.9 (Examples 3.2.4 and 3.3.1 continued). *We apply the fixed point method to ϕ_1 and ϕ_2 starting from the initial value $x^0 = 0.7$.*

```
>>  phi1=@(x)1-sin(2*x);
>>  phi2=@(x).5*asin(1-x);
```

```
>>  [fixp,res,niter]=fixpoint(phi1,0.7,1e-8,1000)
>>  [fixp,res,niter]=fixpoint(phi2,0.7,1e-8,1000)
```

We find that the first method does not converge whereas the second converges to $\alpha = 0.35228846$ *in* 44 *iterations. Computing the derivative of the iteration functions at the root* α *we see that* $\phi_1'(\alpha) = -1.5237713$ *and* $\phi_2'(\alpha) = -0.65626645$. *So our numerical results are consistent with the result of Theorem 3.3.8.*

In light of Theorem 3.3.8, one might imagine that if $\phi'(\alpha) = 0$ the convergence should be super-linear (i.e. faster than linear). Under suitable assumptions on ϕ this is indeed the case.

Theorem 3.3.10 (quadratic convergence in a neighbourhood of α)**.** *Let* $\phi : [a, b] \to \mathbb{R}$ *be twice continuously differentiable and let* $\alpha \in (a, b)$ *be a fixed point of* ϕ *satisfying* $\phi'(\alpha) = 0$. *Then there exists* $\delta > 0$ *such that for all initial guesses* $x^0 \in [\alpha - \delta, \alpha + \delta]$ *the sequence* (x^k) *defined by* $x^{k+1} = \phi(x^k)$ *converges quadratically to* α *as* $k \to \infty$, *with*

$$\lim_{k\to\infty} \frac{x^{k+1} - \alpha}{(x^k - \alpha)^2} = \frac{\phi''(\alpha)}{2}.$$

Proof. A Taylor expansion of $\phi(x^k)$ around $x = \alpha$ gives

$$x^{k+1} - \alpha = \phi(x^k) - \phi(\alpha) = \phi'(\alpha)(x^k - \alpha) + \frac{\phi''(\xi^k)}{2}(x^k - \alpha)^2,$$

for some point ξ^k between x^k and α. Since $\phi'(\alpha) = 0$ and ϕ'' is continuous it follows that

$$\lim_{k\to\infty} \frac{x^{k+1} - \alpha}{(x^k - \alpha)^2} = \lim_{k\to\infty} \frac{\phi''(\xi^k)}{2} = \frac{\phi''(\alpha)}{2}.$$

□

We study an example of a quadratically convergent fixed point method in the next section.

3.4 Newton's method

Let us return to our original problem of finding a zero of a given function $f : \mathbb{R} \to \mathbb{R}$, i.e. a solution α of the equation $f(\alpha) = 0$. Suppose that f is continuously differentiable, and that we have an initial guess x^k. Then by the mean value theorem we know that

$$f(x^k) = f(\alpha) + f'(\xi)(x^k - \alpha)$$

for some ξ between α and x^k. Since $f(\alpha) = 0$, this implies that

$$\alpha = x^k - \frac{f(x^k)}{f'(\xi)}.$$

Unfortunately this doesn't give an explicit formula for α, because $f'(\xi)$ is unknown. However, since f' is continuous, if x^k is sufficiently close to α then $f'(\xi)$ can be well approximated by the known quantity $f'(x^k)$. This approximation suggests the iteration

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}, \qquad k = 0, 1, 2, \ldots. \tag{3.8}$$

This iterative method is known as Newton's method.

Another way of interpreting Newton's method is to notice that x^{k+1} is the unique root of the local linear approximation to f at the point x^k (i.e. the intersection of the tangent to the graph of f at the point x^k with the x axis) - see Figure 3.6. This tangent is the straight line defined by the function

$$r(x) = f'(x^k)(x - x^k) + f(x^k),$$

and solving the equation $r(x^{k+1}) = 0$ gives the Newton step (3.8).

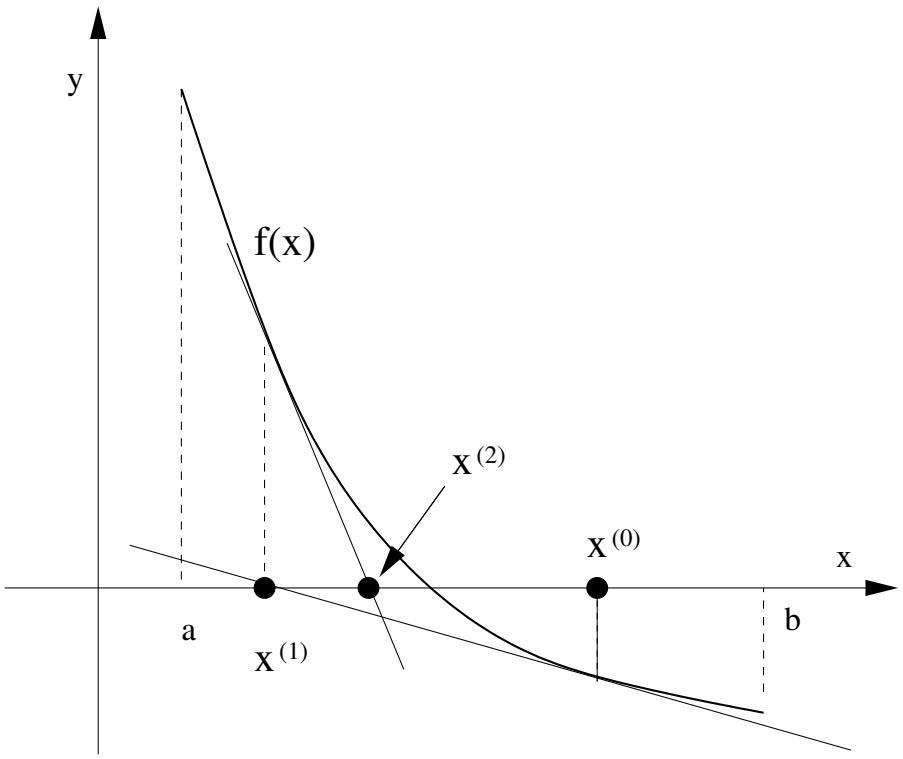


Figure 3.6: Newton's method. Starting from x^0, the sequence (x^k) converges to the root of f.

Newton's method can be written as a fixed point method $x^{k+1} = \phi(x^k)$ using the function

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

Therefore we can analyse the convergence of Newton's method using the theory of fixed point iterations. Provided that f is twice continuously differentiable then, for all x such that $f'(x) \neq 0$, ϕ is continuously differentiable at x with

$$\phi'(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2}.$$

In particular, if α is a root of f with $f(\alpha) = 0$ and $f'(\alpha) \neq 0$ then $\phi'(\alpha) = 0$. Hence by Theorem 3.3.8 the method converges for a sufficiently good initial guess x^0.

In fact the convergence is quadratic, with

$$\lim_{k\to\infty} \frac{x^{k+1} - \alpha}{(x^k - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)}. \tag{3.9}$$

If we make the extra assumption that f is *three* times continuously differentiable then we can prove this using the fixed point theory, since then ϕ is twice differentiable with $\phi'(\alpha) = 0$ (exercise: check this!), so Theorem 3.3.10 applies, and (3.9) follows from the fact that

$$\phi''(\alpha) = \frac{f''(\alpha)}{f'(\alpha)}.$$

However, this extra smoothness assumption on f is not necessary. Once we have deduced convergence using Theorem 3.3.8 we can prove the estimate (3.9) directly for f only twice differentiable, without a further appeal to the fixed point theory, as we now show.

Theorem 3.4.1. *Let $f : [a, b] \to \mathbb{R}$ be twice continuously differentiable and let $\alpha \in (a, b)$ satisfy $f(\alpha) = 0$ and $f'(\alpha) \neq 0$. Then there exists $\delta > 0$ such that for all initial guesses $x^0 \in [\alpha - \delta, \alpha + \delta]$ the Newton iteration (3.8) converges to α quadratically, and*

$$\lim_{k\to\infty} \frac{x^{k+1} - \alpha}{(x^k - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)}. \tag{3.10}$$

Proof. As already outlined above, convergence of the method follows from Theorem 3.3.8. So it remains to prove (3.10). For this, we note that the definition of the Newton iteration gives

$$x^{k+1} - \alpha = x^k - \alpha - \frac{f(x^k)}{f'(x^k)}. \tag{3.11}$$

By the smoothness assumption on f and the fact that $x^k \to \alpha$ as $k \to \infty$ we can Taylor expand both $f(x^k)$ and $f'(x^k)$ around the point α to give (recall that $f(\alpha) = 0$)

$$f(x^k) = f'(\alpha)(x^k - \alpha) + \frac{1}{2}f''(\alpha)(x^k - \alpha)^2 + o((x^k - \alpha)^2), \tag{3.12}$$

$$f'(x^k) = f'(\alpha) + f''(\alpha)(x^k - \alpha) + o(x^k - \alpha). \tag{3.13}$$

Plugging these expansions into (3.11), dividing the numerator and denominator in the fraction by $f'(\alpha)$ (which by hypothesis is non-zero), and introducing the notation $e^k = x^k - \alpha$, gives

$$e^{k+1} = e^k - e^k \frac{1 + \frac{1}{2}\lambda e^k + o(e^k)}{1 + \lambda e^k + o(e^k)},$$

where $\lambda = f''(\alpha)/f'(\alpha)$. Noting that

$$\begin{aligned}\frac{1}{1 + \lambda e^k + o(e^k)} = \frac{1}{1 + (\lambda e^k + o(e^k))} &= 1 - (\lambda e^k + o(e^k)) + (\lambda e^k + o(e^k))^2 + \dots \\ &= 1 - \lambda e^k + o(e^k),\end{aligned}$$

we see that

$$\begin{aligned}e^{k+1} &= e^k - e^k \left(1 + \frac{1}{2}\lambda e^k + o(e^k)\right)\left(1 - \lambda e^k + o(e^k)\right) \\ &= e^k - e^k \left(1 - \frac{1}{2}\lambda e^k + o(e^k)\right) \\ &= \frac{1}{2}\lambda (e^k)^2 + o((e^k)^2),\end{aligned}$$

and dividing both sides by $(e^k)^2$ gives

$$\frac{e^{k+1}}{(e^k)^2} = \frac{1}{2}\lambda + o(1),$$

which is equivalent to (3.10). □

Remark 3.4.2. *If $f'(\alpha) = 0$, then the convergence of Newton's method is only linear. In order to recover quadratic convergence we may consider the modified Newton's method:*

$$x^{k+1} = x^k - p\frac{f(x^k)}{f'(x^k)}, \quad k = 0, 1, 2 \dots, \tag{3.14}$$

where $p \in \mathbb{N}$ is the smallest integer such that $f^{(p)}(\alpha) \neq 0$.

3.5 The secant method

A major attraction of Newton's method is its super-linear (quadratic) convergence. However, one disadvantage of the method is that it requires knowledge of the derivative of $f(x)$. In some applications this might be difficult or costly to compute. One way of obtaining a derivative-free method is to replace $f'(x)$ by a suitable difference quotient. A natural choice is

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

which leads to the iteration scheme (note that both x^0 and x^1 must now be supplied)

$$x^{k+1} = x^k - f(x^k)\frac{x^k - x^{k-1}}{f(x^k) - f(x^{k-1})}, \quad k = 1, 2 \ldots . \tag{3.15}$$

This approximate Newton method is known as the *secant method*. For sufficiently smooth f it still exhibits super-linear convergence, albeit not as fast as Newton. One can prove that the order of convergence of the secant method is $p = (1 + \sqrt{5})/2 \approx 1.63$.

3.6 The chord method

An even simpler method is obtained by approximating $f'(x^k)$ by some fixed number $q \in \mathbb{R}$ (the same for each iteration). This leads to the iteration

$$x^{k+1} = x^k - \frac{f(x^k)}{q}, \quad k = 0, 1, 2 \ldots , \tag{3.16}$$

which is known as the *chord method* (or sometimes the *relaxation method*). One may for instance take $q = f'(x^0)$, or, if we are looking for a root in the interval $[a, b]$, $q = (f(b) - f(a))/(b - a)$.

The chord method is a fixed point method for

$$\phi(x) = x - \frac{f(x)}{q}.$$

It follows that $\phi'(x) = 1 - \frac{f'(x)}{q}$. Hence by Theorem 3.3.8, the method converges for initial guesses sufficiently close to the root, provided that

$$|1 - \frac{f'(\alpha)}{q}| < 1.$$

But in general the convergence is only linear.

Example 3.6.1. *(3.2.4 continued) We apply the chord method and Newton's method to find the root of the function f.*
Chord method in the interval $[-1, 1]$, using $q = (f(b) - f(a))/(b - a)$ with $a = -1$, $b = 1$, and starting from $x^0 = 0.7$:

```
>>  [zero,res,niter]=chord(f,-1,1,0.7,1e-8,1000)
```

the script returns the root after 15 iterations.
Newton's method starting from the same x^0 :

```
>>  df = @(x)2*cos(2*x)+1;
>>  [zero,res,niter]=newton(f,df,0.7,1e-8,1000)
```

The zero is found after only 5 iterations. In Figure 3.7 the graph of the error $|x^k - \alpha|$ is shown as a function of the number of iterations k for the bisection method, the fixed point method ϕ_2, the chord method and Newton's method. We observe the linear convergence of the fixed point method and the chord method. We also observe that the error of Newton's method reduces much faster, reflecting the quadratic convergence. Note that the convergence of the bisection method is non-monotone.

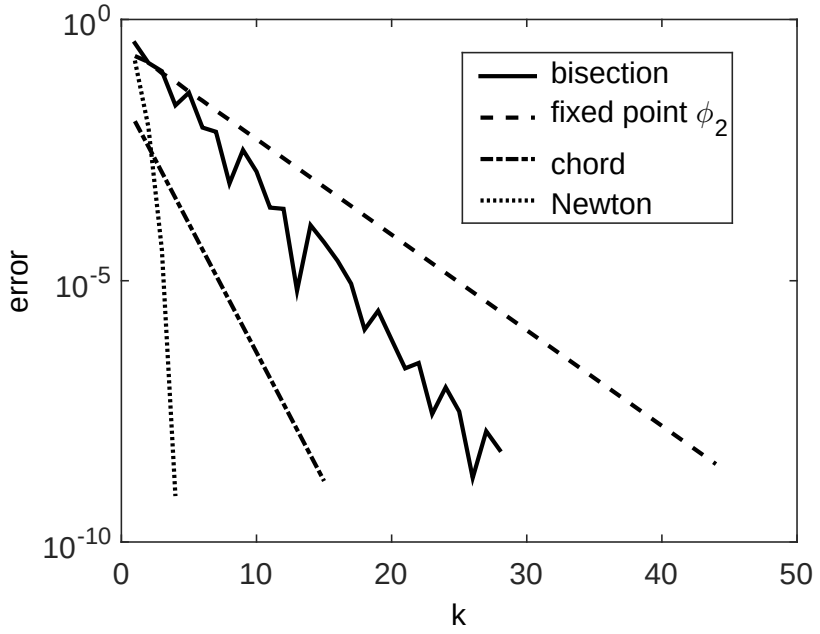


Figure 3.7: Error as a function of the number of iterations for the bisection method, the fixed point method ϕ_2, the chord method and Newton's method - see Example 3.6.1. Note that the y-axis has logarithmic scale.

3.7 Systems of nonlinear equations

We now consider higher-dimensional generalisations. Let $X \subset \mathbb{R}^n$ and $\mathbf{f} : X \to \mathbb{R}^n$, and consider the problem: find $\mathbf{x} \in X$ such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \tag{3.17}$$

This problem is equivalent to the system of equations

$$f_i(x_1, \ldots, x_n) = 0 \text{ for } i = 1, \ldots, n,$$

where f_i and x_i are the components of $\mathbf{f}$ and $\mathbf{x}$ respectively.

Example 3.7.1.

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0 \\ f_2(x_1, x_2) = 5x_1^2 + 21x_2^2 - 9 = 0 \end{cases} \tag{3.18}$$

The first equation defines a circle in $\mathbb{R}^2$ *and the second equation defines an ellipse in* $\mathbb{R}^2$*. The solutions of the system are the intersection points of the circle and ellipse (verify by drawing the two curves), namely* $(-\frac{\sqrt{3}}{2}, \frac{1}{2})$, $(\frac{\sqrt{3}}{2}, \frac{1}{2})$, $(-\frac{\sqrt{3}}{2}, -\frac{1}{2})$ *and* $(\frac{\sqrt{3}}{2}, -\frac{1}{2})$.

3.7.1 Simultaneous iteration

The extension of the fixed point iteration method to the case of higher dimensions is known as *simultaneous iteration*. For the problem (3.17), the first step is to find an equivalent problem

$$\mathbf{x} = \mathbf{g}(\mathbf{x}), \quad \mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n, \tag{3.19}$$

such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad \Leftrightarrow \quad \mathbf{x} = \mathbf{g}(\mathbf{x}).$$

Example 3.7.2. *In the spirit of the chord method one may always take*

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} - \beta \mathbf{f}(\mathbf{x}),$$

for some $0 \neq \beta \in \mathbb{R}$.

The existence of a solution for the fixed point formulation can be shown using a Brouwer fixed point theorem.

Theorem 3.7.3 (Brouwer's fixed point theorem in multiple dimensions)**.** *Assume that* X *is a non-empty, closed, bounded and convex subset of* $\mathbb{R}^n$*. Suppose that* $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$ *is a continuous mapping such that* $\mathbf{g}(X) \subset X$*. Then there exists* $\boldsymbol{\alpha} \in X$ *such that* $\boldsymbol{\alpha} = \mathbf{g}(\boldsymbol{\alpha})$.

An iterative procedure for the construction of an approximating sequence can be defined similarly as in the one-dimensional case, with the difference that in this case we iterate over all the unknowns simultaneously (hence the name).

Definition 3.7.4 (Simultaneous iteration)**.** *Let* $\mathbf{g}$ *satisfy the hypotheses of Theorem 3.7.3. Given* $\mathbf{x}^0 \in X \subset \mathbb{R}^n$*, define* $\mathbf{x}^k$*,* $k = 1, 2, \ldots$*, recursively by*

$$\mathbf{x}^k = \mathbf{g}(\mathbf{x}^{k-1}).$$

The convergence of simultaneous iterations can be proved using the contraction mapping theorem, which, when it applies, also gives the existence of a unique solution to the fixed point formulation (3.19).

Theorem 3.7.5 (Contraction mapping theorem in multiple dimensions)**.** *Suppose that X is a closed subset of $\mathbb{R}^n$ and that $\mathbf{g} : X \to \mathbb{R}^n$ satisfies*

- $\mathbf{g}(X) \subset X$*;*
- $\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| < \Lambda \|\mathbf{x} - \mathbf{y}\|$*, for some $0 < \Lambda < 1$ and some norm $\|\cdot\|$ on $\mathbb{R}^n$. (contraction)*

Then there exists a unique $\boldsymbol{\alpha} \in X$ such that $\boldsymbol{\alpha} = \mathbf{g}(\boldsymbol{\alpha})$. Moreover, the sequence $(\mathbf{x}^k)$ defined by simultaneous iteration converges to $\boldsymbol{\alpha}$ as $k \to \infty$.

Proof. Follows the one-dimensional case. □

Unfortunately, checking that the function $\mathbf{g}(\mathbf{x})$ is a contraction may not be that easy in general. As in the one-dimensional case, things are easier when the function $\mathbf{g}$ is continuously differentiable in a neighbourhood of the root. In the multi-dimensional case we need to consider the behaviour of the *Jacobian matrix* of $\mathbf{g}(\mathbf{x})$, evaluated at the root $\boldsymbol{\alpha}$.

Definition 3.7.6 (Jacobian matrix)**.** *Let $\mathbf{g} = (g_1, g_2, \ldots, g_n)^T : \mathbb{R}^n \to \mathbb{R}^n$ be a function such that the partial derivatives*

$$\frac{\partial g_i}{\partial x_j}, \quad j = 1, \ldots, n,$$

of g_i exist at the point $\mathbf{x} \in \mathbb{R}^n$ for each $i = 1, \ldots, n$. The Jacobian matrix $J_{\mathbf{g}}(\mathbf{x})$ of $\mathbf{g}$ at $\mathbf{x}$ is then defined by the $n \times n$ matrix with elements

$$[J_{\mathbf{g}}(\mathbf{x})]_{ij} = \frac{\partial g_i}{\partial x_j}(\mathbf{x}), \quad i, j = 1, \ldots, n,$$

i.e.

$$J_{\mathbf{g}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1}(\mathbf{x}) & \ldots & \frac{\partial g_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \\ \frac{\partial g_n}{\partial x_1}(\mathbf{x}) & \ldots & \frac{\partial g_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}.$$

Other common notations for the Jacobian matrix include $D\mathbf{g}(\mathbf{x})$ and $\partial \mathbf{g}/\partial \mathbf{x}$.

We now state and prove a technical lemma.

Lemma 3.7.7. *Assume that the partial derivatives of $\mathbf{g}$ are well defined. Then there holds*

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\|_\infty \leq \max_{t \in [0,1]} \|J_{\mathbf{g}}(t\mathbf{x} + (1-t)\mathbf{y})\|_\infty \|\mathbf{x} - \mathbf{y}\|_\infty.$$

Proof. Define $\varphi_i(t) = g_i(t\mathbf{x} + (1-t)\mathbf{y})$, $i = 1, \ldots, n$. By the mean value theorem there holds, for some $\eta_i \in [0, 1]$,

$$g_i(\mathbf{x}) - g_i(\mathbf{y}) = \varphi_i(1) - \varphi(0) = \varphi_i'(\eta_i)(1 - 0) = \varphi_i'(\eta_i).$$

By differentiating φ_i we obtain

$$\varphi_i'(\eta_i) = \sum_{j=1}^{n} \frac{\partial g_i}{\partial x_j}(\eta_i \mathbf{x} + (1-\eta_i)\mathbf{y}) \times (x_j - y_j).$$

It follows that

$$|g_i(\mathbf{x}) - g_i(\mathbf{y})| \leq \|\mathbf{x}-\mathbf{y}\|_\infty \sum_{j=1}^{n} \left| \frac{\partial g_i}{\partial x_j}(\eta_i \mathbf{x} + (1-\eta_i)\mathbf{y}) \right|.$$

Taking the max over i in this expression, and recalling formula (2.4) for the infinity norm of a matrix, we obtain the desired inequality. □

We can now state a corollary of the contraction mapping theorem, that says that if our initial guess is close enough to the root, and the Jacobian of the iteration function is small enough in a neighbourhood of the root, then the simultaneous iterations will converge. This result is a higher-dimensional analogue of Theorem 3.3.8.

Theorem 3.7.8. *Let $\mathbf{g} = (g_1, g_2, \ldots, g_n)^T : \mathbb{R}^n \to \mathbb{R}^n$ be continuously differentiable on some open set $X \subset \mathbb{R}^n$, i.e. the partial derivatives of g are defined and continuous in X. Let $\boldsymbol{\alpha} \in X$ be such that $\boldsymbol{\alpha} = \mathbf{g}(\boldsymbol{\alpha})$. Suppose further that*

$$\|J_{\mathbf{g}}(\boldsymbol{\alpha})\|_\infty < 1.$$

Then there exists $\delta > 0$ such that for any initial guess $\mathbf{x}_0 \in \bar{B}_\delta(\boldsymbol{\alpha}) = \{\mathbf{x} \in X : \|\mathbf{x}-\boldsymbol{\alpha}\|_\infty \leq \delta\}$ the sequence $(\mathbf{x}_k)$ generated using simultaneous iteration converges to $\boldsymbol{\alpha}$ as $k \to \infty$.

Proof. Our proof naturally mimicks that of Theorem 3.3.8. Define $\Lambda = \frac{1}{2}(1 + \|J_{\mathbf{g}}(\boldsymbol{\alpha})\|_\infty)$. Since $\|J_{\mathbf{g}}(\boldsymbol{\alpha})\|_\infty < 1$ we have $0 < \|J_{\mathbf{g}}(\boldsymbol{\alpha})\|_\infty < \Lambda < 1$. By the continuity of the partial derivatives of $\mathbf{g}$ there exists $\delta > 0$ such that $\|J_{\mathbf{g}}(\mathbf{x})\|_\infty \leq \Lambda$ for all $\mathbf{x} \in \bar{B}_\delta(\boldsymbol{\alpha})$, and by Lemma 3.7.7 it follows that

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\|_\infty \leq \Lambda \|\mathbf{x}-\mathbf{y}\|_\infty, \qquad \forall \mathbf{x}, \mathbf{y} \in \bar{B}_\delta(\boldsymbol{\alpha}).$$

In particular, since $\mathbf{g}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}$, this bound (with $\mathbf{y} = \boldsymbol{\alpha}$)) implies that $\mathbf{g}(\bar{B}_\delta(\boldsymbol{\alpha})) \subset \bar{B}_\delta(\boldsymbol{\alpha})$, and the desired result then follows from the contraction mapping theorem. □

3.7.2 Newton's method for systems

To generalize the one-dimensional Newton iteration

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \tag{3.20}$$

to the case of a higher-dimensional nonlinear system

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \qquad \mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n, \tag{3.21}$$

one replaces the reciprocal of the derivative $f'(x^k)$ by the *inverse* of the Jacobian matrix $J_{\mathbf{f}}(\mathbf{x}^k)$. Explicitly, the Newton step in multiple dimensions is

$$\mathbf{x}^{k+1} = \mathbf{x}^k - [J_{\mathbf{f}}(\mathbf{x}^k)]^{-1}\mathbf{f}(\mathbf{x}^k), \quad n = 0, 1, 2 \ldots,$$

where $\mathbf{x}^k$ denotes the kth iteration. In practice one rarely computes the inverse $[J_{\mathbf{f}}(\mathbf{x}^k)]^{-1}$ explicitly, and so the iteration is more commonly written implicitly as

$$[J_{\mathbf{f}}(\mathbf{x}^k)](\mathbf{x}^{k+1} - \mathbf{x}^k) = -\mathbf{f}(\mathbf{x}^k), \quad n = 0, 1, 2 \ldots. \tag{3.22}$$

Regarding convergence of the iteration defined by (3.22), similar results hold as in the scalar case. Provided that $\mathbf{f}$ is sufficiently smooth, and that its derivatives satisfy appropriate conditions (in particular, the Jacobian matrix should be nonsingular in a neighbourhood of the root), the iterates converge quadratically. But the analysis is more involved and we refer to Süli and Mayers for details.

Theorem 3.7.9. *Let $\boldsymbol{\alpha}$ satisfy $\mathbf{f}(\boldsymbol{\alpha}) = \mathbf{0}$. Let $\mathbf{f}$ be twice continuously differentiable in a neighbourhood of $\boldsymbol{\alpha}$, and suppose that the matrix $J_{\mathbf{f}}(\boldsymbol{\alpha})$ is nonsingular. Then the Newton iteration defined by (3.22) converges quadratically to $\boldsymbol{\alpha}$ for a sufficiently good initial guess $\mathbf{x}^0$.*

Proof. See Süli and Mayers Theorem 4.4. □

Calculating the one-dimensional iteration (3.20) is usually straightforward (provided we have a good method for evaluating f and f'). But in the multi-dimensional case at each iteration we need to solve a linear system (3.22) governed by the matrices $A_k = J_{\mathbf{f}}(\mathbf{x}^k)$. This is easier than solving the original nonlinear problem (3.21), but if the dimension n is large (as it often is in practice) the solution of these linear systems is still potentially very costly. In particular, a direct solution method such as Gaussian elimination is often impractical. In the next chapter we will study a number of iterative methods which can provide efficient algorithms for the solution of large linear systems.

Example 3.7.10. *Consider the following system of two nonlinear equations in two unknowns, x_1 and x_2:*

$$\begin{cases} x_1^2 - 2x_1x_2 = 2 \\ x_1 + x_2^2 = -1. \end{cases} \tag{3.23}$$

We can recast the system in the form

$$\mathbf{f} = \mathbf{0}, \qquad \textit{i.e.} \qquad \begin{cases} f_1(x_1, x_2) = 0 \\ f_2(x_1, x_2) = 0 \end{cases}$$

where $\mathbf{f} = (f_1, f_2)$, $f_1(x_1, x_2) = x_1^2 - 2x_1x_2 - 2$ and $f_2(x_1, x_2) = x_1 + x_2^2 + 1$. In Figure 3.8 the two curves $f_1 = 0$ and $f_2 = 0$ are plotted in the square $-6 \le x_1 \le 2$, $-4 \le x_2 \le 4$. This figure has been obtained using the Matlab commands

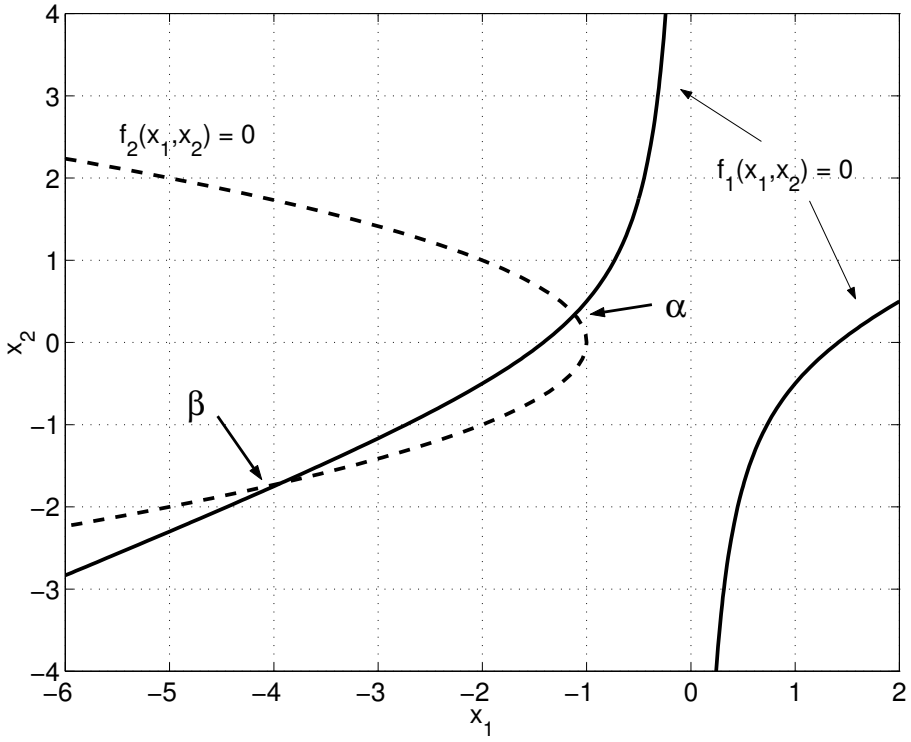


Figure 3.8: Curves $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$. Their intersection point is a solution to the nonlinear system (3.23).

```
>>  f1 = @(x1,x2)x1.^2-2*x1.*x2-2;
>>  f2 = @(x1,x2)x1+x2.^2+1;
>>  [x1,x2] = meshgrid(-6:.1:2,-4:.1:4);
>>  contour(x1,x2,f1(x1,x2),[0,0],'b'); hold on;
>>  contour(x1,x2,f2(x1,x2),[0,0],'b--')
```

The zeros of the system are given by the intersection points of the two curves. From the figure we see that there are two zeros $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$ *and* $\boldsymbol{\beta} = (\beta_1, \beta_2)$.

The Jacobian matrix of the vector valued function $\mathbf{f}$ *is*

$$J_{\mathbf{f}}(\mathbf{x} = (x_1, x_2)) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 - 2x_2 & -2x_1 \\ 1 & 2x_2 \end{bmatrix},$$

and Newton's method is: given $\mathbf{x}^0 = (x_1^0, x_2^0)$, *for each* $n = 0, 1, 2, \ldots$ *find* $\mathbf{x}^{k+1}$ *such that*

$$[J_{\mathbf{f}}(\mathbf{x}^k)](\mathbf{x}^{k+1} - \mathbf{x}^k) = -\mathbf{f}(\mathbf{x}^k). \tag{3.24}$$

We detail the first iteration of the algorithm, using the initial vector $\mathbf{x}^0 = [1, 1]^T$. *Then*

$$J_{\mathbf{f}}(\mathbf{x}^0) = \begin{bmatrix} 0 & -2 \\ 1 & 2 \end{bmatrix},$$

and $\mathbf{x}^1$ *is given by the solution of the linear system:*

$$[J_{\mathbf{f}}(\mathbf{x}^0)](\mathbf{x}^1 - \mathbf{x}^0) = -\mathbf{f}(\mathbf{x}^0).$$

The iterations continue using (3.24) until some stopping criteria is met.

3.8 Stopping criteria for fixed point iterations

An important practical issue concerning fixed point iterations is determining when to stop the iteration. For simplicity we restrict our attention to the one-dimensional case. Suppose we are using a fixed point iteration $x^{k+1} = \phi(x^k)$ to find a root α of the equation $f(\alpha) = 0$ for some $f : \mathbb{R} \to \mathbb{R}$. We would like to specify a criterion on k that guarantees that the error $e^k = x^k - \alpha$ at iteration k is smaller than some user-specified tolerance tol.

Theoretical convergence results tell us something about the rate of convergence to the solution. But they don't usually provide an explicit bound for the error e^k at iteration k, because they usually involve constants that depend on the unknown solution α. Two quantities we do have at our disposal are (i) the computational residual $f(x^k)$; and (ii) the iterative increment $x^k - x^{k+1}$. In this section we briefly discuss their relative merits as the basis for an effective stopping criterion.

First consider the residual $f(x^k)$. Suppose that we have a sequence of approximations x^k converging to the root α. If f is continuous then $f(x^k) \to 0$. And if f is differentiable with $f'(\alpha) \neq 0$ then

$$e^k = \frac{f(x^k)}{f'(\alpha)} + o(e^k), \qquad k \to \infty.$$

Hence if $|f'(\alpha)| \approx 1$ then the residual provides a good estimate of the error and an effective stopping criterion is

$$|f(x^k)| \leq tol.$$

But if $|f'(\alpha)| \ll 1$ the residual will underestimate the error, and if $|f'(\alpha)| \gg 1$ it will overestimate the error. One way to obtain a more reliable stopping criterion is to replace $f'(\alpha)$ in the estimate above by a computable approximation, for example $f'(x^k)$. This leads to the stopping criterion

$$\frac{|f(x^k)|}{|f'(x^k)|} \leq tol.$$

An alternative stopping criterion is found by considering the iterative increment: the iterations are stopped when

$$|x^k - x^{k+1}| < tol. \tag{3.25}$$

To explore the effectiveness of this criterion, consider the following argument. Recalling that $e^k = x^k - \alpha$, if ϕ is differentiable we can write

$$e^{k+1} = x^{k+1} - \alpha = \phi(x^k) - \phi(\alpha) = \phi'(\xi^k)e^k,$$

for some ξ^k between x^k and α. Therefore

$$x^k - x^{k+1} = (x^k - \alpha) - (x^{k+1} - \alpha) = e^k - e^{k+1} = \big(1 - \phi'(\xi^k)\big)\, e^k.$$

Assuming that $|\phi'(\alpha)| < 1$, so that $x^k \to \alpha$ as $k \to \infty$, it also follows that $\phi'(\xi^k) \to \phi'(\alpha)$ as $k \to \infty$. Hence for large k we have

$$e^k \approx \frac{x^k - x^{k+1}}{1 - \phi'(\alpha)}.$$

This shows that the criterion (3.25) gives approximate error control to the accuracy *tol*, provided that $\phi'(\alpha)$ is not close to 1.

Chapter 4

Linear systems

The solution of systems of *linear* equations is a very important special case of the solution of general nonlinear systems considered in the previous chapter. As we saw there, many nonlinear iterative solvers (in particular the Newton method for systems) require at least one linear solve to be carried out at each iteration. As we shall see later, systems of linear equations are also central to the numerical solution of differential equations. They also arise naturally in many mathematical models of real-world problems, as the following example illustrates.

4.1 Example and motivation

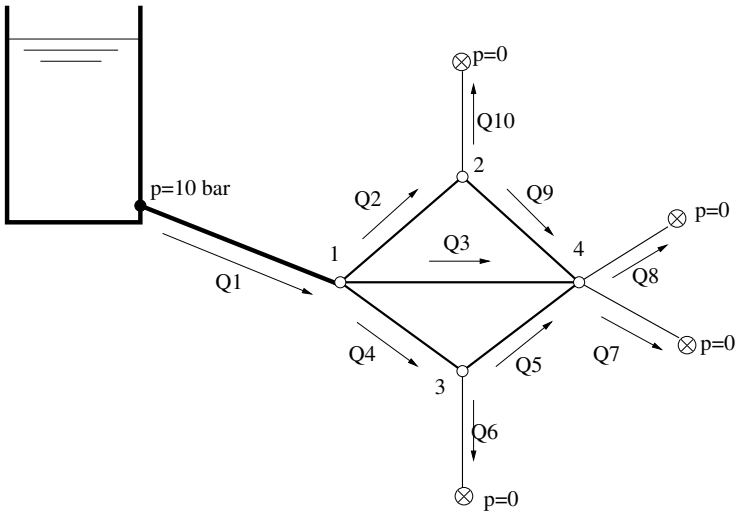


Figure 4.1: Example: hydraulic network

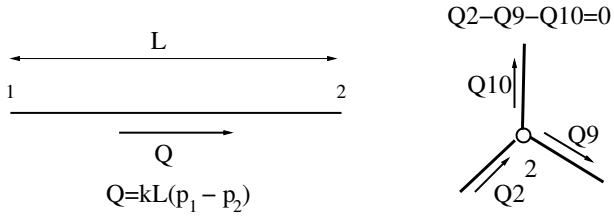
Example 4.1.1 (Hydraulic network)**.** *We consider a hydraulic network (illustrated in Figure*

4.1) formed by 10 *pipes. The network draws water from a reservoir at constant pressure* $p_r = 10$ bar *(all pressures in the example are given as the difference between the actual pressure and the atmospheric pressure measured in bar). In every pipe the following relation between the flow rate* Q $(\mathrm{m}^3/\mathrm{s})$ *and the pressure at the two ends of the pipe* p_1 *and* p_2 *holds:*

$$Q = kL(p_1 - p_2). \tag{4.1}$$

Here k *is the hydraulic resistance* $(\frac{\mathrm{m}^2}{\mathrm{bar\,s}})$ *and* L *is the length of the pipe in meters.*

At every node of the network we can enforce the balance of flows. For example, at node 2 *in the figure below, one has* $Q_2 - Q_{10} - Q_9 = 0$ *(giving a negative sign to exiting flow.)*



Finally, in the exit pipe we assume that the outflow pressure coincides with the atmospheric pressure $(p = 0$ bar*).*

We wish to find the distribution of pressures and flow rates in the network. Using the two given relations it is straightforward to set up a linear system for the pressure on the form

$$A\mathbf{p} = \mathbf{b}, \tag{4.2}$$

where $\mathbf{p} = [p_1, p_2, p_3, p_4]^T$ *is the vector of nodal pressures of the network. Given the following characteristics of the pipes*

Pipe	k	L	Pipe	k	L	Pipe	k	L
1	*0.01*	*20*	*2*	*0.005*	*10*	*3*	*0.005*	*712*
4	*0.005*	*10*	*5*	*0.005*	*10*	*6*	*0.002*	*8*
7	*0.002*	*8*	*8*	*0.002*	*8*	*9*	*0.005*	*10*
10	*0.002*	*8*						

we may compute the matrix A *and vector* $\mathbf{b}$ *as*

$$A = \begin{bmatrix} -0.360 & 0.050 & 0.050 & 0.060 \\ 0.050 & -0.116 & 0.000 & 0.050 \\ 0.050 & 0.000 & -0.116 & 0.050 \\ 0.060 & 0.050 & 0.050 & -0.192 \end{bmatrix}, \qquad b = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The pressure vector is given by the solution of the linear system, i.e.

$$\mathbf{p} = A^{-1}\mathbf{b} = [8.147, 5.943, 5.943, 5.641]^T.$$

Finally, the flow rates may be computed using the relation (4.1):

Q1	*0.371*	*Q2*	*0.110*	*Q3*	*0.150*
Q4	*0.110*	*Q5*	*0.015*	*Q6*	*0.095*
Q7	*0.090*	*Q8*	*0.090*	*Q9*	*0.015*
Q10	*0.095*				

In practical applications a hydraulic network may consist of hundreds of pipes which leads to the need to solve large linear systems.

4.2 Introduction

Our focus in this chapter is on *linear* systems of equations of the form

$$A\mathbf{x} = \mathbf{b}, \tag{4.3}$$

where $A = (a_{ij})$ is an $n \times n$ matrix, $\mathbf{b} = (b_j)$ is a vector in $\mathbb{R}^n$ and $\mathbf{x} = (x_j)$, also a vector in $\mathbb{R}^n$, is the unknown solution. The expression (4.3) is equivalent to the system of n linear equations

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \ i = 1, \ldots, n.$$

We shall assume throughout this chapter that the matrix A is *invertible* (or *nonsingular*), i.e. that the following equivalent conditions hold:

- $\exists$ an $n \times n$ matrix A^{-1} such that $A^{-1}A = AA^{-1} = I$, where I is the identity matrix.
- $\det A \neq 0$.
- $\text{rank}(A) = n$.
- $\ker(A) = \mathbf{0}$.
- A has n linearly independent columns (rows);
- The eigenvalues of A are all non-zero.

If A is invertible then the unique solution of (4.3) is given by $\mathbf{x} = A^{-1}\mathbf{b}$. General analytical formulas for A^{-1} in terms of determinants do exist (Cramer's rule). But they are completely unsuitable for numerical computation. In practice one applies methods for solving (4.3) which do not involve the explicit computation of A^{-1}. These methods split into two types: *direct* methods, which (in exact arithmetic) provide the solution $\mathbf{x}$ in a finite number of operations, and *iterative* methods, which provide a sequence of approximate solutions $\mathbf{x}^k$ which (under appropriate assumptions on A) converge to the solution $\mathbf{x}$ as the iteration count $k \to \infty$.

4.3 Direct solvers

In certain special cases it is very easy to write down a direct method for solving (4.3). The simplest case is when A is a *diagonal* matrix, i.e.

$$a_{ij} = 0 \qquad \forall i \neq j.$$

Then the inverse of A is also a diagonal matrix, with diagonal entries $a_{ii}^{-1} = (a_{ii})^{-1}$ (note that $a_{ii} \neq 0$ for any i else A would not be invertible), and the solution vector has components

$$x_i = \frac{1}{a_{ii}} b_i, \quad i = 1, \ldots, n.$$

In this case the solution can be computed using exactly n floating point operations (flops).

Two other important special cases are when A is either *upper triangular*, meaning that

$$a_{ij} = 0 \quad \forall i, j \ : \ 1 \leq j < i \leq n,$$

or *lower triangular*, meaning that

$$a_{ij} = 0 \quad \forall i, j \ : \ 1 \leq i < j \leq n.$$

Upper triangular systems can be solved using the *backward substitution algorithm*:

$$x_n = b_n / a_{nn},$$

and for $i = n-1, n-2, \ldots, 2, 1$

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right).$$

In the lower triangular case, the analogous *forward substitution algorithm* applies:

$$x_1 = b_1 / a_{11},$$

and for $i = 2, 3, \ldots, n$

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j \right).$$

(Note that in both cases we know $a_{ii} \neq 0$ for all $i = 1, \ldots, n$, because A is invertible and $\det A = \prod a_{ii}$ for a triangular matrix.)

The number of divisions and multiplications involved in these algorithms is of order $n(n+1)/2$ and the number of additions and subtractions is of order $n(n-1)/2$. Hence the algorithms both require $O(n^2)$ floating point operations (flops).

The best-known direct solution method for *general* invertible linear systems is *Gaussian elimination* (GE). This algorithm involves moving row-by-row down the system, subtracting appropriate multiples of the current row from the rows below, so as to arrive at an upper triangular system which can be solved using the backward substitution algorithm above. In fact the operations that transform the system into upper triangular form correspond to multiplying the original matrix A by the inverse of an appropriate lower triangular matrix L. Hence GE is often referred to as "LU factorization", because given a matrix A the algorithm outputs lower and upper triangular matrices L and U such that $A = LU$.

There are many subtleties involved in implementing and analysing the GE algorithm (e.g. "pivoting") that we shall not go into in these lectures. The main thing we note is that the computational cost of the algorithm scales like $O(n^3)$ as the size of the matrix tends to infinity. This means that, while stable implementations (as in Matlab's "backslash" command) are effective for relatively small systems, for very large systems GE is impractical.[1] For such systems one is forced to turn to iterative methods, some examples of which we shall study in the next section. These are often much cheaper than GE, provided that one can obtain an accurate solution with a relatively small number of iterations.

Another fact one should remember about direct methods like GE is that, while in theory the algorithm produces the exact solution $\mathbf{x}$, this only holds if one works in infinite precision arithmetic. In practical computations using finite precision arithmetic the result of the direct solver will not be $\mathbf{x}$, but rather some approximation $\tilde{\mathbf{x}}$. Recalling (2.8), we have

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|},$$

where the *residual* is defined by $\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}}$, and the condition number $K(A) = \|A\|\|A^{-1}\|$. (Recall from (2.6) that for a symmetric positive definite matrix the condition number with respect to the Euclidean norm $\|\cdot\|_2$ is equal to the quotient of the largest and smallest eigenvalues.) Typically $\mathbf{r} \neq \mathbf{0}$ due to rounding errors, and then if $K(A)$ is large, the error can be big even for a small residual $\mathbf{r}$. It follows that the larger the condition number of the matrix, the poorer the results of the direct solver.

Here we find another advantage of iterative methods: as we shall see shortly, they provide a natural framework in which to "precondition" the system (4.3) so as to mitigate the detrimental effect of any ill-conditioning in A. An illustration of the power of preconditioning is given in Example 4.3.2 below.

[1] We note however that once the expensive $O(n^3)$ factorization $A = LU$ has been computed, solving the system $A\mathbf{x} = \tilde{\mathbf{b}}$ for any other right-hand side $\tilde{\mathbf{b}}$ is relatively cheap, as it can be evaluated using a combination of the forward and backward substitution algorithms, at $O(n^2)$ cost.

Example 4.3.1. *Consider the system* $A\mathbf{u} = \mathbf{b}$ *of size* $n \times n$*, where*

$$A = \begin{pmatrix} \frac{2}{h} & -\frac{1}{h} & 0 & \cdots & \cdots & 0 \\ -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & \vdots \\ 0 & -\frac{1}{h} & \ddots & \ddots & & \\ \vdots & & \ddots & & -\frac{1}{h} & \vdots \\ \vdots & & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} \\ 0 & \cdots & \cdots & 0 & -\frac{1}{h} & \frac{2}{h} \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} h \\ h \\ \vdots \\ \\ \vdots \\ h \\ h \end{pmatrix}, \qquad h = \frac{1}{n+1}. \tag{4.4}$$

The solution of this system represents an approximation to the deformation under a uniform load of an elastic string of length one, fixed at $x = 0$ *and* $x = 1$*.*

Using Matlab we construct the matrix and compute its condition number (with respect to $\|\cdot\|_2$*) for a range of values of* n*. The resulting graph is presented in Figure 4.2.*

```
>> for i=1:10
    n=5*i; h=1/(n+1);
    A = (2/h)*diag(ones(n,1)) - (1/h)*diag(ones(n-1,1),1) ...
        - (1/h)*diag(ones(n-1,1),-1)
    K(i) = cond(A)
   end
>> plot([5:5:50],K)
```

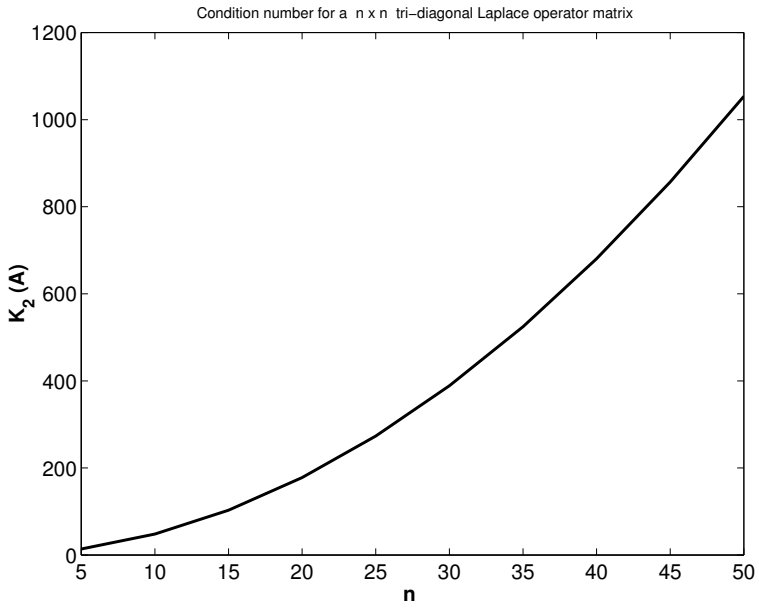


Figure 4.2: Condition number of the matrix A from Example 4.3.1 as a function of the size n.

Example 4.3.2. *The Hilbert matrix of size $n \times n$ is a symmetric matrix defined by*

$$A_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,\ldots,n.$$

This is a classic example of a very ill-conditioned matrix. In Matlab we may construct a Hilbert matrix of size n using the command `hilb(n)`.

We construct linear systems $A\mathbf{x} = \mathbf{b}$ of size $n = 4, 6, 8, 10, 12, 14, \ldots$ where A is a Hilbert matrix, the exact solution $\mathbf{x}_{\text{ex}}$ is the unit vector and the right hand side is chosen as $\mathbf{b} = A\mathbf{x}_{\text{ex}}$. For every n, we compute the condition number of the matrix (with respect to $\|\cdot\|_2$), solve the system using LU factorization, and compare the obtained solution $\mathbf{x}_{\text{lu}}$ with the exact solution.

```
>> for j=2:7;
    n=2*j; i=j-1; nn(i)=n;
    A=hilb(n);
    x_ex=ones(n,1); b=A*x_ex;
    Acond(i)=cond(A);
    x=A\b;
    error_lu(i)=norm(x-x_ex)/norm(x_ex);
    end
>> semilogy(nn,Acond,'-',nn,error_lu,'--')
```

Figure 4.3 shows the condition number as well as the relative error $\|\mathbf{x}_{\text{ex}} - \mathbf{x}_{\text{lu}}\|/\|\mathbf{x}_{\text{ex}}\|$. In all the calculations, $\|\cdot\| = \|\cdot\|_2$ is the Euclidean norm $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \cdot \mathbf{x}}$. We also show the error in a solution $\mathbf{x}_{\text{gr}}$ obtained using a preconditioned iterative method. Note that the error for the direct method increases with the condition number, whereas that for the iterative method remains constant.

4.4 Iterative methods for linear systems

As in the chapter on nonlinear equations, the idea behind an iterative method for the linear system $A\mathbf{x} = \mathbf{b}$ is to construct a sequence of vectors $\mathbf{x}^k$ that converge to the solution $\mathbf{x}$ as $k \to \infty$, i.e.

$$\lim_{k\to\infty} \mathbf{x}^k = \mathbf{x}.$$

The classical setting is where the iterates satisfy a linear recurrence relation

$$\mathbf{x}^{k+1} = B\mathbf{x}^k + \mathbf{c}, \qquad k = 0, 1, 2, \ldots, \tag{4.5}$$

where B is called the *iteration matrix* (depending on A) and $\mathbf{c}$ is a vector (depending on $\mathbf{b}$). If B and $\mathbf{c}$ stay the same at each iteration then the iterative method (4.5) is said to be

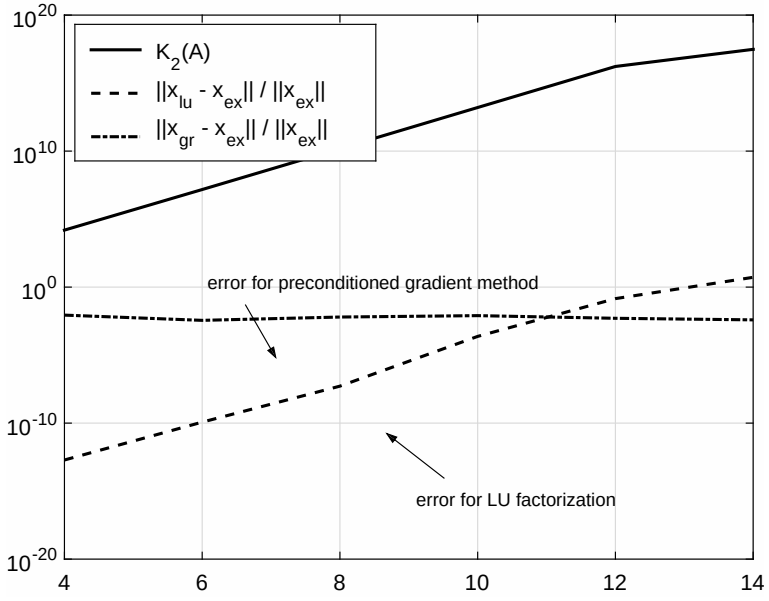


Figure 4.3: Solution of a linear system based on the Hilbert matrix: relative error obtained using LU-factorization and using a preconditioned iterative method. The horizontal axis shows n, the size of the system.

stationary. If B and $\mathbf{c}$ are such that the exact solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$ satisfies

$$\mathbf{x} = B\mathbf{x} + \mathbf{c} \tag{4.6}$$

then the method is said to be *consistent*. We note that a consistent method is defined by its iteration matrix B alone: using the relation $\mathbf{x} = A^{-1}\mathbf{b}$, we can determine the vector $\mathbf{c}$ uniquely as $\mathbf{c} = (I - B)A^{-1}\mathbf{b}$. Note also that consistent stationary methods are nothing more than fixed point formulations (in the sense of §3.7.1) of the problem $\mathbf{f}(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = \mathbf{0}$, in which the fixed point function $\mathbf{g}(\mathbf{x})$ (which should satisfy $\mathbf{x} = \mathbf{g}(\mathbf{x})$) is a *linear* function of $\mathbf{x}$, i.e. $\mathbf{g}(\mathbf{x}) = B\mathbf{x} + \mathbf{c}$. The iteration (4.5) is precisely the "simultaneous iteration" (or "fixed point iteration") of Definition 3.7.4.

For a consistent stationary method the error $\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k$ at iteration k satisfies the recurrence relation

$$\mathbf{e}^{k+1} = B\mathbf{e}^k, \quad k = 0, 1, \ldots, \tag{4.7}$$

which implies that

$$\mathbf{e}^k = B^k\mathbf{e}^0, \quad k = 0, 1, \ldots, \tag{4.8}$$

where $\mathbf{e}^0 = \mathbf{x} - \mathbf{x}^0$ is the initial error. We would like to determine conditions under which the method converges, i.e. $\mathbf{e}^k \to \mathbf{0}$ as $k \to \infty$. The key parameter is the spectral radius $\rho(B) = \max_{\lambda \in \sigma(B)} |\lambda|$, where $\sigma(B)$ is the set of eigenvalues of B.

Theorem 4.4.1. *If the method (4.5) is consistent (in the sense of (4.6)) then* $\lim_{k\to\infty} \mathbf{e}^k = \mathbf{0}$ *for all* $\mathbf{e}^0$ *(and therefore* $lim_{k\to\infty}\mathbf{x}^k = \mathbf{x}$ *for all* $\mathbf{x}^0$*) if and only if* $\rho(B) < 1$.

Proof. If $\rho(B) < 1$ then by Lemma 2.4.4 we have that $\lim_{k\to\infty} B^k = 0$. This means that for any $\mathbf{e}^0$ and any induced matrix norm, $\|\mathbf{e}^k\|_V = \|B^k\mathbf{e}^0\|_V \leq \|B^k\|_M\|\mathbf{e}^0\|_V \to 0$ as $k \to \infty$, i.e. the method converges. Conversely, if $\rho(B) > 1$ then there exists $\lambda \in \sigma(B)$ such that $\lambda > 1$. Let $\mathbf{e}^0$ be an eigenvector associated with λ, so that $B\mathbf{e}^0 = \lambda\mathbf{e}^0$ and $\mathbf{e}^k = B^k\mathbf{e}^0 = \lambda^k\mathbf{e}^0$ for all $k \geq 1$. Since $\lambda > 1$ it does not hold that $\mathbf{e}^k \to \mathbf{0}$ as $k \to \infty$. □

The rate of convergence is slightly tricky to analyse in general. Certainly if $\|\cdot\|_V$ is a vector norm inducing a matrix norm $\|\cdot\|_M$ for which $\|B\|_M < 1$ then it is easy to see from (4.7) and (4.8) that the method converges linearly with constant $C = \|B\|_M$, and that furthermore

$$\|\mathbf{e}^k\|_V \leq (\|B\|_M)^k\|\mathbf{e}^0\|_V, \quad k = 0, 1, \ldots. \tag{4.9}$$

In practice a reasonable surrogate for the convergence constant C is often provided by the spectral radius $\rho(B)$. Indeed, if B is a symmetric matrix and we work with the norms $\|\cdot\|_V = \|\cdot\|_2$ and $\|\cdot\|_M = \|\cdot\|_2$, the estimate (4.9) holds with $\|B\|_M$ replaced by $\rho(B)$, i.e.

$$\|\mathbf{e}^k\|_2 \leq \rho(B)^k\|\mathbf{e}^0\|_2, \quad k = 0, 1, \ldots, \tag{4.10}$$

because $\|B\|_2 = \rho(B)$ for symmetric matrices, by Lemma 2.4.3. (See also Theorem 4.6.1 below.) But this is not always true more generally - recall that in general while $\rho(B) \leq \|B\|_M$ for all induced matrix norms (Lemma 2.4.2), it can hold that $\rho(B) < \|B\|_M$, in which case $\rho(B)$ may provide an overly optimistic convergence constant.

4.5 Basic stationary method and preconditioning

The simplest choice of iteration matrix and vector is $B = I - A$ and $\mathbf{c} = \mathbf{b}$, leading to the *basic stationary method*

$$\mathbf{x}^{k+1} = (I - A)\mathbf{x}^k + \mathbf{b}, \quad k = 0, 1, \ldots. \tag{4.11}$$

By Theorem 4.4.1 this method converges for all initial guesses $\mathbf{x}_0$ if and only if $\rho(I - A) < 1$. Since $\sigma(I - A) = 1 - \sigma(A)$, this is equivalent to requiring that all eigenvalues of A must lie in the unit ball centred at the point 1 in the complex plane. This condition is rather restrictive and for general matrices the basic stationary method (4.11) will not converge. Fortunately there are many ways of obtaining methods which converge for other classes of matrices.

Notice that the iteration (4.11) is obtained from the original equation $A\mathbf{x} = \mathbf{b}$ by splitting the term $A\mathbf{x}$ on the left-hand side into the sum $A\mathbf{x} = \mathbf{x} - (\mathbf{x} - A\mathbf{x})$, moving the second term to the right-hand side, and then replacing $\mathbf{x}$ by $\mathbf{x}^{k+1}$ on the left-hand side and by $\mathbf{x}^k$ on the right-hand side. This corresponds to a splitting of the matrix A into the sum $A = I - (I - A)$.

Suppose that instead we split A as $A = P - N$ for some invertible matrix P, with $N = P - A$. Then following the same procedure we arrive at the iteration

$$P\mathbf{x}^{k+1} = N\mathbf{x}^k + \mathbf{b}, \quad k = 0, 1, \ldots, \tag{4.12}$$

which can also be written in increment form

$$P(\mathbf{x}^{k+1} - \mathbf{x}^k) = \mathbf{r}^k, \quad k = 0, 1, \ldots, \tag{4.13}$$

where $\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$ is the residual. Multiplying (4.12) by P^{-1} gives the equivalent form

$$\mathbf{x}^{k+1} = (I - P^{-1}A)\mathbf{x}^k + P^{-1}\mathbf{b}, \quad k = 0, 1, \ldots, \tag{4.14}$$

which reveals that (4.12) is a consistent stationary method of the form (4.5) with $B = I - P^{-1}A$ and $\mathbf{c} = P^{-1}\mathbf{b}$. (But note that in practical implementations one usually works with (4.12) or (4.13) rather than (4.5), to avoid explicit computation of P^{-1}). Note that one can also derive (4.14) (and hence (4.12)) by applying the basic stationary method (4.11) to the modified system

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}. \tag{4.15}$$

The matrix P is called the *preconditioner* of A. By (4.14) and Theorem 4.4.1 the method (4.12) converges for all initial guesses $\mathbf{x}_0$ if and only if $\rho(I - P^{-1}A) < 1$. Given a matrix A, one would like to choose P in such a way as to make $\rho(I - P^{-1}A)$ as small as possible (in particular, smaller than 1), to guarantee fast convergence. But one also wants the linear system (4.12) to be easy to solve for $\mathbf{x}^{k+1}$, so that each iteration step is cheap to evaluate. These are two competing requirements, and the extreme cases are:

- $P = I$; (4.12) is trivial to solve but there is no preconditioning effect.
- $P = A$; perfect preconditioning, in the sense that the iteration would converge in one step, but computing this step would be as difficult as solving the original system.

The key to the art of preconditioning is to find a balance between the two competing requirements of effectiveness and easy application. The right choice of P depends strongly on the type of matrix A one is trying to precondition.

4.6 Stationary Richardson method

The simplest preconditioner is obtained by taking $P = \frac{1}{\alpha}I$ for some $0 \neq \alpha \in \mathbb{R}$, sometimes called a *relaxation* or *acceleration* parameter. This corresponds to a simple rescaling of the system to $\alpha A\mathbf{x} = \alpha\mathbf{b}$, and an iteration matrix $B_\alpha = I - \alpha A$. (The case $\alpha = 1$ is just the basic stationary method.) Since $\sigma(I - \alpha A) = 1 - \alpha\sigma(A)$, the method converges if and only if all eigenvalues of A lie in the ball of radius $1/|\alpha|$ centred at the point $1/\alpha$ in the complex plane. This method is sometimes called a stationary Richardson method.

The following theorem summarises the convergence theory for symmetric positive definite (SPD) matrices.

Theorem 4.6.1. *Let A be a symmetric positive definite matrix and let $0 < \lambda_{\min} \leq \lambda_{\max}$ denote the smallest and largest eigenvalues of A. Then the stationary Richardson method is convergent if and only if $0 < \alpha < 2/\lambda_{\max}$. Furthermore, when the method converges it does so linearly with*

$$\|\mathbf{e}^{k+1}\|_2 \leq \rho(B_\alpha)\|\mathbf{e}^k\|_2, \quad k = 0, 1, \ldots. \tag{4.16}$$

The optimal choice of $0 < \alpha < 2/\lambda_{\max}$ minimising the convergence constant $\rho(B_\alpha)$ is given by

$$\alpha_* = \frac{2}{\lambda_{\max} + \lambda_{\min}}, \qquad \text{for which} \quad \rho(B_{\alpha_*}) = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{K_2(A) - 1}{K_2(A) + 1}. \tag{4.17}$$

Proof. The fact that the method is convergent if and only if $0 < \alpha < 2/\lambda_{\max}$ follows from our discussion above. The bound (4.16) follows directly from (4.10). That (4.17) gives the optimal convergence constant holds because for $0 < \alpha < 2/\lambda_{\max}$ the eigenvalues of B_α all lie between the extremal eigenvalues $1 - \alpha\lambda_{\max}$ and $1 - \alpha\lambda_{\min}$, which satisfy $-1 < 1 - \alpha\lambda_{\max} \leq 1 - \alpha\lambda_{\min} < 1$. The spectral radius $\rho(B_\alpha) = \max\{|1 - \alpha\lambda_{\max}|, |1 - \alpha\lambda_{\min}|\}$ is minimised when $1 - \alpha\lambda_{\max} = -(1 - \alpha\lambda_{\min})$, which provides the claimed optimal values α_* and $\rho(B_{\alpha_*})$. The degenerate case where $\lambda_{\min} = \lambda_{\max} = \lambda$ corresponds to $A = \lambda I$, in which case the optimal choice $\alpha = \alpha_* = 1/\lambda$ gives $B_{\alpha_*} = 0$ and the iteration converges in one step. □

Remark 4.6.2. *For later reference, we note that when the stationary Richardson converges, the linear convergence estimate (4.16) also holds with the Euclidean norm $\|\cdot\|_2$ replaced by a certain A-weighted norm $\|\cdot\|_A$, defined below. That is,*

$$\|\mathbf{e}^{k+1}\|_A \leq \rho(B_\alpha)\|\mathbf{e}^k\|_A, \quad k = 0, 1, \ldots. \tag{4.18}$$

Here

$$\|\mathbf{v}\|_A = \sqrt{(\mathbf{v}, \mathbf{v})_A} = \sqrt{(A\mathbf{v}, \mathbf{v})} = \sqrt{\mathbf{v}^T A\mathbf{v}}, \qquad \mathbf{v} \in \mathbb{R}^n, \tag{4.19}$$

where the inner product $(\cdot, \cdot)_A$ satisfies

$$(\mathbf{v}, \mathbf{w})_A = (A\mathbf{v}, \mathbf{w}) = \mathbf{v}^T A\mathbf{w} = \mathbf{w}^T A\mathbf{v}, \qquad \mathbf{v}, \mathbf{w} \in \mathbb{R}^n. \tag{4.20}$$

That (4.19) and (4.20) define a norm and an inner product relies on the fact that A is SPD.

4.7 The Jacobi method and the Gauss-Seidel method

The Jacobi and Gauss-Seidel methods exploit the fact that diagonal and triangular systems are easy to solve (recall §4.3). Both methods are obtained by decomposing the matrix A as[2]

$$A = D - E - F,$$

where

[2]One could write $A = D + E + F$ and define E and F to be the negative of the definitions above. But to avoid confusion I have followed the convention found in most textbooks (e.g. Quarteroni, Sacco and Saleri).

- D is the diagonal component of A, i.e. $D = (d_{ij})$ with $d_{ii} = a_{ii}$ for each $i = 1, \ldots, n$ and $d_{ij} = 0$ for $i \neq j$;
- E is minus the strictly lower triangular component of A, i.e. $E = (e_{ij})$ with $e_{ij} = -a_{ij}$ for $i > j$ and 0 otherwise;
- F is minus the strictly upper triangular component of A, i.e. $F = (f_{ij})$ with $f_{ij} = -a_{ij}$ for $i < j$ and 0 otherwise.

(Note here that a strictly upper (lower) triangular matrix is an upper (lower) triangular matrix whose diagonal elements are all zeroes.)

In the Jacobi method we take the preconditioner to be $P = D$ (so that $N = E + F$). This corresponds to an iteration matrix $B_{\mathrm{J}} = I - D^{-1}A = D^{-1}(E + F)$. Note that for D to be invertible, we need A to have all non-zero diagonal elements. Provided this is the case, the iteration step for the Jacobi method can be written explicitly in component form as

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^k \right), \quad i = 1, \ldots, n. \tag{4.21}$$

Example 4.7.1. *Let*

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}.$$

Then

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 16 \end{pmatrix}, \quad E = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -5 & 0 & 0 & 0 \\ -9 & -10 & 0 & 0 \\ -13 & -14 & -15 & 0 \end{pmatrix}, \quad F = \begin{pmatrix} 0 & -2 & -3 & -4 \\ 0 & 0 & -7 & -8 \\ 0 & 0 & 0 & -12 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

To extract D, E and F from A in Matlab, we use the following commands:

```
>> A = [1,2,3,4;5,6,7,8;9,10,11,12;13,14,15,16];
>> D = diag(diag(A));
>> E = -(tril(A) - D);
>> F = -(triu(A) - D);
```

The Gauss-Seidel method uses the preconditioner $P = D - E$ (the lower triangular component of A), so that $N = F$. The iteration matrix is then $B_{\mathrm{GS}} = I - (D - E)^{-1}A = (D - E)^{-1}F$, and the method may be written explicitly in component form as

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right), \quad i = 1, \ldots, n. \tag{4.22}$$

The following convergence results for the Jacobi and Gauss-Seidel methods are classical.

Theorem 4.7.2. *If A is strictly diagonally dominant by rows, that is, if*

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \text{ for } i = 1, \ldots, n,$$

then the Jacobi method and the Gauss-Seidel method are both convergent.

Theorem 4.7.3. *If A is symmetric positive definite then the Gauss-Seidel method converges.*

The proofs of the convergence results for Gauss-Seidel are beyond the scope of this course. Here we only give the proof of Theorem 4.7.2 for the Jacobi method.

Proof of Theorem 4.7.2, Jacobi case. The Jacobi iteration matrix B_{J} has entries (b_{ij}) given by $b_{ij} = -a_{ij}/a_{ii}$ for $i \neq j$ and $b_{ii} = 0$ for each i. Hence the infinity norm of B_{J} is equal to (recall (2.4))

$$\|B_{\text{J}}\|_\infty = \max_{i\in\{1,\ldots,n\}} \sum_{j=1}^{n} |b_{ij}| = \max_{i\in\{1,\ldots,n\}} \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} = \max_{i\in\{1,\ldots,n\}} \frac{1}{|a_{ii}|} \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|.$$

From this it follows that if A is strictly diagonally dominant by rows, then $\|B_{\text{J}}\|_\infty < 1$, which in turn (by Lemma 2.4.2) implies that $\rho(B_{\text{J}}) < 1$, so the Jacobi method converges. In fact, by the discussion leading up to (4.9) the convergence is linear and we have the error bound

$$\|\mathbf{e}^k\|_\infty \leq (\|B_{\text{J}}\|_\infty)^k \|\mathbf{e}^0\|_\infty, \quad k = 0, 1, \ldots.$$

□

4.8 Non-stationary methods

The iterative methods we have considered so far are all *stationary*, meaning that the iteration matrix B and vector $\mathbf{c}$ in (4.5) stay the same at each iteration. One might ask whether there is any benefit in allowing these to change at each iteration. Such *non-stationary* methods do not correspond to fixed-point iterations. We will consider two examples applicable to symmetric positive definite matrices: the gradient method and the conjugate gradient method. Both have unpreconditioned and preconditioned forms.

Our starting point for both methods is the stationary Richardson iteration, for which $B_\alpha = I - \alpha A$ for some $0 \neq \alpha \in \mathbb{R}$. It is instructive to rewrite the iteration in the "update form"

$$\mathbf{x}^{k+1} - \mathbf{x}^k = \alpha \mathbf{r}^k, \qquad k = 0, 1, 2, \ldots, \tag{4.23}$$

where $\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$ is the residual at step k, which itself updates at step $k+1$ by the formula

$$\mathbf{r}^{k+1} - \mathbf{r}^k = -\alpha A \mathbf{r}^k, \qquad k = 0, 1, 2, \ldots. \tag{4.24}$$

Formula (4.23) shows that in the stationary Richardson iteration, to obtain $\mathbf{x}^{k+1}$ from $\mathbf{x}^k$ we move in the direction of the residual $\mathbf{r}^k$, with a relative step length α. We saw in Theorem 4.6.1 that if A is SPD then we can find an optimal value of α so as to minimise the spectral radius $\rho(B_\alpha)$ and obtain the fastest possible convergence. But this requires knowledge of the extremal eigenvalues of A, something which we might not have.

Suppose that we now consider a more general update

$$\mathbf{x}^{k+1} - \mathbf{x}^k = \alpha_k \mathbf{p}^k, \qquad k = 0, 1, 2, \ldots, \tag{4.25}$$

for some update direction $\mathbf{p}^k$ and relative step length α_k, which might change at each iteration. We recall our notation $\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k$ for the error at step k, which is related to the residual by

$$\mathbf{r}^k = A\mathbf{e}^k, \qquad k = 0, 1, 2, \ldots, \tag{4.26}$$

and note that the residual itself updates by

$$\mathbf{r}^{k+1} - \mathbf{r}^k = -\alpha_k A\mathbf{p}^k, \qquad k = 0, 1, 2, \ldots. \tag{4.27}$$

Our aim is to choose $\mathbf{p}^k$ and α_k as the iteration progresses (without knowledge of the spectrum of A), so as to optimize the accuracy of the next iterate in a certain sense. As we shall see, the two methods considered below (gradient and conjugate gradient) are both obtained by making a certain choice for $\mathbf{p}^k$ and then calculating α_k so as to minimize the size of the error $\mathbf{e}^{k+1}$ measured in the A-weighted norm $\|\cdot\|_A$ defined in Remark 4.6.2. This choice is made because it allows the minimizer α_k to be found in closed form. (The same is not true if one attempts to minimize $\|\mathbf{e}^{k+1}\|_2$.)

4.8.1 The gradient method

In the gradient method, given $\mathbf{x}^k$ we take $\mathbf{p}^k = \mathbf{r}^k$ (as in the stationary Richardson method), and choose α_k so as to minimise $\|\mathbf{e}^{k+1}\|_A$. To achieve this we note that

$$\begin{aligned}
\|\mathbf{e}^{k+1}\|_A^2 &= (A\mathbf{e}^{k+1}, \mathbf{e}^{k+1}) \\
&= (\mathbf{r}^{k+1}, A^{-1}\mathbf{r}^{k+1}) \\
&= (\mathbf{r}^k - \alpha_k A\mathbf{r}^k, A^{-1}\mathbf{r}^k - \alpha_k \mathbf{r}^k) \\
&= \|\mathbf{e}^k\|_A^2 - 2\alpha_k \|\mathbf{r}^k\|_2^2 + \alpha_k^2 \|\mathbf{r}^k\|_A^2.
\end{aligned}$$

This is a quadratic in α_k, and by differentiating with respect to α_k and setting the resulting expression equal to zero, we find that the unique minimum is at

$$\alpha_k = \frac{\|\mathbf{r}^k\|_2^2}{\|\mathbf{r}^k\|_A^2}. \tag{4.28}$$

To summarise, the *gradient method* is defined as follows: Given $\mathbf{x}^0$ set $\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0$. Then for $k = 0, 1, 2, \ldots$ until convergence, update

$$\begin{aligned} \alpha_k &= \frac{\|\mathbf{r}^k\|_2^2}{\|\mathbf{r}^k\|_A^2}, \\ \mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha_k \mathbf{r}^k, \\ \mathbf{r}^{k+1} &= \mathbf{r}^k - \alpha_k A\mathbf{r}^k. \end{aligned}$$

Theorem 4.8.1. *If A is SPD then the gradient method converges linearly, with*

$$\|\mathbf{e}^{k+1}\|_A \leq \frac{K_2(A) - 1}{K_2(A) + 1} \|\mathbf{e}^k\|_A, \quad k \geq 0. \tag{4.29}$$

Proof. Given $\mathbf{x}^k$, let $\mathbf{e}_*^{k+1}$ be the error resulting from a *single* stationary Richardson step with $\alpha = \alpha_*$. The definition of α_k says $\|\mathbf{e}^{k+1}\|_A \leq \|\mathbf{e}_*^{k+1}\|_A$, so by Remark 4.6.2 it follows that

$$\|\mathbf{e}^{k+1}\|_A \leq \|\mathbf{e}_*^{k+1}\|_A \leq \rho(B_{\alpha_*}) \|\mathbf{e}^k\|_A = \frac{K_2(A) - 1}{K_2(A) + 1} \|\mathbf{e}^k\|_A.$$

□

Note that the convergence constant in (4.29) for the gradient method (in which we choose the step-length α_k adaptively for each k) is exactly the same as that achieved by the optimal choice of α in the stationary Richardson method (see Theorem 4.6.1 and Remark 4.6.2).

Noting that

$$\frac{K_2(A) - 1}{K_2(A) + 1} = 1 - \frac{2}{K_2(A) + 1},$$

we see that if $K_2(A)$ is large then the convergence constant may be close to 1, giving very slow reduction in the error. To combat this, the gradient method can be generalised to include a preconditioner. For the unpreconditioned version, the (nonstationary) iteration matrix was given by $B = I - \alpha_k A$. For the preconditioned version we change this to $B = I - \alpha_k P^{-1} A$, where P is some symmetric positive definite matrix. This corresponds to using an update direction $\mathbf{p}^k = P^{-1}\mathbf{r}^k$. As before we then choose α_k to minimise $\|\mathbf{e}^{k+1}\|_A$. A similar calculation to that presented above shows that the optimal choice is

$$\alpha_k = \frac{(\mathbf{r}^k, \mathbf{p}^k)}{(\mathbf{p}^k, A\mathbf{p}^k)} = \frac{\|\mathbf{p}^k\|_P^2}{\|\mathbf{p}^k\|_A^2}. \tag{4.30}$$

The resulting *preconditioned gradient method* is defined as follows: Given $\mathbf{x}^0$ and P set $\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0$. Then for $k = 0, 1, 2, \ldots$ until convergence, update

$$\begin{aligned} P\mathbf{p}^k &= \mathbf{r}^k, \\ \alpha_k &= \frac{\|\mathbf{p}^k\|_P^2}{\|\mathbf{p}^k\|_A^2}, \\ \mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha_k \mathbf{p}^k, \\ \mathbf{r}^{k+1} &= \mathbf{r}^k - \alpha_k A\mathbf{p}^k. \end{aligned}$$

If $P^{-1}A$ is SPD then one can prove that the method converges linearly with

$$\|\mathbf{e}^{k+1}\|_A \leq \frac{K_2(P^{-1}A)-1}{K_2(P^{-1}A)+1}\|\mathbf{e}^k\|_A, \quad k \geq 0, \tag{4.31}$$

implying that

$$\|\mathbf{e}^k\|_A \leq \left(\frac{K_2(P^{-1}A)-1}{K_2(P^{-1}A)+1}\right)^k \|\mathbf{e}^0\|_A, \quad k \geq 1. \tag{4.32}$$

Given A, the point is to try and choose P such that $K_2(P^{-1}A)$ is much smaller than $K_2(A)$, so that $\frac{K_2(P^{-1}A)-1}{K_2(P^{-1}A)+1} < \frac{K_2(A)-1}{K_2(A)+1}$, improving the rate of convergence of the method over the unpreconditioned version.

In the following table we compare the definitions of the gradient method (G) and the preconditioned gradient method (GP).

G	**GP**
$\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$	$\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$
	$P\mathbf{p}^k = \mathbf{r}^k$
$\alpha_k = \frac{(\mathbf{r}^k,\mathbf{r}^k)}{(A\mathbf{r}^k,\mathbf{r}^k)}$	$\alpha_k = \frac{(\mathbf{r}^k,\mathbf{p}^k)}{(A\mathbf{p}^k,\mathbf{p}^k)}$
$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{r}^k$	$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{p}^k$

Example 4.8.2. *Consider the system $A\mathbf{x} = \mathbf{b}$, where $A = \left[\begin{smallmatrix} 5 & 1 \\ 6 & 8 \end{smallmatrix}\right]$, and $\mathbf{b}$ is such that the exact solution is given by $\mathbf{x} = (1,1)^T$. We approximate the solution using the gradient method (G) and the preconditioned gradient method with $P = D$ (GP). The relative error, plotted as a function of the iteration count k, is shown in Figure 4.4. Note the faster convergence of GP compared to G. Note also that A is not SPD, so this example shows that the gradient method can work for some non-SPD matrices. (But see also Example 4.8.6.)*

Remark 4.8.3. *Why the name "gradient method"? To explain this, we first note that for a symmetric positive definite matrix A the problem of solving $A\mathbf{x} = \mathbf{b}$ is equivalent to finding the unique minimiser $\mathbf{x}$ of the convex function*

$$\Phi : \mathbb{R}^n \to \mathbb{R}, \quad \Phi(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T A\mathbf{y} - \mathbf{y}^T\mathbf{b} = \frac{1}{2}\sum_{i,j=1}^{n} y_i a_{ij} y_j - \sum_{i=1}^{n} b_i y_i,$$

a quadratic form sometimes referred to as the "energy" of the system. To see the equivalence of these two problems, it is enough to note that the gradient of Φ is given by

$$\nabla\Phi(\mathbf{y}) = \frac{1}{2}(A^T + A)\mathbf{y} - \mathbf{b} = A\mathbf{y} - \mathbf{b}\,,$$

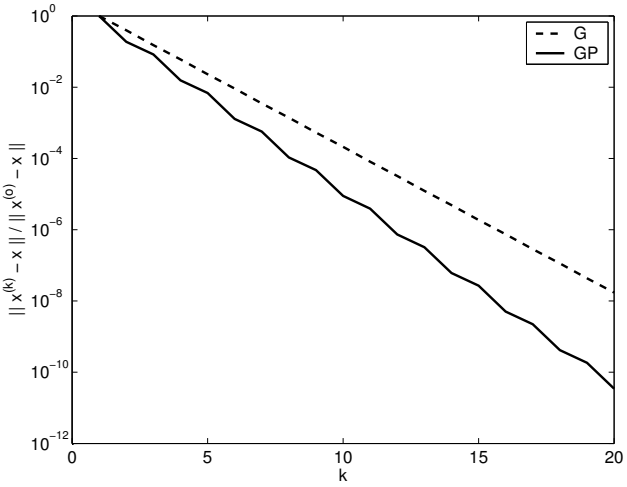


Figure 4.4: Relative error for the gradient method (G) and the preconditioned gradient method with $P = D$ (GP).

and that if $A\mathbf{x} = \mathbf{b}$ *then*

$$\Phi(\mathbf{y}) - \Phi(\mathbf{x}) = \frac{1}{2}(\mathbf{y} - \mathbf{x})^T A(\mathbf{y} - \mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_A^2, \qquad \mathbf{y} \neq \mathbf{x}.$$

These formulas also show that the direction of steepest descent of the function Φ *at a point* $\mathbf{y}$ *is given by* $-\nabla\Phi(\mathbf{y}) = \mathbf{b} - A\mathbf{y}$*, i.e. the residual, and that minimising* $\Phi(\mathbf{y}(\alpha))$ *over some parameter* α *is equivalent to minimising* $\|\mathbf{y}(\alpha) - \mathbf{x}\|_A$*. Hence each iteration of the gradient method corresponds precisely to a steepest descent minimization of* Φ*.*

4.8.2 The conjugate gradient method

One property of the gradient method that limits its speed of convergence is the fact that each increment is orthogonal to the previous increment (with respect to the Euclidean inner product $(\cdot, \cdot)$. To see this, note that for the gradient method

$$(\mathbf{p}^k, \mathbf{p}^{k-1}) = (\mathbf{r}^k, \mathbf{r}^{k-1}) = (\mathbf{r}^{k-1} - \alpha_{k-1}A\mathbf{r}^{k-1}, \mathbf{r}^{k-1}) = \|\mathbf{r}^{k-1}\|_2^2 - \alpha_{k-1}\|\mathbf{r}^{k-1}\|_A^2,$$

which is equal to zero by the definition of α_{k-1}. The effect of this orthogonality is that for bad choices of initial data the iterates can converge in a zig-zag fashion, taking a long time to approach the solution. For an illustration of this see Figure 4.5.

The conjugate gradient method aims to accelerate convergence by adjusting the increment direction so that $\mathbf{p}^k$ is no longer orthogonal to $\mathbf{p}^{k-1}$ in the Euclidean sense, but rather in the A-weighted sense, i.e.

$$(\mathbf{p}^k, \mathbf{p}^{k-1})_A = (A\mathbf{p}^k, \mathbf{p}^{k-1}) = 0. \tag{4.33}$$

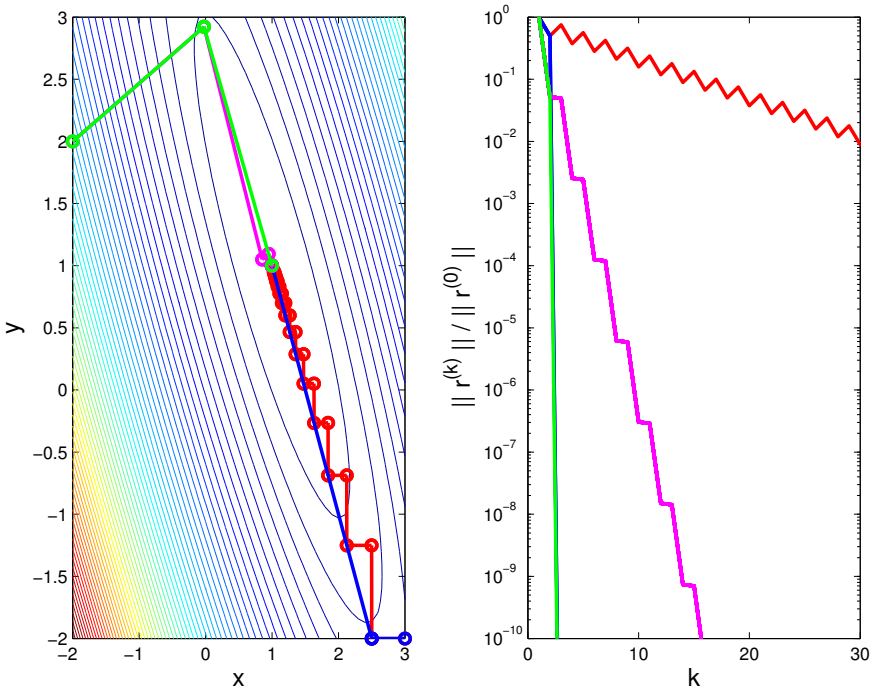


Figure 4.5: Convergence histories for the gradient method (red, pink) and the conjugate gradient method (blue, green) for a 2×2 system, using two different different starting points. The contour lines are level curves of the associated energy functional Φ of Remark 4.8.3. Observe that if an unfortunate choice of initial data is made the gradient method converges very slowly due to a zig-zagging behaviour. This is eliminated by the conjugate gradient method, which always converges after two iterations (see Remark 4.8.4).

In this case we say that $\mathbf{p}^k$ and $\mathbf{p}^{k-1}$ are "A-orthogonal" or "conjugate orthogonal". Precisely, given $\mathbf{p}^{k-1}$ the next increment $\mathbf{p}^k$ is determined by subtracting from the residual $\mathbf{r}^k$ (or $P^{-1}\mathbf{r}^k$ in the preconditioned version) a suitable multiple of $\mathbf{p}^{k-1}$ so as to satisfy (4.33). In fact, one can do this in such a way as to ensure that $(\mathbf{p}^k, \mathbf{p}^j)_A = 0$ for all $j < k$. Once the increment direction has been determined, one proceeds as in the gradient method to choose the relative step length α_k so as to minimise $\|\mathbf{e}^{k+1}\|_A$.

We now give the definition of the *preconditioned conjugate gradient method*. The unpreconditioned version takes the same form but with $P = I$, so that $\mathbf{z}^k = \mathbf{r}^k$. Given an initial guess

$\mathbf{x}^0$, compute $\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0$, $P\mathbf{z}^0 = \mathbf{r}^0$, $\mathbf{p}^0 = \mathbf{z}^0$. Then for $k \geq 0$ until convergence, update

$$\begin{aligned}
\alpha_k &= \frac{(\mathbf{p}^k, \mathbf{r}^k)}{(\mathbf{p}^k, A\mathbf{p}^k)}, \\
\mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha_k \mathbf{p}^k, \\
\mathbf{r}^{k+1} &= \mathbf{r}^k - \alpha_k A\mathbf{p}^k, \\
P\mathbf{z}^{k+1} &= \mathbf{r}^{k+1}, \\
\beta_k &= \frac{(A\mathbf{p}^k, \mathbf{z}^{k+1})}{(A\mathbf{p}^k, \mathbf{p}^k)}, \\
\mathbf{p}^{k+1} &= \mathbf{z}^{k+1} - \beta_k \mathbf{p}^k \,.
\end{aligned}$$

Under the assumption that $P^{-1}A$ is SPD, one can prove the error estimate

$$\|\mathbf{e}^k\|_A \leq 2 \left(\frac{\sqrt{K_2(P^{-1}A)} - 1}{\sqrt{K_2(P^{-1}A)} + 1} \right)^k \|\mathbf{e}^0\|_A \,, \quad k \geq 0 \,. \tag{4.34}$$

Comparing (4.34) to (4.32) suggests that when $K_2(P^{-1}A) \gg 1$, the preconditioned conjugate gradient method should converge much more rapidly than the preconditioned gradient method, because $\sqrt{K_2(P^{-1}A)} \ll K_2(P^{-1}A)$.

Remark 4.8.4. *A surprising fact is that, in the absence of rounding errors, the conjugate gradient (CG) method actually converges to the exact solution in at most n iterations. This is a striking indication of the power of using conjugate increment directions. Hence, strictly speaking, CG should actually be classed as a direct method. But in practice, when n is large CG is almost never run to full convergence, so it is usually classed as an iterative method.*

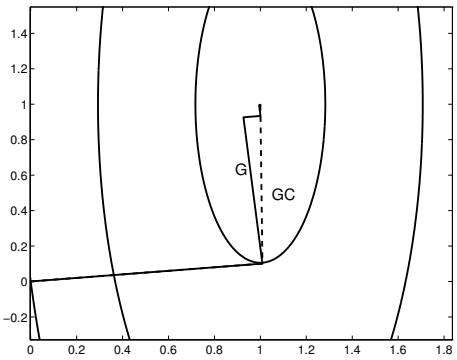


Figure 4.6: Descent directions for the conjugate gradient method (CG, dotted line) and the gradient method (G, solid line). Observe that CG converges to the solution in two iterations.

Example 4.8.5. *Consider the linear system*

$$\begin{cases} 2x_1 + x_2 &= 1 \\ x_1 + 3x_2 &= 0 \end{cases}, \tag{4.35}$$

with corresponding matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$*, which is symmetric and positive definite. The exact solution to the system is* $x_1 = 3/5$*,* $x_2 = -1/5$*.*

We approximate the solution using a number of different iterative methods, all initialised with

$$\mathbf{x}^0 = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}.$$

In Figure 4.7 we show the value of $\frac{\|\mathbf{e}^k\|}{\|\mathbf{e}^0\|} = \frac{\|\mathbf{x}^k - \mathbf{x}\|}{\|\mathbf{x}^0 - \mathbf{x}\|}$ *for the Jacobi, Gauss-Seidel, and diagonally preconditioned gradient methods. We also show results for a further method not described in detail here, the successive over-relaxation (SOR) method, which performs even better than the three other methods for this problem.*

The first steps for the Jacobi, Gauss-Seidel, and diagonally preconditioned gradient methods are as follows.

1. Jacobi method:

$$\begin{cases} x_1^{(1)} &= \frac{1}{2}(1 - x_2^{(0)}) \\ x_2^{(1)} &= -\frac{1}{3}(x_1^0) \end{cases} \Rightarrow \begin{cases} x_1^{(1)} &= \frac{1}{4} \\ x_2^{(1)} &= -\frac{1}{3} \end{cases}$$

2. Gauss-Seidel method:

$$\begin{cases} x_1^{(1)} &= \frac{1}{2}(1 - x_2^{(0)}) \\ x_2^{(1)} &= -\frac{1}{3}x_1^{(1)} \end{cases} \Rightarrow \begin{cases} x_1^{(1)} &= \frac{1}{4} \\ x_2^{(1)} &= -\frac{1}{12} \end{cases}$$

3. Preconditioned gradient method with $P = D = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$***:***

Writing the first iteration in increment form, we have to solve

$$\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} x_1^{(1)} - 1 \\ x_2^{(1)} - \frac{1}{2} \end{pmatrix} = \alpha_0 \mathbf{r}^0,$$

where

$$\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix} \mathbf{x}^0 = \begin{pmatrix} -3/2 \\ -5/2 \end{pmatrix}.$$

Clearly,

$$P^{-1} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/3 \end{pmatrix}$$

so

$$\mathbf{p}^0 = P^{-1}\mathbf{r}^0 = \begin{pmatrix} -3/4 \\ -5/6 \end{pmatrix} \quad \textit{and then} \quad \alpha_0 = \frac{(\mathbf{r}^0, \mathbf{z}^0)}{(A\mathbf{z}^0, \mathbf{z}^0)} = \frac{77}{107}.$$

Thus $\mathbf{x}^1 = \mathbf{x}^0 + \alpha_0 \mathbf{z}^0 = \begin{pmatrix} 1.5397 \\ 1.0997 \end{pmatrix}$.

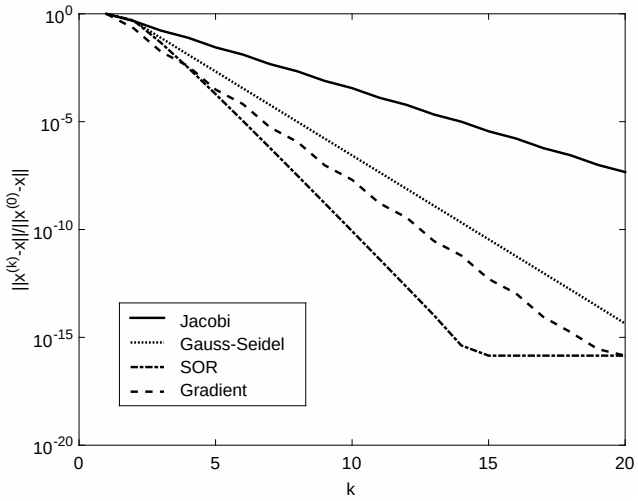


Figure 4.7: Behaviour of the relative error for different iterative methods applied to the system (4.35)

Example 4.8.6. *Now consider the system*

$$\begin{cases} 2x_1 + x_2 & = 1 \\ -x_1 + 3x_2 & = 0 \end{cases} \tag{4.36}$$

the solution of which is $x_1 = 3/7$, $x_2 = 1/7$.

The matrix $A = \begin{pmatrix} 2 & 1 \\ -1 & 3 \end{pmatrix}$ *is not symmetric, but is clearly diagonally dominant by rows, and hence both Jacobi and Gauss-Seidel should converge.*

We initialise both methods using the initial guess

$$\mathbf{x}^0 = \begin{pmatrix} x_1^0 \\ x_2^0 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}.$$

Figure 4.8 shows the resulting plots of $\frac{\|\mathbf{e}^k\|}{\|\mathbf{e}^0\|} = \frac{\|\mathbf{x}^k - \mathbf{x}\|}{\|\mathbf{x}^0 - \mathbf{x}\|}$ *for the Jacobi and Gauss-Seidel methods. Note that Gauss-Seidel converges more rapidly than Jacobi. This is to be expected as the preconditioner is working harder. The price we pay is that each iteration step is slightly more expensive.*

Note that since the matrix A is not symmetric positive definite, if we were to apply the gradient or the conjugate gradient methods there is no guarantee that they would converge. Even if they do converge the convergence may be slow. In fact the conjugate gradient method does not converge for this problem.

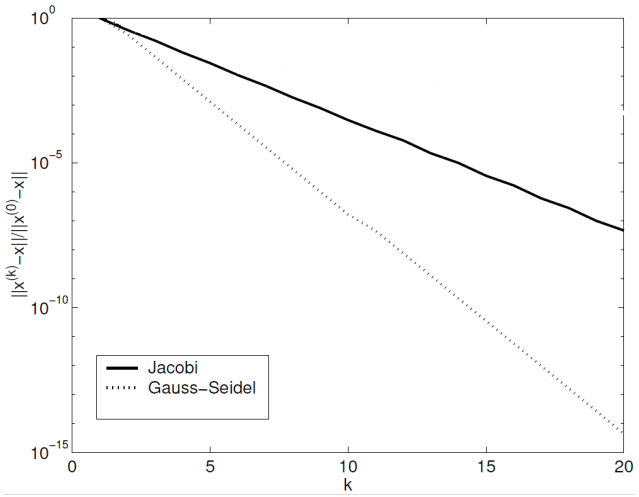


Figure 4.8: Behaviour of the relative error for different iterative methods applied to the system (4.36)

4.9 Stopping criteria for iterative methods

Just as in the nonlinear case, we need to specify stopping criteria to control when our iterative method should terminate. Ideally we would like to stop as soon as the relative error is below some user defined tolerance *tol*, i.e.

$$\frac{\|\mathbf{x}^k - \mathbf{x}\|}{\|\mathbf{x}\|} \leq tol.$$

But since the relative error is unknown we need some *a posteriori* error estimator that is easy to compute at each iterate k. As before, two natural quantities to consider as surrogates for the actual error are the computational residual and the iterative increment.

A residual-based criterion would stop the iteration once

$$\|\mathbf{r}^k\| = \|\mathbf{b} - A\mathbf{x}^k\| \leq tol\|\mathbf{b}\|.$$

Recalling from (2.8) that

$$\frac{\|\mathbf{x}^k - \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(A)\frac{\|\mathbf{r}^k\|}{\|\mathbf{b}\|}, \tag{4.37}$$

we see that the actual relative error could be a factor of $K(A)$ larger than *tol* if this residual-based criterion is used. This may be acceptable if $K(A)$ is moderate, or at least readily estimated. But if $K(A)$ is very large or completely unknown, then the residual-based criterion may not give reliable results. Effective preconditioning will make residual-based error estimation more accurate, because for a preconditioned method one would obtain the modified estimate

$$\frac{\|\mathbf{x}^k - \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(P^{-1}A)\frac{\|P^{-1}\mathbf{r}^k\|}{\|P^{-1}\mathbf{b}\|}.$$

Note that the multiplicative factor in front of the residual is now the condition number of the preconditioned system, which should be significantly smaller than that of the original system. If this holds then the modified residual-based criterion

$$\|P^{-1}\mathbf{r}^k\| \leq tol\|P^{-1}\mathbf{b}\|$$

should be more effective than the unpreconditioned version.

An estimator based on the increment $\boldsymbol{\delta}^k = \mathbf{x}^{k+1} - \mathbf{x}^k$ would terminate the iteration once

$$\|\boldsymbol{\delta}^k\| \leq tol\|\mathbf{b}\|.$$

For a compatible stationary method with iteration matrix B we have $\mathbf{e}^{k+1} = B\mathbf{e}^k$, and hence, assuming that the matrix and vector norms are compatible, the triangle inequality gives

$$\|\mathbf{e}^k\| = \|\mathbf{e}^{k+1} - \boldsymbol{\delta}^k\| \leq \|B\|\|\mathbf{e}^k\| + \|\boldsymbol{\delta}^k\|.$$

Assuming that $\|B\| < 1$, we can rearrange this estimate to give

$$\|\mathbf{e}^k\| \leq \frac{1}{1 - \|B\|}\|\boldsymbol{\delta}^k\|.$$

The norm $\|B\|$ may not be straightforward to compute or estimate. But if B is symmetric then $\|B\|_2 = \rho(B)$ (recall Lemma 2.4.3), and if we work with $\|\cdot\|_2$ throughout, we have that

$$\|\mathbf{e}^k\|_2 \leq \frac{1}{1 - \rho(B)}\|\boldsymbol{\delta}^k\|_2.$$

Recall that a necessary and sufficient condition for convergence is $\rho(B) < 1$. Smaller $\rho(B)$ implies both faster convergence, and a more reliable increment-based stopping criterion.

4.10 Computational cost

As we discussed earlier in the chapter, the cost of a direct method such as Gaussian elimination (LU factorization) generally scales like $O(n^3)$ flops as the size n of the matrix increases. Analysing the cost of iterative methods is more complicated because one has to take into account both the cost of each iteration, and the number of iterations required to achieve the desired tolerance. In the methods considered here, the cost of each iteration will in general be dominated by the cost of matrix-vector multiplication involving the iteration matrix B. In general the cost of this is $O(n^2)$ flops. Therefore, in order for the iterative method to be cheaper than the direct solver we need the number of iterations to be much less than n. Ideally one would like the number of iterations to stay bounded or increase only very slowly as n increases. But achieving this usually requires the construction of a good preconditioner, which is not always straightforward.

This general analysis is somewhat pessimistic because if the matrix is sparse or has a special structure (such as the tridiagonal matrix of (4.3.1)) then matrix-vector multiplication may be significantly cheaper than $O(n^2)$, even as cheap as $O(n)$. In this case iterative methods can be extremely cheap compared to direct methods, unless the latter can also exploit the special structure of the matrix.[3]

We end the chapter by illustrating these results with some numerical examples.

Example 4.10.1. *Consider the matrix*

$$A = \begin{pmatrix} 5 & 7 \\ 7 & 10 \end{pmatrix} \tag{4.38}$$

for which $K(A) \approx 223$. *In Figure 4.9 we compare the residual and the relative error when the Gauss-Seidel method is used. Contrasting this with Figure 4.10, which shows the analogous plot for a better conditioned* 2×2 *system that has* $K(A) \approx 2.6$, *we clearly see the effect of conditioning on the quality of the residual-based error estimator.*

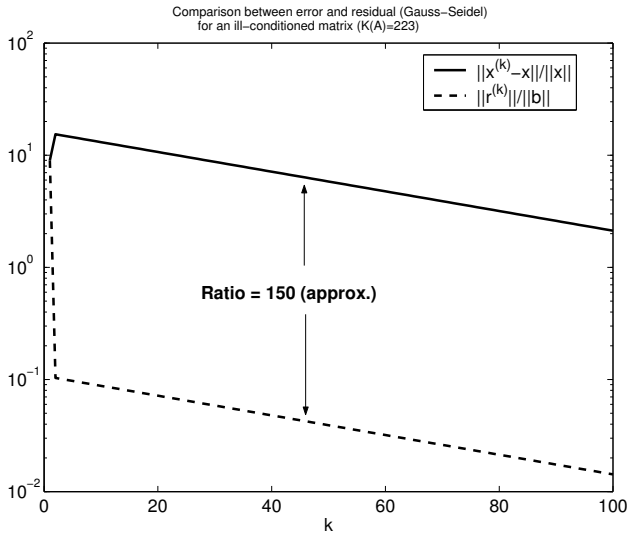


Figure 4.9: Comparison between relative error and the residual (Gauss-Seidel) for the ill-conditioned matrix (4.38).

Example 4.10.2. *(4.3.1* **continued***)*

We revisit Example 4.3.1 and solve the system for $n = 25$ *using the Gauss-Seidel method and the preconditioned gradient method with* $P = D$, *starting from the initial vector* $\mathbf{x}^0 = \mathbf{0}$ *and with a tolerance of* 10^{-6} *on the relative residual. We use the following Matlab commands :*

[3] In the tridiagonal case this is actually possible: the celebrated *Thomas algorithm* is a direct method for tridiagonal systems with $O(n)$ complexity - see section 3.3 of Süli and Mayers for details.

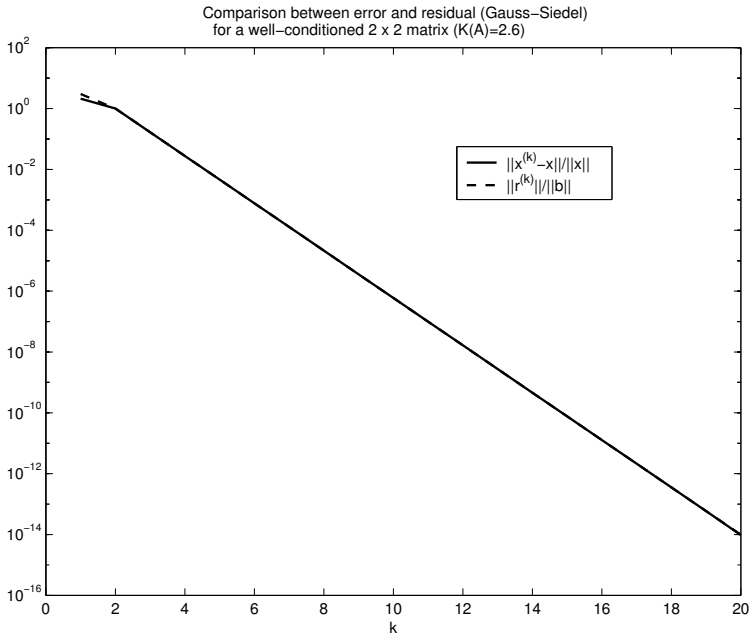


Figure 4.10: Comparison between relative error and the residual (Gauss-Seidel) for a well-conditioned matrix.

```
>> n=25; h=1/(n+1);
>> A = (2/h)*diag(ones(n,1)) - (1/h)*diag(ones(n-1,1),1) ...
       - (1/h)*diag(ones(n-1,1),-1);
>> b = h*ones(n,1);
>> [x_gs,iter_gs] = itermeth(A,b,zeros(25,1),5000,1e-6,'G');
>> [x_grad,iter_grad] = ...
            gradient(A,b,zeros(n,1),5000,1e-6,diag(diag(A)));
```

The number of iterations required by the two methods is as follows:

```
>> iter_gs
iter_gs =
   940
>> iter_grad
iter_grad =
        1896
```

By contrast, for this problem the preconditioned conjugate gradient method with $P = D$ requires just 13 iterations!

Figure 4.11 shows how the number of iterations needed by the three methods grows as a function of the size n of the matrix. These result show the superior performance of the conjugate gradient method for this ill-conditioned problem.

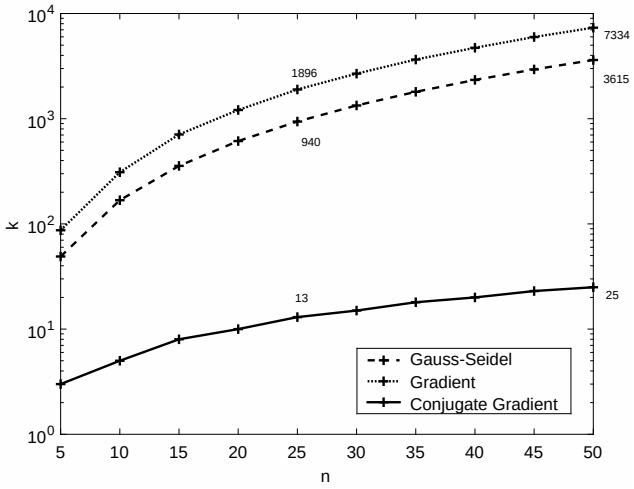


Figure 4.11: Number of iterations as a function of matrix size n, to reach the tolerance 10^{-6} on the residual.

Example 4.10.3. *(4.3.2 continued)*

Now we revisit Example 4.3.2 and solve a linear system involving the Hilbert matrix of size n using the preconditioned gradient method with a diagonal preconditioner, an initial vector $\mathbf{x}_0 = \mathbf{0}$, and a tolerance of 10^{-6} on the relative residual.

The computation was performed using the following Matlab commands.

```
>> for j=2:7;
    n=2*j; i=j-1; nn(i)=n;
    A=hilb(n);
    x_ex=ones(n,1); b=A*x_ex;
    Acond(i)=cond(A);
    tol=1.e-6; maxit=10000;
    R=diag(diag(A));
    x0=zeros(n,1);
    [x,iter_gr(i)]=gradient(A,b,x0,maxit,tol,R);
    error_gr(i)=norm(x-x_ex)/norm(x_ex);
   end
>> semilogy(nn,Acond,'-',nn,error_gr,':')
```

The resulting relative errors for increasing values of n were already reported in Figure 4.3, where we compared them with those obtained using a direct solver (LU-factorization). For ease of reference we also summarize the computational data in the following table. Note that since the matrix is very ill-conditioned the error in the gradient method is significantly bigger than the tolerance 10^{-6}. However, importantly it remains bounded with respect to increasing n, instead of growing rapidly like for the direct solver.

		LU	*Gradient method*		
n	$K(A)$	*Error*	*Error*	*Iterations*	*Residual*
4	*1.55e+04*	*1.94e-13*	*8.72e-03*	*995*	*1.00e-06*
6	*1.50e+07*	*1.18e-10*	*3.60e-03*	*1813*	*9.99e-07*
8	*1.53e+10*	*5.23e-08*	*6.30e-03*	*1089*	*9.96e-07*
10	*1.60e+13*	*2.38e-04*	*7.99e-03*	*875*	*9.99e-07*
12	*1.67e+16*	*1.41e-01*	*5.09e-03*	*1355*	*9.99e-07*
14	*2.04e+17*	*9.82e-01*	*3.91e-03*	*1379*	*9.98e-07*

Example 4.10.4. *(4.3.1* **continued***)*

Finally, we again revisit the problem of the deflected string from Example 4.3.1 and consider discretizations with increasing number of degrees of freedom. This results in linear systems that have increasing size and increasing condition number. Indeed one can show that the condition number of the problem scales as n^2. In the table below we compare the results from computing the solution of the linear system using a direct method (this time the "Cholesky method") and an iterative method (the conjugate gradient method preconditioned using an "incomplete Cholesky factorization"). While these methods have not been described in this course, the purpose of this example is just to show how the computational cost of an iterative method can be significantly lower than that of a direct method.

The first two columns in the table give the size n and the sparsity of the matrix measured as m/n^2 where m denotes the number of non-zero elements in the matrix. Then we report the number of operations and memory needed for the two methods. We see that for the larger systems there is a clear advantage in using the iterative method.

		Cholesky		*Conjugate Gradient*		*flops(Chol.)/ flops(GC)*	*Mem(Chol.)/ Mem(GC)*
n	m/n^2	*flops*	*Memory*	*flops*	*Memory*		
47	*0.12*	*8.05e+03*	*464*	*1.26e+04*	*228*	*0.64*	*2.04*
83	*0.07*	*3.96e+04*	*1406*	*3.03e+04*	*533*	*1.31*	*2.64*
150	*0.04*	*2.01e+05*	*4235*	*8.86e+04*	*1245*	*2.26*	*3.4*
225	*0.03*	*6.39e+05*	*9260*	*1.95e+05*	*2073*	*3.27*	*4.47*
329	*0.02*	*1.74e+06*	*17974*	*3.39e+05*	*3330*	*5.15*	*5.39*
424	*0.02*	*3.78e+06*	*30815*	*5.49e+05*	*4513*	*6.88*	*6.83*
530	*0.01*	*8.31e+06*	*50785*	*8.61e+05*	*5981*	*9.65*	*8.49*
661	*0.01*	*1.19e+07*	*68468*	*1.11e+06*	*7421*	*10.66*	*9.23*

Chapter 5

Ordinary differential equations

Differential equations are a powerful tool in mathematical modelling and computational science. In most practical situations the equations are not solvable in simple closed form, and one needs to adopt numerical approximations. In this chapter we consider some basic aspects of the numerical solution of ordinary differential equations. Our focus is mainly on initial value problems, but at the end of the chapter we briefly discuss a class of two-point boundary value problems.

5.1 Example and motivation

Example 5.1.1. (Biology) *Consider a population of y animals in an environment where a maximum of B individuals can coexist. We assume that the initial population y_0 satisfies $y_0 < B$ and that for small populations $y \ll B$ the reproduction rate of the animals is approximately equal to some constant C. For larger populations the rate should slow down so that the population never exceeds B. The simplest model for such a situation is the logistic equation*

$$y'(t) = Cy(t)\left(1 - \frac{y(t)}{B}\right),\ t > 0, \quad y(0) = y_0, \tag{5.1}$$

where $y' = dy/dt$. By solving this equation we can find the evolution of the population in time.

This equation can be generalised to model two interacting populations in competition for the same resources. For instance, suppose that a predator population y_2 survives by hunting a prey population y_1. Then one can model the evolution of the two populations using the following system of differential equations (sometimes called a Lotka-Volterra predator-prey model)

$$\begin{cases} y_1'(t) = C_1 y_1(t)\left[1 - b_1 y_1(t) - d_2 y_2(t)\right], \\ y_2'(t) = -C_2 y_2(t)\left[1 - b_2 y_2(t) - d_1 y_1(t)\right]. \end{cases} \tag{5.2}$$

Here C_1 and C_2 are the growth factors of the two populations, d_1 and d_2 accounts for interaction between the population (preying, contamination etc.), and b_1 and b_2 are related to the amount of food/resources available for each population.

5.2 Initial value problems - the Cauchy problem

We will consider initial value problems of the following form: given a function $f : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$, continuous with respect to both arguments, and initial data $y_0 \in \mathbb{R}$, find $y : \mathbb{R}_+ \to \mathbb{R}$ satisfying

$$\begin{cases} y'(t) = f(t, y(t)) & \text{for } t > 0, \\ y(0) = y_0. \end{cases} \tag{5.3}$$

This is often called the *Cauchy problem* for the differential equation $y'(t) = f(t, y(t))$. In §5.7 we will consider the extension to systems of equations such as (5.2), but for now we restrict our attention to the one-dimensional case.

Example 5.2.1.

- *An example of a linear Cauchy problem is given by:*

$$\begin{cases} y'(t) = 3y(t) - 3t & \text{for } t > 0, \\ y(0) = 1. \end{cases} \tag{5.4}$$

  Here $f(t, v) = 3v - 3t$ and the exact solution is $y(t) = 2/3e^{3t} + t + 1/3$ for $t > 0$.

- *An example of a nonlinear Cauchy problem is given by:*

$$\begin{cases} y'(t) = \sqrt[3]{y(t)} & \text{for } t > 0, \\ y(0) = 0, \end{cases} \tag{5.5}$$

  with $f(t, v) = \sqrt[3]{v}$. This problem admits the following three solutions : $y(t) = 0$, $y(t) = \sqrt{8t^3/27}$, $y(t) = -\sqrt{8t^3/27}$.

- *Another example of a nonlinear Cauchy problem is:*

$$\begin{cases} y'(t) = 1 + y^2(t) & \text{for } t > 0, \\ y(0) = 0. \end{cases} \tag{5.6}$$

  This problem has exact solution $y(t) = \tan(t)$ in the interval $0 < t < \frac{\pi}{2}$. But note that $y(t) \to \infty$ as $t \to \frac{\pi}{2}$, so the solution is only defined locally rather than on the whole of $\mathbb{R}_+$.

We recall the following result on existence and uniqueness for problem (5.3), which we quote without proof from Süli and Mayers.

Theorem 5.2.2 (Picard's Theorem). *Suppose that $f : D \to \mathbb{R}$ is continuous on the rectangle $D = [0, t_{\max}] \times [y_0 - c, y_0 + c]$, for some $t_{\max} > 0$ and $c > 0$, and that f satisfies a Lipschitz condition in D with respect to y, i.e. $\exists L > 0$ such that*

$$|f(t, v) - f(t, w)| \leq L|v - w|, \qquad \forall t \in [0, t_{\max}] \text{ and } \forall v, w \in [y_0 - c, y_0 + c]. \tag{5.7}$$

Suppose also that

$$K := \max_{t \in [0, t_{\max}]} |f(t, y_0)| \leq \frac{cL}{(\mathrm{e}^{Lt_{\max}} - 1)}. \tag{5.8}$$

Then there exists a unique continuously differentiable function $y : [0, t_{\max}] \to [y_0 - c, y_0 + c]$ such that $y(0) = y_0$ and $y'(t) = f(t, y)$ for $t \in [0, t_{\max}]$.

Remark 5.2.3. *This version of Picard's Theorem might look more complicated than other versions you have seen before. The extra condition (5.8) allows us to prove existence and uniqueness of the solution on the* specified *interval $[0, t_{\max}]$ rather than just on* some *interval $[0, \delta]$ as in more standard versions.*

In particular we note that if $f(t, y)$ is differentiable with respect to y throughout D with uniformly bounded partial derivative $\partial f/\partial y$, then the Lipschitz condition (5.7) is satisfied in D with $L = \max_{(t,y) \in D} |\partial f/\partial y(t, y)|$, by the mean value theorem.

In example (5.4), $|f(t, v) - f(t, w)| = |(3v - 3t) - (3w - 3t)| = 3|v - w|$ for any $t, v, w \in \mathbb{R}$. Hence (5.7) is satisfied with $L = 3$ and $c > 0$ arbitrary. Given any $t_{\max} > 0$, it is clear that $K = \max\{3, 3t_{\max}\} < 3(1 + t_{\max})$, and then we can satisfy (5.8) by taking $c > (1 + t_{\max})(\mathrm{e}^{3t_{\max}} - 1)$. Thus the Cauchy problem (5.3) has a global solution, i.e. $y(t)$ is uniquely defined for $t \in [0, \infty)$.

For example (5.5), the lack of unique solvability does not contradict Theorem 5.2.2 since the theorem doesn't apply in this case. This is because $f(t, y)$ is not Lipschitz with respect to y in any neighbourhood of $(0, 0)$. Indeed, taking $w = 0$ in (5.7) gives

$$|f(t, v) - f(t, w)| = |f(t, v)| = v^{\frac{1}{3}},$$

which tends to zero slower than Lv as $v \to 0$, for any constant $L > 0$. In other words, we cannot find $L > 0$ such that (5.7) is satisfied, no matter how small $t_{\max}$ and c are.

Finally, for example (5.6) we observe that

$$|f(t, v) - f(t, w)| = |v^2 - w^2| = |(v + w)||(v - w)|, \qquad \forall t, v, w \in \mathbb{R}.$$

Hence (5.7) holds with $L = 2c$. Also, $|f(t, 0)| = 1$. So for Theorem 5.2.2 to apply we need $2c^2 \geq (\mathrm{e}^{2ct_{\max}} - 1)$. This can be satisfied for $t_{\max}$ sufficiently small by choosing c appropriately, giving local existence and uniqueness. But for large enough $t_{\max}$ (e.g. $t_{\max} \geq 1$ is sufficient) this inequality cannot be satisfied for any $c > 0$, and hence we cannot use Theorem 5.2.2 to prove global existence and uniqueness. This is consistent with our observation that the exact solution blows up at finite t. (But notice that the theory breaks down before the exact solution actually blows up at $t = \pi/2$.)

5.3 Numerical discretization

To obtain an approximate solution to the Cauchy problem (5.3) on a bounded interval $[0, t_{\max}]$ we will adopt the so-called "finite difference" approach. We start by choosing an integer $N \geq 1$ and discretizing the interval $[0, t_{\max}]$ into N subintervals $[t_n, t_{n+1}]$, where the mesh points (not necessarily evenly spaced) satisfy

$$0 = t_0 < t_1 < \ldots < t_N = t_{\max},$$

and the associated mesh spacings are defined by

$$h_n = t_{n+1} - t_n, \qquad n = 0, 1, \ldots, N-1.$$

Our discrete approximation of the continuously differentiable function $y(t)$ will consist of a set of $N+1$ real numbers $\{u_n\}_{n=0}^N$, where for each n the number u_n represents an approximation of the value of $y(t)$ at the mesh point $t = t_n$.

To derive a model satisfied by $\{u_n\}_{n=0}^N$, we have to replace the differential equation $y'(t) = f(t, y(t))$ by a suitable approximation. Recall that the derivative $y'(t)$ of a differentiable function y is defined as

$$y'(t) = \lim_{h \to 0} \frac{y(t+h) - y(t)}{h}.$$

Hence, one way to approximate the derivative $y'(t)$ at $t = t_n$ is by a difference quotient $(y(t_n + h) - y(t_n))/h$ for some finite $h > 0$ (this is the origin of the term "finite difference" above). A natural choice is to take $h = h_n$, the local mesh spacing, because then we can replace the unknown values $y(t_n + h_n)$ and $y(t_n)$ by their discrete approximations u_{n+1} and u_n. Finally, replacing $f(t, y(t))$ by $f(t_n, u_n)$ leads to our first numerical method for solving (5.3), the **explicit** (or **forward**) **Euler method**:

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h_n f(t_n, u_n), \qquad n = 0, 1, \ldots, N-1. \end{cases} \tag{5.9}$$

The method is called "explicit" because the right-hand side of (5.9) is independent of u_{n+1}, meaning that u_{n+1} can be computed from u_n in a straightforward way. As we shall see later, the price to pay for this convenient feature is that the method may be unstable unless certain conditions are met.

Instead of replacing $f(t, y(t))$ by $f(t_n, u_n)$ we could instead replace it by $f(t_{n+1}, u_{n+1})$, and this leads to a different numerical method, called the **implicit** (or **backward**) **Euler method**:

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h_n f(t_{n+1}, u_{n+1}), \qquad n = 0, 1, \ldots, N-1. \end{cases} \tag{5.10}$$

This method is called "implicit" because the right-hand side of (5.10) depends on u_{n+1}, so in general a nonlinear equation has to be solved at each time step to find u_{n+1} from u_n. This can

be done using one of the methods described in §3, e.g. Newton's method, or another suitable fixed point iteration. (Question: can you suggest a sensible initial guess for the iteration?) While this increases the computational cost of the method compared to explicit Euler, we shall see that it brings certain advantages in terms of stability.

Another important method is obtained by replacing $f(t, y(t))$ by the *average* of $f(t_n, u_n)$ and $f(t_{n+1}, u_{n+1})$, which leads to the **Crank-Nicolson** or **trapezium rule method**:

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + \frac{h_n}{2}\big(f(t_n, u_n) + f(t_{n+1}, u_{n+1})\big), \qquad n = 0, 1, \ldots, N-1. \end{cases} \tag{5.11}$$

Again the method is implicit, so has good stability properties. As we shall see, it also has better approximation properties than either forward or backward Euler.

Example 5.3.1. *Consider the Cauchy problem*

$$\begin{cases} y'(t) = -ty^2(t), \quad t > 0, \\ y(0) = 2, \end{cases} \tag{5.12}$$

the exact solution which is given by $y(t) = 2/(1+t^2)$.

We will solve this equation numerically on the interval $[0, 4]$ *using both the forward and backward Euler methods. In both cases we use a constant time step* $h = 0.2$*, so the unknowns are approximations to the solution* $y(t)$ *at times* $t = 0.2, 0.4, 0.6, \ldots$.

forward Euler

The function **`feuler`** *evaluates the forward Euler iteration at each time step.*

```
>> f=@(t,y)-t.*y.^2;
>> tmax=4; N=20; y0=2;
>> [t_fe, u_fe] = feuler(f,[0,tmax],y0,N)
```

The variable **`t_fe`** *contains the sequence of time levels* t_n *and the variable* **`y_fe`** *contains the associated approximations* u_n *computed by the method.*

backward Euler

The function **`beuler`** *solves the nonlinear equation obtained at each step of the backward Euler method using Newton's method. For this reason we must also supply the derivative* $\frac{\partial f}{\partial y}$.

```
>> dfdy=@(t,y)-2*t.*y;
>> [t_be, u_be] = beuler(f,dfdy,[0,tmax],y0,N)
```

Figure 5.1 shows the exact solution along with the two approximate solutions. This figure was obtained using the commands

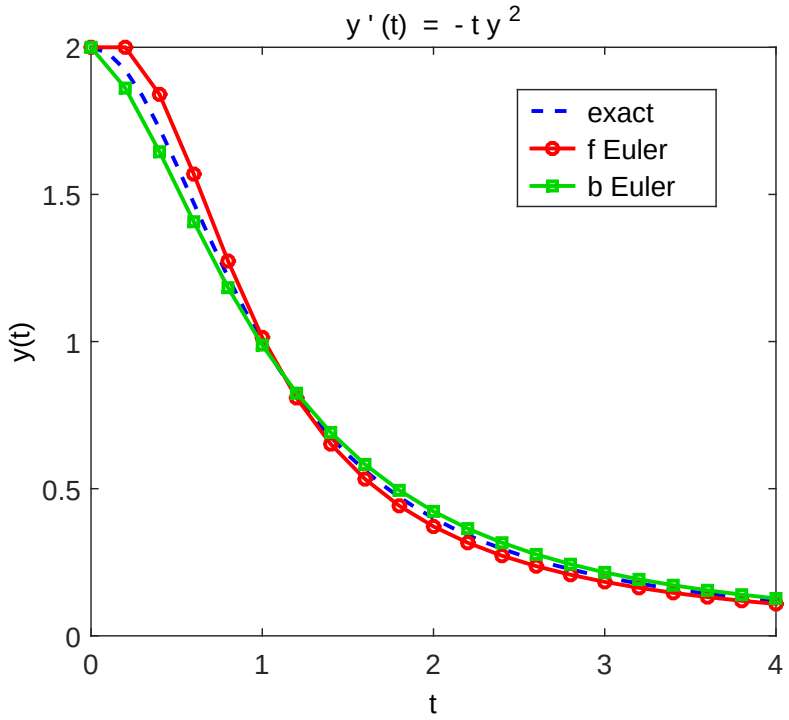


Figure 5.1: Comparison between the exact solution and the approximate solutions obtained using the forward and backward Euler methods.

```
>> t=[0:0.05:4]; y_ex=2./(1+t.^2);
>> plot(t,y_ex,'b',t_fe,u_fe,'ro-',t_be, u_be','go-')
>> legend('exact','f Euler','b Euler')
>> title('y^\prime(t) =- ty^2')
```

5.4 Discretizations from quadrature

One way of systematically deriving approximation schemes for (5.3) is to use numerical quadrature. First one integrates the equation $y'(t) = f(t, y(t))$ from t_n to t_{n+1} to obtain

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{5.13}$$

Then one applies a numerical quadrature rule to approximate the integral on the right-hand side of (5.13) in terms of values of $f(t, y(t))$ at a discrete set of points in the interval $[t_n, t_{n+1}]$. Replacing these values by their discrete counterparts leads to a numerical discretization of the differential equation.

A general class of quadrature rules for (5.13) is obtained by replacing the integrand $f(t, y(t))$ on $[t_n, t_{n+1}]$ by an appropriate polynomial approximation, which is then integrated exactly.

The simplest polynomial approximation is where we replace $f(t, y(t))$ by a constant $f(t_*, y(t_*))$ for some point $t_* \in [t_n, t_{n+1}]$. The resulting integral is trivial to evaluate, and leads to the approximation

$$y(t_{n+1}) - y(t_n) \approx h_n f(t_*, y(t_*)).$$

Different choices of t_* lead to different methods. In particular, the choice $t^* = t_n$ leads to the forward Euler method (5.9), once we replace $y(t_n)$ and $y(t_{n+1})$ by u_n and u_{n+1}. Similarly, the choice $t^* = t_{n+1}$ leads to the backward Euler method (5.10).

More accurate approximations of the integral $\int_{t_n}^{t_{n+1}} f(t, y(t))dt$ can be obtained using higher-order polynomial approximations of the integrand. The *trapezoidal rule* uses the linear approximation

$$f(t, y(t)) \approx \left(f(t_n, y(t_n)) \frac{t_{n+1} - t}{t_{n+1} - t_n} + f(t_{n+1}, y(t_{n+1})) \frac{t - t_n}{t_{n+1} - t_n} \right),$$

which can be integrated exactly (exercise: check this!) to give the approximation

$$y(t_{n+1}) - y(t_n) \approx \frac{h_n}{2} \big(f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1})) \big).$$

Replacing $y(t_n)$ and $y(t_{n+1})$ by u_n and u_{n+1} then leads to the Crank-Nicolson method (5.11).

5.5 One-step methods

The methods introduced above express u_{n+1} in terms of u_n, and as such are called *one-step methods*. There exists a large theory of more general *multi-step methods*, which express u_{n+1} in terms of u_{n+1-j}, $j = 0, 1, \ldots, k$ for some $k \geq 2$, but we shall not study these in this course.

We shall consider one-step methods of the general form

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h_n \Psi(t_n, u_n, u_{n+1}; h_n), \end{cases} \tag{5.14}$$

where $\Psi(\cdot, \cdot, \cdot; \cdot)$ is a continuous function of its arguments.

Example 5.5.1. *For forward Euler,*

$$\Psi(t_n, u_n, u_{n+1}; h_n) = f(t_n, u_n).$$

For backward Euler,

$$\Psi(t_n, u_n, u_{n+1}; h_n) = f(t_{n+1}, u_{n+1}) = f(t_n + h_n, u_{n+1}).$$

For Crank-Nicolson,

$$\Psi(t_n, u_n, u_{n+1}; h_n) = \frac{1}{2} \big(f(t_n, u_n) + f(t_{n+1}, u_{n+1}) \big) = \frac{1}{2} \big(f(t_n, u_n) + f(t_n + h_n, u_{n+1}) \big).$$

5.5.1 Truncation error

For a one-step method of the form (5.14) we define the *truncation error* T_n by

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h_n} - \Psi(t_n, y(t_n), y(t_{n+1}); h_n).$$

Note that the truncation error involves the *exact* values $y(t_n)$ and $y(t_{n+1})$, not their numerical approximations. It is a measure of the extent to which the numerical method approximates the differential operator $y'(t) - f(t, y(t))$. Precisely, rewriting the definition of T_n as

$$T_n = \frac{1}{h_n}\Big(y(t_{n+1}) - \big(y(t_n) + h_n\Psi(t_n, y(t_n), y(t_{n+1}); h_n)\big)\Big)$$

reveals that T_n is $1/h_n$ times the residual obtained by plugging the exact solution into the recurrence relation (5.14).

For simplicity, let us now suppose that we are working on a uniform mesh, with $t_n = nh$, $n = 0, \ldots, N$, where $h = t_{\max}/N$, so that $h_n = h$ for each n. Then for any $t \in [0, t_{\max}]$ we can consider the limiting behaviour of T_n when

$$h \to 0 \text{ and } n \to \infty \text{ in such a way that } t_n \to t. \tag{5.15}$$

We say that the method is *consistent* if $T_n \to 0$ for each $t \in [0, t_{\max}]$ with $n \to \infty$ as in (5.15). Since both y' and Ψ are continuous, it is easy to prove (exercise - check it!) that a one-step method is consistent if and only if $\Psi(t, y, y; 0) = f(t, y)$. To quantify consistency more precisely, we say the method is of order $p \in \mathbb{N}$ if $T_n = O(h^p)$ as $n \to \infty$ as in (5.15).

Example 5.5.2. *For the forward Euler method with constant step size h we have*

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_n, y(t_n)).$$

Assume that y is twice differentiable with y'' bounded on $[0, t_{\max}]$. Then by Taylor's theorem

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

for some $\xi_n \in [t_n, t_{n+1}]$. Since $y'(t_n) = f(t_n, y(t_n))$ (from the differential equation) we see that

$$|T_n| = \left|y'(t_n) - f(t_n, y(t_n)) + \frac{h}{2}y''(\xi_n)\right| = \left|\frac{h}{2}y''(\xi_n)\right| \leq \frac{h}{2}\max_{\xi\in[0,t_{\max}]}|y''(\xi)|,$$

so the method is of first order.

Example 5.5.3. *The backward Euler method is also of first order. For this, note that now*

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_{n+1}, y(t_{n+1})).$$

We expand $y(t_{n+1})$ as in the previous example. But now we also have to expand

$$f(t_{n+1}, y(t_{n+1})) = y'(t_{n+1}) = y'(t_n + h) = y'(t_n) + hy''(\eta_n),$$

for some $\eta_n \in [t_n, t_{n+1}]$, so that, provided y is twice differentiable with y'' bounded,

$$|T_n| = h\left|\frac{1}{2}y''(\xi_n) - y''(\eta_n)\right| \leq \frac{3h}{2}\max_{\xi\in[0,t_{\max}]}|y''(\xi)|.$$

Example 5.5.4. *The Crank-Nicolson method (5.11) is of second order, with $T_n = O(h^2)$. The truncation error is now*

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - \frac{1}{2}\left(f(t_n, y(t_n)) + f(t_{n+1}, y(t_{n+1}))\right),$$

and to prove $T_n = O(h^2)$ we proceed as for backward Euler, except that we expand to higher order than before. Specifically, combining the expansions

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(\xi_n)$$

and

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

shows that, assuming y is three times differentiable with y''' bounded,

$$|T_n| = h^2\left|\frac{1}{6}y'''(\xi_n) - \frac{1}{4}y'''(\eta_n)\right| \leq \frac{5h^2}{12}\max_{\xi\in[0,t_{\max}]}|y'''(\xi)|.$$

The truncation error is a measure of how well the numerical method approximates the differential operator on a local level. To investigate the global behaviour of the numerical solution u_n we consider the concepts of *stability* and *convergence*.

5.5.2 Zero stability

The concept of stability concerns the behaviour of the numerical method in the presence of small perturbations (for example those caused by the rounding errors inherent in finite precision arithmetic). The first type of stability we consider relates to the behaviour of the numerical method on a fixed time interval $[0, t_{\max}]$ as the mesh size h tends to zero, and is called *zero-stability*. Given a one-step method

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h\Psi(t_n, u_n, u_{n+1}; h), \end{cases} \tag{5.16}$$

we introduce the perturbed version

$$\begin{cases} v_0 = y_0 + \delta_0, \\ v_{n+1} = v_n + h\Psi(t_n, v_n, v_{n+1}; h) + h\delta_{n+1}, \end{cases} \tag{5.17}$$

where δ_n, $n = 0, 1, \ldots, N$, are some small perturbations. We can then ask, for perturbations of a given size, how the difference between the solutions u_n and v_n behaves in the limit $h \to 0$.

Definition 5.5.5. *The numerical method* (5.16) *for the approximation of* (5.3) *is said to be* zero-stable *if there exist* $h_* > 0$ *and* $C > 0$ *such that, for all* $\delta > 0$, *if*

$$h < h_* \text{ and } |\delta_n| \leq \delta, \quad 0 \leq n \leq N, \tag{5.18}$$

then the solution of the perturbed problem (5.17) *satisfies*

$$|v_n - u_n| \leq C\delta, \quad 0 \leq n \leq N. \tag{5.19}$$

Note that zero-stability is a property of the numerical method itself and not of the Cauchy problem. It ensures that the numerical method is insensitive to perturbations of data, which is necessary to control the accumulation of errors introduced by working in finite precision arithmetic when the method is implemented on a computer. A method that is not zero-stable is essentially useless for practical computation, because as one refines the mesh (in an attempt to achieve better accuracy) the numerical solution becomes increasingly polluted by rounding errors in the evaluation of y_0 and $f(t_n, u_n)$.

The following theorem provides sufficient conditions for a one-step method to be zero-stable, namely that the increment function Ψ satisfies a uniform Lipschitz condition.

Theorem 5.5.6. *Suppose that there exist* $h_* > 0$, $L_1 > 0$ *and* $L_2 > 0$ *such that*

$$|\Psi(t, u, v; h) - \Psi(t, u', v'; h)| \leq L_1|u - u'| + L_2|v - v'|, \qquad \forall t \in [0, t_{\max}], \forall 0 < h < h_* \text{ and } \forall v, w \in \mathbb{R}. \tag{5.20}$$

Then the one-step method (5.16) *is zero-stable.*

Proof. By the definitions of the numerical methods (5.16) and (5.17), the difference $w_n = v_n - u_n$ satisfies $w_0 = \delta_0$ and, for each $n = 0, \ldots, N-1$, the recurrence relation

$$w_{n+1} = w_n + h\big(\Psi(t_n, v_n, v_{n+1}; h) - \Psi(t_n, u_n, u_{n+1}; h)\big) + h\delta_{n+1}.$$

Applying the triangle inequality and the Lipschitz condition (5.20) shows that, for $0 < h < h_*$,

$$\begin{aligned} |w_{n+1}| &\leq |w_n| + h\big|\Psi(t_n, v_n, v_{n+1}; h) - \Psi(t_n, u_n, u_{n+1}; h)\big| + h|\delta_{n+1}| \\ &\leq |w_n| + hL_1|w_n| + hL_2|w_{n+1}| + h\delta, \end{aligned} \tag{5.21}$$

where $\delta = \max_{n=0,\ldots,N} |\delta_n|$, and hence that

$$(1 - hL_2)\,|w_{n+1}| \leq (1 + hL_1)\,|w_n| + h\delta. \tag{5.22}$$

Now, for $0 < h < 1/L_2$ this is equivalent to

$$|w_{n+1}| \leq \frac{(1+hL_1)}{(1-hL_2)}|w_n| + \frac{h\delta}{(1-hL_2)}. \tag{5.23}$$

Thus $|w_n| \leq x_n$ for each $n = 0, \ldots, N$, where $x_n > 0$ is the solution of the recurrence relation $x_0 = |w_0| = |\delta_0|$, $x_{n+1} = ax_n + b$, $n \geq 0$, where $a = (1+hL_1)/(1-hL_2) = 1 + h(L_1+L_2)/(1-hL_2) > 1$ and $b = h\delta/(1-hL_2) > 0$. A simple induction argument shows that

$$x_n = a^n x_0 + (1 + a + \ldots + a^{n-1})b = a^n x_0 + \frac{a^n - 1}{a-1}b, \qquad n = 1, 2, \ldots, \tag{5.24}$$

which gives (recalling that $w_0 = \delta_0$)

$$|w_n| \leq \left(1 + \frac{h(L_1+L_2)}{1-hL_2}\right)^n |\delta_0| + \frac{\delta}{L_1+L_2}\left(\left(1 + \frac{h(L_1+L_2)}{1-hL_2}\right)^n - 1\right), \quad n = 0, 1, \ldots, N. \tag{5.25}$$

Now, noting that

$$1 + x \leq \exp[x] = 1 + x + \frac{x^2}{2} + \ldots, \qquad x > 0, \tag{5.26}$$

the estimate (5.25) implies that (recalling that $nh = t_n$)

$$|w_n| \leq \exp\left[\frac{t_n(L_1+L_2)}{1-hL_2}\right]|\delta_0| + \frac{\delta}{L_1+L_2}\left(\exp\left[\frac{t_n(L_1+L_2)}{1-hL_2}\right] - 1\right), \quad n = 1, 2, \ldots, N, \tag{5.27}$$

and from this we conclude that

$$|v_n - u_n| = |w_n| \leq C\delta, \quad n = 1, 2, \ldots, N,\ 0 < h < \min(h_*, 1/L_2), \tag{5.28}$$

where

$$C = \exp\left[\frac{t_{\max}(L_1+L_2)}{1-hL_2}\right]\left(1 + \frac{1}{L_1+L_2}\right), \tag{5.29}$$

and hence the method is zero-stable. □

Example 5.5.7. *If f is uniformly Lipschitz on $\mathbb{R}$, i.e.*

$$|f(t,v) - f(t,w)| \leq L_f|v-w|, \qquad \forall t \in [0, t_{\max}] \text{ and } \forall v, w \in \mathbb{R},$$

then the forward Euler, backward Euler and Crank-Nicolson methods are all zero-stable, since they satisfy the uniform Lipschitz condition (5.20) of Theorem 5.5.6 with $(L_1, L_2) = (L_f, 0)$ (FE), $(L_1, L_2) = (0, L_f)$ (BE), and $(L_1, L_2) = (L_f/2, L_f/2)$ (CN).

We end this section by remarking that while the constant C in (5.18) must be independent of h (equivalently, of N), it may depend on the length $t_{\max}$ of the time interval on which we are solving (5.3). In particular it may happen that $C \to \infty$ as $t_{\max} \to \infty$, as is the case in (5.29). We will return to this issue later.

5.5.3 Convergence

Not only do we want our numerical solution u_n to be stable under perturbations of the data, we want it to be a good approximation to the exact solution y of the original Cauchy problem (5.3). Define the *global error* e_n to be the difference between the exact solution and the numerical approximation, i.e.

$$e_n = y(t_n) - u_n.$$

We say that the method is *convergent* if $e_n \to 0$ for each $t \in [0, t_{\max}]$ with $n \to \infty$ as in (5.15).

The following theorem is a version of the celebrated *Lax-Richtmyer equivalence principle*, which says that "stability plus consistency gives convergence". The theorem provides an explicit link between the global error e_n and the truncation error T_n, estimates of which, as we saw in Example 5.5.2, can be obtained by Taylor expansion.

Theorem 5.5.8. *Let f satisfy the assumptions of Theorem 5.2.2 (Picard's theorem). Suppose that the one-step method (5.16) is zero-stable, in the sense of Definition 5.5.5, and uniformly consistent, in the sense that*

$$T = \max_{n=0,\dots,N} |T_n|$$

tends to zero as $h \to 0$ (equivalently, as $N \to \infty$). Then the method is convergent, and

$$|e_n| \leq CT, \quad n = 0, 1, \dots, N, \tag{5.30}$$

for $h < h_*$, where C and h_* are the zero-stability constants from Definition 5.5.5. Hence if $T = O(h^p)$ as $h \to 0$ for some $p \in \mathbb{N}$ then also $\max_{n=0,\dots,N} |e_n| = O(h^p)$ as $h \to 0$.

In particular, for methods satisfying the Lipschitz assumption of Theorem 5.5.6, it holds that

$$|e_n| \leq \frac{T}{L_1 + L_2} \left(\exp\left[\frac{t_{\max}(L_1 + L_2)}{1 - hL_2} \right] - 1 \right), \quad n = 0, 1, \dots, N, \ 0 < h < \min(h_*, 1/L_2). \tag{5.31}$$

Proof. The key to the proof is to note that the exact solution $v_n = y(t_n)$ satisfies the perturbed equations (5.17) with $\delta_0 = 0$ and $\delta_{n+1} = T_n$, $n = 0, 1, \dots, N-1$. Thus, with h_* and C denoting the constants from Definition 5.5.5, if $h < h_*$ then zero-stability (with $\delta = T$) gives that $|e_n| = |v_n - u_n| \leq CT$, $n = 0, 1, \dots, N$, which proves the general result (5.30). For methods satisfying the Lipschitz assumption of Theorem 5.5.6 the more explicit error estimate (5.31) follows from equation (5.27) in the proof of Theorem 5.5.6, recalling that $\delta_0 = 0$. □

Example 5.5.9. *Combining Examples 5.5.2 and 5.5.7, we have that if f is uniformly Lipschitz then the forward and backward Euler methods are first-order convergent, and the Crank-Nicolson method is second-order convergent.*

5.5.4 Absolute stability

So far we have considered the approximation of the Cauchy problem (5.3) on bounded time intervals $[0, t_{\max}]$. Often one wishes to integrate the Cauchy problem on a very long time interval, typically to recover some asymptotic behaviour of the solution. In this case one wants a method that gives accurate approximation over long time intervals, using a time step h that remains relatively large, so as to keep the computational cost at an acceptable level.

An indication that this might not always be straightforward is provided by our convergence analysis from the previous section (specifically, Theorem 5.5.8). This showed that, for methods satisfying the Lipschitz assumption of Theorem 5.5.6, the global error satisfies

$$|e_n| \leq \frac{T}{L_1 + L_2} \left(\exp \left[\frac{t_{\max}(L_1 + L_2)}{1 - hL_2} \right] - 1 \right), \quad n = 1, 2, \ldots, N, \ 0 < h < 1/L_2, \tag{5.32}$$

where L_1, L_2 are the Lipschitz constants and T is the maximum truncation error. Thus, if the method is uniformly consistent (i.e. $T \to 0$ as $h \to 0$), then the numerical solution converges uniformly to the exact solution as $h \to 0$, on the fixed interval $[0, t_{\max}]$. However, because of the presence of the factor $\exp \left[\frac{t_{\max}(L_1+L_2)}{1-hL_2} \right]$, if we fix $h > 0$ and send $t_n \to \infty$ then the right-hand side of (5.32) blows up exponentially. Of course, it may be that the estimate (5.32) is not sharp and that the numerical solution is perfectly well behaved. But if the estimate (5.32) is sharp, and if $t_{\max}$ is very large ($t_{\max} \gg 1/(L_1 + L_2)$), then we may need to take h very small in order for the right-hand side of (5.32) to be of an acceptable size, for large n.

To correctly capture long-time behaviour, we need to consider methods which enjoy a stronger stability property than zero-stability, namely *absolute stability*. This concept is defined with reference to the following model problem:

$$\begin{cases} y'(t) = \lambda y(t), & t \in (0, \infty), \\ y(0) = 1, \end{cases} \tag{5.33}$$

where λ is real and negative. The exact solution is given by $y(t) = e^{\lambda t}$, and since $\lambda < 0$ we have that $y(t) \to 0$ as $t \to \infty$. Absolutely stable methods are those which correctly capture this asymptotic decay property.

Definition 5.5.10. *A one-step method of the form* (5.16) *is said to be* (conditionally) absolutely stable *for the problem* (5.33) *(for a given $\lambda < 0$) if there exists $h_* > 0$ such that for all fixed $h < h_*$ the numerical solution u_n of* (5.33) *computed using the method* (5.16) *satisfies*

$$u_n \longrightarrow 0 \quad \text{as} \quad t_n \longrightarrow \infty. \tag{5.34}$$

If this property holds for all $h > 0$ the method is said to be unconditionally absolutely stable, or $\mathcal{A}$-stable.

The simple nature of the problem (5.33) means that it is often fairly straightforward to investigate the absolute stability of a one-step method by computing the numerical solution

by hand. For example, for the forward Euler method we find the numerical solution

$$u_0 = 1, \quad u_{n+1} = u_n(1 + \lambda h) = (1 + \lambda h)^{n+1}, \quad n \geq 0, \tag{5.35}$$

and, consequently, $\lim_{n\to\infty} u_n = 0$ if and only if

$$-1 < 1 + h\lambda < 1, \quad i.e. \quad h < 2/|\lambda|. \tag{5.36}$$

By contrast, neither the backward Euler method nor the Crank-Nicolson method need any constraint on h for absolute stability to hold. Indeed, if we consider the backward Euler method applied to (5.33) we get $u_{n+1} = u_n + \lambda h u_{n+1}$ and therefore

$$u_{n+1} = \left(\frac{1}{1 - \lambda h}\right)^{n+1},$$

which tends to zero as $n \to \infty$ for every $h > 0$. Similarly, for the Crank-Nicolson method we find that

$$u_{n+1} = \left[\left(1 + \frac{h\lambda}{2}\right) \Big/ \left(1 - \frac{h\lambda}{2}\right)\right]^{n+1},$$

which again tends to zero as $n \to \infty$ for every $h > 0$. We conclude from this that the forward Euler method is conditionally absolutely stable (with $h_* = 2/|\lambda|$), whereas the backward Euler and Crank-Nicolson methods are unconditionally absolutely stable. As a general rule only implicit methods can be unconditionally absolutely stable.

Example 5.5.11. *We solve the model problem (5.33) with $\lambda = -2$ in the interval $[0, 10]$ using the forward and backward Euler methods with $h = 0.9$ and $h = 1.1$. In the case $h = 0.9$ we use the following Matlab commands:*

```
>> f=@(t,y)-2*y; tmax=10.8; N=12; y0=1;
% We take tmax=10.8 so that we have a whole number of subintervals.
>> [t_fe, u_fe] = feuler(f,[0,tmax],y0,N)
>> dfdy=@(t,y)-2;
>> [t_be, u_be] = beuler(f,dfdy,[0,tmax],y0,N)
>> t=[0:0.1:10]; y_ex=exp(-2*t);
>> plot(t,y_ex,'b',t_fe,u_fe,'ro-',t_be, u_be','go-')
>> xlim([0,10])
>> legend('exact','f Euler','b Euler')
>> title('y^\prime =-2y(t)')
```

Figure 5.2 shows the solutions obtained for $h = 0.9$ (left) and $h = 1.1$ (right) as well as the exact solution e^{-2t}. Note that, in accordance with the absolute stability theory, the forward Euler solution decays with increasing t for the case $h = 0.9 < 1$ but grows with increasing t for the case $h = 1.1 > 1$, and the backward Euler solution decays for both cases.

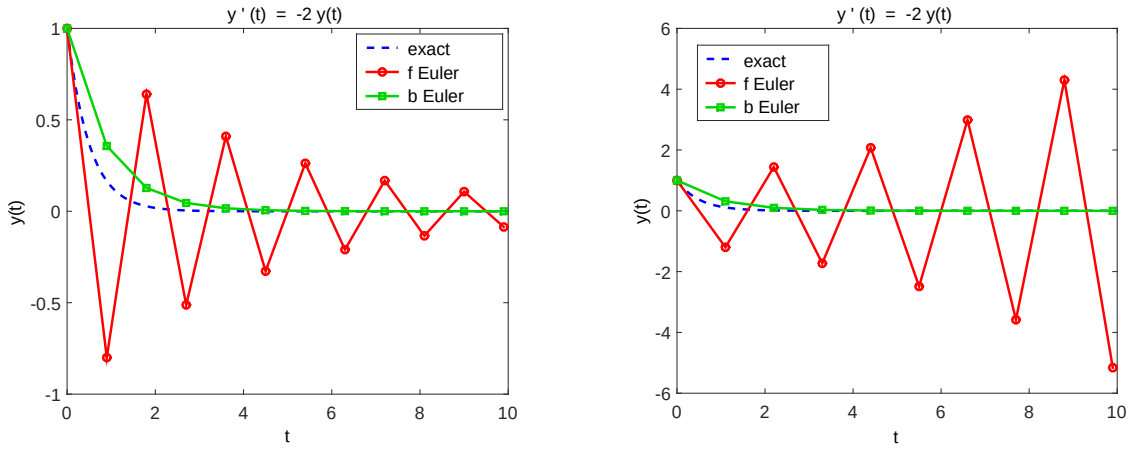


Figure 5.2: Comparison between the solutions of (5.33) with $\lambda = -2$ obtained using the forward and backward Euler methods with $h = 0.9$ (left) and $h = 1.1$ (right).

Remark 5.5.12. *The analysis above can be generalised to the case where λ is a negative function of t in (5.33). But if one wishes to use a uniform time step h then $|\lambda|$ must be replaced by $\max_{t\in[0,\infty)} |\lambda(t)|$ in the stability conditions. If one uses a variable time step h_n, taking into account the local behaviour of λ on each interval $[t_n, t_{n+1}]$, then the stability condition can be relaxed. One approach consists of an adaptive forward Euler method of the following form: Let $u_0 = y_0$ and $h_0 = 2\tilde{h}/|\lambda(t_0)|$ for some $\tilde{h} < 1$; and for $n = 0, 1, \ldots$ set*

$$\begin{aligned} t_{n+1} &= t_n + h_n, \\ u_{n+1} &= u_n + h_n \lambda(t_n) u_n, \\ h_{n+1} &= 2\tilde{h}/|\lambda(t_{n+1})|. \end{aligned} \tag{5.37}$$

The requirement that $\tilde{h} < 1$ ensures that the method is absolutely stable, in the sense that the asymptotic decay is correctly captured by the numerical solution.

At first glance, the concept of absolute stability appears to relate only to the very special Cauchy problem (5.33), for which one can easily write down an analytic solution. The real power of absolute stability is that it controls the behaviour of perturbations (e.g. those induced by rounding errors) in the numerical solution to much more general Cauchy problems.

To explore this idea, let us return to the general Cauchy problem (5.3), and the pair of numerical solutions u_n and v_n generated by the methods (5.16) and (5.17). Recall that (5.17) is supposed to represent a perturbed version of (5.16) in which small errors (e.g. due to rounding) are introduced at each step of the method. When studying zero-stability we considered the behaviour of the difference $u_n - v_n$ for fixed $t_{\max}$ as $h \to 0$. We now consider the behaviour of this difference for fixed h and $t_{\max} \to \infty$, and how this relates to absolute stability.

For simplicity we shall restrict our attention to explicit methods, for which $\Psi(t_n, u_n, u_{n+1}; h) =$

$\Psi(t_n, u_n; h)$. In this case, we recall from the proof of Theorem 5.5.6 (setting $L_2 = 0$ throughout) that the difference $w_n = v_n - u_n$ satisfies the recurrence relation

$$w_{n+1} = w_n + h\big(\Psi(t_n, v_n; h) - \Psi(t_n, u_n; h)\big) + h\delta_{n+1}. \tag{5.38}$$

When studying zero-stability we assumed Lipschitz continuity of $\Psi(\cdot, \cdot; \cdot)$ with respect to the second argument, which allowed us to bound the magnitude of the second term in (5.38) in terms of magnitude of w_n. This led (eventually) to the error estimate (5.32), which, as we discussed above, may not always be sharp in terms of its dependence on t_n.

To obtain a sharper bound we will make a stronger assumption about the increment function $\Psi(\cdot, \cdot; \cdot)$, namely that it is differentiable with respect to its second argument. By the mean value theorem one then has that

$$w_{n+1} = (1 + h\lambda_n)w_n + h\delta_{n+1}, \tag{5.39}$$

where $\lambda_n = \Psi_y(t_n, \xi_n; h)$ for some ξ_n between u_n and v_n; here Ψ_y denotes the partial derivative of Ψ with respect to its second argument. Then, taking absolute values and applying the triangle inequality, we obtain the estimate

$$|w_{n+1}| \leq |1 + h\lambda_n||w_n| + h|\delta_{n+1}|. \tag{5.40}$$

Suppose that Ψ_y is strictly negative, lying in the interval $-\lambda_{\max} \leq \Psi_y(t, y; h) \leq -\lambda_{\min}$ for all $t > 0$ and $y \in \mathbb{R}$, for some $0 < \lambda_{\min} \leq \lambda_{\max} < \infty$. Then by (5.24) we can show that

$$|w_n| \leq \delta\left(a^n + h\frac{a^n - 1}{a - 1}\right) = \delta\left(a^n\left(1 - \frac{h}{1-a}\right) + \frac{h}{1-a}\right), \quad n = 0, 1, \ldots, N, \tag{5.41}$$

where $\delta = \max_{n=0,\ldots,N} |\delta_n|$ and $a = \max\{|1 - h\lambda_{\min}|, |1 - h\lambda_{\max}|\}$, provided that $a \neq 1$. In particular we note that the difference $|w_n|$ remains bounded as $n \to \infty$ (rather than blowing up) provided that $a < 1$, in which case $\lim_{n\to\infty} |w_n| \leq \delta h/(1 - a)$. This requirement that $a < 1$ is equivalent to requiring that $h < 2/\lambda_{\max}$.

Now we can make the link to absolute stability. For the forward Euler method the increment function is given by $\Psi(t, y; h) = f(t, y)$, and so our assumptions on Ψ above correspond to requiring that the function $f(\cdot, \cdot)$ is differentiable with respect to its second argument, with partial derivative f_y satisfying $-\lambda_{\max} \leq f_y(t, y) \leq -\lambda_{\min}$ for all $t > 0$ and $y \in \mathbb{R}$. The requirement that $h < 2/\lambda_{\max}$, which ensures that numerical perturbations remain bounded as $n \to \infty$, is precisely the requirement that the method is absolutely stable for $\lambda = \lambda_{\max}$.

Remark 5.5.13. *The optimal approximation strategy (in terms of minimising computational cost) when computing long-time behaviour usually consists of choosing the largest possible h such that the method is absolutely stable (which for forward Euler requires $h < 2/\lambda_{max}$), and that sufficient accuracy is obtained. However, it can sometimes be difficult to determine the largest permissible step size because it might be difficult to compute $\lambda_{\max} = \max |f_y|$. One*

heuristic approach consists of an adaptive forward Euler method using a variable timestep strategy. Let $u_0 = y_0$ *and* $h_0 = 2\tilde{h}/|\lambda(t_0)|$ *for some* $\tilde{h} < 1$*; and for* $n = 0, 1, \ldots$ *set*

$$\begin{aligned} t_{n+1} &= t_n + h_n, \\ u_{n+1} &= u_n + h_n f(t_n, u_n), \\ h_{n+1} &= 2\tilde{h}/|f_y(t_{n+1}, u_{n+1})|. \end{aligned} \tag{5.42}$$

Example 5.5.14. *Consider the Cauchy problem*

$$\begin{cases} y'(t) = \arctan(3y) - 4y + t, \quad t > 0, \\ y(0) = 1. \end{cases}$$

The derivative $f_y = \partial f/\partial y = 3/(1+9y^2) - 4$ *is negative, with* $-4 < f_y < -1$ *for all* $t > 0$ *and* $y \in \mathbb{R}$*. Hence* $\lambda_{\max} = 4$ *and so the forward Euler method is stable to perturbations provided* $h < 2/4 = 1/2$*.*

Example 5.5.15. *Consider the Cauchy problem*

$$\begin{cases} y' = 1 - y^2, \quad t > 0, \\ y(0) = 0, \end{cases} \tag{5.43}$$

the exact solution to which is $y(t) = (\mathrm{e}^{2t} - 1)/(\mathrm{e}^{2t} + 1) = \tanh t$*. Note that* $f_y = \partial f/\partial y = -2y$*, which is unbounded for* $y \in \mathbb{R}$*. However, since the exact solution* y *satisfies* $y(t) \in (0, 1)$ *for* $t > 0$*, we have that* $f_y(t, y(t)) \in (-2, 0)$ *for all* $t > 0$*. Because of this we might try taking* $\lambda_{\max} = 2$*, which would suggest that for the forward Euler method to be stable to perturbations we need to take the step size* $h < 2/2 = 1$*. In Figure 5.3 we show the exact solution (dotted line), along with the approximate solutions obtained using* $h = 20/19 < 1$ *(dashed line) and* $h = 20/21 > 1$ *(continuous line). One can clearly see the error in the case* $h < 1$ *decaying with increasing* t*, whereas the error in the case* $h > 1$ *appears to be blowing up. These results suggest that in this particular case our proposed value of the critical step size is accurate, despite the fact that (i) it was computed only by considering the behaviour of* f_y *on the solution trajectory; and (ii) on this trajectory* $\lambda_{\min} = 0$*.*

Example 5.5.16. (Example 5.3.1 cont'd) *We return to the Cauchy problem (5.12) considered in Example 5.3.1. There we used a step size of* $h = 0.2$ *and the forward and backward Euler solutions on the interval* $[0, 4]$ *plotted in Figure 5.1 appeared to be roughly equivalent in their accuracy. However, when the step size is increased the quality of the forward Euler approximation rapidly deteriorates due to the lack of absolute stability. Figure 5.4 shows the solutions obtained using the forward and backward Euler methods with* $h = 0.8$*.*

5.6 Runge-Kutta methods

Despite its rather delicate stability properties, the forward Euler method is popular because it is an explicit method which is simple and cheap to implement. However, we noted in

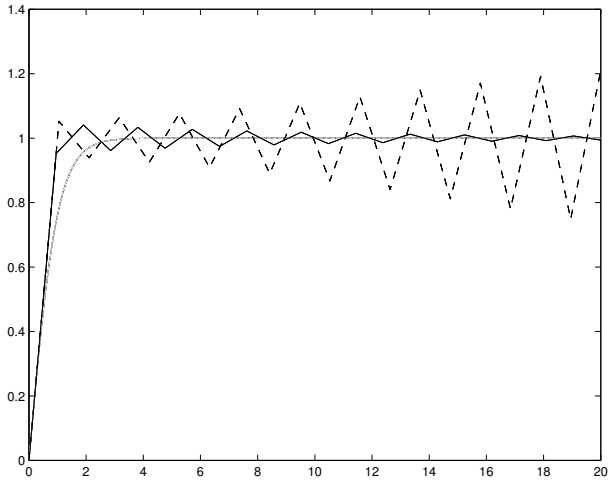


Figure 5.3: Numerical solutions for the problem (5.43) obtained using the forward Euler method and $h = 20/19$ (dashed line) and $h = 20/21$ (full line), along with the exact solution (dotted line).

Example 5.5.9 that it is only first order convergent. It is possible to derive more accurate (albeit slightly more expensive) explicit one-step methods by evaluating $f(\cdot, \cdot)$ not just at (t_n, u_n) but also at other points intermediate between (t_n, u_n) and (t_{n+1}, u_{n+1}). Such methods are called *Runge-Kutta* methods.

We recall from §5.4 that the methods considered so far can all be derived from the integral formula

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{5.44}$$

In particular, using the trapezoidal rule to approximate the integral we obtained the Crank-Nicolson method

$$u_{n+1} - u_n = \frac{h_n}{2} \left[f(t_n, u_n) + f(t_{n+1}, u_{n+1}) \right], \tag{5.45}$$

an implicit scheme which was shown to be unconditionally absolutely stable and second order accurate. It is possible to modify the scheme to render it explicit, but still conserve the second order accuracy (albeit at the expense of losing unconditional absolute stability). Indeed, by replacing the unknown u_{n+1} in the right-hand side of (5.45) by the result of a forward Euler step, we obtain so-called **improved Euler method**, or **Heun's method**:

$$u_{n+1} - u_n = \frac{h_n}{2} \left[f(t_n, u_n) + f(t_{n+1}, u_n + h_n f(t_n, u_n)) \right]. \tag{5.46}$$

Heun's method may also be written in the form

$$u^n \to \begin{cases} p_1 = f(t_n, u_n) \\ p_2 = f(t_{n+1}, u_n + h_n p_1) \\ u_{n+1} = u_n + \dfrac{h_n}{2}(p_1 + p_2). \end{cases}$$

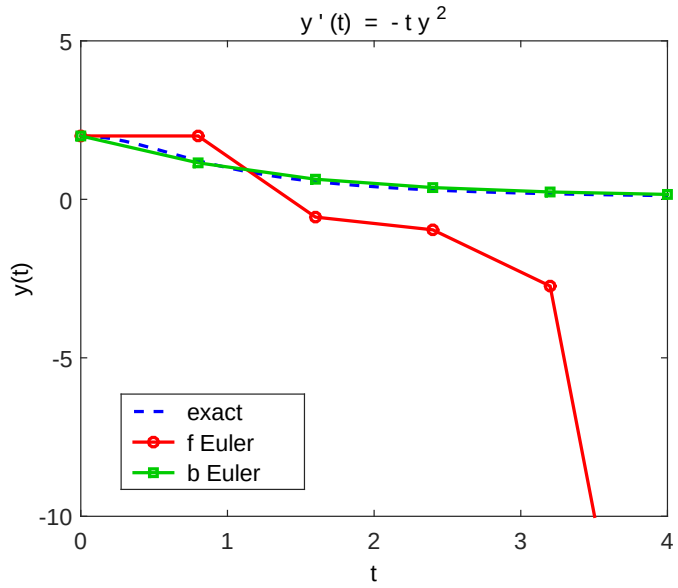


Figure 5.4: Comparison between the solutions of (5.12) obtained using the forward and backward Euler methods with $h = 0.8$.

A different method can be obtained by returning to (5.44) and using the midpoint rule rather than the trapezoidal rule, i.e. using a constant approximation to the integrand with $t_* = t_{n+1/2} = t_n + h_n/2 = (t_n + t_{n+1})/2$ in the notation of §5.4. This gives

$$u_{n+1} - u_n = h_n\, f(t_{n+\frac{1}{2}}, u_{n+\frac{1}{2}}), \tag{5.47}$$

where $u_{n+1/2}$ represents some approximation of the solution $y(t_{n+1/2})$. Calculating $u_{n+1/2}$ using half a forward Euler step, i.e. setting

$$u_{n+\frac{1}{2}} = u_n + \frac{h_n}{2} f(t_n, u_n),$$

gives the **modified Euler method**:

$$u_{n+1} - u_n = h_n\, f(t_{n+\frac{1}{2}}, u_n + \frac{h_n}{2} f(t_n, u_n)), \tag{5.48}$$

which can also be written in the form

$$u^n \to \begin{cases} p_1 = f(t_n, u_n) \\ p_2 = f(t_n + \dfrac{h_n}{2}, u_n + \dfrac{h_n}{2} p_1) \\ u_{n+1} = u_n + h_n p_2. \end{cases}$$

Theorem 5.6.1. *Heun's method and the modified Euler method are both of order 2. Both methods are absolutely stable under the condition that $h < 2/|\lambda|$.*

Proof. Exercise. To show $T_n = O(h^2)$, use Taylor expansion and the chain rule

$$\frac{d}{dh}F(T(h),Y(h)) = T'(h)\frac{\partial F}{\partial t}(T(h),Y(h)) + Y'(h)\frac{\partial F}{\partial y}(T(h),Y(h)).$$

□

Example 5.6.2. *Consider the Cauchy problem*

$$\begin{cases} y'(t) = (-0.1 + \cos t)\, y(t), \quad t > 0, \\ y(0) = 1, \end{cases} \tag{5.49}$$

which has exact solution $y(t) = \mathrm{e}^{-0.1t+\sin t}$.

We can solve this problem using the forward Euler method and Heun's method on the interval $[0, 12]$*, with* $h = 0.4$*, using the following Matlab code:*

```
>> f=@(t,y)(cos(t)-0.1)*y;
>> tmax=12; N=30; y0=1;
>> [t_fe, u_fe] = feuler(f,[0,tmax],y0,N)
>> [t_heun, u_heun] = heun(f,[0,tmax],y0,N)
```

In Figure 5.5 we present the solutions obtained using the two methods, along with the exact solution. It is clear that for this fixed step size the approximation produced by Heun's method is much more accurate than that of the forward Euler method. Nevertheless if the step size is reduced in the latter method it will still converge (albeit slowly) to the exact solution, as can be seen in Figure 5.6 where the approximations obtained for $h = 0.4, 0.2, 0.1, 0.05$ *are reported.*

```
>> t=linspace(0,tmax,100);
>> y_ex = exp(-0.1*t+sin(t));
>> plot(t,y_ex,'b--'); hold on
>> N=30;
>> for i=1:4
     [t_fe, u_fe] = feuler(f,[0,tmax],y0,N);
     plot(t_fe,u_fe)
     N=2*N;
   end
```

Finally we study the convergence order of the two methods. To do this we solve the problem using a number of different time steps and compute the errors in the resulting approximations, evaluated at time $t = 6$*, when compared to the exact solution.*

```
>> N=15; tmax=6;
>> y6 = exp(-0.1*tmax+sin(tmax));
```

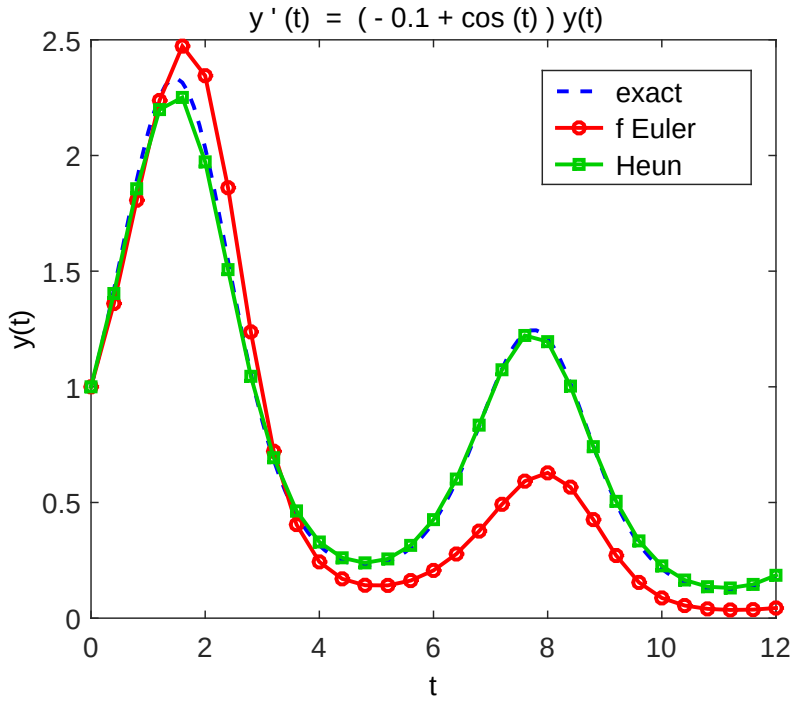


Figure 5.5: Comparison between the solutions of (5.49) obtained using the forward Euler method and Heun's method, with $h = 0.4$.

```
>> for i=1:5
     [t_fe, u_fe] = feuler(f,[0,tmax],y0,N);
     err_fe(i) = abs(y6 - u_fe(end));
     [t_heun, u_heun] = heun(f,[0,tmax],y0,N);
     err_heun(i) = abs(y6 - u_heun(end));
     N=2*N;
   end
>> h=[0.4, 0.2, 0.1, 0.05, 0.025];
>> loglog(h,err_fe,'b',h,err_heun,'r')
```

In Figure 5.7 the errors of the two methods are plotted against h on a logarithmic scale. First order convergence is observed for the explicit Euler method and second order convergence for Heun's method, as predicted by the theory.

One can construct higher order Runge-Kutta methods by increasing the number of interior stages of the method. The following scheme is a classical explicit Runge-Kutta method of

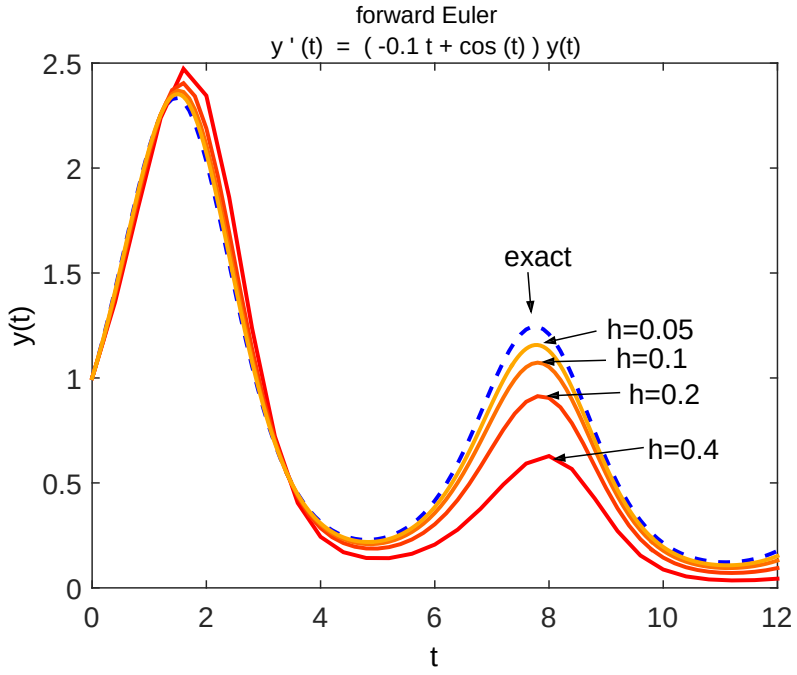


Figure 5.6: Solutions of (5.49) obtained using forward Euler and different timesteps.

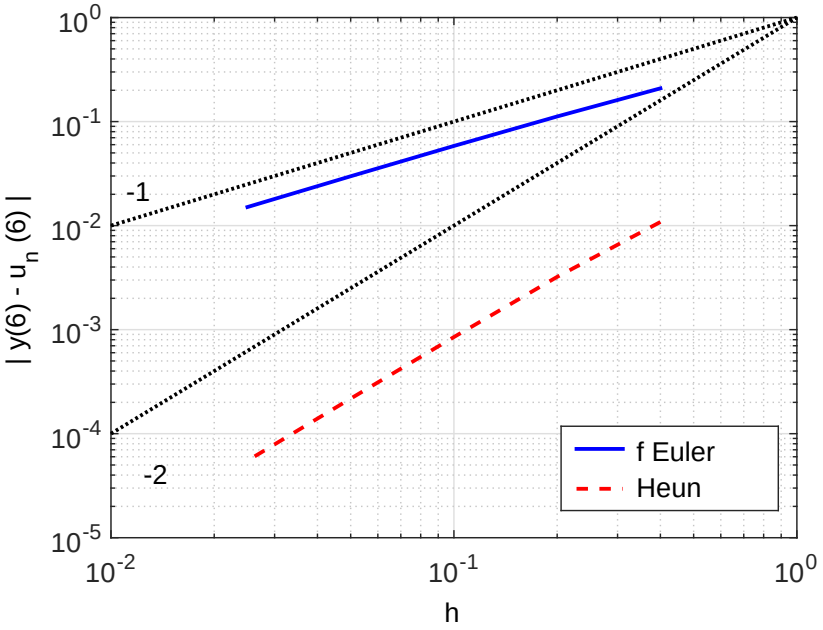


Figure 5.7: Errors on a logarithmic scale plotted against the timestep h for the forward Euler method and Heun's method approximating $y(6)$ for the problem (5.49).

order 4, often known simply as RK4.

$$u_n \to \begin{cases} p_1 = f(t_n, u_n) \\ p_2 = f(t_n + \dfrac{h_n}{2}, u_n + \dfrac{h_n}{2} p_1) \\ p_3 = f(t_n + \dfrac{h_n}{2}, u_n + \dfrac{h_n}{2} p_2) \\ p_4 = f(t_{n+1}, u_n + h_n p_3) \\ u_{n+1} = u_n + \dfrac{h_n}{6}\big(p_1 + 2p_2 + 2p_3 + p_4\big). \end{cases}$$

One can also consider implicit Runge-Kutta methods which have improved stability properties. However, we will not study these methods here.

Remark 5.6.3. *In Matlab several algorithms are already implemented for the solution of initial value problems (see for example* `ode45`*,* `ode23`*,* `ode23s`*,* `ode15s`*). In particular,* `ode45` *uses a fourth order method similar to the fourth order Runge-Kutta method RK4 presented above, combined with a fifth order method to compute an estimate of the error and adapt the local time step in order to satisfy a prescribed tolerance on the estimated error.*

5.7 Systems of ordinary differential equations

Much of the above analysis extends in a natural way to higher-dimensional systems of ODEs, where the Cauchy problem (5.3) is replaced by the multi-dimensional analogue (cf. (5.2))

$$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), & t > 0, \\ \mathbf{y}(0) = \mathbf{y}_0. \end{cases}$$

Here $\mathbf{y} : [0, \infty) \to \mathbb{R}^p$ for some $p \in \mathbb{N}$ and $\mathbf{f} : [0, \infty) \times \mathbb{R}^p \to \mathbb{R}^p$.

For instance, the forward Euler method becomes

$$\begin{cases} \mathbf{u}_0 = \mathbf{y}_0, \\ \mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{f}(t_n, \mathbf{u}_n), & n = 0, 1, 2, \dots, \end{cases} \tag{5.50}$$

the backward Euler method takes the form

$$\begin{cases} \mathbf{u}_0 = \mathbf{y}_0, \\ \mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{f}(t_{n+1}, \mathbf{u}_{n+1}), & n = 0, 1, 2, \dots, \end{cases} \tag{5.51}$$

and the Crank-Nicolson method is

$$\begin{cases} \mathbf{u}_0 = \mathbf{y}_0, \\ \mathbf{u}_{n+1} = \mathbf{u}_n + \dfrac{h}{2}\left(\mathbf{f}(t_n, \mathbf{u}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}_{n+1})\right), & n = 0, 1, 2, \dots. \end{cases} \tag{5.52}$$

Concerning stability, one has to consider the eigenvalues of the Jacobian matrix of partial derivatives

$$\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t,\mathbf{y}) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1}(t,\mathbf{y}) & \dots & \frac{\partial f_1}{\partial y_n}(t,\mathbf{y}) \\ \vdots & & \\ \frac{\partial f_n}{\partial y_1}(t,\mathbf{y}) & \dots & \frac{\partial f_n}{\partial y_n}(t,\mathbf{y}) \end{bmatrix}.$$

Suppose that the eigenvalues $\lambda_j(t,\mathbf{y})$, $j = 1,\dots,p$, of $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t,\mathbf{y})$ are all distinct, real and negative. Then the backward Euler method and the Crank-Nicolson method are both unconditionally stable to perturbations, whereas the forward Euler method is stable under the condition that

$$h < \frac{2}{\max_{t,y} \max_{j=1,\dots,p} |\lambda_j(t,\mathbf{y})|} = \frac{2}{\max_{t,y} \rho(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t,\mathbf{y}))}. \tag{5.53}$$

Regarding implementation, explicit methods (such as forward Euler) just require evaluations of the function $\mathbf{f}$ to compute each increment. But implicit methods (such as backward Euler or Crank-Nicolson) require the solution of a system of nonlinear equations, using for example a fixed point method such as Newton's method. (In the special case where the function $\mathbf{f}$ is a linear function of $\mathbf{y}$, i.e. $\mathbf{f}(t,\mathbf{y}) = A(t)\mathbf{y} + \mathbf{b}(t)$ for some $A \in \mathbb{R}^{p\times p}$ and $\mathbf{b}(t) \in \mathbb{R}^p$, one has to solve a linear system of equations, using for example one of the iterative methods considered in §4.) This makes implicit methods considerably more expensive than explicit methods in general, and so in practical applications a great deal of thought goes into deciding how to trade off computational cost against stability, especially when the size of the systems involved is large.

5.8 Applications

We conclude our study of initial value problems by revisiting the examples introduced at the beginning of the chapter.

Example 5.8.1. *(***Example 5.1.1 continued***)* *Let us first consider the scalar equation (5.1). Let the initial population consist of* 40 *rabbits with a reproductive factor of* $C = 0.08$ *(where the time unit is one month) and a maximum population of* $B = 70$ *rabbits. We solve the equation using Heun's method with* $h = 1$ *(month) over a period of three years:*

```
>> f=@(t,y)0.08*y.*(1-(y/70)); tmax=36; N=36; y0=40;
>> [t,y] = heun(f,[0,tmax],y0,N); plot(t,y)
```

The resulting solution is shown in Figure 5.8.

To compute the solution using the built-in Matlab routine `ode45` *(which, as discussed above, uses an adaptive fourth order method similar to the fourth order Runge-Kutta method presented above), one could use the commands*

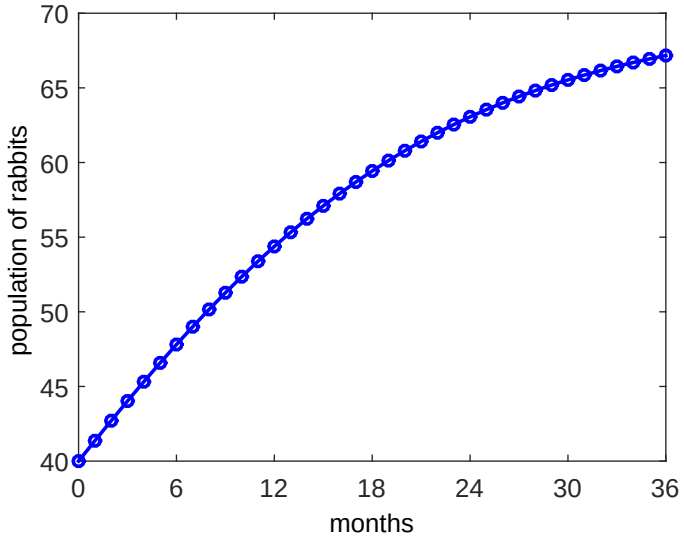


Figure 5.8: Solution of the scalar Lotka-Volterra equation (5.1).

```
>> options=odeset('RelTol',1e-4);
>> [t,y] = ode45(f,[0,tmax], y0, options);
```

The first command sets the options for the solver. Here we use a relative tolerance of 10^{-4} *(type* ***`help odeset`*** *to see all the options). The second command calls the routine approximating the solution of the Cauchy problem defined by the function* ***`f`****, over the interval* ***`[0, tmax]`*** *and with initial condition* ***`y0`****.*

Let us now turn to the system (5.2). Once again we consider an initial population of $y_1(0) = 40$ *rabbits but now we also introduce an initial population* $y_2(0)$ *of 20 foxes and consider the Lotka-Volterra system*

$$\begin{cases} y_1'(t) = 0.08\, y_1(t) - 0.004\, y_1(t) y_2(t), \\ y_2'(t) = -0.06\, y_2(t) + 0.002\, y_1(t) y_2(t). \end{cases} \tag{5.54}$$

If the following vectors are introduced

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}, \qquad \mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} 0.08\, y_1(t) - 0.004\, y_1(t) y_2(t) \\ -0.06\, y_2(t) + 0.002\, y_1(t) y_2(t) \end{bmatrix},$$

then the system (5.54) may be written in the form:

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \quad t > 0, \qquad \mathbf{y}(0) = [y_1(0), y_2(0)]^T. \tag{5.55}$$

The forward Euler method

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{h_n} = \mathbf{f}(t_n, \mathbf{u}^n)$$

would give the numerical scheme

$$\begin{cases} \dfrac{u_1^{n+1}-u_1^n}{h_n} = 0.08\,u_1^n - 0.004\,u_1^n u_2^n, \quad n \geq 0 \\ \dfrac{u_2^{n+1}-u_2^n}{h_n} = -0.06\,u_2^n + 0.002\,u_1^n u_2^n, \quad n \geq 0 \\ u_1^0 = y_1(0), \quad u_2^0 = y_2(0), \end{cases}$$

which can be easily implemented in Matlab. Alternatively, one can use a built-in Matlab solver such as `ode45`*:*

```
>>  y0=[40;20]; tmax=120;
>>  f=@(t,y)[0.08*y(1) - 0.004*y(1)*y(2);-0.06*y(2) + 0.002*y(1)*y(2)];
>>  options=odeset('RelTol',1e-4);
>>  [t,y] = ode45(f,[0,tmax], y0, options);
>>  plot(t,y(:,1),'b', t,y(:,2),'r')
```

The first column of `y` *contains the solution* y_1 *and the second* y_2*.*

In Figure 5.9 the evolution of the two populations over a time interval of 10 years (120 months) is plotted. Observe that the timestep used by the `ode45` *solver is not uniform across the whole time interval.*

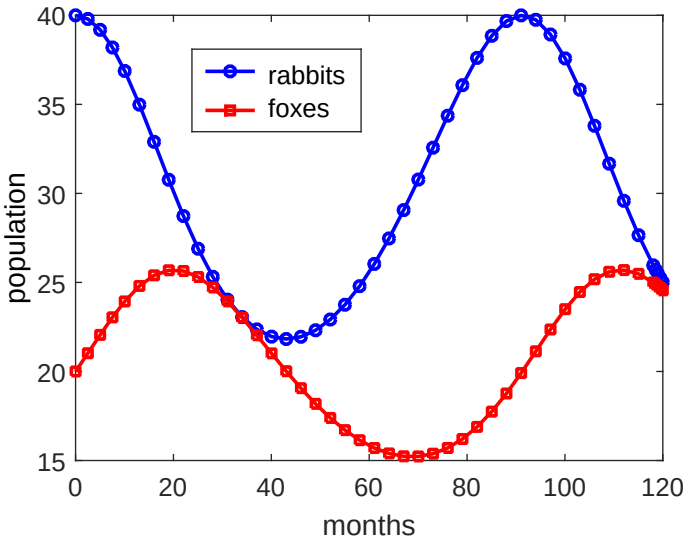


Figure 5.9: Evolution of the populations of rabbits and foxes during a 10 year period, computed using the Matlab `ode45` solver.

5.9 Boundary value problems

So far we have considered the numerical solution of initial value problems, where initial conditions are prescribed at one end of the interval on which we solve the differential equation. In this section we turn our attention briefly to the study of *boundary value problems* (BVPs), in which boundary conditions are prescribed at both ends of the interval. The study of BVPs has a huge literature and there are many powerful numerical methods available. In this course we shall just consider the application of *finite difference* approximations, as we did in the study of initial value problems. More advanced techniques such as the finite element method will not be considered here.

We shall restrict our attention mostly to one-dimensional second-order linear BVPs of the form

$$\begin{cases} L(y)(x) := -y''(x) + r(x)y(x) = f(x), & x \in (0, X), \\ y(0) = y_0, \quad y(X) = y_X. \end{cases} \tag{5.56}$$

Here $X > 0$ is the length of the interval on which the problem is posed, y_0 and y_X are the prescribed boundary values, and r and f are given continuous functions on $[0, X]$ with

$$r(x) \geq 0, \quad \forall x \in [0, X]. \tag{5.57}$$

The problem (5.56) can be shown to be well-posed. We shall consider two possible ways of solving it numerically.

5.9.1 Finite difference approximation

To obtain a numerical approximation to the solution of (5.56) we begin, as in the case of initial value problems, by discretizing the interval $[0, X]$ using a uniform mesh with mesh points $x_n = nh$, $n = 0, 1, \ldots, N$, where $h = X/N$ for some $N \in \mathbb{N}$. Letting u_n, $n = 0, 1, \ldots, N$, denote our approximation to $y(x_n)$, we replace the continuous system (5.56) by the discrete approximation

$$\begin{cases} L_N(u_n) := -D^2 u_n + r_n u_n = f_n, & n = 1, \ldots, N-1, \\ u_0 = y_0, \quad u_N = y_X, \end{cases} \tag{5.58}$$

where $r_n = r(x_n)$, $f_n = f(x_n)$, and the operator D^2 is defined, for any set of numbers v_n, $n = 0, 1, \ldots, N$, by

$$D^2 v_n = \frac{v_{n-1} - 2v_n + v_{n+1}}{h^2}, \qquad n = 1, \ldots, N-1.$$

Provided that u is four times continuously differentiable on $[0, X]$ (see Exercise sheet)

$$u''(x_n) - D^2 u(x_n) = O(h^2), \qquad \text{as } h \to 0, \tag{5.59}$$

uniformly on $[0, X]$.

The discrete system (5.58) can be written more compactly as a linear system

$$A\mathbf{u} = \mathbf{b} \tag{5.60}$$

of size $(N-1)$-by-$(N-1)$, where $\mathbf{u} = (u_1, \ldots, u_{N-1})^T$,

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & & & \vdots \\ 0 & -1 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & & -1 & 0 \\ \vdots & & & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} + \begin{pmatrix} r_1 & 0 & \cdots & & \cdots & 0 \\ 0 & r_2 & 0 & & & \vdots \\ \vdots & 0 & \ddots & \ddots & & \\ & & \ddots & & 0 & \vdots \\ \vdots & & & 0 & r_{N-2} & 0 \\ 0 & \cdots & & \cdots & 0 & r_{N-1} \end{pmatrix}, \tag{5.61}$$

and

$$\mathbf{b} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{pmatrix} + \frac{1}{h^2}\begin{pmatrix} u_0 \\ 0 \\ \vdots \\ 0 \\ u_N \end{pmatrix}. \tag{5.62}$$

Note that the matrix A describes the action of the discrete operator L_N and the boundary conditions are incorporated into the first and last elements of the vector $\mathbf{b}$.

Note that A is symmetric and diagonally dominant by rows, but not strictly diagonally dominant unless $r_n > 0$ for all n. That A is invertible follows from the fact that it is positive definite, which can be checked by noting that, for any $\mathbf{0} \neq \mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^{N-1}$,

$$\mathbf{x}^T A\mathbf{x} \geq x_1^2 + x_{N-1}^2 + \sum_{n=2}^{N-1}(x_n - x_{n-1})^2 > 0. \tag{5.63}$$

Hence the numerical solution u_n is uniquely defined.

To determine the accuracy of the numerical approximation we introduce the truncation error

$$T_n = L_N(y(x_n)) - f_n,$$

and the global error

$$e_n = y(x_n) - u_n,$$

which, by the definition of the numerical solution, satisfy the relation

$$L_N(e_n) = T_n. \tag{5.64}$$

From (5.59) and the fact that $y(t)$ satisfies the BVP (5.56) we know that

$$T = \max_{n=1,\ldots,N-1} |T_n| = O(h^2), \qquad \text{as } h \to 0. \tag{5.65}$$

Hence if we can prove a stability bound on the operator L_N (i.e. a bound on $|e_n|$ given a bound on $|T_n|$), we can derive an estimate of the global error e_n. There are different ways to achieve this; here we shall prove stability with respect to the supremum norm using the *discrete maximum principle* and the concept of a so-called *companion function*.

Theorem 5.9.1 (Discrete maximum principle)**.** *Let a_n, b_n, c_n, $n = 1, \ldots, N-1$ be positive real numbers such that $b_n \geq a_n + c_n$, and let U_n, $n = 0, \ldots, N$, be real numbers such that*

$$-a_n U_{n-1} + b_n U_n - c_n U_{n+1} \leq 0, \qquad n = 1, \ldots, N-1.$$

Then

$$U_n \leq \max\{U_0, U_N, 0\}, \qquad n = 1, \ldots, N-1.$$

Proof. See exercise sheet. □

To use this result to prove stability, we introduce the quadratic "companion function"

$$\varphi(x) = \frac{x(X-x)}{2},$$

which satisfies the BVP

$$\varphi''(x) = -1, \quad x \in (0, X), \qquad \text{with } \varphi(0) = \varphi(X) = 0.$$

Furthermore, we note that $\varphi(x) \geq 0$ for all $x \in [0, X]$, with $\phi(x)$ attaining its maximum value of $X^2/8$ at the midpoint $x = X/2$. The discrete samples $\varphi_n = \varphi(x_n)$ satisfy

$$D^2\varphi_n = -1, \quad n = 1, \ldots, N-1, \qquad \text{with } \varphi_0 = \varphi_N = 0,$$

and $0 \leq \varphi_n \leq X^2/8$ for $n = 0, \ldots, N$.

The point of the companion function is that it allows us to bound the solution error in terms of the truncation error. Indeed, where $T = \max_{n=1,\ldots,N-1} |T_n|$, it holds that

$$|e_n| \leq T\varphi_n, \qquad n = 0, \ldots, N. \tag{5.66}$$

To prove this, we first note that

$$L_N(\varphi_n) = 1 + r_n\varphi_n \geq 1, \qquad n = 1, \ldots, N-1, \tag{5.67}$$

which means that, by (5.64),

$$L_N(e_n - T\varphi_n) = T_n - TL_N(\varphi_n) \leq T_n - T \leq 0, \qquad n = 1, \ldots, N-1. \tag{5.68}$$

Then, recalling the definition of L_N and its matrix representation A, application of the discrete maximum principle (Theorem 5.9.1) with $a_n = c_n = 1/h^2$, $b_n = 2/h^2 + r_n$ and $U_n = e_n - T\varphi_n$, tells us that $e_n - T\varphi_n \leq 0$ for all $n = 0, \ldots, N$. (Note that $e_0 - T\varphi_0 = e_N - T\varphi_N = 0$.) The same argument, with e_n replaced by $-e_n$, shows that $-e_n - T\varphi_n \leq 0$ for all $n = 0, \ldots, N$, and together these two inequalities imply (5.66).

Finally, recalling that $0 \leq \varphi_n \leq X^2/8$ for $n = 0, \ldots, N$, (5.66) implies the stability bound

$$\max_{n=0,\ldots,N} |e_n| \leq \frac{X^2 T}{8}, \tag{5.69}$$

and combining this with the consistency result (5.65), we deduce that the numerical method is second order convergent, with

$$\max_{n=0,\ldots,N} |e_n| = O(h^2), \qquad \text{as } h \to 0. \tag{5.70}$$

5.9.2 Shooting methods

The finite difference approximation in the previous section was developed for the *linear* BVP (5.56). It is possible to extend this methodology to deal with more general nonlinear BVPs of the form

$$\begin{cases} y''(x) = f(x, y(x)), & x \in (0, X), \\ y(0) = y_0, \quad y(X) = y_X, \end{cases} \tag{5.71}$$

where f is assumed to be continuous with respect to both arguments, and continuously differentiable with respect to the second argument. But rather than describing how to do this here, we instead briefly describe another classical solution methodology, which builds on the methods we studied earlier in this chapter for the solution of initial value problems (IVPs).

The basic idea is to construct from the BVP (5.71) a one-parameter family of IVPs

$$\begin{cases} y''(x) = f(x, y(x)), & x \in (0, X), \\ y(0) = y_0, \quad y'(0) = s, \end{cases} \tag{5.72}$$

where s is a real parameter, called the *shooting parameter*. This second-order IVP can be expressed in the form we considered before, if we rewrite it as a first-order system (cf. §5.7)

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)), & x \in (0, X), \\ \mathbf{y}(0) = \mathbf{y}_0 = (s, y_0)^T, \end{cases} \tag{5.73}$$

where

$$\mathbf{y}(x) = (y_1(x), y_2(x))^T = (y'(x), y(x))^T \quad \text{and} \quad \mathbf{f}(x, \mathbf{y}(x)) = (f(x, y_2(x)), y_1(x))^T.$$

For any fixed s the system (5.73) can be solved using one of the methods described earlier in the chapter. Suppose we write the resulting numerical solution as $u_n(s)$, so as to make clear the dependence on the parameter s. Then the goal is to choose the parameter s so as to make $u_N(s)$, the value of the numerical solution at the right endpoint $x = X$, equal to the prescribed boundary data y_X.[1] To achieve this we have to solve the nonlinear equation

$$g(s) = 0,$$

where

$$g(s) = u_N(s) - y_X,$$

which we can do using one of the nonlinear equation solvers described in Chapter 3. Note that each evaluation of the function $g(s)$ requires us to carry out an IVP solve. For Newton's method one also requires the derivative $dg/ds(s) = du_N/ds(s)$, but it turns out that this can also be computed as the numerical solution to a certain IVP. For further details we refer to section 13.7 of Süli and Mayers.

[1] This explains the origin of the term "shooting parameter" - one can think of this procedure in terms of "shooting" or "launching" solutions trajectories at different launch angles, and trying to find the launch angle for which the resulting trajectory hits the "target" point (X, y_X).