Chapter 5

Introduction to Boundary Value Problems

When we studied IVPs we saw that we were given the initial value of a function and a differential equation which governed its behavior for subsequent times. Now we consider a different type of problem which we call a boundary value problem (BVP). In this case we want to find a function defined over a domain where we are given its value or the value of its derivative on the entire boundary of the domain and a differential equation to govern its behavior in the interior of the domain; see Figure 5.1.

In this chapter we begin by discussing various types of boundary conditions that can be imposed and then look at our prototype BVPs. A BVP which only has one independent variable is an ODE but when we consider BVPs in higher dimensions we need to use PDEs. We briefly review partial differentiation, classification of PDEs and examples of commonly encountered PDEs. We discuss the implications of discretizing a BVP as compared to an IVP and give examples of different types of grids. We will see that the solution of a discrete BVP requires the solution of a linear system of algebraic equations $A\mathbf{x} = \mathbf{b}$ so we end this chapter with a review of direct and iterative methods for solving $A\mathbf{x} = \mathbf{b}$ for a square invertible matrix A.

5.1 Types of boundary conditions

In the sequel we will use Ω to denote the domain for a BVP and Γ to denote its boundary. In one dimension, the only choice for a domain is an interval. However, in two dimensions there are a myriad of choices; common choices include a rectangle, a circle, a portion of a circle such as a wedge or annulus, or a polygon. Likewise in three dimensions there are many choices for domains. We will always assume that our domain is bounded; e.g., we will not allow the right half plane in $\mathbb{R}^2$ defined by $\Omega = \{(x, y) \mid x > 0\}$ to be a domain. In addition, we will assume that our

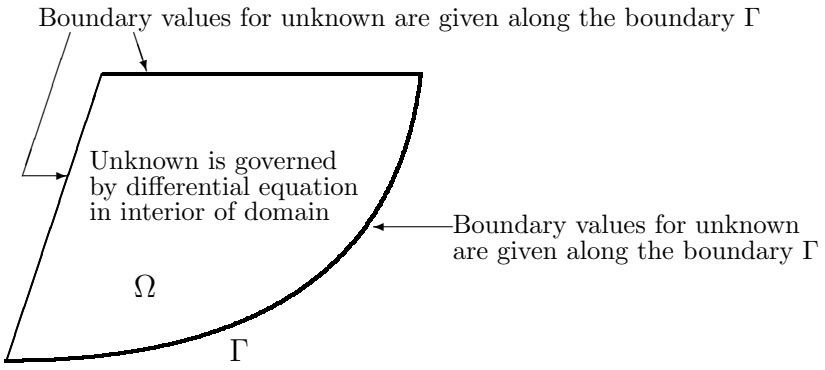


Figure 5.1: Sample domain of a BVP with a differential equation governing the unknown in the interior of the domain and boundary values specified on all boundaries.

boundary is sufficiently smooth.

The conditions that we impose on the boundary of the domain are called *boundary conditions*. The most common boundary condition is to specify the value of the function on the boundary; this type of constraint is called a **Dirichlet**[1] **boundary condition**. For example, if we specify Dirichlet boundary conditions for the interval domain $[a, b]$, then we must give the unknown at the endpoints a and b; this problem is then called a Dirichlet BVP. In two dimensions we have to specify the boundary values along the entire boundary curve and in three dimensions on the entire boundary surface.

A second type of boundary condition is to specify the derivative of the unknown function on the boundary; this type of constraint is called a **Neumann**[2] **boundary condition**. For example, if we specify $u'(a) = \alpha$ then we are imposing a Neumann boundary condition at the right end of the interval domain $[a, b]$. If we specify only Neumann boundary conditions, then the problem is a purely Neumann BVP.

A third type of boundary condition is to specify a weighted combination of the function value and its derivative at the boundary; this is called a **Robin**[3] **boundary condition** or **mixed boundary condition**. For example, for the

[1] Named after the German mathematician Gustav Lejeune Dirichlet (1805-1859)
[2] Named after the German mathematician Carl Neumann (1832-1925)
[3] Named after the French mathematician Victor Gustave Robin (1855-1897)

unknown $u(x)$ on $[a, b]$ we might specify the Robin condition $u(a) - 2u'(a) = 0$. We can have a **mixed BVP** by specifying one type of boundary condition on a portion of the boundary and another type on the remainder of the boundary. For example, on the interval $[a, b]$ we might specify a Dirichlet condition for the unknown $u(x)$ at $x = a$ by setting $u(a) = \alpha$ and a Neumann boundary condition at $x = b$ by setting $u'(b) = \beta$.

We say a boundary condition is **homogeneous** if its value is set to zero; otherwise it is called **inhomogeneous**. For example, consider a purely Dirichlet BVP on $[0, 1]$ where we specify $u(0) = 5$ and $u(1) = 0$. Then the boundary condition on the left at $x = 0$ is inhomogeneous and the boundary condition on the right at $x = 1$ is homogeneous. Thus if someone tells you they have a purely Dirichlet (or Neumann) BVP on $[a, b]$ with homogeneous boundary data, then you completely know the boundary conditions without explicitly writing them.

5.2 Prototype BVPs in one dimension

We begin with the simplest scenario which is a linear second order BVP in one spatial dimension so the domain is an interval $[a, b]$. This is often called a two-point BVP because we must specify boundary conditions at the two points $x = a$ and $x = b$. Because our unknown must satisfy two boundary conditions we know that the governing differential equation can't be first order as in the case of the IVP where only one auxiliary condition was imposed; rather it must be second order. For example, a Dirichlet BVP for $u(x)$ on the domain $\Omega = [0, 3]$ is

$$\begin{aligned} -u''(x) &= 2x \qquad 0 < x < 3 \\ u(0) &= 0 \qquad u(3) = 9\,. \end{aligned}$$

From inspection, we know that $u(x)$ must be a cubic polynomial because its second derivative is a linear polynomial. To find the analytic solution we simply integrate the equation twice and then apply the boundary conditions to determine the two arbitrary constants to get $u(x) = -x^3/3 + 6x$. If we have the purely Neumann problem

$$\begin{aligned} -u''(x) &= 2x \qquad 0 < x < 3 \\ u'(0) &= 0 \qquad u'(3) = -9 \end{aligned}$$

we have a different situation. The general solution to the differential equation is $u(x) = -x^3/3 + C_1 x + C_2$ so $u'(x) = -x^2 + C_1$. Satisfying the boundary condition $u'(0) = 0$ gives $C_1 = 0$ and the other condition is also satisfied with this choice of C_1. Then the solution to the BVP is $-x^3/3 + C_2$ i.e., it is not unique but rather only unique up to a constant! We can see this from the BVP because neither the differential equation nor the boundary conditions impose any condition on $u(x)$ itself. If the boundary condition at the right was $u'(3) = 1$ then this could not have been satisfied and there would have been no solution to the BVP. Consequently care must be taken in using a purely Neumann problem.

These BVPs are specific examples of a more general class of linear two-point boundary value problems governed by the differential equation

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x) \quad a < x < b\,, \tag{5.1}$$

where $p(x)$, $q(x)$ and $f(x)$ are given functions. Clearly if $p = 1$, $q = 0$, $a = 0$, $b = 3$ and $f(x) = 2x$ then we have our specific example. For a general second order BVP which may be linear or nonlinear we write the differential equation as

$$y''(x) = f(x, y, y') \quad a < x < b\,. \tag{5.2}$$

Before we attempt to approximate the solution to a given BVP we want to know that the continuous problem has a unique solution. For (5.1) it is well known that under specific conditions on $p(x)$, $q(x)$ and $f(x)$ there is a unique solution to the Dirichlet BVP. In particular we assume that

$$0 < p_{\min} \le p \le p_{\max} \quad \text{and} \quad q_{\min} = 0 \le q(x) \le q_{\max}\,.$$

The coefficient $p(x)$ is not allowed to be zero otherwise we would not have a differential equation. For existence and uniqueness we require that f and q be continuous functions on the domain $[a, b]$ and that p has a continuous first derivative there in addition to the given bounds. For the general nonlinear equation (5.2) the theory is more complicated; in the sequel we will concentrate on the linear two-point BVP.

We can also consider a higher order equation in one dimension. For example consider the fourth order linear equation

$$\begin{aligned} &\frac{d^2}{dx^2}\left(r(x)\frac{d^2u}{dx^2}\right) - \frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x) \quad a < x < b \\ &u(0) = u(1) = 0 \qquad u''(0) = u''(1) = 0\,. \end{aligned} \tag{5.3}$$

This equation can either be solved as a fourth order equation or written as two second order equations.

5.3 Prototype BVPs in higher dimensions

In the last section we looked at a two-point BVP which is just an ODE. When we consider BVPs in higher dimensions the unknown will be a function of more than one variable so the differential equation will be a PDE. In this section we first review differentiation in higher dimensions and then look at the classification of PDEs and some commonly encountered examples. Then we proceed to consider our prototype equation which is the Poisson equation; in one dimension it is just $-u''(x) = f(x)$.

5.3.1 Partial differential equations

Partial differential equations (PDEs) are differential equations where the unknown is a function of more than one independent variable; this is in contrast to ODEs

where the unknown is a function of only one independent variable. For example, in two spatial dimensions we could have a function $u = u(x, y)$ or in three dimensions $u = u(x, y, z)$; in the case of time dependent problems we could have $u = u(x, t)$ in one spatial dimension, $u = u(x, y, t)$ in two dimensions and $u = u(x, y, z, t)$ in three dimensions. Of course equations can depend upon other variables than time and space.

Recall from calculus that if a function depends on two or more independent variables then to differentiate it we must take partial derivatives. For example, if $u = u(x, y)$ then we can determine its two first partial derivatives denoted u_x, u_y or equivalently $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$. The definition of u_x where $u = u(x, y)$ is

$$u_x \equiv u_x = \lim_{h \to 0} \frac{u(x + h, y) - u(x, y)}{h} .$$

Thus a partial derivative gives the change in the function in the direction of the coordinate axis so when we take a partial derivative with respect to x it gives the change in the horizontal direction; thus y is held constant. Therefore if $u(x, y) = y^3 e^{2x}$, we have $u_x = 2y^3 e^{2x}$ and $u_y = 3y^2 e^{2x}$. Higher partial derivatives are determined in an analogous manner. We will assume continuity of the derivative so that the order of differentiation does not matter, e.g., $u_{xy} = u_{yx}$. For example, if $u(x, y) = y^3 e^{2x}$ then $u_{xx} = 4y^3 e^{2x}$, $u_{yy} = 6y e^{2x}$, $u_{xy} = 6y^2 e^{2x} = u_{yx}$.

Differential operators

There are three differential operators that we will use extensively. Recall that in calculus we learned that we take the gradient of a scalar and get a vector field. So the magnitude of the gradient of u is the magnitude of the change in u, analogous to the magnitude of the slope in one dimension. The standard notation used is the Greek symbol nabla, ∇ or simply "grad". Remember that it is an operator and so just writing ∇ does not make sense but rather we must write, e.g., ∇u. The ∇ operator is the vector of partial derivatives so in three dimensions it is $(\partial/\partial x, \partial/\partial y, \partial/\partial z)^T$.

One use we will have for the gradient is when we want to impose a flux boundary condition. Clearly there are times when we want to know the rate of change in $u(x, y)$ in a direction other than parallel to the coordinate axis; remember that the standard partial derivative gives the change in the coordinate axis. When this is the case we define a unit vector in the direction of the desired change and we use the gradient of u. If $\hat{\mathbf{n}}$ is the unit vector giving the direction then the derivative and the notation we use is

$$\frac{\partial u}{\partial \hat{\mathbf{n}}} \equiv \nabla u \cdot \hat{\mathbf{n}} . \tag{5.4}$$

Note that if $\hat{\mathbf{n}} = (1, 0)^T$, i.e., in the direction of the x-axis, then we just get u_x which is the standard partial derivative in the direction of the x-axis; similarly if $\hat{\mathbf{n}} = (0, 1)^T$ then we get u_y. We will have a particular use for this notation when we specify a boundary condition such as the flux on the boundary. In one dimension the flux is just $u'(x)$ but in higher dimensions it is the change in u along the normal

to the boundary. So in higher dimensions we will specify $\partial u/\partial \hat{\mathbf{n}}$ as a Neumann boundary condition.

The next differential operator that we need is the divergence. Recall that the divergence is a *vector* operator. It is also represented by ∇ or simply "div" but typically we use a dot after it to indicate that it operates on a vector; other sources will use a bold face ∇. So if $\mathbf{w} = (w_1, w_2, w_3)$ then the divergence of $\mathbf{w}$, denoted $\nabla \cdot \mathbf{w}$, is the scalar $\partial w_1/\partial x + \partial w_2/\partial y + \partial w_3/\partial z$.

The last differential operator that we need is called the Laplacian.[4] It combines the gradient and the divergence to get a second order operator but of course the order is critical. If $u(x, y, z)$ is a scalar function then we can take its gradient to get a vector function, then the divergence may be applied to this vector function to get a scalar function. Because this operator is used so extensively in PDEs it is given a special notation, Δ which is the Greek symbol for capital delta. In particular we have $\Delta = \nabla \cdot \nabla$ so if $u = u(x, y, z)$ then

$$\Delta u \equiv \nabla \cdot \nabla u = u_{xx} + u_{yy} + u_{zz} \tag{5.5}$$

because

$$\nabla \cdot \nabla u = \nabla \cdot \left[(u_x, u_y, , u_z)^T\right] = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T \cdot (u_x, u_y, u_z)^T = u_{xx} + u_{yy} + u_{zz}\,.$$

In the sequel we will typically use the notation Δu instead of $\nabla \cdot \nabla u$. In some contexts $\nabla^2 u$ is used for Δu but we will not use this notation.

Classification

Like ODEs, PDEs can be broadly classified by

- their order
- linearity.

The order is determined by the highest derivative occurring in the equation. For linearity the equation must be linear in the unknown and its derivatives; nonlinearity in the independent variables such as x, y do not affect the linearity of the equation. We may have a single PDE or a coupled system of PDEs. For example, for $u = u(x, y)$, $v = v(x, y)$ we have the single PDE

$$-\Delta u = f(x, y)$$

or a system of PDEs for (u, v)

$$-\Delta u + v(x, y) = f(x, y)$$

$$-\Delta v + 3u(x, y) = g(x, y)\,.$$

[4]Named after the French mathematician Pierre-Simon de Laplace (1749-1827).

A second order linear PDE in two independent variables (ξ, η) has the general form

$$au_{\xi\xi} + bu_{\xi\eta} + cu_{\eta\eta} + du_{\xi} + eu_{\eta} + gu = f(\xi, \eta) \tag{5.6}$$

where a, b, c, d, e, g are given coefficients (they can be constants or functions of (ξ, η)) and $f(\xi, \eta)$ is a given right hand side or source term. The equation is called *homogeneous* if $f \equiv 0$; otherwise *inhomogeneous*.

The second order linear PDE (5.6) is classified as **elliptic**, **parabolic**, or **hyperbolic** in a domain based upon the sign of the discriminant $b^2 - 4ac$. We have the equation classified as

elliptic	if $b^2 - 4ac < 0$	for all points in the domain
parabolic	if $b^2 - 4ac = 0$	for all points in the domain
hyperbolic	if $b^2 - 4ac > 0$	for all points in the domain

The names elliptic, parabolic, and hyperbolic come from the classification of conic sections $ax^2 + bxy + cy^2 + dx + ey + g$ which is classified as elliptic if $b^2 - 4ac < 0$, etc. For example, for the unit circle $x^2 + y^2 = 1$ we have $b^2 - 4ac = -4 < 0$ so it is elliptic.

Different types of phenomena are modeled by each type of equation. Throughout this course we will consider prototype equations for each type of equation. It is important to know if a given PDE is elliptic, parabolic or hyperbolic because this tells us a lot about how to solve it.

Example 5.1. Classify each equation by order and linearity. If the equation is a linear second order equation in two independent variables classify it as elliptic, parabolic or hyperbolic.

1. Let $u = u(x, y)$, then

   $$-\Delta u = -(u_{xx} + u_{yy}) = f(x, y)\,.$$

   This equation is called the Poisson equation for $f \neq 0$; if $f \equiv 0$, it is called the Laplace equation. This is a second order linear PDE and is elliptic because $b = 0$, $a = c = -1$ so $b^2 - 4ac < 0$.

2. Let $u = u(x, t)$, then

   $$u_t - u_{xx} = f(x, t)\,.$$

   This is called the heat or diffusion equation in one space variable. It is a second order linear PDE and is parabolic because $a = -1$, $b = c = 0$ so $b^2 - 4ac = 0$.

3. Let $u = u(x, y, t)$, then

   $$u_t - \Delta u = f(x, y, t)$$

   is called heat or diffusion equation in two space variables. It is parabolic although it doesn't fit into the framework above because it is a function of three independent variables.

4. Let $u = u(x,t)$, then
$$u_{tt} - u_{xx} = f(x,t)$$
is called the wave equation. It is a second order linear PDE and is hyperbolic because $a = 1$, $b = 0$, $c = -1$ so $b^2 - 4ac > 0$.

5. Let $u = u(x,y)$, then
$$\Delta(\Delta u) = f(x,y) \quad \text{where } \Delta\Delta u = u_{xxxx} + 2u_{xxyy} + u_{yyyy}$$
is called the biharmonic equation. It is a fourth order linear equation.

6. Let $u = u(x,t)$, then
$$u_t + uu_x - u_{xx} = f(x,t)$$
is called the Burger equation in one space variable. It is a second order nonlinear equation due to the uu_x term.

7. Let $u = u(x,y)$, then
$$u_{xx} = xu_{yy}$$
is called the Tricomi equation. It is a second order linear equation and it changes type depending on the value of x. Here $a = 1$, $b = 0$ and $c = -x$ so $b^2 - 4ac = 4x$. If $x = 0$ then it is parabolic (we just have $u_{xx} = 0$), it is elliptic in the left half plane $x < 0$ and hyperbolic for the right half plane $x > 0$.

8. Let $\mathbf{u} = \mathbf{u}(x,y,t)$ which is a vector with components (u,v); then
$$\mathbf{u}_t - \Delta\mathbf{u} + \mathbf{u} : \nabla\mathbf{u} + \nabla p = \mathbf{f}(x,y)$$
$$\nabla \cdot \mathbf{u} = 0$$
is a second order system. It is called the incompressible Navier-Stokes equations and is nonlinear.

5.3.2 The Poisson equation

The Poisson[5] equation is the prototype equation for BVPs. The differential operator is the laplacian denoted Δu which we defined by (5.5). The Poisson equation in three dimensions is
$$-\Delta u = f(x,y,z) \quad (x,y,z) \in \Omega \tag{5.7}$$
and in two dimensions we have the analogous definition without the dependence on z. When $f = 0$ it is typically called the Laplace equation.

The Poisson equation (5.7) is an elliptic equation. If we specify a Dirichlet BVP then we must specify u on the boundary Γ. If we have a purely Neumann BVP, i.e.,

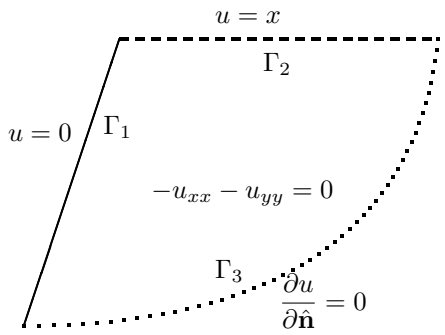


Figure 5.2: A sample mixed BVP for $u(x, y)$ where $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$. Here Dirichlet boundary conditions are specified on $\Gamma_1 \cup \Gamma_2$ and a Neumann boundary condition is specified on Γ_3. The notation $\dfrac{\partial u}{\partial \hat{\mathbf{n}}}$ represents $\nabla u \cdot \hat{\mathbf{n}}$ where $\hat{\mathbf{n}}$ is the unit outer normal to the given boundary.

we specify $\dfrac{\partial u}{\partial \hat{\mathbf{n}}}$ then our solution is not unique, just like in the one dimensional case. An example of a mixed BVP for the Laplace equation is illustrated in Figure 5.2.

If we add homogeneous Dirichlet boundary conditions (i.e., $u = 0$ on Γ) and set the domain to be the unique square, i.e., $\Omega = (0, 1) \times (0, 1)$, then an exact solution can be found:

$$u(x, y) = \sum_{n,m=1}^{\infty} \gamma_{n,m} \sin(n\pi x) \sin(m\pi y)$$

where

$$\gamma_{n,m} = \frac{4}{n^2(m^2 + n^2)} \int_0^1 \int_0^1 f(x, y) \sin(n\pi x) \sin(n\pi y)\, dxdy\,.$$

This is an infinite series where the coefficients are approximated by integrals and convergence may be slow. So even in this case it may be faster to use numerical

[5]Named after the French mathematician, geometer and physicist Siméon-Denis Poisson (1781-1840).

techniques to approximate the PDE. If the domain becomes more complicated, then even these types of solutions are typically not available.

There are various techniques for finding the solution to PDEs such as separation of variables, Greens functions, series expansions, integration, etc. For example, separation of variables was used to find the infinite series solution to the above Dirichlet BVP. However these usually only work for simple domains and constant coefficient problems. Consequently we need to look at methods for approximating their solution.

When we do need an exact solution for verifying that our computer code is working properly we can use a simple technique called *method of manufactured solutions* which was introduced for IVPs in §1.3. For example, suppose we want to solve the Poisson equation on the unit square and we want to satisfy homogeneous Dirichlet boundary conditions, i.e., $u(x,0) = u(x,1) = 0$ and $u(0,y) = u(1,y) = 0$. Then we choose a $u(x,y)$ that satisfies these boundary conditions and then plug it into Δu to get a particular $f(x,y)$ and then we solve that problem with the given right hand side. For homogeneous Dirichlet boundary conditions we could choose $u(x,y) = y(y-1)\sin(\pi x)$ (of course there are lots of other choices for u) and then $f(x,y) = -\Delta u = -(-\pi^2 \sin(\pi x) + 2)$ and we solve the BVP

$$-\Delta u = \pi^2 \sin(\pi x) - 2, \quad (x,y) \in \Omega, \qquad u = 0 \quad \text{on } \Gamma.$$

5.4 Discretization

When we approximated the solution to IVPs our goal was to determine an approximation at a set of discrete times using a step size Δt and if everything was done correctly as $\Delta t \to 0$ our approximate solution converged to our exact solution. In developing algorithms for the IVPs we typically obtained algorithms where we "march in time", i.e., we computed the value at t_1, then used that to get the solution at t_2, etc. In a BVP like our two-point BVP the boundary conditions influence the solution at the interior so we can't hope to simply start at one end of the domain and march to the other. Rather in most methods we will have to solve for the solution at all the discrete points at once. This means that discretization of a BVP results in solving a linear algebraic system of equations of the form $A\mathbf{x} = \mathbf{b}$ if the BVP is linear; otherwise we have a system of nonlinear algebraic equations to solve.

As in the case of IVPs, when we turn to approximating the solution to the BVP we give up having an analytic solution everywhere and instead seek an approximate solution at a finite number of discrete points or regions (typically called elements or cells) in the domain. So our first task in approximating the solution to a BVP is to discretize the spatial domain using a *grid* or *mesh*.

In one dimension, discretization of the domain is clear cut. We partition the interval $[a,b]$ into $n+1$ subintervals $[x_i, x_{i+1}]$ where

$$x_0 = a, \quad x_1 = x_0 + \Delta x_1, \quad \cdots \quad x_i = x_{i-1} + \Delta x_i, \quad \cdots \quad x_{N+1} = b\,.$$

If $\Delta x_i = \Delta x$ for all $i = 1, n+1$ then the grid is *uniform*. The points x_i are called the grid points or *nodes* of the mesh.

For a second order differential equation we know either the value of the unknown, its derivative or a combination of the two at $x = a$ and $x = b$. For example, if we have Dirichlet boundary conditions then there are n interior nodes where we approximate the solution. If the problem is a purely Neumann BVP then we must approximate the solution at all $n + 2$ points in the domain.

It is important to realize that obtaining a discrete approximation on a fixed grid is somewhat meaningless. Our convergence results apply in the case that $\Delta x \to 0$ so we need to approximate the solution on a set of grids where the spacing tends to zero in a uniform sense; that is, we must *refine* our grid several times. If the grid is nonuniform, we can't refine in only one portion of the domain but rather must refine throughout the domain. It is possible to get reasonable looking results on a single grid but as $\Delta x \to 0$, the results do not converge. Once a computer code has been developed to approximate the solution of a problem, one typically solves a problem whose exact solution is known; results are then produced on a sequence of refined grids and it is verified that the results converge at the predicted theoretical rate.

In higher dimensions generation of grids is much more involved but luckily there are many available software packages to assist the user. In one dimension we divided the domain into intervals but in two dimensions we can use rectangles, triangles, hexagons, curved regions, etc. and in three dimensions we can use prisms, quadrilaterals, tetrahedrons, etc. If the domain is a rectangular region then a Cartesian grid can easily be generated by taking the tensor product of a uniform one-dimensional grid in each direction. In many instances we will simply use a rectangular domain so the grid generation will be easy.

The grid one uses can have a significant impact on convergence, accuracy and CPU time required. Some desirable properties for grids include the ability to correctly mimic the shape of the domain (e.g., a square is easy but a car is not); ease in refining the grid, ability to grade smoothly from fine to coarse mesh; the ability to control the quality of the mesh (e.g., if we are using triangles we want to maintain a minimum angle condition).

5.5 Review of solving linear algebraic systems

Because discretization of BVPs typically reduces to solving a linear system, we review some topics from linear algebra here. Even if the BVP is nonlinear, we solve the resulting nonlinear algebraic system by iteration where each iteration typically requires the solution of a linear system; for example, when we use a method like the Newton-Raphson method.

5.5.1 Classes of matrices

In this section we review some of the basic definitions for matrices. We say that A is an $m \times n$ matrix if it has m rows and n columns. We will refer to the entries of

A as a_{ij} where i refers to the row and j to the column. For our purposes we are mainly concerned with square $n \times n$ matrices which are invertible, i.e., there exists an $n \times n$ matrix B such that $AB = BA = I$ where I is the $n \times n$ identity matrix (a diagonal matrix with $I_{ii} = 1$); we denote the inverse of A by A^{-1}.

Matrices are classified by their structure of zeros and by certain properties they possess. It is important to take advantage of the attributes of the matrix when we solve systems because we can often save work. We first recall the terminology used for the structure of zero entries in the matrix.

Definition 1. Let A be an $n \times n$ matrix with entries a_{ij}.

A is a **diagonal matrix** if $a_{ij} = 0$ for all $i \neq j$.

A is an **upper triangular matrix** if $a_{ij} = 0$ for all $i > j$.

A is a **lower triangular matrix** if $a_{ij} = 0$ for all $j > i$.

A is a **unit upper triangular matrix** if it is upper triangular and $a_{ii} = 1$ for $i = 1, 2, \ldots, n$.

A is a **unit lower triangular matrix** if it is lower triangular and $a_{ii} = 1$ for $i = 1, 2, \ldots, n$.

A is a **tridiagonal matrix** if $a_{ij} = 0$ for all $|i - j| > 1$.

A is a **banded matrix** of bandwidth q if $a_{ij} = 0$ for all $|i - j| > q/2$.

A is a **permutation matrix** if it can be formed by interchanging rows or columns of the identity matrix.

A is called a **sparse matrix** if it has a large portion of zero entries even if there is no pattern to the zero entries.

A is called a **dense** or **full** matrix if there are no or only a few zero entries.

Another important way to classify matrices is by their inherent properties. First recall that the transpose of a matrix A, denoted A^T is a matrix found by reflecting A along its main diagonal, i.e., the (i, j) entry of A^T is a_{ji}.

Definition 2. Let A be an $n \times n$ matrix with real entries a_{ij}.

A is a **symmetric matrix** if $a_{ij} = a_{ji}$ for all i, j.

A is **positive definite matrix** if $\mathbf{x}^T A\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

A is **positive semi-definite matrix** if $\mathbf{x}^T A\mathbf{x} \geq 0$ for all $\mathbf{x} \neq 0$.

A is an **orthogonal matrix** if $A^{-1} = A^T$, i.e., $AA^T = A^T A$.

When multiplying two matrices together (where the procedure is defined) it is important to remember that matrix multiplication is not commutative. By this we mean that in general

$$AB \neq BA.$$

5.5.2 Gauss elimination

The first method that one typically learns for solving a linear system is Gauss elimination. The basic idea is to transform the system $A\mathbf{x} = \mathbf{b}$ into an equivalent system $Ux = \mathbf{c}$ where U is an upper triangular matrix. Then this upper triangular system can be solved by a method called back solving. We will describe the matrix form of Gauss elimination but when implementing the method we do not actually construct the transformation matrices and multiply the system by them. However, this approach illustrates that LU factorization and Gauss elimination are equivalent methods "on paper". However there are applications when LU factorization is more efficient.

We define a special type of matrix called an *elementary matrix* or *Gauss transformation matrix*. This type of matrix is unit lower triangular and differs from the identity matrix in only one column below the diagonal. In addition, its inverse is easily attainable. These matrices are used to "zero out" entries in A below the diagonal; of course if we premultiply A by a matrix we must do the same thing to the right hand side of the equation. The goal is to find Gauss transformation matrices $\mathcal{M}^i$ such that

$$\mathcal{M}^q\mathcal{M}^{q-1}\cdots\mathcal{M}^2\mathcal{M}^1A = U$$

where U is upper triangular. If we can do this then we have the system

$$\mathcal{M}^q\mathcal{M}^{q-1}\cdots\mathcal{M}^2\mathcal{M}^1A\mathbf{x} = \mathcal{M}^q\mathcal{M}^{q-1}\cdots\mathcal{M}^2\mathcal{M}^1\mathbf{b} = \mathbf{c} \Rightarrow U\mathbf{x} = \mathbf{c}\,.$$

The upper triangular system $U\mathbf{x} = \mathbf{c}$ can be solved by a process called back solving. To determine the equations for $\mathbf{x}$ we equate corresponding entries of $U\mathbf{x}$ and $\mathbf{c}$. Doing this, we get the following equations.

Set $x_n = \dfrac{c_n}{u_{nn}}$

For $i = n-1, n-2, \ldots, 1$

$$x_i = \frac{c_i - \displaystyle\sum_{j=i+1}^{n} u_{i,j}x_j}{u_{ii}}\,.$$

So all that is left to solve $A\mathbf{x} = \mathbf{b}$ is to determine the matrices $\mathcal{M}^i$ and their inverse. We will explicitly give $\mathcal{M}^1$ and the remaining matrices are determined in an analogous way. We will illustrate with an example. If we have an $n \times n$ system then $\mathcal{M}^1$ has the form

$$\mathcal{M}^1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21}^1 & 1 & 0 & \cdots & 0 \\ m_{31}^1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1}^1 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

where

$$m_{21}^1 = -a_{21}/a_{11} \qquad m_{31}^1 = -a_{31}/a_{11} \qquad m_{i1}^1 = -a_{i1}/a_{11}\,.$$

The inverse of $\mathcal{M}^1$ is just

$$(\mathcal{M}^1)^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{21}^1 & 1 & 0 & \cdots & 0 \\ -m_{31}^1 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_{n1}^1 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Example 5.2. Find the Gauss transformation matrices $\mathcal{M}^1$ and $\mathcal{M}^2$ which converts

$$A = \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

to an upper triangular matrix. We have

$$\mathcal{M}^1 A = \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 \\ 3 & 4 & 7 \\ -2 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 7 & 7 \\ 0 & -2 & 1 \end{pmatrix}$$

and

$$\mathcal{M}^2(\mathcal{M}^1 A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2/7 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 \\ 0 & 7 & 7 \\ 0 & -2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 7 & 7 \\ 0 & 0 & 3 \end{pmatrix} = U\,.$$

When we have transformed a general system to an upper triangular system by using Gauss transformation matrices then we have essentially performed an LU decomposition of A, i.e., written A as the product of a unit lower triangular matrix and an upper triangular matrix. To see this note that because

$$\mathcal{M}^q \mathcal{M}^{q-1} \cdots \mathcal{M}^2 \mathcal{M}^1 A = U$$

where U is an upper triangular matrix and because each $\mathcal{M}^i$ has an inverse which is unit lower triangular we have

$$A = \left[(\mathcal{M}^1)^{-1} (\mathcal{M}^2)^{-1} \cdots (\mathcal{M}^{q-1})^{-1} (\mathcal{M}^q)^{-1} \right] U\,.$$

Because the product of two unit lower triangular matrices is also unit lower triangular, then we have $A = LU$ where

$$L = (\mathcal{M}^1)^{-1} (\mathcal{M}^2)^{-1} \cdots (\mathcal{M}^{q-1})^{-1} (\mathcal{M}^q)^{-1}\,.$$

5.5.3 LU factorization

The process of GE essentially factors a matrix A into LU where L is unit lower triangular and U is upper triangular. Now we want to see how this factorization allows us to solve linear systems and why in many cases it is the preferred algorithm compared with GE. Remember on paper, these methods are the same but computationally they can be different.

First, suppose we want to solve $A\mathbf{x} = \mathbf{b}$ and we are given the factorization $A = LU$. It turns out that the system $LU\mathbf{x} = \mathbf{b}$ is "easy" to solve because we do a *forward solve* followed by a *backward solve*.

Forward Solve: $L\mathbf{y} = \mathbf{b}$ Back Solve: $U\mathbf{x} = \mathbf{y}$.

We have seen that we can easily implement the equations for the back solve and it is straightforward to write out the equations for the forward solve.

Example 5.3. If

$$A = \begin{pmatrix} 2 & -1 & 2 \\ 4 & 1 & 9 \\ 8 & 5 & 24 \end{pmatrix} = LU = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 2 \\ 0 & 3 & 5 \\ 0 & 0 & 1 \end{pmatrix}$$

solve the linear system $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b} = (0, -5, -16)^T$.

We first solve $L\mathbf{y} = \mathbf{b}$ to get $y_1 = 0$; $2y_1 + y_2 = -5$ implies $y_2 = -5$ and $4y_1 + 3y_2 + y_3 = -16$ implies $y_3 = -1$. Now we solve $U\mathbf{x} = \mathbf{y} = (0, -5, -1)^T$. Back solving yields $x_3 = -1$, $3x_2 + 5x_3 = -5$ implies $x_2 = 0$ and finally $2x_1 - x_2 + 2x_3 = 0$ implies $x_1 = 1$ giving the solution $(1, 0, -1)^T$.

If GE and LU factorization are equivalent on paper, why would one be computationally advantageous in some settings? Recall that when we solve $A\mathbf{x} = \mathbf{b}$ by GE we must also multiply the right hand side by the Gauss transformation matrices. Often in applications, we have to solve many linear systems where the coefficient matrix is the same but the right hand side vector changes. If we have all of the right hand side vectors at one time, then we can treat them as a rectangular matrix and multiply this by the Gauss transformation matrices. However, in many instances we solve a single linear system and use its solution to compute a new right hand side, i.e., we don't have all the right hand sides at once. This will be the case when we solve time dependent BVPs, i.e., initial boundary value problems. When we perform an LU factorization then we overwrite the factors onto A and if the right hand side changes, we simply do another forward and back solve to find the solution.

One can easily derive the equations for an LU factorization by writing $A = LU$

and equating entries. Consider the matrix equation $A = LU$ written as

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{pmatrix}$$

Now equating the $(1,1)$ entry gives

$$a_{11} = 1 \cdot u_{11} \Rightarrow u_{11} = a_{11}$$

In fact, if we equate each entry of the first row of A, i.e., a_{1j} we get

$$u_{1j} = a_{1j} \quad \text{for } j = 1, \ldots, n.$$

Now we move to the second row and look at the (2,1) entry to get $a_{21} = \ell_{21} \cdot u_{11}$ which implies $\ell_{21} = a_{21}/u_{11}$. Now we can determine the remaining terms in the first column of L by

$$\ell_{i1} = a_{i1}/u_{11} \quad \text{for } i = 2, \ldots, n.$$

We now find the second row of U. Equating the (2,2) entry gives $a_{22} = \ell_{21}u_{12} + u_{22}$ implies $u_{22} = a_{22} - \ell_{21}u_{12}$. In general

$$u_{2j} = a_{2j} - \ell_{21}u_{1j} \quad \text{for } j = 2, \ldots, n.$$

We now obtain formulas for the second column of L. Equating the (3,2) entries gives

$$\ell_{31}u_{12} + \ell_{32}u_{22} = a_{32} \Rightarrow \ell_{32} = \frac{a_{32} - \ell_{31}u_{12}}{u_{22}}$$

and equating $(i,2)$ entries for $i = 3, 4, \ldots, n$ gives

$$\ell_{i2} = \frac{a_{i2} - \ell_{i1}u_{12}}{u_{22}} \quad i = 3, 4, \ldots, n$$

Continuing in this manner, we get the following algorithm.

Theorem 5.1. *Let A be a given $n \times n$ matrix. Then if no pivoting is needed, the LU factorization of A into a unit lower triangular matrix L with entries ℓ_{ij} and an upper triangular matrix U with entries u_{ij} is given by the following equations.*

Set $u_{1j} = a_{1j}$ for $j = 1, \ldots, n$

For $k = 1, 2, 3 \ldots, n-1$

for $i = k+1, \ldots, n$

$$\ell_{i,k} = \frac{a_{i,k} - \sum_{m=1}^{k-1} \ell_{im} u_{m,k}}{u_{k,k}}$$

for $j = k+1, \ldots, n$

$$u_{k+1,j} = a_{k+1,j} - \sum_{m=1}^{k} \ell_{k+1,m} u_{m,j} \, .$$

Note that this algorithm clearly demonstrates that you can NOT find all of L and then all of U or vice versa. One must determine a row of U, then a column of L, then a row of U, etc.

Does LU factorization work for all systems that have a unique solution? The following example demonstrates that not every invertible matrix has an LU factorization without row or column interchanges. The following theorem states that if we interchange rows of a matrix and then find an LU factorization.

Example 5.4. Consider $A\mathbf{x} = \mathbf{b}$ where

$$A\mathbf{x} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

which has the unique solution $\mathbf{x} = (1, 1)^T$. Can you find an LU factorization of A? Just like in GE the (1,1) entry is a zero pivot and so we can't find u_{11}.

Theorem 5.2. *Let A be an $n \times n$ matrix. Then there exists a permutation matrix P such that*

$$PA = LU$$

where L is unit lower triangular and U is upper triangular.

There are several variants of LU factorization which we briefly describe.

- $A = LU$ where L is lower triangular and U is unit upper triangular.
- $A = LDU$ where L is unit lower triangular, U is unit upper triangular and D is diagonal.
- If A is symmetric and positive definite then $A = LL^T$ where L is lower triangular. This is known as Cholesky decomposition. If the diagonal entries of L are chosen to be positive, then the decomposition is unique. This is an important decomposition for us because our matrices will often be symmetric and positive definite.

To see the equations for the Cholesky decomposition we equate entries on each side of the matrix equation:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \ell_{nn} \end{pmatrix} \begin{pmatrix} \ell_{11} & \ell_{21} & \ell_{31} & \cdots & \ell_{n1} \\ 0 & \ell_{22} & \ell_{32} & \cdots & \ell_{n2} \\ 0 & 0 & \ell_{33} & \cdots & \ell_{n3} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \ell_{nn} \end{pmatrix}.$$

Equating the (1,1) entry gives

$$\ell_{11} = \sqrt{a_{11}}\,.$$

Clearly, a_{11} must be ≥ 0 which is guaranteed by the fact that A is positive definite (just choose $\mathbf{x} = (1, 0, \ldots, 0)^T$). Next we see that

$$\ell_{11}\ell_{i1} = a_{1i} = a_{i1} \Rightarrow \ell_{i1} = \frac{a_{i1}}{\ell_{11}}, \quad i = 2, 3, \ldots, n$$

Then to find the next diagonal entry we have

$$\ell_{21}^2 + \ell_{22}^2 = a_{22} \Rightarrow \ell_{22} = \left(a_{22} - \ell_{21}^2\right)^{1/2}$$

and the remaining terms in the second row are found from

$$\ell_{i2} = \frac{a_{i2} - \ell_{i1}\ell_{21}}{\ell_{22}} \quad i = 3, 4, \ldots, n\,.$$

Continuing in this manner we have the following algorithm.

Theorem 5.3. *Let A be a symmetric, positive definite matrix. Then the Cholesky factorization $A = LL^T$ is given by the following algorithm.*

For $i = 1, 2, 3, \ldots, n$

$$\ell_{ii} = \left(a_{ii} - \sum_{j=1}^{i-1} \ell_{ij}^2\right)^{1/2}$$

for $k = i + 1, \ldots, n$

$$\ell_{ki} = \ell_{ik} = \frac{1}{\ell_{ii}}\left[a_{ki} - \sum_{j=1}^{i-1} \ell_{kj}\ell_{ij}\right]$$

One can show that if A is a symmetric matrix, then it is positive definite if and only if $A = LL^T$. So this means that if we have a symmetric matrix and want to determine if it is positive definite we can attempt to perform a Cholesky decomposition. If it is not positive definite the algorithm will fail when a square root of a negative number is attempted.

Operation Count

One way to compare the work required to solve a linear system by different methods is to determine the number of operations required to find the solution. In Table 5.1 we summarize the operation counts for various operations. So that you can see how these values are obtained, we provide the details for the operation count of a back solve in the following example.

Example 5.5. OPERATION COUNT Suppose we are given an $n \times n$ upper triangular matrix U with entries u_{ij} and an n-vector $\mathbf{b}$ with components b_i then we know that the solution of $U\mathbf{x} = \mathbf{b}$ is given by the following steps.

Set $x_n = \dfrac{b_n}{u_{nn}}$
For $i = n-1, n-2, \ldots, 1$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} u_{i,j}x_j}{u_{ii}}\,.$$

Provide the number of multiplication/divisions and additions/subtractions to solve an $n \times n$ upper triangular system using these equations.

For x_n we require one division; we will count multiplications and divisions the same. For x_{n-1} we have one multiplication, one division and one addition. For x_{n-2} we have two multiplications, one division and two additions. We have

entry	multiplications	divisions	additions
x_n	0	1	0
x_{n-1}	1	1	1
x_{n-2}	2	1	2
x_{n-3}	3	1	3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_1	n	1	n-1

So counting multiplications and divisions as the same we have

$$(n) + (1 + 2 + 3 + \cdots + n) = n + \sum_{i=1}^{n} i \quad \text{multiplications/divisions}$$

and

$$\sum_{i=1}^{n-1} i \quad \text{additions.}$$

Now we would like to have the result in terms of $\mathcal{O}(n^r)$ for some r. If you recall from calculus

$$\sum_{i=1}^{p} i = \frac{p(p+1)}{2} \qquad \sum_{i=1}^{p} i^2 = \frac{p(p+1)(2p-1)}{6}$$

Using this first expression we obtain

$$n + \frac{n^2 + n}{2} = \mathcal{O}(n^2) \quad \text{multiplications/divisions}$$

and

$$\frac{(n-1)^2 + (n-1)}{2} = \mathcal{O}(n^2) \quad \text{additions}$$

So we say that performing a back solve requires $\mathcal{O}(n^2)$ operations.

procedure	# multiplications/divisions	# square roots
dot product of two vectors	n	
matrix times vector	n^2	
back solve	$\frac{n^2}{2}$	
forward solve	$\frac{n^2}{2}$	
LU factorization	$\frac{n^3}{3}$	
LL^T factorization	$\frac{n^3}{6}$	n
LU factorization where A is tridiagonal	$4n$	
LU factorization where A has bandwidth q	$q^2 n$	

Table 5.1: Operation count for various calculations in linear algebra. We assume that the given matrix is $n \times n$ and any vectors are of length n.

It is important to realize that solving a full $n \times n$ matrix requires $\mathcal{O}(n^3)$ operations. This means that if we double the size of the matrix to $2n \times 2n$ the work does not double but rather goes up by a factor of eight!

5.5.4 Iterative methods

There are basically two broad classes of methods for solving $A\mathbf{x} = \mathbf{b}$. The first is direct methods such as GE and LU factorization and its variants. If we used exact arithmetic then direct methods find the *exact* solution in a finite number of steps. Iterative methods form a sequence of approximations $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^k$ to the exact solution $\mathbf{x}$ and we hope that this sequence converges to the exact solution but it will typically never reach the exact solution even if we do exact arithmetic. However, we perform iterations until the approximation is within a desired tolerance.

Why do we want to look at solvers other than direct solvers? The main reason is storage. Oftentimes (especially in 3D calculations) we have a working code but we are unable to store our coefficient matrix when we attempt to do fine grid calculations even when we take advantage of the structure of our matrix. In addition,

sometimes iterative methods may take less work than a direct method if we have a good initial guess. For example, if we are solving a time dependent problem for a sequence of time steps, $\Delta t, 2\Delta t, 3\Delta t, \ldots$ then we can use the solution at $k\Delta t$ as a starting guess for our iterative method at $(k+1)\Delta t$ and if Δt is sufficiently small, the method may converge in just a handful of iterations. If we have a large sparse matrix (the structure of zeros is random or there are many zeros inside the bandwidth) then one should use a method which only stores the nonzero entries in the matrix. There are direct methods for sparse matrices but they are much more complicated than iterative methods for sparse matrices.

A good (free) online source for iterative methods for solving $A\mathbf{x} = \mathbf{b}$ is given in the description of a set of iterative solvers called TEMPLATES found at `netlib`:

`http://www.netlib.org/linalg/html_templates/Templates.html`

There are complications to implementing iterative methods that do not occur in direct methods. When we use an iterative method, the first thing we realize is that we have to decide how to terminate our method; this was not an issue with direct methods. Another complication is that many methods are only guaranteed to converge for certain classes of matrices. We will also find in some methods that it is straightforward to find a search direction but determining how far to go in that direction is not known. Lastly we need a starting guess for the iterative method; in many applications such as time dependent problems we will have a ready initial guess. However, unlike iterative methods for nonlinear equations, if an iterative method for a linear system converges, then it will do so for any initial guess.

There are two basic types of iterative methods:

1. **Stationary methods**. These are methods where the data in the equation to find x^{k+1} remains fixed; they have the general form

   $$x^{k+1} = P\mathbf{x}^k + \mathbf{c} \qquad \text{for a fixed matrix } P \text{ and a fixed vector } \mathbf{c}$$

   We call P the *iteration matrix* and its dominant eigenvalue dictates whether the method will converge or not.

2. **Nonstationary methods.** These are methods where the data changes at each iteration; they have the form

   $$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$$

   Note here that the data, α_k and $\mathbf{p}^k$ change for each iteration k. Here $\mathbf{p}^k$ is called the *search direction* and α_k the *step length.*

Stationary Iterative Methods

The basic idea is to *split* (not factor) A as the sum of two matrices

$$A = M - N$$

where M is easily invertible. Then we have

$$A\mathbf{x} = \mathbf{b} \Rightarrow (M-N)\mathbf{x} = \mathbf{b} \Rightarrow M\mathbf{x} = N\mathbf{x} + \mathbf{b} \Rightarrow \mathbf{x} = M^{-1}N\mathbf{x} + M^{-1}\mathbf{b}$$

This suggests the iteration

$$\text{Given } \mathbf{x}^0 \text{ then} \qquad \mathbf{x}^{k+1} = M^{-1}N\mathbf{x}^k + M^{-1}\mathbf{b}, \quad k = 0, 1, 2, \ldots$$

which is a stationary method with $P = M^{-1}N$ and $\mathbf{c} = M^{-1}\mathbf{b}$. There are three basic methods which make different choices for M and N. We will look at all three.

The simplest choice for M is a *diagonal* matrix because it is the easiest to invert. This choice leads to the **Jacobi Method**. We write

$$A = L + D + U$$

where here L is the lower portion of A, D is its diagonal and U is the upper part. For the Jacobi method we choose $M = D$ and $N = -(L+U)$ so $A = M - N$ is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix} - \begin{pmatrix} 0 & -a_{12} & -a_{13} & \cdots & -a_{1n} \\ -a_{21} & 0 & -a_{23} & \cdots & -a_{2n} \\ -a_{31} & -a_{32} & 0 & \cdots & -a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & 0 \end{pmatrix}$$

Then our iteration becomes

$$\mathbf{x}^{k+1} = -D^{-1}(L+U)\mathbf{x}^k + D^{-1}\mathbf{b} \qquad \text{with iteration matrix } P = -D^{-1}(L+U)$$

However, to implement this method we don't actually form the matrices rather we look at the equations for each component. The point form (i.e., the equation for each component of $\mathbf{x}^{k+1}$) is given by the following:

Jacobi Method: Given $\mathbf{x}^0$, then for $k = 0, 1, 2, \ldots$ find

$$x_i^{k+1} = -\frac{1}{a_{ii}}\Big(\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^k\Big) + \frac{1}{a_{ii}}b_i$$

This corresponds to the iteration matrix $P_{\text{J}} = -D^{-1}(L+U)$.

Example 5.6. Apply the Jacobi Method to the system $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

with an exact solution $(1, 2, 3)^T$. Use $\mathbf{x}^0 = (0, 0, 0)^T$ as a starting guess. We find each component by our formula above.

$$x_1^1 = -\frac{1}{1}(0) + \frac{2}{1} = 2$$

$$x_2^1 = -\frac{1}{20}(0) + \frac{36}{20} = 1.8$$

$$x_3^1 = -\frac{1}{10}(0) + \frac{25}{10} = 2.5$$

Continuing in this manner we get the following table

k	x_1^k	x_2^k	x_3^k
0	0	0	0
1	2	1.8	2.5
2	0.9	1.85	3.06
3	1.36	2.016	2.96
5	1.12	2.010	2.98
10	0.993	1.998	3.00

The **Gauss-Seidel method** is based on the observation that when we calculate, e.g., x_1^{k+1}, it is supposedly a better approximation than x_1^k so why don't we use it as soon as we calculate it. The implementation is easy to see in the point form of the equations.

Gauss-Seidel Method: Given $\mathbf{x}^0$, then for $k = 0, 1, 2, \ldots$ find

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

This corresponds to the iteration matrix $P_{\text{GS}} = -(D + L)^{-1}U$.

We now want to determine the matrix form of the method so we can determine the iteration matrix P. We note that the point form becomes

$$\mathbf{x}^{k+1} = -D^{-1}\Big(L\mathbf{x}^{k+1}\Big) - D^{-1}\Big(U\mathbf{x}^k\Big) + D^{-1}\mathbf{b}\,.$$

Now grouping the terms at the $(k+1)$st iteration on the left and multiplying through by D we have

$$D\mathbf{x}^{k+1} + \Big(L\mathbf{x}^{k+1}\Big) = -\Big(U\mathbf{x}^k\Big) + \mathbf{b}$$

or

$$(D + L)\mathbf{x}^{k+1} = -U\mathbf{x}^k + \mathbf{b} \Rightarrow P = -(D + L)^{-1}U\,.$$

Example 5.7. Apply the Gauss-Seidel method to the system in the previous example, i.e.,

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

with an exact solution $(1, 2, 3)^T$. Use $\mathbf{x}^0 = (0, 0, 0)^T$ as a starting guess. We find each component by our formula above.

$$x_1^1 = -\frac{1}{1}(0) + \frac{2}{1} = 2$$

$$x_2^1 = -\frac{1}{20}(2 * 2) + \frac{36}{20} = -.2 + 1.8 = 1.6$$

$$x_3^1 = -\frac{1}{10}((-1)2 - 2(1.6)) + \frac{25}{10} = .52 + 2.5 = 3.02$$

Continuing in this manner we get the following table

k	x_1^k	x_2^k	x_3^k
0	0	0	0
1	2	1.6	3.02
2	1.82	1.92	3.066
3	1.226	1.984	3.0194
5	1.0109	1.99	3.001

So for this example, the Gauss-Seidel method is converging much faster because remember for the fifth iteration of Jacobi we had $(1.12, 2.01, 2.98)^T$ as our approximation.

The **Successive Over Relaxation (SOR)** method takes a *weighted average* between the previous iteration and the result we would get if we took a Gauss-Seidel step. For one choice of the weight, it reduces to the Gauss-Seidel method.

SOR Method: Given $\mathbf{x}^0$, then for $k = 0, 1, 2, \ldots$ find

$$x_i^{k+1} = (1-\omega)\mathbf{x}^k + \omega\Big[-\frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} a_{ij}x_j^{k+1}\Big) - \frac{1}{a_{ii}}\Big(\sum_{j=i+1}^{n} a_{ij}x_j^k\Big) + \frac{1}{a_{ii}}b_i\Big]$$

where $0 < \omega < 2$. This corresponds to the iteration matrix $P_{\text{SOR}} = (D + \omega L)^{-1}\Big((1-\omega)D - \omega U\Big)$.

We first note that if $\omega = 1$ we get the Gauss-Seidel method. If $\omega > 1$ then we say that we are over-relaxing and if $\omega < 1$ we say that we are under-relaxing. Of course there is a question as to how to choose ω which we will address shortly.

We need to determine the iteration matrix for SOR. From the point form we have

$$D\mathbf{x}^{k+1} + \omega L\mathbf{x}^{k+1} = (1-\omega)D\mathbf{x}^k - \omega U\mathbf{x}^k + \mathbf{b}$$

which implies

$$\mathbf{x}^{k+1} = (D+\omega L)^{-1}\Big((1-\omega)D - \omega U\Big)\mathbf{x}^k + (D+\omega L)^{-1}\mathbf{b}$$

so that $P = (D+\omega L)^{-1}\Big((1-\omega)D - \omega U\Big)$.

Example 5.8. Let's return to our example and compute some iterations using different values of ω. Recall that

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

We find each component by our formula above. Using $\omega = 1.1$ and $\omega = 0.9$ we have the following results.

	$\omega = 1.1$				$\omega = 0.9$		
k	x_1^k	x_2^k	x_3^k		x_1^k	x_2^k	x_3^k
0	0	0	0		0	0	0
1	2.2	1.738	3.3744		1.8	1.458	2.6744
2	1.862	1.9719	3.0519		1.7626	1.8479	3.0087
3	1.0321	2.005	2.994		1.3579	1.9534	3.0247
5	0.9977	2.0000	2.9999		1.0528	1.9948	3.0051

Example 5.9. As a last example consider the system $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 4 \\ 3 & 4 & 5 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 13 \\ 13 \\ 26 \end{pmatrix}$$

and apply Jacobi, Gauss-Seidel and SOR with $\omega = 1.1$ with an initial guess of $(1,1,1)$. The exact solution is $(1,2,3)^T$.

	Jacobi				Gauss-Seidel				SOR		
k	x_1^k	x_2^k	x_3^k		x_1^k	x_2^k	x_3^k		x_1^k	x_2^k	x_3^k
1	1	1	1		1	1	1		1	1	1
1	4.5	-8.0	3.8		4.5	-4.5	6.1		4.85	-4.67	6.52
2	4.8	6.7	8.9		-.4	11	-3.36		-1.53	13.18	-5.52
3	-10.2	27.4	-3.04		6.04	-20.4	17.796		9.16	-29.84	26.48

As you can see from these calculations, all methods fail to converge even though A was symmetric.

One complication with iterative methods is that we have to decide when to terminate the iteration. A criteria that is often used is to make sure that the *residual* $\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$ is sufficiently small. Of course, the actual size of $\|\mathbf{r}^k\|$ is

not as important as its *relative* size. If we have a tolerance τ, then one criteria for termination is

$$\frac{\|\mathbf{r}^k\|}{\|\mathbf{r}^0\|} < \tau\,,$$

where $\|\mathbf{r}^0\|$ is the initial residual $\mathbf{b} - A\mathbf{x}^0$.

The problem with this criterion is that it depends on the initial iterate and may result in unnecessary work if the initial guess is too good and an unsatisfactory approximation $\mathbf{x}^k$ if the initial residual is large. For these reasons it is usually better to use

$$\frac{\|\mathbf{r}^k\|}{\|\mathbf{b}\|} < \tau\,.$$

Note that if $\mathbf{x}^0 = \vec{0}$, then the two are identical.

Another stopping criteria is to make sure that the difference in successive iterations is less than some tolerance; again the magnitude of the actual difference is not as important as the relative difference. Given a tolerance σ we could use

$$\frac{\|\mathbf{x}^{k+1} - \mathbf{x}^k\|}{\|\mathbf{x}^{k+1}\|} \leq \sigma$$

Often a combination of both criteria are used.

5.5.5 Nonstationary Iterative Methods

Recall that nonstationary iterative methods have the general form

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$$

The vector $\mathbf{p}^k$ is called the *search direction* and the scalar α_k is called the *step length*. Unlike stationary methods, nonstationary methods do not have an iteration matrix.

The classic example of a nonstationary method is the **Conjugate Gradient (CG)** method. However, CG only works for symmetric positive definite matrices but there have been many variants of it (e.g., BICG, BICGSTAB) developed which handle even indefinite matrices. Another example of a nonstationary method is the **Steepest Descent** method which is based on the following simple fact from calculus. If we want to minimize a function f then we know that $-\nabla f$ points in the direction of the maximum decrease in f. So a simple iterative method to minimize a function is to start at a point, compute the gradient of the function at the point and take a step in the direction of minus the gradient. But what does minimizing a function have to do with solving a linear system? The following proposition gives us the answer in the case of a symmetric positive definite matrix.

Proposition 5.1. *If A is an $n \times n$ symmetric positive definite matrix then the solution $\mathbf{x} = A^{-1}\mathbf{b}$ of the linear system $A\mathbf{x} = \mathbf{b}$ is equivalent to*

$$\mathbf{x} = \min_{\mathbf{y}\in\mathbb{R}^n} \phi(\mathbf{y}) \quad \textit{where} \quad \phi(\vec{y}) = \frac{1}{2}\mathbf{y}^T A\mathbf{y} - \mathbf{b}^T\mathbf{y}\,.$$

To understand the CG method it is advantageous to analyze the method of steepest descent. Due to time constraints we will not go into these methods here but the interested reader is referred to standard texts in numerical analysis.

Chapter 6

Finite Difference Methods for Boundary Value Problems

In this chapter we look at finite difference methods for boundary value problems (BVPs). The main idea in the finite difference approach to solving differential equations is to replace the derivatives in the equation with difference quotients. The first step is to overlay the domain with a grid or mesh and then a difference equation is written at the appropriate nodes. As mentioned in the last chapter discretization of BVPs requires the solution of a system of algebraic equations unlike IVPs where we "marched in time."

In this chapter we begin by looking at a BVP in one dimension which is often called a two-point BVP. We consider different difference quotients to approximate first and second derivatives and determine their accuracy. We will see how to implement different boundary conditions in the context of finite difference methods. In finite difference methods one often refers to a method's stencil which we will define and depict in a diagram. Our results in one dimension will allow us to easily move to approximating our prototype equation, the Poisson equation, in two or three dimensions. We want to see the difference in the structure of our coefficient matrix as we move from one dimension to higher dimensions. Numerical results will be presented.

6.1 The two-point BVP

We first want to develop finite difference schemes for a BVP which is governed by the differential equation

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x) \quad a < x < b\,, \tag{6.1}$$

plus boundary conditions at $x = a$ and $x = b$. Here $p(x)$, $q(x)$ are required to satisfy the bounds

$$0 < p_{\min} \le p \le p_{\max} \quad \text{and} \quad q_{\min} = 0 \le q(x) \le q_{\max}\,. \tag{6.2}$$

When $p = 1$ and $q = 0$ we have the Poisson equation $-u''(x) = f(x)$ in one dimension.

For existence and uniqueness we require that f and q be continuous functions on the domain $[a, b]$ and that p has a continuous first derivative there in addition to the given bounds (6.2). This equation always contains the second derivative $u''(x)$ because $p(x) > 0$; thus it is second order. If $p(x)$ is not a constant the equation also includes the first derivative $u'(x)$ because when we use the product rule for differentiation we have

$$\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) = p(x)\frac{d^2u}{dx^2} + p'(x)\frac{du}{dx}\,.$$

Consequently to solve the general equation we need difference quotients to approximate both the first and second derivatives. In Chapter 1 we obtained both the forward and backward difference quotients for the first derivative and saw that each was first order accurate. Specifically we have

$$\textbf{Forward Difference:} \quad u'(x) = \frac{u(x+h) - u(x)}{h} + \mathcal{O}(h)$$

$$\textbf{Backward Difference:} \quad u'(x) = \frac{u(x) - u(x-h)}{h} + \mathcal{O}(h)$$

Before we decide if either of these difference quotients are appropriate for this problem, we first derive our difference quotient for the second derivative.

Recall that Taylor series are useful in deriving difference quotients. A Taylor series expansion for $u(x + h)$ is

$$u(x+h) = u(x) + h\,u'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u'''(x) + \mathcal{O}(h^4)\,. \tag{6.3}$$

Now we want an approximation for $u''(x)$ but if we solve for it in (6.3) then we still have the $u'(x)$ term. However, if we add (6.3) to the expansion for $u(x - h)$ given by

$$u(x-h) = u(x) - h\,u'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + \mathcal{O}(h^4) \tag{6.4}$$

then we can eliminate the $u'(x)$ term by adding the two expansions; we have

$$u(x+h) + u(x-h) - 2u(x) = h^2u''(x) + \mathcal{O}(h^4)$$

which gives

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2)\,.$$

Note that the terms involving h^3 cancel. This difference quotient is called a **second centered difference quotient** or a second order central difference approximation to $u''(x)$ and is second order accurate.

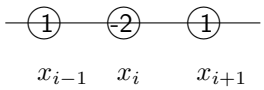
Second centered difference: $$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2) \tag{6.5}$$

Another way to derive this approximation is to difference the forward and backward approximations to the first derivative, i.e.,

$$\frac{1}{h}\left(\frac{u(x+h) - u(x)}{h} - \frac{u(x) - u(x-h)}{h}\right);$$

hence the name second difference.

Finite difference approximations are often described in a pictorial format by giving a diagram indicating the points used in the approximation. These are called finite difference **stencils** and this second centered difference is called a *three point stencil* for the second derivative in one dimension.



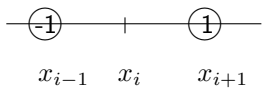
Now that we have an approximation for $u''(x)$ we need to decide which difference quotient to use to approximate the first derivative. Because our approximation to $u''(x)$ is second order, we would like to use the same accuracy for approximating the first derivative. However, both the forward and backward difference quotients are only first order. Consequently, if we use either of these two then the error in approximating $u'(x)$ will dominate and make the overall error first order. A reasonable approach is to find a second order approximation to $u'(x)$. Returning to our Taylor series expansions (6.3) and (6.4) we see that if we subtract these expansions, then we obtain

$$u(x+h) - u(x-h) = 2hu'(x) + \mathcal{O}(h^3)$$

which gives the (first) centered difference

First centered difference: $$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + \mathcal{O}(h^2) \tag{6.6}$$

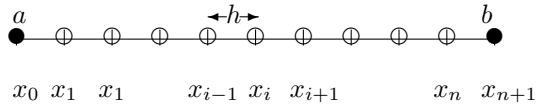
and thus if we use this difference quotient both approximations are second order accurate. It is described by the stencil



Now that we have chosen the difference quotients to approximate $u''(x)$ and $u'(x)$ we now discretize the domain. Suppose that we subdivide our domain $[a, b]$ into $n+1$ subintervals using the $(n+2)$ uniformly spaced points x_i, $i = 0, 1, \ldots n+1$ with

$$x_0 = a, x_1 = x_0 + h, \ldots, x_i = x_{i-1} + h, \ldots, x_{N+1} = x_n + h = b \qquad (6.7)$$

where $h = (b-a)/(n+1)$. The points x_i are called the *grid points* or *nodes*. The nodes $x_1, x_2, \ldots, x_n$ are *interior nodes* (denoted by open circles in the diagram below) and the two nodes x_0, x_{n+1} are *boundary nodes* (denoted by solid circles in the diagram).



We now have the machinery to write a difference equation for (6.1) at the point x_i using the two difference quotients (6.5) and (6.6). If we let $U_i \approx u(x_i)$ then our finite difference equation at the node x_i is

$$p(x_i)\left(\frac{-U_{i+1} + 2U_i - U_{i-1}}{h^2}\right) - p'(x_i)\left(\frac{U_{i+1} - U_{i-1}}{2h}\right) + q(x_i)U_i = f(x_i)\,. \quad (6.8)$$

Suppose now that we have homogeneous Dirichlet boundary conditions $u(a) = u(b) = 0$; clearly this implies $U_0 = 0$ and $U_{n+1} = 0$ so we have n unknowns U_i, $i = 1, \ldots, n$ and an equation of the form (6.8) at each of the n interior grid points $x_1, x_2, \ldots, x_n$. Let's write the difference equation at the first interior node x_1; we have

$$p(x_1)\left(\frac{-U_2 + 2U_1}{h^2}\right) - p'(x_1)\left(\frac{U_2}{2h}\right) + q(x_1)U_1 = f(x_1)\,,$$

where we have used the boundary condition to set $U_0 = 0$. We immediately realize that this equation contains two unknowns U_1 and U_2 and so we can't solve it. When we write the difference equation at the next point x_2, it contains three unknowns U_1, U_2 and U_3

$$p(x_2)\left(\frac{-U_3 + 2U_2 - U_1}{h^2}\right) - p'(x_2)\left(\frac{U_3 - U_1}{2h}\right) + q(x_2)U_2 = f(x_2)\,.$$

In fact, when we look at the equation at the point x_i we see that it will always have three unknowns U_{i-1}, U_i and U_{i+1} except at the nodes adjacent to the boundary. It makes sense that we can't solve for U_1 and then U_2, etc. because the right boundary condition must affect the solution too. Consequently, we must solve for all of the unknowns at one time by writing the n difference equations as a linear system of algebraic equations which can be represented by a matrix problem $A\mathbf{x} = \mathbf{b}$.

For simplicity of exposition, let's look at the case where $p(x)$ is a constant, say $p(x) = 1$, and $q = 0$. Then we have the simplified difference equation

$$\frac{-U_{i+1} + 2U_i - U_{i-1}}{h^2} = f(x_i) \quad i = 1, \ldots, n$$

at each interior grid point $x_1, x_2, \ldots, x_n$. Multiplying by h^2 produces

$$-U_{i-1} + 2U_i - U_{i+1} = h^2 f(x_i) \quad i = 1, \ldots, n\,.$$

The corresponding matrix problem is $A\mathbf{U} = \mathbf{f}$ where A is the matrix

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & & 0 \\ -1 & 2 & -1 & 0 & \cdots & & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ 0 & \cdots & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} \tag{6.9}$$

and $\mathbf{U} = \big(U_1, U_2, \cdots, U_n\big)^T$, $\mathbf{f} = h^2\big(f(x_1), f(x_2), \ldots, f(x_n)\big)^T$ for homogeneous Dirichlet boundary data. Clearly this matrix is symmetric and tridiagonal; in addition, it can be shown to be positive definite so the Cholesky factorization $A = LL^T$ for a tridiagonal matrix can be used. Recall that tridiagonal systems require only $\mathcal{O}(n)$ operations to solve and only three vectors must be stored to specify the matrix. In our case the matrix is symmetric and so only two vectors are required; this should be contrasted with a full $n \times n$ matrix which requires n^2 storage and $\mathcal{O}(n^3)$ operations to solve.

Example 6.1. Suppose we want to use finite differences to approximate the solution of the BVP

$$\begin{aligned} -u''(x) &= \pi^2 \sin(\pi x) \quad 0 < x < 1 \\ u(0) &= 0, \quad u(1) = 0 \end{aligned}$$

using $h = 1/4$. Our grid will contain five total grid points $x_0 = 0$, $x_1 = 1/4$, $x_2 = 1/2$, $x_3 = 3/4$, $x_4 = 1$ and three interior points x_1, x_2, x_3. Thus we have three unknowns U_1, U_2, U_3. We will write the equation at each interior node to

demonstrate that we get the tridiagonal system. We have

$$\begin{aligned} 2U_1 - U_2 &= \frac{\pi^2}{16}\sin(\frac{\pi}{4}) \\ -U_1 + 2U_2 - U_3 &= \frac{\pi^2}{16}\sin(\frac{\pi}{2}) \\ -U_2 + 2U_3 &= \frac{\pi^2}{16}\sin(\frac{3\pi}{4})\,. \end{aligned}$$

Writing these three equations as a linear system gives

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} = \frac{\pi^2}{16} \begin{pmatrix} \sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{2}) \\ \sin(\frac{3\pi}{4}) \end{pmatrix} = \begin{pmatrix} 0.436179 \\ 0.61685 \\ 0.436179 \end{pmatrix}.$$

Solving this system gives $U_1 = 0.7446$, $U_2 = 1.0530$ and $U_3 = 0.7446$; the exact solution to this problem is $u = \sin(\pi x)$ so at the interior nodes we have the exact solution $(0.7071, 1, 0.7071)$.

Example 6.2. In this example we modify our boundary conditions to be inhomogeneous Dirichlet so we can see how to handle these; we will discuss other boundary conditions in detail in § 6.2.1. Consider the BVP

$$\begin{aligned} -u''(x) &= \pi^2\cos(\pi x) \quad 0 < x < 1 \\ u(0) &= 1, \quad u(1) = -1 \end{aligned}$$

whose exact solution is $u(x) = \cos(\pi x)$. Using the same grid as in the previous example, we still have three unknowns so we write the equations at the three interior nodes

$$\begin{aligned} -U_0 + 2U_1 - U_2 &= \frac{\pi^2}{16}\cos(\frac{\pi}{4}) \\ -U_1 + 2U_2 - U_3 &= \frac{\pi^2}{16}\cos(\frac{\pi}{2})) \\ -U_2 + 2U_3 - U_4 &= \frac{\pi^2}{16}\cos(\frac{3\pi}{4})) \end{aligned}$$

Now $U_0 = 1$ and $U_4 = -1$ so we simply substitute these values into the equations

and move the terms to the right hand side to get

$$\begin{aligned} 2U_1 - U_2 &= \frac{\pi^2}{16}\cos(\frac{\pi}{4}) + 1) \\ -U_1 + 2U_2 - U_3 &= \frac{\pi^2}{16}\cos(\frac{\pi}{2})) \\ -U_2 + 2U_3 &= \frac{\pi^2}{16}\cos(\frac{3\pi}{4}) - 1) \end{aligned}$$

Writing these three equations as a linear system gives

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} = \begin{pmatrix} 1.4362 \\ 0.0 \\ -1.4362 \end{pmatrix}.$$

Solving this system gives $U_1 = 0.7181$, $U_2 = 0$ and $U_3 = -0.7181$; the exact solution at these interior nodes is $(0.7071, 0.0, -0.7071)$.

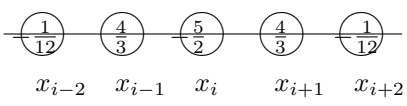
In the numerical examples in the next section we will verify that our results are second order. What if we want a scheme that is more accurate than second order. What can we do? In the previous difference quotient for approximating $u''(x_i)$ we used x_{i-1}, x_i and x_{i+1} so to get a higher order approximation it makes sense that we would have to use more points. The natural thing to do is to include a point to the right of x_{i+1} and one to the left of x_{i-1} so we can keep the terms symmetrically. Thus we want to use a combination of u at the points x_{i-2}, x_{i-1}, x_i, x_{i+1} and x_{i+1}. The correct linear combination is given by

$$u''(x) = \frac{1}{h^2}\Big[-\frac{1}{12}u(x-2h) + \frac{4}{3}u(x-h) - \frac{5}{2}u(x) + \frac{4}{3}u(x+h) - \frac{1}{12}u(x+2h)\Big] + \mathcal{O}(h^4) \tag{6.10}$$

which is fourth order accurate. This can be verified by taking the appropriate linear combination of the Taylor series expansions for $u(x-2h)$, $u(x-h)$, $u(x+h)$ and $u(x+2h)$. Thus for the differential equation $-u''(x) = f(x)$ we have the difference equation

$$\frac{1}{12}U_{i-2} - \frac{4}{3}U_{i-1} + \frac{5}{2}U_i - \frac{4}{3}U_{i+1} + \frac{1}{12}U_{i+2} = h^2 f(x_i)\,.$$

This is called a five point stencil in 1D and is often displayed pictorially as the following.



Before proceeding further, we summarize the finite difference approximations that we have derived along with their accuracy for future reference in Table 6.1. When we move to higher dimensions we will typically just use these approximations but in directions other than x.

$u'(x)$	forward difference difference height	$\frac{u(x+h)-u(x)}{h}$	$\mathcal{O}(h)$
$u'(x)$	backward difference	$\frac{u(x)-u(x-h)}{h}$	$\mathcal{O}(h)$
$u'(x)$	centered difference	$\frac{u(x+h)-u(x-h)}{2h}$	$\mathcal{O}(h^2)$
$u''(x)$	second centered difference	$\frac{u(x+h)-2u(x)+u(x-h)}{h^2}$	$\mathcal{O}(h^2)$
$u''(x)$	five-point stencil (1D)	$\frac{1}{12h^2}\big[-u(x-2h)+16u(x+h)$ $-30u(x)+16u(x-h)-u(x+2h)\big]$	$\mathcal{O}(h^4)$

Table 6.1: Finite difference approximations in one dimension and their order of accuracy.

6.1.1 Numerical results

In this section we look at two specific two point BVPs and demonstrate that we get second order accuracy when we use the three point stencil. Note that in the text we have labeled the grid points starting at x_0 through x_{n+1} because when we have Dirichlet boundary data we then have an $n \times n$ system. Some compilers such as `C` or `Fortran 90` allow the use of an array starting at zero whereas Matlab requires arrays to start at one. Consequently you may have to adjust the indices of the arrays to account for this; for example, the first interior point might be labeled x_2 if you are using Matlab.

We want to calculate the numerical rate of convergence for our simulations as we did for IVPs. However, in this case our solution is a vector rather than a single solution. To calculate the numerical rate we need a single number which represents the error so we use a vector norm. A commonly used norm is the standard Euclidean

norm defined by

$$\|\mathbf{x}\|_2 = \Big[\sum_{i=1}^{n} x_i^2\Big]^{1/2}$$

for a vector in $\mathbf{x} \in \mathbb{R}^n$. Other choices include the maximum norm $\|\cdot\|_\infty$ or the one-norm $\|\cdot\|_1$ defined by

$$\|\mathbf{x}\|_\infty = \max_{1\le i\le n} |x_i| \quad \text{and} \quad \|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|\,.$$

Although these are the standard definitions for the vector norms, when we output the error norm we need to normalize it. For example, suppose you compute an error vector, all of whose components are 0.1. Clearly we expect the Euclidean norm to be 0.1 for a vector of any length but if we compute the norm using the definition above for a vector of length 10 then the result is 0.316 and for a vector of length 100 it is 1. So what we need to do is either normalize by the vector of the exact solution evaluated at the same grid points to give a relative error or use an alternate definition; for example for the Euclidean norm we use

$$\|\mathbf{x}\|_2 = \Big[\frac{1}{n}\sum_{i=1}^{n} x_i^2\Big]^{1/2}$$

which gives the answer of 0.1 for a vector all of whose components are 0.1 no matter what its length.

Before presenting results of our numerical simulations for specific problems we briefly outline a possible structure for a program to solve the two-point Dirichlet BVP on and interval $[a, b]$ using a finite difference approach with a uniform grid. In this outline, we use the notation introduced in this section. First, the user needs to provide the following:

- n, the number of interior grid points (alternately the grid spacing h);
- a, b the right and left endpoints of interval;
- the boundary value at $x = a$ and $x = b$, e.g., $\mathsf{u}_{\mathrm{left}}$ and $\mathsf{u}_{\mathrm{right}}$;
- a routine for the forcing function $f(x)$ and the exact solution, if known.

Then the code can be structured as follows:

- compute $h = (b - a)/(n + 1)$;
- compute grid points $x(i)$, $i = 0, 1, 2, \ldots, n + 1$;
- set up the coefficient matrix and store efficiently; for example, for the three-point stencil the matrix can be stored as two vectors;
- set up the right hand side for all interior points;

- modify the first and last entries of the right hand side to account for inhomogeneous Dirichlet boundary data;
- solve the resulting linear system using an appropriate solver;
- output solution to file for plotting, if desired;
- compute the error vector and output a norm of the error (normalized) if the exact solution is known.

Example 6.3. We return to the homogeneous Dirichlet BVP we solved by hand in the first example of this chapter; recall that the BVP is given by

$$\begin{aligned} -u''(x) &= \pi^2 \sin(\pi x) \quad 0 < x < 1 \\ u(0) &= 0, \quad u(1) = 0 \end{aligned}$$

and has an exact solution of $u(x) = \sin(\pi x)$. For our hand calculation we only solved it using a coarse grid so now we want to refine our mesh and demonstrate that the solution is converging with accuracy $\mathcal{O}(h^2)$. In the table below we give the ℓ_2 norm (i.e., the standard Euclidean norm) of the error normalized by the ℓ_2 error of the exact solution. As can be seen from the table, the numerical rate of convergence is two as predicted by theory.

h	$\frac{\|E\|_2}{\|u\|_2}$	numerical rate
$\frac{1}{4}$	5.03588×10^{-2}	
$\frac{1}{8}$	1.27852×10^{-2}	1.978
$\frac{1}{16}$	3.20864×10^{-3}	1.994
$\frac{1}{32}$	8.02932×10^{-4}	1.999
$\frac{1}{64}$	2.00781×10^{-4}	2.000

Example 6.4. The next example we look at is

$$\begin{aligned} -u''(x) &= -2 \quad 0 < x < 1 \\ u(0) &= 0, \quad u(1) = 0 \end{aligned}$$

whose exact is $u(x) = x^2 - x$. We modify our code to incorporate the new right hand side $f(x) = -2$ and the exact solution. The computations give us the following results.

h	$\frac{\|E\|_2}{\|u\|_2}$ numerical rate
$\frac{1}{4}$	3.1402×10^{-16}
$\frac{1}{8}$	3.1802×10^{-16}

Why are we getting essentially zero for the error whereas in the previous example we got errors of approximately 10^{-2} for these grids? The reason is that our exact solution is a quadratic and the third and higher derivatives of the solution are zero so we should be able to obtain it exactly if we didn't have roundoff. To see this, recall that when we derived the three point approximation to $u''(x)$ we combined Taylor series for $u(x+h)$ and $u(x-h)$ to get

$$u(x+h) + u(x-h) - 2u(x) = h^2u''(x) + 2\frac{h^4}{4!}u''''(x) + \mathcal{O}(h^5)$$

Thus the first term in the Taylor series that didn't cancel is $\mathcal{O}(h^4)$ and for this example, it is zero so the approximation is exact.

6.1.2 Systems

Suppose now that we have two coupled BVPs in one dimension, e.g.,

$$\begin{array}{rcll} -u''(x) + v(x) & = & f(x) & a < x < b \\ -v''(x) + u(x) & = & g(x) & a < x < b \\ u(a) = 0 \quad u(b) & = & 0 & \\ v(a) = 0 \quad v(b) & = & 0\,. & \end{array}$$

We use the three point stencil to approximate each equation to get the following equations for $i = 1, 2, \ldots, n$ using the discretization $x_0 = a$, $x_i = x_{i-1} + h$, and $x_{n+1} = b$

$$\begin{array}{rcl} -U_{i-1} + 2U_i - U_{i+1} + V_i & = & f(x_i) \\ -V_{i-1} + 2V_i - V_{i+1} + U_i & = & g(x_i) \end{array}$$

for $i = 1, 2, \ldots, n$. So at grid point (or node) x_i we have two unknowns U_i and V_i. This means we have a choice of how we want to number the unknowns. For example, we could number all of the U_i, $i = 1, \ldots, n$ and then the V_i or we could mix them up, e.g., U_1, V_1, U_2, V_2, ..., U_n, V_n. Now we will get the same solution either way but one leads to a matrix problem which is easier to solve.

First we will look at the resulting system if our solution vector is numbered as

$$(U_1, U_2, \cdots, U_n, V_1, V_2, \cdots, V_n)^T$$

In this case we write all the equations for U in the first half of the matrix and then all the equations for V in the second half. For example,

$$2U_1 - U_2 + V_1 \quad = \quad f(x_1)$$

$$\begin{aligned}
-U_1 + 2U_2 - U_3 + V_2 &= f(x_2) \\
&\vdots \\
-U_{n-1} + 2U_n + V_n &= f(x_n) \\
2V_1 - V_2 + U_1 &= g(x_1) \\
-V_1 + 2V_2 - V_3 + U_2 &= g(x_2) \\
&\vdots \\
-V_{n-1} + 2V_n + U_n &= g(x_n)\,.
\end{aligned}$$

We have the coefficient matrix in block form

$$A = \begin{pmatrix} S & I \\ I & S \end{pmatrix}$$

where I is the $n \times n$ identity matrix and

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

This is a block matrix and there are ways to efficiently solve it but if one was using a library package, then typically a banded solver would be used and the matrix would be $2n \times 2n$ with a total bandwidth of $n + 1$.

Another choice for numbering the unknowns is to have

$$(U_1, V_1, U_2, V_2, \cdots, U_n, V_n)^T\,.$$

In this case we write an equation for U_i and then for V_i alternating in this way for $i = 1, 2, \ldots, n$. For example,

$$\begin{aligned}
2U_1 - U_2 + V_1 &= f(x_1) \\
2V_1 - V_2 + U_1 &= g(x_1) \\
-U_1 + 2U_2 - U_3 + V_2 &= f(x_2) \\
-V_1 + 2V_2 - V_3 + U_2 &= g(x_2) \\
&\vdots \\
-U_{n-1} + 2U_n + V_n &= f(x_n) \\
-V_{n-1} + 2V_n + U_n &= g(x_n)\,.
\end{aligned}$$

In this case the coefficient matrix is

$$A = \begin{pmatrix} 2 & 1 & -1 & & & & & & \\ 1 & 2 & 0 & -1 & & & & & \\ -1 & 0 & 2 & 1 & -1 & & & & \\ & -1 & 1 & 2 & 0 & -1 & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & -1 & 0 & 2 & 1 & -1 & & \\ & & & -1 & 1 & 2 & 0 & -1 & \\ & & & & -1 & 0 & 2 & 1 & \\ & & & & & -1 & 1 & 2 & \end{pmatrix}.$$

This is a $2n \times 2n$ matrix but the bandwidth is only five so if we use a banded solver the procedure of alternating the unknowns is more efficient. One should be aware of how the unknowns are numbered because this can yield different matrices. Of course the resulting linear systems are equivalent but one system may be easier to solve than another.

6.2 The Poisson equation

We now want to use our knowledge gained from approximating $u''(x)$ in one dimension to approximate Δu in two or three dimensions. The first step of the discretization process is to overlay the domain with a grid. For simplicity we will begin with the unit square $(0,1) \times (0,1)$ and basically we will take the grid defined by (6.7) and use it in both the x and y directions. For now we set $\Delta x = \Delta y = h = 1/(n+1)$ and set

$$\begin{aligned} x_0 &= 0,\ x_1 = x_0 + h, \ldots,\ x_i = x_{i-1} + h, \ldots,\ x_{N+1} = x_n + h = 1 \\ y_0 &= 0,\ y_1 = y_0 + h, \ldots,\ y_j = y_{j-1} + h, \ldots,\ y_{n+1} = y_n + h = 1\,. \end{aligned}$$

When we discretized our two-point BVP we first used a three point stencil in the x direction to approximate $u''(x)$. We can easily extend this to two dimensions by differencing in both the x and y directions to obtain a difference equation for the Poisson equation in two dimensions. Suppose we want to solve

$$\begin{aligned} -\Delta u &= -\Big(u_{xx} + u_{yy}\Big) = f(x,y) \quad \forall (x,y) \in (0,1) \times (0,1) \\ u &= 0 \quad \text{on } \Gamma \end{aligned}$$

with a finite difference scheme that is second order in x and y. We discretize the domain as in Figure 6.1. We let $U_{i,j} \approx u(x_i, y_j)$ for $i, j = 0, 1, 2, \ldots, n+1$. Clearly $U_{0,j} = U_{n+1,j} = 0$, $j = 0, 1, \ldots, n+1$ and $U_{i,0} = U_{i,n+1} = 0$ for $i = 0, 1, 2, \ldots, n+1$ from the boundary conditions; these are just the values at the nodes denoted by a solid circle in Figure 6.1. To write our difference equation we simply use the second centered difference in x (holding y fixed)

$$u_{xx}(x_i, y_j) \approx \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h^2}$$

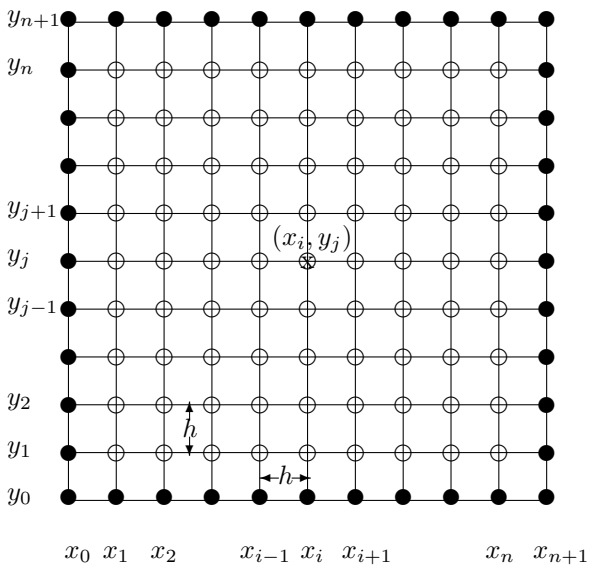


Figure 6.1: Uniform Cartesian grid on a unit square with a total of $(n+2)^2$ nodes. The boundary nodes are denoted by a solid circle. For a Dirichlet BVP we only have unknowns at the interior nodes which are marked by an open circle.

and then use the analogous difference quotient in the y direction

$$u_{yy}(x_i, y_j) \approx \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h^2}\,.$$

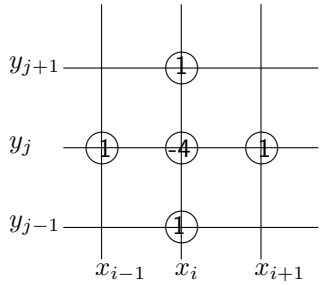
Combining these results we have the finite difference equation for $-\Delta u(x,y) = f(x,y)$ at the point (x_i, y_j)

$$\frac{-U_{i-1,j} + 2U_{i,j} - U_{i+1,j}}{h^2} + \frac{-U_{i,j-1} + 2U_{i,j} - U_{i,j+1}}{h^2} = f(x_i, y_j)\,.$$

Multiplying by h^2 and combining terms yields

$$-U_{i-1,j} + 4U_{i,j} - U_{i+1,j} - U_{i,j-1} - U_{i,j+1} = h^2 f(x_i, y_j) \qquad i,j = 1,2,\ldots,n\,. \tag{6.11}$$

This is called the **five point stencil** for the Laplacian in two dimensions because it uses the five points (x_{i-1}, y_j), (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j-1}), and (x_i, y_{j+1}); it is illustrated schematically in the diagram below.



In one dimension the resulting matrix had a bandwidth of three so we want to see what structure the matrix has in two dimensions. We choose to number our unknowns across each horizontal row. It really doesn't matter in this case because our domain is a square box. We have the unknown vector

$$\Big(U_{1,1}, U_{2,1}, \cdots, U_{n,1} \Big| \, U_{1,2}, U_{2,2}, \cdots, U_{n,2} \Big| \, \cdots \Big| \, U_{1,n}, U_{2,n}, \cdots, U_{n,n}\Big)^T$$

because we only have unknowns at interior nodes. The first n values are the unknowns at the interior nodes on the first horizontal grid line $y = h$, the second set of n values are the unknowns at the interior nodes on the second horizontal grid line $y = 2h$, etc. To see the structure of the matrix we write the first few equations and then we can easily see the structure; for $j = 1, 2$ we have the equations

$$\begin{aligned}
4U_{1,1} - U_{2,1} - U_{1,2} &= h^2 f(x_1, y_1) \\
-U_{1,1} + 4U_{2,1} - U_{3,1} - U_{2,2} &= h^2 f(x_2, y_1) \\
&\vdots \\
-U_{n-1,1} + 4U_{n,1} - U_{n,2} &= h^2 f(x_n, y_1) \\
\\
4U_{1,2} - U_{2,2} - U_{1,3} - U_{1,1} &= h^2 f(x_1, y_2) \\
-U_{1,2} + 4U_{2,2} - U_{3,2} - U_{2,3} - U_{2,1} &= h^2 f(x_2, y_2) \\
&\vdots \\
-U_{n-1,2} + 4U_{n,2} - U_{n,3} - U_{n,1} &= h^2 f(x_n, y_2),
\end{aligned}$$

where we have used the homogeneous boundary conditions. Thus, we see that the matrix has the block structure

$$A = \begin{pmatrix}
S & -I & & & & & \\
-I & S & -I & & & & \\
& -I & S & -I & & & \\
& & \ddots & \ddots & \ddots & & \\
& & & & -I & S & -I \\
& & & & & -I & S
\end{pmatrix}$$

where S is an $n \times n$ matrix defined by

$$S = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}$$

and I is the $n \times n$ identity matrix. So the coefficient matrix is now $n^2 \times n^2$ whereas in one dimension it was $n \times n$. As in one dimension it is a symmetric positive definite matrix. There are efficient ways to solve this block tridiagonal matrix such as using a block tridiagonal solver, a banded Cholesky solver or an iterative solver.

6.2.1 Handling boundary conditions

We have seen that when we have homogeneous Dirichlet boundary data then we have an unknown at each interior grid point because the solution is zero at all boundary points; even if the Dirichlet boundary data is inhomogeneous we will still only have unknowns at the interior nodes. We will often have BVPs with Neumann boundary conditions or even Robin boundary conditions which specify a combination of the unknown and its derivative. So we need to figure out how to impose different conditions. It turns out that it is easy to impose Dirichlet boundary conditions with finite difference methods but imposing a derivative boundary condition introduces some difficulty.

If we specify $u(x, y) = g(x, y)$, $g \neq 0$, on the boundary of our domain then we still only have unknowns at the interior grid points. There are two common ways that inhomogeneous boundary conditions are satisfied. We saw in an example that one way was to just modify the right hand side vector; the other way is to add an equation for each boundary node and set it equal to g evaluated at that boundary node. We will look at both approaches.

First suppose we are using the five point stencil to approximate the Poisson equation with inhomogeneous Dirichlet boundary data $u = g$ on $\Gamma = (0, 1) \times (0, 1)$. Suppose that we number our nodes along each horizontal line and we write the difference equation at the first interior horizontal line, i.e., at (x_i, y_1). At the first interior grid point (x_1, y_1) we have

$$-U_{0,1} + 4U_{1,1} - U_{2,1} - U_{1,0} - U_{1,2} = h^2 f(x_1, y_1)\,.$$

Now two of these terms are known due to the boundary condition; in fact any term where $i = 0$ or $j = 0$ is known. We have $U_{0,1} = g(x_0, y_1)$ which is a boundary node on the left side of the domain and $U_{1,0} = g(x_1, y_0)$ which is a boundary node on the bottom of the domain. We can move these terms to the right-hand side to get

$$4U_{1,1} - U_{2,1} - U_{1,2} = h^2 f(x_1, y_1) + g(x_0, y_1) + g(x_1, y_0)\,.$$

For $i = 2, 3, \ldots, n-1$ and $j = 1$ we will only have one boundary term to move to the right hand side; at $i = n$ we will have a point from the right and bottom

boundaries. Writing the difference equation at $j = 2$ and $i = 2, 3, 4, \ldots, n-1$ we have

$$-U_{i-1,2} + 4U_{i,2} - U_{i+1,2} - U(i,1) - U_{i,3} = h^2 f(x_i, y_2)$$

and there are no boundary terms. At $i = 1$ or $i = n$ there will be two boundary terms, $U_{0,2}$ on the right and $U_{n+1,2}$ on the left boundary. So for each difference equation that we write that contains a known boundary term we move it to the right hand side. In this approach, the matrix is not modified.

A second approach is to add an equation for each boundary node and "pretend" to have an unknown at every grid point. So if we have a Cartesian grid for $(0,1) \times (0,1)$ with $n+2$ points on a side we have a total of $2(N+2) + 2(N) = 4N + 4$ boundary nodes. For each boundary node we add an equation; for example along the bottom of the domain we have

$$U_{i,0} = g(x_i, y_0), \quad i = 0, 1, \ldots, n+1\,.$$

This adds a diagonal entry to the matrix so it does not affect the bandwidth of the matrix but of course it increases the size.

Imposing derivative boundary conditions in the context of finite difference methods is often viewed as a shortcoming of the method because they have to be implemented with care and often require the addition of "fictitious" grid points. Remember that when we impose a Neumann or Robin boundary condition the unknown itself is not given at the boundary so we have to solve for it there. Suppose for example, that we have a domain $(0,1) \times (0,1)$ and wish to impose a flux condition which is represented by a Neumann boundary condition. To be concrete, we assume $\partial u / \partial \hat{\mathbf{n}} = g(x,y)$ along the sides $x = 0$ and $x = 1$. Because the outer normal is $\pm\hat{\mathbf{i}}$ this flux condition is just $\pm u_x = g$. This means that we have to replace u_x with a difference quotient and write this equation at the boundary node. We can use a one-sided difference such as

$$u_x(x_0, y_j) = \frac{U_{1,j} - U_{0,j}}{h} = g(x_0, y_j)$$

at the left boundary. The problem with this is that it is a first order accurate approximation whereas in the interior we are using a second order accurate approximation. We have seen that the centered difference approximation

$$u_x(x_i, y_j) \approx \frac{u(x_{i+1}, y_j) - u(x_{i-1}, y_j)}{2h}$$

is second order accurate. But if we write this at the point (x_0, y_j), then there is no grid point to its left because (x_0, y_j) lies on the boundary. To see how to implement this centered difference approximation for the Neumann boundary condition first consider the simplified case where $\partial u / \partial \hat{\mathbf{n}} = 0$. The finite difference equation at the point (x_0, y_j) is

$$U_{-1,j} + 4U_{0,j} - U_{1,j} - U_{0,j+1} - U_{0,j-1} = h^2 f(x_0, y_j)$$

and the centered difference approximation to $-u_x = 0$ at (x_0, y_j) is

$$\frac{U_{1,j} - U_{-1,j}}{2h} = 0 \quad \Rightarrow \quad U_{-1,j} = U_{1,j}\,.$$

We then substitute this into the difference equation at (x_0, y_j) to get

$$U_{1,j} + 4U_{0,j} - U_{1,j} - U_{0,j+1} - U_{0,j-1} = h^2 f(x_0, y_j)$$

or

$$4U_{0,j} - U_{0,j+1} - U_{0,j-1} = h^2 f(x_0, y_j)\,.$$

So we need to modify all the equations written at $x = 0$ or $x = 1$ where the Neumann boundary condition is imposed.

If the Neumann boundary condition is inhomogeneous we have

$$\frac{U_{1,j} - U_{-1,j}}{2h} = g(x_0, y_j) \quad \Rightarrow \quad U_{-1,j} = U_{1,j} - 2hg(x_0, y_j)$$

and we substitute this into the difference equation for $U_{-1,j}$. Of course if the domain is not a rectangle then the procedure is more complicated because the Neumann boundary condition $\partial u/\partial \hat{\mathbf{n}}$ does not reduce to u_x or u_y.

6.3 Summary

In this chapter we saw how we could obtain approximations to the differential equation by replacing derivatives with finite difference quotients; so the method is called the finite difference method. These difference quotients are easy to derive using Taylor series. The finite difference method has the advantage that it is easy to understand and straightforward to program. Although one may claim that finite differences go back to the time of Newton, it was actually in the 1920's that they were first used to solve differential equations in mathematical physics. So historically the finite difference method was the first method used to solve PDEs extensively.

The main impetus for developing additional methods for approximating the solution of PDEs was the desire to compute on more complex domains. For example, if one has a polygonal domain in $\mathbb{R}^2$ it is much easier to cover it with triangles than with rectangles which the finite difference method uses. Also, if you recall how we derived the second centered difference we combined the Taylor series for $u(x + h)$ and $u(x - h)$ and several terms cancelled to get second order accuracy. However, this assumed a uniform grid. So if we have a nonuniform grid we do not have second order accuracy. There have been methods derived for nonuniform grids but they are not straightforward extensions of the uniform case. Another issue with finite difference approximations is that it can be more difficult to enforce derivative boundary conditions or interface conditions which are conditions between two regions where different equations hold. For these reasons, we want to look at additional methods to solve PDEs.