Introduction

This course is the second in a two-course sequence on numerical analysis, the study of numerical algorithms for solving mathematical problems that arise in science and engineering. These numerical algorithms differ from the analytical methods that are presented in other mathematics courses in that they rely exclusively on the four basic arithmetic operations, addition, subtraction, multiplication and division, so that they can be implemented on a computer.

Initial Value Problems

First, we study various algorithms for solving an initial value problem, which consists of an ordinary differential equation

\[
\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b,
\]

and an initial condition

\[ y(t_0) = \alpha. \]

Unlike analytical methods for solving such problems, that are used to find the exact solution in the form of a function \( y(t) \), numerical methods typically compute values \( y_1, y_2, y_3, \ldots \) that approximate \( y(t) \) at discrete time values \( t_1, t_2, t_3, \ldots \). At each time \( t_{n+1} \), for \( n > 0 \), value of the solution is approximated using its values at previous times.

We will learn about two general classes of methods: one-step methods, which are derived using Taylor series and compute \( y_{n+1} \) only from \( y_n \), and multistep methods, which are based on polynomial interpolation and compute \( y_{n+1} \) from \( y_n, y_{n-1}, \ldots, y_{n-s+1} \), where \( s \) is the number of steps in the method. Either type of method can be explicit, in which \( y_{n+1} \) can be described in terms of an explicit formula, or implicit, in which \( y_{n+1} \) is described implicitly using an equation, normally nonlinear, that must be solved during each time step.

The difference between consecutive times \( t_n \) and \( t_{n+1} \), called the time step, need not be uniform; we will learn about how it can be varied to achieve a desired level of accuracy as efficiently as possible. We will also learn about how the methods used for the first-order initial-value problem described above can be generalized to solve higher-order equations, as well as systems of equations.

One key issue with time-stepping methods is stability. If the time step is not chosen to be sufficiently small, the computed solution can grow without bound, even if the exact solution is bounded. Generally, the need for stability imposes a more severe restriction on the size of the time step for explicit methods, which is why implicit methods are commonly used, even though they
tend to require more computational effort per time step. Certain systems of differential equations can require an extraordinarily small time step to be solved by explicit methods; such systems are said to be stiff.

**Systems of Linear Equations**

Next, we will learn about how to solve a system of linear equations

\[
\begin{align*}
  a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n &= b_1 \\
  a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n &= b_2 \\
  \vdots \\
  a_{n1} x_1 + a_{n2} x_2 + \cdots + a_{nn} x_n &= b_n,
\end{align*}
\]

which can be more conveniently written in matrix-vector form

\[Ax = b,\]

where \(A\) is an \(n \times n\) matrix, because the system has \(n\) equations (corresponding to rows of \(A\)) and \(n\) unknowns (corresponding to columns).

Last semester, we discussed similar systems of equations, for which the number of equations, \(m\), was greater than the number of unknowns, \(n\). This was the *least-squares* problem, which was reduced to a system with \(n\) equations and unknowns,

\[A^T A x = A^T b,\]

called the *normal equations*; however, we did not discuss how the normal equations were actually solved.

To solve a general system with \(n\) equations and unknowns, we can use *Gaussian elimination* to reduce the system to *upper-triangular form*, which is easy to solve. In some cases, this process requires *pivoting*, which entails interchanging of rows or columns of the matrix \(A\). Gaussian elimination with pivoting can be used not only to solve a system of equations, but also to compute the inverse of a matrix, even though this is not normally practical. It can also be used to efficiently compute the determinant of a matrix.

Gaussian elimination with pivoting can be viewed as a process of factorizing the matrix \(A\). Specifically, it achieves the decomposition

\[PA = LU,\]

where \(P\) is a *permutation matrix* that describes any row interchanges, \(L\) is a *lower-triangular* matrix, and \(U\) is an upper-triangular matrix. This decomposition, called the *LU decomposition*, is particularly useful for solving \(Ax = b\) when the right-hand side vector \(b\) varies. We will see that
for certain special types of matrices, such as those that arise in the normal equations, variations of the general approach to solving \( Ax = b \) can lead to improved efficiency.

Gaussian elimination and related methods are called direct methods for solving \( Ax = b \), because they compute the exact solution (up to roundoff error, which can be significant in some cases) in a fixed number of operations that depends on \( n \). However, such methods are often not practical, especially when \( A \) is very large, or when it is sparse, meaning that most of its entries are equal to zero. Therefore, we also consider iterative methods. Two general classes of iterative methods are:

- **stationary iterative methods**, which can be viewed as fixed-point iterations, and rely primarily on splittings of \( A \) to obtain a system of equations that can be solved rapidly in each iteration, and

- **non-stationary methods**, which tend to rely on matrix-vector multiplication in each iteration and a judicious choice of search direction and linesearch to compute each iterate from the previous one.

### Systems of Nonlinear Equations

Last semester, we learned how to solve nonlinear equations of the form \( f(x) = 0 \) using iterative methods such as Newton’s method. This semester, we will learn how to generalize such methods to solve systems of nonlinear equations of the form \( f(x) = 0 \), where \( f: \mathbb{R}^n \to \mathbb{R}^n \). In particular, for Newton’s method, computing \( x_{n+1} - x_n = -f(x_n)/f'(x_n) \) in the single-variable case is generalized to solving the system of equations \( J_f(x_n)s_n = -f(x_n) \), where \( J_f(x_n) \) is the Jacobian of \( f \) evaluated at \( x_n \), and \( s_n = x_{n+1} - x_n \) is the step from each iterate to the next.

### Boundary Value Problems

We conclude with a discussion of solution methods for the two-point boundary value problem

\[
y'' = f(x, y, y'), \quad a \leq x \leq b,
\]

with boundary conditions

\[
y(a) = \alpha, \quad y(b) = \beta.
\]

One approach, called the shooting method, transforms this boundary-value problem into an initial-value problem so that methods for such problems can then be used. However, it is necessary to find the correct initial values so that the boundary condition at \( x = b \) is satisfied. An alternative approach is to discretize \( y'' \) and \( y' \) using finite differences, the approximation schemes covered last semester, to obtain a system of equations to solve for an approximation of \( y(x) \); this system can be linear or nonlinear. We conclude with the Rayleigh-Ritz method, which treats the boundary value problem as a continuous least-squares problem.