Introduction

Goals of computer simulation

In a wide variety of scientific disciplines, ranging from physics to biology and economics, the phenomena under consideration are well-described by mathematical equations. More often than not, these equations are too difficult to be solved analytically, and so one has to recur to *computer calculations* in order to obtain approximate solutions. Computer simulation enables to gain understanding of the phenomena examined, to explain observations and to make predictions. Practical applications in which it plays a crucial role include weather forecasting, drug discovery through molecular modeling, flight simulation, and structural engineering, to mention just a few.

Numerical simulation may also be employed in order to calibrate mathematical models of physical phenomena, particularly when observation through experiment is impractical or too costly. For example, it is frequently the case that the parameters in mathematical models for turbulence are estimated not from real data, but from synthetic data generated by computer simulation of the fundamental equations of fluid mechanics. Relying on "computer experiments" is attractive in this context because these enable to perform accurate measurements without disturbing the system being observed. Numerical simulation is also very useful to understand and build simplified models for physical phenomena at very small scales, if direct observation is beyond the capabilities of experimental physics.

Sources of error in computational science

It is important for practitioners of computer simulation to be aware of the different sources of error likely to affect numerical results obtained in applications, which may be classified as follows:

- **Modeling error.** There may be a discrepancy between the mathematical model and the underlying physical phenomenon.
- **Data error.** The data of the problem, such as the initial conditions or the parameters entering the equations, are usually known only approximately.
- **Discretization error.** The *discretization* of mathematical equations, i.e. turning them into finite-dimensional problems amenable to computer simulation, adds another source of error.

- **Discrete solver error.** The method employed to solve the discretized equations, especially if it is of iterative nature, may also introduce an error.
- **Round-off errors.** Finally, the limited accuracy of computer arithmetics causes additional errors.

Of these, only the last three are in the domain of numerical analysis, and in this course we focus mainly on the *solver* and *round-off* errors. The order of magnitude of the overall error is dictated by the largest among the above sources errors.

Aims of this course

The aim of this course is to present the standard numerical methods for performing the tasks most commonly encountered in applications: the solution of linear and nonlinear systems of equations, the solution of eigenvalue problems, interpolation and approximation of functions, and numerical integration. For a given task, there are usually several numerical methods to choose from, and these often include parameters which must be fixed appropriately in order to guarantee a good efficiency. In order to guide these choices, we study carefully the *convergence* and *stability* of the different methods we present. Six topics will be covered in these lecture notes.

- **Floating point arithmetic.** In Chapter 1, we discuss how real numbers are represented, manipulated and stored on a computer. There is an uncountable infinity of real numbers, but only a finite subset of these can be represented exactly on a machine. This subset is specified in the *IEEE 754* standard, which is widely accepted today and employed in most programming languages, including `Julia`.
- **Solution of linear systems.** In Chapter 2, we study the standard numerical methods for solving linear systems. Linear systems are ubiquitous in science, often arising from the discretization of linear elliptic partial differential equations, which themselves govern a large number of physical phenomena including heat propagation, electromagnetism, gravitation and the deformation of solids.
- **Solution of nonlinear equations.** In Chapter 3, we present widely used methods for solving nonlinear equations. Like linear equations, nonlinear equations are omnipresent in science, a prime example being the Navier–Stokes equation describing the motion of fluid flows. They are usually much more difficult to solve and require dedicated techniques.
- **Solution of eigenvalue problems.** In Chapter 4, we present and study the standard iterative methods for calculating the eigenfunctions and eigenvalues of a matrix. Eigenvalue problems have a large number of applications, for instance in quantum physics and vibration analysis. They are also at the root of the PageRank algorithm for ranking web pages, which played a key role in the early success of Google search.
- **Interpolation and extrapolation of functions.** In Chapter 5, we focus on the topics of interpolation and approximation. *Interpolation* is concerned with the construction of

a function within a given set, for example that of polynomials, that takes given values when evaluated at a discrete set of points. The aim of *approximation*, on the other hand, is usually to determine, within a class of simple functions, which one is closest to a given function. Depending on the metric employed to measure closeness, this may or may not be a well-defined problem.

- **Numerical integration.** In Chapter 6, we study numerical methods for computing definite integrals. This chapter is strongly related to the previous one, as numerical approximations of the integral of a function are often obtained by first approximating the function, say by a polynomial, and then integrating this approximation exactly.

Why Julia?

Throughout the course, the `Julia` programming language is employed to exemplify some of the methods and key concepts. In the author's opinion, the `Julia` language has several advantages compared to other popular languages in the context of scientific computing, such as `Matlab` or `Python`.

- Its main advantage over `Matlab` is that it is free and open source, with the byproduct that it benefits from contributions from a large number of contributors around the world. Additionally, `Julia` is a fully-fledged programming language that can be used for applications unrelated to mathematics.

- Its main advantages over `Python` are significantly better performance and a more concise syntax for mathematical operations, especially those involving vectors and matrices. It should be recognized, however, that although use of `Julia` is rapidly increasing, `Python` still enjoys a more mature ecosystem and is much more widely used.