PART ONE

PRELIMINARIES

1
Introduction

In this book, we are concerned with the basic problem of solving a linear system

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

where $A \in \mathbb{R}^{n\times n}$ is a given invertible matrix, $\mathbf{b} \in \mathbb{R}^n$ is a given vector and $\mathbf{x} \in \mathbb{R}^n$ is the solution we seek. The solution is, of course, given by

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

but does this really help if we are interested in actually computing the solution vector $\mathbf{x} \in \mathbb{R}^n$? What are the problems we are facing? First of all, such linear systems have a certain background. They are the results of other mathematical steps. Usually, they are at the end of a long processing chain which starts with setting up a partial differential equation to model a real-world problem, continues with discretising this differential equation using an appropriate approximation space and method, and results in such a linear system. This is important because it often tells us something about the structure of the matrix. The matrix might be symmetric or *sparse*. It is also important since it tells us something about the size n of the matrix. With simulations becoming more and more complex, this number nowadays becomes easily larger than a million, even values of several hundreds of millions are not unusual. Hence, the first obstacle that we encounter is the size of the matrix. Obviously, for larger dimensions n it is not possible to solve a linear system by hand. This means we need an algorithmic description of the solution process and a computer to run our program.

Unfortunately, using a computer leads to our second obstacle. We cannot represent real numbers accurately on a computer because of the limited number system used by a computer. Even worse, each calculation that we do might lead to a number which is not representable in the computer's number system.

Hence, we have to address questions like: Is a matrix that is invertible in the real numbers also invertible in the number system used by a computer? What are the errors that we make when representing the matrix in the computer and when using our algorithm to compute the solution. Further questions that easily come up are as follows.

1. How expensive is the algorithm? How much time (and space) does it require to solve the problem? What is the best way of measuring the cost of an algorithm?
2. How stable is the algorithm? If we slightly change the input, i.e. the matrix A and/or the right-hand side $\mathbf{b}$, how does this affect the solution?
3. Can we exploit the structure of the matrix A, if it has a special structure?
4. What happens if we do not have a square system, i.e. a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $\mathbf{b} \in \mathbb{R}^m$. If $m > n$ then we have an *over-determined* system and usually cannot find a (unique) solution but might still be interested in something which comes close to a solution. If $m < n$ we have an *under-determined* system and we need to choose from several possible solutions.

Besides solving a linear system, we will also be interested in a related topic, the computation of *eigenvalues* and *eigenvectors* of a matrix. This means we are interested in finding numbers $\lambda \in \mathbb{C}$ and vectors $\mathbf{x} \in \mathbb{C}^n \setminus \{\mathbf{0}\}$ such that

$$A\mathbf{x} = \lambda\mathbf{x}.$$

Finding such eigenvectors and eigenvalues is again motivated by applications. For example, in structural mechanics a vibrating system is represented by finite elements and the eigenvectors of the corresponding discretisation matrix reflect the shape modes and the roots of the eigenvalues reflect the frequencies with which the system is vibrating. But eigenvalues will also be helpful in better understanding some of the questions above. For example, they have a crucial influence on the stability of an algorithm.

In this book, we are mainly interested in systems of real numbers, simply because they arise naturally in most applications. However, as the problem of finding eigenvalues indicates, it is sometimes necessary to consider complex valued systems, as well. Fortunately, most of our algorithms and findings will carry over from the real to the complex case in a straightforward way.

We will look at *direct* and *iterative* methods to solve linear systems. Direct methods compute the solution in a finite number of steps, iterative methods construct a sequence of approximations to the solution.

We will look at how efficient and stable these methods are. The former means that we are interested in how much time and computer memory they require. Particular emphasis will be placed on the number of *floating point operations*

required with respect to the dimension of the linear system. The latter means for example investigating whether these methods converge at all, under what conditions they converge and how they respond to small changes in the input data.

1.1 Examples Leading to Linear Systems

As mentioned above, linear systems arise naturally during the discretisation process of mathematical models of real-world problems. Here, we want to collect three examples leading to linear systems. These examples are our model problems, which we will refer to frequently in the rest of this book. They comprise the problem of interpolating an unknown function only known at discrete data sites, the solution of a one-dimensional boundary value problem with finite differences and the solution of a (one-dimensional) integral equation with a Galerkin method. We have chosen these three examples because they are simple and easily explained, yet they are significant enough and each of them represents a specific class of problems. In particular, the second problem leads to a linear system with a matrix A which has a very simple structure. This matrix will serve us as a role model for testing and investigating most of our methods since it is simple to analyse yet complicated enough to demonstrate the advantages and drawbacks of the method under consideration.

1.1.1 Interpolation

Suppose we are given data sites $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ and observations $f_1, \dots, f_n \in \mathbb{R}$. Suppose further that the observations follow an unknown generation process, i.e. there is a function f such that $f(\mathbf{x}_i) = f_i$, $1 \le i \le n$.
One possibility to approximately reconstruct the unknown function f is to choose *basis functions* $\phi_1, \dots, \phi_n \in C(\mathbb{R}^d)$ and to approximate f by a function s of the form

$$s(\mathbf{x}) = \sum_{j=1}^{n} \alpha_j \phi_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^d,$$

where the coefficients are determined by the interpolation conditions

$$f_i = s(\mathbf{x}_i) = \sum_{j=1}^{n} \alpha_j \phi_j(\mathbf{x}_i), \qquad 1 \le i \le n.$$

This leads to a linear system, which can be written in matrix form as

$$\begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \dots & \phi_n(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \dots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \dots & \vdots \\ \phi_1(\mathbf{x}_n) & \phi_2(\mathbf{x}_n) & \dots & \phi_n(\mathbf{x}_n) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}. \tag{1.1}$$

From standard Numerical Analysis courses we know this topic usually in the setting that the dimension is $d = 1$, that the points are ordered $a \le x_1 < x_2 < \cdots < x_n \le b$ and that the basis is given as a basis for the space of polynomials of degree at most $n-1$. This basis could be the basis of monomials $\phi_i(x) = x^{i-1}$, $1 \le i \le n$, in which case the matrix in (1.1) becomes the transpose of a so-called *Vandermonde matrix*, i.e. a matrix of the form

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix}.$$

This matrix is a *full matrix*, meaning that each entry is different from zero, so that the determination of the interpolant requires the solution of a linear system with a full matrix.

However, we also know, from basic Numerical Analysis, that we could alternatively choose the so-called *Lagrange functions* as a basis:

$$\phi_j(x) = L_j(x) = \prod_{\substack{i=1 \\ i \ne j}}^{n} \frac{x - x_i}{x_j - x_i}, \qquad 1 \le j \le n.$$

They obviously have the property $L_j(x_j) = 1$ and $L_j(x_i) = 0$ for $j \ne i$. Thus, with this basis, the matrix in (1.1) simply becomes the identity matrix and the interpolant can be derived without solving a linear system at all.

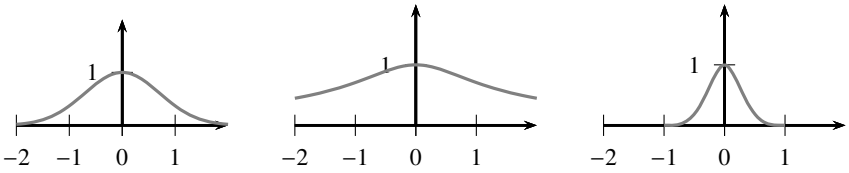

Figure 1.1 Typical radial basis functions: Gaussian, inverse multiquadric and a compactly supported one (from left to right).

In higher dimensions, i.e. $d \ge 2$, polynomial interpolation can become quite problematic and a more elegant way employs a basis of the form $\phi_i = \Phi(\cdot - \mathbf{x}_i)$,

where $\Phi : \mathbb{R}^d \to \mathbb{R}$ is a fixed function. In most applications, this function is chosen to be *radial*, i.e. it is of the form $\Phi(\mathbf{x}) = \phi(\|\mathbf{x}\|_2)$, where $\phi : [0, \infty) \to \mathbb{R}$ is a univariate function and $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_d^2}$ denotes the Euclidean norm. Examples of possible univariate functions are

$$\begin{aligned} \text{Gaussian:} &\qquad \phi(r) = \exp(-r^2), \\ \text{Multiquadric:} &\qquad \phi(r) = (r^2 + 1)^{1/2}, \\ \text{Inverse Multiquadric:} &\qquad \phi(r) = (r^2 + 1)^{-1/2}, \\ \text{Compactly Supported:} &\qquad \phi(r) = (1 - r)_+^4 (4r + 1), \end{aligned}$$

where $(x)_+$ is defined to be x if $x \geq 0$ and to be 0 if $x < 0$. The functions are visualised in Figure 1.1.

In all these cases, except for the multiquadric basis function, it is known that the resulting interpolation matrix is positive definite (with the restriction of $d \leq 3$ for the compactly supported function). Such functions are therefore called *positive definite*. More generally, a good choice of a basis is given by $\phi_j(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_j)$ with a *kernel* $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, which is *positive definite* in the sense that for all possible, pairwise distinct points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, the matrix $(K(\mathbf{x}_i, \mathbf{x}_j))$ is symmetric and positive definite.

In the case of the multiquadric basis function, it is known that the interpolation matrix is invertible and has only real, non-vanishing eigenvalues and that all but one of these eigenvalues are negative.

Note that in the case of the inverse multiquadric and the Gaussian the matrices are dense while in the case of the compactly supported basis function the matrix can have a lot of zeros depending on the distribution of the data sites. Details on this topic can be found in Wendland [133].

1.1.2 Boundary Value Problem

Another application is to compute a stationary solution to the heat equation. In one dimension, we could imagine an infinitely thin rod of length one, which is heated in the interior of $(0, 1)$ with a heat source f and is kept at zero degrees at the boundary points 0, 1. Mathematically, this means that we want to find a function $u : [0, 1] \to \mathbb{R}$ with

$$-u''(x) = f(x), \qquad x \in (0, 1),$$

with boundary conditions $u(0) = u(1) = 0$. If the function f is too complicated or even given only at discrete points then it is not possible to compute the solution u analytically. In this case a numerical scheme has to be used and the

simplest idea is to approximate the derivative by differences:

$$u'(x) \approx \frac{u(x+h)-u(x)}{h}, \quad \text{or} \quad u'(x) \approx \frac{u(x)-u(x-h)}{h}.$$

The first rule could be referred to as a forward rule while the second is a backward rule. Using first a forward and then a backward rule for the second derivative leads to

$$\begin{aligned} u''(x) &\approx \frac{u'(x+h)-u'(x)}{h} \approx \frac{1}{h}\left(\frac{u(x+h)-u(x)}{h} - \frac{u(x)-u(x-h)}{h}\right) \\ &= \frac{u(x+h)-2u(x)+u(x-h)}{h^2}. \end{aligned}$$

For finding a numerical approximation using such finite differences we may divide the domain $[0,1]$ into $n+1$ pieces of equal length $h = 1/(n+1)$ with nodes

$$x_i = ih = \frac{i}{n+1}, \qquad 0 \le i \le n+1,$$

and set $u_i := u(x_i)$. We now define the finite difference approximation u^h to u, as follows: find u^h such that $u_0^h = u_{n+1}^h = 0$ and

$$-\left(\frac{u_{i+1}^h - 2u_i^h + u_{i-1}^h}{h^2}\right) = f_i, \qquad 1 \le i \le n.$$

Alternatively this linear system of n equations may be written in the form

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & 0 \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ 0 & & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1^h \\ u_2^h \\ u_3^h \\ \vdots \\ u_{n-1}^h \\ u_n^h \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}. \tag{1.2}$$

This system of equations is *sparse*, meaning that the number of non-zero entries is much smaller than n^2. This sparsity can be used to store the matrix and to implement matrix–vector and matrix–matrix multiplications efficiently.
To obtain an accurate approximation to u, we may have to choose h very small, thereby increasing the size of the linear system.
For a general boundary value problem in d-dimensions the size of the linear system can grow rapidly. For example, three-dimensional problems grow over eight times larger with each uniform refinement of the domain.

1.1.3 Integral Equations

In the last section we have introduced a way of solving a differential equation. A differential equation can also be recast as an integral equation but integral equations often also come up naturally during the modelling process. Hence, let us consider a typical integral equation as another example.
We now seek a function $u : [0,1] \to \mathbb{R}$ satisfying

$$\int_0^1 \log(|x-y|)u(y)dy = f(x), \qquad x \in [0,1],$$

where $f : [0,1] \to \mathbb{R}$ is given. Note that the integral on the left-hand side contains the *kernel* $K(x,y) := \log(|x-y|)$, which is singular on the diagonal $x = y$.
To solve this integral equation numerically we will use a *Galerkin approximation*. The idea here is to choose an approximate solution u_n from a fixed, finite-dimensional subspace $V = \operatorname{span}\{\phi_1, \dots, \phi_n\}$ and to test the approximate solution via

$$\int_0^1 \int_0^1 \log(|x-y|)u_n(y)dy\,\phi_i(x)dx = \int_0^1 f(x)\phi_i(x)dx, \qquad 1 \le i \le n. \quad (1.3)$$

Since we choose $u_n \in V$ it must have a representation $u_n = \sum_{j=1}^n c_j\phi_j$ with certain coefficients c_j. Inserting this representation into (1.3) and changing the order of summation and integration yields

$$\sum_{j=1}^n c_j \int_0^1 \int_0^1 \log(|x-y|)\phi_j(y)\phi_i(x)dy\,dx = \int_0^1 f(x)\phi_i(x)dx, \qquad 1 \le i \le n,$$

which we easily identify as a linear system $A\mathbf{c} = \mathbf{f}$ with the matrix A having entries

$$a_{ij} = \int_0^1 \int_0^1 \log(|x-y|)\phi_j(y)\phi_i(x)dy\,dx, \qquad 1 \le i, j \le n.$$

A typical choice for the space V is the space of piece-wise constant functions. To be more precise, we can choose

$$\phi_i(x) = \begin{cases} 1 & \text{if } \frac{i-1}{n} \le x < \frac{i}{n}, \\ 0 & \text{else,} \end{cases}$$

but other basis functions and approximation spaces are possible. But we note that particularly in this case the matrix A is once again a full matrix as its entries are given by

$$a_{ij} = \int_{(i-1)/n}^{i/n} \int_{(j-1)/n}^{j/n} \log(|x-y|)dy\,dx.$$

An obvious generalisation of this problem to arbitrary domains $\Omega \subseteq \mathbb{R}^d$ leads to matrix entries of the form

$$a_{ij} = \int_\Omega \int_\Omega K(\mathbf{x}, \mathbf{y}) \phi_i(\mathbf{x}) \phi_j(\mathbf{y}) d\mathbf{y}\, d\mathbf{x}$$

with a given kernel $K : \Omega \times \Omega \to \mathbb{R}$.

1.2 Notation

Now, it is time to set up the notation which we will use throughout this book. However, we will specify only the most basic notation and definitions here and introduce further concepts whenever required.

1.2.1 Mathematics

We will denote the real, complex, natural and integer numbers as usual with $\mathbb{R}$, $\mathbb{C}$, $\mathbb{N}$ and $\mathbb{Z}$, respectively. The natural numbers will not include zero. We will use the notation $\mathbf{x} \in \mathbb{R}^n$ to denote vectors. The components of $\mathbf{x}$ will be denoted by $x_j \in \mathbb{R}$, i.e. $\mathbf{x} = (x_1, \dots, x_n)^T$. Thus vectors will always be column vectors. We will denote the unit standard basis of $\mathbb{R}^n$ by $\mathbf{e}_1, \dots, \mathbf{e}_n$, where the ith unit vector $\mathbf{e}_i$ has only zero entries except for a one at position i. In general, we will suppress the dimension n when it comes to this basis and we might use the same notation to denote the ith unit vector for $\mathbb{R}^n$ and, say, $\mathbb{R}^m$. It should be clear from the context which one is meant. On $\mathbb{R}^n$ we will denote the inner product between two vectors $\mathbf{x}$ and $\mathbf{y}$ by either $\mathbf{x}^T\mathbf{y}$ or $\langle \mathbf{x}, \mathbf{y} \rangle_2$, i.e.

$$\mathbf{x}^T\mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle_2 = \sum_{j=1}^{n} x_j y_j.$$

For a matrix A with m rows, n columns and real entries we will write $A \in \mathbb{R}^{m \times n}$ and $A = (a_{ij})$, where the index i refers to the rows and the index j refers to the columns:

$$A := (a_{ij}) := \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$

For a non-square matrix $A \in \mathbb{R}^{m \times n}$, we can write $a_{ij} = \mathbf{e}_i^T A \mathbf{e}_j$, where the first unit vector is from $\mathbb{R}^m$ while the second unit vector is from $\mathbb{R}^n$.

We will use the *Kronecker δ-symbol* δ_{ij}, which is defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

For the *identity matrix* in $\mathbb{R}^n$ we will use the symbol $I \in \mathbb{R}^{n\times n}$. We obviously have $I = (\delta_{ij})_{1\leq i,j\leq n}$. Again, as in the case of the unit vectors, we will usually refrain from explicitly indicating the dimension n of the underlying space $\mathbb{R}^n$. We will also denote the columns of a matrix $A \in \mathbb{R}^{m\times n}$ by $\mathbf{a}_j := A\mathbf{e}_j \in \mathbb{R}^m$, $1 \leq j \leq n$. Hence, we have

$$A = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n).$$

For a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $\mathbf{x} \in \mathbb{R}^n$, we can write $\mathbf{x} = \sum_j x_j \mathbf{e}_j$ and hence

$$A\mathbf{x} = \sum_{j=1}^{n} x_j \mathbf{a}_j.$$

We will encounter specific forms of matrices and want to use the following, well-known names.

Definition 1.1 A matrix $A = (a_{ij}) \in \mathbb{R}^{m\times n}$ is

- a *square matrix*, if $m = n$,
- a *diagonal matrix*, if $a_{ij} = 0$ for $i \neq j$,
- an *upper triangular matrix*, if $a_{ij} = 0$ for $i > j$,
- a *lower triangular matrix*, if $a_{ij} = 0$ for $i < j$,
- a *band-matrix*, if there are $k, \ell \in \mathbb{N}_0$ such that $a_{ij} = 0$ if $j < i - k$ or $j > i + \ell$,
- *sparse*, if more than half of the entries are zero,
- *dense* or *full*, if it is not sparse.

In the case of a diagonal matrix A with diagonal entries $a_{ii} = \lambda_i$, we will also use the notation

$$A = \operatorname{diag}(\lambda_1, \ldots, \lambda_n).$$

In particular, we have for the identity matrix $I = \operatorname{diag}(1, \ldots, 1) \in \mathbb{R}^{n\times n}$.

Most of these names are self-explanatory. In the case of a band matrix, we have all entries zero outside a diagonally bordered band. Only those entries a_{ij} with indices $i - k \leq j \leq i + \ell$ may be different from zero. This means we have at most k sub-diagonals and ℓ super-diagonals with non-zero entries. The most prominent example is given by $k = \ell = 1$, which has one super-diagonal and one sub-diagonal of non-zero entries and is hence called a *tridiagonal* matrix.