Chapter 4

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 4.1.

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.

4.1 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

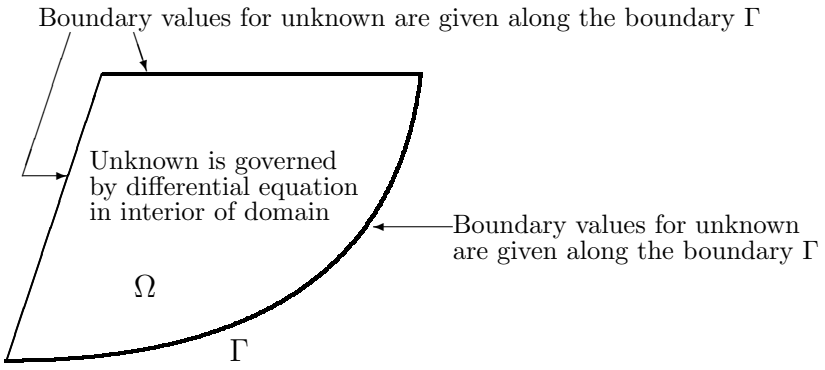


Figure 4.1: Sample domain of a BVP with a differential equation governing the unknown in the interior of the domain and boundary values specified on all boundaries.

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.

4.2 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

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x) \quad a < x < b\,, \tag{4.1}$$

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

$$y''(x) = f(x, y, y') \quad a < x < b\,. \tag{4.2}$$

Before we attempt to approximate the solution to a given BVP we want to know that the continuous problem has a unique solution. For (4.1) 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 (4.2) 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

$$\begin{aligned} &\frac{d^2}{dx^2}\left(r(x)\frac{d^2u}{dx^2}\right) - \frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x) \quad a < x < b \\ &u(0) = u(1) = 0 \qquad u''(0) = u''(1) = 0\,. \end{aligned} \tag{4.3}$$

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

4.3 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 $-u''(x) = f(x)$.

4.3.1 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

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

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 $\mathbf{n}$ is the unit vector giving the direction then the derivative and the notation we use is

$$\frac{\partial u}{\partial \mathbf{n}} \equiv \nabla u \cdot \mathbf{n} . \tag{4.4}$$

Note that if $\mathbf{n} = (1, 0)^T$, 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 $\mathbf{n} = (0, 1)^T$ 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 $\partial u/\partial \mathbf{n}$ 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

$$\Delta u \equiv \nabla \cdot \nabla u = u_{xx} + u_{yy} + u_{zz} \tag{4.5}$$

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

$$au_{\xi\xi} + bu_{\xi\eta} + cu_{\eta\eta} + du_{\xi} + eu_{\eta} + gu = f(\xi, \eta) \tag{4.6}$$

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 (4.6) 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

$$\begin{array}{ll} \text{elliptic} & \text{if } b^2 - 4ac < 0 \quad \text{for all points in the domain} \\ \text{parabolic} & \text{if } b^2 - 4ac = 0 \quad \text{for all points in the domain} \\ \text{hyperbolic} & \text{if } b^2 - 4ac > 0 \quad \text{for all points in the domain} \end{array}$$

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 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.

4.3.2 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 (4.5). The Poisson equation in three dimensions is

$$-\Delta u = f(x, y, z) \quad (x, y, z) \in \Omega \tag{4.7}$$

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 (4.7) 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., we specify $\dfrac{\partial u}{\partial \mathbf{n}}$ 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 4.2.

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

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

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

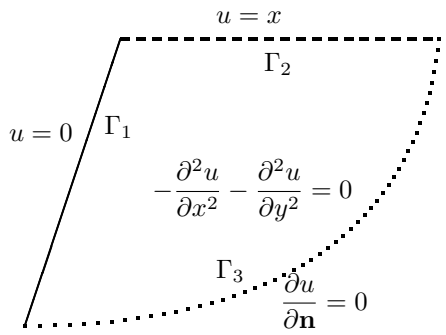


Figure 4.2: A sample mixed BVP for $u(x, y)$ where $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$. Here Dirichlet boundary conditions are specified on $\Gamma_1 \cup \Gamma_2$ and a Neumann boundary condition is specified on Γ_3. The notation $\dfrac{\partial u}{\partial \mathbf{n}}$ represents $\nabla u \cdot \mathbf{n}$ where $\mathbf{n}$ is the unit outer normal to the given boundary.

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 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

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

4.4 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).

4.5 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.

4.5.1 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

$$AB \neq BA.$$

4.5.2 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

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

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

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

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

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

where

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

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

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

Example 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

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

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

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

4.5.3 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 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 4.1. *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 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 4.2. *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 4.3. *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 4.1 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. 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 4.1: 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!

4.5.4 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 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 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 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 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.

4.5.5 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 4.1. *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 5

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.

5.1 The two-point BVP

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

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

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

$$0 < p_{\min} \le p \le p_{\max} \quad \text{and} \quad q_{\min} = 0 \le q(x) \le q_{\max}\,. \tag{5.2}$$

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 (5.2). 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 theforward 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

$$u(x+h) = u(x) + h\,u'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u'''(x) + \mathcal{O}(h^4)\,. \tag{5.3}$$

Now we want an approximation for $u''(x)$ but if we solve for it in (5.3) then we still have the $u'(x)$ term. However, if we add (5.3) to the expansion for $u(x - h)$ given by

$$u(x-h) = u(x) - h\,u'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + \mathcal{O}(h^4) \tag{5.4}$$

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.

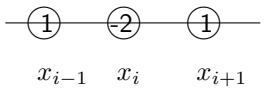
Second centered difference: $$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2) \tag{5.5}$$

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.



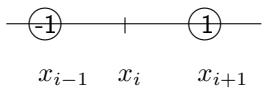
Now that we have an approximation for $u''(x)$ we need to decide which difference quotient to use to approximate the first derivative. Because our approximation to $u''(x)$ is second order, we would like to use the same accuracy for approximating the first derivative. However, both the forward and backward 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 (5.3) and (5.4) 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: $$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + \mathcal{O}(h^2) \tag{5.6}$$

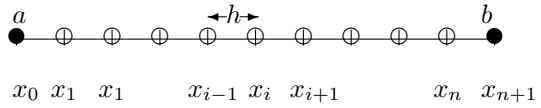
and thus if we use this difference quotient both approximations are second order accurate. It is described by the stencil



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

$$x_0 = a, x_1 = x_0 + h, \ldots, x_i = x_{i-1} + h, \ldots, x_{n+1} = x_n + h = b \tag{5.7}$$

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 (5.1) at the point x_i using the two difference quotients (5.5) and (5.6). If we let $U_i \approx u(x_i)$ then our finite difference equation at the node x_i is

$$p(x_i)\left(\frac{-U_{i+1} + 2U_i - U_{i-1}}{h^2}\right) - p'(x_i)\left(\frac{U_{i+1} - U_{i-1}}{2h}\right) + q(x_i)U_i = f(x_i)\,. \tag{5.8}$$

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 (5.8) 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

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & & 0 \\ -1 & 2 & -1 & 0 & \cdots & & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ 0 & \cdots & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} \tag{5.9}$$

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 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 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 § 5.2.1. 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

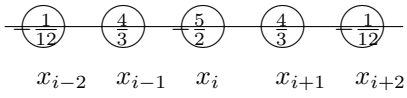
the right of x_{i+1} and one to the left of x_{i-1} so we can keep the terms symmetrically. Thus we want to use a combination of u at the points x_{i-2}, x_{i-1}, x_i, x_{i+1} and x_{i+1}. The correct linear combination is given by

$$u''(x) = \frac{1}{h^2}\Big[-\frac{1}{12}u(x-2h)+\frac{4}{3}u(x-h)-\frac{5}{2}u(x)+\frac{4}{3}u(x+h)-\frac{1}{12}u(x+2h)\Big]+\mathcal{O}(h^4) \tag{5.10}$$

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 5.1. When we move to higher dimensions we will typically just use these approximations but in directions other than x.

5.1.1 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

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

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

Table 5.1: Finite difference approximations in one dimension and their order of accuracy.

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., u_{left} and $\mathsf{u}_{\text{right}}$;
- 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 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 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

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

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.

5.1.2 Systems

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

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

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 $x_{n+1} = b$

$$-U_{i-1} + 2U_i - U_{i+1} + V_i = f(x_i)$$

$$-V_{i-1} + 2V_i - V_{i+1} + U_i \quad = \quad g(x_i)$$

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,

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

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,

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

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

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.

5.2 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 (5.7) 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

$$-\Delta u = -\Big(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\Big) = f(x,y) \quad \forall (x,y) \in (0,1)\times(0,1)$$

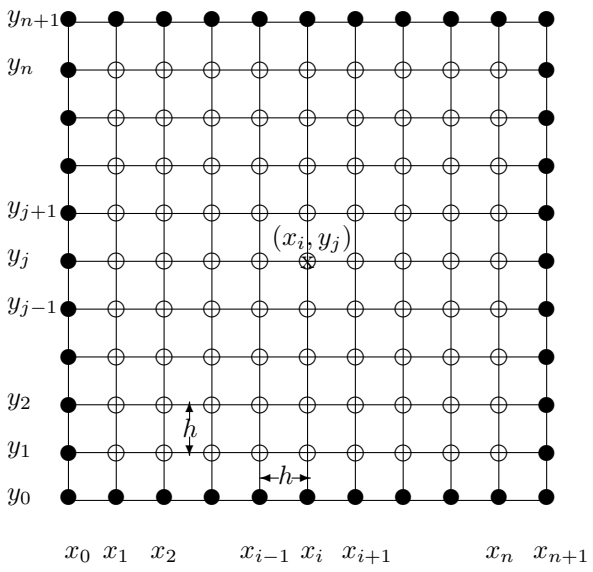


Figure 5.1: Uniform Cartesian grid on a unit square with a total of $(n+2)^2$ nodes. The boundary nodes are denoted by a solid circle. For a Dirichlet BVP we only have unknowns at the interior nodes which are marked by an open circle.

$$u = 0 \quad \text{on } \Gamma$$

with a finite difference scheme that is second order in x and y. We discretize the domain as in Figure 5.1. 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 5.1. To write our difference equation we simply use the second centered difference in x (holding y fixed)

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

and then use the analogous difference quotient in the y direction

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

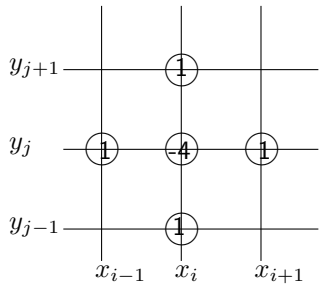
Combining these results we have the finite difference equation for $-\Delta u(x, y) = f(x, y)$ at the point (x_i, y_j)

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

Multiplying by h^2 and combining terms yields

$$-U_{i-1,j} + 4U_{i,j} - U_{i+1,j} - U_{i,j-1} - U_{i,j+1} = h^2 f(x_i, y_j) \qquad i,j = 1,2,\ldots,n\,. \tag{5.11}$$

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.

5.2.1 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 $\partial u / \partial \mathbf{n} = g(x, y)$ along the sides $x = 0$ and $x = 1$. Because the outer normal is $\pm\mathbf{i}$ 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 $\partial u/\partial \mathbf{n} = 0$. 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 $\partial u/\partial \mathbf{n}$ does not reduce to u_x or u_y.

5.3 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.

Chapter 6

Method of Mean Weighted Residuals

The finite element method (FEM) and the finite difference method (FDM) are the two most commonly used methods for approximating the solution of PDEs. Before we introduce the FEM we consider a technique called the *mean weighted residuals* (MWR) method. This approach to solving differential equations existed before the FEM but, moreover, the FEM can be viewed as a particular MWR method. In addition, there are several other methods of interest which are MWR methods. We will introduce the general concept of the FEM in this chapter but will not delve into the details until the following two chapters.

The MWR method is not a specific method but rather it is a general framework under which different methods can be described. Theoretically there are an infinite number of methods that can be described under this framework.

We begin this chapter by giving a general description of the MWR. Then we look at three methods which can be viewed as an MWR method. The first is a least squares approach; you have probably encountered the discrete least squares method when you wanted to fit a polynomial to some discrete data in a least squares sense. The second approach is the Collocation Method and the final one we consider is the Galerkin Method. The FEM is a specific Galerkin method.

6.1 Overview

Suppose we have a set of n linearly independent[1] functions $\{\phi_i\}$, $i = 1, \ldots, n$. We define the space $V = \text{span}(\{\phi_i\}_{i=1}^n)$; that is, V is all linear combinations of these functions. Because the ϕ_i are linearly independent they form a *basis* for V. The basis functions ϕ_i could be polynomials, sines and cosines, wavelets or a myriad of

[1] A set of vectors $\{v_i\}_{i=1}^n$ are linearly independent provided $\sum_{i=1}^n c_i v_i = 0$ implies $c_i = 0$ for all i; otherwise they are called linearly dependent.

other choices. At this point we are not saying what we choose for the vectors ϕ_i but rather assuming that we have them "in hand".

For concreteness let's assume we are solving the Poisson equation $-\Delta u = f$ in two dimensions with homogeneous Dirichlet boundary conditions. We seek our approximation, say u^h to u, the solution of the Poisson equation, as a linear combination of the basis vectors $\phi_i(x, y)$, i.e.,

$$u^h = \sum_{i=1}^{n} c_i \phi_i(x, y)\,.$$

Once we specify the basis functions ϕ_i then our goal is to find the coefficients c_i, $i = 1, 2, \ldots, n$. Now these basis vectors are not discrete functions but typically continuous functions which can be differentiated like polynomials or trigonometric functions. If we plug our approximationu^h into the original PDE we will not get zero because it is only an approximation to the exact solution; we call this remainder the **residual** R. For the Poisson equation we have

$$R(x, y) = -\Delta u^h - f(x, y) \neq 0\,.$$

The goal is to try and force the residual to zero in some way. We don't just want the residual to be small in one part of the domain and large in another so we must enforce a condition over the entire domain. The easiest way to do this is with an integral over the domain. Now the method is called mean weighted residuals so we know where the residual part comes in and the mean just refers to the integral over the domain but what about the "weighted" part? It turns out that this part is what distinguishes different weighted residual methods. We don't simply take the integral of R over the domain we "weight" it against $\mathit{weight\ functions}$. For example, in $\mathbb{R}^2$ we require

$$\int_\Omega R(x, y) W_i(x, y)\, d\Omega = 0 \quad i = 1, 2, \ldots, n\,.$$

Note that there are n weight functions which is the same as the number of unknowns in the expansion $u^h = \sum_{i=1}^{n} c_i \phi_i$; this is good because if we end up with a linear system to solve it will have n equations (one for each weight function) and n unknowns so it will be square.

For our example of the Poisson equation we substitute the expression for the residual into the above integral equation to get

$$\int_\Omega \left[(-\Delta u^h - f) W_i\right] d\Omega = 0 \Rightarrow -\int_\Omega \Delta u^h W_i\, d\Omega = \int_\Omega f W_i\, d\Omega \quad i = 1, 2, \ldots, n\,.$$

Remember that our goal is to find $u^h = \sum_{j=1}^{n} c_j \phi_j$ which means we have to find the scalar coefficients c_j. We substitute this expression for u^h into the equation above to get

$$-\int_\Omega \Delta\Big(\sum_{j=1}^{n} c_j \phi_j\Big) W_i\, d\Omega = \int_\Omega f W_i\, d\Omega \quad i = 1, 2, \ldots, n$$

or

$$-\sum_{j=1}^{n} c_j \int_\Omega \Delta\phi_j W_i \, d\Omega = \int_\Omega f W_i \, d\Omega \quad i = 1, 2, \ldots, n \,.$$

If we know the basis functions $\phi_j(x, y)$, $i = 1, \ldots, n$ and the weight functions $W_i(x, y)$, $i = 1, \ldots, n$ then this equation reduces to the $n \times n$ matrix problem

$$A\mathbf{c} = \mathbf{f}$$

where

$$A_{ij} = -\int_\Omega \Delta\phi_j W_i \, d\Omega \qquad (\mathbf{f})_i = \int_\Omega f W_i \, d\Omega \,.$$

Of course we have added the complication that entries in the matrix and right hand side are integrals but these can always be approximated by a numerical quadrature rule.

Different choices of the weighting functions W_i result in different methods. We will look at the three most common choices which are the Least Squares method, the Collocation method, and the Galerkin[2] method. The FEM is a Galerkin method. Before looking at these methods we consider a concrete problem which we will approximate by each of the three methods.

Example 1. We consider the two point BVP

$$-u''(x) + u(x) = -3e^{2x} - xe^2 \quad 0 < x < 1, \quad u(0) = 1, u(1) = 0$$

whose exact solution is $u(x) = e^{2x} - xe^2$. We choose to approximate the solution as a linear combination of polynomials of degree two or less in one independent variable. The dimension of the space of all polynomials of degree two or less is three and a basis is

$$\phi_1 = 1, \quad \phi_2 = x, \qquad \phi_3 = x^2 \,.$$

So our approximate solution is

$$u^h = \sum_{i=1}^{3} c_i \phi_i(x) = c_1 + c_2 x + c_3 x^2$$

and our goal is to determine c_1, c_2 and c_3. When we satisfy the Dirichlet boundary conditions we get

$$u(0) = c_1 = 1 \qquad \text{and} \quad u(1) = 1 + c_2 + c_3 = 0 \Rightarrow c_3 = -(1 + c_2) \equiv c \,.$$

Thus $c_2 = -(1 + c)$ and we have $u^h = 1 - (1 + c)x + cx^2$, i.e., we only have one unknown. The MWR method requires us to calculate the residual $R(x)$ so for this BVP we have

$$R(x) = -\frac{d^2}{dx^2} u^h + u^h + 3e^{2x} + xe^2 \,.$$

Substituting $u^h = 1 - (1 + c)x + cx^2$ and $(u^h)''(x) = 2$ gives

$$R(x) = -2c + (1 - (1 + c)x + cx^2) + 3e^{2x} + xe^2 = c\big(-2 - x + x^2\big) + 1 - x + 3e^{2x} + xe^2 \,.$$

For this problem the MWR method requires

$$\int_0^1 \Big[c\big(-2 - x + x^2\big) + 1 - x + 3e^{2x} + xe^2\Big] W_i \, dx = 0 \quad \text{for } i = 1 \,.$$

We will return to this problem and solve it as we explore different MWR methods and their choice for the weighting functions W_i.

---

[2] Named after the Russian mathematician and engineer Boris Galerkin (1871-1945).

6.2 Least Squares Method

We begin this section by recalling the discrete least squares method and seeing that we can generalize this to approximating functions in a least squares sense. We then use these ideas in the MWR method and see what choice of weight functions this leads to.

Typically one first encounters the method of least squares when there is a set of data and we want to fit this data with a line or a higher degree polynomial in a least squares sense. This is called *discrete least squares* because you have discrete data. For example, assume that we have a set of discrete data points (x_i, y_i), $i = 1, 2, \ldots, m$ and we want to fit a line $a_0 x + a_1$ to the data in a least squares sense. If we have only two points, i.e., $m = 2$, then we can find the line that passes through the two points but typically we have $m > 2$. In this case you want to minimize the ℓ_2-norm (Euclidean length) of the distance between the line evaluated at each x_i and the data y_i. Typically we minimize the square of this distance, i.e.,

$$\text{minimize } \rho \text{ where} \quad \rho(a_0, a_1) = \sum_{i=1}^{m} |y_i - (a_0 x_i + a_1)|^2 .$$

The coefficients a_0, a_1 can be found by solving a 2×2 system called the normal equations.

We can also approximate *functions* in a least squares sense. Remember that functions are defined over a domain and not just at a set of discrete points. For concreteness assume that $f(x)$ is a continuous function on $[a, b]$ which we want to approximate by a polynomial $p_n(x)$ of degree n in the least squares sense. For the discrete least squares method we used the sum of the difference squared so now we much replace the sum with an integral because we have a function. So in the continuous least squares method we want to

$$\text{minimize } \mathcal{F} \text{ where} \quad \mathcal{F} = \int_a^b \big(f(x) - p_n(x)\big)^2 \, dx .$$

For simplicity we assume $n = 1$ and $p_1(x) = a_0 x + a_1$; substituting this into our function to minimize we have

$$\mathcal{F}(a_0, a_1) = \int_a^b \big(f(x) - (a_0 + a_1 x)\big)^2 \, dx .$$

We know from calculus that to minimize a function $g(x, y)$ we find the critical points by setting g_x and g_y equal to zero (and check any boundary values). In our case the variables are a_0 and a_1 so we have

$$\frac{\partial \mathcal{F}}{\partial a_0} = 0 \quad \text{and} \quad \frac{\partial \mathcal{F}}{\partial a_1} = 0$$

which in our case yields

$$\frac{\partial \mathcal{F}}{\partial a_0} = -2 \int_a^b \big(f(x) - (a_0 + a_1 x)\big) \, dx = 0 \Rightarrow \int_a^b (a_0 + a_1 x) \, dx = \int_a^b f(x) \, dx$$

and

$$\frac{\partial \mathcal{F}}{\partial a_1} = -2\int_a^b x\big(f(x)-(a_0+a_1x)\big)\,dx = 0 \Rightarrow \int_a^b (a_0x+a_1x^2)\,dx = \int_a^b xf(x)\,dx\,.$$

The function $f(x)$ is given to us and we know the interval $[a,b]$ where we are approximating $f(x)$ so we perform the integration (either exactly or with numerical quadrature) and have two equations for a_0 and a_1 to satisfy.

Now we want to see how this approach of least squares can be applied in the MWR method. The basic idea is to minimize the square of the residual integrated over the domain, i.e.,

$$\text{minimize } \mathcal{F} \text{ where } \quad \mathcal{F} = \int_\Omega R^2(x,y)\,d\Omega\,.$$

As in the continuous least squares approach we take the partials of $\mathcal{F}$ with respect to the unknown parameters and set to zero. What are our unknowns in this equation? Recall that $u^h = \sum c_i\phi_i$ and our residual is the result of plugging u^h into the differential equation. Consequently our unknowns are the c_i, $i = 1, 2, \ldots, n$ so we differentiate $\mathcal{F}$ with respect to each c_i and set it to zero

$$\frac{\partial \mathcal{F}}{\partial c_i} = 0 \quad \forall i = 1, 2, \ldots, n\,.$$

Using the definition of $\mathcal{F}$ we have

$$\frac{\partial \mathcal{F}}{\partial c_i} = 2\int_\Omega R(x,y)\frac{\partial R}{\partial c_i}\,d\Omega = 0 \Rightarrow \int_\Omega R(x,y)\frac{\partial R}{\partial c_i}\,d\Omega = 0\,.$$

Now we want to compare this with our general equation

$$\int_\Omega R(x,y)W_i(x,y)\,d\Omega = 0 \quad i = 1, 2, \ldots, n$$

for the MWR method to see what choice of weighting function the Least Squares method makes. Comparing the equations we see that the weighting function W_i is chosen as

$$W_i = \frac{\partial R}{\partial c_i}\,.$$

That is, each W_i is the derivative of the residual with respect to the unknown coefficient c_i. Note once again that there are n unknown coefficients and so there are n first partials of R so we have n weighting functions.

Example 2. We return to our example in § 6.1 and solve it using the choice of weighting function dictated by the Least Squares method. Recall that after satisfying the boundary conditions we had only one unknown c so the equation we have to solve is

$$\int_0^1 RW_1\,dx = \int_0^1 \Big[c\big(-2-x+x^2\big)+1-x+3e^{2x}+xe^2\Big]W_1\,dx = 0\,.$$

We now set $W_1 = \partial R/\partial c$ where

$$R(x) = c\big(-2-x+x^2\big)+1-x+3e^{2x}+xe^2$$

so we have $W_1 = -2 - x + x^2$. Substitution into the integral gives

$$\int_0^1 \left[c\big(-2 - x + x^2\big) + 1 - x + 3e^{2x} + xe^2\right]\big[-2 - x + x^2\big]\, dx = 0\,.$$

We need to multiply these two expressions and then integrate the resulting polynomials and the terms involving e^{2x} which is straightforward but a bit tedious. With the help of a symbolic algebra package have

$$\begin{aligned}\int_0^1 \Big[&c(4 + 4x - 3x^2 - 2x^3 + x^4) - 2 + x + 2x^2 - x^3 \\ &+e^{2x}\big(-6 - 3x + 3x^2\big) + e^2\big(-2x - x^2 + x^3\big)\Big]\, dx = 0 \\ &\Rightarrow 4.7c - 29.7553 = 0 \Rightarrow c = 6.33092\,.\end{aligned}$$

Our approximate solution is $u^h = c_1 + c_2 x + c_3 x^2$ and after satisfying the boundary conditions we had $u^h = 1 - (1 + c)x + cx^2$ so with $c = 6.33092$ we have

$$u^h = 1 - 7.330924x + 6.33092x^2\,.$$

In Figure 6.1 we graph the exact solution $u(x) = e^{2x} - xe^2$ and this quadratic approximation. Remember that this is a very coarse approximation because we are using a quadratic polynomial over the entire interval $[0, 1]$ and after we satisfy the boundary conditions we only have one degree of freedom.

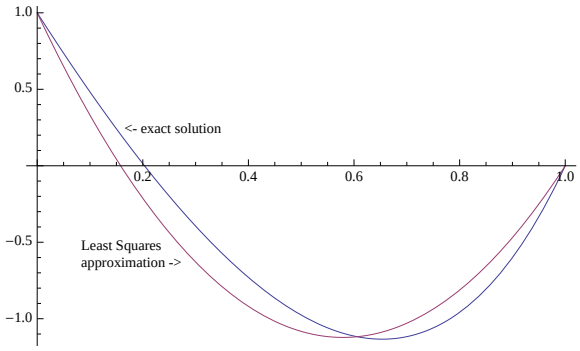


Figure 6.1: Comparison of the exact solution $u(x) = e^{2x} - xe^2$ for the example of this chapter with its MWR approximation $u^h = 1 - 7.330924x + 6.33092x^2$ where the weighting function is chosen by the Least Squares approach.

6.3 Collocation Method

The Collocation Method makes a choice for the weighting function W_i, $i = 1, 2, \ldots, n$ which depends on a family of functions called **Dirac delta functions**.[3] We first review the basic idea of the Dirac delta function and see the effect of its use as a weighting function.

The Dirac delta function is typically denoted by $\delta(x)$ and can be thought of as a function which has an infinitely high spike at the origin and is zero everywhere else;

[3]Named after the theoretical physicist Paul Dirac (1902-1984) who was at Florida State University for the last decade of his life.

the area under the spike is required to be one. In theoretical physics it represents an idealized point mass or point charge. It is not a true function because if it is zero everywhere except at one point then it should have an integral of zero. Consequently the Dirac delta function only makes sense when it is used inside an integral. In that case it can typically be handled as if it were a function. Mathematically it is called a *generalized function*. The Dirac delta function is also useful in modeling impulse phenomena such as a large force applied over a very small time step such as occurs with the strike of a hammer.

Oftentimes the Dirac delta function is defined by the condition

$$\int f(x)\delta(x-a)\,dx = f(a) \tag{6.1}$$

because the Dirac delta function $\delta(x-a)$ is zero every except at $x=a$. This property (6.1) allows us to see the result if we choose the Dirac delta function as the weighting function. For the MWR method we have

$$\int_\Omega RW_i\,d\Omega = 0 \quad i = 1,2,\ldots,n\,.$$

The choice of the weighting function as the Dirac delta function forces the integral to zero at a set of discrete points x_i, $i=1,2,\ldots,n$. The choice of the x_i is dependent on the problem. Substitution of $W_i = \delta(x-x_i)$ into the MWR equation for one dimension gives

$$\int_\Omega R(x)\delta(x-x_i)\,dx = 0 \Rightarrow R(x_i) = 0 \quad i = 1,2,3,\ldots,n\,.$$

In higher dimensions x and x_i are just vectors. Thus the Collocation Method sets the residual to zero at a set of finite points x_i. Of course the choice of the points is critical but we will not go into that here.

Example 3. We return to our example and solve it using the Dirac delta function as the weighting function. Recall that after satisfying the boundary conditions we had only one unknown c so the equation we have to impose is

$$\int_0^1 \Big[c\big(-2-x+x^2\big)+1-x+3e^{2x}+xe^2\Big]W_1\,dx = 0\,.$$

Because we only have one unknown we only need one collocation point. The domain is $[0,1]$ so an obvious choice is to choose the midpoint $x_1 = 0.5$. Thus we choose $W_1 = \delta(x-0.5)$ so we set the residual to zero at $x=0.5$. Recall that

$$R(x) = c\big(-2-x+x^2\big)+1-x+3e^{2x}+xe^2 = 0$$

so evaluating $R(x)$ at $x_1 = 0.5$ and setting it to zero gives

$$R(0.5) = c\big(-2-0.5+0.25\big)+1-0.5+3e^1+0.5e^2 = 0 \Rightarrow 1.75c = 12.3494 \Rightarrow c = 7.05678$$

Thus the quadratic polynomial is $1-8.05678x+7.05678x^2$. The exact solution and the result from the Collocation method is plotted in Figure 6.2.

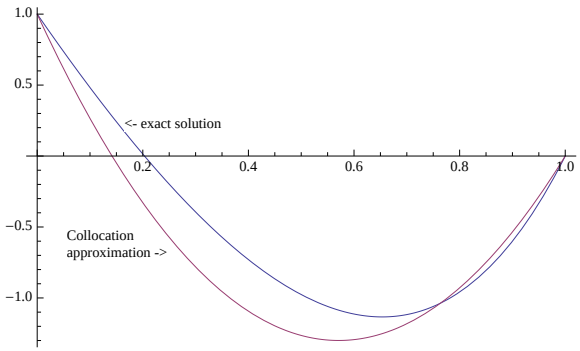


Figure 6.2: Comparison of the exact solution $u(x) = e^{2x} - xe^2$ for the BVP in the example of this chapter with its MWR approximation $u^h = 1 - 8.05678x + 7.05678x^2$ where the weighting function is chosen by the collocation approach by setting the residual to be zero at specific points; in this case a single point is used and is chosen to be the midpoint of the domain.

6.4 Galerkin Methods

Galerkin Methods may be the most popular of all MWR methods. They make the simple choice of the basis functions ϕ_i as the weight functions. One can view them as a modification to the Least Squares approach where the weighting function is the derivative of the residual with respect to the variables. Instead of differentiating the residual, Galerkin methods differentiate the approximating function u^h. Recall that

$$u^h = \sum_{i=1}^{n} c_i \phi_i$$

so differentiating with respect to each c_i gives

$$W_i = \frac{\partial u^h}{\partial c_i} = \phi_i$$

which is just the basis function.

Example 4. We return to our example and solve it using a combination of the basis functions ϕ_i as the weighting functions. Recall that after satisfying the boundary conditions we had only one unknown c so the equation we have to impose is

$$\int_0^1 \Big[c\big(-2 - x + x^2\big) + 1 - x + 3e^{2x} + xe^2 \Big] W_1 \, dx = 0 \, .$$

Our unknown is $u^h = 1 - (1+c)x + cx^2$ (after satisfying the boundary conditions) so differentiating with respect to c gives

$$W_1 = -x + x^2 \, .$$

Substituting into the integral gives

$$\int_0^1 \Big[c\big(-2 - x + x^2\big) + 1 - x + 3e^{2x} + xe^2 \Big] \Big[-x + x^2 \Big] \, dx = 0 \, .$$

Again we expand the terms in the integrand and integrate from zero to one. We have

$$\int_0^1 \Big[c(2x - x^2 - 2x^3 + x^4) + \big(- x + 2x^2 - x^3 + e^{2x}(-3x + 3x^2) + e^2(-x^2 + x^3)\big) = 0$$
$$\Rightarrow c(0.366667) - 2.19909 = 0 \Rightarrow c = 5.99751$$

so the approximate solution u^h is given by

$$u^h = 1 - 6.99751x + 5.99751x^2 \, .$$

A comparison of the exact solution and the Galerkin approximation is given in Figure 6.3.

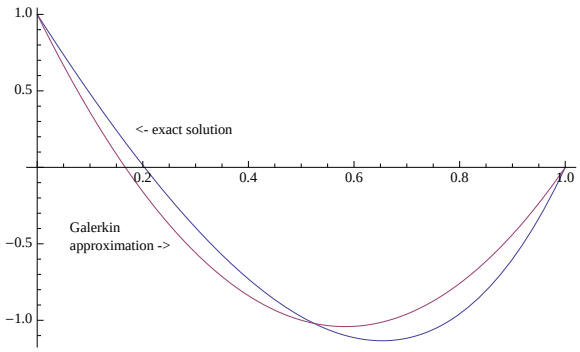


Figure 6.3: Comparison of the exact solution $u(x) = e^{2x} - xe^2$ for the BVP given in the example of this chapter with its MWR approximation $u^h = 1 - 6.99751x + 5.99751x^2$ where the weighting function is chosen by the Galerkin approach.

6.5 Summary

Although the example we did in this chapter used polynomials as our approximating functions, keep in mind that there are many other choices. For example, we can use polynomials, sines and cosines, rational functions, wavelets, etc. Once we use these approximating functions our problem becomes discrete because we choose an approximating space that is finite dimensional and so we have a basis $\{\phi_i\}_{i=1}^n$ of size n and the goal is to find the coefficients of the expansion of our unknown in terms of these basis functions, i.e., $u^h = \sum_{i=1}^n c_i\phi_i$. We know that for a finite dimensional space like $\mathbb{R}^n$ or the space of all polynomials of degree less than n there are an infinite number of choices for the basis. Using one choice of a basis or another will not change the answer but it may make the computations easier. When we solve a BVP we know that we have to solve a linear system $A\mathbf{x} = \mathbf{b}$ so the choice of basis can affect the structure of the matrix, e.g., the bandwidth. Consequently choosing an appropriate basis will be an integral part of making the method efficient and competitive with the FD method.

In the remainder of this section we discuss a weak formulation of the problem which is the starting point for many methods such as the FEM. We compare its

solution with that of solving

$$\int_\Omega R(\mathbf{x})W_i \, d\Omega = 0$$

and discuss reasons for using this alternate formulation. Then we introduce some additional terminology which is commonly used.

6.5.1 Weak formulation

Many methods, such as the FEM, Spectral-Galerkin methods, Wavelet-Galerkin methods, etc. do not solve the equation

$$\int_\Omega R(\mathbf{x})W_i(\mathbf{x}) \, d\Omega = 0$$

but rather solve a $weak$ or$variational$ form of the equation. This alternate form is found by integrating the equation by parts in one dimension or the equivalent in higher dimensions to balance the derivatives. For example, suppose that we have a homogeneous Dirichlet BVP with $-u''(x) = f(x)$ on $[0,1]$ and our weight function is chosen by Galerkin's method so that $W_i = \phi_i$ where $u^h = \sum_{i=1}^n c_i\phi(x)$. Then we have the equation

$$\int_0^1 \big[-\frac{d^2u^h}{dx^2} - f(x)\big]\phi_i(x) \, dx = 0$$

or

$$-\int_0^1 \frac{d^2u^h}{dx^2}\phi_i(x) \, dx = \int_0^1 f(x)\phi_i(x) \, dx \, .$$

In this equation we must take two derivatives of $u^h(x)$ and none on the weight function $\phi_i(x)$. This is called the "strong" formulation; to get the "weak" formulation we integrate the term involving u^h by parts to balance the derivatives, i.e., to have one derivative on u^h and one on ϕ_i. Integrating by parts gives the weak problem

$$\int_0^1 \frac{du^h}{dx}\frac{d\phi_i}{dx} \, dx - \big[(u^h)'(1)\phi_i(1) - (u^h)'(0)\phi_i(0)\big] = \int_0^1 f(x)\phi_i(x) \, dx \quad i = 1, 2, \ldots, n \, . \tag{6.2}$$

The boundary terms will disappear if we require $\phi_i(x)$ to satisfy the homogeneous Dirichlet boundary conditions for this problem.

Are the solutions to the strong and weak problems identical? Clearly, in our case if u^h satisfies the strong equation then it satisfies the weak equation because all we did is integrate by parts. However, the converse may not always hold. Why is this? The weak problem only requires that u^h possess one derivative whereas the strong problem requires it to have two derivatives. So you could imagine a weak problem whose solution has one derivative but not two so that the integration by parts step can not be reversed. Hence the name "weak" because we are imposing weaker conditions on the unknown. However, in the majority of problems we will consider the solution to the weak problem and the strong problem will coincide. One of the

reasons for working with a weak problem is that the approximating space span$\{\phi_i\}$ is much easier to construct if only one derivative is imposed. Another advantage of using the weak formulation is that it is trivial to impose Neumann boundary conditions. For example, if we have a Neumann condition like $u'(0) = 4$ then we simply substitute this into the weak form (6.2) and it becomes a known value and moves to the right hand side. Contrast this with the FDM where we had to set up fictitious points on the boundary.

We introduce some terminology that is prevalent when using the Galerkin method. Remember that we seek an approximation u^h which we write as $u^h = \sum_{i=1}^{n} \phi_i(x)$ for our basis $\{\phi_i\}_{i=1}^n$ so we "try" our solution as a linear combination of the basis vectors. In this case the ϕ_i are called **trial functions**. On the other hand, our equation (6.2) has to hold for each $W_i = \phi_i(x)$, $i = 1, 2, \ldots, n$ so we say that we "test" the equation against each W_i. These are called **test functions**.

Example 5. We end this chapter by returning to our example and comparing the results from the three methods. First the solutions are plotted in Figure 6.4 and in Table 6.1 we compare pointwise errors. Finally Figure 6.5 plots the errors for each method. Note that the error for the Collocation Method is smallest at $x = 0.5$ which is expected because that was the collocation point we chose.

x	Exact	Collocation	Galerkin	Least Squares
0.0	1.0	1.0	1.0	1.0
0.1	0.482497	0.26489	0.360225	0.330217
0.2	0.0140135	-0.329085	-0.1596	-0.212948
0.3	-0.394598	-0.781924	-0.559475	-0.629494
0.4	-0.730082	-1.09363	-0.8394	-0.919422
0.5	-0.976246	-1.2642	-0.999375	-1.08273
0.6	-1.11332	-1.29363	-1.0394	-1.11942
0.7	-1.11714	-1.18192	-0.959475	-1.0295
0.8	-0.958212	-0.929085	-0.7596	-0.81295
0.9	-0.600503	-0.53511	-0.439775	-0.469786
1.	0.0	0.0	0.0	0.0

Table 6.1: Pointwise comparison of solutions for the Least Squares, Galerkin and Collocation approximations for our example.

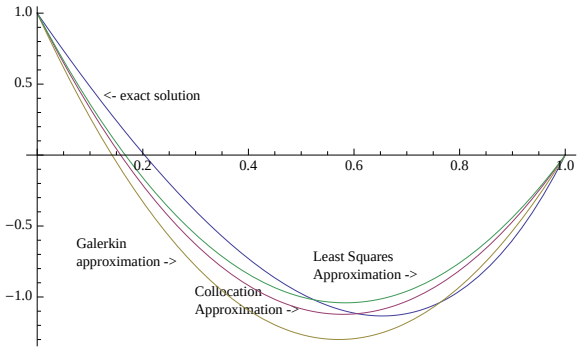


Figure 6.4: Comparison of the approximate solutions for the BVP given in the example of this chapter using Least Squares, Galerkin, and Collocation approximations.

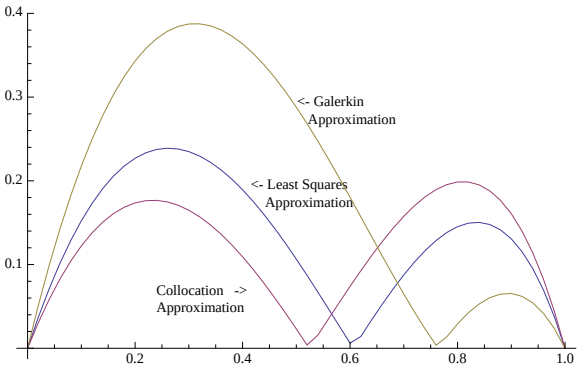


Figure 6.5: Comparison of the pointwise errors (not normalized) for the BVP given in Example **??** with its Least Squares, Galerkin, and Collocation approximations.

Chapter 7

The Finite Element Method in 1D

The finite element method (FEM) is a method for approximating the solution of differential equations; it has gained widespread use in a diverse range of areas such as fluid mechanics, structural mechanics, biological science, chemistry, electromagnetism, financial modeling, and superconductivity, to name a few. One can find articles where finite element methods have been employed to study everything from stress analysis of a human tooth to design of an airplane wing.

Although the foundations for the finite element method were laid in the first half of the twentieth century, it did not become widely used until much later. Structural engineers were the first to use the technique in the 1940's and 1950's; mathematicians became interested in analyzing and implementing the method in the late 1960's. The first symposium on the mathematical foundations of the finite element method was held in June of 1972 with over 250 participants. Prior to this symposium there had already been numerous national and international conferences held on the finite element method but mainly with an emphasis on engineering applications. In the following decades the finite element method has grown in popularity as a useful tool in design and application as well as a fertile area for mathematical analysis.

The FEM is a Galerkin method where the choice of weighting function in the Mean Weighted Residual approach is the same as the trial function or basis. Finite element methods are a class of methods for obtaining approximate solutions of differential equations, especially partial differential equations.[1] As such, they can be compared to other methods that are used for this purpose, e.g., finite difference methods, finite volume methods or spectral methods. There are seemingly countless finite element methods in use, so that one cannot refer to any method as *the* finite element method any more that one can refer to any particular method as being *the* finite difference method. In fact, there are numerous subclasses of finite element

[1] Finite element methods were not always thought of in this manner, at least in the structural mechanics community. In an alternate definition, structural systems are directly discretized into approximate submembers such as beams, plates, shells, etc., without any recourse to differential equations. These submembers are then called "finite elements."

methods, each saddled with a modifier, e.g., *Galerkin-Petrov*, *mixed*, or *collocation* finite element methods.

The finite element method is distinguished from other approaches to approximating differential equations by two key factors - the use of a weak formulation and the use of piecewise polynomial approximation. Piecewise polynomial approximation is very attractive due to the ease of use, its approximation properties, and the availability of bases which are locally supported; that is, bases that are nonzero over a small portion of the domain.

Although the principal use of finite element methods is for approximating solutions to partial differential equations, it is instructive to look at one-dimensional problems for their simplicity and ease of understanding. In addition, when we approximate PDEs using rectangular elements, then we take tensor products of one-dimensional elements. The goal of this chapter is to investigate the FEM for a two-point BVP.

7.1 A Simple Example

In order to begin to understand the basic idea of the finite element method and the steps involved, we define a finite element method for the very simple two-point boundary value problem

$$-u''(x) = f(x) \quad 0 < x < 1\,, \tag{7.1a}$$

$$u(0) = 0\,, \tag{7.1b}$$

and

$$u'(1) = 0\,. \tag{7.1c}$$

The finite element approximation $u^h(x)$ to the solution $u(x)$ of (7.1) is defined to be the solution of the following problem:

$$\textit{find } u^h(x) \in V^h \textit{ such that} \quad \int_0^1 (u^h)'(v^h)'\,dx = \int_0^1 f v^h\,dx \quad \forall v^h \in V^h\,, \tag{7.2}$$

where V^h is a finite dimensional space of functions[2] that vanish at $x = 0$ and are sufficiently smooth. Actually, Problem 7.2 defines a finite element method only if the approximating space V^h is chosen to consist of *piecewise polynomial functions*. This choice of approximating functions, along with a judicious choice of basis for V^h, are primarily responsible for the success of the finite element method as a computational method.

We now ask ourselves what (7.2) has to do with the original problem (7.1). Recall that for the MWR method we force the residual to zero over the domain and test it against a weight function; in the Galerkin method the weight function is

[2]Recall that a finite dimensional function space is characterized by the fact that a basis of size $m < \infty$ can be chosen.

chosen the same as the basis which in the finite element case consists of piecewise polynomials. So we have

$$-\int_0^1 \frac{d^2u^h}{dx^2}v^h\,dx - \int_0^1 fv^h\,dx = 0$$

where v^h is our weighting function. We want to integrate the first integral by parts to balance the derivatives; recall that the integration by parts formula is

$$\int_a^b w\,dz = w(b)z(b) - w(a)z(a) - \int_a^b z\,dw$$

so if we let $w = -v^h$ and $dz = (u^h)''dx$ then we have

$$-\int_0^1 \frac{d^2u^h}{dx^2}v^h\,dx = \int_0^1 (u^h)'(v^h)'\,dx - (u^h)'(1)v^h(1) + (u^h)'(0)v^h(0) = \int_0^1 fv^h\,dx\,.$$

Now we said that V^h was a space of piecewise polynomials that are zero at $x = 0$ so $v^h(0) = 0$ and that term disappears. The other term also disappears because the boundary condition $u'(1) = 0$ should also be satisfied by the approximating function u^h. Consequently we are left with (7.2).

However, most FEM books or papers don't refer to the weighted residual method. Instead they view (7.2) as an approximation to the following problem:

$$\begin{gathered} \textit{find } u(x) \textit{ such that } u(0) = 0 \textit{ and} \\ \int_0^1 u'v'\,dx = \int_0^1 fv\,dx \quad \forall v \in V\,, \end{gathered} \tag{7.3}$$

where for each $v \in V$, $v(0) = 0$ and v is "sufficiently smooth". Note that this is posed over an infinite dimensional space V, i.e., there is no basis for V. Of course (7.3) is not the original BVP we started with but rather a weak form of it. To obtain the weak form of the BVP we typically multiply the differential equation by a test function (or weighting function) which we require to satisfy any homogeneous Dirichlet boundary conditions and integrate over the domain. We then balance the derivatives by integrating by parts in one dimension or in higher dimensions we use an analogous approach. Neumann boundary conditions are typically satisfied by the weak form. For example, for our BVP we have

$$-u''v = fv \Rightarrow -\int_0^1 u''v\,dx = \int_0^1 fv\,dx$$

and integrating by parts yields

$$\int_0^1 u'v'\,dx - u'(1)v(1) + u'(0)v(0) = \int_0^1 fv\,dx\,.$$

Now requiring $v(0) = 0$ makes the term $u(0)v(0)$ disappear and imposing the boundary condition $u'(1) = 0$ makes the term $u'(1)v(1)$ disappear so we are left with the continuous weak problem (7.3). We can reverse the steps if u is sufficiently smooth, i.e., it possesses two derivatives. However, the weak problem only requires one derivative on $u(x)$.

7.1.1 Some Terminology

Let us now introduce some terminology that is used in the FEM. We call $u(x)$ a *classical solution* of (7.1) if, upon substitution into these relations, equality holds at every point $x \in (0,1)$. We call solutions of (7.3) that are not classical solutions of (7.1) *weak solutions* of the latter problem and (7.3) itself is referred to as a *weak formulation* of (7.1). Analogously, problem (7.2) is termed a *discrete weak problem* because it is posed over a finite dimensional space which means the space has a basis so we have a discrete problem. For most problems we will consider, the classical solution and the weak solution will coincide.

The functions v^h and u^h in (7.2) are called *test* and *trial* functions, respectively. The same terminology is used for the corresponding functions v and u appearing in (7.3). Where do these names come from? Suppose someone gave us a function $u^h(x)$ and claimed that it was a solution of the discrete weak problem (7.2). To verify the claim, we would put the function $u^h(x)$ on "trial," i.e., we would determine if substituting it into (7.2) results in the left-hand side equal to the right-hand side for all possible test functions $v^h(x) \in V^h$.

The Dirichlet boundary condition (7.1b) and the Neumann boundary condition (7.1c) are treated differently within the framework of the weak formulation (7.3) or its approximation (7.2). First, we note that the Neumann boundary condition (7.1c) is not imposed on the test or trial functions; however, we saw that if $u(x)$ satisfies the weak problem (7.3), then this Neumann boundary condition is indeed satisfied. Such boundary conditions, i.e., boundary conditions that are not required of the trial functions but are satisfied "naturally" by the weak formulation, are called *natural boundary conditions.* On the other hand, nothing in the process we used to go from the weak problem (7.3) to the classical problem (7.1) implied that the Dirichlet boundary condition (7.1b) was satisfied. For this reason, we imposed the boundary condition as a constraint on the possible trial functions. Such boundary conditions are called *essential boundary conditions.* Note that for the discrete problem, the approximate solution $u^h(x)$ satisfies (by construction) the essential boundary condition (7.1b) exactly, but the natural boundary condition (7.1c) is only satisfied in a weak sense. This will be explored in the examples.

7.1.2 Polynomial Approximation

The two main components of the FEM are its variational principles which take the form of weak problems and the use of piecewise polynomial approximation. In our example we use the discrete weak or variational formulation (7.2) to define a finite element method but we have not used piecewise polynomials yet. In this example we choose the simple case of approximating with continuous piecewise linear polynomials; that is, a polynomial which is linear when restricted to each subdivision of the domain. To define these piecewise polynomials, we first discretize the domain $[0,1]$ by letting N be a positive integer and setting the *grid points* or *nodes* $\{x_j\}_{j=0}^N$ so that $0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = 1$. Consequently we have $N+1$ nodes and N elements. These nodes serve to define a partition of the interval $[0,1]$ into the subintervals $T_i = [x_{i-1}, x_i]$, $i = 1, \ldots, N$. For now we

will take the grid to be uniform but we will see that we can easily modify the process to include nonuniform grids. The subintervals T_i are the simplest examples of *finite elements*. We choose the finite dimensional space V^h in (7.2) to be the space of continuous piecewise linear polynomials over this partition of the given interval so that each v^h is a continuous piecewise linear polynomial. This means that when we look at v^h in an element T_i it is a linear polynomial and it is continuous across elements. In Figure 7.1 we plot a function which is a continuous piecewise linear function defined over a uniform partition of $[0, 1]$ using seven grid points and which is zero at $x = 0$.

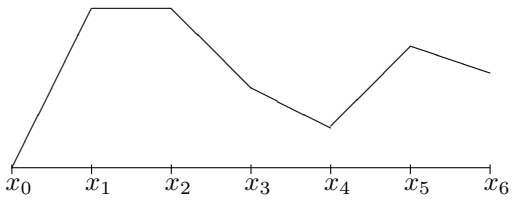


Figure 7.1: Example of a continuous piecewise linear function in V^h defined over the partition $\{x_i\}_0^6$ of $[0, 1]$.

7.1.3 How do you implement finite element methods?

We now translate the finite element method defined by (7.2) into something closer to what a computer can understand. To do this, we first show that (7.2) is equivalent to a linear algebraic system once a basis for V^h is chosen. Next we indicate how the entries in the matrix equation can be evaluated.

Let $\{\phi_i(x)\}_{i=1}^N$ be a basis for V^h, i.e., a set of linearly independent functions such that any function belonging to V^h can be expressed as a linear combination of these basis functions. The dimension of V^h is N which we will demonstrate later if we define V^h to be all continuous piecewise polynomials defined over our partition of $[0, 1]$ which are zero at $x = 0$. Thus, the set $\{\phi_i(x)\}|_{i=1}^N$ has the property that it is linearly independent, i.e.,

$$\sum_{i=1}^{N} \alpha_i \phi_i(x) = 0 \quad \text{implies} \quad \alpha_i = 0 \quad \text{for } i = 1, \ldots, N$$

and it spans the space. That is, for each $w^h \in V^h$ there exists real numbers w_i, $i = 1, \ldots, N$, such that

$$w^h(x) = \sum_{i=1}^{N} \omega_i \phi_i(x)\,.$$

In the weak problem (7.2), the solution $u^h(x)$ belongs to V^h and the test function $v^h(x)$ is arbitrary in V^h. Since the set spans V^h we can set $u^h = \sum_{j=1}^N \mu_j \phi_j$ and then express (7.2) in the following equivalent form: find $\mu_j \in \mathbb{R}^1$, $j = 1, \ldots, N$, such that

$$\int_0^1 \frac{d}{dx}\left(\sum_{j=1}^N \mu_j \phi_j(x)\right) \frac{d}{dx}\left(v^h\right)\, dx = \int_0^1 f(x) v^h\, dx \quad \forall v^h \in V^h$$

or equivalently

$$\sum_{j=1}^N \mu_j \int_0^1 \phi_j'(x)(v^h)'\, dx = \int_0^1 f(x) v^h\, dx \quad \forall v^h \in V^h\,.$$

Now this equation must hold for each function $v^h \in V^h$. We claim that it is enough to test the equation for each element in the basis; that is,

$$\sum_{j=1}^N \mu_j \int_0^1 \phi_j'(x)\phi_i'(x)\, dx = \int_0^1 f(x)\phi_i(x)\, dx \quad \text{for } i = 1, \ldots, N.$$

To see this, let $v^h \in V^h$ where v^h is arbitrary; it can be written as $v^h = \nu_1\phi_1 + \cdots \nu_n\phi_N$ because the $\phi_i(x)$ form a basis for V^h. We substitute this v^h into our weak formulation (7.2) to get

$$\int_0^1 \frac{du^h}{dx}\Big[\nu_1 \frac{d\phi_1}{dx} + \cdots + \nu_N \frac{d\phi_N}{dx}\Big]\, dx = \int_0^1 f(x)\big[\nu_1\phi_1 + \cdots + \nu_N\phi_N\big]\, dx$$

which can be written as

$$\nu_1 \int_0^1 \frac{du^h}{dx}\frac{d\phi_1}{dx}\, dx + \cdots + \nu_N \int_0^1 \frac{du^h}{dx}\frac{d\phi_N}{dx}\, dx = \nu_1 \int_0^1 f(x)\phi_1\, dx + \cdots + \nu_N \int_0^1 f(x)\phi_N\, dx\,.$$

On the other hand, the weak formulation has to hold for each $v^h = \phi_i$ so when we choose v^h to be each basis function we have

$$\int_0^1 \frac{du^h}{dx}\frac{d\phi_1}{dx}\, dx = \int_0^1 f(x)\phi_1\, dx \quad \cdots \quad \int_0^1 \frac{du^h}{dx}\frac{d\phi_N}{dx}\, dx = \int_0^1 f(x)\phi_N\, dx\,.$$

If we multiply the first equation by ν_1, the second by ν_2, etc. and add the equations then we get the previous result; consequently requiring the weak formulation to hold for each ϕ_i means it holds for any $v^h = \nu_1\phi_1 + \cdots \nu_n\phi_N$. Using this fact, the discrete problem is rewritten as

find μ_j, $j = 1, \ldots, N$, *such that*

$$\sum_{j=1}^N \mu_j \left(\int_0^1 \phi_i'(x)\phi_j'(x)\right)\, dx = \int_0^1 f\phi_i(x)\, dx \qquad \text{for } i = 1, \ldots, N. \tag{7.4}$$

Clearly (7.4) is a linear algebraic system of N equations in N unknowns. Indeed, if the entries of the matrix K and the vectors $\mathbf{U}$ and $\mathbf{b}$ are defined by

$$K_{ij} = \int_0^1 \phi_i'(x)\phi_j'(x)\,dx\,, \quad U_j = \mu_j\,, \quad \text{and} \quad b_i = \int_0^1 f(x)\phi_i\,dx$$

for $i, j = 1, \ldots, N$ then, in matrix notation, (7.4) is given by

$$K\mathbf{U} = \mathbf{b}\,. \tag{7.5}$$

However, we have not yet completely formulated our problem so that it can be implemented on a computer. We first need to choose a particular basis and then the integrals appearing in the definition of K and $\mathbf{b}$ must be evaluated or approximated. Clearly there are many choices for a basis for the space of continuous piecewise linear functions defined over our subdivision of $[0, 1]$. We will see that a judicious choice of the basis set will result in (7.5) being a tridiagonal system of equations and thus one which can be solved efficiently in $\mathcal{O}(N)$ operations. Recall that using a second centered difference quotient for $u''(x)$ resulted in a tridiagonal system so the two system would require equivalent work to solve.

For now, let's assume that we have chosen a specific basis and turn to the problem of evaluating or approximating the integrals appearing in K and $\mathbf{b}$. For a simple problem like ours we can often determine the integrals exactly; however, for a problem with variable coefficients or one defined on a general polygonal domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ this would not be practical. Even if we have software available that can perform the integrations, this would not lead to an efficient implementation of the finite element method. Thus to obtain a general procedure which would be viable for a wide range of problems, we approximate the integrals by a quadrature rule. For example, for the particular implementation we are developing here, we use the midpoint rule in each element to define a composite rule. Recall that the midpoint rule on a generic interval $[x_{k-1}, x_k]$ is

$$\int_{x_{k-1}}^{x_k} g(x)\,dx \approx (x_k - x_{k-1})g\big(\frac{x_{k-1} + x_k}{2}\big)$$

and the composition rule on $[0, 1]$ is found by applying this rule over each subinterval to get

$$\int_0^1 g(x)\,dx = \sum_{k=1}^{N} \int_{x_{k-1}}^{x_k} g(x)\,dx \approx \sum_{k=1}^{N} g\left(\frac{x_{k-1} + x_k}{2}\right)(x_k - x_{k-1})\,.$$

Using this rule for the integrals that appear in (7.5), we are led to the problem

$$K^h\mathbf{U}^h = \mathbf{b}^h\,, \tag{7.6}$$

where the superscript h on the matrix K and the vector $\mathbf{b}$ denotes the fact that we have approximated the entries of K and $\mathbf{b}$ by using a quadrature rule to evaluate the integrals. The entries of K^h, and $\mathbf{b}^h$ are given explicitly by

$$K_{ij}^h = \sum_{k=1}^{N} (x_k - x_{k-1})\phi_i'\left(\frac{x_{k-1} + x_k}{2}\right)\,\phi_j'\left(\frac{x_{k-1} + x_k}{2}\right)\,, \quad \text{for } i, j = 1, \ldots, N$$

and

$$b_i^h = \sum_{k=1}^{N} (x_k - x_{k-1}) f\left(\frac{x_{k-1}+x_k}{2}\right) \phi_i\left(\frac{x_{k-1}+x_k}{2}\right), \quad \text{for } i = 1, \ldots, N\,.$$

In our example, $K^h = K$. To see this, recall that we have chosen V^h as the space of continuous piecewise linear functions on our partition of $[0,1]$ and thus the integrands in K are constant on each element T_i. The midpoint rule integrates constant functions exactly so even though we are implementing a quadrature rule, we have performed the integrations exactly. However, in general, $\mathbf{b}^h \neq \mathbf{b}$ so that $\mathbf{U}^h \neq \mathbf{U}$. In the sequel we will not include the superscript h on our matrices but keep in mind that we will be typically making an approximation to the entries of the matrix because of numerical quadrature. As long as we choose a quadrature rule that is sufficiently accurate for our choice of basis, then the error in the quadrature rule will not dominate the error made by our choice of basis.

Once the specific choice of a basis set for V^h is made, the matrix problem (7.6) can be directly implemented on a computer. A standard linear systems solver can be used to obtain $\mathbf{U}$ and should be chosen based upon the structure and properties of K .

There are an infinite number of possible basis sets for our space V^h. For example, for $\mathbb{R}^3$ we can choose $\{(1,1,1)^T, (1,0,1)^T, (0,5,0)^T\}$ or $\{(7,1,7)^T$,$(0,2,1)^T$, $(4,-4,1)^T\}$ but a more common choice is the set $\{(1,0,0)^T$, $(0,1,0)^T$, $(0,0,1)^T\}$ because it is much easier to write a vector in $\mathbb{R}^3$ as a linear combination of these vectors. Of course this is because each vector is zero in all but one component. So we want to choose basis functions for the space of continuous piecewise linear functions on our partition of $[0,1]$ which have an analogous property. If the basis functions are nonzero over the whole interval $(0,1)$, then, in general, the resulting discrete systems such as (7.5) or (7.6) will involve *full matrices*, i.e., matrices having possibly all nonzero entries. In this case we say the basis functions have *global* support.

In order to achieve maximum sparsity in the discrete systems such as (7.5) or (7.6), the basis functions should be chosen to have *local support*, i.e., to be nonzero on as small a portion of the domain as possible. In the one dimensional case we have considered here, the basis functions should be nonzero over as small a number of subintervals as possible. Such a basis set is provided by the "hat" functions defined by

$$\text{for } i = 1, \ldots, N, \qquad \phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & \text{for } x_{i-1} \leq x \leq x_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & \text{for } x_i \leq x \leq x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{7.7}$$

and

$$\phi_N(x) = \begin{cases} \dfrac{x - x_{N-1}}{x_N - x_{N-1}} & \text{for } x_{N-1} \le x \le x_N \\ 0 & \text{otherwise.} \end{cases} \tag{7.8}$$

A sketch of these functions for the case $N = 4$ on a uniform partition of $[0, 1]$ is given in Figure 7.2. Note that for all $i = 1, \ldots, N$, $\phi_i(0) = 0$, $\phi_i(x)$ is continuous on $[0, 1]$, is a linear polynomial on each subinterval $[x_{j-1}, x_j]$, $j = 1, \ldots, N$, and $\phi_i(x)$ is nonzero only in $[x_{i-1}, x_{i+1}]$. It can be shown that the set $\{\phi_i(x)\}_{i=1}^N$ given by (7.7) and (7.8) is linearly independent and forms a basis for the space V^h.

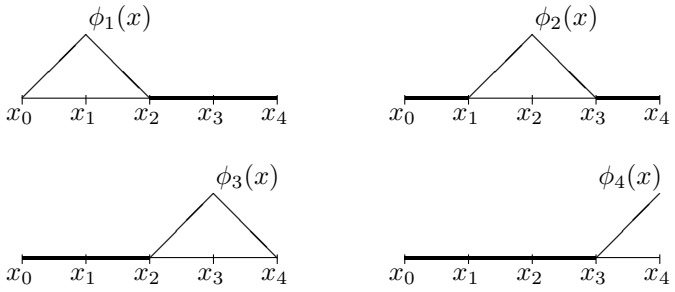


Figure 7.2: Example of the hat basis functions for V^h for a subdivision of $[0, 1]$ using four uniform elements. All basis functions are zero at $x = 0$ and have a maximum height of one.

Now let's examine the entries of the matrices K and K^h appearing in the linear systems (7.5) or (7.6), respectively, for the basis functions defined in (7.7) and (7.8). It is easy to see that $K_{ij} = 0$ unless $|i - j| \le 1$. Thus, for any number of elements N, these matrices have nonzero entries only along the main diagonal and the first upper and lower subdiagonals, i.e., they are tridiagonal matrices. This is the optimal sparsity achievable with piecewise linear finite elements and is the same structure that we got when we solved the same BVP using a second centered difference approximation. When we delve into the method more we will explicitly determine the entries in the matrix.

7.1.4 A comparison with finite difference methods

Like finite difference methods, particular finite element methods are ultimately defined based on a grid, i.e., a partition of a given domain in Euclidian space into subdomains. More often than not, the grid itself is defined by selecting a finite number of points in the given domain. Thus, both classes of methods may be thought of as grid-based methods.

What separates the two classes of methods? We can immediately point out one difference, albeit a somewhat philosophical one, between finite difference and finite

element methods. The finite difference method is derived primarily by *approximating operators*, i.e., derivatives. On the other hand, the primary approximation step in deriving the FEM (7.2) was to replace the solution u in (7.3) by an approximation u^h, i.e., by *approximating the solution.*

Finite element methods possess many desirable properties that account for their popularity in a variety of settings. Some of these we have already encountered. For example, within the finite element framework, natural boundary conditions such as (7.1c) are very easily enforced. Also there is no difficulty in treating problems with nonuniform grids. A third advantage that we have alluded to is that, due to being able to introduce sophisticated function theoretic machinery, finite element methods can be "easily" analyzed with complete rigor. All three of these are thought of as posing difficulties within the finite difference framework.

There are other good features inherent in finite element methodologies. Perhaps the most important one is the ability of the FEM to "easily" treat problems in complicated, e.g., non-rectangular, domains.[3] Another good feature is that finite element methods preserve certain symmetry and positivity properties possessed by problems such as (7.1). In particular, in this case, the matrix K appearing in (7.5) is symmetric and positive definite.

A final desirable feature of finite element methods is that, when they are properly implemented, they lead to sparse discrete problems. This spasity property is crucial to the efficiency of the method and results from a judicious choice for the basis set $\{\phi_i(x)\}_{i=1}^N$ for the finite element space V^h.

Which method is best? Unfortunately, there is no best method for all problems. Which method is best, be it of the finite difference, finite element, finite volume, spectral, etc., depends on the class of problems under consideration or, often, on the specific problem itself. It is not even possible to say that one finite element method is better than another one in all settings. In order to determine which method is best (or even good) for a given problem, one must understand its definition, implementation, and analysis. The purpose of studying different methods is to obtain the tools, knowledge, and experience so that rational comparisons and decisions can be made.

7.2 Two point boundary value problems

We now want to look at the FEM in more detail for our prototype BVP in one dimension. We will begin by defining an appropriate continuous weak formulation and show that for sufficiently smooth solutions the classical and weak solutions coincide. Then we discuss the finite element approximation using different choices of approximating polynomials, investigate the resulting matrix equations, and determine error estimates. In addition, we provide computational results for several examples.

Before taking a closer look at the FEM we want to define the space of functions defined on a domain Ω called $L_2(\Omega)$. This linear (vector) space and an associated

[3] Of course, since we have only looked at problems in one dimension, we have not yet been exposed to such domains.

norm is used throughout the discussion of the FEM. Recall that when we computed an error using finite differences, we had a vector containing the difference in the approximate and exact solution at each node. Then we used a vector norm such as the ℓ_2 norm, i.e., the standard Euclidean norm, to measure the error. In the case of the FEM our approximation u^h is a continuous function because it is a linear combination of continuous piecewise polynomials. So in this case we don't measure the error in the ℓ_2 vector norm; instead we use the analogue for continuous functions. We define $L_2(\Omega)$ as the space of all square integrable functions [4]

$$L_2(\Omega) \equiv \{v \in C^0(\Omega) \mid \int_\Omega v^2 \, d\Omega < \infty\} \tag{7.9}$$

and the associated norm

$$\|v\|_2 = \Big[\int_\Omega v^2 \, d\Omega\Big]^{1/2}. \tag{7.10}$$

In words we say that $L_2(\Omega)$ consists of all continuous functions on Ω which are square integrable, i.e., the integral of the square of the function is finite. It is important to realize the similarity between the standard Euclidean norm and the L_2 norm defined here. We have replaced the sum with an integral and instead of taking the dot product of a vector with itself we simply square the function. Note also that we have used the same notation $\|\cdot\|_2$ for both the ℓ_2 and L_2 norms; the particular norm we are using should be clear from the context. When we talk about finite element approximations we will always use the L_2 norm and for the finite difference method we will always use the ℓ_2 norm.

7.2.1 A two-point BVP with homogeneous Dirichlet boundary data

We begin by returning to our prototype two-point boundary value problem on $[0,1]$; specifically we seek a function $u(x)$ satisfying

$$\begin{aligned} -\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u &= f(x) \quad \text{for } 0 < x < 1 \\ u(0) &= 0 \\ u(1) &= 0\,, \end{aligned} \tag{7.11}$$

where $p(x)$, $q(x)$, and $f(x)$ are given functions defined on $[0,1]$. As before, we assume that $0 < p_{\min} \le p(x) \le p_{\max}$ and $q_{\min} = 0 \le q(x) \le q_{\max}$ where $p_{\min}$, $p_{\max}$, and $q_{\min}$ are positive constants and $f \in L_2(0,1)$. The solution u to problem (7.11) is the *classical solution*. We are interested in a *weak solution* of (7.11); i.e., in a function $u(x)$ that satisfies (7.11) in some sense even when f, p, q are not continuous; if f, p, q are sufficiently smooth then we want the weak solution to coincide with the classical solution.

[4] The rigorous definition of L_2 is that it is the completion of the space we defined, i.e., the limit of every Cauchy sequence is added to this space; specifically it is a complete inner product space which we call a Hilbert space. We will not be proving results which require this precise definition so we will be a bit less rigorous here.

7.2.2 Weak formulation

In choosing the underlying function space for our weak formulation of (7.11), we know that multiplication of the differential equation by an appropriate test function, integrating over the domain and then integrating by parts to balance the order of the derivatives results in both the test and trial functions having one derivative. Consequently we require our solution to be integrable and to possess at least one derivative. In addition, we constrain our space so that we only consider functions which satisfy the homogeneous Dirichlet boundary conditions. We call V_0 to be the function space[5] in which we seek a solution $u(x)$ and the space of test functions. The weak problem is stated as:

$$\begin{cases} \text{seek } u \in V_0 \text{ satisfying} \\ \quad \int_0^1 p(x)u'(x)v'(x)\,dx + \int_0^1 q(x)u(x)v(x)\,dx = \int_0^1 fv\,dx \quad \forall v \in V_0\,. \end{cases} \tag{7.12}$$

Note that if u is the classical solution of (7.11) then $u(x)$ also satisfies the weak problem because for $v \in V_0$

$$\begin{aligned} \int_0^1 fv\,dx &= \int_0^1 \big(-(pu')' + qu\big)v\,dx \\ &= \int_0^1 pu'v'\,dx + \int_0^1 quv\,dx - \big[pu'v\big]\,\big|_0^1 \\ &= \int_0^1 pu'v'\,dx + \int_0^1 quv\,dx \end{aligned}$$

because v is zero at $x = 0$ and $x = 1$. Conversely, if $u \in V_0$ satisfies (7.12) and u is sufficiently smooth, i.e., $u \in C^2(0,1)$, a situation which can be guaranteed if p, q and f are themselves sufficiently smooth, then u coincides with the classical solution of (7.11). To see this, note that the homogeneous Dirichlet boundary conditions are satisfied because $u \in V_0$; the differential equation holds because the weak problem implies

$$\int_0^1 pu'v'\,dx + \int_0^1 quv\,dx - \int_0^1 fv\,dx = 0$$

and thus integrating by parts gives

$$\int_0^1 \big[(-pu')' + qu - f\big]v\,dx = 0 \quad \forall v \in V_0$$

which has to hold for every $v \in V_0$ and so the only way this integral is zero is if $(-pu')' + qu - f = 0$. Recall that if we can find a function $u \in V_0$ which is the unique solution of (7.12), then we call u the *weak solution* of (7.11) in V_0. In this problem we constrained our function space to consist of functions which satisfy the homogenous Dirichlet boundary conditions. We recall that boundary conditions which are satisfied by constraining the trial space are called *essential*.

[5] The appropriate function space is a Hilbert space called a Sobolev space but we will not get into the mathematical details here.

Because the solution of the weak problem may not be a solution of the classical problem, one needs to demonstrate that the solution of the weak problem exists and is unique. The result that givens conditions for existence and uniqueness is called the Lax-Milgram theorem but we will not investigate it here. The interested reader is referred to a standard text in the FEM. For our exposition, we will assume that the weak problem has a unique solution.

7.2.3 Approximation using piecewise linear polynomials

We now turn to approximating u, the solution of the weak problem (7.12), by its Galerkin approximation u^h in a finite dimensional subspace V_0^h of V_0. The approximate solution is required to satisfy (7.12) but only for all $v^h \in V_0^h$; the discrete weak problem is

$$\left\{ \begin{array}{l} \text{seek } u^h \in V_0^h \text{ satisfying} \\ \quad \int_0^1 p(x)(u^h)'(x)(v^h)'(x)\,dx + \int_0^1 q(x)v^h(x)w^h(x)\,dx = \int_0^1 fv^h\,dx \quad \forall v^h \in V_0^h\,. \end{array} \right. \tag{7.13}$$

Because $V_0^h \subset V^h$ the conditions that guaranteed a unique solution to the continuous weak problem are automatically satisfied on V_0^h so we are guaranteed that there exists a unique $u^h \in V_0^h$ which satisfies (7.13).

To write the discrete weak problem as a linear system of equation we must choose a specific basis. In this section we choose V_0^h to be the space of continuous linear piecewise polynomials defined on a partition of $[0,1]$ which satisify the homogeneous Dirichlet boundary conditions; in a later section we will consider other choices of V_0^h. In particular, we consider the following partition of $[0,1]$:

$$0 = x_0 < x_1 < \cdots < x_{N+1} = 1 \quad \text{where} \quad x_i = x_{i-1} + h_i, \quad 1 \le i \le N+1\,, \tag{7.14}$$

and where h_i, $1 \le i \le N+1$ are given numbers such that $0 < h_i < 1$ and $\sum_{i=1}^{N+1} h_i = 1$. We define $h = \max_{1 \le i \le N+1} h_i$; if $h_i = h$ for all i then we call the subdivision *uniform*. Here we consider the general case and will demonstrate that it is no more difficult to handle a nonuniform grid in finite elements. A continuous piecewise linear function with respect to the given subdivision on $[0,1]$ is a function $\phi(x)$ defined on $[0,1]$ which is linear on each subinterval i.e., $\phi(x) = \alpha_i x + \beta_i$ on $[x_i, x_{i+1}]$, $0 \le i \le N$. Consequently $\phi(x)$ can be a different linear function on each interval. To impose continuity we require that the constants α_i, β_i, $i = 1, 2, \ldots, N$ defining $\phi(x)$ satisfy $\alpha_{i-1}x_i + \beta_{i-1} = \alpha_i x_i + \beta_i$, $i = 1, \ldots, N$. In addition, for $\phi(x)$ to be in V_0^h it must be zero at $x = 0$ and $x = 1$. We define

$$\begin{array}{l} V_0^h = \{\phi(x) \mid \phi \in C[0,1], \\ \qquad \phi(x) \text{ linear on } [x_i, x_{i+1}] \text{ for } 0 \le i \le N, \phi(0) = \phi(1) = 0\}\,. \end{array}$$

We want to choose a basis whose functions have as small support as possible so that the resulting coefficient matrix is sparse; that is, they are nonzero over as small a portion of the domain as possible and zero elsewhere. For $1 \le i \le N$ we

consider again the "hat" functions

$$\phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h_i} & \text{for } x_{i-1} \le x \le x_i \\ \dfrac{x_{i+1} - x}{h_{i+1}} & \text{for } x_i \le x \le x_{i+1} \\ 0 & \text{elsewhere.} \end{cases} \tag{7.15}$$

Clearly $\phi_i(x) \in V_0^h$ for $1 \le i \le N$. Moreover, we easily see that

$$\phi_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{7.16}$$

for $1 \le i \le N$ and $0 \le j \le N+1$. We call this type of basis function a **nodal basis** because it is one at a single node and zero at all other grid points.

The following proposition justifies our intuition that the functions defined in (7.15) form a basis for V_0^h.

Proposition 7.1. *The functions $\{\phi_i(x)\}_{i=1}^N$ defined in (7.15) form a basis for V_0^h.*

Proof. We must show that $\{\phi_i(x)\}$, $i = 1, \ldots, N$ are linearly independent and span the space V_0^h. To see that we have a linearly independent set, let $\psi(x) = \sum_{i=1}^N c_i \phi_i(x)$; we want to show that the only way $\psi(x) = 0$ for all x is if $c_i = 0$ for $i = 1, \ldots, N$. Using (7.16), we see that $\psi(x_i) = c_i$ for $1 \le i \le N$. Thus if $\psi(x) = 0$ for all x we have that $c_i = 0$ for $i = 1, \ldots, N$; in addition if $c_i = 0$ for all $1 \le i \le N$ then the nodal values of ψ are zero and since it is piecewise linear, it is zero everywhere. Hence we conclude that the functions are linearly independent. To show that the set spans V_0^h we let $\psi(x)$ be an arbitrary element of V_0^h and show that we can write $\psi(x)$ as a linear combination of the $\phi_i(x)$, $i = 1, \ldots, N$; i.e., $\psi(x) = \sum_{i=1}^N c_i \phi_i(x)$. But this can be done by letting $c_i = \psi(x_i)$, i.e., setting c_i to be the *nodal values* of ψ.

Once we have chosen a basis for V_0^h, the problem (7.13) reduces to solving a system of N algebraic equations in N unknowns. Since $u^h \in V_0^h$, we let $u^h(x) = \sum_{j=1}^N c_j \phi_j(x)$ and write (7.13) as

$$\sum_{j=1}^N c_j \int_0^1 p(x)\phi_i'(x)\phi_j'(x)\,dx + \sum_{j=1}^N c_j \int_0^1 q(x)\phi_i(x)\phi_j(x)\,dx = \int_0^1 f\phi_i\,dx$$

for $1 \le i \le N$. Recall that previously we demonstrated that testing the equation against each $v^h \in V_0^h$ was equivalent to testing against each element of the basis of V_0^h. Then $\mathbf{c} = (c_1, c_2, \ldots, c_N)^T$ satisfies the matrix system

$$\mathcal{A}\mathbf{c} = \mathbf{b}\,, \tag{7.17}$$

where $\mathbf{b} = \left(\int_0^1 f\phi_1\,dx, \int_0^1 f\phi_2\,dx, \ldots, \int_0^1 f\phi_N\,dx\right)^T$ and $\mathcal{A}$ is the $N \times N$ matrix whose elements are given by

$$\mathcal{A}_{ij} = \int_0^1 p\phi_j'\phi_i'\,dx + \int_0^1 q\phi_j\phi_i\,dx$$

or

$$\mathcal{A}_{ij} = \mathcal{S}_{ij} + \mathcal{M}_{ij}$$

with $\mathcal{S}_{ij} = \int_0^1 p\phi_j'\phi_i'\ dx$ and $\mathcal{M}_{ij} = \int_0^1 q\phi_j\phi_i\ dx$. The matrix $\mathcal{A}$ is symmetric, positive definite and tridiagonal. If $p(x) = q(x) = 1$ on $[0,1]$ and we use a uniform mesh with continuous piecewise linear basis functions then the integrals can be computed exactly; in this case the matrices $\mathcal{S}$ and $\mathcal{M}$ are explicitly given by

$$\mathcal{S} = \frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ & & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{pmatrix} \tag{7.18}$$

and

$$\mathcal{M} = \frac{h}{6}\begin{pmatrix} 4 & 1 & 0 & & \cdots & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 \\ & & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & 1 & 4 & 1 \\ 0 & \cdots & & 0 & 1 & 4 \end{pmatrix}. \tag{7.19}$$

Note that $\mathcal{S}$ is the same coefficient that we got when we discretized the equation $-u''(x) = f$ using the second centered finite difference quotient if we multiply through by h. In the case $p = 1$ the matrix $\mathcal{S}$ is called the *stiffness matrix* of the basis $\{\phi_i\}_{i=1}^N$ while in the case $q = 1$, the matrix $\mathcal{M}$ is called the *Gram matrix* or the *mass matrix* associated with the basis $\{\phi_i\}_{i=1}^N$.

7.2.4 Error estimates

We would like to bound the error between the solution $u(x)$ of the weak problem and $u^h(x)$ the solution of the discrete weak problem. However, the natural estimate that comes out is a bound in the error in the derivatives $u'(x) - (u^h)'(x)$. Extra work has to be done to estimate the error in the solution itself rather than the derivative. In addition, the goal is to measure the error in terms of powers of h because we have seen that this is useful. To do this we will need some results from interpolation theory.

For simplicity of exposition let's prove the error estimate for the case where $p(x) = 1$ and $q(x) = 0$; this will just make the steps clearer. First recall that the continuous weak problem

$$\int_0^1 u'(x)v'(x)\ dx = \int_0^1 fv(x)\ dx \quad \forall v \in V_0$$

holds for every $v \in V_0$ and because $v^h \in V_0^h \subset V_0$ the equation has to hold for

every $v^h \in V_0^h$, i.e.,

$$\int_0^1 u'(x)(v^h)'(x)\,dx = \int_0^1 fv^h\,dx \quad \forall v^h \in V_0^h\,.$$

The reason we write this equation like this is so that we can combine it with the discrete weak problem

$$\int_0^1 (u^h)'(x)(v^h)'(x)\,dx = \int_0^1 fv^h\,dx \quad \forall v^h \in V_0^h\,.$$

Subtracting these two equations gives

$$\int_0^1 \big[u'(x) - (u^h)'(x)\big](v^h)'(x)\,dx = 0 \quad \forall v^h \in V_0^h\,.$$

Note that $u'(x) - (u^h)'(x)$ is the error in the derivative of the solution so that this equation says the derivative of the error $E = u(x) - u^h(x)$ is orthogonal to all elements in V_0^h in this sense, i.e.,

$$\int_0^1 \big[u'(x) - (u^h)'(x)\big](v^h)'(x)\,dx = \int_0^1 \frac{dE}{dx}\frac{dv^h}{dx}\,dx = 0 \quad \forall v^h \in V_0^h \qquad (7.20)$$

and thus it is called the *orthogonality condition* for the error.

We now use this orthogonality condition and properties of integrals to prove an error estimate. We have

$$\begin{aligned}\int_0^1 \big[u'(x) - (u^h)'(x)\big]^2\,dx &= \int_0^1 \big[u'(x) - (u^h)'(x)\big]u'(x)\,dx \\ &\quad - \int_0^1 \big[u'(x) - (u^h)'(x)\big](u^h)'(x)\,dx \\ &= \int_0^1 \big[u'(x) - (u^h)'(x)\big]\big[u'(x) - (w^h)'(x)\big]\,dx \\ &\quad - \int_0^1 \big[u'(x) - (u^h)'(x)\big]\big[(u^h)'(x) - (w^h)'(x)\big]\,dx\end{aligned}$$

where for the last step we have added and subtracted a term containing an arbitrary element $w^h \in V_0^h$. From the orthogonality condition (7.20) we see that the second term is zero where we set $v^h = u^h - w^h \in V_0^h$. We will denote the norm on $L^2([0,1])$ by $\|\cdot\|_0$ where

$$\|v\|_0 = \Big[\int_0^1 (v(x))^2\,dx\Big]^{1/2}.$$

We have

$$\|u'(x) - (u^h)'(x)\|_0^2 = \left|\int_0^1 \big[u'(x) - (u^h)'(x)\big]\big[u'(x) - (w^h)'(x)\big]\,dx\right|.$$

There is a useful inequality called the Cauchy-Schwartz inequality which is

$$\left|\int_\Omega uv\ d\Omega\right| \leq \left[\int_\Omega u^2\ d\Omega\right]^{1/2}\left[\int_\Omega v^2\ d\Omega\right]^{1/2} = \|u\|_0\,\|v\|_0\ . \tag{7.21}$$

Using this inequality yields

$$\begin{aligned}\|u'(x)-(u^h)'(x)\|_0^2 &= \left|\int_0^1 \left[u'(x)-(u^h)'(x)\right]\left[u'(x)-(w^h)'(x)\right]\,dx\right| \\ &\leq \|u'(x)-(u^h)'(x)\|_0\,\|u'(x)-(w^h)'(x)\|_0\end{aligned}$$

and thus

$$\|u'(x)-(u^h)'(x)\|_0 \leq \|u'(x)-(w^h)'(x)\|_0 \quad \forall w^h \in V_0^h\ .$$

Now this has to hold for every $w^h \in V_0^h$ so it has to be true for the particular w^h whose derivative is closest to the derivative of the solution to the weak problem. We write this as

$$\|u'(x)-(u^h)'(x)\|_0 \leq \inf_{\chi^h\in V_0^h}\|u'(x)-(\chi^h)'(x)\|_0\,. \tag{7.22}$$

This results says that the $L_2(0,1)$ error in the derivative of the error is less than or equal to a constant times the *best approximation* to $u'(x)$ in the space V_0^h. However, our goal was to estimate the error in terms of h. It turns out that it is not easy to estimate the difference in a function and its best approximation; however there are readily available estimates for how well you can approximate a function with its piecewise interpolant. How can this help us? We have that for any $w^h \in V_0^h$

$$\|u'(x)-(u^h)'(x)\|_0 \leq \inf_{\chi^h\in V_0^h}\|u'(x)-(\chi^h)'(x)\|_0 \leq \|u'(x)-(w^h)'(x))\|_0$$

because $\inf_{\chi^h\in V_0^h}\|u'(x)-(\chi^h)'(x)\|_0$ is the smallest value for all $\chi^h \in V_0^h$ and w^h is just some other element of V_0^h. So if we have an estimate for a particular w^h then we can use these.

Recall that one way to approximate a function is to use a polynomial interpolant; i.e., to find a polynomial which agrees with the given function or its derivatives at a set of points. One such example is a *Lagrange interpolant* which interpolates given data or function values. Due to the fact that one cannot guarantee that the norm of the difference in the function and the Lagrange interpolating polynomial approaches zero as the degree of the polynomial increases, one typically uses piecewise polynomial interpolation. In piecewise Lagrange interpolation we put together Lagrange polynomials of a fixed degree to force them to interpolate the given function values or data. For example, a continuous piecewise linear Lagrange polynomial is a continuous function which is a linear polynomial over each subinterval and matches the given function at the nodes. Clearly, a piecewise linear Lagrange polynomial over the subdivision of $[0,1]$ given in (7.14) which is zero at $x=0$ and $x=1$ is an element of V_0^h.

We state the estimates for the error in a function or its derivatives and its V^h-interpolant where V^h is the space of continuous piecewise linear functions defined over the given partition with no boundary conditions imposed; i.e.,

$$V^h = \{\phi(x) \in C[0,1] \mid \phi(x) \text{ linear on } [x_i, x_{i+1}] \text{ for } 0 \le i \le N\}\,. \tag{7.23}$$

Then these results also hold for $V_0^h \subset V^h$. If $v(x)$ is a continuous function on $[0,1]$ then we can find a unique element which agrees with $v(x)$ at each of the points x_i, $i = 0, \ldots, N+1$; we call this element of V^h the V^h-interpolant of v and denote it by $I^h v$. Once we have the standard estimate for the approximation of a function by its piecewise linear Lagrange interpolant, then we can use it to obtain an estimate in terms of powers of h. The following lemma gives standard results for approximating a function or its derivatives by its piecewise linear Lagrange interpolant in the L_2 norm.

Lemma 7.1. *Let $f \in C^2(0,1)$ and $V^h \subset V$ be defined by (7.23); let $I^h f$ denote the V^h-interpolant of f. Then there exists positive constants C_1 and C_2, independent of h and f, such that*

$$\left\|f - I^h f\right\|_0 \le C_1 h^2 \left\|f''\right\|_0 \tag{7.24}$$

and

$$\left\|(f - I^h f)'\right\|_0 \le C_2 h \left\|f''\right\|_0 . \tag{7.25}$$

Note that the error in the function and its interpolant is one degree higher in h than the error in the derivative. This is true in general and a way to think about this is to recall that the definition of the first derivative has an h in the denominator just like our forward, backward or centered differences for the first derivative; hence you lose a power of h when you approximate the derivative. The other thing to notice about the lemma is that these *optimal* error estimates hold only when the function we are approximating has two derivatives. By optimal we mean that they are the best that can be obtained; no matter how smooth the function is, this is the best accuracy one can obtain. Recall that our weak formulation only requires one derivative on the solution whereas this lemma requires two. This does not have anything to do with finite elements but rather is a result of approximation theory. In fact, if you choose a solution which doesn't have two derivatives then we don't get the optimal rates of convergence; we will see this when we look at some numerical examples.

Now we have the result

$$\|u'(x) - (u^h)'(x)\|_0 \le \|u'(x) - I^h(u'(x))\|_0 \le h\|u''\|_0 \tag{7.26}$$

if $u \in C^2(0,1)$ and V^h is the space of continuous piecewise linear functions over the given partition; thus we say the error in the derivatives is linear in h. It turns out that the error in the solution itself is optimal as long as $u''(x) \in C(0,1)$, i.e.,

$$\|u(x) - u^h(x)\|_0 \le h^2 \|u''(x)\|_0 .$$

Thus the error in the solution itself is quadratic in h. It takes more work to prove this rigorous but it can be done by something called "Nitsche's trick" and using regularity of the differential equation. However we will not prove the result here.

We have now completed our analysis of a finite element solution of (7.11) using continuous, piecewise linear polynomials. Before turning our attention to implementing the method to obtain some numerical results we consider approximating using higher degree polynomials and then remind ourselves how the entries in the matrix and right-hand side of (7.17) are obtained.

7.2.5 Approximation using higher degree polynomials

From the error estimate (7.26) we see that the rate of convergence is linear in h if we measure the error in the derivatives of the solution. If we want our calculations to converge at a higher rate, such as quadratically, then we have to choose a higher degree polynomial for our approximating space V_0^h. In this section we give some general results for the error in the interpolating polynomial for a kth degree polynomial and then use these to state optimal error estimates for our problem. We also consider a basis for quadratic polynomials and the structure of the resulting linear system which is no longer tridiagonal as it was when we used linear polynomials. The case of continuous, cubic polynomials is left to the exercises.

We now define V^h to be the space of continuous, piecewise polynomials of degree k or less over the partition of $[0,1]$ defined in (7.14), i.e.,

$$V^h = \{\phi(x) \mid \phi \in C[0,1], \phi(x) \text{ polynomial of degree} \leq k \text{ on } [x_i, x_{i+1}] \text{ for } 0 \leq i \leq N\}. \tag{7.27}$$

V_0^h is defined in the same way except we require $\phi(x)$ to be zero at the endpoints;

$$V_0^h = \{\phi(x) \mid \phi \in C[0,1], \phi(x) \text{ polynomial of degree} \leq k \text{ on } [x_i, x_{i+1}] \text{ for } 0 \leq i \leq N, \phi(0) = \phi(1) = 0\}. \tag{7.28}$$

A result for the V^h-interpolant of functions and their derivatives for piecewise polynomials of degree k is provided in the following lemma. The situation where $k = 1$, i.e., we are using linear polynomials, is a special case of this result.

Lemma 7.2. *Let $f \in C^{k+1}(0,1)$ and V^h is defined by (7.27); let $I^h f$ denote the V^h-interpolant of f. Then there exists positive constants C_1, C_2, independent of h and f, such that*

$$\left\|f - I^h f\right\|_0 \leq C_1 h^{k+1} \left\|f^{[k+1]}\right\|_0 \tag{7.29}$$

and

$$\left\|(f - I^h f)'\right\|_0 \leq C_2 h^k \left\|f^{[k+1]}\right\|_0. \tag{7.30}$$

Note that (7.29) reduces to (7.24) and (7.30) reduces to (7.25) when $k = 1$. These are the best rates of convergence possible with a kth degree polynomial. If f is not smooth enough then there is a loss in the rates of convergence. If our finite element solution is sufficiently smooth then the optimal rates of convergence are given in the following theorem.

Theorem 7.1. *Let $u \in C^{k+1}(0,1)$ be the solution of (7.12) and let u^h be the Galerkin approximation of u in the space V_0^h defined by (7.28) satisfying (7.13). Then there exists positive constants C_1, C_2, independent of u, h, or u^h such that*

$$\left\|(u-u^h)'\right\|_0 \leq C_1 h^k \left\|u^{[k+1]}\right\|_0 . \tag{7.31}$$

and

$$\left\|u-u^h\right\|_0 \leq C_2 h^{k+1} \left\|u^{[k+1]}\right\|_0 . \tag{7.32}$$

We note that this estimate says that if the solution is sufficiently smooth, then increasing the degree of the polynomial by one increases the rate of convergence by one. So when we use continuous piecewise quadratics for V^h then we expect the optimal rate of convergence for u to be h^3 and for its derivative to be h^2.

We now turn to the concrete problem of finding a basis for V^h or V_0^h when we choose quadratic polynomials, i.e., $k = 2$. In this case we know that the rates of convergence are $\mathcal{O}(h^2)$ for the L_2 norm of the derivative of the error and $\mathcal{O}(h^3)$ in the L_2 norm of the error, if the solution is sufficiently smooth. We use the same partition of $[0,1]$ as before, i.e., that given in (7.14). The problem now is that over each element $[x_{i-1}, x_i]$ the basis function must be a quadratic; however, it takes three points to uniquely determine a quadratic. To this end, we add a node in each subinterval; the easiest thing to do is add a node at the midpoint of each subinterval, $x_{i-\frac{1}{2}} = (x_{i-1}+x_i)/2$. We still have $N+1$ elements, but now have the $N+2$ points from the endpoints of the intervals plus the $N+1$ midpoints giving a total of $2N+3$ points. Analogous to the continuous, piecewise linear case, we expect that a basis for V^h for $k = 2$ consists of $2N+3$ members and for V_0^h we don't need the endpoints so we have $2N+1$ vectors in a basis.

For simplicity of exposition, we renumber our $2N+3$ nodes as x_i, $i = 0, \ldots, 2N+2$. However, we must remember that the elements are $[x_{2j-2}, x_{2j}]$ for $j = 1, \ldots, N+1$. To determine a nodal basis for V^h we require each ϕ_i in the basis to have the property that it is one at node x_i and zero at all other nodes. In the basis for piecewise linear polynomials we were able to make the support of the basis functions to be two adjacent elements; the same is true in this case. However, now we have two different formulas for the basis functions determined by whether the function is centered at an endpoint of an interval or the midpoint.

To easily get an idea what these quadratic functions look like, we first write the polynomials on $[-1,1]$ with nodes $x = -1, 0, 1$; we can then translate them to the desired interval. From these we can determine the shape of our basis functions. For the quadratic function which is one at the midpoint of $[-1,1]$, i.e., at $x = 0$, and zero at $x = \pm 1$ we have $\phi(x) = 1 - x^2$. For the quadratic function which is one at $x = -1$ and zero at $x = 0, 1$ we have $\phi(x) = \frac{1}{2}(x^2 - x)$. Similarly for the quadratic function which is one at $x = 1$ and zero at $x = -1, 0$ we have $\phi(x) = \frac{1}{2}(x^2 + x)$. These functions are illustrated in Figure 7.3 and have the same shape as the ones on $[x_{2j-2}, x_{2j}]$. We can splice together the two functions centered at the endpoints of the interval to get a complete picture of the basis function centered at an endpoint which has support over two intervals; this is demonstrated in the right plot in Figure 7.3. Note that analogous to the case of continuous

piecewise linear polynomials the quadratic basis functions will be continuous but not continuously differentiable at the grid points.

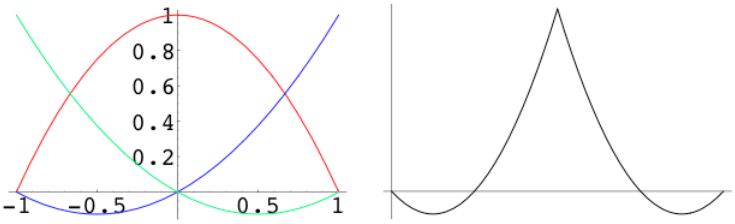


Figure 7.3: Plot on left shows nodal quadratic functions on $[-1,1]$ and plot on right shows shape of quadratic basis function centered at endpoint of an interval having support over two intervals.

To find the analogous polynomials on $[x_{2j-2}, x_{2j}]$ we need to translate our functions on $[-1,1]$ to the desired interval or equivalently solve linear systems. For example, a straightforward way to find the quadratic which is one at x_{2j-1} and zero at the endpoints is to solve

$$\begin{aligned} 0 &= a + b\big(x_{2j-2}\big) + c\big(x_{2j-2}\big)^2 \\ 1 &= a + b\big(x_{2j-1}\big) + c\big(x_{2j-2}\big)^2 \\ 0 &= a + b\big(x_{2j}\big) + c\big(x_{2j}\big)^2 . \end{aligned}$$

However, there are more efficient approaches to finding basis functions. The support of the quadratic basis functions for V_0^h on a uniform partition of $[0,2]$ with $h = 0.5$ are illustrated in Figure 7.4.

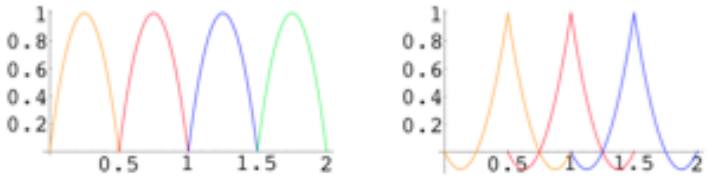


Figure 7.4: Support of quadratic basis functions on the uniform partition of $[0,2]$ with $h = .5$ assuming homogeneous Dirichlet boundary conditions.

We have seen that once a basis for the finite dimensional space is chosen, the discrete problem can be converted to solving a linear system of equations. The (i,j) entry of the coefficient matrix $\mathcal{A}$ is given by the same expression as in the case of piecewise linear functions except we are using a different basis; specifically, we have

$$\mathcal{A}_{ij} = (p\phi_j', \phi_i') + (q\phi_j, \phi_i)$$

where ϕ_i is now a quadratic polynomial. We recall that when the standard "hat" functions were used as a basis for V_0^h the resulting matrix was $N \times N$, symmetric,

positive definite and tridiagonal. In the case of our quadratic basis functions in V_0^h, the matrix is still symmetric and positive definite but we note that the size of our matrix has increased to $2N+1$. Also, it is no longer tridiagonal. To determine the bandwidth of the matrix, we need to ascertain where the zero entries begin in each row. We return to Figure 7.4 and note that for a basis function ϕ_i centered at a midpoint node x_i, the integral $\int_0^1 \phi_i \phi_j \, dx$ is zero when $j > i+1$ or $j < i-1$, i.e., outside of the interval; the same is true for the term $\int_0^1 \phi_i' \phi_j' \, dx$. However, for a basis function ϕ_i centered at the right endpoint node x_i, the integral $\int_0^1 \phi_i \phi_j \, dx$ is potentially nonzero in that interval and the next which includes a total of five basis functions, counting itself. Thus the integral is zero when $j > i+2$ or $j < i-2$ and the maximum bandwidth of the matrix is five. This system can be efficiently solved by a direct method such as a banded Cholesky algorithm or an iterative method such as conjugate gradient or one of its variants.

If we desire to have a method where the error in the derivative converges cubically, then we can choose continuous, piecewise cubic polynomials for V^h. Because we need four points to uniquely determine a cubic, we add two points to each interval in our original partition given in (7.14). For V_0^h we now have $N+2(N+1) = 3N+2$ points and we expect that this is the dimension of the space and thus the dimension of the resulting matrix.

7.2.6 Numerical quadrature

If we are implementing our example given in (7.11) in the case $p = q = 1$ with continuous, piecewise linear polynomials for V_0^h and where we are using a uniform grid, then (7.18) and (7.19) explicitly give the coefficient matrices. However, entries in the right-hand side of (7.17) must be computed and also entries for the coefficient matrix for general p, q. For some choices of f we could evaluate the integrals exactly. However, if we want to write a general finite element program then we should be able to do problems where the integrals can not be evaluated exactly. In this case, we must use *quadrature rules* to approximate the integrals. Recall that in our error analysis, we have assumed that the integrals are computed exactly. For now, we present some widely used quadrature formulas in one-dimension and give general rules for choosing a formula.

In numerical integration we approximate the integral by the sum of the integrand evaluated at a prescribed set of points multiplied by weights; i.e.,

$$\int_a^b f(x) \, dx \approx \sum_k f(q_k) w_k \,, \tag{7.33}$$

where q_k represent the quadrature points and w_k the quadrature weights. Of particular interest in one dimension are the Gauss quadrature rules; in these rules the quadrature points and weights are chosen so that the rule integrates exactly as high a degree polynomial as possible. Specifically, if we use n Gaussian quadrature points then the rule integrates polynomials of degree $2n-1$ exactly. The Gaussian quadrature rule for one point is the well known midpoint rule which integrates lin-

ear functions exactly. The following table gives the Gaussian quadrature points and weights on the interval $[-1, 1]$.

Table 7.1: Gauss quadrature formulas on $[-1, 1]$

n	nodes	weights
1	0.0000000000	2.0000000000
2	$\pm\frac{1}{\sqrt{3}} = \pm 0.5773502692$	1.0000000000
3	± 0.7745966692	0.5555555556
	0.0000000000	0.8888888889
4	± 0.8611363116	0.3478548451
	± 0.3399810436	0.6521451549
5	± 0.9061798459	0.2369268850
	± 0.5384693101	0.4786286701
	0.0000000000	0.5688888889

If the domain of integration is different from $(-1, 1)$, then a change of variables is needed. For example, to compute the integral $\int_a^b f(\hat{x})\ d\hat{x}$ we use the linear mapping $\hat{x} = a + \frac{b-a}{2}(x + 1)$ to map to the integral over $(-1, 1)$. In this case we have

$$\int_a^b f(\hat{x})\ d\hat{x} = \frac{b-a}{2} \int_{-1}^{1} f\Big(a + \frac{b-a}{2}(x+1)\Big)\ dx\,.$$

Then we apply the quadrature rule to the integral over $(-1, 1)$. Note that we have just modified the quadrature weight by multiplying by $\frac{b-a}{2}$ and mapping the quadrature point to the interval (a, b).

When choosing a quadrature rule, we want to use as low a degree rule as possible for efficiency but as high a degree rule as necessary for accuracy. It is not necessary to evaluate the integrals exactly, even if this is possible; however, we must assure that the error in the numerical quadrature does not contaminate the power of h accuracy in our estimate. When using piecewise linear polynomials for the finite element space in one-dimension for the problem (7.11), it is adequate to use a one-point Gauss quadrature rules; for piecewise quadratic polynomials a two-point rule is adequate.

7.2.7 Computational examples

In this section we implement two specific examples of the boundary value problem given in (7.11) where we know the exact solution so that errors and rates of convergence can be calculated. These problems differ in the choice of p, q and f. The choice of f is especially important because a lack of smoothness in f results in the solution not being smooth enough to guarantee the optimal rates of convergence. In all computations we use continuous, piecewise polynomials on a uniform grid, an appropriate Gauss quadrature rule to evaluate the integrals in the coefficient matrix and the right-hand side, and a direct solver for the linear system. For the error computation we use a higher order quadrature rule to evaluate the integrals. The

reason for the higher order rule in the error computation is to make absolutely sure that no error from the numerical integration contaminates the calculation of the error. The computations are performed using $h = 1/4, 1/8, 1/16$, and $1/32$ with linear, quadratic and cubic elements; the $L_2 2$ norm of the error in the unknown and its derivative are computed for each grid.

For each example we are interested in calculating the numerical rate of convergence and comparing it with the theoretical results. The errors for each grid can be used to compute an approximate rate of convergence. For example, we have $\left\|u - u^h\right\| \approx Ch^r$ where we expect r to approach some value as the grid size decreases. If we have the error, E_i, on two separate meshes then we have that $E_1 \approx Ch_1^r$ and $E_2 \approx Ch_2^r$ where E_1 and E_2 represent $\left\|u - u^h\right\|$ on the grid with mesh spacing h_1 and h_2, respectively. If we solve for C and set the two relationships equal, we have $E_1/h_1^r \approx E_2/h_2^r$; solving for r we obtain

$$r \approx \frac{\ln E_1/E_2}{\ln h_1/h_2}\,. \tag{7.34}$$

We note that if the grid spacing is halved, i.e., $h_2 = h_1/2$ then the error should be approximately decreased by $\left(\frac{1}{2}\right)^r$ since $E_2 \approx \left(\frac{h_2}{h_1}\right)^r E_1$. This implies that if $r = 1$ the error is approximately halved when the grid spacing is halved; if the rate is two, then the error is reduced by a factor of one-fourth when the grid spacing is halved, etc.

Example 1. We first consider the problem

$$\begin{array}{rcl} -u'' + \pi^2 u &=& 2x\pi^2 \sin \pi x - 2\pi \cos \pi x \quad \text{for } 0 < x < 1 \\ u(0) = u(1) &=& 0\,, \end{array} \tag{7.35}$$

whose exact solution is given by $u = x \sin \pi x$. Since our solution $u(x) = x \sin \pi x$ possesses all derivatives we expect the optimal rates of convergence; in particular if we use continuous, piecewise linear polynomials then the rate r, calculated from (7.34), should approach two as $h \to 0$ for the L_2-norm of the error itself and approach one for the L_2 norm of the derivative of the error. These values for r are calculated in Table 7.2 along with the errors and rates using continuous, piecewise quadratic and cubic polynomials; in the table we computed the rate using the errors at $h = 1/4$ and $1/8$, at $h = 1/8$ and $1/16$, and at $h = 1/16$ and $h = 1/32$. Note that, in fact, $r \to 1$ for the error in the derivatives and $r \to 2$ in the L_2-error as predicted when piecewise linear polynomials are used; the optimal rates for quadratic and cubic polynomials are also obtained. In these calculations we used a one-point Gauss rule for linear polynomials, a two-point Gauss rule for quadratic polynomials, and a three-point Gauss rule for cubic polynomials. In Table 7.3 we illustrate what happens if we use continuous quadratic polynomials using a one-point, a two-point and a three-point Gauss quadrature rule. Note that the rates of convergence using a two-point and a three-point rule are essentially the same, but when we use the one-point rule the results are meaningless.

Example 2. The next problem we want to consider is

$$\begin{array}{rcl} -u'' &=& -\alpha(\alpha-1)x^{\alpha-2} \quad \text{for } 0 < x < 1 \\ u(0) = u(1) &=& 0\,, \end{array} \tag{7.36}$$

where $\alpha > 0$; the exact solution u is given by $u(x) = x^\alpha - x$. The results for various values of α are presented in Table 7.4 using continuous, piecewise linear polynomials and a one-point Gauss quadrature rule. Recall that the optimal rates in this case are $\mathcal{O}(h)$ in the L^2 norm of the

Table 7.2: Numerical results for Example 7.2.7 using continuous, piecewise linear polynomials.

p^k	h	$\left\|(u-u^h)'\right\|_2$	rate	$\left\|u-u^h\right\|_2$	rate
linear	$1/4$	0.47700		0.28823×10^{-1}	
linear	$1/8$	0.23783	1.0041	0.69831×10^{-2}	2.0459
linear	$1/16$	0.11885	1.0007	0.17313×10^{-2}	2.0120
linear	$1/32$	0.059416	1.0002	0.43199×10^{-3}	2.0028
quadratic	$1/4$	0.49755×10^{-1}		0.15707×10^{-2}	
quadratic	$1/8$	0.12649×10^{-1}	1.9758	0.20227×10^{-3}	2.9570
quadratic	$1/16$	0.31747×10^{-2}	1.9940	0.25553×10^{-4}	2.9847
quadratic	$1/32$	0.79445×10^{-3}	1.9986	0.32031×10^{-5}	2.9960
cubic	$1/4$	0.51665×10^{-2}		0.10722×10^{-3}	
cubic	$1/8$	0.64425×10^{-3}	3.003	0.67724×10^{-5}	3.985
cubic	$1/16$	0.80496×10^{-4}	3.001	0.42465×10^{-6}	3.9953
cubic	$1/32$	0.10061×10^{-4}	3.000	0.26564×10^{-7}	3.9987

derivatives and $\mathcal{O}(h^2)$ in the L_2 norm of the function. Note that for $\alpha = 7/3$ we get the optimal rates of convergence. However, for $\alpha = 4/3$ we have less than optimal rates and for $\alpha = 1/3$ the error in the derivative is almost constant and the rate in the L_2-norm of the error is less than one. Of course, the reason for this is that when $\alpha = 3/2$ the exact solution $u = x^{4/3} - x$ does not possess two continuous derivatives on $[0,1]$ and when $\alpha = 1/3$ the exact solution $u = x^{1/3} - x$ does not possess even one derivative on $[0,1]$. Thus the interpolation results (7.24) and (7.25) do not hold and we can't expect to get optimal rates of convergence.

7.3 A two-point BVP with Neumann boundary data

In this section we consider the same prototype two point BVP as before but now we impose Neumann boundary data instead of homogeneous Dirichlet data. In particular we seek a function $u(x)$ satisfying

$$\begin{aligned} -\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u &= f(x) \quad \text{for } 0 < x < 1 \\ u'(0) &= 0 \\ u'(1) &= \alpha\,. \end{aligned} \tag{7.37}$$

As before, p and q are bounded functions on $[0,1]$ satisfying $0 < p_{min} \le p(x) \le p_{max}$ but now we impose $0 < q_{min} \le q(x) \le q_{max}$ for all $x \in [0,1]$. Again if $f, q \in C[0,1]$ and $p \in C^1[0,1]$ the boundary value problem (7.37) possesses a unique *classical solution* $u(x) \in C^2(0,1)$ which satisfies (7.37) for every $x \in [0,1]$. Note that here we require that $q_{min} > 0$ to guarantee a unique solution; this is because if $q = 0$ and u satisfies (7.37) then so does $u + C$ for any constant C.

In this case we call our underlying finite element space V because we have no

Table 7.3: Numerical results for Example 7.2.7 using continuous, piecewise quadratic polynomials with three different quadrature rules.

Gauss Quadrature Rule	h	$\left\|(u-u^h)'\right\|_2$	rate	$\left\|u-u^h\right\|_2$	rate
one-point	1/4	8.885		0.3904	
one-point	1/8	18.073		0.3665	
one-point	1/16	36.391		0.3603	
one-point	1/32	72.775		0.3587	
two-point	1/4	0.49755×10^{-3}		0.15707×10^{-4}	
two-point	1/8	0.12649×10^{-3}	1.9758	0.20227×10^{-5}	2.9570
two-point	1/16	0.31747×10^{-4}	1.9940	0.25553×10^{-6}	2.9847
two-point	1/32	0.79445×10^{-5}	1.9986	0.32031×10^{-7}	2.9960
three-point	1/4	0.49132×10^{-3}		0.18665×10^{-4}	
three-point	1/8	0.12109×10^{-3}	1.9620	0.24228×10^{-5}	2.9456
three-point	1/16	0.31724×10^{-4}	1.9911	0.30564×10^{-6}	2.9868
three-point	1/32	0.79430×10^{-5}	1.9978	0.38292×10^{-7}	2.9967

boundary conditions to impose on the space. The weak formulation is

$$\left\{\begin{array}{l}\text{seek } u\in V \text{ satisfying}\\ \quad \int_0^1 p(x)u'(x)v'(x)\,dx+\int_0^1 q(x)u(x)v(x)\,dx=\int_0^1 fv\,dx+\alpha p(1)v(1)\quad \forall v\in V\,,\end{array}\right. \tag{7.38}$$

Clearly, if $u(x)$ satisfies the classical problem (7.37), then $u(x)$ satisfies (7.38) because

$$\begin{aligned}\int_0^1 f(x)v\,dx &= \int_0^1 \big(-(p(x)u'(x))'+q(x)u(x)\big)v(x)\,dx\\ &= -pu'v\big|_0^1+\int_0^1 p(x)u'(x)v'(x)\,dx+\int_0^1 q(x)u(x)v(x)\,dx\\ &= -p(1)u'(1)v(1)+p(0)u'(0)v(0)+\int_0^1 p(x)u'(x)v'(x)\,dx+\int_0^1 q(x)u(x)v(x)\,dx\\ &= \int_0^1 p(x)u'(x)v'(x)\,dx+\int_0^1 q(x)u(x)v(x)\,dx-\alpha p(1)v(1)\,,\end{aligned}$$

where we have imposed the homogenous Neumann boundary condition $u'(0)=0$ and the inhomogeneous condition $u'(1)=\alpha$. Note that these boundary conditions are not imposed on the space, but rather on the weak formulation; these are called *natural* boundary conditions.

If we want to seek an approximation to $u(x)$ in the space of continuous, piecewise linear functions defined over the subdivision (7.14) then we cannot use the space V_0^h defined in (7.15) since this space was designed to approximate functions in V_0. Instead we consider V^h where

$$V^h=\{\phi(x)\in C[0,1],\phi(x)\ \text{ linear on } (x_i,x_{i+1}) \text{ for } 0\le i\le N\}\,. \tag{7.39}$$

Table 7.4: Numerical results for Example 7.2.7.

α	h	$\left\|(u-u^h)'\right\|_2$	rate	$\left\|u-u^h\right\|_2$	rate
7/3	1/4	0.1747		0.17130×10^{-1}	
7/3	1/8	0.08707	1.0046	0.33455×10^{-2}	1.9726
7/3	1/16	0.04350	1.0012	0.84947×10^{-3}	1.9776
7/3	1/32	0.02174	1.0007	0.21495×10^{-3}	1.9826
4/3	1/4	0.47700		0.28823×10^{-1}	
4/3	1/8	0.23783	0.7690	0.69831×10^{-2}	1.8705
4/3	1/16	0.11885	0.7845	0.17313×10^{-2}	1.8834
4/3	1/32	0.059416	0.7965	0.43199×10^{-3}	1.8005
1/3	1/4	0.43332		0.14594	
1/3	1/8	0.43938		0.10599	0.4615
1/3	1/16	0.46661		0.07922	0.4200
1/3	1/32	0.50890		0.06064	0.3857

Similar to the homogeneous Dirichlet case, it can be shown that V^h is an $N+2$ dimensional subspace of V; a basis for V^h is given by the standard "hat" functions that we used for V_0^h along with one defined at each endpoint. Specifically, we have the functions ψ_i, $i = 1, \ldots, N+2$ defined by

$$\psi_i(x) = \begin{cases} \phi_0(x) & \text{for } j = 1 \\ \phi_{i-1}(x) & \text{for } 2 \le i \le N+1 \\ \phi_{N+1}(x) & \text{for } j = N+2 \end{cases} \tag{7.40}$$

where $\phi_i(x)$, $i = 1, \ldots N$ are given by (7.15) and

$$\phi_0(x) = \begin{cases} \dfrac{x_1 - x}{h_1} & \text{for } 0 \le x \le x_1 \\ 0 & \text{elsewhere} \end{cases} \tag{7.41}$$

and

$$\phi_{N+1}(x) = \begin{cases} \dfrac{x - x_N}{h_{N+1}} & \text{for } x_N \le x \le 1 \\ 0 & \text{elsewhere}\,. \end{cases} \tag{7.42}$$

The discrete weak problem is

$$\begin{cases} \text{seek } u^h \in V^h \text{ satisfying} \\ \quad \int_0^1 p(x)(u^h)'(x)(v^h)'(x)\,dx + \int_0^1 q(x)u^h(x)v^h(x)\,dx = \int_0^1 fv^h\,dx + \alpha p(1)v^h(1) \\ \qquad \forall v^h \in V^h. \end{cases} \tag{7.43}$$

As before, the problem of finding a $u^h \in V^h$ which satisfies (7.43) reduces to solving a linear system of equations; in this case the coefficient matrix has dimension $N+2$ but is still tridiagonal when piecewise linear polynomials are used. In addition, we can use the interpolation results given in Lemma 7.1 to get the identical optimal error estimates as before.

One purpose of the following computations is to demonstrate the difference in satisfying a boundary condition by imposing it on the space (an essential boundary condition) as the previous examples did and imposing a boundary condition weakly through the weak formulation (a natural boundary condition).

Example 3. We consider the problem

$$\begin{aligned} -u'' + \pi^2 u &= 2x\pi^2 \sin \pi x - 2\pi \cos \pi x \quad \text{for } 0 < x < 1 \\ u'(0) &= 0 \\ u'(1) &= -\pi \,, \end{aligned} \tag{7.44}$$

whose exact solution is given by $u = x \sin \pi x$. Note that this is the same differential equation as in Example 7.2.7 but now we are imposing Neumann boundary conditions. Since our solution $u(x) = x \sin \pi x$ is actually in $C^\infty(0,1)$ we expect the optimal rates of convergence which we can see are obtained from Table 7.5. The approximate solutions using uniform grids of $h = \frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{16}$ along with the exact solution are plotted in Figure 7.5. Note that although our exact solution is zero at the endpoints, our approximate solution is not because we imposed Neumann boundary conditions. It is important to realize that the approximate solution does not satisfy the exact derivative boundary condition because we have satisfied it weakly. This is analogous to the finite difference case where we satisfy the Neumann boundary condition by approximating the derivative by a difference quotient. In the last plot in Figure 7.5 we have blown up the approximate solutions at the right end point which should have a slope of $-\pi$. The approximate derivative at the right boundary is -1.994, -2.645, -2.917 and -3.036 for $h = 1/4$, $1/8$, $1/16$, and $1/32$ respectively. These correspond to errors of 1.147, 0.4968, 0.2244 and 0.1055. As $h \to 0$ the derivative of the approximate solution at $x = 1$ approaches the exact value of $-\pi$ linearly; this is expected because the rate of convergence in the L_2 norm of the derivative is one. Note that this is in contrast to Example 7.2.7 where our approximate solution exactly satisfied the homogeneous Dirichlet boundary condition because we imposed it on our space.

Table 7.5: Numerical results for Example 7.3 using continuous, piecewise linear polynomials.

h	$\left\|(u-u^h)'\right\|_2$	rate	$\left\|u-u^h\right\|_2$	rate
$1/4$	0.48183		0.22942×10^{-1}	
$1/8$	0.23838	1.0153	0.56235×10^{-2}	2.0281
$1/16$	0.11892	1.0033	0.13988×10^{-2}	2.0073
$1/32$	0.059425	1.0009	0.34924×10^{-3}	2.0019

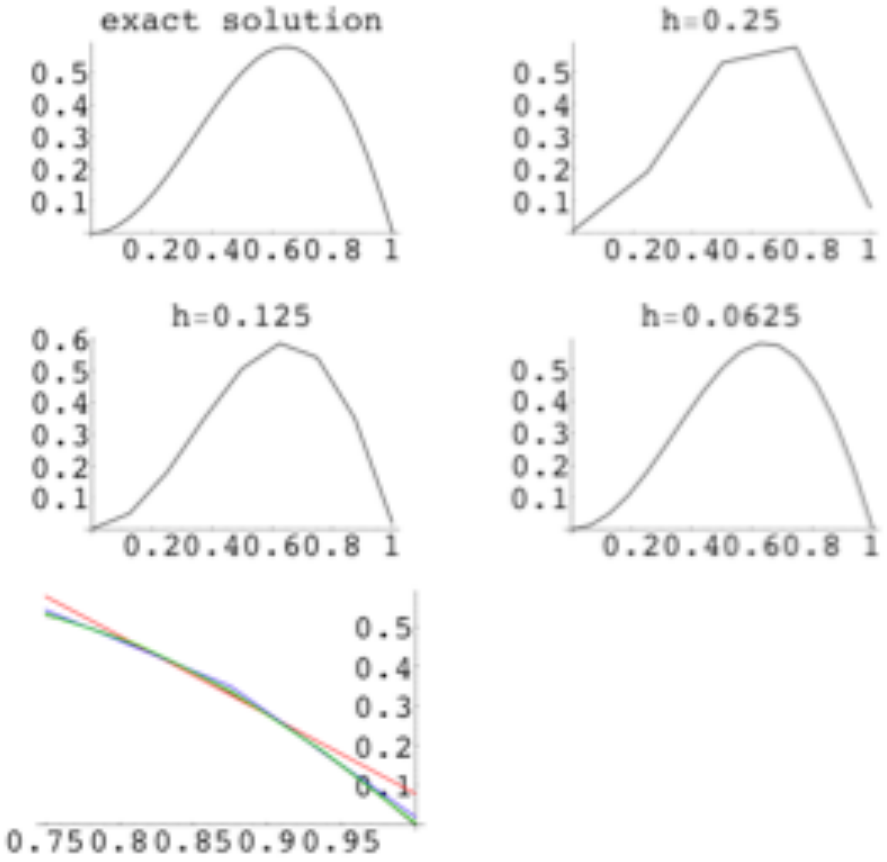


Figure 7.5: Plots of the exact solution and three piecewise linear approximations. The last plot gives a blow-up of the right endpoint demonstrating that the natural boundary condition is only satisfied weakly.

Chapter 8

The Finite Element Method in Higher Dimensions

8.1 Simple Examples on Rectangular Domains

We first consider simple elliptic boundary value problems in rectangular domains in $\mathbb{R}^2$ or $\mathbb{R}^3$; as before our prototype example is the Poisson equation. Similar to our exposition of the two-point boundary value problem in the previous chapter we consider the implementation of different boundary conditions for the Poisson equation. Much of this exposition is a straightforward extension of the results presented in the previous chapter for the two-point boundary value problem. However, a few important differences will be evident.

Because we first look at problems defined on rectangular domains we approximate the solution with finite elements spaces which are obtained by taking tensor products of one-dimensional finite element spaces that we defined in the previous chapter. Later in this chapter we will consider triangular elements which are useful for problems defined over non-rectangular domains such as a polygonal region.

8.1.1 The Poisson equation with homogeneous Dirichlet boundary data

We first consider the Poisson equation defined in a bounded domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ with homogeneous Dirichlet boundary data. We let $\mathbf{x}$ denote a point in $\mathbb{R}^2$ or $\mathbb{R}^3$. We let Ω be an open, connected, bounded set in $\mathbb{R}^2$ or $\mathbb{R}^3$ and let Γ denote its boundary. At this point in our discussion of the finite element method, we only have the background to use finite element spaces which are tensor products of the one dimensional finite element spaces discussed in the last chapter. Consequently, when we move to the discretization stage we require that Ω be a rectangular domain. However, the weak formulations that we present hold for more general domains.

Specifically we want to approximation $u(\mathbf{x})$ which is the solution of the Poisson equation

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega \tag{8.1a}$$

$$u(\mathbf{x}) = 0 \quad \text{for } \mathbf{x} \in \Gamma\,, \tag{8.1b}$$

where, as usual, $\Delta u = u_{xx} + u_{yy}$ in $\mathbb{R}^2$ or analogously $\Delta u = u_{xx} + u_{yy} + u_{zz}$ in $\mathbb{R}^3$. It is well known that for sufficiently smooth Γ there exists a unique classical solution of (8.1).

In the sequel, we assume enough smoothness of the boundary so that the domain admits the application of the divergence theorem. Every polygonal domain or a domain with a piecewise smooth boundary has sufficient smoothness for our purposes.

We will make extensive use of Green's formula which is the analogue of the integration by parts formula in higher dimensions and is derived from the divergence theorem of vector calculus. Let $\mathbf{n}$ denote the unit outer normal to Γ and let dS denote the measure defined on the boundary and $d\Omega$ the measure of volume. We have that for $v \in C^1(\Omega)$, $w \in C^2(\Omega)$

$$\int_\Omega v\Delta w \; d\Omega = \int_\Gamma v(\mathbf{n}\cdot\nabla w) - \int_\Omega \nabla w \cdot \nabla v \; d\Omega$$

or equivalently

$$\int_\Omega v\Delta w \; d\Omega = \int_\Gamma v\frac{\partial w}{\partial \mathbf{n}} - \int_\Omega \nabla w \cdot \nabla v \; d\Omega\,. \tag{8.2}$$

8.1.2 Weak formulation

To define the weak formulation we first define the function space where we seek the solution. As before, we impose the homogeneous Dirichlet boundary conditions by constraining our space V; in particular we have the space

$$V_0 = \{v \in C^1(\Omega) \mid v = 0 \text{ on } \Gamma\}\,.$$

The weak formulation which we consider is

$$\left\{ \begin{array}{l} \text{seek } u \in V_0 \text{ such that} \\ \displaystyle\int_\Omega \nabla v \cdot \nabla w \; d\Omega = \int_\Omega f v \; d\Omega \quad \forall v \in V_0\,. \end{array} \right. \tag{8.3}$$

The solution $u \in V_0$ of (8.3) is called the weak solution of (8.1).

If $u(\mathbf{x})$ satisfies the classical problem (8.1) then $u(\mathbf{x})$ satisfies the weak formulation (8.1) because

$$\begin{aligned} \int_\Omega f v \; d\Omega &= -\int_\Omega \Delta u v \; d\Omega \quad \forall v \in V_0 \\ &= \int_\Omega \nabla u \cdot \nabla v \; d\Omega - \int_\Gamma \frac{\partial u}{\partial \mathbf{n}} v \; d\Gamma \\ &= \int_\Omega \nabla u \cdot \nabla v \; d\Omega \end{aligned}$$

where we have used Green's formula (8.2) and imposed the fact that $v = 0$ on Γ. The existence and uniqueness of a weak solution to (8.3) can be verified.

8.1.3 Approximation using bilinear functions

At present, we restrict the domain so that we can use rectangular elements; therefore, the finite element spaces can be constructed from the spaces used in the previous chapter. As in the one-dimensional case, we must now choose a finite dimensional subspace of $V_0^h(\Omega) \subset V_0$ in which to seek the approximate solution. For the discrete problem we have

$$\begin{cases} \text{seek } u^h \in V_0^h(\Omega) \text{ satisfying} \\ \int_\Omega \left(\nabla u^h \cdot \nabla v^h\right)\, d\Omega = \int_\Omega f v^h \, d\Omega \quad \forall v^h \in V_0^h \,. \end{cases} \tag{8.4}$$

To approximate our finite element solution we consider the concrete case where Ω is the $(a_1, b_1) \times (a_2, b_2)$ in $\mathbb{R}^2$ or $(a_1, b_1) \times (a_2, b_2) \times (a_3, b_3)$ in $\mathbb{R}^3$ We choose the space $V_0^h(\Omega)$ to be continuous, piecewise bilinear functions defined on $\Omega \subset \mathbb{R}^2$ or continuous, piecewise trilinear functions[1] for $\Omega \subset \mathbb{R}^3$. We formally construct the bilinear basis functions; the trilinear basis functions are defined analogously. Let N, M be positive integers and let $h_x = (b_1 - a_1)/(N+1)$, $h_y = (b_2 - a_2)/(M+1)$ and consider the subdivision of Ω into rectangles of size $h_x \times h_y$ where

$$x_i = a_1 + ih_x, \;\; 0 \le i \le N+1, \quad y_j = a_2 + jh_y, \;\; 0 \le j \le M+1 \,.$$

Let $\phi_i(x)$, $1 \le i \le N$ represent the standard "hat" piecewise linear basis functions in x and let $\phi_j(y)$, $1 \le j \le M$, be similarly defined i.e.,

$$\phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h_x} & \text{for } x_{i-1} \le x \le x_i \\ \dfrac{x_{i+1} - x}{h_x} & \text{for } x_i \le x \le x_{i+1} \\ 0 & \text{elsewhere} \end{cases} \quad \phi_j(y) = \begin{cases} \dfrac{y - y_{j-1}}{h_y} & \text{for } y_{j-1} \le y \le y_j \\ \dfrac{y_{j+1} - y}{h_y} & \text{for } y_j \le y \le y_{j+1} \\ 0 & \text{elsewhere.} \end{cases}$$

On $\Omega = (0,1) \times (0,1)$ we now define the NM *bilinear functions*

$$\phi_{ij}(x, y) = \phi_i(x)\phi_j(y) \quad \text{for } 1 \le i \le N,\, 1 \le j \le M \,. \tag{8.5}$$

We easily see that $\phi_{ij}(x_i, y_j) = 1$ and $\phi_{ij}(x_k, y_l) = 0$ for $k \ne i$ **or** $l \ne j$. Also $\phi_{ij}(x, y)$ is zero outside of $[(i-1)h_x, (i+1)h_x] \times [(j-1)h_y, (j+1)h_y]$. The support of $\phi_{ij}(x, y)$ is illustrated in Figure 8.1 and the shape of a specific bilinear function $\phi_{2,3}$ which is one at node (x_2, y_3) is given in Figure 8.2.

For Ω the unit square, we choose $V_0^h(\Omega) \equiv V_0^h(0,1) \otimes V_0^h(0,1)$ to be the tensor product of the subspaces $V_0^h(0,1)$ (one each in the $x-$ and $y-$ directions) of one-dimensional piecewise linear, continuous functions which vanish at zero and one.

[1] A bilinear or trilinear function is a function which is linear with respect to its variables because if we hold one variable fixed, it is linear in the other; for example $f(x, y) = xy$ is a bilinear function but $f(x, y) = x^2 y$ is not.

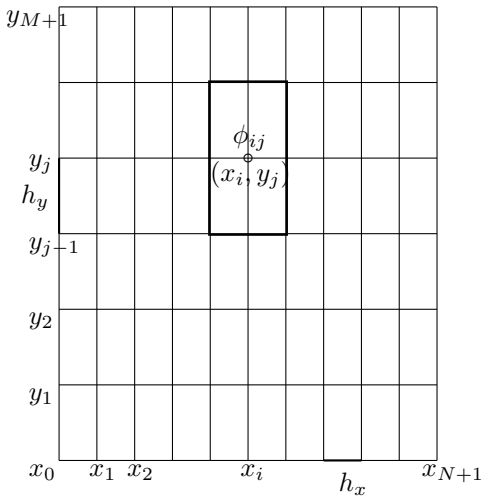


Figure 8.1: Grid on a unit square with support of basis function $\phi_{ij}(x, y)$ indicated.

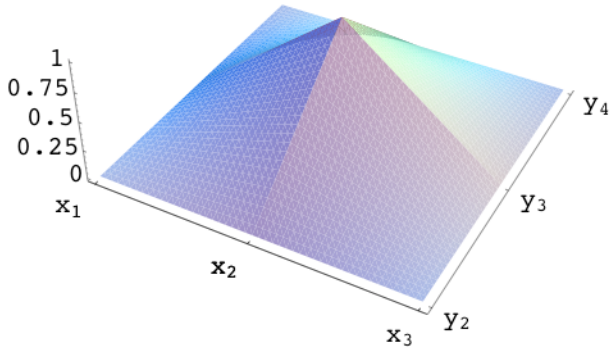


Figure 8.2: Support of bilinear basis function $\phi_{2,3}$.

$V_0^h(\Omega)$ consists of all functions $v(x, y)$ on $(0, 1) \times (0, 1)$ of the form

$$v(x,y) = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\phi_i(x)\phi_j(y) = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\phi_{ij}(x,y)\,. \tag{8.6}$$

Note that the general form of a bilinear function in $\mathbb{R}^2$ is $a_0 + a_1x + a_2y + a_3xy$ compared with a linear function in two variables which has the general form $a_0 + a_1x + a_2y$. Clearly $V_0^h(\Omega)$ is the space of all continuous, piecewise bilinear functions (with respect to the given subdivision) which vanish on the sides of the unit square. Also, every piecewise bilinear function $f(x, y)$ can be written in the form (8.6) with $c_{ij} = f(x_i, y_j)$; i.e., it is a linear combination of the $P = NM$ linearly independent functions $\phi_{ij}(x, y)$. $V_0^h(\Omega)$ is an P-dimensional subspace of V_0; note that for $M = N$, V_0^h is an N^2 dimensional subspace whereas in one dimension, it was an N dimensional subspace. Of course this affects the size of our resulting matrix problem. However, this was the same increase that we saw using the FD method.

From previous discussions we know that once a basis is chosen for the approximating subspace, the discrete problem can be written as a linear system of equations. To investigate the structure of the coefficient matrix for our choice of bilinear basis functions, we let the basis functions $\phi_{ij}(x, y)$ for $V_0^h(\Omega)$ be rewritten in single index notation; for simplicity of exposition we choose $M = N$. We have

$$\{\psi_k(x, y)\}_{k=1}^{N^2} = \{\phi_{ij}(x, y)\}_{i,j=1}^{N} .$$

For example, for $1 \le k \le N$ $\psi_k = \phi_{k1}$; for $1 \le k \le N$ $\psi_{N+k} = \phi_{k2}$; etc. Our discrete weak formulation (8.4) is equivalent to seeking $u^h \in V_0^h$ satisfying

$$\int_\Omega \nabla u^h \nabla \psi_i \, d\Omega = \int_\Omega f\psi_i \, d\Omega \quad \text{for } 1 \le i \le N^2 .$$

We now let $u^h = \sum_{j=1}^{N^2} c_j\psi_j$ and substitute into the above expression. The result is a linear system of N^2 equations in the N^2 unknowns $\{c_j\}_{j=1}^{N^2}$; i.e., $\mathcal{A}\mathbf{c} = \mathbf{f}$ where $\mathbf{c} = (c_1, \ldots, c_{N^2})^T$, $\mathcal{F}_i = \int_\Omega f\psi_i \, d\Omega$ and $\mathcal{A}_{ij} = A(\psi_i, \psi_j)$. Note that with the numbering scheme we are using for the basis functions, we are numbering our unknowns which correspond to the coefficients c_j across rows. Because we have assumed the same number of points in the x and y directions we could have easily numbered them along columns of the grid.

To determine the structure of the resulting matrix we consider the ith row of the matrix and decide how many nonzero entries are in the row. Because we know the matrix is symmetric, we only consider terms above the diagonal. Clearly there can be nonzero entries in columns i and $i + 1$. The next nonzero entries occur for unknowns corresponding to basis functions in the next row of the grid. Specifically we can have nonzero entries in columns $i + N - 1$, $i + N$ and $i + N + 1$ where N is the number of unknowns across the row. The coefficient matrix $\mathcal{A}$ is an $N^2 \times N^2$ symmetric, positive definite matrix which has a block tridiagonal structure of the form

$$\mathcal{A} = \begin{pmatrix} A_0 & A_1 & 0 & & \cdots & 0 \\ A_1 & A_0 & A_1 & 0 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & A_1 & A_0 & A_1 \\ 0 & \cdots & & 0 & A_1 & A_0 \end{pmatrix} , \tag{8.7}$$

where A_0 and A_1 are $N \times N$ tridiagonal matrices. A matrix of this form can be solved efficiently by a banded Cholesky algorithm, a block tridiagonal solver or an iterative solver.

Error estimates

One can prove error estimates in an analogous manner to the one-dimensional case. Specifically one demonstrates that

$$\left\|\nabla(u-u^h)\cdot\nabla(u-u^h)\right\|_2 \leq \inf_{\chi^h\in V_0^h}\left\|\nabla(u-\chi^h)\right\|_2$$

We then use results from interpolation theory to bound the error in the derivative of $(u(\mathbf{x})-u^h(\mathbf{x}))$. We state the results here for bilinear elements for completeness.

Lemma 8.1. *Let $v \in C^2(\Omega)$. Then if $I^h v$ is the interpolant of v in $V^h(\Omega)$, the space of continuous, piecewise bilinear functions, then there exist constants C_i, $i = 1, 2$ independent of v and h such that*

$$\left\|v - I^h v\right\|_0 \leq C_1 h^2 \left\|v\right\|_2 \tag{8.8}$$

and

$$\left\|v - I^h v\right\|_1 \leq C_2 h \left\|v\right\|_2 . \tag{8.9}$$

Theorem 8.1. *Let $u \in C^2(\Omega) \cap V_0$ be the solution of (8.3) where $\Omega = (0,1) \times (0,1)$. Let $V_0^h(\Omega)$ be the space of piecewise bilinear functions which vanish on Γ and let u^h be the Galerkin approximation to u in $V_0^h(\Omega)$ defined by (8.4). Then*

$$\left\|\nabla(u-u^h)\right\|_2 \leq Ch \left\|u\right\|_2 \tag{8.10}$$

and

$$\left\|u-u^h\right\|_2 \leq Ch^2 \left\|u\right\|_2 \tag{8.11}$$

for some constants C independent of h and u.

8.1.4 Higher order elements

Our discussion of approximating the problem (8.1) posed on $\Omega = (0,1) \times (0,1)$ has so far included only piecewise bilinear function spaces. Of course, we can also use tensor products of higher order spaces such as the quadratic or cubic functions in one space dimension. Note that a general biquadratic function has the form $a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2 + a_6 x^2 y + a_7 xy^2 + a_8 x^2 y^2$ compared with a general quadratic function in two dimensions which has the form $a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2$. As in the one-dimensional case, for a smooth enough solution, these spaces yield higher rates of convergence then that achieved with piecewise bilinear approximations. The construction of the basis functions in two or three dimensions is done analogous to the piecewise bilinear case; the details are left to the exercises.

8.1.5 Numerical quadrature

Once again, the entries in the matrix and right-hand side of our linear system are calculated using a numerical quadrature rule which has the form

$$\int_\Omega f(\mathbf{x})\ d\Omega \approx \sum_i f(\mathbf{q}_i)\omega_i\,,$$

where the points $\mathbf{q}_i$ are the quadrature points and ω_i are the quadrature weights. Because we are using rectangular elements with basis functions obtained by taking the tensor product of one-dimensional basis functions, the most straightforward approach is to use tensor products of the quadrature rules in one spatial dimension. Typically, we use the same quadrature rule in each spatial dimension. For example, if we have the rule

$$\int_a^b f(x)\ dx = \sum_i f(q_i)w_i$$

then we can write

$$\int_a^b \int_c^d f(x,y)\ dydx \approx \int_a^b \Big(\sum_j f(x,q_j)w_j\Big) \approx \sum_i \sum_j f(q_i,q_j)w_j w_i\,.$$

In one dimension we employed the Gauss-Legendre quadrature rules on $[-1,1]$. If we take the tensor products of a p-point Gauss rule in each direction then we would have one point for the tensor product of the one-point rule, four points for the tensor product of the two-point rule, nine points for the tensor product of the three-point rule, etc. The quadrature points in two dimensions formed by the tensor product of one-point through three-point Gauss quadrature rules are described in Table 8.1. Note that in three dimensions we have 1, 8, and 27 quadrature points for tensor products of these three quadrature rules. To apply these rules to an integral over an arbitrary rectangular domain, we must perform a change of variables in both the x and y directions analogous to the one-dimensional case. For our example, if we are using bilinear or trilinear elements, then the tensor product of the one-point Gauss rule is adequate; for biquadratics or triquadratics we need to use the tensor product of the two-point Gauss rule.

8.1.6 The Poisson equation with Neumann boundary data

In this section we consider solving Poisson's equation on an open, bounded domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ where we specify Neumann data on a portion of the boundary and Dirichlet data on the remainder of the boundary. In particular, we seek a function u satisfying

$$\begin{aligned} -\Delta u(\mathbf{x}) &= f(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega \\ u(\mathbf{x}) &= 0 \quad \text{for } \mathbf{x} \in \Gamma_1 \\ \frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) &= g(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Gamma_2\,, \end{aligned} \tag{8.12}$$

	1-D rule	# points in $\mathbb{R}^2$	points q_i & weights w_i
	1 point Gauss	1	$q_1 = (0,0) \quad w_1 = 4$
	2 point Gauss	4	$q_i = \frac{1}{\sqrt{3}}\{(-1,-1),(1,-1),(-1,1),(1,1)$ $w_i = 1$
	3 point Gauss	9	$q_i = \sqrt{\frac{3}{5}}\Big\{(-1,-1),(0,-1),(1,-1),(-1,0),$ $((0,0),(1,0),(-1,1),(0,1),(1,1)\Big\}$ $w_i = \frac{1}{81}\{25,40,25,40,64,40,25,40,25\}$

Table 8.1: Tensor product of Gauss quadrature rules in two dimensions

where $\Gamma_1 \cup \Gamma_2 = \Gamma$, $\Gamma_1 \cap \Gamma_2$ is not empty and $\partial u/\partial \mathbf{n}$ denotes the directional derivative of u in the direction of the unit outward normal $\mathbf{n}$ to the boundary of the domain. We note that if Γ_1 is the entire boundary Γ then we have the purely Dirichlet problem discussed in Section (8.1.1); in the case Γ_2 is the entire boundary we have a purely Neumann problem. As expected, in the latter case the problem does not have a unique solution. It is well known that for sufficiently smooth boundary there exists a unique classical solution of (8.12).

8.1.7 Weak Formulation

For this problem we define $\hat{V_0}$ as

$$\hat{V_0} = \{u \in C^1(\Omega) \mid u = 0 \text{ on } \Gamma_1\}\,. \tag{8.13}$$

Our weak formulation is

$$\begin{cases} \text{seek } u \in \hat{V_0} \text{ satisfying} \\ \int_\Omega \nabla u \cdot \nabla v \; d\Omega = \int_\Omega f v d\Omega + \int_{\Gamma_2} g v \; ds \quad \forall v \in \hat{V_0}\,. \end{cases} \tag{8.14}$$

If u is a solution of the classical problem (8.12) then by Green's theorem u satisfies

$$\begin{aligned} \int_\Omega f v \; d\Omega = -\int_\Omega \Delta u v \; d\Omega &= \int_\Omega \nabla u \cdot \nabla v - \int_\Gamma \frac{\partial u}{\partial \vec{n}} v \; ds \\ &= \int_\Omega \nabla u \cdot \nabla v - \int_{\Gamma_1} \frac{\partial u}{\partial \vec{n}} v \; ds - \int_{\Gamma_2} \frac{\partial u}{\partial \vec{n}} v \; ds \\ &= \int_\Omega \nabla u \cdot \nabla v - \int_{\Gamma_2} g(\mathbf{x}) v \; ds \quad \forall v \in \hat{V_0}\,, \end{aligned}$$

where we have used the fact that the boundary integral over Γ_1 is zero since $v \in \hat{V}_0$ and for the boundary integral over Γ_2 we have used $\partial u/\partial \vec{n} = g(\mathbf{x})$. In this problem the Dirichlet boundary condition on Γ_1 is essential whereas the Neumann boundary condition on Γ_2 is natural. It's interesting to compare the weak formulation (8.14) with the analogous weak formulation for the two-point boundary value problem. In the one-dimensional case, we simply have the value of the derivative at a point times the test function at the same point. In two spatial dimensions with inhomogeneous Neumann boundary conditions we have a line integral on the right-hand side of the weak form and in three spatial dimensions we have a surface integral. This complicates the implementation of the method but it is straightforward; for example, for $\Omega \subset \mathbb{R}^2$ we have a line integral on the boundary which can be approximated using a Gauss quadrature rule. The existence and uniqueness of a solution to (8.14) is demonstrated in an analogous manner to the purely Dirichlet problem discussed in Section 8.1.1.

8.1.8 Approximation using bilinear functions

As a concrete example we once again take $\Omega = (0,1) \times (0,1)$; we choose Γ_1 to be the top and bottom portions of the boundary, i.e., when $y = 0$ and $y = 1$; Γ_2 is the remainder of the boundary. We subdivide our domain into rectangles of size $h \times h$ where $h = 1/(N+1)$, $x_i = ih$, $y_j = jh$, $i, j = 0, \ldots, N+1$. If we approximate using continuous, piecewise bilinear functions as in Section 8.1.1, then we seek our solution in the space $\hat{V}_0^h$ which is the space of all continuous, piecewise bilinear functions on Ω which are zero at $y = 0$ and $y = 1$. In the x-direction we have the $N+2$ basis functions $\phi_i(x)$, $i = 0, 1, \ldots, N+1$ and N basis functions in the y-direction $\phi_j(y)$, $j = 1, \ldots, N$. In this case we have the $N(N+2)$ basis functions $\phi_{ij}(x, y)$ which are the tensor products of the one-dimensional basis functions. The basic structure of the matrix is the same as in the previous example. Optimal error estimates are derived in a completely analogous manner to the previous section.

We note that if we attempt to discretize the purely Neumann problem, i.e., when $\Gamma_2 = \Gamma$, then the resulting $(N+2)^2$ matrix would be singular. This is to be expected because we could not prove uniqueness of the solution to the weak problem. A unique solution to the system can be found by imposing an additional condition on u^h such as specifying u^h at one point or requiring the solution to have zero mean, i.e., $\int_\Omega u \, d\Omega = 0$.

8.1.9 A Neumann problem for the Helmholtz equation

We have seen that the purely Neumann problem for Poisson's equation; i.e., when $\Gamma_2 = \Gamma$, does not have a unique solution and if we attempt to discretize then we have a singular matrix. If, however, we consider the Neumann problem for the Helmholtz equation

$$\begin{aligned} -\Delta u + \sigma^2 u &= f \quad \text{in } \Omega \\ \frac{\partial u}{\partial \mathbf{n}} &= 0 \quad \text{on } \Gamma \end{aligned}$$

then the problem possesses a unique solution for $u \in C^2$ and sufficiently smooth boundary. In this case the weak formulation is to find $u \in V$ such that

$$\int_\Omega \left(\nabla u \cdot \nabla v + uv\right) \, d\Omega = \int_\Omega fv \, d\Omega \quad \forall v \in V \, .$$

The discrete weak form is defined in an analogous manner.

8.2 Computational examples

Before looking at a specific example, we first compare the number of nodes, the number of unknowns, and the number of quadrature points required to approximate the solution of the problem $-\Delta u + u = f$ with homogeneous, Neumann boundary conditions in one, two and three dimensions. Note that in this purely Neumann problem the number of unknowns is the same as the number of nodes. Specifically we compare the number of unknowns for various values of h for linear, bilinear and trilinear elements as well as for tensor products of quadratic and cubic spaces. We also provide the minimum number of quadrature points that are used in each case. Recall that the number of unknowns corresponds to the size of the matrix and the number of quadrature points influences the amount of work required to compute the entries in the matrix and right-hand sides. In all cases we assume a uniform grid with spacing h in each dimension. The "curse of dimensionality" can clearly be seen from Table 8.2.

	Number of unknowns			Number of
	$h = 0.1$	$h = 0.01$	$h = 0.001$	quadrature pts.
linear	11	101	1001	1
bilinear	121	10,201	1.030×10^6	1
trilinear	1331	1.030×10^6	1.003×10^9	1
quadratic	21	201	2001	2
biquadratic	441	40,401	4.004×10^6	4
triquadratic	9261	8.121×10^6	8.012×10^9	8
cubic	31	301	3001	3
bicubic	961	90,601	9.006×10^6	9
tricubic	29,791	2.727×10^7	2.703×10^{10}	27

Table 8.2: Comparison of number of unknowns for solving a problem on a domain $(0,1)^n$, $n = 1, 2, 3$ using tensor products of one-dimensional elements.

We now turn to providing some numerical results for the specific problem

$$\begin{aligned} -u''(x) &= (x^2 + y^2)\sin(x, y) \qquad \forall (x, y) \in \Omega \\ u &= \sin(xy) \quad \text{on } \Gamma \end{aligned} \tag{8.15}$$

where $\Omega = \{(x, y) \mid 0 \le x \le 3, \quad 0 \le y \le 3\}$. The exact solution to this problem is $u(x, y) = \sin(xy)$ whose solution is plotted in Figure 8.3 along with a contour

plot of the solution. Note that we are imposing inhomogeneous Dirichlet boundary conditions in this example. The results presented here use bilinear and biquadratic elements on a uniform grid of size h in each dimension; for the quadrature rule we use the tensor product of the one point Gauss rule for bilinears and the tensor product of the two point Gauss rule for biquadratics. As usual, a higher order quadrature rule is used to calculate the error. The numerical rates of convergence are obtained in the same manner as before. The results are presented in Table 8.3 and some results are plotted for the bilinear case in Figure 8.4. Note that as expected, the optimal rates of convergence are obtained.

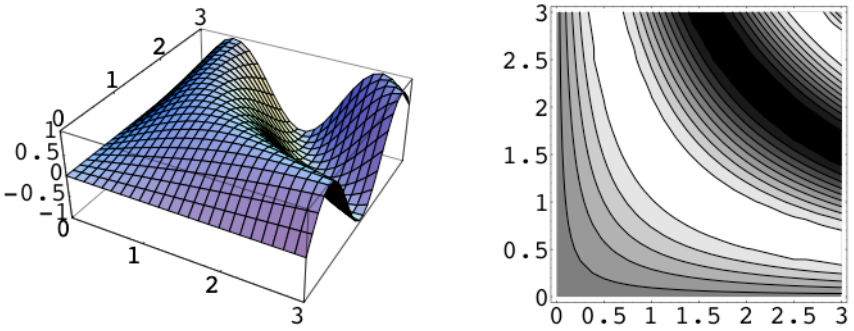


Figure 8.3: Exact solution

Table 8.3: Numerical results for (8.15) using bilinear and biquadratic elements.

element	h	No. of unknowns	$\|u-u^h\|_1$	rate	$\|u-u^h\|_0$	rate
bilinear	1/4	144	0.87717		0.76184×10^{-1}	
bilinear	1/8	529	0.0.43836	1.0007	0.19185×10^{-1}	1.9895
bilinear	1/16	2209	0.21916	1.0001	0.48051×10^{-2}	1.9973
bilinear	1/32	9216	0.0.10958	1.0000	0.12018×10^{-3}	1.9994
biquadratic	1/4	529	0.70737×10^{-1}		0.22488×10^{-2}	
biquadratic	1/8	2209	0.17673×10^{-1}	1.9758	0.28399×10^{-3}	2.9853
biquadratic	1/16	9025	0.44175×10^{-2}	1.9940	0.35604×10^{-4}	2.9957
biquadratic	1/32	36,491	0.11043×10^{-2}	1.9986	0.44539×10^{-5}	2.9990

8.3 Finite Element Spaces

One of the advantages of the finite element method is that it can be used with relative ease to find approximations to solutions of differential equations on general domains. So far we have only considered approximating in one dimension or in higher dimensions using rectangular elements. Our goal now is to present some examples

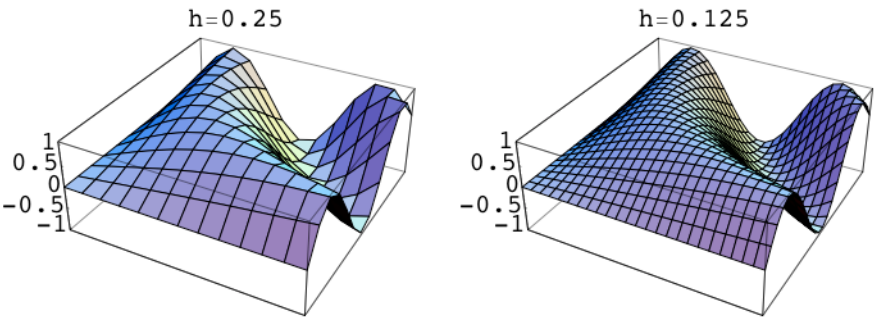


Figure 8.4: Approximations to Problem (8.15) using $h = 1/4$ and $h = 1/8$.

of commonly used elements. When we have a domain with curved boundaries there are special elements called $isoparametric$ elements; however we will not discuss those here.

To precisely describe a particular finite element, it is not enough to give the geometric figure, e.g., a triangle, rectangle, etc. One must also specify the degree of polynomial that is used. Does describing these two pieces of information uniquely determine the choice? In fact, no. For example, in $\mathbb{R}^1$ using an interval as the geometric element and specifying a cubic polynomial on each interval does not completely describe the finite element because we can determine the cubic by function values at four points or by function and derivative values at two points; the latter is called a Hermite cubic. Consequently, three pieces of information must be provided to give an adequate description of a finite element; we must specify

(i) the geometric element,

(ii) the degree of polynomial, and

(iii) the degrees of freedom which are used to uniquely determine the polynomial.

Once we have chosen a particular finite element, we subdivide the domain into a finite number of geometric elements; this meshing must be "admissible", i.e., satisfy certain properties. We want to construct a finite element space over this mesh which possesses specific properties. A basic property which we said is a distinguishing feature of the finite element method is that we use a piecewise polynomial which is a kth degree polynomial when restricted to the specific element. For second order differential equations this piecewise polynomial has to be continuous but for fourth order problems more smoothness is required; this is why fourth order problems are typically written as a system of second order equations. Also, for the finite element method to be computationally efficient we must be able to construct a basis which has small support. Before addressing some of these issues we consider the admissible "triangulations" of a domain.

8.3.1 Admissible triangulations

Once a specific geometric element is chosen, we subdivide the domain Ω into a finite number of individual subsets or geometric elements. We will use the terminology *triangulation* to refer to a subdivision of Ω even if the specific geometric element is not a triangle. The subsets form a triangulation of Ω, denoted $\mathcal{T}^h$, which must satisfy certain properties. Some of these properties are obvious, such as the fact that their union is $\bar{\Omega}$ (the domain including the boundary), while others may not be as obvious. For example, we must add a condition which guarantees there are no "hanging nodes" as indicated in Figure 8.5.

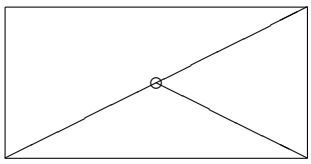


Figure 8.5: Inadmissible triangulation due to "hanging node"

Definition 1. A subdivision $\mathcal{T}^h$ of Ω into subsets $\{\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_M\}$ is an *admissible triangulation* of Ω if it satisfies the following properties:

(i) $\bar{\Omega} = \cup_{j=1}^{M} \mathcal{K}_j$;

(ii) for each j, $j = 1, 2, \ldots, M$, the set $\mathcal{K}_j$ is closed and the interior of $\mathcal{K}_j$ is non-empty;

(iii) for each $\mathcal{K}_j$, $j = 1, 2, \ldots, M$, the boundary $\partial\mathcal{K}_j$ is Lipschitz continuous[2] ;

(iv) if the intersection of two elements $\mathcal{K}_j$ and $\mathcal{K}_\ell$ is nonempty then the intersection must be a common vertex of the elements if the intersection is a single point; otherwise the intersection must be an entire edge or face common to both $\mathcal{K}_\ell$ and $\mathcal{K}_j$.

See Figure 8.5 for an example of a triangulation which does not satisfy condition (iii).
The penultimate condition in Definition 8.3.1 allows the application of Green's formula over each element.

The parameter h in the triangulation $\mathcal{T}^h$ is related to the size of the geometric elements and generally gives a measure of the coarseness or fineness of the mesh. It

[2]A domain in Euclidean space with Lipschitz boundary is one whose boundary is "sufficiently regular". Formally, this means that the boundary can be written as, e.g., $z = f(x, y)$ where f is Lipschitz continuous.

is usually taken to be the diameter of the largest element. If we have a mesh where all the geometric elements are congruent, then the triangulation is $uniform$ if all the elements are the same size; otherwise the triangulation is called $nonuniform$.

8.4 Examples of finite elements

In $\mathbb{R}^2$ the common choices for a geometric element are a triangle and a quadrilateral. If the domain is polygonal and not rectangular, then triangular elements are needed to discretize. In $\mathbb{R}^3$ the commonly used elements are tetrahedra, prisms and cubes or bricks. In this section we look at some of the more commonly used triangular elements and their variants.

We have seen that to completely specify a finite element, it is not enough to just choose a geometric element. We must also specify the degree of polynomial on the element and the degrees of freedom which uniquely determine the polynomial. To use the element we must also specify a basis which has small support. We saw that for rectangular elements we could simply use tensor products of the basis in one-dimension. For triangles or tetrahedra, this approach does not work.

8.4.1 Simplices

To gain insight into how finite element families are defined we look at a particular example called $simplices$ which are line segments in $\mathbb{R}^1$, triangles in $\mathbb{R}^2$ or tetrahedra in $\mathbb{R}^3$. Formally, we define an n-$simplex$ in the following way.

Definition 2. Let z_k, $k = 1, \ldots, n+1$, denote $n+1$ points in $\mathbb{R}^n$. The intersection of all convex sets [3] containing these $n+1$ points, the containing z_k, $k = 1, \ldots, n+1$, is called an n-$simplex$ and the points z_k, $k = 1, \ldots, n+1$, are called the $vertices$ of the n-simplex.

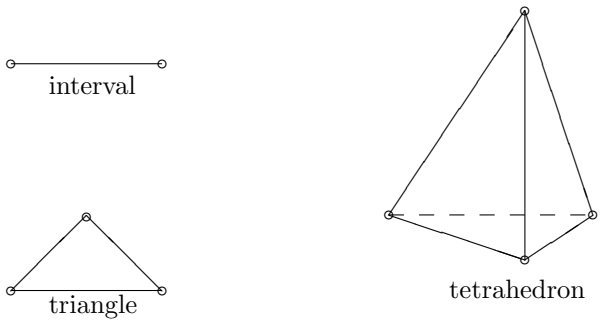
For example, for $n = 1$ we specify two points $\{z_1, z_2\}$ and a 1-simplex is an interval with the endpoints z_1, z_2 where we require $z_1 \neq z_2$. For $n = 2$ we specify three points $\{z_1, z_2, z_3\}$ and a 2-simplex is simply a triangle with vertices (z_{i_1}, z_{i_2}), $i = 1, 2, 3$, provided the three points are not collinear. To enforce the non-collinearity of the points, we require that the matrix

$$\begin{pmatrix} z_{1_1} & z_{2_1} & z_{3_1} \\ z_{1_2} & z_{2_2} & z_{3_2} \\ 1 & 1 & 1 \end{pmatrix}$$

is nonsingular. Note that the magnitude of the determinant of this matrix is just the area of the parallelogram formed by the vectors $z_2 - z_1$ and $z_3 - z_1$. For $n = 3$, we specify four points $\{z_1, z_2, z_3, z_4\}$ and a 3-simplex is just a tetrahedron with

[3] Recall that a set $\mathcal{S}$ is convex if given any two points x and y in $\mathcal{S}$ then the line segment joining x and y lies entirely in $\mathcal{S}$.

Figure 8.6: n-simplicies of type(1)



vertices z_i, $i = 1, \ldots, 4$, provided the four points are not coplanar, i.e., provided the matrix

$$\begin{pmatrix} z_{1_1} & z_{2_1} & z_{3_1} & z_{4_1} \\ z_{1_2} & z_{2_2} & z_{3_2} & z_{4_2} \\ z_{1_3} & z_{2_3} & z_{3_3} & z_{4_3} \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

is nonsingular. Note that the magnitude of the determinant of this matrix is the volume of the parallelepiped formed by the vectors $z_i - z_1$, $i = 2, 3, 4$.

We have seen how to construct a basis in one dimension so that it is nodal (i.e., zero at all nodes except one where it takes on the value of one) and thus it is nonzero over a large portion of the domain. This property resulted in a sparse banded matrix equation to solve. In addition, in one dimension we had an explicit formula for our basis. We now want to see how we can construct a nodal basis in $\mathbb{R}^2$.

Constructing a nodal basis

We will only consider the case of triangles in $\mathbb{R}^2$ but the results easily generalize to $\mathbb{R}^3$. Suppose that we are given a set of three points z_1, z_2, z_3 which are not collinear so they defined a triangle. Suppose further that we want to construct a linear function $p(x, y) = a_0 + a_1 x + a_2 y$ which is one at vertex z_1 and zero at the other two vertices. To simplify the exposition assume that $z_1 = (0, 0)$, $z_2 = (1, 0)$ and $z_3 = (1, 1)$. In this case we have to solve the system of equations

$$\begin{aligned} p((0,0)) = a_0 + a_1 \cdot 0 + a_2 \cdot 0 &= 1 \\ p((1,0)) = a_0 + a_1 \cdot 1 + a_2 \cdot 0 &= 0 \\ p((1,1)) = a_0 + a_1 \cdot 1 + a_2 \cdot 1 &= 0 \end{aligned}$$

which gives the polynomial $1 - x$. Similarly the linear polynomial that is one at z_2 and zero at the other two nodes satisfies the system

$$\begin{aligned} p((0,0)) = a_0 + a_1 \cdot 0 + a_2 \cdot 0 &= 0 \\ p((1,0)) = a_0 + a_1 \cdot 1 + a_2 \cdot 0 &= 1 \\ p((1,1)) = a_0 + a_1 \cdot 1 + a_2 \cdot 1 &= 0 \end{aligned}$$

so it is given by $x - y$. The polynomial that is one at z_3 and zero at the other two nodes satisfies the system

$$\begin{aligned} p((0,0)) = a_0 + a_1 \cdot 0 + a_2 \cdot 0 &= 0 \\ p((1,0)) = a_0 + a_1 \cdot 1 + a_2 \cdot 0 &= 0 \\ p((1,1)) = a_0 + a_1 \cdot 1 + a_2 \cdot 1 &= 1 \end{aligned}$$

so it is just y. So on our specific triangle we have three linear polynomials, $1-x$, $x-y$ and y which any polynomial $a_0 + a_1 x + a_2 y$ can be written as a linear combination of these three polynomials. For example, $3 - 2x + 4y = 3(1 - x) + 1(x - y) + 5(y)$. The polynomials are linearly independent and span the space of all linear functions defined over our given triangle.

If we have a general triangle then we would have to solve a 3×3 system of equations to determine the basis function. However, if we take this approach then when we define a quadratic on our triangle, then we have a 6×6 system to solve. There are two approaches to avoid this problem. One is to determine formulas for the basis functions on a *reference* element such as the triangle we chose. Then we map these basis functions into the desired triangle much like we map our Gauss quadrature formulas defined on $[-1, 1]$ into our desired interval of integration. The other approach is to use something called barycentric coordinates which were first defined by Möbius in 1827. We know that if we are given a frame in $\mathbb{R}^n$, then we can define a local coordinate system with respect to the frame;e.g., Cartesian coordinates. If we are given a set of $n + 1$ points in $\mathbb{R}^n$ then we can also define a local coordinate system with respect to these points; such coordinate systems are called *barycentric coordinates*. These allow the calculation of higher order basis functions defined on a triangle by solving a 3×3 system of equations. We will not go into the details here.

In general, one can demonstrate rigorously that a linear function defined on a triangle can be uniquely determined by its values at the vertices of the triangle. In $\mathbb{R}^3$ a linear function on a tetrahedron can be uniquely determined by its values at the four vertices.

What do we do if we want to use higher degree polynomials? If we want to define a quadratic polynomial $a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 y^2 + a_5 xy$ on a triangle then we have six degrees of freedom $a_0, \ldots, a_5$ so we need to specify six conditions. We use the three vertices of the triangle and to get three additional conditions we choose the midpoints of the side of the triangle. Likewise for a cubic we have an additional four degrees of freedom so that we need to specify ten nodes. We choose the vertices, two points on each side of the triangle and an additional one at the

barycenter of the triangle. These nodes are illustrated in Figure 8.7. So for linear polynomials on a triangle we have three nodes, for quadratics on a triangle we have six nodes and for cubics on a triangle we have ten nodes. Recall that as we increase the number of nodes we increase the size of the resulting system of equations.

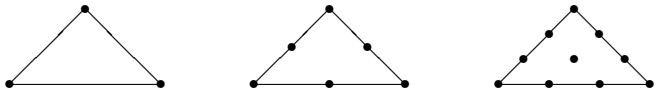


Figure 8.7: From left to right: three node linear triangle; six node quadratic triangle; ten node cubic triangle

All of the finite element spaces we have defined so far are called *Lagrangian* elements because they use the value at the nodes as constraints. We can also use derivative values as constraints; these are typically called *Hermite* elements. The interested reader is referred to a standard text in FEM for details.

8.4.2 Quadrature rules on triangles

When we used rectangular elements we were able to choose our quadrature rules as tensor products of Gauss quadrature rules in one dimension. However, on triangles we can no longer do this. If one uses continuous piecewise linear functions defined over triangles than the quadrature rule

$$\int_T g(x,y)\,dxdy \approx g(\mathbf{m}_T)\mathsf{area}(T)$$

is adequate for the assembly of the matrix and right hand side. Here $\mathbf{m}_T$ represents the barycenter of the triangle we are integrating over.

If we are using continuous piecewise quadratic functions defined over triangles then we must use a three point rule

$$\int_T g(x,y)\,dxdy \approx \frac{1}{3}\Big[g(\mathbf{z}_{12}) + g(\mathbf{z}_{23}) + g(\mathbf{z}_{31})\Big]\mathsf{area}(T)$$

where $\mathbf{z}_{ij}$ represents the midpoint of the side of the triangle with vertices z_i and z_j. Higher accurate rules can be found in standard texts on finite element methods.