Computational Methods in Aerospace Engineering

David L. Darmofal
Aeronautics & Astronautics
M.I.T.
Cambridge, MA 02139

May 11, 2005

Contents

Lecture 1

Numerical Integration of Ordinary Differential Equations: An Introduction

Ordinary differential equations (ODE's) occur throughout science and engineering.

Example 1.1 *A model for the velocity u of a spherical object falling freely through the atmosphere can be derived by applying Newton's Law. Specifically,*

$$m_p u_t = m_p g - D(u) \tag{1.1}$$

where m_p is the mass of the particle, g is the gravity, and D is the aerodynamic drag acting on the particle. For low speeds, this drag can be modeled as,

$$D = \frac{1}{2}\rho_g \pi a^2 u^2 C_D(Re) \tag{1.2}$$

$$Re = \frac{2\rho_g u a}{\mu_g} \tag{1.3}$$

$$C_D = \frac{24}{Re} + \frac{6}{1+\sqrt{Re}} + 0.4 \tag{1.4}$$

where ρ_g and μ_g are the density and dynamic viscosity of the atmosphere, a is the sphere radius, Re is the Reynolds number for the sphere, and C_D is the drag coefficient. To complete the problem specification, an initial condition is required on the velocity. For example, at time $t = 0$, let $u(0) = u_0$. By solving Equation 1.1 with this initial condition, the velocity at any time $u(t)$ can be found.

A general ODE is typically written in the form,

$$u_t = f(u, t), \tag{1.5}$$

where $f(u, t)$ is the forcing function that results in the evolution of u in time. When $f(u, t)$ depends on u in a nonlinear manner, then the ODE is called a nonlinear ODE.

Example 1.2 *For Example 1.1, the forcing function is,*

$$f(u,t) = g - \frac{1}{m_p} D(u)$$

From the definition of $D(u)$, $f(u,t)$ is nonlinear in u. Note also that in this example, f does not depend on t directly rather $f(u,t) = f(u)$.

Although the use of numerical integration is most important for nonlinear ODE's (since analytic solutions rarely exist), the study of numerical methods applied to linear ODE's is often quite helpful in understanding the behavior for nonlinear problems. The general form For a single, linear ODE, is

$$u_t = \lambda(t)u + g(t), \tag{1.6}$$

where $\lambda(t)$ is independent of u. When $\lambda(t)$ is a constant, the ODE is referred to as a linear ODE with constant coefficients.

In many situations, the linear ODE is derived by linearizing a nonlinear ODE about a constant state. Specifically, define the dependent state $u(t)$ as a sum of u_0 and a perturbation, $\tilde{u}(t)$,

$$u(t) = u_0 + \tilde{u}(t). \tag{1.7}$$

A linearized equation for the evolution of $\tilde{u}$ can be derived by substitution of this into Equation 1.5:

$$\begin{aligned} u_t &= f(u,t), \\ \tilde{u}_t &= f(u_0 + \tilde{u}, t) \\ \tilde{u}_t &= f(u_0, 0) + \left.\frac{\partial f}{\partial u}\right|_{u_0,0} \tilde{u} + \left.\frac{\partial f}{\partial t}\right|_{u_0,0} t + O(t^2, \tilde{u}t, \tilde{u}^2). \end{aligned}$$

Thus, when t and $\tilde{u}$ are small, the perturbation satisfies the linear equation,

$$\tilde{u}_t \approx f(u_0, 0) + \left.\frac{\partial f}{\partial u}\right|_{u_0,0} \tilde{u} + \left.\frac{\partial f}{\partial t}\right|_{u_0,0} t \tag{1.8}$$

Comparing Equation 1.6 to 1.8, we see that in this example,

$$\begin{aligned} \lambda(t) &= \left.\frac{\partial f}{\partial u}\right|_{u_0,0}, \\ g(t) &= f(u_0, 0) + \left.\frac{\partial f}{\partial t}\right|_{u_0,0} t \end{aligned}$$

Note, this is a constant coefficient linear ODE.

Example 1.3 *For the falling sphere problem in Examples 1.1 and 1.2, a linear ODE can be derived by linearizing about the initial velocity u_0. As shown above, this requires the calculation of $\partial f/\partial u$ and $\partial f/\partial t$. For the sphere,*

$$\frac{\partial f}{\partial u} = \frac{\partial}{\partial u}\left[g - \frac{1}{m_p} D(u)\right] = -\frac{1}{m_p}\frac{\partial D}{\partial u}$$

The value of $\partial D/\partial u$ is

$$\begin{aligned}\frac{\partial D}{\partial u} &= \frac{\partial}{\partial u}\left[\frac{1}{2}\rho_g \pi a^2 u^2 C_D(Re)\right], \\ &= \rho_g \pi a^2 u C_D(Re) + \frac{1}{2}\rho_g \pi a^2 u^2 \frac{\partial C_D}{\partial Re}\frac{\partial Re}{\partial u},\end{aligned}$$

and $\partial C_D/\partial Re$ and $\partial Re/\partial u$ can be found from their definitions given in Example 1.1. Also, since f does not directly depend on t for this problem, $\partial f/\partial t = 0$.

1.1 The Forward Euler Method

We now consider our first numerical method for ODE integration, the forward Euler method. The general problem we wish to solve is to approximate the solution $u(t)$ for Equation 1.5 with an appropriate initial condition, $u(0) = u_0$. Usually, we are interested in approximating this solution over some range of t, say from $t = 0$ to $t = T$. Or, we may not know a precise final time but wish to integrate forward in time until an event occurs (e.g. the problem reaches a steady state). In either case, the basic philosophy of numerical integration using finite difference methods is to start from a known initial state, $u(0)$, and somehow approximate the solution a small time forward, $u(\Delta t)$ where Δt is a small time increment. Then, we repeat this process and move forward to the next time to find, $u(2\Delta t)$, and so on. Initially, we will consider the situation in which Δt is fixed for the entire integration from $t = 0$ to T. However, the best methods for solving ODE's tend to be adaptive methods in which Δt is adjusted depending on the current approximation.

Before moving on to the specific form of the forward Euler method, let's put some notations in place. Superscripts will be used to indicate a particular iteration. Thus, assuming constant Δt,

$$t^n = n\Delta t.$$

The approximation from the numerical integration will be defined as v. Thus, using the superscript notation,

$$v^n = \text{the approximation of } u(t^n).$$

Now, let's derive the forward Euler method. There are several ways to motivate the forward Euler method. We will start with an approach based on Taylor series. Specifically, the Taylor expansion of $u(t^{n+1})$ about t^n is,

$$u(t^{n+1}) = u(t^n) + \Delta t u_t(t^n) + \frac{1}{2}\Delta t^2 u_{tt}(t^n) + O(\Delta t^3).$$

Using only the first two terms in this expansion,

$$u(t^{n+1}) \approx u(t^n) + \Delta t u_t(t^n).$$

Finally, the term $u_t(t^n)$ is in fact just $f(u(t^n), t^n)$ since the governing equation is Equation 1.5. Thus,

$$u(t^{n+1}) \approx u(t^n) + \Delta t f(u(t^n), t^n). \tag{1.9}$$

In-class Discussion 1.1 (Graphical interpretation of Equation 1.9)

Since we do not know $u(t^n)$, we will instead use the approximation from the previous timestep, v^n. Thus, the forward Euler algorithm is,

$$v^{n+1} = v^n + \Delta t f(v^n, t^n) \qquad \text{for} \qquad n \geq 0, \tag{1.10}$$

and $v^0 = u(0)$.

Example 1.4 *Now, let's apply the forward Euler method to solving the falling sphere problem. Suppose the sphere is actually a small particle of ice falling in the atmosphere at an altitude of approximately 3000 meters. Specifically, let's assume the radius of the particle is* $a = 0.01m$. *Then, since the density of ice is approximately* $\rho_p = 917\, kg/m^3$, *the mass of the particle can be calculated from,*

$$m_p = \rho_p \, Volume_p = \rho_p \frac{4}{3}\pi a^3 = 0.0038\, kg$$

At that altitude, the properties of the atmosphere are:

$$\begin{aligned} \rho_g &= 0.9\, kg/m^3 \\ \mu_g &= 1.69E\text{-}5\, kg/(m\, sec) \\ g &= 9.8 m/sec^2 \end{aligned}$$

We expect the particle to accelerate until it reaches its terminal velocity which will occur when the drag force is equal to the gravitational force. But, a priori, we do not know how long that will take (see In-class Discussion 1.2 for someways to make this estimate). For now, let's set $T = 25\, sec$ *and use a timestep of* $\Delta t = 0.25\, sec$. *The results are shown in Figure 1.1.*

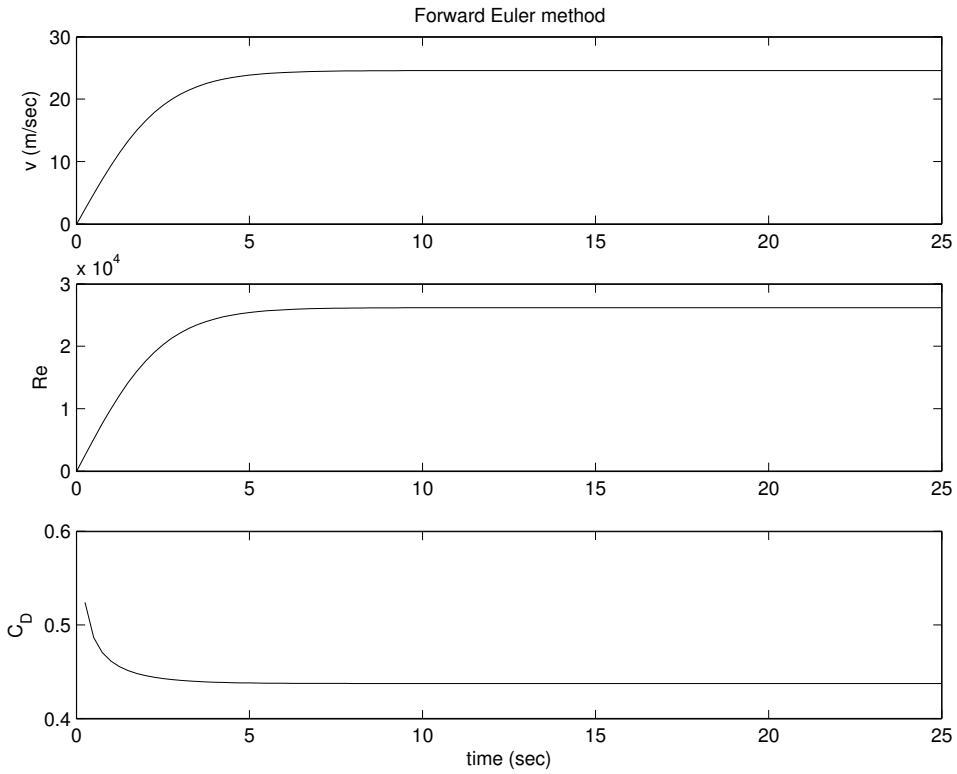


Figure 1.1: Behavior of velocity, Reynolds number, and drag coefficient as a function of time for an ice particle falling through the atmosphere. Simulation performed using the forward Euler method with $\Delta t = 0.25\,sec.$

In-class Discussion 1.2 (Estimating time to reach terminal velocity)

1.2 The Midpoint Method

Now, let's look at a second integration method known as the midpoint method. For this method, we will use a slightly different point of view to derive it. Specifically, let's start from the definition of a derivative,

$$u_t(t) = \lim_{\Delta t \to 0} \frac{u(t + \Delta t) - u(t - \Delta t)}{2\Delta t}$$

Now, instead of taking the limit, assume a finite Δt. Then, we end up with an approximation to du/dt:

$$u_t(t) \approx \frac{u(t + \Delta t) - u(t - \Delta t)}{2\Delta t} \qquad \text{for small } \Delta t$$

Then, we can re-arrange this to the following estimate for $u(t + \Delta t)$,

$$u(t + \Delta t) \approx u(t - \Delta t) + 2\Delta t u_t(t) \tag{1.11}$$

In-class Discussion 1.3 (Graphical interpretation of Equation 1.11)

Then, following the same process as in the forward Euler method, we arrive at the midpoint method,

$$v^{n+1} = v^{n-1} + 2\Delta t f(v^n, t^n) \qquad \text{for} \qquad n \geq 1. \tag{1.12}$$

However, because of the use of v^{n-1}, the midpont method can only be applied for $n \geq 1$. Thus, for the first timestep a different numerical method must be applied (e.g. the forward Euler method).

Example 1.5 *We will now solve the same problem as in Example 1.4 using the midpoint method. Using the same values of Δt and T as before, the results are shown in Figure 1.2. Clearly, something has gone wrong here as the results show non-physical oscillations. Perhaps the oscillations will disappear if we take a smaller timestep. To test out this hypothesis, let's re-run the midpoint method with $\Delta t = 0.025\,sec$ which is one-tenth the previous timestep. Those results are shown in Figure 1.3. Unfortunately, while the results are better, the oscillations are clearly still present. For this problem, clearly the forward Euler method is a better choice than the midpoint method. We will see why this has happened in a few lectures.*

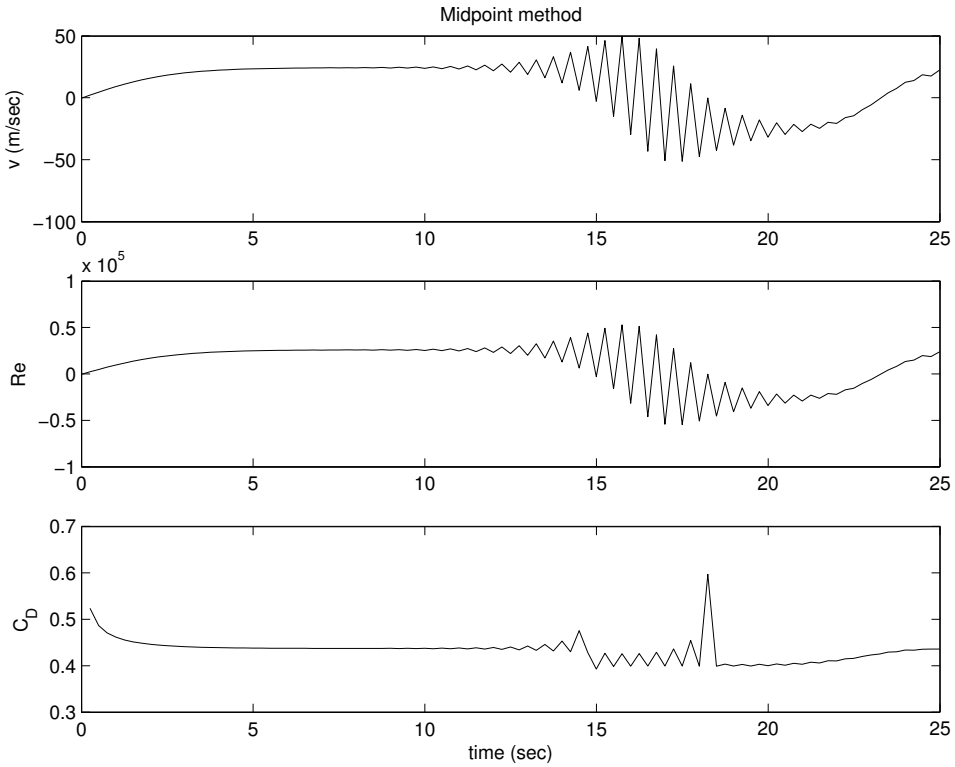


Figure 1.2: Behavior of velocity, Reynolds number, and drag coefficient as a function of time for an ice particle falling through the atmosphere. Simulation performed using the midpoint method with $\Delta t = 0.25\, sec.$

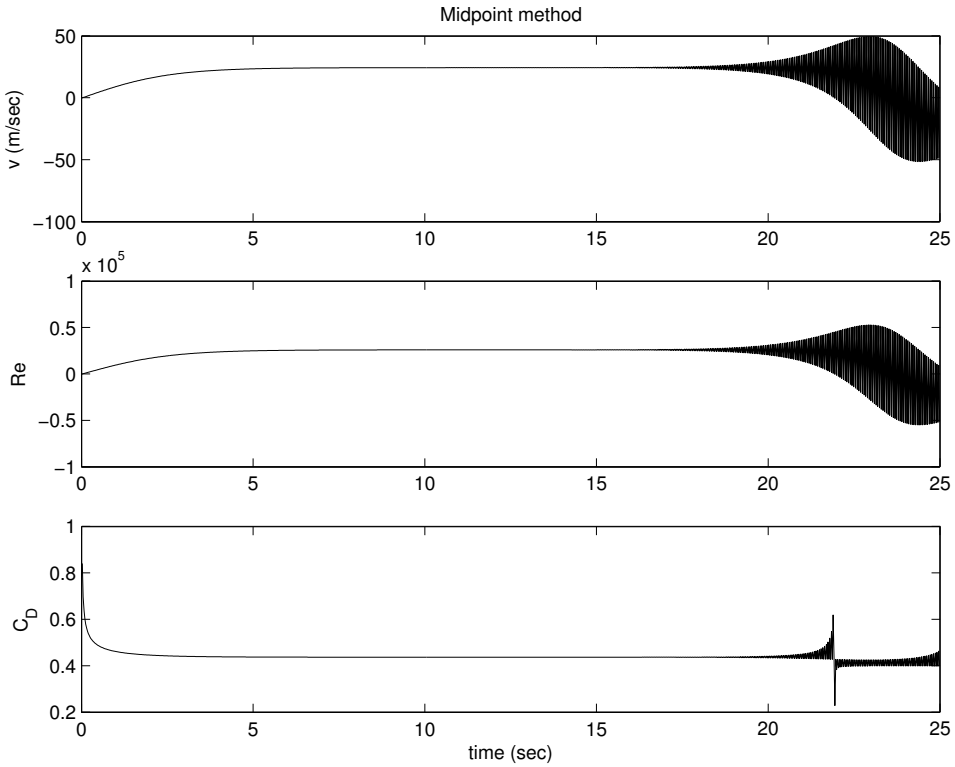


Figure 1.3: Behavior of velocity, Reynolds number, and drag coefficient as a function of time for an ice particle falling through the atmosphere. Simulation performed using the midpoint method with $\Delta t = 0.025\, sec$.

Lecture 2

Convergence and Accuracy

This lecture is a bit on the technical side, but the concepts introduced are critical to the analysis of finite difference methods for ODE's.

2.1 Convergence and Global Accuracy

As the timestep is decreased, i.e. $\Delta t \to 0$, the approximation from a finite difference method should converge to the solution of the ODE. This concept is known as convergence and is stated mathematically as follows:

Definition 2.1 (Convergence) *A finite difference method for solving,*

$$u_t = f(u,t) \qquad with \qquad u(0) = u_0$$

from $t = 0$ to T is convergent if

$$\max_{n=[0,T/\Delta t]} |v^n - u(n\Delta t)| \to 0 \qquad as \qquad \Delta t \to 0.$$

While convergence is a clear requirement for a good numerical method, the rate at which the method converges is also important. This rate is known as the global order of accuracy.

Definition 2.2 (Global Order of Accuracy) *A method has a global order of accuracy of p if,*

$$\max_{n=[0,T/\Delta t]} |v^n - u(n\Delta t)| \leq O(\Delta t^p) \qquad as \qquad \Delta t \to 0,$$

for any $f(u,t)$ that has p continuous derivatives (i.e. up to and including $\partial^p f/\partial t^p$ and $\partial^p f/\partial u^p$).

Thus, methods with higher p will converge to $u(t)$ more rapidly than those methods with lower p.

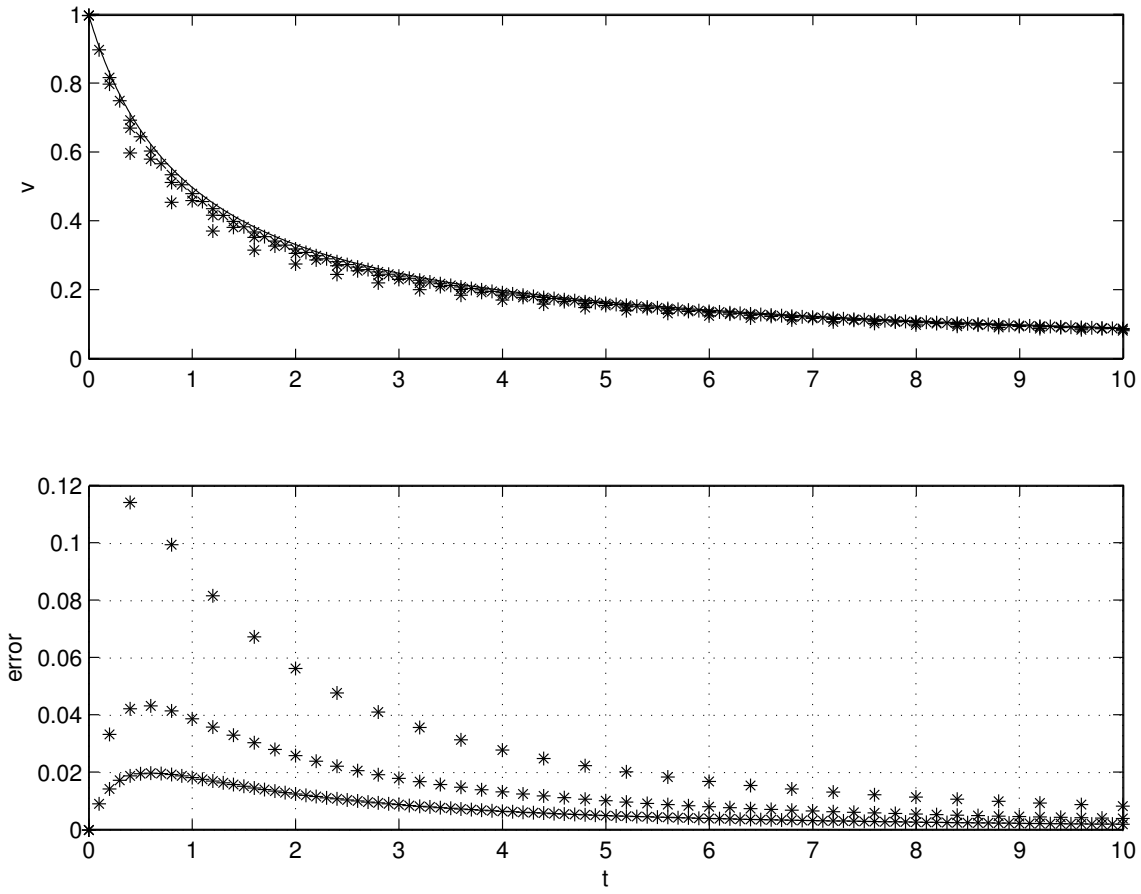


Figure 2.1: Forward Euler solution for $u_t = -u^2$ with $u(0) = 1$ with $\Delta t = 0.1$, 0.2, and 0.4. Forward Euler (symbols) and exact solution (line) are shown in first plot. Error is shown in second plot.

Example 2.1 *To demonstrate the ideas of global accuracy, we will consider an ODE with* $f = -u^2$ *and an initial condition of* $u(0) = 1$. *The solution to this ODE is* $u = (1+t)^{-1}$. *Now, let us apply the forward Euler method to solving this problem for* $t = 0$ *to* 10. *The approximate solutions for a range of* Δt *are shown Figure 2.1 along with the exact solution. The forward Euler solutions are clearly approaching the exact solution as* Δt *decreases. Furthermore, the error appears to be decreasing by approximately a factor of 2 for every factor of 2 decrease in* Δt. *For example, if we look at* $t = 4$, *the error is seen to be* 0.028, 0.013, *and* 0.0065 *for* $\Delta t = 0.4$, 0.2, *and* 0.1, *respectively. Thus, from these results, we would conclude that the global accuracy of the forward Euler method is* $p = 1$ *since the error is proportional to* Δt.

Example 2.2 *Now, let's apply the midpoint method on the problem from Example 2.1. Similar to the results observed in Example 1.5, the midpoint method shows an oscillatory behavior (this may be a little hard to see because of scale of the figure, but the midpoint results are basically oscillated about the exact solution, with the oscillations reducing for the smaller timesteps). Note that the timesteps used in these results are a factor of 10 smaller than those used with the forward Euler method in Example 2.1. Since the midpoint and the for-*

ward Euler method require essentially the same work per timestep, the midpoint results took about a factor of 10 more work than the forward Euler method for this problem. Another interesting aspect of these results is that the error is actually increasing as t increases (in the forward Euler results in Figure 2.1, the error decreased as t increased). Regardless, the method does appear convergent since as the timestep decreases, so are the errors. In fact, it appears that the errors are decreasing by a factor of 4 for a factor of 2 decrease in Δt*. For example, if we look at* $t = 4$*, the error (averaged to remove the oscillations) is seen to be approximately* 0.02, 0.005, *and* 0.00125 *for* $\Delta t = 0.04$, 0.02, *and* 0.01, *respectively. Thus, from these results, we would conclude that the global accuracy of the midpoint method is* $p = 2$ *since the error is proportional to* Δt^2*.*

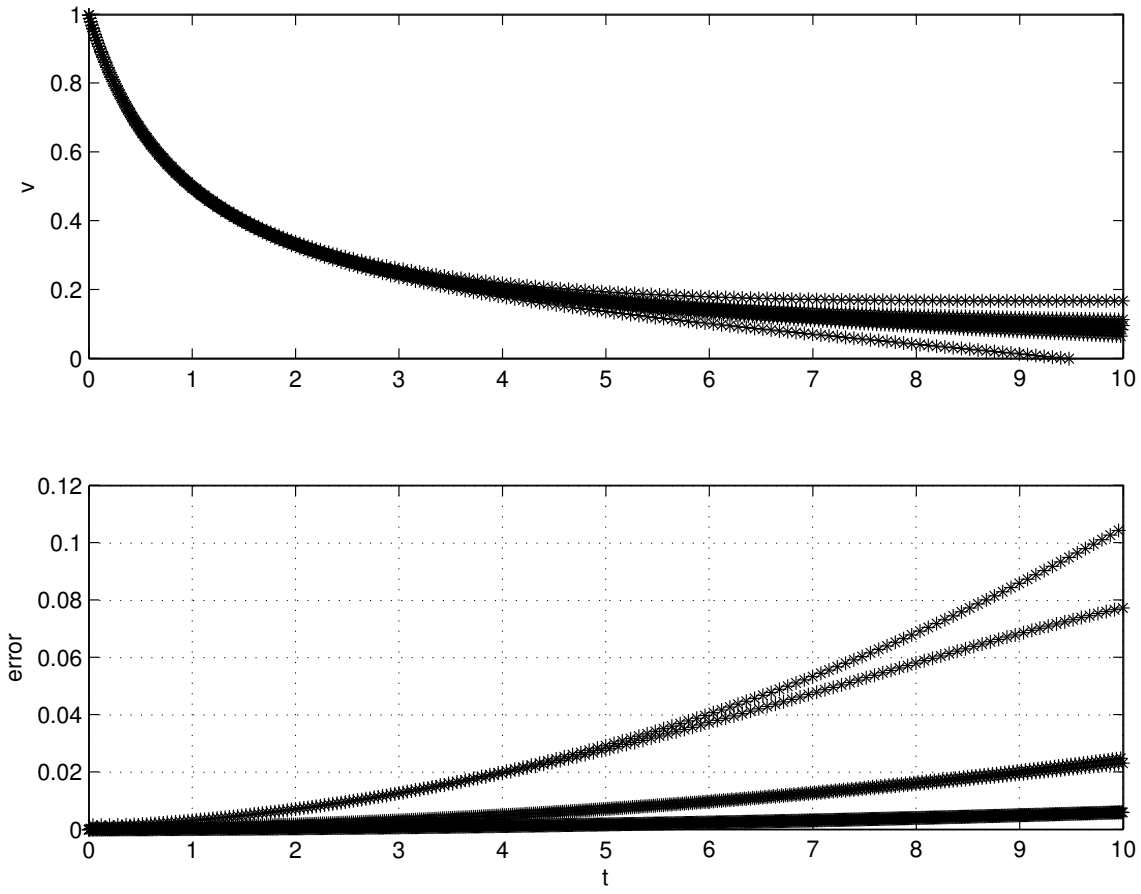


Figure 2.2: Midpoint solution for $u_t = -u^2$ with $u(0) = 1$ with $\Delta t = 0.01$, 0.02, and 0.04. Midpoint method (symbols) and exact solution (line) are shown in first plot. Error is shown in second plot.

2.2 Local Accuracy

The analysis of convergence and global accuracy usually relies on the analsysi of consistency and local accuracy. Both convergence/global accuracy and consistency/local accuracy are

related to the behavior of the error as $\Delta t \to 0$. However, while convergence/global accuracy is associated with the behavior of the error over a finite time (i.e. from $t = 0$ to T), consistency/local accuracy is associated with the behavior of the error for a single timestep. If we can quantify how much the error changes in a single timestep, then we will have an indication of how much the error could change over a series of timesteps. Specifically, let's write the solution error at $t = T$ as a sum of the change in error at each timestep,

$$e(T) = u(T) - v^{T/\Delta t} = \sum_{n=1}^{T/\Delta t} \Delta e^n,$$

where Δe^n is the change in the error from iteration $n-1$ to n (i.e. the local error). Suppose the local error is $O(\Delta t^{p+1})$, then the global error might be expected to behave as,

$$\begin{aligned} e(T) &= \sum_{n=1}^{T/\Delta t} \Delta e^n, \\ &= \sum_{n=1}^{T/\Delta t} O(\Delta t^{p+1}), \\ &= \frac{T}{\Delta t} O(\Delta t^{p+1}) \\ &= O(\Delta t^p). \end{aligned}$$

Thus, the global error would be one order less than the local error because the local errors sum for $T/\Delta t$ timesteps. However, the local errors do not have to sum this way if the numerical method is not stable. But, if a numerical method is both consistent and stable, this will be enough to guarantee convergence. For now, we concentrate on quantifying the local accuracy and leave the discussion of consistency and stability for another lecture.

The local error (usually called the local truncation error) is the difference between the approximate solution and the exact solution when using the exact solution for all of the required data. Let's consider the forward Euler method as an example. Recall, the forward Euler method is,

$$v^{n+1} = v^n + \Delta t f(v^n, t^n).$$

Thus, for the forward Euler method, $v^{n+1} = v^{n+1}(v^n, \Delta t, t^n)$. Then, if we substitute the exact solution into the right-hand side, we find,

$$v^{n+1}(u^n, \Delta t, t^n) = u^n + \Delta t f(u^n, t^n).$$

Recall our notation that u is the exact solution; in this discussion we use the superscript notation $u^n = u(n\Delta t)$ realizing that $u = u(t)$. The local truncation error for the forward Euler method is then,

$$\text{Local truncation error} \equiv v^{n+1}(u^n, \Delta t, t^n) - u^{n+1}. \tag{2.1}$$

Substitution gives,

$$\text{Local truncation error} = u^n + \Delta t f(u^n, t^n) - u^{n+1}.$$

The local order of accuracy is then found using a Taylor series expansion about $t = t^n$. Recall that $f(u^n, t^n) = u_t(t^n)$ and

$$u(t^{n+1}) = u(t^n) + \Delta t u_t(t^n) + \frac{1}{2}\Delta t^2 u_{tt}(t^n) + O(\Delta t^3).$$

Substitution gives the local truncation error as,

$$\begin{aligned} \text{Local truncation error} &= u^n + \Delta t f(u^n, t^n) - u^{n+1}, \\ &= u(t^n) + \Delta t u_t(t^n) - \left[u(t^n) + \Delta t u_t(t^n) + \frac{1}{2}\Delta t^2 u_{tt}(t^n) + O(\Delta t^3)\right] \\ &= -\frac{1}{2}\Delta t^2 u_{tt}(t^n) + O(\Delta t^3). \end{aligned}$$

Thus, the leading term of the local trunction error for the forward Euler method is $-\frac{1}{2}\Delta t^2 u_{tt}(t^n) = O(\Delta t^2)$. Based on our previous argument, we expect that the global accuracy of the forward Euler method should be $O(\Delta t)$ (i.e. first order accuracy). This was in fact observed in Example 2.1.

In-class Discussion 2.1 (Local accuracy of the midpoint method)

Definition 2.3 (Local Order of Accuracy) *Suppose we are given a numerical method for solving $u_t = f(u, t)$ which we write in the following form,*

$$v^{n+1} = N(v^{n+1}, v^n, v^{n-1}, \ldots, \Delta t)$$

For simplicity, the possible dependence on t at various n has been omitted in the definition of N (though it should be there). The local truncation error, τ, is defined as,

$$\tau \equiv N(u^{n+1}, u^n, u^{n-1}, \ldots, \Delta t) - u^{n+1}, \tag{2.2}$$

and the local order of accuracy p is,

$$|\tau| = O(\Delta t^{p+1}) \qquad as \qquad \Delta t \to 0.$$

Note: the local order of accuracy is defined to be one less than the order of the leading term of the local truncation error so that the local and global accuracy will be the same.

By the definition of the local order of accuracy, we see that the forward Euler method is first order ($p = 1$) and the midpoint method is second order ($p = 2$).

Lecture 3

Convergence of Multi-Step Methods

In Lecture 2, we began the discussion of convergence. In this lecture, we will complete that discussion for the class of numerical methods known as multi-step methods (a class that includes the forward Euler and midpoint methods we have previously discussed).

3.1 Multi-step Methods

The class of finite difference methods known as multi-step methods is one of the most widely-used approaches for solving ordinary differential equations, and forms the basis for solving partial differential equations as well.

Definition 3.1 (Multi-step Methods) *The generic form of an s-step multi-step method is,*

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} = \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i}.$$

*A multi-step method with $\beta_0 = 0$ is known as an **explicit** method since in this case the new value v^{n+1} can be determined as an explicit function of known values (i.e. from v^i and f_i with $i \leq n$). A multi-step method with $\beta_0 \neq 0$ is known as an **implicit** method since in this case the new value v^{n+1} is an implicit function of itself through the forcing function, $f^{n+1} = f(v^{n+1}, t^{n+1})$.*

Example 3.1 *Using the notation given in Definition 3.1, the forward Euler method is:*

$$\alpha_1 = -1 \quad all\ other \quad \alpha_i = 0$$

$$\beta_1 = 1 \quad all\ other \quad \beta_i = 0$$

Example 3.2 *Using the notation given in Definition 3.1, the midpoint method is:*

$$\alpha_2 = -1 \quad all\ other \quad \alpha_i = 0$$

$$\beta_1 = 2 \quad all\ other \quad \beta_i = 0$$

Example 3.3 *In this example, we will derive the most accurate multi-step method of the following form:*

$$v^{n+1} + \alpha_1 v^n + \alpha_2 v^{n-1} = \Delta t \left[\beta_1 f^n + \beta_2 f^{n-1}\right]$$

The local truncation error for this method is,

$$\tau = -\alpha_1 u^n - \alpha_2 u^{n-1} + \Delta t \left[\beta_1 f^n + \beta_2 f^{n-1}\right] - u^{n+1}$$

Substitution of $f^n = u_t^n$ and $f^{n-1} = u_t^{n-1}$ gives,

$$\tau = -\alpha_1 u^n - \alpha_2 u^{n-1} + \Delta t \left[\beta_1 u_t^n + \beta_2 u_t^{n-1}\right] - u^{n+1}$$

Then, Taylor series about $t = t^n$ are substituted for u^{n-1}, u_t^{n-1}, and u^{n+1} to give,

$$\begin{aligned}
\tau \;=\; & -\alpha_1 u^n - \alpha_2 \left[u^n - \Delta t u_t^n + \frac{1}{2}\Delta t^2 u_{tt}^n - \frac{1}{6}\Delta t^3 u_{ttt}^n + \frac{1}{24}\Delta t^4 u_{tttt}^n + O(\Delta t^5)\right] \\
& + \Delta t \beta_1 u_t^n + \Delta t \beta_2 \left[u_t^n - \Delta t u_{tt}^n + \frac{1}{2}\Delta t^2 u_{ttt}^n - \frac{1}{6}\Delta t^3 u_{tttt}^n + O(\Delta t^4)\right] \\
& - \left[u^n + \Delta t u_t^n + \frac{1}{2}\Delta t^2 u_{tt}^n + \frac{1}{6}\Delta t^3 u_{ttt}^n + \frac{1}{24}\Delta t^4 u_{tttt}^n + O(\Delta t^5)\right]
\end{aligned}$$

Next, collect the terms in powers of Δt, which gives the following coefficients:

$$\begin{array}{rccccccccc}
u^n\text{:} & - & \alpha_1 & - & \alpha_2 & & & & - & 1 \\
\Delta t u_t^n\text{:} & & & & \alpha_2 & + & \beta_1 & + & \beta_2 & - & 1 \\
\Delta t^2 u_{tt}^n\text{:} & & & - & \frac{\alpha_2}{2} & & & - & \beta_2 & - & \frac{1}{2} \\
\Delta t^3 u_{ttt}^n\text{:} & & & & \frac{\alpha_2}{6} & & & + & \frac{\beta_2}{2} & - & \frac{1}{6} \\
\Delta t^4 u_{tttt}^n\text{:} & & & - & \frac{\alpha_2}{24} & & & - & \frac{\beta_2}{6} & - & \frac{1}{24}
\end{array}$$

To find the most accurate multi-step method of the given form, we solve for the values of α_1, α_2, β_1, and β_2 that result in the coefficients of the first four terms being identically zero. The resulting values are:

$$\alpha_1 = 4 \qquad \alpha_2 = -5 \qquad \beta_1 = 4 \qquad \beta_2 = 2$$

Note, with these values, the leading error term is $-\frac{1}{6}\Delta t^4 u_{tttt}^n$. Thus, the scheme is third order accurate ($p = 3$).

3.2 Dahlquist Equivalence Theorem

In order for a multi-step method to be convergent (as described in Definition 2.1), two conditions must be met:

Consistency: In the limit of $\Delta t \to 0$, the method must be a consistent discretization of the ordinary differential equation.

Stability: In the limit of $\Delta t \to 0$, the method must not have solutions that can grow unbounded as $n = T/\Delta t \to \infty$.

The Dahlquist Equivalence Theorem in fact guarantees that a consistent and stable multi-step method is convergent, and vice-versa:

Theorem 3.1 (Dahlquist Equivalence Theorem) *A multi-step method is convergent if and only if it is consistent and stable.*

3.3 Consistency

As given in Definition 3.1, a s-step multi-step method can be written as,

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} - \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i} = 0,$$

where the forcing terms have been moved to the left-hand side. Substituting the exact solution, $u(t)$, into the left-hand side will produce a remainder which is in fact the opposite of the truncation error (see Equation 2.2),

$$u^{n+1} + \sum_{i=1}^{s} \alpha_i u^{n+1-i} - \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i} = u^{n+1} - N(u^{n+1}, u^n, \ldots, \Delta t) = -\tau \tag{3.1}$$

If we only require that $\tau \to 0$ (i.e. $\tau = O(\Delta t)$) as $\Delta t \to 0$, the method will not generally be consistent with the ODE. To see why, note that in the limit of $\Delta t \to 0$, the forcing terms will vanish since they are scaled by Δt. Thus, $\tau \to 0$ would place a constraint only on the α's. Let's look at that constraint on the α's to build some insight. Substituting Taylor series about $t = t^n$ for the values of u gives,

$$u^{n+1} + \sum_{i=1}^{s} \alpha_i u^{n+1-i} = \left(1 + \sum_{i=1}^{s} \alpha_i\right) u^n + O(\Delta t).$$

Thus, for $\tau \to 0$ as $\Delta t \to 0$ requires,

$$1 + \sum_{i=1}^{s} \alpha_i = 0. \tag{3.2}$$

This constraint can be interpreted as requiring a constant solution, i.e. $u(t) =$ constant, to be a valid solution of the multi-step method. Clearly, this is not enough to guarantee consistency with the ODE since the ODE requires $u_t = f(u, t)$.

To achieve a consistent discretization, we force $\tau/\Delta t \to 0$ (i.e. $\tau = O(\Delta t^2)$). This stronger constraint can be shown to enforce that the ODE is satisfied in the limit of $\Delta t \to 0$:

$$\begin{aligned}
\frac{\tau}{\Delta t} &= \frac{N(u^{n+1}, u^n, \ldots, \Delta t) - u^{n+1}}{\Delta t} \\
&= \frac{N(u^{n+1}, u^n, \ldots, \Delta t) - (u^n + \Delta t u_t^n + O(\Delta t^2))}{\Delta t} \\
&= \frac{N(u^{n+1}, u^n, \ldots, \Delta t) - u^n}{\Delta t} - u_t^n + O(\Delta t) \\
&= \frac{N(u^{n+1}, u^n, \ldots, \Delta t) - u^n}{\Delta t} - f(u^n, t^n) + O(\Delta t)
\end{aligned}$$

Thus, in the limit of $\tau/\Delta t \to 0$ as $\Delta t \to 0$, then the slope of the numerical method (i.e. the first term) must be equal to the forcing at t^n. In other words, the multi-step discretization would satisfy the governing equation in the limit. An equivalent way to write this consistency constraint is,

$$\lim_{\Delta t \to 0} \frac{1}{\Delta t} \left[u^{n+1} + \sum_{i=1}^{s} \alpha_i u^{n+1-i} \right] - \sum_{i=0}^{s} \beta_i f^{n+1-i} = u_t(t^n) - f(u(t^n), t^n) = 0. \tag{3.3}$$

In terms of the local accuracy, consistency requires that the multi-step method be at least first-order ($p = 1$) since $\tau = O(\Delta t^{p+1})$ and consistency requires that $\tau/\Delta t = O(\Delta t^p)$ must go to zero (i.e. $p \geq 1$).

3.4 Stability

The remaining issue to determine is whether the solutions to the multi-step method can grow unbounded as $\Delta t \to 0$ for finite time T. Consider again the s-step multi-step method:

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} = \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i}.$$

In the limit of $\Delta t \to 0$, the multi-step approximation will satisfy the following recurrence relationship,

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} = 0. \tag{3.4}$$

This recurrence relationship can be viewed a providing the characteristic or unforced behavior of the multi-step method. In terms of stability, the question is whether or not the solutions to Equation 3.4 can grow unbounded.

Definition 3.2 (Stability) *A multi-step method is stable (also known as zero stable) if all solutions to*

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} = 0,$$

are bounded as $n \to \infty$.

To determine if a method if stable, we assume that the solution to the recurrence has the following form,

$$v^n = v^0 z^n,$$

where the superscript in the z^n term is in fact a power. Note: z can be a complex number. If the recurrence relationship has solutions with $|z| > 1$, then the multi-step method would be unstable.

Example 3.4 *In Example 3.3, the most accurate two-step, explicit method was found to be,*

$$v^{n+1} + 4v^n - 5v^{n-1} = \Delta t \left(4f^n + 2f^{n-1} \right).$$

We will determne if this algorithm is stable. The recurrence relationship is,

$$v^{n+1} + 4v^n - 5v^{n-1} = 0.$$

Then, substitution of $v^n = v^0 z^n$ *gives,*

$$z^{n+1} + 4z^n - 5z^{n-1} = 0.$$

Factoring this relationship gives,

$$z^{n-1}\left(z^2 + 4z - 5\right) = z^{n-1}(z-1)(z+5) = 0.$$

Thus, the recurrence relationship has roots at $z = 1$, $z = -5$, *and* $z = 0$ *(*$n-1$ *of these roots). The root at* $z = -5$ *will grow unbounded as* n *increases so this method is not stable. By the Dahlquist Equivalence Theorem, this means the method is not convergent (even though it has local accuracy* $p = 3$ *and is therefore consistent).*

To demonstrate the lack of convergence for this method (due to its lack of stability), we again consider the solution of $u_t = -u^2$ *with* $u(0) = 1$. *These results are shown in Figure 3.1 These results clearly show the instability. Note that the solution oscillates as is expected since the large parasitic root is negative (*$z = -5$*). Furthermore, decreasing* Δt *from* 0.1 *to* 0.05 *only causes the instability to manifest itself in shorter time (though the same number of steps). Clearly, though the method is consistent, it will not converge because of this instability.*

In-class Discussion 3.1 (Stability of the midpoint method)

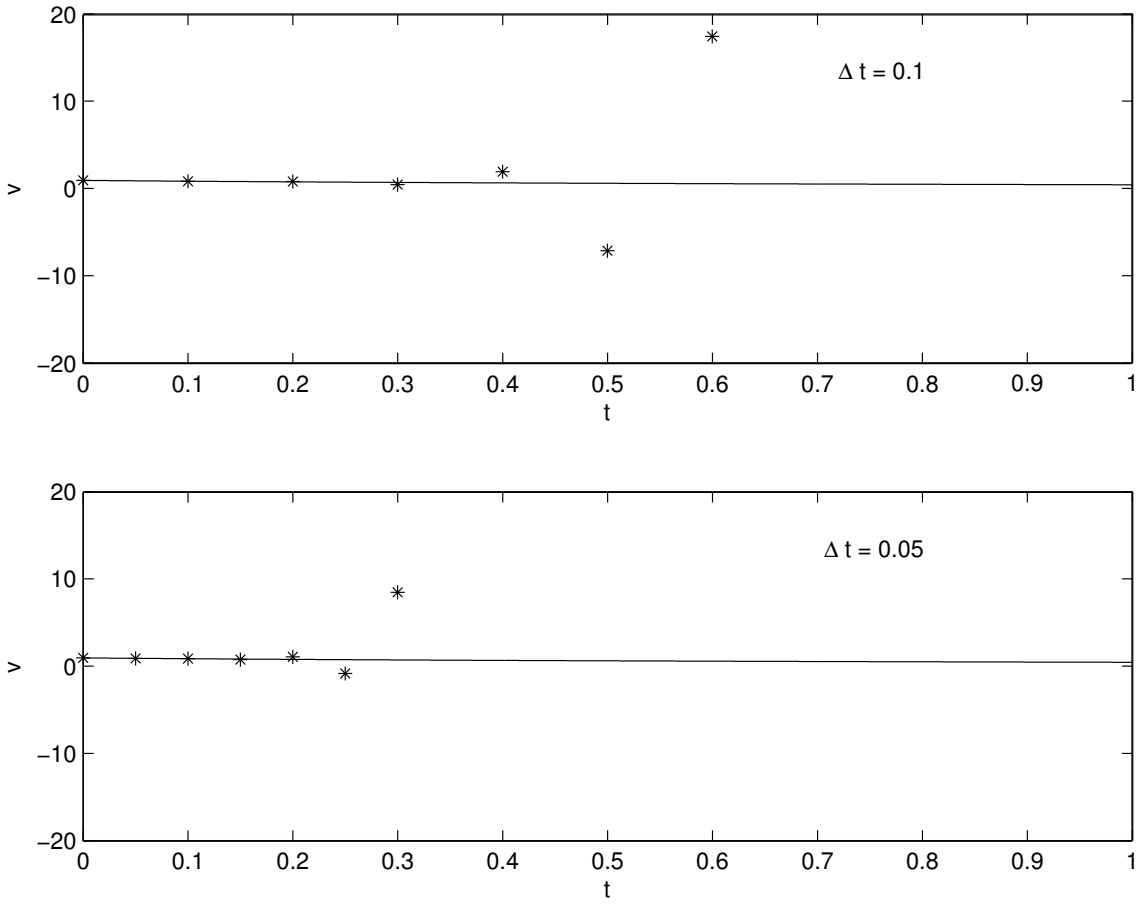


Figure 3.1: Most-accurate explicit, two-step multi-step method applied to $\dot{u} = -u^2$ with $u(0) = 1$ with $\Delta t = 0.1$ (upper plot) and 0.05 (lower plot).

Lecture 4

Systems of ODE's and Eigenvalue Stability

Until know, we have only addressed the integration of a single ODE. In this lecture, we consider numerical methods for systems of ODE's.

4.1 Nonlinear Systems

For a system of ODE's, we have the same canonical form as for a scalar (see Equation 1.5),

$$u_t = f(u, t), \tag{4.1}$$

except that u and f are vectors of the same length, d:

$$u = [u_1, u_2, u_3, \cdots, u_d]^T \qquad f = [f_1, f_2, f_3, \cdots, f_d]^T$$

Example 4.1 *Nonlinear Pendulum*

One manner in which a system of ODE's occurs is for higher-order ODE's. A classic example of this are second-order oscillators such as a pendulum. The nonlinear dynamics of a pendulum of length L satisfy the following second-order system of equations:

$$\theta_{tt} + \frac{g}{L}\sin\theta = 0. \tag{4.2}$$

To transform this into a system of first-order equations, we define the angular rate, ω,

$$\theta_t = \omega.$$

Then, Equation 4.2 becomes,

$$\omega_t + \frac{g}{L}\sin\theta = 0.$$

For this example,

$$u = \begin{pmatrix} \omega \\ \theta \end{pmatrix} \qquad f = \begin{pmatrix} -\frac{g}{L}\sin\theta \\ \omega \end{pmatrix}$$

A forward Euler method was used to simulate the motion of a pendulum (with $L = 1$ m, $g = 9.8$ m/sec^2) released from rest at an angle of $45°$ at a timestep of $\Delta t = 0.02$ seconds. The results are shown in Figure 4.1. While the oscillatory motion is evident, the amplitude is growing which is not expected physically. This would indicate some kind of numerical stability problem. Note, however, that if a smaller Δt were used, the amplification would still be present but not as significant.

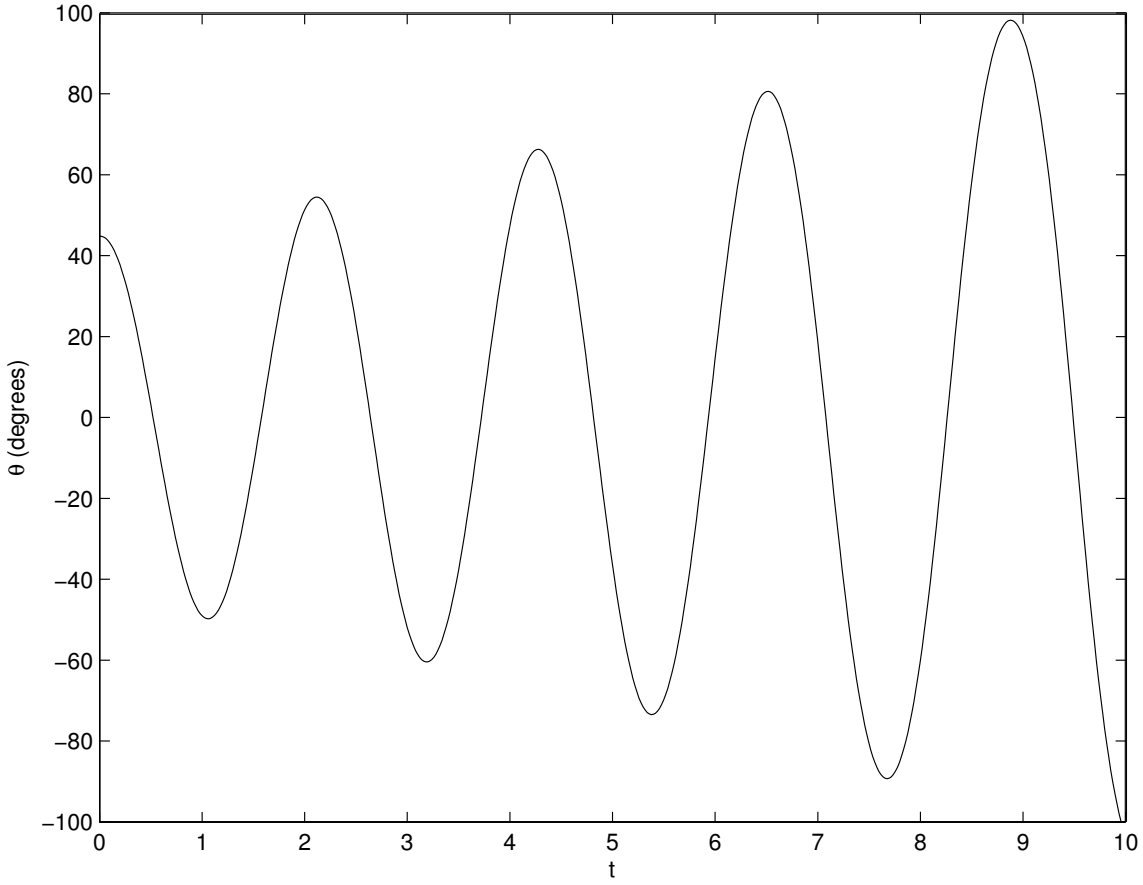


Figure 4.1: Forward Euler solution for nonlinear pendulum with $L = 1$ m, $g = 9.8$ m/sec^2, and $\Delta t = 0.02$ seconds.

The same problem was also simulated using the midpoint method. These results are shown in Figure 4.2. For this method and Δt choice, the oscillation amplitude is constant and indicates that the midpoint method is a better choice for this problem than the forward Euler method.

4.2 Linear Constant Coefficient Systems

The analysis of numerical methods applied to linear, constant coefficient systems can provide significant insight into the behavior of numerical methods for nonlinear problems. Consider

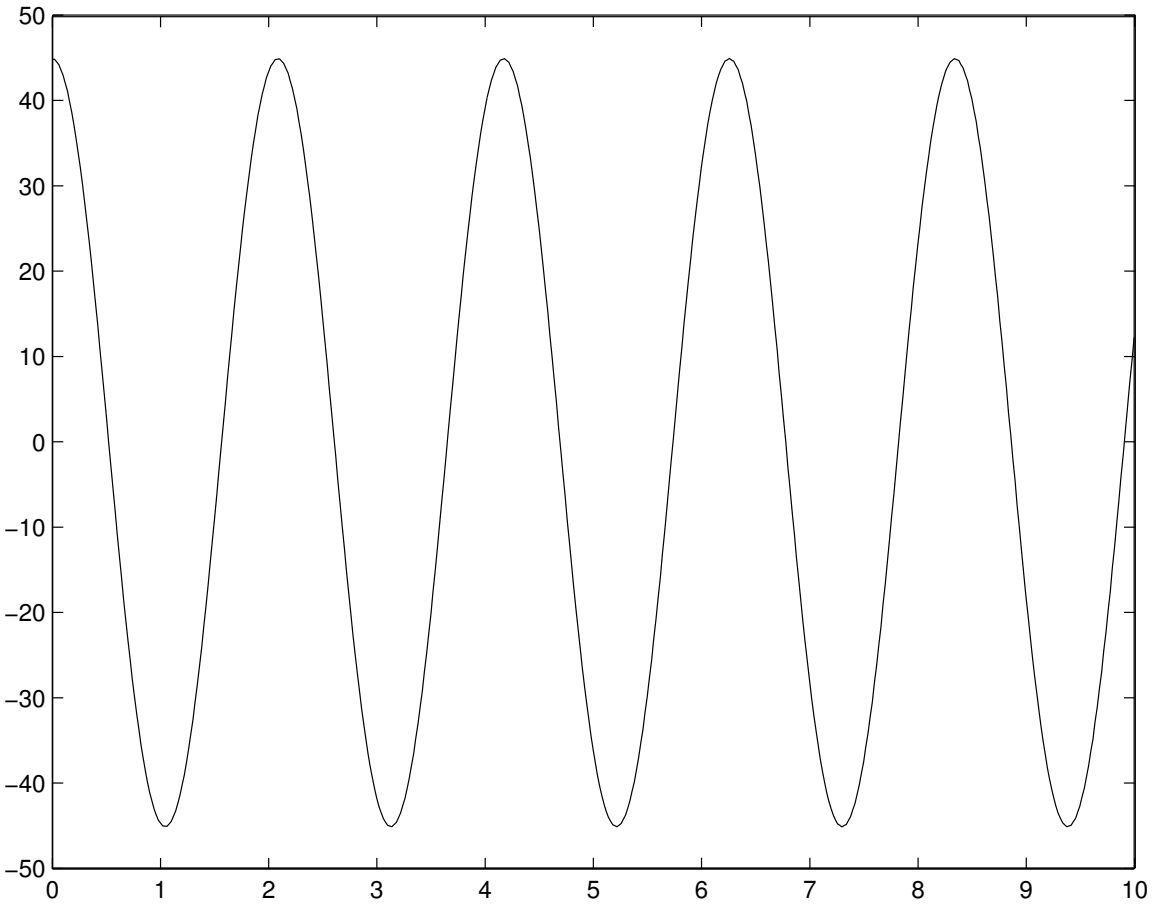


Figure 4.2: Midpoint solution for nonlinear pendulum with $L = 1$ m, $g = 9.8$ m/sec^2, and $\Delta t = 0.02$ seconds.

the following problem,

$$u_t = Au, \tag{4.3}$$

where A is a $d \times d$ matrix. Assuming that a complete set of eigenvectors exists, the matrix A can be decomposed as,

$$A = R\Lambda R^{-1}, \qquad \Lambda = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_d), \qquad R = \left(\begin{array}{c|c|c|c|c} & & & & \\ r_1 & r_2 & r_3 & r_4 & r_5 \\ & & & & \end{array} \right) \tag{4.4}$$

The solution to Equation 4.3 can be derived as follows,

$$\begin{aligned} u_t &= Au \\ u_t &= R\Lambda R^{-1}u \\ R^{-1}u_t &= \Lambda R^{-1}u \end{aligned}$$

Then, defining $w = R^{-1}u$,

$$w_t = \Lambda w.$$

This system of equations is actually uncoupled from each other, so that each of the eigenmodes has its own independent evolution equation,

$$(w_j)_t = \lambda_j w_j, \qquad \text{for each } i = 1 \text{ to } d$$

Since each of the eigenmodes has a solution $w_j(t) = w_j(0)\exp(\lambda_j t)$, then the solution for $u(t)$ can be written as,

$$u(t) = \sum_{i=1}^{d} w_j(0) r_j e^{\lambda_j t}. \tag{4.5}$$

Note that the eigenvalues are in general complex, $\lambda_j = \lambda_{j_r} + i\lambda_{j_i}$. The imaginary part of the eigenvalues determines the frequency of oscillations, and the real part of the eigenvalues determines the growth or decay rate. Specifically,

$$e^{\lambda t} = e^{(\lambda_r + i\lambda_i)t} = (\cos\lambda_i t + i \sin\lambda_i t)\, e^{\lambda_r t}.$$

Thus, when $\lambda_r > 0$, the solution will grow unbounded as $t \to \infty$.

4.3 Eigenvalue Stability for a Linear ODE

As we have seen, while numerical methods can be convergent, they can still exhibit instabilities as n increases for finite Δt. For example, when applying the midpoint method to either the ice particle problem in Example 1.5 or the simpler model problem in Example 2.2, instabilities were seen in both cases as n increased. Similarly, for the nonlinear pendulum problem in Example 4.1, the forward Euler method had a growing amplitude again indicating an instability. The key to understanding these results is to analyze the stability for finite Δt. This analysis is different than the stability analysis we performed in Section 3.4 since that analysis was for the limit of $\Delta t \to 0$.

Suppose we are interested in solving the linear ODE,

$$u_t = \lambda u.$$

Consider the Forward Euler method applied to this problem,

$$v^{n+1} = v^n + \lambda \Delta t v^n. \tag{4.6}$$

Similar to the zero stability analysis, we will assume that the solution has the following form,

$$v^n = g^n v^0, \tag{4.7}$$

where g is the amplification factor (and the superscript n acting on g is again raising to a power). As in the zero stability analysis, we wish to determine under what conditions $|g| > 1$ since this would mean that v^n will grow unbounded as $n \to \infty$. Substituting Equation 4.7 into Equation 4.6 gives,

$$g^{n+1} = (1 + \lambda \Delta t) g^n.$$

Thus, the only non-zero root of this equation gives,

$$g = 1 + \lambda \Delta t,$$

which is the amplification factor for the forward Euler method. Now, we must determine what values of $\lambda \Delta t$ lead to instability (or stability). A simple way to do this for multi-step methods is to solve for the stability boundary for which $|g| = 1$. To do this, let $g = e^{i\theta}$ (since $|e^{i\theta}| = 1$) where $\theta = [0, 2\pi]$. Making this substitution into the amplification factor,

$$e^{i\theta} = 1 + \lambda \Delta t \quad \Rightarrow \quad \lambda \Delta t = e^{i\theta} - 1.$$

Thus, the stability boundary for the forward Euler method lies on a circle of radius one centered at -1 along the real axis and is shown in Figure 4.3.

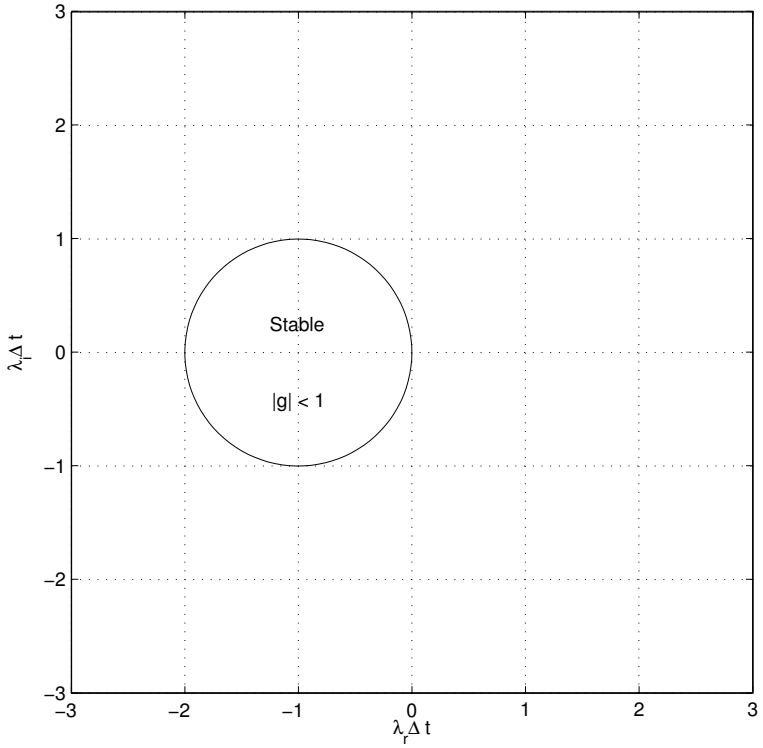


Figure 4.3: Forward Euler stability region

For a given problem, i.e. with a given λ, the timestep must be chosen so that the algorithm remains stable for $n \to \infty$. Let's consider some examples.

Example 4.2 *Let's return to the previous example, $u_t = -u^2$ with $u(0) = 1$. To determine the timestep restrictions, we must estimate the eigenvalue for this problem. As described in Lecture 1, linearizing this problem about a known state gives the eigenvalue as $\lambda = \partial f / \partial u = -2u$. Since the solution will decay from the initial condition (since $u_t < 0$ because $-u^2 < 0$), the largest magnitude of the eigenvalue occurs at the initial condition when $u(0) = 1$ and thus, $\lambda = -2$. Since this eigenvalue is a negative real number, the maximum Δt will occur*

at the maximum extent of the stability region along the negative real axis. Since this occurs when $\lambda\Delta t = -2$, this implies the $\Delta t < 1$. To test the validity of this analysis, the forward Euler method was run for a $\Delta t = 0.9$ and 1.1. The results are shown in Figure 4.4 which are stable for $\Delta t = 0.9$ but are unstable for $\Delta t = 1.1$.

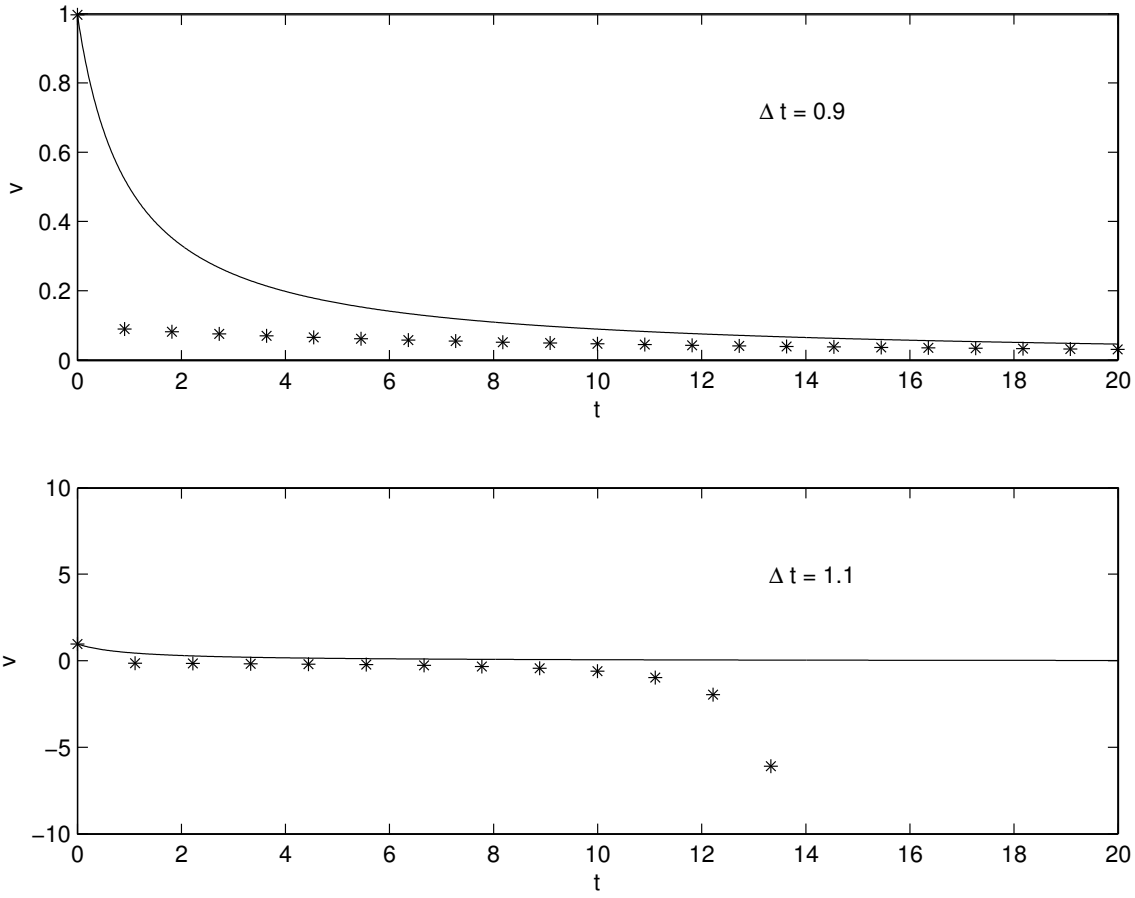


Figure 4.4: Forward Euler solution for $u_t = -u^2$ with $u(0) = 1$ with $\Delta t = 0.9$ and 1.1.

Example 4.3 *Next, let's consider the application of the forward Euler method to the pendulum problem. For this case, the linearization produces a matrix,*

$$\frac{\partial f}{\partial u} = \begin{pmatrix} 0 & -\frac{g}{L}\cos\theta \\ 1 & 0 \end{pmatrix}$$

The eigenvalues can be found from the roots of the determinant of $\partial f/\partial u - \lambda I$:

$$\begin{aligned} \det\left(\frac{\partial f}{\partial u} - \lambda I\right) &= \det\begin{pmatrix} -\lambda & -\frac{g}{L}\cos\theta \\ 1 & -\lambda \end{pmatrix} \\ &= \lambda^2 + \frac{g}{L}\cos\theta = 0 \\ \Rightarrow \quad \lambda &= \pm i\sqrt{\frac{g}{L}\cos\theta} \end{aligned}$$

Thus, we see that the eigenvalues will always be imaginary for this problem. As a result, since the forward Euler stability region does not contain any part of the imaginary axis (except the origin), no finite timestep exists which will be stable. This explains why the amplitude increases for the pendulum simulations in Figure 4.1.

In-class Discussion 4.1 (Midpoint method eigenvalue stability region)

Lecture 5

Stiffness and Implicit Methods

5.1 Stiffness

Stiffness is a general (though somewhat fuzzy) term to describe systems of equations which exhibit phenoma at widely-varying scales. For the ODE's we have been studying, this means widely-varying timescales.

One way for stiffness to arise is through a difference in timescales between a forcing timescale and any characteristic timescales of the unforced system. For example, consider the following problem:

$$u_t + 1000u = 100 \sin t, \qquad u(0) = 1. \tag{5.1}$$

The forcing term oscillates with a frequency of 1. By comparison, the unforced problem decays very rapidly since the eigenvalue is $\lambda = -1000$. Thus, the timescales are different by a factor of 1000.

Suppose we are only interested in the long time behavior of $u(t)$, not the initial transient. We would like to take a timestep that would be set by the requirements to resolve the $\sin t$ forcing. For example, one might expect that setting $\Delta t = 2\pi/100$ (which would result in 100 timesteps per period of the forcing) would be sufficient to have reasonable accuracy. However, if the method does not have a large eigenvalue stability region, this may not be possible. If a forward Euler method is applied to this problem, eigenvalue stability would limit the $\Delta t \leq 0.002$ (since the eigenvalue is $\lambda = -1000$ and the forward Euler stability region crosses the real axis at -2). The results from simulations for a variety of Δt using forward Euler are shown in Figure 5.1. For $\Delta t = 0.001$, the solution is well behaved and looks realistic. For $\Delta t = 0.0019$, the approach of eigenvalue instability is evident as there are oscillations during the first few iterations which eventually decay. For $\Delta t = 0.002$, the oscillations no longer decay but remain throughout the entire simulation. Finally, for $\Delta t = 0.0021$, the oscillations grow unbounded. A zoomed image of these results concentrating on the initial time behavior is shown in Figure 5.2.

A more efficient approach to numerically integrating this stiff problem would be to use a method with eigenvalue stability for large negative real eigenvalues. Implicit methods often have excellent stability along the negative real axis. The simplest implicit method is the backward Euler method,

$$v^{n+1} = v^n + \Delta t f(v^{n+1}, t^{n+1}). \tag{5.2}$$

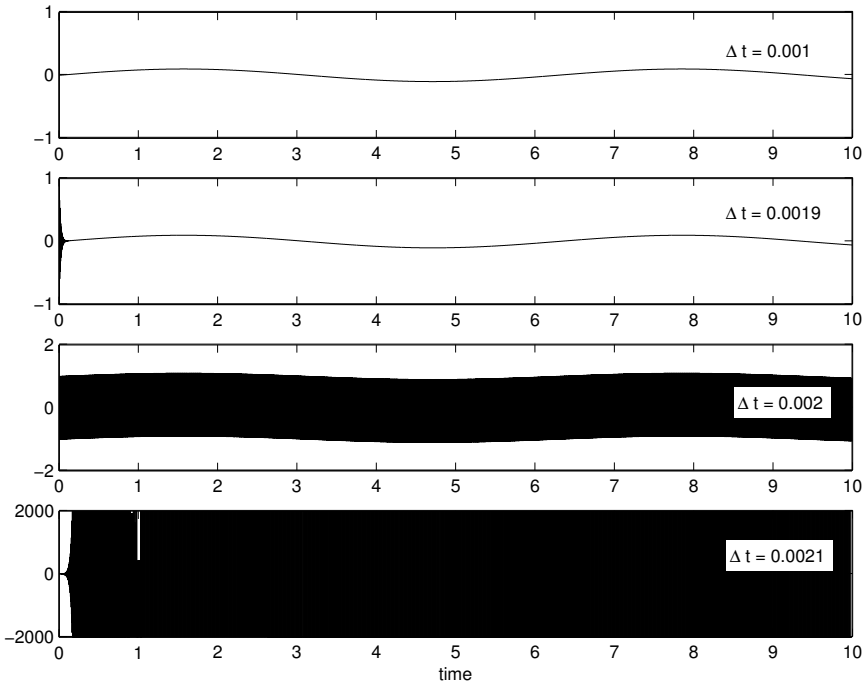


Figure 5.1: Forward Euler solution for $u_t + 1000u = 100 \sin t$ with $u(0) = 1$ at $\Delta t = 0.001$, 0.0019, 0.002, and 0.0021.

The backward Euler method is first order accurate ($p = 1$). The amplification factor for this method is,

$$g = \frac{1}{1 - \lambda \Delta t} \tag{5.3}$$

When λ is negative real, then $g < 1$ for all Δt. The eigenvalue stability region for the backward Euler method is shown in Figure 5.3. Only the small circular portion in the right-half plane is unstable while the entire left-half plane is stable. Results from the application of the backward Euler method to Equation 5.1 are shown in Figure 5.4. The excellent stability properties of this method are clearly seen as the solution looks acceptable for all of the tested Δt. Clearly, for the larger Δt, the initial transient is not accurately simulated, however, it does not effect the stability. Thus, as long as the initial transient is not desired, the backward Euler method will likely be a more effective solution strategy than the forward Euler method for this problem.

Another popular implicit method is trapezoidal integration,

$$v^{n+1} = v^n + \frac{1}{2}\Delta t \left[f(v^{n+1}, t^{n+1}) + f(v^n, t^n) \right]. \tag{5.4}$$

Trapezoidal integration is second-order accurate ($p = 2$). The amplification factor is,

$$g = \frac{1 + \frac{1}{2}\lambda \Delta t}{1 - \frac{1}{2}\lambda \Delta t}. \tag{5.5}$$

The stability boundary for trapezoidal integration lies on the imaginary axis (see Figure 5.5). Again, this method is stable for the entire left-half plane thus it will work well for stiff

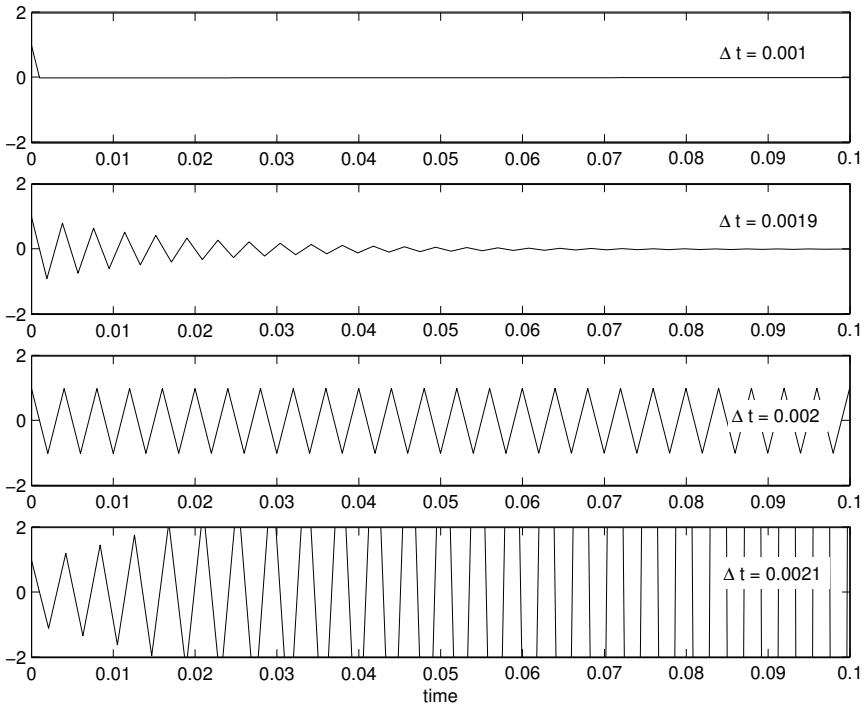


Figure 5.2: Forward Euler solution for $u_t + 1000u = 100 \sin t$ with $u(0) = 1$ at $\Delta t = 0.001$, 0.0019, 0.002, and 0.0021. Same results as in Figure 5.1 just showing the small t behavior in more detail.

problems.

The accuracy of the forward Euler, backward Euler, and trapezoidal integration methods are compared in Figure 5.6 for Equation 5.1. The error is computed as the maximum across all timesteps of the difference between numerical and exact solutions,

$$E = \max_{n=[0,T/\Delta t]} |v^n - u(n\Delta t)|.$$

These results show that the forward Euler method is first order accurate (since the slope on the log-log scaling is 1) once the Δt is small enough to have eigenvalue stability (for $\Delta t > 0.002$ the algorithm is unstable and the errors are essentially unbounded). In contrast, the two implicit methods have reasonable errors for all Δt's. As the Δt become small, the slope of the backward Euler and trapezoidal methods become essentially 1 and 2 (indicating first and second order accuracy). Clearly, if high accuracy is required, the trapezoidal method will require fewer timesteps to achieve this accuracy.

Stiffness can also arise in linear or linearized systems when eigenvalues exist with significantly different magnitudes. For example,

$$u_t = Au, \qquad A = \begin{pmatrix} -1 & 1 \\ 0 & -1000 \end{pmatrix}.$$

The eigenvalues of A are $\lambda = -1$ and $\lambda = -1000$. Since the timestep must be set so that both eigenvalues are stable, the larger eigenvalue will dominate the timestep. The spectral

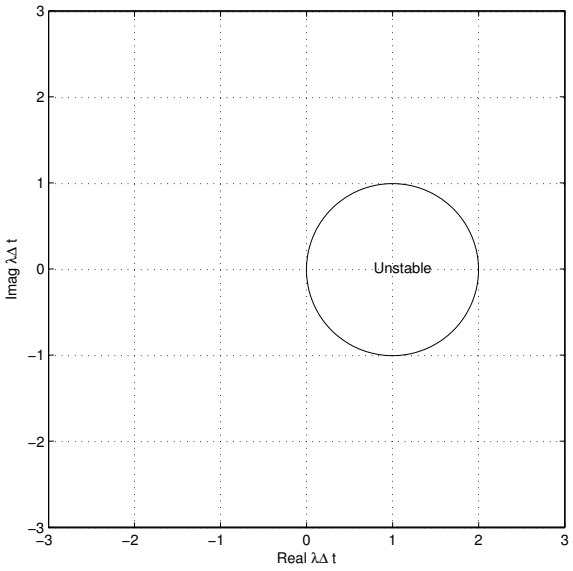


Figure 5.3: Backward Euler stability region

condition number is the ratio of the largest to smallest eigenvalue magnitudes,

$$\text{Spectral Condition Number} = \frac{\max |\lambda_j|}{\min |\lambda_j|}$$

When the spectral condition number is greater than around 1000, problems are starting to become stiff and implicit methods are likely to be more efficient than explicit methods.

Example 5.1 (Stiffness from PDE discretizations) *One of the more common ways for a stiff system of ODE's to arise is in the discretization of time-dependent partial differential equations (PDE's). For example, consider a one-dimensional heat diffusion problem that is modeled by the following PDE for the temperature, T:*

$$T_t = \frac{k}{\rho c_p} T_{xx}.$$

where ρ, c_p, and k are the density, specific heat, and thermal conductivity of the material, respectively. Suppose the physical domain for of length L from $x = 0$ to $x = L$. A finite difference approximation in x might divide the physical domain into a set of equally spaced nodes with distance $h = L/(N-1)$ where N is the total number of nodes including the endpoints. So, node i would be located at $x_i = ih$. Then, at each node, T_{xx} is approximated using a finite difference derivative. For example, at node i we might use the following approximation,

$$T_{xx}|_i = \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}.$$

Note: we will discuss finite difference discretizations of PDE's in detail in Lecture 8. Using this in the heat diffusion equation, we can find the time rate of change at node i as,

$$T_t|_i = \frac{k}{\rho c_p} \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}.$$

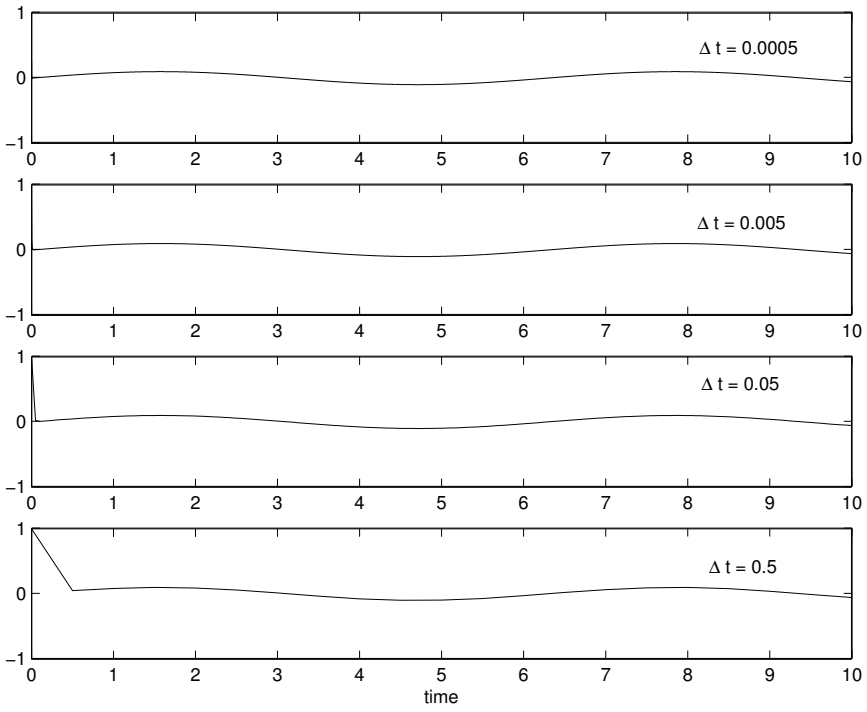


Figure 5.4: Backward Euler solution for $u_t + 1000u = 100 \sin t$ with $u(0) = 1$ at $\Delta t = 0.0005$, 0.005, 0.05, and 0.5.

Thus, the T_t at node i depends on the values of T at nodes $i-1$, i, and $i+1$ in a linear manner. Since each node in the interior of the domain will satisfy the same equation, we can put the finite difference discretization of the heat diffusion problem into our standard system of ODE's form,

$$u_t = Au + b, \qquad u = [T_2, T_3, T_4, \ldots, T_N]^T,$$

where A will be a tri-diagonal matrix (i.e. only the main diagonal and the two neighboring diagonals will be non-zero) since the finite difference approximation only depends on the neighboring nodes. The vector b will depend on the specific boundary conditions.

The question is how do the eigenvalues of A behave, in particular, as the node spacing h is decreased. To look at this, we arbitrarily choose,

$$\frac{k}{\rho c_p} = 1 \qquad L = 1$$

since the magnitudes of these parameters will scale the magnitude of the eigenvalues of A by the same value but not alter the ratio of eigenvalues (the ratio is only altered by the choice of h/L). Figure 5.7 shows the locations of the eigenvalues for $h/L = 0.1$ and $h/L = 0.05$. The eigenvalues are negative real numbers. The smallest magnitude eigenvalues appear to be nearly unchanged by the different values of h. However, the largest magnitude eigenvalues appear to have increased by a factor of 4 from approximately -400 to -1600 when h decreased by a factor of 2. This suggests that the ratio of largest-to-smallest magnitude eigenvalues (i.e. the spectral condition number) is $O(1/h^2)$. Table 5.1 confirms this depends for a range of

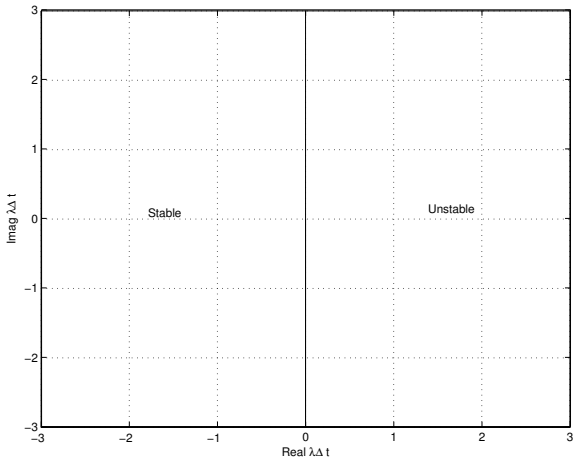


Figure 5.5: Trapezoidal intgeration stability region

h/L	$\min \lvert\lambda\rvert$	$\max \lvert\lambda\rvert$	$\max \lvert\lambda\rvert / \min \lvert\lambda\rvert$
0.001	9.87	3999990	405284
0.01	9.86	39990	4052
0.1	9.79	390	40

Table 5.1: Minimum and maximum magnitude eigenvalues for one-dimensional diffusion

h/L values and also confirms that the smallest eigenvalue changes very little as h decreases.

In-class Discussion 5.1 (How does Δt vary with h for forward Euler?)

5.2 Implicit Methods for Linear Systems of ODE's

While implicit methods can allow significantly larger timesteps, they do involve more work than explicit methods. Consider the forward method applied to $u_t = Au$ where A is a $d \times d$ matrix.

$$v^{n+1} = v^n + \Delta t A v^n.$$

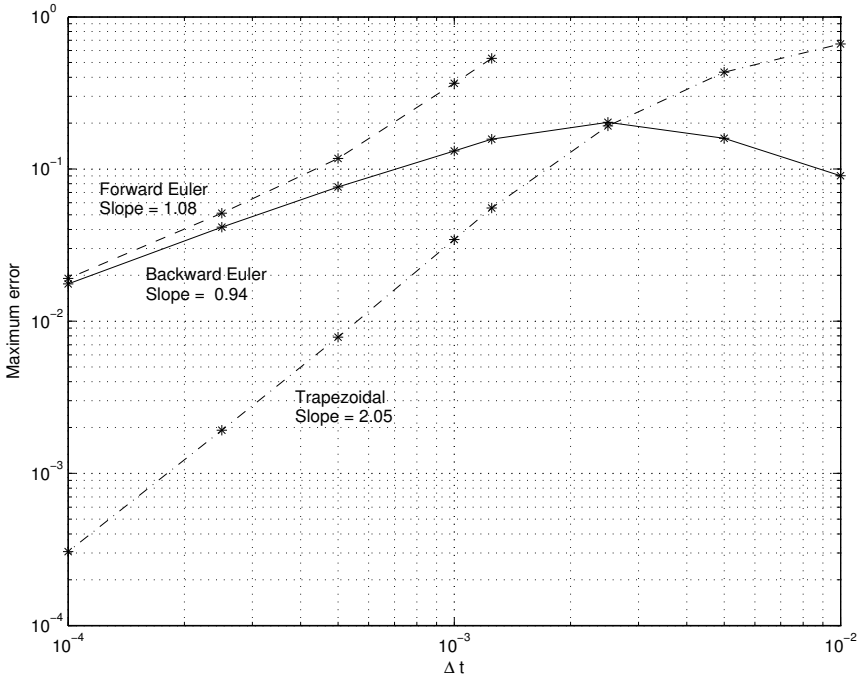


Figure 5.6: Comparison of error for forward Euler, backward Euler, and trapezoidal integration versus Δt for $u_t + 1000u = 100 \sin t$ with $u(0) = 1$.

In this explicit algorithm, the largest computational cost is the matrix vector multiply, Av^n which is an $O(d^2)$ operation. Now, for backward Euler,

$$v^{n+1} = v^n + \Delta t A v^{n+1}.$$

Re-arranging to solve for v^{n+1} gives:

$$\begin{aligned} v^{n+1} &= v^n + \Delta t A v^{n+1}, \\ v^{n+1} - \Delta t A v^{n+1} &= v^n, \\ (I - \Delta t A)\, v^{n+1} &= v^n, \end{aligned}$$

Thus, to find v^{n+1} requires the solution of a $d \times d$ system of equations which is an $O(d^3)$ cost. As a result, for large systems, the cost of the $O(d^3)$ linear solution may begin to outweigh the benefits of the larger timesteps that are possible when using implicit methods.

5.3 Implicit Methods for Nonlinear Problems

When the ODE's are nonlinear, implicit methods require the solution of a nonlinear system of algebraic equations at each iteration. To see this, consider the use of the trapezoidal method for a nonlinear problem,

$$v^{n+1} = v^n + \frac{1}{2}\Delta t \left[f(v^{n+1}, t^{n+1}) + f(v^n, t^n) \right].$$

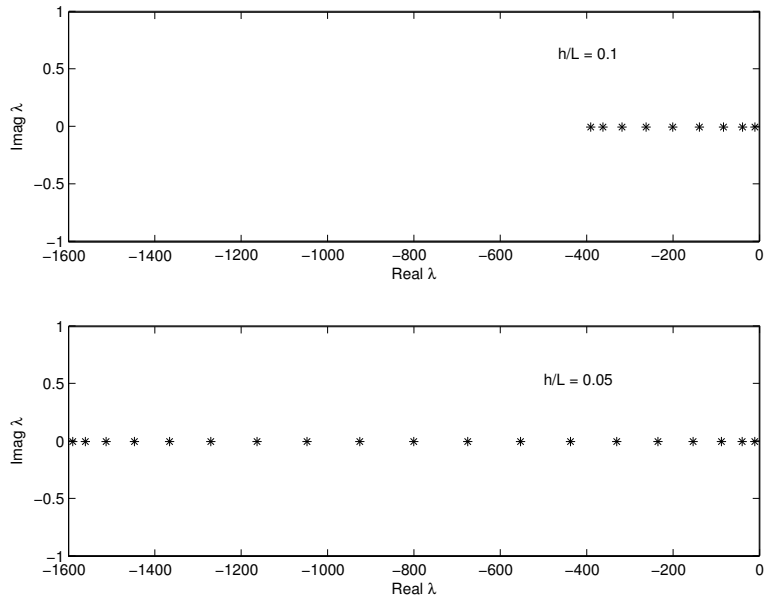


Figure 5.7: Eigenvalues for discretization of one-dimenionsal diffusion equation for $h/L = 0.1$ and $h/L = 0.05$.

We can define the following residual vector for the trapezoidal method,

$$R(w) \equiv w - v^n - \frac{1}{2}\Delta t \left[f(w, t^{n+1}) + f(v^n, t^n) \right].$$

Thus, v^{n+1} for the trapezoidal method is given by the solution of,

$$R(v^{n+1}) = 0,$$

which is a nonlinear algebraic system of equations for v^{n+1}.

One of the standard methods for solving a nonlinear system of algebraic equations is the Newton-Raphson method. It begins with an initial guess for v^{n+1} and solves a linearized version of $R = 0$ to find a correction to the initial guess for v^{n+1}. So, define the current guess for v^{n+1} as w^m where m indicates the sub-iteration in the Newton-Raphson method. Note, common useage is to call the iterations in the Newton-Raphson solution for v^{n+1} sub-iterations since these are iterations which occur within every time iteration from n to $n+1$. To find the correction, Δw, where

$$w^{m+1} = w^m + \Delta w,$$

we linearize and solve the nonlinear residual equation,

$$\begin{aligned} R(w^{m+1}) &= 0, \\ R(w^m + \Delta w) &= 0, \\ R(w^m) + \left.\frac{\partial R}{\partial w}\right|_{w^m} \Delta w &= 0, \\ \left.\frac{\partial R}{\partial w}\right|_{w^m} \Delta w &= -R(w^m). \end{aligned} \tag{5.6}$$

p	α_1	α_2	α_3	α_4	β_0
1	-1				1
2	$-\frac{4}{3}$	$\frac{1}{3}$			$\frac{2}{3}$
3	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$		$\frac{6}{11}$
4	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$	$\frac{12}{25}$

Table 5.2: Coefficients for backward differentiation methods

This last line is a linear system of equations for the correction since $\partial R/\partial w$ is a $d \times d$ matrix when the original ODE's are a system of d equations. For example, for the trapezoidal method,

$$\left.\frac{\partial R}{\partial w}\right|_{w^m} = I - \frac{1}{2}\Delta t \left.\frac{\partial f}{\partial w}\right|_{w^m}.$$

Usually, the initial guess for v^{n+1} is the previous iteration, i.e. $w^0 = v^n$. So, the entire iteration from n to $n+1$ has the following form,

1. Set initial guess: $w^0 = v^n$ and $m = 0$.
2. Calculate residual $R(w^m)$ and linearization $\partial R/\partial w|_{w^m}$.
3. Solve Equation 5.6 for Δw.
4. Update $w^{m+1} = w^m + \Delta w$.
5. Check if $R(w^{m+1})$ is small. If not, perform another sub-iteration.

5.4 Backwards Differentiation Methods

Backwards differentiation methods are one of the best multi-step methods for stiff problems. The backwards differentiation formulae are of the form,

$$v^{n+1} + \sum_{i=1}^{s} v^{n+1-i} = \Delta t \beta_0 f^{n+1}. \tag{5.7}$$

The coefficients for the first through fourth order methods are given in Table 5.2. The stability boundary for these methods are shown in Figure 5.8. As can be seen, all of these methods are stable everywhere on the negative real axis, and are mostly stable in the left-half plane in general. Thus, backwards differentiation work well for stiff problems in which stong damping is present.

Example 5.2 (Matlab's ODE Integrators) *Matlab has a a set of tools for integration of ODE's. We will briefly look at two of them:* ***ode45*** *and* ***ode15s****.* ***ode45*** *is designed to solve problems that are not stiff while* ***ode15s*** *is intended for stiff problems.* ***ode45*** *is based on a four and five-stage Runge-Kutta integration (discussed in Lecture 6), while* ***ode15s*** *is based*

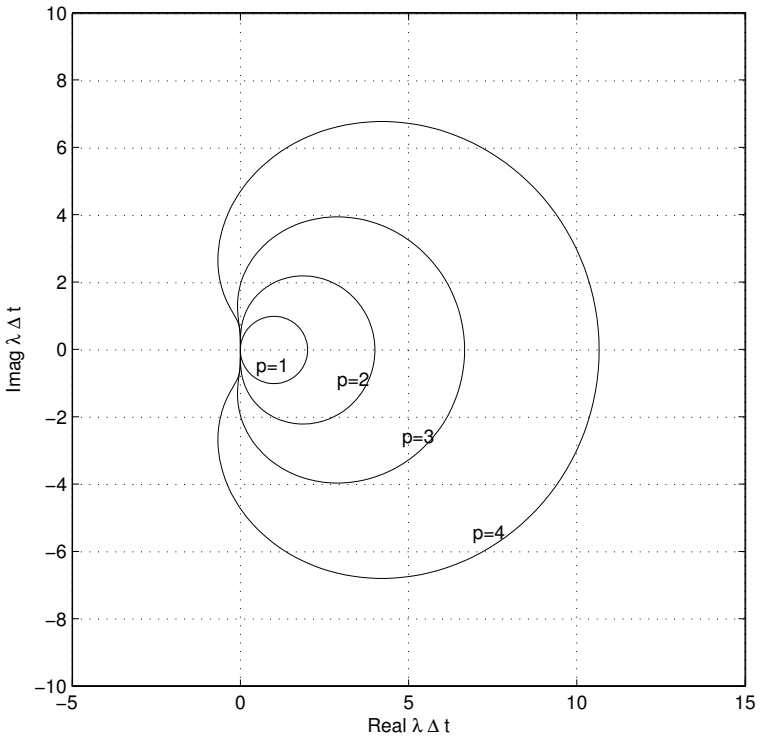


Figure 5.8: Backwards differentiation stability regions for $p = 1$ through $p = 4$ method. Note: interior of curves is unstable region.

on a range of highly stable implicit integration formulas (one option when using ***ode15s*** *is to use the backwards differentiation formulas). As a short illustration on how these Matlab ODE integrators are implemented, the following script solves the one-dimensional diffusion problem from Example 5.1 using either* ***ode45*** *or* ***ode15s****. The specific problem we consider here is a bar which is initially at a temperature* $T_{init} = 400K$ *and at* $t = 0$*, the temperature at the left and right ends is suddenly raised to* $800K$ *and* $1000K$*, respectively.*

```
% Matlab script: dif1d_main.m
%
% This code solve the one-dimensional heat diffusion equation
% for the problem of a bar which is initially at T=Tinit and
% suddenly the temperatures at the left and right change to
% Tleft and Tright.
%
% Upon discretization in space by a finite difference method,
% the result is a system of ODE's of the form,
%
% u_t = Au + b
%
% The code calculates A and b.  Then, uses one of Matlab's
% ODE integrators, either ode45 (which is based on a Runge-Kutta
```

```
% method and is not designed for stiff problems) or ode15s (which
% is based on an implicit method and is designed for stiff problems).
%

clear all; close all;

sflag = input('Use stiff integrator? (1=yes, [default=no]): ');

% Set non-dimensional thermal coefficient
k   = 1.0; % this is really k/(rho*cp)

% Set length of bar
L = 1.0; % non-dimensional

% Set initial temperature
Tinit = 400;

% Set left and right temperatures for t>0
Tleft = 800;
Tright = 1000;

% Set up grid size
Nx = input(['Enter number of divisions in x-direction: [default=' ...
            '51]']);
if (isempty(Nx)),
  Nx = 51;
end

h  = L/Nx;
x  = linspace(0,L,Nx+1);

% Calculate number of iterations (Nmax) needed to iterate to t=Tmax
Tmax = 0.5;

% Initialize a sparse matrix to hold stiffness & identity matrix
A = spalloc(Nx-1,Nx-1,3*(Nx-1));
I = speye(Nx-1);

% Calculate stiffness matrix

for ii = 1:Nx-1,

  if (ii > 1),
    A(ii,ii-1) = k/h^2;
```

```
  end

  if (ii < Nx-1),
    A(ii,ii+1) = k/h^2;
  end

  A(ii,ii) = -2*k/h^2;

end

% Set forcing vector
b = zeros(Nx-1,1);
b(1)    = k*Tleft/h^2;
b(Nx-1) = k*Tright/h^2;

% Set initial vector
v0 = Tinit*ones(1,Nx-1);

if (sflag == 1),

  % Call ODE15s
  options = odeset('Jacobian',A);
  [t,v] = ode15s(@dif1d_fun,[0 Tmax],v0,options,A,b);

else

  % Call ODE45
  [t,v] = ode45(@dif1d_fun,[0 Tmax],v0,[],A,b);

end

% Get midpoint value of T and plot vs. time
Tmid = v(:,floor(Nx/2));
plot(t,Tmid);
xlabel('t');
ylabel('T at midpoint');
```

As can be seen, this script pre-computes the linear system A and the column vector b since the forcing function for the one-dimensional diffusion problem can be written as the linear function, $f = Av + b$. Then, when calling either ODE integrator, the function which returns f is the first argument in the call and is named, ***dif1d_fun****. This function is given below:*

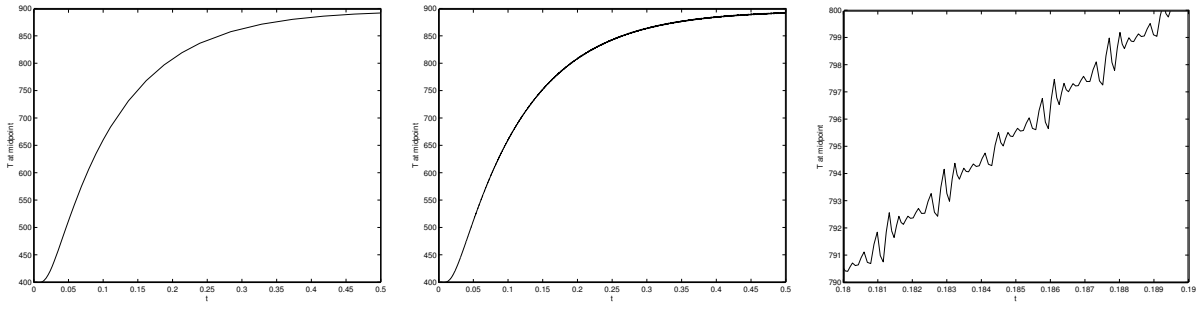
```
% Matlab function: dif1d_fun.m
%
```

```
% This routine returns the forcing term for
% a one-dimensional heat diffusion problem
% that has been discretized by finite differences.
% Note that the matrix A and the vector b are pre-computed
% in the main driver routine, dif1d_main.m, and passed
% to this function.  Then, this function simply returns
% f(v) = A*v + b.  So, in reality, this function is
% not specific to 1-d diffusion.

function [f] = dif1d_fun(t, v, A, b)

f = A*v + b;
```

As can be seen from ***dif1d_fun****, A and b have been passed into the function and thus the calculation of f simply requires the multiplication of v by A and the addition of b.*

The major difference between the implementation of the ODE integrators in Matlab and our discussions is that Matlab's implementations are adaptive. Specifically, Matlab's integrators estimate the error at each iteration and then adjust the timestep to either improve the accuracy (i.e. by decreasing the timestep) or efficiency (i.e. by increasing the timestep).

The results for the stiff integrator, ***ode15s*** *are shown in Figure 5.9(a). These results*



(a) **ode15s** (b) **ode45** (c) **ode45** zoom

Figure 5.9: Temperature evolution at the middle of bar with suddenly raised end temperatures using Matlab's **ode15s** and **ode45** integrators.

look as expected (note: in integrating from $t = 0$ *to* $t = 0.5$*, a total of 64 timesteps were taken).*

The results for the non-stiff integrator are shown in Figure 5.9(b) and in a zoomed view in Figure 5.9(c). The presence of small scale oscillations can be clearly observed in the ***ode45*** *results. These oscillations are a result of the large negative eigenvalues which require small* Δt *to maintain stability. Since the* ***ode45*** *method is adaptive, the timestep automatically decreases to maintain stability, but the oscillatory results clearly show that the stability is barely achieved. Also, as a measure of the relative inefficiency of the* ***ode45*** *integrator for this stiff problem, note that 6273 timesteps were required to integrate from* $t = 0$ *to* $t = 0.5$*.*

One final concern regarding the efficiency of the stiff integrator ***ode15s****. In order for this method to work in an efficient manner for large systems of equations such as in this example, it is very important that the Jacobian matrix,* $\partial f/\partial u$ *be provided to Matlab. If this is not done, then* ***ode15s*** *will construct an approximation to this derivative matrix using finite differences and for large systems, this will become a significant cost. In the script* ***dif1d_main****, the Jacobian is communicated to the* ***ode15s*** *integrator using the* ***odeset*** *routine. Note:* ***ode45*** *is an explicit method and does not need the Jacobian so it is not provided in that case.*

Lecture 6

Runge-Kutta Methods

In the previous lectures, we have concentrated on multi-step methods. However, another powerful set of methods are known as multi-stage methods. Perhaps the best known of multi-stage methods are the Runge-Kutta methods. In this lecture, we give some of the most popular Runge-Kutta methods and briefly discuss their properties.

6.1 Two-stage Runge-Kutta Methods

A popular two-stage Runge-Kutta method is known as the modifed Euler method:

$$\begin{aligned} a &= \Delta t f(v^n, t^n) \\ b &= \Delta t f(v^n + a/2, t^n + \Delta t/2) \\ v^{n+1} &= v^n + b \end{aligned}$$

Another popular two-stage Runge-Kutta method is known as the Heun method:

$$\begin{aligned} a &= \Delta t f(v^n, t^n) \\ b &= \Delta t f(v^n + a, t^n + \Delta t) \\ v^{n+1} &= v^n + \frac{1}{2}(a + b) \end{aligned}$$

As can been seen with either of these methods, f is evaluated twice in finding the new value of v^{n+1}: once to determine a and once to determine b. Both of these methods are second-order accurate, $p = 2$.

6.2 Four-stage Runge-Kutta Method

The most popular form of a four-stage Runge-Kutta method is:

$$\begin{aligned} a &= \Delta t f(v^n, t^n) \\ b &= \Delta t f(v^n + a/2, t^n + \Delta t/2) \\ c &= \Delta t f(v^n + b/2, t^n + \Delta t/2) \end{aligned}$$

$$\begin{aligned} d &= \Delta t f(v^n + c, t^n + \Delta t) \\ v^{n+1} &= v^n + \frac{1}{6}(a + 2b + 2c + d) \end{aligned}$$

Note that this method requires four evaluations of f per iteration. This method is fourth-order accurate, $p = 4$.

6.3 Stability Regions

The eigenvalue stability regions for Runge-Kutta methods can be found using essentially the same approach as for multi-step methods. Specifically, we consider a linear problem in which $f = \lambda u$ where λ is a constant. Then, we determine the amplification factor $g = g(\lambda \Delta t)$. For example, let's look at the modified Euler method,

$$\begin{aligned} a &= \Delta t \lambda v^n \\ b &= \Delta t \lambda \left(v^n + \Delta t \lambda v^n / 2\right) \\ v^{n+1} &= v^n + \Delta t \lambda \left(v^n + \Delta t \lambda v^n / 2\right) \\ v^{n+1} &= \left[1 + \Delta t \lambda + \frac{1}{2}(\Delta t \lambda)^2\right] v^n \\ \Rightarrow g &= 1 + \lambda \Delta t + \frac{1}{2}(\lambda \Delta t)^2 \end{aligned}$$

A similar derivation for the four-stage scheme shows that,

$$g = 1 + \lambda \Delta t + \frac{1}{2}(\lambda \Delta t)^2 + \frac{1}{6}(\lambda \Delta t)^3 + \frac{1}{24}(\lambda \Delta t)^4.$$

When analyzing multi-step methods, the next step would be to determine the locations in the $\lambda \Delta t$-plane of the stability boundary (i.e. where $|g| = 1$). This however is not easy for Runge-Kutta methods and would require the solution of a higher-order polynomial for the roots. Instead, the most common approach is to simply rely on a contour plotter in which the $\lambda \Delta t$-plane is discretized into a finite set of points and $|g|$ is evaluated at these points. Then, the $|g| = 1$ contour can be plotted. The following is the Matlab code which produces the stability region for the second-order Runge-Kutta methods (note: $g(\lambda \Delta t)$ is the same for both second-order methods):

```
% Specify x range and number of points
x0 = -3;
x1 =  3;
Nx = 301;

% Specify y range and number of points
y0 = -3;
y1 =  3;
Ny = 301;
```

```
% Construct mesh
xv    = linspace(x0,x1,Nx);
yv    = linspace(y0,y1,Ny);
[x,y] = meshgrid(xv,yv);

% Calculate z
z = x + i*y;

% 2nd order Runge-Kutta growth factor
g = 1 + z + 0.5*z.^2;

% Calculate magnitude of g
gmag = abs(g);

% Plot contours of gmag
contour(x,y,gmag,[1 1],'k-');
axis([x0,x1,y0,y1]);
axis('square');
xlabel('Real \lambda\Delta t');
ylabel('Imag \lambda\Delta t');
grid on;
```

The plots of the stability regions for the second and fourth-order Runge-Kutta algorithms is shown in Figure 6.1. These stability regions are larger than those of multi-step methods. In particular, the stability regions of the multi-stage schemes grow with increasing accuracy while the stability regions of multi-step methods decrease with increasing accuracy (see Appendix A).

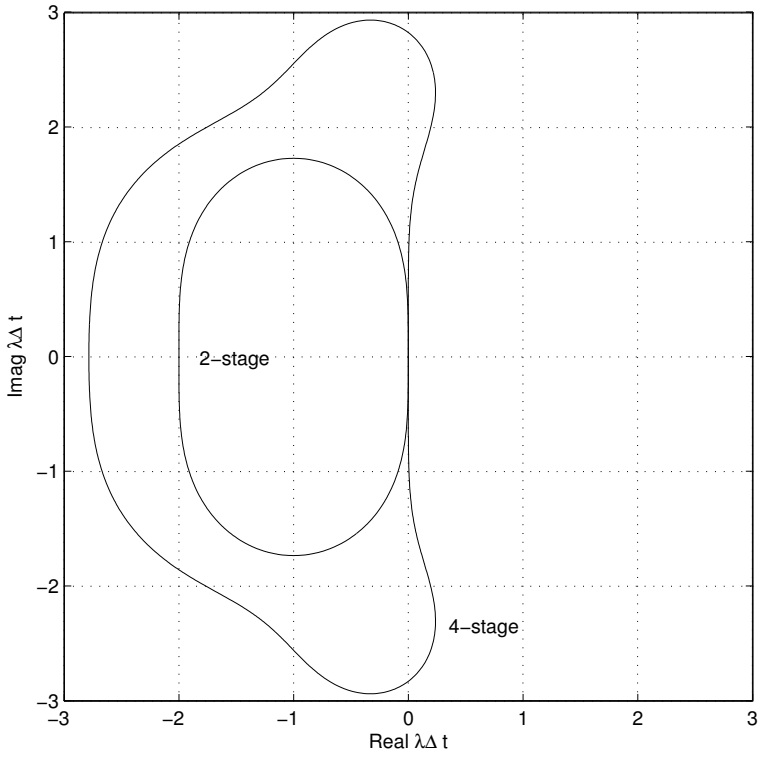


Figure 6.1: Stability boundaries for second-order and fourth-order Runge-Kutta algorithms (stable within the boundaries).

Lecture 7

Conservation Laws and the Finite Volume Method

7.1 Conservation Laws

In most engineering applications, the physical system is governed by a set of conservation laws. For example, in gas dynamics, the conservation of mass, momentum, and energy are applied to the gas. These conservation laws are often written in integral form for a fixed physical domain. Suppose we have a two-dimensional physical domain, Ω, with the boundary of the domain, $\delta\Omega$. Then, the canonical conservation equation assuming that the physical domain is fixed is of the form,

$$\frac{d}{dt}\int_{\Omega} U\,dA + \int_{\delta\Omega}\left[F(U)\vec{i} + G(U)\vec{j}\right]\cdot\vec{n}\,ds = \int_{\Omega} S(U,t)\,dA, \tag{7.1}$$

where U is the conserved state, F and G are the flux of the conserved state in the x and y directions, $\vec{n}$ is the outward pointing unit normal on the boundary of the domain, and S is a source term.

This conservation law can be written as a partial differential equation by applying the divergence theorem which states that,

$$\int_{\delta\Omega}\left[F\vec{i} + G\vec{j}\right]\cdot\vec{n}\,ds = \int_{\Omega}\left(\frac{\partial F}{\partial x} + \frac{\partial G}{\partial y}\right)\,dA. \tag{7.2}$$

Thus, Equation 7.1 becomes,

$$\frac{d}{dt}\int_{\Omega} U\,dA + \int_{\delta\Omega}\left(F\vec{i} + G\vec{j}\right)\cdot\vec{n}\,ds = \int_{\Omega} S\,dA,$$

$$\int_{\Omega}\frac{\partial U}{\partial t}\,dA + \int_{\Omega}\left(\frac{\partial F}{\partial x} + \frac{\partial G}{\partial y}\right)\,dA = \int_{\Omega} S\,dA,$$

$$\int_{\Omega}\left(\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} - S\right)\,dA = 0.$$

Thus, since this last equation would remain valid for any arbitrary domain, Ω, this means that the integrand must be zero everywhere, or, equivalently,

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S. \tag{7.3}$$

Equation 7.3 is the conservation law written as a partial differential equation.

Example 7.1 (Conservation of Mass for a Compressible Fluid) *An example of a conservation law is the conservation of mass for a compressible fluid. Let the fluid density be $\rho(x, y, t)$ and the fluid x and y velocity components be $u(x, y, t)$ and $v(x, y, t)$, respectively. Then, the conservation of mass for the fluid is,*

$$\frac{d}{dt}\int_\Omega \rho \, dA + \int_{\delta\Omega} \left[\rho u \vec{i} + \rho v \vec{j}\right] \cdot \vec{n} \, ds = 0.$$

In terms of the canonical form,

$$\begin{aligned} U &= \rho, \\ F &= \rho u, \\ G &= \rho v, \\ S &= 0. \end{aligned}$$

Example 7.2 (The Euler Equations for a Compressible Fluid) *Often, multiple conservation laws are of interest. In this case, U is a vector of conserved states. Furthermore, F, G, and S are vectors. As an example, the Euler equations for a compressible fluid in two-dimensions are the combination of conservation of mass, x-momentum, y-momentum, and energy. In this case,*

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \qquad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix} \qquad G = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix} \qquad S = 0.$$

The first row of these vectors represents the conservation of mass (see Example 7.1 for more). The second and third row represent conservation of x and y momentum, respectively. And, the fourth row represents conservation of energy. Note that in addition to the fluid density (ρ) and the x and y velocity components (u and v), the other quantities in these equations are:

$$\begin{aligned} p &= \textit{static pressure}, \\ E &= \textit{total energy}, \\ H &= \textit{total enthalpy}. \end{aligned}$$

The total energy and total enthalpy are related by,

$$H = E + \frac{p}{\rho}.$$

This system of equations is not quite complete, however, since the number of equations does not equal the number of dependent variables in the equations. In particular, note that we have given five equations thus far (the four conservation equations and the relationship of H to E) while the number of dependent variables is six (i.e. ρ, u, v, p, E, and H). The final equation is an equation of state. Often, we assume an ideal gas and use the ideal gas law. In terms of the dependent variables we have introduced, the ideal gas law can be written as,

$$p = (\gamma - 1)\left[\rho E - \frac{1}{2}\rho(u^2 + v^2)\right], \tag{7.4}$$

where γ is the ratio of specific heats (for air, $\gamma \approx 1.4$). Many of you may be more familiar with the ideal gas law in the form, $p = \rho RT$ where R is the gas constant and T is the temperature. Equation 7.4 is (in fact) equivalent to $p = \rho RT$ but Equation 7.4 is used since $p = \rho RT$ introduces a new dependent variable (i.e. the temperature) and would therefore require yet another state equation to complete the system.

7.2 Convection

In many applications, especially those in fluid dynamics, convection is the dominant physical transport mechanism over much of the domain of interest. While diffusion is always present, often its affects are smaller except in limited regions (often near solid boundaries where boundary layers form due to the combined effects of diffusion and convection).

In this section, we will derive the convection equation using the conservation law as given in Equation 7.1. Specifically, let U be the 'conserved' scalar quantity and let the fluxes be given by,

$$F = uU, \qquad G = vU, \qquad S = 0, \tag{7.5}$$

where $u(x, y, t)$ and $v(x, y, t)$ are known velocity components. Note, a non-zero source term could be included, but for simplicity is assumed to be zero. As a PDE, this scalar conservation law is,

$$\frac{\partial U}{\partial t} + \frac{\partial}{\partial x}(uU) + \frac{\partial}{\partial y}(vU) = 0.$$

This equation can be manipulated by expanding the x and y derivatives into,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} = -U\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right).$$

Often a reasonable assumption is that the velocity field is divergence free such that $\partial u/\partial x + \partial v/\partial y = 0$. In this case, we arrive at what is commonly referred to as the convection equation,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} = 0. \tag{7.6}$$

Physically, this equation states that following along the streamwise direction (i.e. convecting with the velocity), the quantity U does not change.

In developing numerical methods for convection-dominated problems, we will often rely on insight that can be gained from the convection equation for the specific case when the

velocity field is a constant value, i.e. $u(x, y, t) = u$ and $v(x, y, t) = v$. In this situation, the solution to Equation 7.6 has the following form,

$$U(x, y, t) = U_0(\xi, \eta) \quad \text{where} \quad \xi = x - ut, \quad \eta = y - vt, \tag{7.7}$$

where $U_0(x, y)$ is the distribution of U at time $t = 0$. By substitution, we can confirm that this indeed is a solution of Equation 7.6,

$$\begin{aligned} \frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} &= \frac{\partial}{\partial t}U_0(\xi, \eta) + u\frac{\partial}{\partial x}U_0(\xi, \eta) + v\frac{\partial}{\partial y}U_0(\xi, \eta) \\ &= \frac{\partial U_0}{\partial \xi}\frac{\partial \xi}{\partial t} + \frac{\partial U_0}{\partial \eta}\frac{\partial \eta}{\partial t} + u\left(\frac{\partial U_0}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial U_0}{\partial \eta}\frac{\partial \eta}{\partial x}\right) + v\left(\frac{\partial U_0}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial U_0}{\partial \eta}\frac{\partial \eta}{\partial y}\right) \end{aligned}$$

The partial derivatives of ξ and η are,

$$\frac{\partial \xi}{\partial t} = -u, \quad \frac{\partial \xi}{\partial x} = 1, \quad \frac{\partial \xi}{\partial y} = 0,$$

$$\frac{\partial \eta}{\partial t} = -v, \quad \frac{\partial \eta}{\partial x} = 0, \quad \frac{\partial \eta}{\partial y} = 1.$$

Upon substitution of these partial derivatives,

$$\begin{aligned} \frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} &= -u\frac{\partial U_0}{\partial \xi} + -v\frac{\partial U_0}{\partial \eta} + u\frac{\partial U_0}{\partial \xi} + v\frac{\partial U_0}{\partial \eta}, \\ &= 0. \end{aligned}$$

Thus, $U(x, y, t) = U_0(x - ut, y - vt)$ is a solution to the convection equation.

Example 7.3 (Two-dimensional Convection) *To illustrate the behavior of the 2-D convection equation, we consider a specific problem. Suppose that the velocity is at 45 degrees with respect to the x-axis such that,*

$$u = 1, \qquad v = 1.$$

Also, suppose that the initial distribution of U is,

$$U_0(x, y) = e^{-x^2 - 20y^2}.$$

In Figure 7.1, the initial distribution of U is shown (i.e. at $t = 0$) as well as the distribution of U at $t = 1$. Clearly, the contours have moved along the 45 degree line (shown as a dashed line in the figure). Initially, the contours are centered at the origin and then at $t = 1$ they are centered at $x = y = 1$. A plot of U along the dashed line is shown in Figure 7.2. Since this line is tangent to the convection direction, the distribution of U convects without changing shape as t increases.

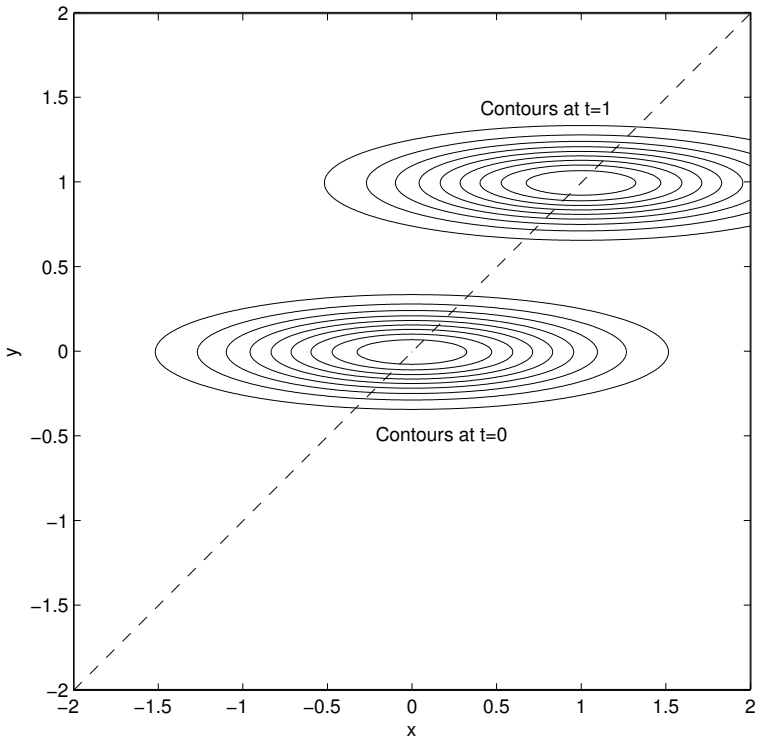


Figure 7.1: Distribution of U at $t = 0$ and at $t = 1$ for a convection problem with velocities $u = v = 1$.

7.3 Finite Volume Method for Convection

In this section, we will discuss the finite volume method. Our initial focus will be on convection and we will assume that the velocity field is divergence free. Thus, the integral conservation law (i.e. Equation 7.1) with fluxes given by Equation 7.5 is completely equivalent to the PDE for convection (i.e. Equation 7.6).

7.3.1 One-Dimensional Convection

In one-dimensional problems, the assumption that the velocity is divergence free, i.e. $\partial u/\partial x = 0$, forces the velocity to be constant with respect to x (though u could change with t).

The basis of the finite volume method is the integral convervation law. The essential idea is to divide the domain into many control volumes and approximate the integral conservation law on each of the control volumes. For example, as shown in Figure 7.3, cell i lies between the points at $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. Note that the points do not have to be equally-spaced.

The one-dimensional form of Equation 7.1 is

$$\frac{d}{dt}\int_{x_L}^{x_R} U\,dx + F(U)|_{x_R} - F(U)|_{x_L} = \int_{x_L}^{x_R} S(U,t)\,dx. \tag{7.8}$$

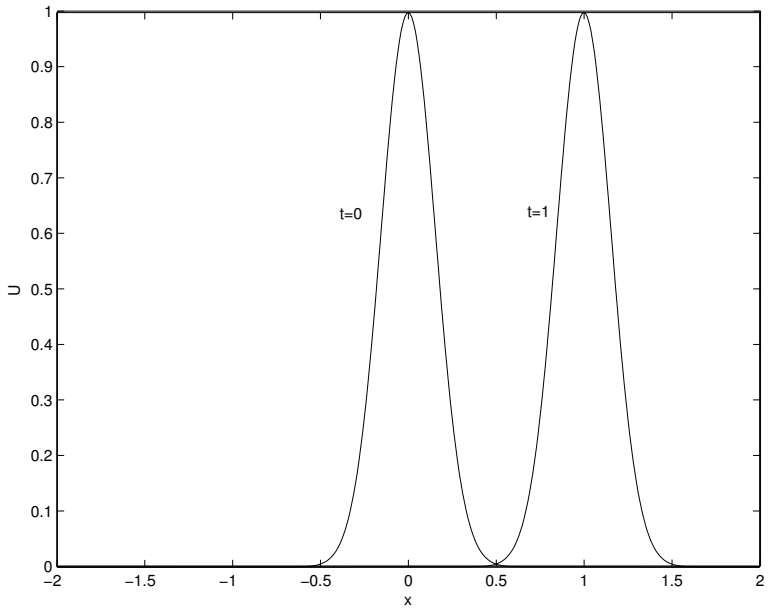


Figure 7.2: Distribution of U along $y = x$ at $t = 0$ and at $t = 1$ for a convection problem with velocities $u = v = 1$.

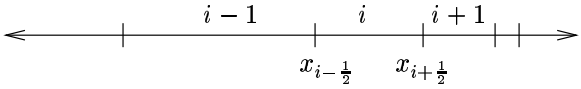


Figure 7.3: Mesh and notation for one-dimensional finite volume method.

Thus, applying this to control volume i (and recalling that $S = 0$ for convection) gives,

$$\frac{d}{dt}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} U\,dx + F(U)|_{x_{i+\frac{1}{2}}} - F(U)|_{x_{i-\frac{1}{2}}} = 0. \tag{7.9}$$

Next, we define the mean value of U in control volume i as,

$$U_i \equiv \frac{1}{\Delta x_i}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} U\,dx, \quad \text{where} \quad \Delta x_i \equiv x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}.$$

Then, Equation 7.9 becomes,

$$\Delta x_i \frac{dU_i}{dt} + F(U)|_{x_{i+\frac{1}{2}}} - F(U)|_{x_{i-\frac{1}{2}}} = 0. \tag{7.10}$$

At this point, no approximations have been made thus Equation 7.10 is exact. Now, we make the first approximation. Specifically, we assume that the solution in each control volume is constant,

$$U(x,t) = U_i(t) \quad \text{for} \quad x_{i-\frac{1}{2}} < x < x_{i+\frac{1}{2}}.$$

Thus, the finite volume approximation will be piecewise constant as shown in Figure 7.4.

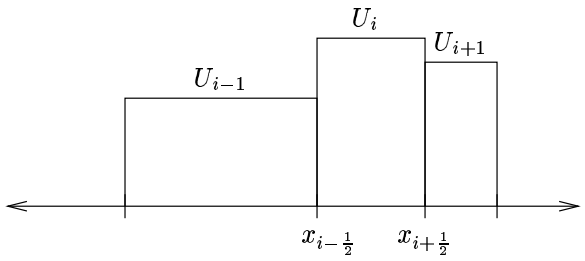


Figure 7.4: Piecewise constant solution for one-dimensional finite volume method.

With this assumed form of the solution, the next issue is to determine the flux at $i \pm \frac{1}{2}$ at a time t. This can be done with the knowledge that the solution convects with the velocity $u(t)$. Thus, for the initial instant after t (which we denote as $t^+ = t + \epsilon$ where ϵ is an infinitessimal, positive number):

$$U(x_{i+\frac{1}{2}}, t^+) = \begin{cases} U_i(t) & \text{if } u(t) > 0 \\ U_{i+1}(t) & \text{if } u(t) < 0 \end{cases}$$

The flux can be calculated directly from this value of U,

$$F(x_{i+\frac{1}{2}}, t^+) = \begin{cases} u(t)U_i(t) & \text{if } u(t) > 0 \\ u(t)U_{i+1}(t) & \text{if } u(t) < 0 \end{cases}$$

An alternative way to write this flux which is valid regardless of the sign of $u(t)$ is,

$$F(x_{i+\frac{1}{2}}, t^+) = \frac{1}{2}u(t)\left[U_{i+1}(t) + U_i(t)\right] - \frac{1}{2}|u(t)|\left[U_{i+1}(t) - U_i(t)\right]. \tag{7.11}$$

These fluxes, which use the upstream value of U to determine the flux. are known as an 'upwind' flux.

The final step in arriving at a full-discrete approximation for one-dimensional convection is to discretize Equation 7.10 in time. This can be done choosing any of the ODE integration methods we studied previously. For simplicity, we choose the forward Euler method so that the final fully-discrete form of the finite volume method is,

$$\Delta x_i \frac{U_i^{n+1} - U_i^n}{\Delta t} + F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n = 0, \tag{7.12}$$

where we use the notation,

$$F_{i+\frac{1}{2}}^n = \frac{1}{2}u^n\left(U_{i+1}^n + U_i^n\right) - \frac{1}{2}|u^n|\left(U_{i+1}^n - U_i^n\right). \tag{7.13}$$

Example 7.4 (Finite Volume Method applied to 1-D Convection) *The following Matlab script solves the one-dimensional convection equation using the finite volume algorithm given by Equation 7.12 and 7.13. The problem is assumed to be periodic so that whatever leaves the domain at $x = x_R$ re-enters it at $x = x_L$.*

```
% Script: convect1d.m

clear all;

% Set-up grid
xL = -4;
xR =  4;
Nx = 40; % number of control volumes
x = linspace(xL,xR,Nx+1);

% Calculate midpoint values of x in each control volume
xmid = 0.5*(x(1:Nx) + x(2:Nx+1));

% Calculate cell size in control volumes (assumed equal)
dx = x(2) - x(1);

% Set velocity
u = 1;

% Set final time
tfinal = 1;

% Set timestep
CFL = 0.5;
dt = CFL*dx/abs(u);

% Set initial condition to U0 = exp(-x^2)
% Note: technically, we should average the initial
% distribution in each cell but I chose to just set
% the value of U in each control volume to the midpoint
% value of U0.

U = exp(-xmid.^2);
t = 0;

% Loop until t > tfinal
while (t < tfinal),

  Ubc = [U(Nx), U, U(1)]; % This enforces the periodic bc

  % Calculate the flux at each interface
  F =   0.5*    u *( Ubc(2:Nx+2) + Ubc(1:Nx+1)) ...
      - 0.5*abs(u)*( Ubc(2:Nx+2) - Ubc(1:Nx+1));
```

```
  % Calculate residual in each cell
  R = F(2:Nx+1) - F(1:Nx);

  % Forward Euler step
  U = U - (dt/dx)*R;

  % Increment time
  t = t + dt;

  % Plot current solution
  stairs(x,[U, U(Nx)]);
  axis([xL, xR, -0.5, 1.5]);
  grid on;
  drawnow;

end
```

7.3.2 Two-Dimensional Convection

The finite volume discretization can be extended to two-dimensional problems. Suppose the physical domain is divided into a set of triangular control volumes, as shown in Figure 7.5. Application of Equation 7.1 to control volume A gives,

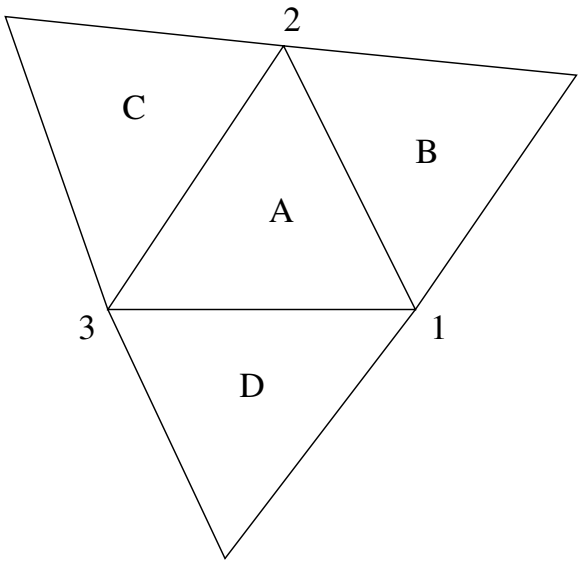


Figure 7.5: Triangular mesh and notation for finite volume method.

$$\frac{d}{dt}\int_{\Omega_A} U\, dA + \int_{\delta\Omega_A} H(U,\vec{n})\, ds = \int_{\Omega_A} S(U,t)\, dA, \tag{7.14}$$

where $H(U, \vec{n})$ is the flux normal to the face,

$$H(U, \vec{n}) \equiv \left[F(U)\vec{i} + G(U)\vec{j}\right] \cdot \vec{n}. \tag{7.15}$$

As in the one-dimensional case, we define the cell average,

$$U_A \equiv \frac{1}{A_A} \int_{\Omega_A} U \, dA,$$

where A_A is the area of control volume A. Thus, Equation 7.14 becomes,

$$A_A \frac{dU_A}{dt} + \int_{\delta\Omega_A} H(U, \vec{n}) \, ds = \int_{\Omega_A} S(U, t) \, dA.$$

In the case of convection, we again assume $S = 0$. Also, we expand the surface integral into the contributions for the three edges,

$$A_A \frac{dU_A}{dt} + \int_1^2 H(U, \vec{n}_{AB}) \, ds + \int_2^3 H(U, \vec{n}_{AC}) \, ds + \int_3^1 H(U, \vec{n}_{AD}) \, ds = 0,$$

where $\vec{n}_{AB}$ is the unit normal pointing from cell A to cell B, and similarly for $\vec{n}_{AC}$ and $\vec{n}_{AD}$.

As in one-dimensional case, we assume that the solution everywhere in the control volume is equal to the cell average value. Finally, the flux at each interface is determined by the 'upwind' value using the velocity component normal to the face. For example, at the interface between cell A and B,

$$H(U, \vec{n}_{AB}) \approx \hat{H}(U_A, U_B, \vec{n}_{AB}) \equiv \frac{1}{2} \vec{u}_{AB} \cdot \vec{n}_{AB} \left(U_B + U_A\right) - \frac{1}{2} |\vec{u}_{AB} \cdot \vec{n}_{AB}| \left(U_B - U_A\right), \tag{7.16}$$

where $\vec{u}_{AB}$ is the velocity between the control volumes. Thus, when $\vec{u}_{AB} \cdot \vec{n}_{AB} > 0$, the flux is determined by the state from cell A, i.e. U_A. Likewise, when $\vec{u}_{AB} \cdot \vec{n}_{AB} < 0$, the flux is determined by the state from cell B, i.e. U_B. The velocity, $\vec{u}_{AB}$ is usually approximated as the velocity at the midpoint of the edge (note: $\vec{u}$ can be a function of $\vec{x}$ in two-dimensions even though the velocity is assumed to be divergence free, i.e. $\partial u/\partial x + \partial v/\partial y = 0$). We use the notation $\hat{H}$ to indicate that the flux is an approximation to the true flux when $\vec{u}$ is not constant. Thus, the finite volume algorith prior to time discretization would be given by,

$$A_A \frac{dU_A}{dt} + \hat{H}(U_A, U_B, \vec{n}_{AB}) \Delta s_{AB} + \hat{H}(U_A, U_C, \vec{n}_{AC}) \Delta s_{AC} + \hat{H}(U_A, U_D, \vec{n}_{AD}) \Delta s_{AD} = 0.$$

The final step is to integrate in time. As in the one-dimensional case, we might use a forward Euler algorithm which would result in the final fully discrete finite volume method,

$$A_A \frac{U_A^{n+1} - U_A^n}{\Delta t} + \hat{H}(U_A^n, U_B^n, \vec{n}_{AB}) \Delta s_{AB} + \hat{H}(U_A^n, U_C^n, \vec{n}_{AC}) \Delta s_{AC} + \hat{H}(U_A^n, U_D^n, \vec{n}_{AD}) \Delta s_{AD} = 0. \tag{7.17}$$

Example 7.5 (Finite Volume Method for 2-D Convection on a Rectangular Mesh) *The following Matlab script solves the two-dimensional convection equation using a two-dimensional finite volume algorithm on rectangular cells. The algorithm is the extension of Equation 7.17 from triangular to rectangular cells. The problem is assumed to be periodic and have a constant velocity.*

```
% Script: convect2d.m

close all;
clear all;

% Specify x range and number of points
x0 = -2;
x1 =  2;
Nx = 40;

% Specify y range and number of points
y0 = -2;
y1 =  2;
Ny = 40;

% Construct mesh
x       = linspace(x0,x1,Nx+1);
y       = linspace(y0,y1,Ny+1);
[xg,yg] = ndgrid(x,y);

% Construct mesh needed for plotting
xp = zeros(4,Nx*Ny);
yp = zeros(4,Nx*Ny);
n = 0;
for j = 1:Ny,
  for i = 1:Nx,

    n = n + 1;
    xp(1,n) = x(i);
    yp(1,n) = y(j);

    xp(2,n) = x(i+1);
    yp(2,n) = y(j);

    xp(3,n) = x(i+1);
    yp(3,n) = y(j+1);

    xp(4,n) = x(i);
    yp(4,n) = y(j+1);

  end
end

% Calculate midpoint values in each control volume
```

```
xmid = 0.5*(x(1:Nx) + x(2:Nx+1));
ymid = 0.5*(y(1:Ny) + y(2:Ny+1));

[xmidg,ymidg] = ndgrid(xmid,ymid);

% Calculate cell size in control volumes (assumed equal)
dx = x(2) - x(1);
dy = y(2) - y(1);
A  = dx*dy;

% Set velocity
u = 1;
v = 1;

% Set final time
tfinal = 10;

% Set timestep
CFL = 1.0;
dt = CFL/(abs(u)/dx + abs(v)/dy);

% Set initial condition to U0 = exp(-x^2 - 20*y^2)
% Note: technically, we should average the initial
% distribution in each cell but I chose to just set
% the value of U in each control volume to the midpoint
% value of U0.
U = exp(-xmidg.^2 - 20*ymidg.^2);
t = 0;


% Loop until t > tfinal
while (t < tfinal),

  % The following implement the bc's by creating a larger array
  % for U and putting the appropriate values in the first and last
  % columns or rows to set the correct bc's
  Ubc(2:Nx+1,2:Ny+1) = U; % Copy U into Ubc
  Ubc(   1,2:Ny+1)   = U(Nx, :); % Periodic bc
  Ubc(Nx+2,2:Ny+1)   = U( 1, :); % Periodic bc
  Ubc(2:Nx+1,   1)   = U( :,Ny); % Periodic bc
  Ubc(2:Nx+1,Ny+2)   = U( :, 1); % Periodic bc

  % Calculate the flux at each interface
```

```
  % First the i interfaces
  F =   0.5*    u *( Ubc(2:Nx+2,2:Ny+1) + Ubc(1:Nx+1,2:Ny+1)) ...
      - 0.5*abs(u)*( Ubc(2:Nx+2,2:Ny+1) - Ubc(1:Nx+1,2:Ny+1));

  % Now the j interfaces
  G =   0.5*    v *( Ubc(2:Nx+1,2:Ny+2) + Ubc(2:Nx+1,1:Ny+1)) ...
      - 0.5*abs(v)*( Ubc(2:Nx+1,2:Ny+2) - Ubc(2:Nx+1,1:Ny+1));

  % Add contributions to residuals from fluxes
  R = (F(2:Nx+1,:) - F(1:Nx,:))*dy + (G(:,2:Ny+1) - G(:,1:Ny))*dx;

  % Forward Euler step
  U = U - (dt/A)*R;

  % Increment time
  t = t + dt;

  % Plot current solution
  Up = reshape(U,1,Nx*Ny);
  clf;
  [Hp] = patch(xp,yp,Up);
  set(Hp,'EdgeAlpha',0);
  axis('equal');
  caxis([0,1]);
  colorbar;
  drawnow;

end
```

7.4 Extensions of the Finite Volume Method

7.4.1 Nonlinear Systems

The basic finite volume approach can be extended to nonlinear systems of equations such as the Euler equations (see Example 7.2). The main issue in this extension is how to calculate an upwind flux when there is a system of equations. In one dimension, the basic finite volume discretization remains the same as given by Equation 7.13,

$$\Delta x_i \frac{U_i^{n+1} - U_i^n}{\Delta t} + F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}} = 0.$$

The flux, however, must upwind (to some degree) all of the states in the equation. One relatively simple way in which this can be done is using what is known as the local Lax-

Friedrichs flux. In this case, the flux is given by,

$$F_{i+\frac{1}{2}}(U_i, U_{i+1}) = \frac{1}{2}\left[F(U_{i+1}) + F(U_i)\right] - \frac{1}{2}s_{\max}\left(U_{i+1} - U_i\right), \tag{7.18}$$

where $s_{\max}$ is the maximum speed of propagation of any small disturbance for either state U_i or U_{i+1}.

Example 7.6 (Lax-Friedrichs Flux for 1-D Euler Equations) *For the one-dimensional Euler equations, there are three equations which are approximated, i.e. conservation of mass, conservation of x-momentum, and conservation of energy. A small perturbation analysis can be performed which shows that the three speeds of propagation for this set of equations are* u, $u-a$, *and* $u+a$ *where* u *is the flow velocity and* a *is the speed of sound. Thus, the maximum speed will always be* $|u| + a$ *and the corresponding value of* $s_{\max}$ *for the Lax-Friedrichs flux is,*

$$s_{\max} = \max\left(|u|_i + a_i,\ |u|_{i+1} + a_{i+1}\right).$$

7.4.2 Higher-order Accuracy

The extension to higher-order accuracy will be discussed in class.

Lecture 8

Finite Difference Methods for Convection-Diffusion

In this lecture, we introduce the finite difference method for the solution of PDE's. We will limit our discussion of PDE's to convection-diffusion. Recall from Equation 7.6 that the convection equation is,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} = 0,$$

where U is the scalar quantity which is convected. Adding diffusion to this equation gives the convection-diffusion equation,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + v\frac{\partial U}{\partial y} = \mu\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right), \tag{8.1}$$

where μ is the diffusion coefficient.

8.1 Finite Difference Approximations

The finite difference approximation of a PDE is constructed using a grid over the domain of interest. The finite difference approximations are easiest to derive using a structured, rectangular as shown in Figure 8.1. For a finite difference approximation of a PDE, the solution is sought at the nodes of the mesh. We use the notation that $U_{i,j}$ is the value of U at the (i,j) node.

A common finite difference approximation of $\partial U/\partial x$ at node (i,j) is,

$$\left.\frac{\partial U}{\partial x}\right|_{i,j} \approx \delta_{2x}U_{i,j} \equiv \frac{1}{2\Delta x}\left(U_{i+1,j} - U_{i-1,j}\right). \tag{8.2}$$

The finite difference operator δ_{2x} is called a central difference operator. Finite difference approximations can also be one-sided. For example, a backward difference approximation is,

$$\left.\frac{\partial U}{\partial x}\right|_{i,j} \approx \delta_x^- U_{i,j} \equiv \frac{1}{\Delta x}\left(U_{i,j} - U_{i-1,j}\right), \tag{8.3}$$

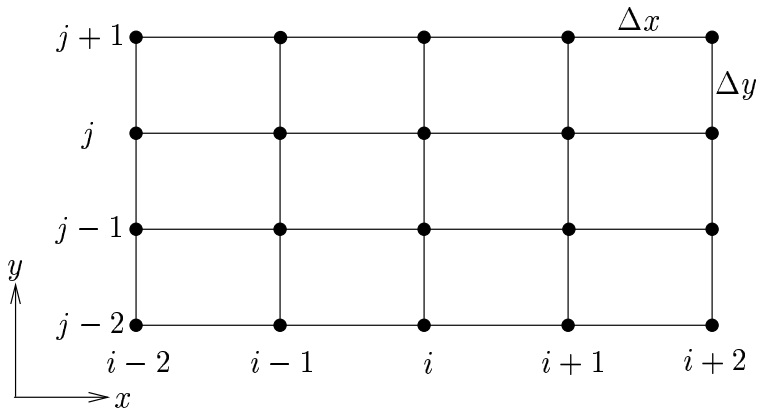


Figure 8.1: Two-dimensional structured mesh for finite difference approximations.

and a forward difference approximation is,

$$\left.\frac{\partial U}{\partial x}\right|_{i,j} \approx \delta_x^+ U_{i,j} \equiv \frac{1}{\Delta x}\left(U_{i+1,j} - U_{i,j}\right), \tag{8.4}$$

A very common finite difference approximation for a second derivative is,

$$\left.\frac{\partial^2 U}{\partial x^2}\right|_{i,j} \approx \delta_x^2 U_{i,j} \equiv \frac{1}{\Delta x^2}\left(U_{i+1,j} - 2U_{i,j} + U_{i-1,j}\right). \tag{8.5}$$

As in the first derivative case, since this derivative approximation uses both $i \pm 1$ values, it is known as a central difference approximation of the second derivative. This approximation can be motivated by approximating the second derivatives as a difference of first derivatives,

$$\begin{aligned}
\frac{\partial^2 U}{\partial x^2}(x) &= \lim_{\Delta x \to 0} \frac{\partial U/\partial x(x + \Delta x/2) - \partial U/\partial x(x - \Delta x/2)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{1}{\Delta x}\left[\frac{U(x+\Delta x) - U(x)}{\Delta x} - \frac{U(x) - U(x-\Delta x)}{\Delta x}\right] \\
&= \lim_{\Delta x \to 0} \frac{1}{\Delta x^2}\left[U(x+\Delta x) - 2U(x) + U(x-\Delta x)\right].
\end{aligned}$$

To approximate the convection-diffusion equation we can combine various finite difference derivative approximations. For example, consider the one-dimensional convection-diffusion equation,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} = \mu\frac{\partial^2 U}{\partial x^2}. \tag{8.6}$$

Approximating this using central differences for all derivatives, the convection-diffusion equation at node i can be approximated as,

$$\frac{dU_i}{dt} + u_i \delta_{2x} U_i = \mu \delta_x^2 U_i. \tag{8.7}$$

This is an ordinary differential equation for U_i which is coupled to the nodal values at $U_{i\pm1}$. To make this a fully discrete approximation, we need to discretize in time. To do this, we could apply any of the ODE integration methods that we discussed previously. For example, the simple forward Euler integration method would give,

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + u_i^n \delta_{2x} U_i^n = \mu \delta_x^2 U_i^n. \tag{8.8}$$

This method is widely known as a Forward Time-Central Space (FTCS) approximation.

In finite differences, we often make use of the spatial stencil of a discretization. The spatial stencil is simply the subset of nodes which are used to discretize the problem in space. The central difference approximation in Equation 8.7 gives a three-point spatial stencil since dU/dt at node i depends on three nodal values of U (including itself), specifically U_{i-1}, U_i, and U_{i+1}. To illustrate this, note that Equation (8.7) can be written as,

$$\frac{dU_i}{dt} = a_{i-1}U_{i-1} + a_i U_i + a_{i+1}U_{i+1}$$

where

$$\begin{aligned} a_{i-1} &= \frac{u_i}{2\Delta x} + \frac{\mu}{\Delta x^2}, \\ a_i &= -2\frac{\mu}{\Delta x^2}, \\ a_{i+1} &= -\frac{u_i}{2\Delta x} + \frac{\mu}{\Delta x^2}, \end{aligned}$$

We can also place all of the nodal states into a vector,

$$U = (U_1, U_2, \ldots, U_{i-1}, U_i, U_{i+1}, \ldots, U_{N_x-1}, U_{N_x})^T,$$

where N_x is the total number of points in the x-direction. Then, the finite difference approximation of convection-diffusion can be written in the following form,

$$\frac{dU}{dt} = AU + b, \tag{8.9}$$

where b will contain boundary condition related data (boundary conditions are discussed in Section 8.2) and the matrix A is,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \ldots & A_{1,N_x} \\ A_{2,1} & A_{2,2} & A_{2,3} & \ldots & A_{2,N_x} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{Nx,1} & A_{N_x,2} & A_{N_x,3} & \ldots & A_{N_x,N_x} \end{pmatrix}$$

Note that row i of this matrix contains the coefficients of the nodal values for the ODE governing node i. Except for rows requiring boundary condition data, the values of $A_{i,j}$ are related to the coefficients $a_{i\pm1}$ and a_i, specifically,

$$A_{i,i-1} = a_{i-1}, \quad A_{i,i} = a_i, \quad A_{i,i+1} = a_{i+1},$$

and all other entries in row i are zero.

In-class Discussion 8.1 (Finite Volume vs. Finite Difference Discretizations)

8.2 Boundary Conditions

In this section, we discuss the implementation of finite difference methods at boundaries. This discussion is not meant to be comprehensive, as the issues are many and often subtle. In particular, we only focus on Dirichlet boundary conditions.

A Dirichlet boundary condition is one in which the state is specified at the boundary. For example, in a heat transfer problem, the temperature might be known at the boundary. Dirichlet boundary conditions can be implemented in a relatively straightforward manner. For example, suppose that we are solving a one-dimensional convection-diffusion problem and we want the value of U at $i = 1$, to be U_{inlet},

$$U_1 = U_{inlet}.$$

To implement this, we fix $U_1 = U_{inlet}$ and apply the finite difference discretization only over the interior of the computational domain accounting for the known value of U_1 at any place where the interior discretization depends on it. For example, at the first interior node (i.e. $i = 2$), the central difference discretization of 1-D convection-diffusion gives,

$$\frac{dU_2}{dt} + u_2 \frac{U_3 - U_1}{2\Delta x} = \mu \frac{U_3 - 2U_2 + U_1}{\Delta x^2}.$$

Accounting for the known value of U_1, this becomes,

$$\frac{dU_2}{dt} + u_2 \frac{U_3 - U_{inlet}}{2\Delta x} = \mu \frac{U_3 - 2U_2 + U_{inlet}}{\Delta x^2}. \tag{8.10}$$

In terms of the vector notation, when a Dirichlet boundary condition is applied we usually remove that state from the vector U. So, in the situation where U_1 is known, the state vector is defined as,

$$U = (U_2, U_3, \ldots, U_{i-1}, U_i, U_{i+1}, \ldots, U_{N_x-1}, U_{N_x})^T,$$

The b vector then will contain the contributions from the known boundary values. For example, by re-arranging Equation (8.10), the first row of b contains,

$$b_1 = u_2 \frac{U_{inlet}}{2\Delta x} + \mu \frac{U_{inlet}}{\Delta x^2}.$$

Since U_{inlet} does not enter any of the other node's stencils, the remaining rows of b will be zero (unless the are altered by the other boundary). Note, the first row of A is,

$$A_{1,1} = -2\frac{\mu}{\Delta x^2}, \qquad A_{1,2} = -\frac{u_2}{2\Delta x} + \frac{\mu}{\Delta x^2},$$

and $A_{1,j} = 0$ for $j > 2$.

Example 8.1 (Finite Difference Method applied to 1-D Convection) *In this example, we solve 1-D convection,*

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} = 0,$$

using a central difference spatial approximation with a forward Euler time integration,

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + u_i^n \delta_{2x} U_i^n = 0.$$

Note: this approximation is the Forward Time-Central Space method from Equation (8.8) with the diffusion terms removed.

We will solve a problem that is nearly the same as that in Example 7.4. Specifically, the initial condition is

$$U_0(x) = e^{-x^2}.$$

We let the velocity, $u = 1$. Instead of solving a problem with periodicity, we enforce the inflow boundary (which is at $x = x_L$ since $u > 0$),

$$U(t, x_L) = e^{-x_L^2}.$$

At the outflow, since $u > 0$, no boundary condition is needed at $x = x_R$ (i.e. the scalar just convects out the domain at this location). But, the central difference discretization cannot be applied at the outlet node (i.e. at $i = N_x$) because it would require the $N_x + 1$ nodal value (which does not exist). To handle this problem, we switch the discretization at $i = N_x$ to a one-sided backwards difference formula.

A Matlab script that implements this algorithm is:

```
% This Matlab script solves the one-dimensional convection
% equation using a finite difference algorithm.  The
% discretization uses central differences in space and forward
% Euler in time.  An inflow bc is set and at the outflow a
% one-sided (backwards) difference is used.  The initial condition
% is a Gaussian distribution.
%

clear all;

% Set-up grid
xL = -4;
xR =  4;
```

```
Nx = 51; % number of control volumes
x = linspace(xL,xR,Nx);

% Calculate cell size in control volumes (assumed equal)
dx = x(2) - x(1);

% Set velocity
u = 1;

% Set final time
tfinal = 100;

% Set timestep
CFL = 0.1;
dt = CFL*dx/abs(u);

% Set initial condition
U = exp(-x.^2);
t = 0;

% Set bc state at left (assumes u>0)
UL = exp(-xL^2);

% Loop until t > tfinal
while (t < tfinal),

  % Copy old state vector
  U0 = U;

  % Inflow boundary
  U(1) = UL;

  % Interior nodes
  for i = 2:Nx-1,
    U(i) = U0(i) - dt*u*(U0(i+1)-U0(i-1))/(2*dx);
  end

  % Outflow boundary uses backward difference
  i = Nx;
  U(i) = U0(i) - dt*u*(U0(i)-U0(i-1))/(dx);

  % Increment time
  t = t + dt;
```

```
  % Plot current solution
  plot(x,U,'*');
  xlabel('x'); ylabel('U');
  title(sprintf('time = %f\n',t));
  axis([xL, xR, -0.5, 1.5]);
  grid on;
  drawnow;

end
```

When this method is run, the initial Gaussian disturbance convects out the domain, however small oscillations are observed which start at the outlet and move upstream (i.e. to the left). Eventually, these oscillations grow until the entire solution is contaminated. In Lecture 9 we will show that the FTCS algorithm is unstable for any Δt for pure convection. Thus, what we are observing is an instability that can be predicted through some analysis.

8.3 Truncation Error Analysis

In the discussion of ODE integration, we used the ideas of consistency and stability to prove convergence through the Dahlquist Equivalence Theorem. Similar concepts also exist for PDE discretizations, however, we cannot cover these here. We will briefly look at the local truncation error for finite difference approximations of derivatives, and as well the governing partial differential equations.

8.3.1 Truncation Error for a Derivative Approximation

Suppose we use a backwards difference, $\delta_x^- U_i$ to approximate the first derivative, U_x at point i. The local truncation error for this derivative approximation can be calculated using Taylor series as we have done in the past:

$$\begin{aligned}
\tau &\equiv \delta_x^- U_i - U_{xi}, \\
&= \frac{1}{\Delta x}\left(U_i - U_{i-1}\right) - U_{xi}, \\
&= \frac{1}{\Delta x}\left[U_i - \left(U_i - \Delta x U_{xi} + \frac{1}{2}\Delta x^2 U_{xxi} + O(\Delta x^3)\right)\right] - U_{xi}, \\
&= -\frac{1}{2}\Delta x U_{xxi} + O(\Delta x^2).
\end{aligned}$$

Thus, the analysis shows that the backwards difference is a first-order accurate discretization of the derivative at node i.

In-class Discussion 8.2 (Central difference approximation of U_{xi})

In-class Discussion 8.3 (Central difference approximation of U_{xxi})

8.3.2 Truncation Error for a PDE

Similar to the ODE case, the truncation error is defined as the remainder after an exact solution to the governing equation is substituted into the finite difference approximation. For example, suppose we are using the FTCS algorithm in Equation (8.8) to approximate the one-dimensional convection-diffusion equation. Then the local truncation error for the PDE approximation is defined as,

$$\tau \equiv \frac{U_i^{n+1} - U_i^n}{\Delta t} + u_i^n \delta_{2x} U_i^n - \mu \delta_x^2 U_i^n, \tag{8.11}$$

where $U(x,t)$ is an exact solution to Equation (8.6). Note that the truncation error defined here for PDE's is not quite a direct analogy with the standard definition of local truncation error used in ODE integration, specifically in Equation (2.2). In particular, in the ODE case, the truncation error is defined as the difference between the numerical solution and the exact solution after one step of the method (starting from exact values). However, in the PDE case, we have defined the truncation error as the remainder after an exact solution is substituted into the numerical method when the numerical method is written in a form that approximates the governing PDE.

Except for this difference in the definition, the calculation of the local truncation follows the same procedure as in the ODE case in which Taylor series substitutions are used to

expand the error in powers of Δt. Continuing on with our example, we use Taylor series of U about $t = t^n$ and $x = x_i$. However, we can use our previous results from the analysis of the truncation error of spatial derivatives. Specifically, we showed in during in-class discussions that,

$$\begin{aligned} \delta_{2x} U_i &= U_{xi} + \frac{1}{6}\Delta x^2 U_{xxxi} + O(\Delta x^4) \\ \delta_x^2 U_i &= U_{xxi} + \frac{1}{12}\Delta x^2 U_{xxxxi} + O(\Delta x^4) \end{aligned}$$

Using these results,

$$\tau = \frac{U_i^{n+1} - U_i^n}{\Delta t} + u_i^n \left[{U_{xi}}^n + \frac{1}{6}\Delta x^2 {U_{xxxi}}^n + O(\Delta x^4) \right] - \mu \left[{U_{xxi}}^n + \frac{1}{12}\Delta x^2 {U_{xxxxi}}^n + O(\Delta x^4) \right]$$

Then, performing a similar Taylor series of the time derivative approximation gives,

$$\frac{U^{n+1} - U^n}{\Delta t} = {U_t}^n + \frac{1}{2}\Delta t {U_{tt}}^n + O(\Delta t^2).$$

Substituting this into τ and collecting terms in powers of Δt and Δx gives,

$$\begin{aligned} \tau &= {U_{ti}}^n + u_i^n {U_{xi}}^n + -\mu {U_{xxi}}^n + \\ &\quad \frac{1}{2}\Delta t {U_{tti}}^n + O(\Delta t^2) + \\ &\quad \frac{1}{6}\Delta x^2 u_i^n {U_{xxxi}}^n - \frac{1}{12}\Delta x^2 \mu {U_{xxxxi}}^n + O(\Delta x^4) \end{aligned}$$

The first line of thie equation is actually just the PDE, evaluated at $t = t^n$ and $x = x_i$. Since U is an exact solution to the PDE, this is zero. The second line shows that the time discretization introduces an $O(\Delta t)$. The third line shows that the spatial discretization introduces an $O(\Delta x^2)$ error. Thus, this numerical method is first-order accurate in time and second-order accurate in space.

Lecture 9

Matrix Stability Analysis of Finite Difference Methods

In this lecture, we analyze the stability of PDE discretizations using a matrix approach. This method builds upon our understanding of eigenvalue stability for systems of ODE's.

As we saw in Section 8.1, finite difference (or finite volume) approximations can potentially be written in a semi-discrete form as,

$$\frac{dU}{dt} = AU + b. \tag{9.1}$$

While there are some PDE discretization methods that cannot be written in that form, the majority can be. So, we will take the semi-discrete Equation (9.1) as our starting point. Note: the term semi-discrete is used to signify that the PDE has only been discretized in space.

Let $U(t)$ be the exact solution to the semi-discrete equation. Then, consider perturbation $e(t)$ to the exact solution such that the perturbed solution, $V(t)$, is:

$$V(t) = U(t) + e(t).$$

The questions that we wish to resolve are: (1) can the perturbation $e(t)$ grow in time for the semi-discrete problem, and (2) what the stability limits are on the timestep for a chosen time integration method.

First, we substitute $V(t)$ into Equation (9.1),

$$\begin{aligned} \frac{dV}{dt} &= AV + b \\ \frac{d(U+e)}{dt} &= A(U+e) + b \\ \frac{de}{dt} &= Ae. \end{aligned}$$

Thus, the perturbation must satisfy the homogeneous equation, $e_t = Ae$. Having studied the behavior of linear system of equations in Section 4.2, we know that $e(t)$ will grow unbounded as $t \to \infty$ if the real parts of the eigenvalues of A are positive.

The problem is that determining the eigenvalues of A can be non-trivial. In fact, for a general problem finding the eigenvalues of A can be about as hard as solving the specific problem. So, while the matrix stability method is quite general, it can also require a lot of time to perform. Still, the matrix stability method is an indispensible part of the numerical analysis toolkit.

As we saw in the eigenvalue analysis of ODE integration methods, the integration method must be stable for all eigenvalues of the given problem. One manner that we can determine whether the integrator is stable is by plotting the eigenvalues scaled by the timestep in the complex $\lambda\Delta t$ plane and overlaying the stability region for the desired ODE integrator. In fact, we have already plotted the eigenvalues for one-dimensional diffusion using a central difference discretization in Example 5.1. Then, Δt can be adjusted to attempt to bring all eigenvalues into the stability region for the desired ODE integrator.

Example 9.1 (Matrix Stability of FTCS for 1-d convection) *In Example 8.1, we used a forward time, central space (FTCS) discretization for 1-d convection,*

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + u_i^n \delta_{2x} U_i^n = 0. \tag{9.2}$$

Since this method is explicit, the matrix A does not need to be constructed directly, rather Equation (9.2) can be used to find the new values of U at each point i. The Matlab script given in Example 8.1 does exactly that. However, if we are interested in calculating the eigenvalues to analyze the eigenvalue stability, then the A matrix is required. The following script does exactly that (i.e. calculates A, determines the eigenvalues of A, and then plots the eigenvalues scaled by Δt overlayed with the forward Euler stability region). The script can set either the inflow/outflow boundary conditions described in Example 8.1, or can set periodic boundary conditions. We will look at the eigenvalues of both cases.

```
% This Matlab script calculates the eigenvalues of
% the one-dimensional convection equation discretized by
% finite differences.  The discretization uses central
% differences in space and forward Euler in time.
%
% Periodic bcs are set if periodic_flag == 1.
%
% Otherwise, an inflow (dirichlet) bc is set and at
% the outflow a one-sided (backwards) difference is used.
%

clear all;

periodic_flag = 1;

% Set-up grid
xL = -4;
xR =  4;
```

```
Nx = 21; % number of points
x = linspace(xL,xR,Nx);

% Calculate cell size in control volumes (assumed equal)
dx = x(2) - x(1);

% Set velocity
u = 1;

% Set timestep
CFL = 1;
dt = CFL*dx/abs(u);

% Set bc state at left (assumes u>0)
UL = exp(-xL^2);

% Allocate matrix to hold stiffness matrix (A).
%
A = zeros(Nx-1,Nx-1);

% Construct A except for first and last row
for i = 2:Nx-2,
  A(i,i-1) =  u/(2*dx);
  A(i,i+1) = -u/(2*dx);
end

if (periodic_flag == 1), % Periodic bcs

  A(1   ,2   ) = -u/(2*dx);
  A(1   ,Nx-1) =  u/(2*dx);
  A(Nx-1,1   ) = -u/(2*dx);
  A(Nx-1,Nx-2) =  u/(2*dx);

else % non-periodic bc's

  % At the first interior node, the i-1 value is known (UL).
  % So, only the i+1 location needs to be set in A.
  A(1,2) = -u/(2*dx);

  % Outflow boundary uses backward difference
  A(Nx-1,Nx-2) =  u/dx;
  A(Nx-1,Nx-1) = -u/dx;

end
```

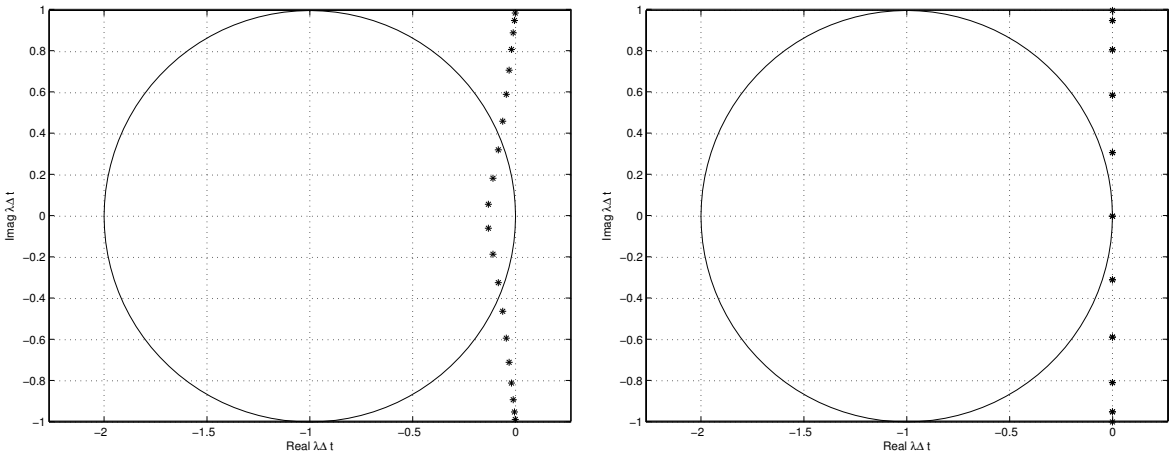
```
% Calculate eigenvalues of A
lambda = eig(A);

% Plot lambda*dt
plot(lambda*dt,'*');
xlabel('Real \lambda\Delta t');
ylabel('Imag \lambda\Delta t');

% Overlay Forward Euler stability region
th = linspace(0,2*pi,101);
hold on;
plot(-1 + sin(th),cos(th));
hold off;
axis('equal');
grid on;
```

Figure 9.1 shows a plot of $\lambda\Delta t$ for a CFL set to one. Recall that for this one-dimensional problem, the CFL number was defined as,

$$CFL = \frac{|u|\Delta t}{\Delta x}.$$

In the inflow/outflow boundary condition case (shown in Figure 9.1) the eigenvalues lay slightly inside the negative real half-plane. As they move away from the origin, they approach the imaginary axis at $\pm i$. The periodic boundary conditions give purely imaginary eigenvalues but these also approach $\pm i$ as the move away from the origin. Note that the periodic boundary conditions actually give a zero eigenvalue so that the matrix A is actually singular (Why is this?). Regardless what we see is that for a $CFL = 1$, some $\lambda\Delta t$ exist which are outside of the forward Euler stability region. We could try to lower the timestep to bring all of the $\lambda\Delta t$ into the stability region, however that will prove to be practically impossible since the extreme eigenvalues approach $\pm i$ (i.e. they are purely imaginary). Thus, no finite value of Δt exists for which these eigenvalues can be brought inside the circular stability region of the forward Euler method (i.e. the FTCS is unstable for convection).

In-class Discussion 9.1 (Behavior of FTCS eigenvalues with decreased Δx) *We will discuss Figure 9.2.*



(a) Dirichlet inflow and upwinded outflow

(b) Periodic boundary conditions

Figure 9.1: $\lambda\Delta t$ distribution for one-dimensional convection example using two different boundary conditions. Note: Δt set such that CFL $= 1$.

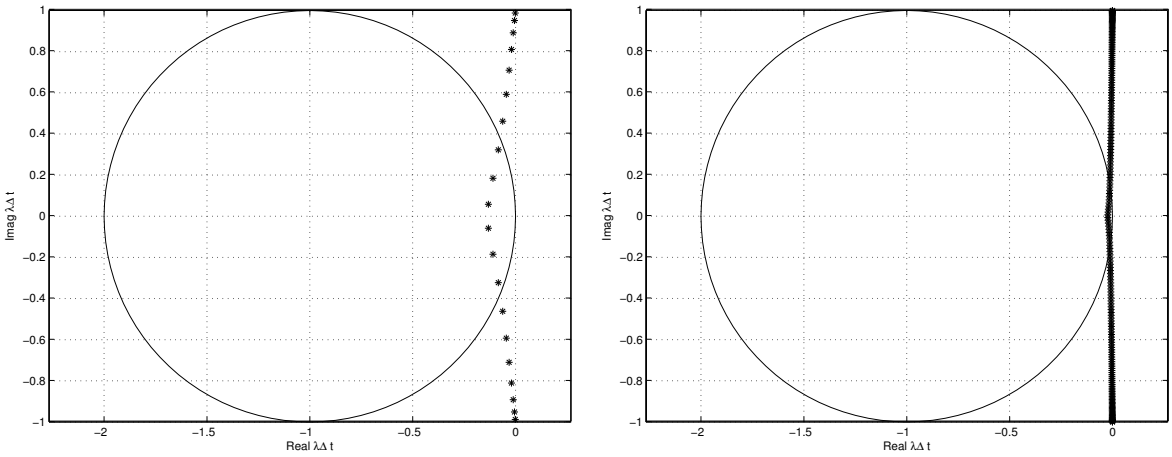
In-class Discussion 9.2 (Behavior of FTCS eigenvalues with diffusion) *We will discuss Figure 9.3.*

Though the eigenvalues of A typically require numerical techniques for the general problem, a special case of practical interest occurs when the matrix is 'periodic'. That is, the column entries shift a column every row. Thus, the matrix has the form,

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_N \\ a_N & a_1 & a_2 & \dots & a_{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_2 & a_3 & a_4 & \dots & a_1 \end{pmatrix}$$

This type of matrix is known as a circulant matrix. Circulant matrices have eigenvalues given by,

$$\lambda_n = \sum_{j=1}^{N} a_j e^{i\,2\pi(j-1)\frac{n}{N}} \qquad \text{for} \quad n = 0, 1, \dots, N-1 \tag{9.3}$$



(a) $\Delta x = 0.4$ (b) $\Delta x = 0.04$

Figure 9.2: Effect of Δx on $\lambda \Delta t$ distribution for one-dimensional convection example using Dirichlet inflow and upwinded outflow conditions. Note: Δt set such that CFL $= 1$.

Example 9.2 *As we saw in Example 9.1, when periodic boundary conditions are assumed, the central space discretization of one-dimensional convection gives purely imaginary eigenvalues, and when scaled by a timestep for which the CFL number is one, the eigenvalues stretch along the axis until $\pm i$. Since for a convection problem with constant velocity and periodic boundary conditions gives a circulant matrix, we can use Equation (9.3) to determine the eigenvalues analytically. We begin by finding the coefficients, a_j. For a central space discretization, we find,*

$$a_2 = -\frac{u}{2\Delta x}, \quad a_N = \frac{u}{2\Delta x}, \text{ and for all other } j, \quad a_j = 0.$$

Then, substituting these a_j into Equation (9.3) gives,

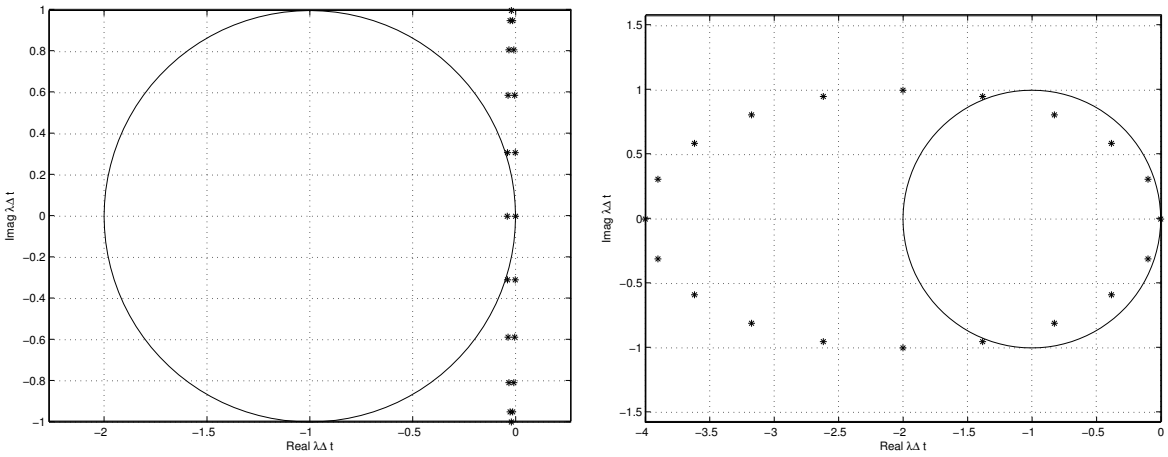
$$\begin{aligned} \lambda_n &= -\frac{u}{2\Delta x} e^{i\,2\pi \frac{n}{N}} + \frac{u}{2\Delta x} e^{i\,2\pi (N-1) \frac{n}{N}}, \\ &= -\frac{u}{2\Delta x} e^{i\,2\pi \frac{n}{N}} + \frac{u}{2\Delta x} e^{i\,2\pi n} e^{-i\,2\pi \frac{n}{N}}. \end{aligned}$$

Since $e^{i\,2\pi n} = 1$ (because n is an integer), then,

$$\begin{aligned} \lambda_n &= -\frac{u}{2\Delta x} e^{i\,2\pi \frac{n}{N}} + \frac{u}{2\Delta x} e^{-i\,2\pi \frac{n}{N}}, \\ &= -\frac{u}{2\Delta x} \left(e^{i\,2\pi \frac{n}{N}} - e^{-i\,2\pi \frac{n}{N}} \right), \\ &= -i \frac{u}{\Delta x} \sin\left(2\pi \frac{n}{N} \right). \end{aligned}$$

Multiplying by the timestep,

$$\lambda_n \Delta t = -i \frac{u \Delta t}{\Delta x} \sin\left(2\pi \frac{n}{N} \right).$$



(a) $u\Delta x/\mu = 100$ (b) $u\Delta x/\mu = 1$

Figure 9.3: Effect of viscosity on $\lambda\Delta t$ distribution for one-dimensional convection-diffusion example using periodic boundary conditions.Note: Δt set such that CFL $= 1$.

As observed in Example 9.1, the eigenvalues are purely imaginary and will extend to $\pm i$ *when* $CFL = |u|\Delta t/\Delta x = 1$.

Lecture 10

Fourier Analysis of Finite Difference Methods

In this lecture, we determine the stability of PDE discretizations using Fourier analysis. First, we begin with Fourier analysis of PDE's, and then extend these ideas to finite difference methods.

10.1 Fourier Analysis of PDE's

We will develop Fourier analysis in one dimension. The basic ideas extend easily to multiple dimensions. We will consider the convection-diffusion equation,

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} = \mu\frac{\partial^2 U}{\partial x^2}.$$

We will assume that the velocity, u, and the viscosity, μ are constant.

The solution is assumed to be periodic over a length L. Thus,

$$U(x + mL, t) = U(x, t)$$

where m is any integer. As we saw in Lecture 9, as the mesh is refined, the eigenvalues of the systems with general boundary conditions tend to approach the eigenvalues of the periodic case. Thus, we expect this periodicity assumption to still lead to insight into more general boundary conditions especially as the mesh is refined.

A Fourier series with periodicity over length L is given by,

$$U(x,t) = \sum_{m=-\infty}^{+\infty} \hat{U}_m(t) e^{ik_m x} \qquad \text{where} \qquad k_m = \frac{2\pi m}{L}. \tag{10.1}$$

k_m is generally called the wavenumber, though m is the number of waves occurring over the length L. We note that $\hat{U}_m(t)$ is the amplitude of the m-th wavenumber and it is generally complex (since we have used complex exponentials). Substituting the Fourier series into the convection-diffusion equation gives,

$$\frac{\partial}{\partial t}\left[\sum_{m=-\infty}^{+\infty} \hat{U}_m(t) e^{ik_m x}\right] + u\frac{\partial}{\partial x}\left[\sum_{m=-\infty}^{+\infty} \hat{U}_m(t) e^{ik_m x}\right] = \mu\frac{\partial^2}{\partial x^2}\left[\sum_{m=-\infty}^{+\infty} \hat{U}_m(t) e^{ik_m x}\right].$$

$$\sum_{m=-\infty}^{+\infty} \frac{d\hat{U}_m}{dt} e^{ik_m x} + u \sum_{m=-\infty}^{+\infty} ik_m \hat{U}_m e^{ik_m x} = \mu \sum_{m=-\infty}^{+\infty} (ik_m)^2 \hat{U}_m e^{ik_m x}.$$

Noting the $i^2 = -1$ and collecting terms gives,

$$\sum_{m=-\infty}^{+\infty} \left[\frac{d\hat{U}_m}{dt} + \left(iuk_m + \mu k_m^2 \right) \hat{U}_m \right] e^{ik_m x} = 0. \tag{10.2}$$

The next step is to utilize the orthogonality of the different Fourier modes over the length L, specifically,

$$\int_0^L e^{-ik_n x} e^{ik_m x} = \begin{cases} 0 & \text{if } m \neq n \\ L & \text{if } m = n \end{cases} \tag{10.3}$$

By multiplying Equation (10.2) by $e^{-ik_n x}$ and integrating from 0 to L, the orthogonality condition gives,

$$\frac{d\hat{U}_n}{dt} + \left(iuk_n + \mu k_n^2 \right) \hat{U}_n = 0, \qquad \text{for any integer value of } n. \tag{10.4}$$

Thus, the evolution of the amplitude for an individual wavenumber is independent of the other wavenumbers. The solution to Equation (10.4),

$$\hat{U}_n(t) = \hat{U}_n(0) e^{-iuk_n t} e^{-\mu k_n^2 t}.$$

The convection term, which results in the complex time dependent behavior, $e^{-iuk_n t}$, only oscillates and does not change the magnitude of $\hat{U}_n$. The diffusion term causes the magnitude to decrease as long as $\mu > 0$. But, if the diffusion coefficient were negative, then the magnitude would increase unbounded with time. Thus, in the case of the convection-diffusion PDE, as long as $\mu \geq 0$, this solution is stable.

10.2 Semi-Discrete Fourier Analysis

Now we move on to Fourier analysis of a semi-discrete equation. That is, we discretize in space but not time. To be specific and simple, let consider pure convection discretized with central differences. This is exactly the case considered in Example 9.2 so we already know the results. But here, we derive the eigenvalues of the central difference discretization using Fourer analysis. The semi-discrete equation is,

$$\frac{dU_j}{dt} + u\, \delta_{2x} U_j = 0, \qquad \Rightarrow \qquad \frac{dU_j}{dt} + \frac{u}{2\Delta x} \left(U_{j+1} - U_{j-1} \right) = 0. \tag{10.5}$$

Note that we have switched our indices from the usual i's to j's to avoid confusion in this discussion because the Fourier series we are about to introduce will give rise to the imaginary number, i.

For the analysis of PDE's, a Fourier series of infinite dimension was used (i.e. m ranged from $\pm\infty$ in Equation (10.1)). In that case, the infinite number of terms in the Fourier series corresponds to the fact that there are infinitely many values of U over the periodic domain

(because the space from $0 \leq x \leq L$ has an infinite number of points). In the semi-discrete case, only a finite number of values exist over the periodic domain. Specifically, if there are N equally-spaced nodes in the domain, then there will be $N-1$ unique values (because the values at the beginning and end of the domain must be the same due to periodicity). Summarizing, there can only be $N-1$ terms in the Fourier series used to describe a finite difference solution on a periodic domain with N points.

The most common set of $N-1$ modes used to analyze finite difference schemes is,

$$U_j(t) = \sum_{m=-N/2+1}^{N/2-1} \hat{U}_m(t) e^{ik_m j \Delta x} \qquad \text{for even } N, \tag{10.6}$$

and,

$$U_j(t) = \sum_{m=-(N-3)/2}^{(N-1)/2} \hat{U}_m(t) e^{ik_m j \Delta x} \qquad \text{for odd } N \tag{10.7}$$

Note that for large N, the limits on either Fourier series approach $\pm N/2$.

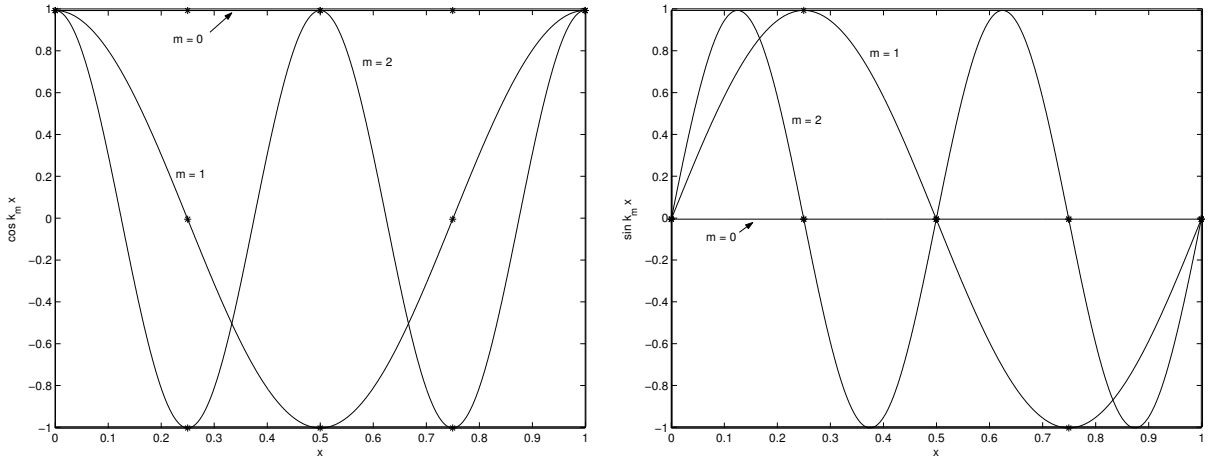
So what happens to the modes with higher wavenumbers? The answer is that the higher wavenumbers are aliased with the lower wavenumber when fewed only at a finite number of nodes. To demonstrate this, consider the case with $N = 5$ nodes. In this case, the values of m in the Fourier series are from -1 to 2. The real and imaginary parts of the modes are plotted in Figure 10.1 for m =0, 1, and 2. Although the mode shape is shown for all values of x, the only values of concern are those at the nodes. We can already see the potential for aliasing by looking at the imaginary part of the $m = 2$ mode. In this case, the nodal values of this mode are all zero. Thus, at the nodes, the imaginary part of $m = 2$ mode is identical to the imaginary part of the $m = 0$ mode (i.e. the are both zero at the nodes). The $m = 2$ mode is not completely aliased with the $m = 0$ mode, however, because the real part of the modes are different. Specifically, the real part of the $m = 0$ mode is constant (equal to one at all nodes), while the real part of the $m = 2$ mode oscillates between ± 1. This ± 1 oscillation between nodes is often called an odd-even or a sawtooth mode. To demonstrate what happens with higher wave numbers, Figure 10.2 shows the $m = 3$ mode, overlayed with the $m = -1$ mode. As can be seen from this plot, these two modes are indistinguishable from each other at the nodes.

If we proceed along the analogous lines to the Fourier analysis of PDE's in the previous section, we would substitute the Fourier series into Equation (10.5) and utilize a similar orthogonality relationship to arrive at the conclusion that each mode of the Fourier series behaves independently. Thus, from now on, we will simply substitute an individual mode into the discrete equations. That is, we assume that

$$U_j(t) = \hat{U}_m(t) e^{ik_m j \Delta x},$$

for a valid m. Substitution of this individual mode into Equation (10.5) gives,

$$\frac{d\hat{U}_m}{dt} e^{ik_m j \Delta x} + \frac{u}{2\Delta x} \left[e^{ik_m (j+1) \Delta x} - e^{ik_m (j-1) \Delta x} \right] \hat{U}_m = 0.$$



(a) Real $e^{ik_m x} = \cos k_m x$

(b) Imag $e^{ik_m x} = \sin k_m x$

Figure 10.1: Plots of $\cos k_m x$ and $\sin k_m x$ for $L = 1$ including the nodal values for a five node grid ($\Delta x = 0.2$).

Factoring out the term $e^{ik_m j\Delta x}$ gives,

$$\frac{d\hat{U}_m}{dt} + \frac{u}{2\Delta x}\left[e^{ik_m\Delta x} - e^{-ik_m\Delta x}\right]\hat{U}_m = 0.$$

$$\frac{d\hat{U}_m}{dt} + i\frac{u}{\Delta x}\sin(k_m\Delta x)\,\hat{U}_m = 0.$$

Thus, for each mode m we may write,

$$\frac{d\hat{U}_m}{dt} = \lambda_m \hat{U}_m,$$

where

$$\lambda_m = -i\frac{u}{\Delta x}\sin(k_m\Delta x).$$

Comparing these λ_m to the eigenvalues found in Example 9.2, we can show that they are identical.

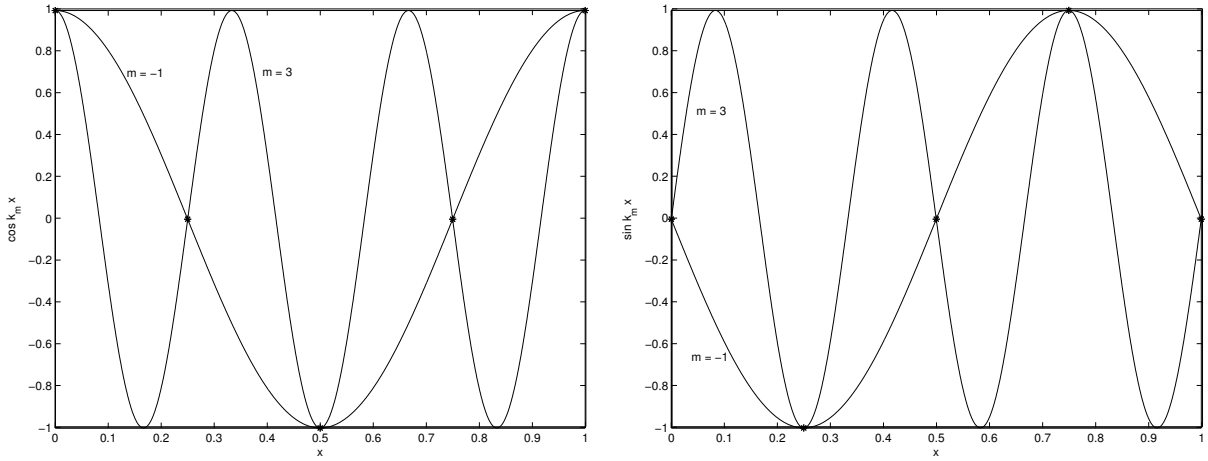
A parameter which occurs throughout Fourier analysis of spatial discretizations is,

$$\beta_x \equiv k_m\Delta x. \tag{10.8}$$

Since $k_m = 2\pi m/L$, then

$$\beta_m = 2\pi\frac{m\Delta x}{L}.$$

Since β_m varies with m, we can determine the range of β_m values corresponding to the range of m values. In particular, for large N, the limiting values of m approach $\pm N/2$, thus the



(a) Real $e^{ik_m x} = \cos k_m x$

(b) Imag $e^{ik_m x} = \sin k_m x$

Figure 10.2: Demonstration of aliasing of an $m = 3$ mode to $m = -1$ for a five node grid.

limiting values of β_m will be,

$$\lim_{\substack{m \to \pm N/2 \\ N \to \infty}} \beta_m = \pm 2\pi \frac{N\Delta x}{2L} = \pm\pi \frac{N\Delta x}{L} = \pm\pi \frac{N}{N-1} = \pm\pi.$$

Thus, assuming large N, the eigenvalue calculation for any spatial differencing method reduces to:

1. Substitute $U_j(t) = \hat{U}_m(t) e^{ij\beta_m}$ into the semi-discrete equation.
2. Determine the $\lambda_m(\beta_m)$ such that $\frac{d}{dt}\hat{U}_m = \lambda_m \hat{U}_m$.
3. Determine the limitations on the timestep such that $\lambda_m(\beta_m)\Delta t$ will be inside the stability region for a chosen time integration method for all $-\pi \leq \beta_m \leq \pi$.

Example 10.1 (Analysis of a first-order upwind discretization of convection) *In this example, we perform a Fourier analysis of a first-order upwind discretization of convection. Assuming the velocity is positive, then an upwind discretization of convection is,*

$$\frac{dU_j}{dt} + u\,\delta_x^- U_j = 0, \qquad \Rightarrow \qquad \frac{dU_j}{dt} + \frac{u}{\Delta x}\left(U_j - U_{j-1}\right) = 0.$$

Following the steps outlined above, we substitute the Fourier mode,

$$\frac{d\hat{U}_m}{dt} e^{ij\beta_m} + \frac{u}{\Delta x}\left[e^{ij\beta_m} - e^{i(j-1)\beta_m}\right]\hat{U}_m = 0.$$

Factoring out the term $e^{ik_m j\Delta x}$ gives,

$$\frac{d\hat{U}_m}{dt} + \frac{u}{\Delta x}\left(1 - e^{-i\beta_m}\right)\hat{U}_m = 0.$$

Thus,

$$\lambda_m \Delta t = -\frac{u\Delta t}{\Delta x}\left(1 - e^{-i\beta_m}\right) = -\frac{u\Delta t}{\Delta x}\left(1 - \cos\beta_m + i\sin\beta_m\right).$$

The upwind approximation gives both a real and imaginary part to the eigenvalue. In fact, the imaginary part is identical to the central difference part. The real part is negative, thus the upwind approximation adds stability (in the sense that a negative real eigenvalue corresponds to amplitudes that decrease in time). A plot of the eigenvalues is shown in Figure 10.3 for $u\Delta t/\Delta x = 1$. *The Matlab script for generating these eigenvalue plots is given below:*

```
bm = linspace(-pi,pi,21);
CFL = 1;
lamdt = -CFL*(1 - exp(-i*bm));

plot(lamdt,'*');
xlabel('Real \lambda\Delta t');
ylabel('Imag \lambda\Delta t');
grid on;
axis('equal');
```

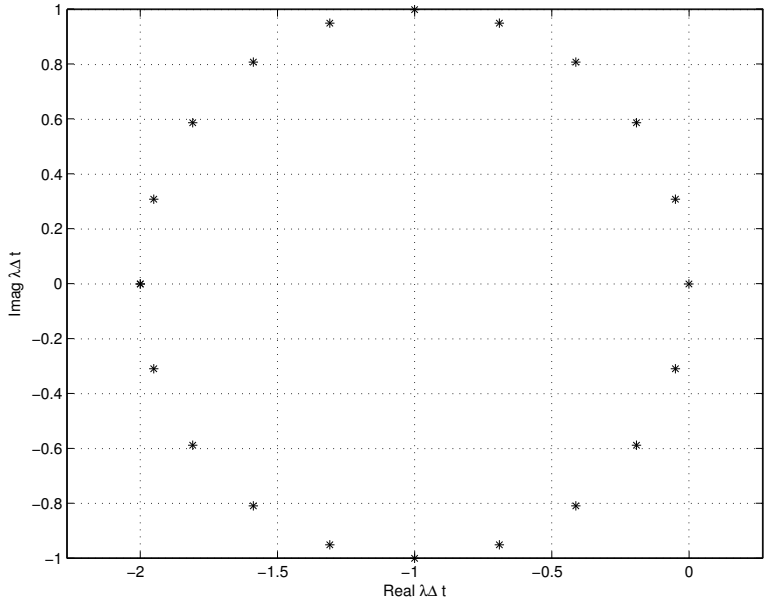


Figure 10.3: $\lambda_m \Delta t$ for a first-order upwind discretization of convection for CFL $= u\Delta t/\Delta x = 1$.

Lecture 11

Method of Weighted Residuals

In this lecture, we introduce the method of weighted residuals which provides the most general formulation for the finite element method. To begin, let's focus on the particular problem of steady heat diffusion in a rod. This problem can be modeled as a one-dimensional PDE for the temperature, T:

$$(kT_x)_x = -q, \tag{11.1}$$

where $k(x)$ thermal conductivity of the material and $q(x)$ is the heat source (per unit area), respectively. Note that both k and q could be functions of x. Also, let the physical domain for the problem be from $x = -L/2$ to $x = L/2$.

Example 11.1 (Steady heat diffusion) *Suppose that the rod has a length of $L = 2$, the thermal conductivity is constant, $k = 1$, and the heat source, $q(x) = 50e^x$. Assume that the temperature at the ends of the rod are to be maintained at $T(\pm 1) = 100$. Equation (11.1) can be integrated twice to obtain:*

$$\begin{aligned} (kT_x)_x &= -q, \\ T_{xx} &= -50e^x, \\ T_x &= -50e^x + a, \\ T &= -50e^x + ax + b. \end{aligned}$$

Now, applying boundary conditions so that $T(\pm 1) = 100$,

$$\begin{aligned} -50e^1 + a + b &= 100, \\ -50e^{-1} - a + b &= 100. \end{aligned}$$

This is a 2×2 system which can be solved for a and b,

$$\begin{aligned} a &= 50 \sinh 1 \\ b &= 100 + 50 \cosh 1, \end{aligned}$$

where $\cosh y = (e^y + e^{-y})/2$ and $\sinh y = (e^y - e^{-y})/2$. Thus, the exact solution is,

$$T = -50e^x + 50x \sinh 1 + 100 + 50 \cosh 1. \tag{11.2}$$

A plot of this solution is shown in Figure 11.1.

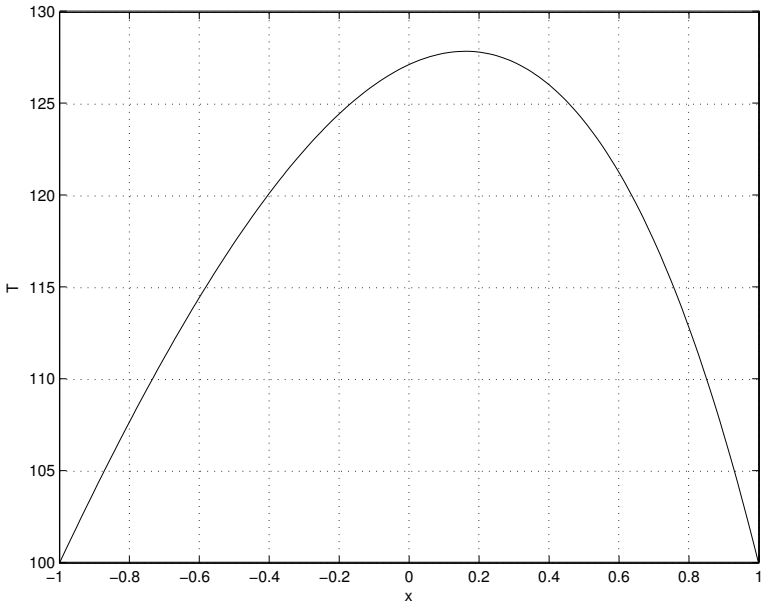


Figure 11.1: Temperature distribution for $q = 50e^x$, $L = 2$, and $k = 1$.

A common approach to approximating the solution to a PDE such as heat diffusion is to use a series of weighted functions. For example, for the temperature in Example 11.1 we might assume that,

$$\tilde{T}(x) = 100 + \sum_{i=1}^{N} a_i \phi_i(x),$$

where N is the number of terms (functions) in the approximation, $\phi_i(x)$ are the (known) functions, and a_i are the unknown function weights. The functions $\phi_i(x)$ are usually designed to satisfy the boundary conditions. So, in this example where the temperature is 100 at $x = \pm 1$, then $\phi_i(\pm 1) = 0$ (don't forget that $\tilde{T}$ was defined to include the constant term of 100).

The question remains what functions (and how many) to choose for $\phi_i(x)$. While many good choices exist, we will use polynomials in x because polynomial approximations are used extensively in finite element methods (our main interest). For this problem, the following ideas can be used to determine the form of the $\phi_i(x)$:

- First, we note that requiring $\phi_i(\pm 1) = 0$ places two conditions on each $\phi_i(x)$. These two conditions can be satisfied with a linear function of x, but the linear function which equals 0 at $x = \pm 1$ is simply 0. Since this does not add anything to the solution even after multiplying by a weight, the first non-trivial function would be a quadratic function.

- A quadratic function can be designed to satisy the boundary conditions in the following manner,

$$\phi_1(x) = (1 + x)(1 - x).$$

By including factors which go to zero at the end points, we have constructed a quadratic function which will satisfy the required boundary conditions. A plot of $\phi_1(x)$ is shown in Figure 11.2.

- Suppose we wanted to include a cubic polynomial in the approximation, then one way we could do this is multiply $\phi_1(x)$ by x.

$$\phi_2(x) = x\phi_1(x) = x(1+x)(1-x).$$

Since $\phi_1(x)$ goes to zero at the end points, then so will $\phi_2(x)$. A plot of $\phi_2(x)$ is shown in Figure 11.2. There are actually some better ways to choose these higher-order polynomials then simply multiplying the lowest order polynomial by powers of x. The problem with the current approach is that if the number of terms were large (so that the powers of x would be large), then the set of polynomials (i.e. $\phi_i(x)$) become very poorly conditioned resulting in many numerical difficulties. We will not discuss issues of conditioning but more advanced texts on finite element methods or related subjects can be consulted. For low order polynomial approximations, the issues of conditioning do not play an important role.

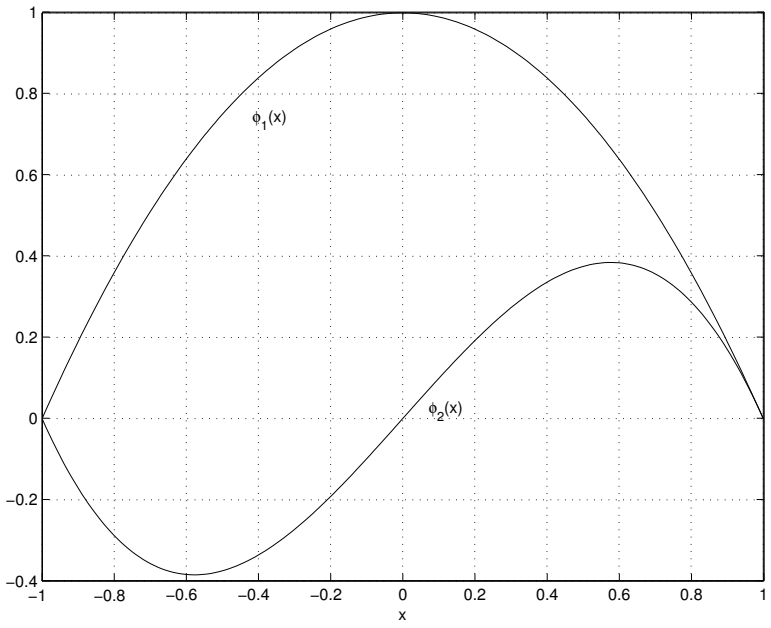


Figure 11.2: $\phi_1(x)$ and $\phi_2(x)$ for example heat diffusion problem.

Having selected a set of functions, we must now develop a way to determine values of a_i that will lead to a good approximation of the actual $T(x)$. While several ways exist to do this, we will focus on two methods in these introductory notes: the collocation method and the method of weighted residuals.

11.1 The Collocation Method

One approach to determine the N values of a_i would be to enforce the governing PDE at N points. Note that in general, the exact solution will not be a linear combination of the $\phi_i(x)$, so it will not be possible to enforce the PDE at every point in the domains. To see this, let's substitute $\tilde{T}(x)$ into Equation (11.1). Note that,

$$\begin{aligned} \tilde{T}_{xx} &= \frac{\partial^2}{\partial x^2}\left[a_1\phi_1(x) + a_2\phi_2(x)\right], \\ (\phi_1)_{xx} &= -2, \\ (\phi_2)_{xx} &= -6x, \\ \Rightarrow \tilde{T}_{xx} &= -2a_1 - 6a_2x. \end{aligned}$$

Next, we define a residual for Equation (11.1),

$$R(\tilde{T}, x) \equiv \left(k\tilde{T}_x\right)_x + q.$$

If the solution were exact, then the $R = 0$ for all x. Now, substitution of our chosen $\tilde{T}$ into the residual (recall $k = 1$ and $q = 50e^x$ in this example) gives,

$$R(\tilde{T}, x) = -2a_1 - 6a_2x + 50e^x. \tag{11.3}$$

Clearly, since a_1 and a_2 are constants (i.e. they do not depend on x), there is no way for this residual to be zero for all x.

The question remains, where should the N points be selected. The points at which the governing equation will be enforced are known as the collocation points. We will choose the relatively simple approach of equally subdividing the domain with $N = 2$ interior collocation points. For this domain from $-1 \le x \le 1$, the equi-distant collocation points would be at $x = \pm 1/3$. Thus, the two conditions for determining a_1 and a_2 are,

$$\begin{aligned} R(\tilde{T}, -1/3) &= 0, \\ R(\tilde{T}, 1/3) &= 0. \end{aligned}$$

From Equation (11.3) this gives,

$$\begin{aligned} -2a_1 + 2a_2 + 50e^{-1/3} &= 0, \\ -2a_1 - 2a_2 + 50e^{1/3} &= 0. \end{aligned}$$

Re-arranging this into a matrix form gives,

$$\begin{pmatrix} -2 & 2 \\ -2 & -2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -50e^{-1/3} \\ -50e^{1/3} \end{pmatrix}.$$

$$\Rightarrow \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 25\cosh 1/3 \\ 25\sinh 1/3 \end{pmatrix} = \begin{pmatrix} 26.402 \\ 8.489 \end{pmatrix}.$$

The results using this collocation method are shown in Figure 11.3 which includes plots of $\tilde{T}$, the error (i.e. $\tilde{T} - T$), and the residual. Note that the residual is clearly exactly zero at the collocation points (i.e. $x = \pm 1/3$), though the approximation is not exact at these points (i.e. $\tilde{T} \neq T$ at $x = \pm 1/3$).

11.2 The Method of Weighted Residuals

While the collocation method enforces the residual to be zero at N points, the method of weighted residuals requires N weighted integrals of the residual to be zero. A weighted residual is simply the integral of a weight function, $w(x)$ and the residual over the domain. For example, in the one-dimensional diffusion problem we are considering, a weighted residual is,

$$\int_{-1}^{1} w(x)\, R(\tilde{T}, x)\, dx.$$

By choosing N weight functions, $w_j(x)$ for $1 \leq j \leq N$ and setting these N weighted residuals to zero, we may determine N values of a_i. We define the weighted residual for $w_j(x)$ to be,

$$R_j(\tilde{T}) \equiv \int_{-1}^{1} w_j(x)\, R(\tilde{T}, x)\, dx.$$

And, the method of weighted residuals requires,

$$R_j(\tilde{T}) = 0 \qquad \text{for } 1 \leq j \leq N.$$

In the method of weighted residuals, the next step is to determine appropriate weight functions. A common approach, known as the Galerkin method, is to set the weight functions equal to the functions used to approximate the solution. That is,

$$w_j(x) = \phi_j(x). \quad \text{(Galerkin)}.$$

For the heat diffusion example we have been considering,

$$\begin{aligned} w_1(x) &= (1-x)(1+x), \\ w_2(x) &= x(1-x)(1+x). \end{aligned}$$

Now, we must calculate the weighted residuals. For the example,

$$\begin{aligned} R_1(\tilde{T}) &= \int_{-1}^{1} w_1(x)\, R(\tilde{T}, x)\, dx, \\ &= \int_{-1}^{1} (1-x)(1+x)\ (-2a_1 - 6a_2 x + 50e^x)\ dx, \\ &= -\frac{8}{3}a_1 + 200e^{-1}. \end{aligned}$$

To do this integral, the following results were used (the constants of integration are neglected),

$$\begin{aligned} \int (1-x)(1+x)\, dx &= x - \frac{1}{3}x^3, \\ \int x(1-x)(1+x)\, dx &= \frac{1}{2}x^2 - \frac{1}{4}x^4, \\ \int x^2 e^x\, dx &= x^2 e^x - 2xe^x + 2e^x. \end{aligned}$$

Similarly, calculating R_2:

$$\begin{aligned} R_2(\tilde{T}) &= \int_{-1}^{1} w_2(x)\, R(\tilde{T}, x)\, dx, \\ &= \int_{-1}^{1} x(1-x)(1+x)\ (-2a_1 - 6a_2 x + 50e^x)\ dx, \\ &= -\frac{8}{5}a_2 + 100e^1 - 1200e^{-1}, \end{aligned}$$

where the following results have been used,

$$\begin{aligned} \int x^2(1-x)(1+x)\, dx &= \frac{1}{3}x^3 - \frac{1}{5}x^5, \\ \int xe^x\, dx &= xe^x - e^x, \\ \int x^3 e^x\, dx &= x^3e^x - 3x^2e^x + 6xe^x - 6e^x. \end{aligned}$$

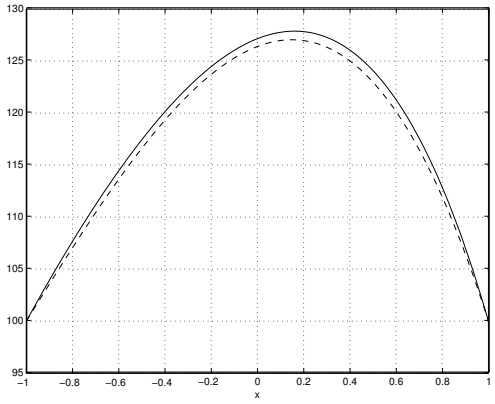
Finally, we can solve for a_1 and a_2 by setting the weighted residuals R_1 and R_2 to zero,

$$\begin{aligned} -\frac{8}{3}a_1 + 200e^{-1} &= 0, \\ -\frac{8}{5}a_2 + 100e^1 - 700e^{-1} &= 0. \end{aligned}$$

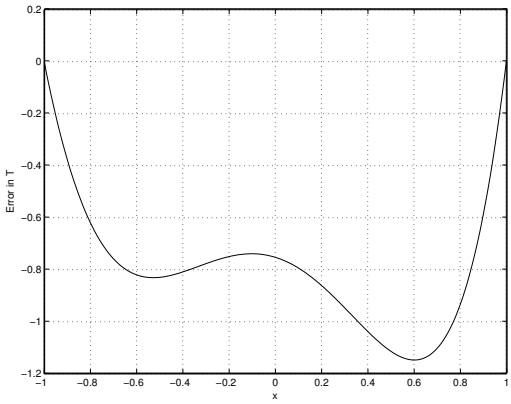
This could be written as a 2×2 matrix equation and solved, but the equations are decoupled and can be readily solved,

$$\Rightarrow \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 75e^{-1} \\ \frac{125}{2}e^1 - \frac{875}{2}e^{-1} \end{pmatrix} = \begin{pmatrix} 27.591 \\ 8.945 \end{pmatrix}.$$

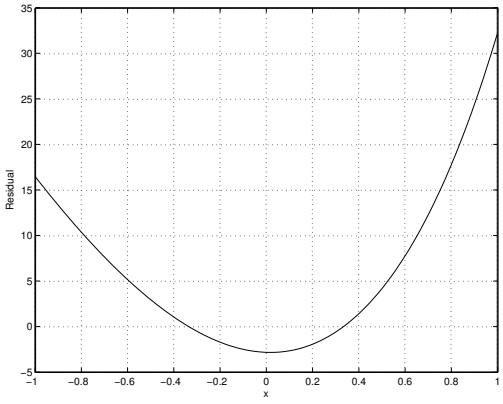
The results using this method of weighted residuals are shown in Figure 11.4. Comparison with the collocation method results shows that the method of weighted residuals is clearly more accurate.



(a) Comparison of T (solid) and $\tilde{T}$ (dashed)

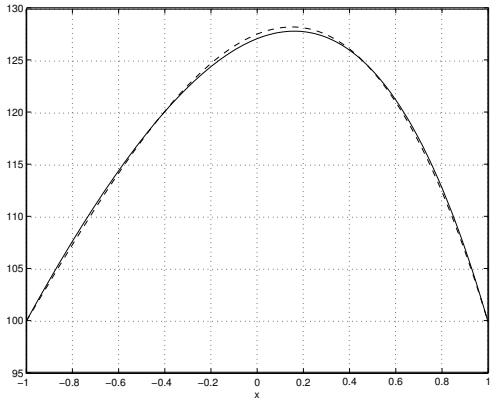


(b) Error, $\tilde{T} - T$

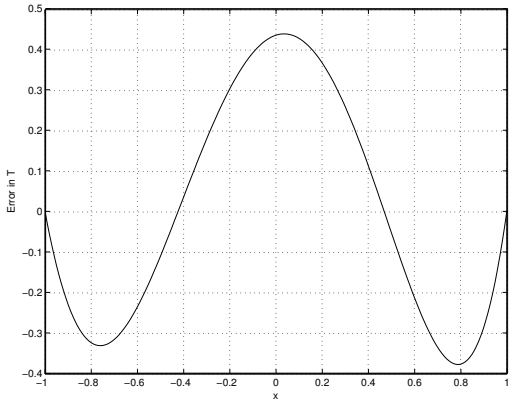


(c) Residual, $R(\tilde{T}, x)$

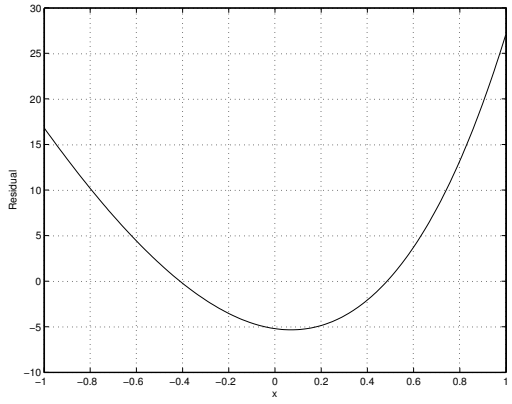
Figure 11.3: Results for collocation method.



(a) Comparison of T (solid) and $\tilde{T}$ (dashed)



(b) Error, $\tilde{T} - T$



(c) Residual, $R(\tilde{T}, x)$

Figure 11.4: Results for method of weight residuals.

Lecture 12

The Finite Element Method for One-Dimensional Diffusion

12.1 Motivation

In this lecture, we introduce the finite element method (FEM). In its most general form, FEM is based on the method of weighted residuals. The application of the method of weighted residuals as described in Lecture 11 becomes difficult when the complexity of the problems increases, specifically in multiple dimensions and with varying properties (e.g. varying thermal conductivity in the heat diffusion problem). The heart of the difficulty in these applications is the construction of the weighted residual integrals. For the simple one-dimensional problem discussed in Lecture 11, these integrals were relatively easy to do, however, the analogous integrals in multiple dimensions with complex geometries are very difficult to evaluate without some additional form of numerical approximation. For example, consider the heat transfer problem shown in Figure 12.1 and imagine the difficulty in constructing a set of functions which satisfy the boundary conditions and then integrating the weighted residuals of these functions over the entire domain.

The finite element method offers one approach to approximating the solution and the weighted residual integrals in general situations and, therefore, makes possible the approximation of complex physical problems. The basic idea behind the finite element method is to discretize the domain into small cells (called elements in FEM) and use these elements to approximate the solution and evaluate the weighted residuals (an example mesh can be seen in Figure 12.2). Typically, in each element, the solution is approximated using polynomial functions. Then, the weighted residuals are evaluated an element at a time and the resulting system of equations are solved to determine the weight coefficients on the polynomials in each element.

To introduce the basic concepts of the finite element method, we will first discuss the one-dimensional case. In future lectures, we will consider multiple dimensions and return to this complex heat transfer problem.

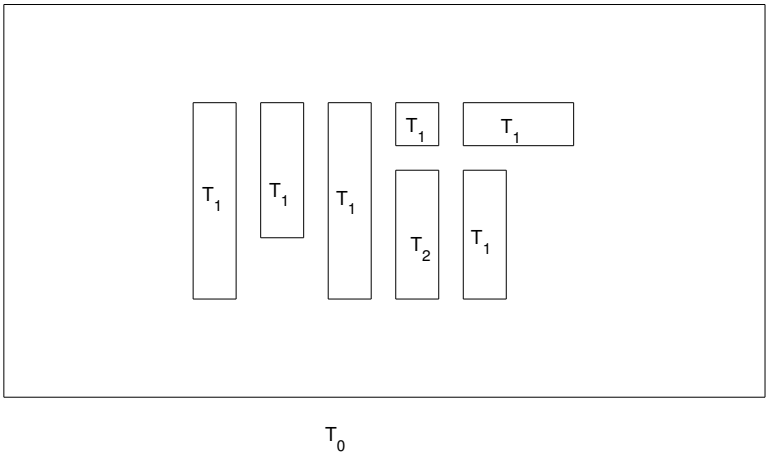


Figure 12.1: Heat transfer problem in a complex domain. Temperature at outer boundary is maintained at T_0. Temperature of all of the internal rectangles except one are maintained at T_1, while the remaining internal rectangle is maintained at T_2.

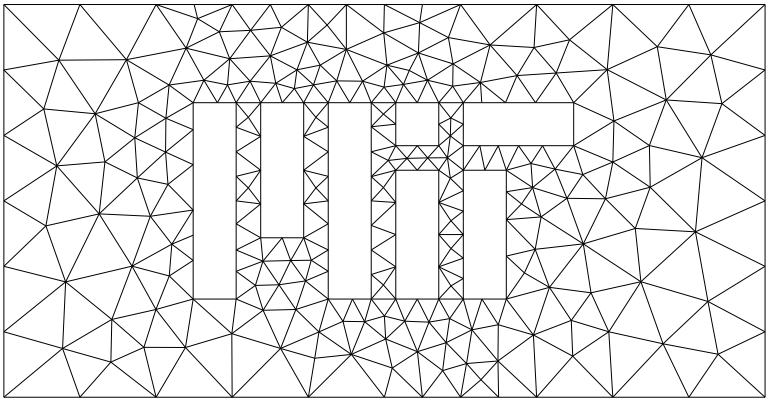


Figure 12.2: Mesh for heat transfer problem in a complex domain.

12.2 1-D Finite Element Mesh and Notation

Consider a mesh of one-dimensional elements as shown in Figure 12.3. As shown in the figure, element i is the region from $x_i \leq x \leq x_{i+1}$. Note, each element can have its own length,

$$\Delta x_i \equiv x_{i+1} - x_i.$$

12.3 1-D Linear Elements and the Nodal Basis

The finite element method typically uses polynomial functions inside each element. Furthermore, the approximation is usually required to be continuous from element to element. The simplest element which permits continuous functions would be to assume linear variations of x inside each element. This type of element is called a linear element (not too surprisingly).

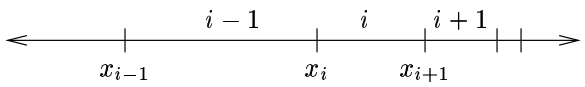


Figure 12.3: Mesh and notation for one-dimensional finite element method.

Using linear finite elements, a sample solution might look like that shown in Figure 12.4.

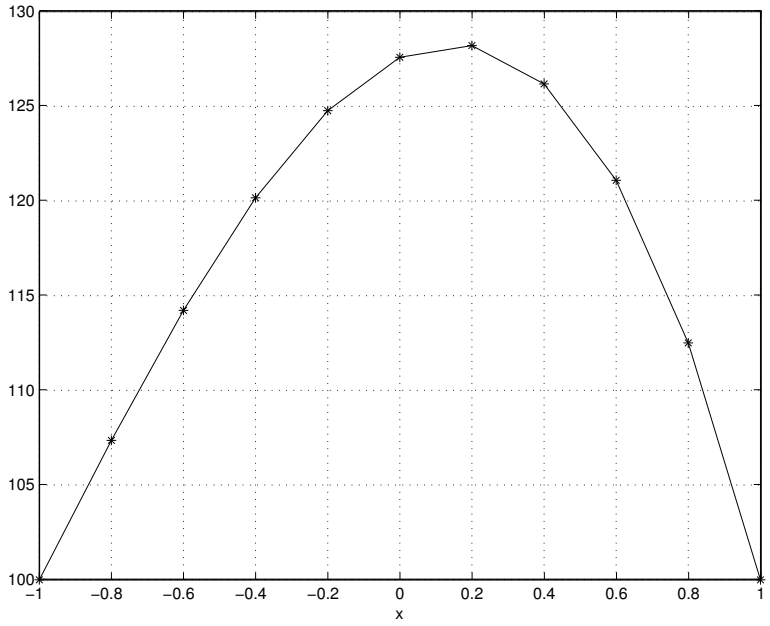


Figure 12.4: A linear element solution on a mesh with constant element size, $\Delta x_i = 0.2$.

A linear function can be described by two degrees of freedom. For example, a linear function within an element could be uniquely determined by the following combinations of information:

- the function values at the two endpoints (i.e. nodes) of the element,
- the function value at some point in the element and the function slope in the element.

For a mesh composed of N elements, a total of $N+1$ degrees of freedom exist to describe a continuous, piecewise linear function. If the approximation were allowed to be discontinuous from element to element, then there would be $2N$ degrees of freedom but continuity removes $N-1$ degrees of freedom (one per internal node). Thus, the degrees of freedom to describe a solution over a domain discretized into N linear elements could be (among other choices),

- the values of the function at the $N+1$ nodes,
- the value of the function at some point in the domain and the slopes in the N elements.

The first choice is the most common and leads to what is known as the nodal basis.

We now derive the nodal basis functions (i.e. the $\phi_i(x)$) for linear elements. As discussed above, for N linear elements we have $N+1$ degrees of freedom and therefore $N+1$ basis functions, i.e.,

$$\tilde{T}(x) = \sum_{i=1}^{N+1} a_i \phi_i(x), \tag{12.1}$$

where a_i are the $N+1$ degrees of freedom. For a nodal basis, we choose $a_i = \tilde{T}(x_i)$. Substituting this into Equation (12.1) gives,

$$\tilde{T}(x) = \sum_{i=1}^{N+1} \tilde{T}(x_i)\phi_i(x).$$

Now, evaluating this expansion at node j,

$$\begin{aligned} \tilde{T}(x_j) &= \sum_{i=1}^{N+1} \tilde{T}(x_i)\phi_i(x_j), \\ \Rightarrow 0 &= \sum_{i=1}^{j-1} \tilde{T}(x_i)\phi_i(x_j) + \tilde{T}(x_j)\left[\phi_j(x_j) - 1\right] + \sum_{i=j+1}^{N+1} \tilde{T}(x_i)\phi_i(x_j). \end{aligned}$$

Since this equation must be satisfied for any $\tilde{T}(x)$, the basis functions must satisfy,

$$\phi_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Finally, since the solutions vary linearly, this means that $\phi_i(x)$ is zero for $x < x_{i-1}$ and $x > x_{i+1}$, increases linearly from zero to one from x_{i-1} to x_i, and decreases linearly back to zero at x_{i+1}. $\phi_i(x)$ is shown in Figure 12.5. The, specific function is,

$$\phi_i(x) = \begin{cases} 0, & \text{for } x < x_{i-1}, \\ \frac{x - x_{i-1}}{\Delta x_{i-1}}, & \text{for } x_{i-1} < x < x_i, \\ \frac{x_{i+1} - x}{\Delta x_i}, & \text{for } x_i < x < x_{i+1}, \\ 0, & \text{for } x > x_{i+1}. \end{cases} \tag{12.2}$$

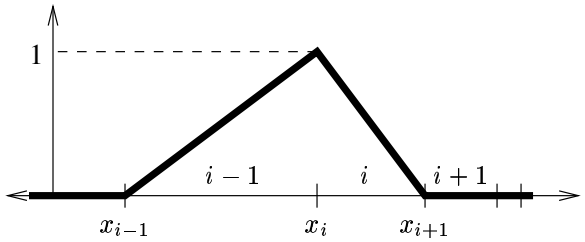


Figure 12.5: $\phi_i(x)$ for linear elements using a nodal basis.

12.4 1-D Diffusion Equation and Weighted Residual

In this lecture, we will concentrate on the one-dimensional, steady diffusion equation with a source term. As previously noted in Equation (11.1), the one-dimensional steady diffusion equation with a source term is,

$$(kT_x)_x = -q,$$

where $k(x)$ thermal conductivity of the material and $q(x)$ is the heat source (per unit area), respectively. Note that both k and q could be functions of x. Also, let the physical domain for the problem be from $x = -L/2$ to $x = L/2$.

The residual corresponding to this equation is,

$$R(\tilde{T}, x) \equiv \left(k\tilde{T}_x\right)_x + q. \tag{12.3}$$

The weighted residual is defined as,

$$\int_{-L/2}^{L/2} wR(\tilde{T}, x)\, dx = \int_{-L/2}^{L/2} w\left[\left(k\tilde{T}_x\right)_x + q\right],$$

where $w(x)$ is an arbitrary weight function. In the finite element method, the weighted residual statement is usually integrated by parts for diffusion problems so that the number of derivatives on the weight function and the dependent variable ($\tilde{T}$) are equal. Thus, performing integration by part gives the weighted residual as,

$$\left[w\, k\tilde{T}_x\right]_{-L/2}^{L/2} - \int_{-L/2}^{L/2} w_x\, k\tilde{T}_x\, dx + \int_{-L/2}^{L/2} wq\, dx.$$

The first term is on the boundaries of the domain and its use in setting boundary conditions is discussed in Section 12.6. The second and third terms are integrals over the entire domain.

Using the method of weighted residuals requires $N+1$ weight functions. We will consider Galerkin methods in which the weight functions are chosen to be equal to the solution basis functions. Thus, following the method of weighted residuals, we define the j^{th} weighted residual as,

$$R_j(\tilde{T}) \equiv \left[\phi_j\, k\tilde{T}_x\right]_{-L/2}^{L/2} - \int_{-L/2}^{L/2} \phi_{j_x}\, k\tilde{T}_x\, dx + \int_{-L/2}^{L/2} \phi_j q\, dx. \tag{12.4}$$

Example 12.1 (Evaluation of $\int_{-L/2}^{L/2} \phi_{j_x}\, k\tilde{T}_x\, dx$ for constant k) *While $\int_{-L/2}^{L/2} \phi_{j_x}\, k\tilde{T}_x\, dx$ is a global integral (i.e. over the entire domain), in reality $\phi_j(x)$ is non-zero only over the two elements that include node j,*

$$\Rightarrow \int_{-L/2}^{L/2} \phi_{j_x}\, k\tilde{T}_x\, dx = \int_{x_{j-1}}^{x_{j+1}} \phi_{j_x}\, k\tilde{T}_x\, dx.$$

The derivative of the basis functions is,

$$\phi_{i_x}(x) = \begin{cases} 0, & for\ x < x_{i-1}, \\ \frac{1}{\Delta x_{i-1}}, & for\ x_{i-1} < x < x_i, \\ \frac{-1}{\Delta x_i}, & for\ x_i < x < x_{i+1}, \\ 0, & for\ x > x_{i+1}. \end{cases} \tag{12.5}$$

Thus,

$$\int_{x_{j-1}}^{x_{j+1}} \phi_{j_x}\, k\tilde{T}_x\, dx = \frac{1}{\Delta x_{j-1}} \int_{x_{j-1}}^{x_j} k\tilde{T}_x\, dx - \frac{1}{\Delta x_j} \int_{x_j}^{x_{j+1}} k\tilde{T}_x\, dx.$$

Next, taking the derivative of $\tilde{T}$*,*

$$\tilde{T}_x(x) = \sum_{i=1}^{N+1} a_i \phi_{i_x}(x),$$

For the integral in element $j-1$*, the only non-zero contributions to* $\tilde{T}_x$ *are from* ϕ_{j-1} *and* ϕ_j*, specifically,*

$$\textit{In element } j-1: \qquad \tilde{T}_x = a_{j-1}\phi_{j-1_x} + a_j \phi_{j_x} = \frac{a_j - a_{j-1}}{\Delta x_{j-1}}.$$

Similarly,

$$\textit{In element } j: \qquad \tilde{T}_x = a_j \phi_{j_x} + a_{j+1}\phi_{j+1_x} = \frac{a_{j+1} - a_j}{\Delta x_j}.$$

Substituting these expressions for the derivatives into the integral gives,

$$\int_{x_{j-1}}^{x_{j+1}} \phi_{j_x}\, k\tilde{T}_x\, dx = \frac{a_j - a_{j-1}}{(\Delta x_{j-1})^2} \int_{x_{j-1}}^{x_j} k\, dx - \frac{a_{j+1} - a_j}{(\Delta x_j)^2} \int_{x_j}^{x_{j+1}} k\, dx.$$

At this point, we must integrate $k(x)$ *in each element. Efficient numerical methods to approximate this integral are discussed in Section 12.5. For the situation in which* k *is constant throughout the domain, then the integral reduces to,*

$$\textit{For } k = \textit{constant}: \qquad \int_{x_{j-1}}^{x_{j+1}} \phi_{j_x}\, k\tilde{T}_x\, dx = k\frac{a_j - a_{j-1}}{\Delta x_{j-1}} - k\frac{a_{j+1} - a_j}{\Delta x_j}.$$

This result should look somewhat familiar (hint: what does this formula reduce to when the element size is constant, $\Delta x_{j-1} = \Delta x_j$*).*

Example 12.2 (An FEM example for 1-D diffusion) *The Matlab implementation of the finite element method for the problem described in Example 11.1 is shown below. Note, at the bottom of the script the exact solution and the error in the finite element solution are calculated and plotted. Interestingly, the FEM results for linear elements are exact at the nodes. However, in between the nodes (i.e. within the elements), there is error since a linear function is being used to represent a higher-order (curved) solution. The error, i.e.* $\tilde{T}(x) - T(x)$*, is shown in Figure 12.6(c) for both* $N = 5$ *and* $N = 10$ *solutions. A clear factor of four reduction is observed with the increased grid resolution leading to the conclusion that the method is second order accurate. Note: to construct this plot, each element was subdivided into 20 points and the FEM and exact solution were calculated at these points and compared.*

```
% FEM solver for d2T/dx2 + q = 0 where q = 50 exp(x)
%
```

```
% BC's: T(-1) = 100 and T(1) = 100.
%
% Note: the finite element degrees of freedom are
%       stored in the vector, v.

% Number of elements
Ne = 5;
x = linspace(-1,1,Ne+1);

% Zero stiffness matrix
K = zeros(Ne+1, Ne+1);
b = zeros(Ne+1, 1);

% Loop over all elements and calculate stiffness and residuals
for ii = 1:Ne,

  kn1 = ii;
  kn2 = ii+1;

  x1 = x(kn1);
  x2 = x(kn2);

  dx    = x2 - x1;

  % Add contribution to kn1 weighted residual due to kn1 function
  K(kn1, kn1) = K(kn1, kn1) - (1/dx);

  % Add contribution to kn1 weighted residual due to kn2 function
  K(kn1, kn2) = K(kn1, kn2) + (1/dx);

  % Add contribution to kn2 weighted residual due to kn1 function
  K(kn2, kn1) = K(kn2, kn1) + (1/dx);

  % Add contribution to kn2 weighted residual due to kn2 function
  K(kn2, kn2) = K(kn2, kn2) - (1/dx);

  % Add forcing term to kn1 weighted residual
  b(kn1) = b(kn1) - (50*(exp(x2)-x2*exp(x1) + x1*exp(x1) - exp(x1))/dx);

  % Add forcing term to kn2 weighted residual
  b(kn2) = b(kn2) - (50*(x2*exp(x2)-exp(x2)-x1*exp(x2)+exp(x1))/dx);

end
```

```
% Set Dirichlet conditions at x=-1
kn1 = 1;
K(kn1,:)    = zeros(size(1,Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Set Dirichlet conditions at x=1
kn1 = Ne+1;
K(kn1,:)    = zeros(size(1,Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Solve for solution
v = K\b;


% Plot solution
figure(1);
plot(x,v,'*-');
xlabel('x');
ylabel('T');

% For the exact solution, we need to use finer spacing to plot
% it correctly.  If we only plot it at the nodes of the FEM mesh,
% the exact solution would also look linear between the nodes.  To
% make sure there is always enough resolution relative to the FEM
% nodes, the size of the vector for plotting the exact solution is
% set to be 20 times the number of FEM nodes.
Npt = 20*Ne+1;
xe = linspace(-1,1,Npt);
Te = -50*exp(xe) + 50*xe*sinh(1) + 100 + 50*cosh(1);
hold on; plot(xe,Te); hold off;

% Plot the error.  To do this, calculate the error on the same
% set of points in which the exact solution was plot.  This
% requires that the location of the point xx(i) be found in the
% FEM mesh to construct the true solution at this point by linearly
% interpolating between the two nodes of the FEM mesh.

verr(1) = v(1) - Te(1);
h = x(2)-x(1);
```

```
for i = 2:Npt-1,
  xxi = xe(i);
  Ti  = Te(i);
  j = floor((xxi-xe(1))/h) + 1;
  x0 = x(j);
  x1 = x(j+1);
  v0 = v(j);
  v1 = v(j+1);
  xi = 2*(xxi - x0)/(x1-x0)-1;  % This gives xi between +/-1
  vi = 0.5*(1-xi)*v0 + 0.5*(1+xi)*v1;
  verr(i) = vi - Ti;
end
verr(Npt) = v(Ne+1) - Te(Npt);

figure(2);
plot(xe,verr);
xlabel('x');
ylabel('Error');
```

In-class Discussion 12.1 (Implementation of 1-D FEM) *In class, we will discuss the details of the implementation of the FEM into a computer program using the above Matlab script.*

12.5 Gaussian Quadrature

The finite element method requires the calculation of integrals over individual elements, for example,

$$\int_{x_j}^{x_{j+1}} \phi_{j_x} k\tilde{T}_x \, dx, \quad \text{or} \quad \int_{x_j}^{x_{j+1}} \phi_j \, q \, dx.$$

While in some settings these integrals can be calculated analytically, often they are too difficult. In this situation, numerical integration methods are used.

Gaussian quadrature is one of the most commonly applied numerical integration methods to this task. Gaussian integration approximates an integral as the weighted sum of the values of its integrand,

$$\int_{-1}^{+1} g(\xi) \, d\xi = \sum_{i=1}^{N_q} \alpha_i g(\xi_i), \tag{12.6}$$

where N_q is the number of quadrature points (i.e. integrand evaluations) and α_i and ξ_i are the weights and locations of integrand evaluations. Note that Gaussian quadrature rules are developed for specific integration limits, in this case between $\xi = -1$ to $+1$. Thus, for integration in an element, we will need to transform from x to ξ. See Example 12.3 for more on applying Gauss quadrature to finite element methods.

Gaussian quadrature integration rules are determined by requiring exact integration of polynomial integrands, i.e.,

$$g(\xi) = c_0 + c_1\xi + c_2\xi^2 + c_3\xi^3 + \cdots + c_M\xi^M,$$

for all values of c_i. Note that,

$$\int_{-1}^{+1} \xi^i \, d\xi = \begin{cases} 0 & \text{if } i = \text{odd} \\ \frac{2}{i+1} & \text{if } i = \text{even} \end{cases}$$

$$\Rightarrow \int_{-1}^{+1} \sum_{i=1}^{M} c_i\xi^i \, d\xi = 2\left(c_0 + \frac{1}{3}c_2 + \frac{1}{5}c_4 + \cdots\right). \tag{12.7}$$

12.5.1 $N_q = 1$ Quadrature Rule

For $N_q = 1$, the quadrature rule is,

$$\int_{-1}^{+1} g(\xi) \, d\xi \approx \alpha_1 g(\xi_1).$$

Now, using Equation (12.7), we determine the highest order polynomial that is integrable by a single quadrature point,

$$\begin{aligned} 2\left(c_0 + \frac{1}{3}c_2 + \frac{1}{5}c_4 + \cdots\right) &= \alpha_1 g(\xi_1) \\ &= \alpha_1\left(c_0 + c_1\xi_1 + c_2\xi_1^2 + c_3\xi_1^3 + \cdots + c_M\xi_1^M\right). \end{aligned}$$

Matching term by term gives,

$$\begin{aligned} c_0 &: 2 = \alpha_1 \qquad \Rightarrow \alpha_1 = 2, \\ c_1 &: 0 = \alpha_1\xi_1 \qquad \Rightarrow \xi_1 = 0. \end{aligned}$$

Then, checking the c_2 term shows that it is not integrating exactly. So, with one point, a linear polynomial is the highest order polynomial that can be evaluated exactly and the quadrature rule is,

$$\int_{-1}^{+1} g(\xi) \, d\xi \approx \alpha_1 g(\xi_1), \qquad \alpha_1 = 2, \quad \xi_1 = 0.$$

12.5.2 $N_q = 2$ Quadrature Rule

For $N_q = 2$, the quadrature rule is,

$$\int_{-1}^{+1} g(\xi) \, d\xi \approx \alpha_1 g(\xi_1) + \alpha_2 g(\xi_2).$$

Again using Equation (12.7), we determine the highest order polynomial that is integrable by two quadrature points,

$$\begin{aligned} 2\left(c_0 + \frac{1}{3}c_2 + \frac{1}{5}c_4 + \cdots\right) &= \alpha_1 g(\xi_1) \\ &= \alpha_1\left(c_0 + c_1\xi_1 + c_2\xi_1^2 + c_3\xi_1^3 + \cdots + c_M\xi_1^M\right) + \\ &\quad \alpha_2\left(c_0 + c_1\xi_2 + c_2\xi_2^2 + c_3\xi_2^3 + \cdots + c_M\xi_2^M\right). \end{aligned}$$

Matching the first four terms gives the following constraints,

$$\begin{aligned} c_0 &: 2 = \alpha_1 + \alpha_2, \\ c_1 &: 0 = \alpha_1\xi_1 + \alpha_2\xi_2, \\ c_2 &: \frac{2}{3} = \alpha_1\xi_1^2 + \alpha_2\xi_2^2, \\ c_3 &: 0 = \alpha_1\xi_1^3 + \alpha_2\xi_2^3. \end{aligned}$$

These constraints can be meet with,

$$\alpha_1 = \alpha_2 = 1, \quad \xi_1 = -\frac{1}{\sqrt{3}}, \quad \xi_2 = \frac{1}{\sqrt{3}}.$$

Thus, this rule will integrate cubic polynomials exactly.

Example 12.3 (Calculation of Forcing Integral with Gauss Quadrature) *We wish to apply Gaussian quadrature to evaluate the forcing integral,*

$$\int_{x_j}^{x_{j+1}} \phi_j \, q \, dx. \tag{12.8}$$

To do this, we first transform between the x and ξ space. For this integral in element j, the transformation would be,

$$x(\xi) = x_j + \frac{1}{2}(1+\xi)(x_{j+1} - x_j), \quad \Rightarrow dx = \frac{1}{2}(x_{j+1} - x_j)d\xi.$$

Substitution in the forcing integral gives,

$$\int_{x_j}^{x_{j+1}} \phi_j \, q \, dx = \int_{-1}^{+1} \frac{1}{2}(x_{j+1} - x_j)\phi_j \, q \, d\xi, \qquad \Rightarrow \qquad g(\xi) = \frac{1}{2}(x_{j+1} - x_j)\, \phi_j[x(\xi)]\, q[x(\xi)].$$

Note that the dependence of ϕ_j and q on ξ is shown through the dependence of these functions on $x = x(\xi)$. However, for the basis functions, it is often easier to directly determine ϕ_j from ξ. For example, in the case of linear polynomial basis functions, the basis functions with element j can be written as,

$$\begin{aligned} \phi_1(\xi) &= \frac{1}{2}(1-\xi), \\ \phi_2(\xi) &= \frac{1}{2}(1+\xi). \end{aligned}$$

Clearly, these functions vary linearly with ξ. $\phi_1(\xi)$ is one at $\xi = -1$ and decreases linearly to zero at $\xi = +1$. And, vice-versa for $\phi_2(\xi)$. For the integral, we are considering in this example, $\phi_j(x)$ is equivalent to $\phi_1(\xi)$.

The following is a Matlab script that uses Gaussian quadrature to evaluate the forcing integral and solve the problem described in Example 11.1. The number of points being used is set at the beginning of the script. Results for both 1-point and 2-point quadrature are shown in Figures 12.7 and 12.8 for 5 *and* 10 *elements. While the 2-point quadrature rule has lower error than the 1-point rule, both appear to be second-order accurate since the errors reduce by nearly a factor of* 4 *for the factor 2 change in mesh size. Also, the results are no longer exact at the nodes like they were in Example 12.2 (though the 2-point quadrature rules are quite close).*

```
% FEM solver for d2T/dx2 + q = 0 where q = 50 exp(x)
%
% BC's: T(-1) = 100 and T(1) = 100.
%
% Gaussian quadrature is used in evaluating the forcing integral.
%
% Note: the finite element degrees of freedom are
%       stored in the vector, v.

% Number of elements
Ne = 5;
x = linspace(-1,1,Ne+1);

% Set quadrature rule
Nq = 2;
if (Nq == 1),
  alphaq(1) = 2.0; xiq(1) = 0.0;
elseif (Nq == 2),
  alphaq(1) = 1.0; xiq(1) = -1/sqrt(3);
  alphaq(2) = 1.0; xiq(2) =  1/sqrt(3);
else
  fprintf('Error: Unknown quadrature rule (Nq = %i)\n',Nq);
  return;
end

% Zero stiffness matrix
K = zeros(Ne+1, Ne+1);
b = zeros(Ne+1, 1);

% Loop over all elements and calculate stiffness and residuals
for ii = 1:Ne,
```

```
kn1 = ii;
kn2 = ii+1;

x1 = x(kn1);
x2 = x(kn2);

dx    = x2 - x1;

% Add contribution to kn1 weighted residual due to kn1 function
K(kn1, kn1) = K(kn1, kn1) - (1/dx);

% Add contribution to kn1 weighted residual due to kn2 function
K(kn1, kn2) = K(kn1, kn2) + (1/dx);

% Add contribution to kn2 weighted residual due to kn1 function
K(kn2, kn1) = K(kn2, kn1) + (1/dx);

% Add contribution to kn2 weighted residual due to kn2 function
K(kn2, kn2) = K(kn2, kn2) - (1/dx);

% Evaluate forcing term using quadrature
for nn = 1:Nq,

  % Get xi location of quadrature point
  xi = xiq(nn);

  % Calculate x location of quadrature point
  xq = x1 + 0.5*(1+xi)*dx;

  % Calculate q
  qq = 50*exp(xq);

  % Calculate phi1 and phi2
  phi1 = 0.5*(1-xi);
  phi2 = 0.5*(1+xi);

  % Subtract forcing term to kn1 weighted residual
  b(kn1) = b(kn1) - alphaq(nn)*0.5*phi1*qq*dx;

  % Add forcing term to kn2 weighted residual
  b(kn2) = b(kn2) - alphaq(nn)*0.5*phi2*qq*dx;

end
```

```
end


% Set Dirichlet conditions at x=-1
kn1 = 1;
K(kn1,:)    = zeros(size(1,Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Set Dirichlet conditions at x=1
kn1 = Ne+1;
K(kn1,:)    = zeros(size(1,Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Solve for solution
v = K\b;


% Plot solution
figure(1);
plot(x,v,'*-');
xlabel('x');
ylabel('T');

% For the exact solution, we need to use finer spacing to plot
% it correctly.  If we only plot it at the nodes of the FEM mesh,
% the exact solution would also look linear between the nodes.  To
% make sure there is always enough resolution relative to the FEM
% nodes, the size of the vector for plotting the exact solution is
% set to be 20 times the number of FEM nodes.
Npt = 20*Ne+1;
xe = linspace(-1,1,Npt);
Te = -50*exp(xe) + 50*xe*sinh(1) + 100 + 50*cosh(1);
hold on; plot(xe,Te); hold off;

% Plot the error.  To do this, calculate the error on the same
% set of points in which the exact solution was plot.  This
% requires that the location of the point xx(i) be found in the
% FEM mesh to construct the true solution at this point by linearly
% interpolating between the two nodes of the FEM mesh.
```

```
verr(1) = v(1) - Te(1);
h = x(2)-x(1);
for i = 2:Npt-1,
  xxi = xe(i);
  Ti  = Te(i);
  j = floor((xxi-xe(1))/h) + 1;
  x0 = x(j);
  x1 = x(j+1);
  v0 = v(j);
  v1 = v(j+1);
  xi = 2*(xxi - x0)/(x1-x0)-1;  % This gives xi between +/-1
  vi = 0.5*(1-xi)*v0 + 0.5*(1+xi)*v1;
  verr(i) = vi - Ti;
end
verr(Npt) = v(Ne+1) - Te(Npt);

figure(2);
plot(xe,verr);
xlabel('x');
ylabel('Error');
```

12.6 Boundary Conditions

Boundary conditions generally fall into one of three types:

- Set $\tilde{T}$ at the boundary (known as a Dirichlet boundary condition). For heat transfer problems, this type of boundary condition occurs when the temperature is known at some portion of the boundary.
- Set $\tilde{T}_x$ at the boundary (known as a Neumann boundary condition). For heat transfer problems, this type of boundary condition occurs when the heat transfer rate is known at the boundary. For example, an adiabatic boundary would require that $\tilde{T}_x = 0$.
- Set $\alpha_0\tilde{T} + \alpha_1\tilde{T}_x$ at the boundary (known as a Robin boundary condition) where α_0 and α_1 do not depend on the temperature. For heat transfer problems, this type of boundary condition occurs when modeling convection (see Example 12.4).

Example 12.4 (Convection boundary condition for heat transfer) *Consider the flow of air over a solid object (see Figure 12.9) with the velocity and temperature of the air being, U_{ext} and T_{ext}, respectively. Aligning the x direction into the surface, then the heat transfer rate at the surface is,*

$$Q_{wall} = -kT_x, \tag{12.9}$$

where $T(x)$ is the temperature inside the solid and k is the thermal conductivity of the solid. A common approach to modeling the heat transfer into the solid as a result of the airflow is based on specifying a heat transfer coefficient for the airflow, h_{ext}, which is related to the heat transfer rate by,

$$h_{ext} \equiv \frac{Q_{wall}}{T_{ext} - T_{wall}}. \tag{12.10}$$

Note that h_{ext} is generally a function of the external velocity and other flow properties. Combining Equations (12.9) and (12.10) gives the following boundary condition at the surface of the solid,

$$-kT_x = h_{ext}\left(T_{ext} - T_{wall}\right), \tag{12.11}$$

where T_{wall} is the temperature of the solid at its surface. This equation can be re-arranged into the Robin boundary condition form,

$$-kT_x + h_{ext}T_{wall} = h_{ext}T_{ext}.$$

12.6.1 Implementation of Dirichlet Boundary Conditions

To demonstrate the implementation of a Dirichlet boundary condition, suppose that the value of the temperature is known at $x = -L/2$, specifically,

$$T(-L/2) = T_{left}.$$

This condition is set by forcing the corresponding nodal degree of freedom to be the desired value. At $x = -L/2$, the corresponding nodal degree of freedom would be a_1 (the value of the temperature at the first node), thus, the boundary condition is implemented as,

$$a_1 = T_{left}.$$

12.6.2 Implementation of Neumann Boundary Conditions

To demonstrate the implementation of a Neumann boundary condition, suppose that the heat transfer rate were known at $x = L/2$, specifically,

$$-kT_x(L/2) = Q_{right}.$$

This condition is enforced through the weighted residual for the last node, $j = N + 1$. Specifically,

$$R_{N+1} = \left[\phi_{N+1}\, k\tilde{T}_x\right]_{-L/2}^{L/2} - \int_{-L/2}^{L/2} \phi_{N+1_x}\, k\tilde{T}_x\, dx + \int_{-L/2}^{L/2} \phi_{N+1} q\, dx.$$

Because $\phi_{N+1}(x)$ is zero except in the last element, this weighted residual reduces to,

$$R_{N+1} = \left(\phi_{N+1} k\tilde{T}_x\right)\Big|_{x=x_{N+1}} - \int_{x_N}^{x_{N+1}} \phi_{N+1_x}\, k\tilde{T}_x\, dx + \int_{x_N}^{x_{N+1}} \phi_{N+1} q\, dx.$$

The two integral terms are calculated in the standard manner. The first term is where the Neumann boundary condition is set through substitution of $-k\tilde{T}_x = Q_{right}$. Specifically, the weighted residual becomes,

$$R_{N+1} = -\phi_{N+1}(x_{N+1})Q_{right} - \int_{x_N}^{x_{N+1}} \phi_{N+1_x}\, k\tilde{T}_x\, dx + \int_{x_N}^{x_{N+1}} \phi_{N+1} q\, dx.$$

Note that the boundary term, $-\phi_{N+1}(x_{N+1})Q_{right}$ does not depend on the temperature and thus this boundary condition does not impact the stiffness matrix.

12.6.3 Implementation of Robin Boundary Conditions

To demonstrate the implementation of a Robin boundary condition, suppose that a convective heat transfer boundary condition were to be set at $x = L/2$, specifically,

$$-kT_x(L/2) = h_{ext}\left[T_{ext} - T(L/2)\right].$$

For more on convective boundary conditions, see Example 12.4. Following the basic process outlined in the Neumann boundary condition, the weighted residual for $j = N+1$ is,

$$R_{N+1} = \left(\phi_{N+1} k\tilde{T}_x\right)\Big|_{x=x_{N+1}} - \int_{x_N}^{x_{N+1}} \phi_{N+1_x}\, k\tilde{T}_x\, dx + \int_{x_N}^{x_{N+1}} \phi_{N+1} q\, dx.$$

Substituting $-k\tilde{T}_x = h_{ext}\left[T_{ext} - \tilde{T}(x_{N+1})\right]$ in the boundary term gives,

$$R_{N+1} = -\phi_{N+1}(x_{N+1}) h_{ext}\left[T_{ext} - \tilde{T}(x_{N+1})\right] - \int_{x_N}^{x_{N+1}} \phi_{N+1_x}\, k\tilde{T}_x\, dx + \int_{x_N}^{x_{N+1}} \phi_{N+1} q\, dx.$$

As opposed to the Neumann boundary condition, the Robin boundary condition implementation does introduce a new dependence on the solution, specifically on $\tilde{T}(x_{N+1})$. This will cause a change in the stiffness matrix. Furthermore, the T_{ext} term will alter the right-hand side vector in the FEM numerical implementation.

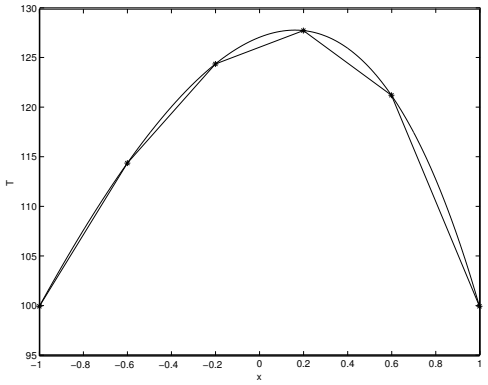
In-class Discussion 12.2 (Implementation of boundary conditions) *In class, we will discuss the details of the implementation of the boundary conditions into a computer program using the following Matlab script.*

```
% FEM solver for k d2T/dx2 + q = 0 where q = 50 exp(x)
%
% Thermal conductivity is set to one, k=1.
%
% BC's:
%
% At x=-1:   T(-1) = 100   (Dirichlet)
%
% At x=1, two options exist:
```

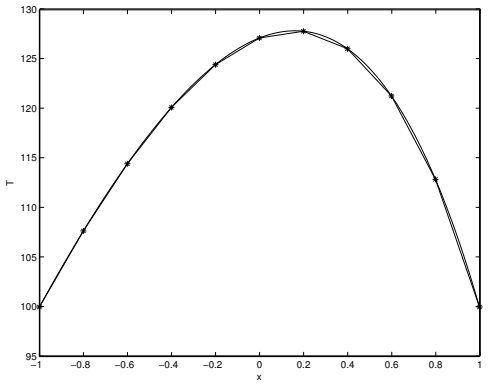
```
%
%            Specified heat transfer: dT/dX = Qright
%
%            Convection: -k dT/dx = hext * (Text - T(1))
%
% The choice of which bc to use is made through RightBC.
% If RightBC = 0, heat transfer rate is specified.  Otherwise,
% a convection BC is applied.
%
% Gaussian quadrature is used in evaluating the forcing integral.
%
% Note: the finite element degrees of freedom are
%       stored in the vector, v.

clear all;

% Number of elements
Ne = 5;
x = linspace(-1,1,Ne+1);

% Set RightBC info
RightBC = 0;
if (RightBC == 0),
  Qright = 0;
else,
  hext = 10000;
  Text = 100;
end

% Set quadrature rule
Nq = 2;
if (Nq == 1),
  alphaq(1) = 2.0; xiq(1) = 0.0;
elseif (Nq == 2),
  alphaq(1) = 1.0; xiq(1) = -1/sqrt(3);
  alphaq(2) = 1.0; xiq(2) =  1/sqrt(3);
else
  fprintf('Error: Unknown quadrature rule (Nq = %i)\n',Nq);
  return;
end

% Zero stiffness matrix
K = zeros(Ne+1, Ne+1);
b = zeros(Ne+1, 1);
```

```
% Loop over all elements and calculate stiffness and residuals
for ii = 1:Ne,

  kn1 = ii;
  kn2 = ii+1;

  x1 = x(kn1);
  x2 = x(kn2);

  dx = x2 - x1;

  % Add contribution to kn1 weighted residual due to kn1 function
  K(kn1, kn1) = K(kn1, kn1) - (1/dx);

  % Add contribution to kn1 weighted residual due to kn2 function
  K(kn1, kn2) = K(kn1, kn2) + (1/dx);

  % Add contribution to kn2 weighted residual due to kn1 function
  K(kn2, kn1) = K(kn2, kn1) + (1/dx);

  % Add contribution to kn2 weighted residual due to kn2 function
  K(kn2, kn2) = K(kn2, kn2) - (1/dx);

  % Evaluate forcing term using quadrature
  for nn = 1:Nq,

    % Get xi location of quadrature point
    xi = xiq(nn);

    % Calculate x location of quadrature point
    xq = x1 + 0.5*(1+xi)*dx;

    % Calculate q
    qq = 50*exp(xq);

    % Calculate phi1 and phi2
    phi1 = 0.5*(1-xi);
    phi2 = 0.5*(1+xi);

    % Subtract forcing term to kn1 weighted residual
    b(kn1) = b(kn1) - alphaq(nn)*0.5*phi1*qq*dx;

    % Add forcing term to kn2 weighted residual
```

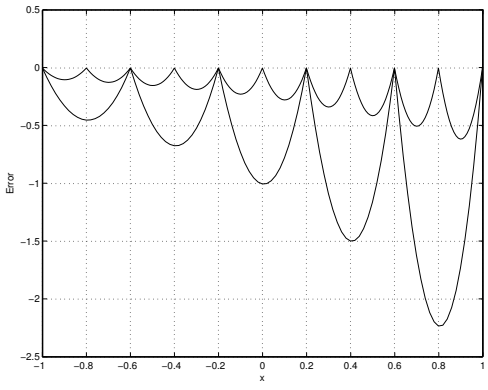
```
    b(kn2) = b(kn2) - alphaq(nn)*0.5*phi2*qq*dx;

  end

end


% Set Dirichlet conditions at x=-1
kn1 = 1;
K(kn1,:)    = zeros(size(1,Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Set boundary condition at x=1
kn1 = Ne+1;
if (RightBC == 0), % Specify heat transfer rate (Neumann)

  b(kn1)     = b(kn1) + Qright;

else, % Convective

  K(kn1,kn1) = K(kn1,kn1) + hext;
  b(kn1)     = b(kn1) + hext*Text;

end


% Solve for solution
v = K\b;


% Plot solution
plot(x,v,'*-');
xlabel('x');
ylabel('T');
```



(a) $N = 5$ elements

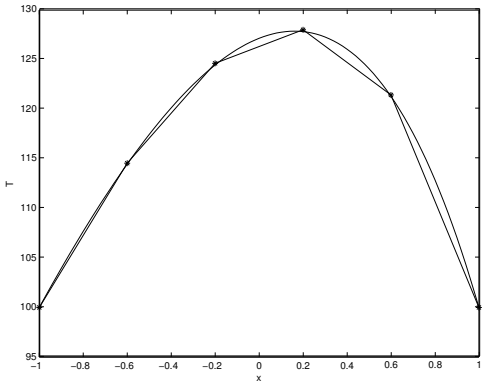


(b) $N = 10$ elements

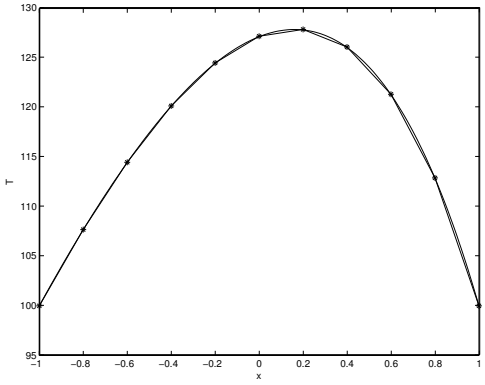


(c) Error

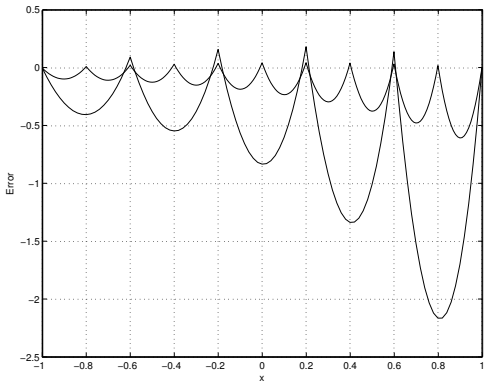
Figure 12.6: Comparison of finite element solution to exact solution.



(a) $N = 5$ elements

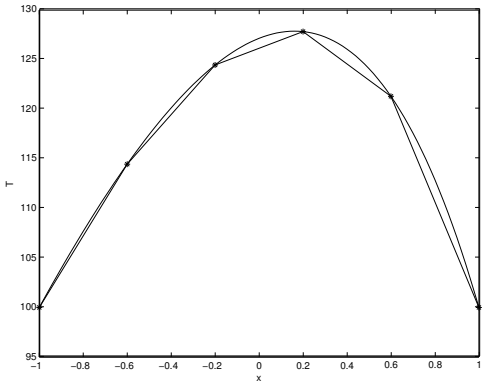


(b) $N = 10$ elements

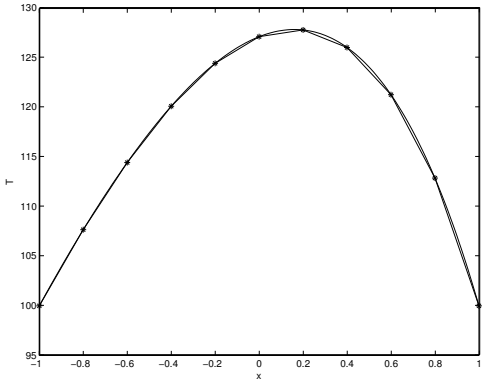


(c) Error

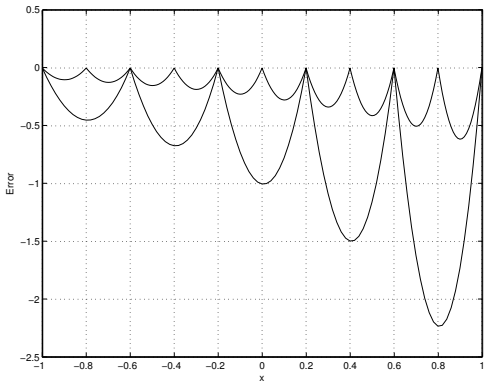
Figure 12.7: Comparison of finite element solution using $N_q = 1$ point Gaussian quadrature to exact solution.



(a) $N = 5$ elements



(b) $N = 10$ elements



(c) Error

Figure 12.8: Comparison of finite element solution using $N_q = 2$ point Gaussian quadrature to exact solution.

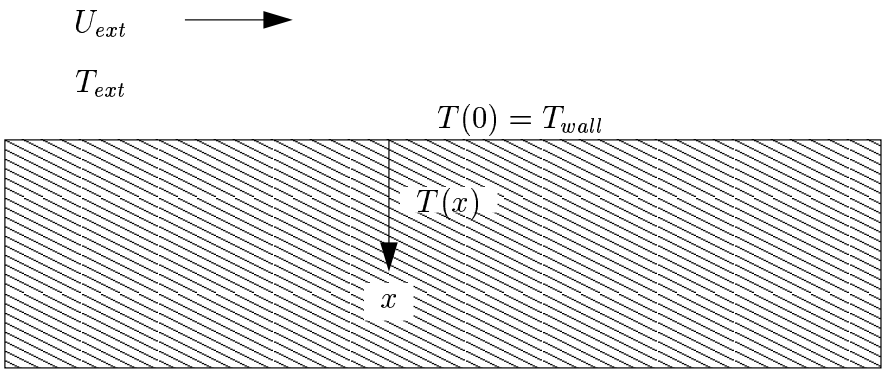


Figure 12.9: Convection above a wall of a flow with velocity U_{ext} and temperature T_{ext}. The temperature of the wall at the surface is T_{wall} (the surface is at $x = 0$).

Lecture 13

The Finite Element Method for Two-Dimensional Diffusion

In this lecture, we will consider the finite element approximation of the two-dimensional diffusion problem,

$$\nabla \cdot (k \nabla T) + q = 0. \tag{13.1}$$

As in the previous discussion of the method of weighted residuals and the finite element method, the approximate solution will have the form,

$$\tilde{T}(x, y) = \sum_{i=1}^{N} a_i \phi_i(x, y),$$

where $\phi_i(x, y)$ are the known basis functions and the a_i are the unknown weights to be determine for the specific problem. Following the Galerkin method of weighted residuals, we will weight Equation (13.1) by one of the basis functions and integrate the diffusion term by parts to give the following weighted residual,

$$R_j \equiv \int_{\delta\Omega} \phi_j \, k \nabla \tilde{T} \cdot \vec{n} \, ds - \int_{\Omega} \nabla \phi_j \cdot \left(k \nabla \tilde{T} \right) \, dA + \int_{\Omega} \phi_j q \, dA = 0. \tag{13.2}$$

13.1 Reference Element and Linear Elements

In multiple dimensions, a common practice in defining the polynomial functions within an element is to transform each element into a canonical, or so-called 'reference' element. Figure 13.1 shows the mapping commonly used for triangular elements which maps a generic triangle in (x, y) into a right triangle in (ξ_1, ξ_2).

In the reference element space, the nodal basis for linear polynomials will be one at one of the nodes, and reduce linearly to zero at the other nodes. These functions are,

$$\phi_1(\xi_1, \xi_2) = 1 - \xi_1 - \xi_2, \tag{13.3}$$

$$\phi_2(\xi_1, \xi_2) = \xi_1, \tag{13.4}$$

$$\phi_3(\xi_1, \xi_2) = \xi_2. \tag{13.5}$$

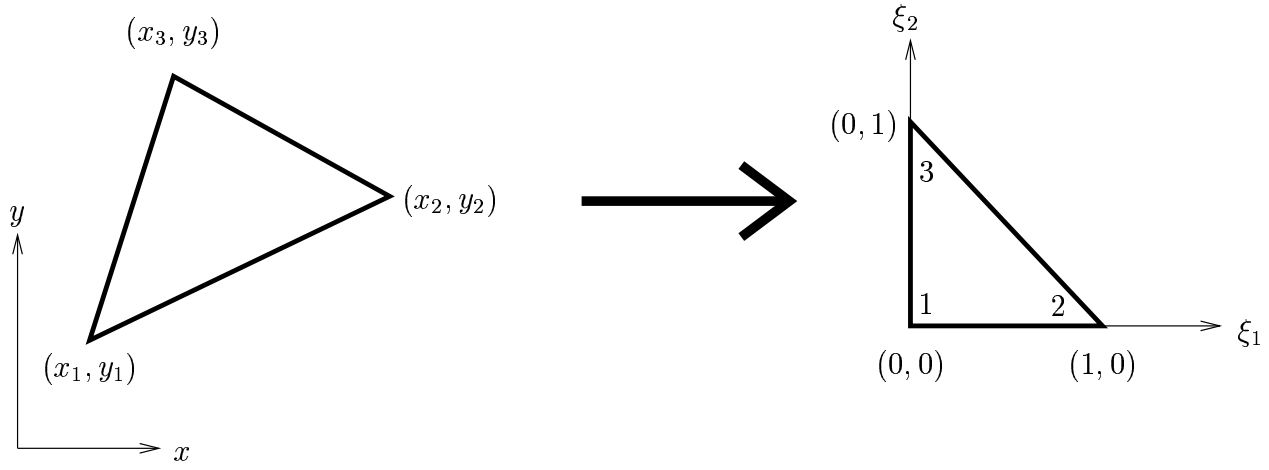


Figure 13.1: Transformation of a generic triangular element in (x, y) into the reference element in (ξ_1, ξ_2).

Then, within the element, the solution $\tilde{T}$ in the (ξ_1, ξ_2) space is the combination of these three basis functions multiplied by the corresponding nodal weights,

$$\tilde{T}(\xi_1, \xi_2) = \sum_{i=1}^{3} a_i \phi_i(\xi_1, \xi_2). \tag{13.6}$$

Using Equation (13.6), the value of $\tilde{T}$ can be found at any (ξ_1, ξ_2). To find the (x, y) locations in terms of the (ξ_1, ξ_2), we can expand them using the nodal locations and the nodal basis functions, i.e.,

$$\vec{x}(\xi_1, \xi_2) = \sum_{i=1}^{3} \vec{x}_i \phi_i(\xi_1, \xi_2).$$

Since the $\phi_i(\xi_1, \xi_2)$ are linear functions of ξ_1 and ξ_2, this amounts to a linear transformation between (ξ_1, ξ_2) and (x, y). Specifically, substituting the nodal basis functions gives,

$$\begin{aligned} \vec{x}(\xi_1, \xi_2) &= \vec{x}_1 (1 - \xi_1 - \xi_2) + \vec{x}_2 \xi_1 + \vec{x}_3 \xi_2, \\ \Rightarrow \vec{x}(\xi_1, \xi_2) &= \vec{x}_1 + (\vec{x}_2 - \vec{x}_1) \xi_1 + (\vec{x}_3 - \vec{x}_1) \xi_2. \end{aligned}$$

This can be written in a matrix notation as,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} \tag{13.7}$$

This equation can be inverted to also determine (ξ_1, ξ_2) as a function of (x, y),

$$\begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}^{-1} \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} \tag{13.8}$$

13.2 Differentiation using the Reference Element

To find the derivative of $\tilde{T}$ with respect to x (or similarly y) within an element, we differentiate the three nodal basis functions within the element,

$$\begin{aligned}\tilde{T}_x &= \frac{\partial}{\partial x}\left(\sum_{i=1}^{3} a_i \phi_i\right), \\ &= \sum_{i=1}^{3} a_i \frac{\partial \phi_i}{\partial x}.\end{aligned}$$

To find the x-derivatives of each of the ϕ_i's, the chain rule is applied,

$$\frac{\partial \phi_i}{\partial x} = \frac{\partial \phi_i}{\partial \xi_1}\frac{\partial \xi_1}{\partial x} + \frac{\partial \phi_i}{\partial \xi_2}\frac{\partial \xi_2}{\partial x}.$$

Similarly, to find the y-derivatives, the derivatives with respect to y,

$$\frac{\partial \phi_i}{\partial y} = \frac{\partial \phi_i}{\partial \xi_1}\frac{\partial \xi_1}{\partial y} + \frac{\partial \phi_i}{\partial \xi_2}\frac{\partial \xi_2}{\partial y}.$$

The calculation of the derivatives of ϕ_i with respect to the ξ's gives,

$$\begin{aligned}\frac{\partial \phi_1}{\partial \xi_1} &= -1, & \frac{\partial \phi_1}{\partial \xi_2} &= -1, \\ \frac{\partial \phi_2}{\partial \xi_1} &= 1, & \frac{\partial \phi_2}{\partial \xi_2} &= 0, \\ \frac{\partial \phi_3}{\partial \xi_1} &= 0, & \frac{\partial \phi_3}{\partial \xi_2} &= 1.\end{aligned}$$

The only remaining terms are the calculation of $\frac{\partial \xi_1}{\partial x}$, $\frac{\partial \xi_2}{\partial x}$, etc. which can be found by differentiating Equation (13.8),

$$\begin{aligned}\frac{\partial \vec{\xi}}{\partial \vec{x}} &= \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}^{-1}, \\ &= \frac{1}{J}\begin{pmatrix} y_3 - y_1 & -(x_3 - x_1) \\ -(y_2 - y_1) & x_2 - x_1 \end{pmatrix},\end{aligned}$$

where

$$J = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1).$$

Note that the Jacobian, J, is equal to twice the area of the triangular element.

13.3 Construction of the Stiffness Matrix

The stiffness matrix arises in the calculation of $\int_\Omega \nabla \phi_j \cdot \left(k \nabla \tilde{T}\right)\, dA$. As in the one-dimensional case, the j-th row of the stiffness matrix K corresponds to the weighted residual of ϕ_j. The

i-th column in the j-th row corresponds to the dependence of the j-th weighted residual on a_i. Further drawing on the one-dimensional example, the weighted residuals are assembled by calculating the contribution to all of the residuals from within a single element. In the two-dimensioal linear element situation, three weighted residuals are impacted by a given element, specifically, the weighted residuals corresponding to the nodal basis functions of the three nodes of the triangle. For example, in each element we must calculate,

$$\int_{\Omega_e} \nabla\phi_1 \cdot \left(k\nabla\tilde{T}\right)\, dA, \qquad \int_{\Omega_e} \nabla\phi_2 \cdot \left(k\nabla\tilde{T}\right)\, dA, \qquad \int_{\Omega_e} \nabla\phi_3 \cdot \left(k\nabla\tilde{T}\right)\, dA,$$

where Ω_e is spatial domain for a specific element. As described in Section 13.2, the gradient of $\tilde{T}$ can be written,

$$\nabla\tilde{T}(x,y) = \sum_{i=1}^{3} a_i \nabla\phi_i(x,y),$$

thus the weighted residuals expand to,

$$\int_{\Omega} \nabla\phi_j \cdot \left(k\nabla\tilde{T}\right)\, dA = \sum_{i=1}^{3} a_i K_{j,i}, \qquad \text{where} \qquad K_{j,i} \equiv \int_{\Omega} \nabla\phi_j \cdot (k\nabla\phi_i)\, dA.$$

For the situation in which k is constant, and linear elements are used, then this reduces to,

$$K_{j,i} \equiv k\nabla\phi_j \cdot \nabla\phi_i A_e$$

where A_e is the area of element e.

13.4 Integration in the Reference Element

The reference element can also be used to evaluate integrals. For example, consider the evaluation of the forcing function integral within an element,

$$\int_{\delta\Omega_k} w(\vec{x})\, q(\vec{x})\, dA.$$

In transforming the integral from (x,y) to (ξ_1,ξ_2), the differential area of integration must be transform using the following result,

$$dA = dx\, dy = J d\xi_1\, d\xi_2 = J\, dA_\xi. \tag{13.9}$$

Thus, the integrals can now be evaluated in reference element space,

$$\int_{\Omega_\xi} w(\vec{x}(\vec{\xi}))\, q(\vec{x}(\vec{\xi}))\, J dA_\xi.$$

In-class Discussion 13.1 (Calculation of the Mass Matrix)

Lecture 14

Higher-order Finite Elements

14.1 Nodal Basis for Higher Order Elements

Until now, we have considered solutions which were allowed to vary at most linearly across an element. We now consider higher-order functions within the element. For simplicity, we will restrict our attention to one-dimensional problems. A p-th order polynomial has $p+1$ degrees of freedom, i.e. the coefficients of each term,

$$\tilde{T}(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_p x^p.$$

Thus, a general basis for a p-th order polynomial will require $p+1$ basis functions within an element,

$$\tilde{T}(x) = \sum_{i=1}^{p+1} a_i \phi_i(x) \quad \text{in an element.}$$

Building on the nodal basis approach described in Section 12.3 for linear elements, a common approach to choosing a basis for higher-order elements is to insert nodes within an element in addition to the nodes at the boundaries of the elements, specifically $p-1$ additional nodes internal to the element.

Let's consider building a nodal basis for quadratic elements. In this case, one additional node is added and this node is placed at the midpoint of the element. Using the one-dimensional reference element which extends from $-1 \le \xi \le 1$, this places nodes at $\xi = -1$, 0, and 1. The unknowns are assumed to be the values at these nodes,

$$a_1 = \tilde{T}(-1), \qquad a_2 = \tilde{T}(0), \qquad a_3 = \tilde{T}(1).$$

which leads to the following constraints on $\phi_i(\xi)$'s,

$$\phi_1(-1) = 1, \quad \phi_1(0) = 0, \quad \phi_1(1) = 0.$$

$$\phi_2(-1) = 0, \quad \phi_2(0) = 1, \quad \phi_2(1) = 0.$$

$$\phi_3(-1) = 0, \quad \phi_3(0) = 0, \quad \phi_3(1) = 1.$$

Applying these constraints and solving for the quadratic $\phi_i(\xi)$ gives,

$$\begin{aligned}\phi_1(\xi) &= -\frac{1}{2}\xi(1-\xi),\\ \phi_2(\xi) &= (1-\xi)(1+\xi),\\ \phi_3(\xi) &= \frac{1}{2}\xi(1+\xi).\end{aligned}$$

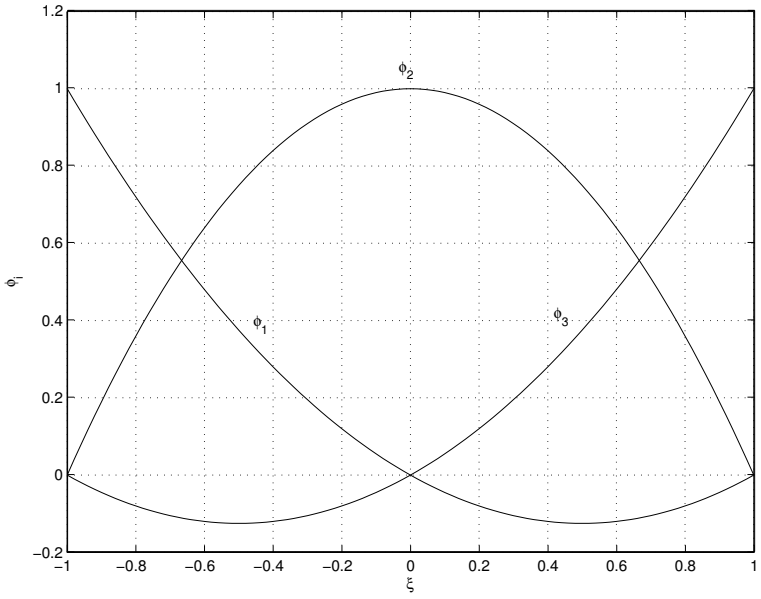


Figure 14.1: Nodal basis functions for a quadratic element with equally-spaced nodes.

In-class Discussion 14.1 (Construction of global $\phi(x)$ for quadratic elements)

14.2 Implementation of higher-order FEM

The implementation details are essentially the same as in the linear element case. Shown below is a Matlab script for a quadratic FEM using a nodal basis.

```
% FEM solver for d2T/dx2 + q = 0 where q = 50 exp(x)
%
% BC's: T(-1) = 100 and T(1) = 100.
%
% Note: the finite element degrees of freedom are
%       stored in the vector, v.

% Set number of Gauss points (only used for forcing term in this example)
NGf = 3;
if (NGf == 3),
  xiGf = [-sqrt(3/5);          0; sqrt(3/5)];
  aGf  = [       5/9;        8/9;       5/9];
elseif (NGf == 2),
  xiGf = [-1/sqrt(3); +1/sqrt(3)];
  aGf  = [       1.0;        1.0];
else,
  NGf = 1;
  xiGf = [0.0];
  aGf  = [2.0];
end

% Number of elements
Ne = 10;
x = linspace(-1,1,Ne+1);

% Zero stiffness matrix
K = zeros(2*Ne+1, 2*Ne+1);
b = zeros(2*Ne+1, 1);

% Loop over all elements and calculate stiffness and residuals
for ii = 1:Ne,

  kn1 = 1 + 2*(ii-1);
  kn2 = 2 + 2*(ii-1);
  kn3 = 3 + 2*(ii-1);

  x1 = x(ii);
  x3 = x(ii+1);

  dx    = x3 - x1;
  dxidx = 2/dx;
  dxdxi = 1/dxidx;
```

```
% Add contribution to kn1 weighted residual
K(kn1, kn1) = K(kn1, kn1) - dxidx*( 7/6);
K(kn1, kn2) = K(kn1, kn2) - dxidx*(-4/3);
K(kn1, kn3) = K(kn1, kn3) - dxidx*( 1/6);

% Add contribution to kn2 weighted residual
K(kn2, kn1) = K(kn2, kn1) - dxidx*(-4/3);
K(kn2, kn2) = K(kn2, kn2) - dxidx*( 8/3);
K(kn2, kn3) = K(kn2, kn3) - dxidx*(-4/3);

% Add contribution to kn3 weighted residual
K(kn3, kn1) = K(kn3, kn1) - dxidx*( 1/6);
K(kn3, kn2) = K(kn3, kn2) - dxidx*(-4/3);
K(kn3, kn3) = K(kn3, kn3) - dxidx*( 7/6);

% Use Gaussian quadrature to evaluate forcing term integral
for nn = 1:NGf,

  % Get xi for Gauss point
  xiG = xiGf(nn);

  % Find N1, N2 and N3 (i.e. weighting/intepolants) at xiG
  N1 = -0.5*xiG*(1-xiG);
  N2 =  (1-xiG)*(1+xiG);
  N3 =  0.5*xiG*(1+xiG);

  % Find x for Gauss point
  xG = 0.5*(1-xiG)*x1 + 0.5*(1+xiG)*x3;

  % Find f for Gauss point
  fG = -50*exp(xG);

  % Evaluate integrand at Gauss point for weight functions at nodes
  gG1 = N1*fG*dxdxi;
  gG2 = N2*fG*dxdxi;
  gG3 = N3*fG*dxdxi;

  % Send to correct right-hand side term
  b(kn1) = b(kn1) + aGf(nn)*gG1;
  b(kn2) = b(kn2) + aGf(nn)*gG2;
  b(kn3) = b(kn3) + aGf(nn)*gG3;

end
```

```
end


% Set Dirichlet conditions at x=0
kn1 = 1;
K(kn1,:)    = zeros(size(1,2*Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Set Dirichlet conditions at x=1
kn1 = 2*Ne+1;
K(kn1,:)    = zeros(size(1,2*Ne+1));
K(kn1, kn1) = 1.0;
b(kn1)      = 100.0;


% Solve for solution
v = K\b;


% Plot it and compare.  Note: since even the finite element
% solution varies more than linearly across an element, we need to
% subdivide each element, evaluate the basis functions, and plot
% the FEM solution to see the higher order variations.

Nplot = 20; % Number of points per element to plot
nnn = 0;
for ii = 1:Ne,

  kn1 = 1 + 2*(ii-1);
  kn2 = 2 + 2*(ii-1);
  kn3 = 3 + 2*(ii-1);

  x1 = x(ii);
  x3 = x(ii+1);

  v1 = v(kn1);
  v2 = v(kn2);
  v3 = v(kn3);

  for nn = 1:Nplot,

    % Get xi for plot point
```

```
    xiG = -1 + 2*(nn-1)/(Nplot-1);

    % Find N1, N2 and N3 (i.e. weighting/intepolants) at xiG
    N1 = -0.5*xiG*(1-xiG);
    N2 =  (1-xiG)*(1+xiG);
    N3 =  0.5*xiG*(1+xiG);

    % Find x and v for plot point
    xG = 0.5*(1-xiG)*x1 + 0.5*(1+xiG)*x3;
    vG = v1*N1 + v2*N2 + v3*N3;

    nnn = nnn + 1;
    xp(nnn) = xG;
    vp(nnn) = vG;
    up(nnn) = -50*exp(xG) + 50*xG*sinh(1) + 100 + 50*cosh(1);

  end
end

figure(1);
plot(xp,vp,'r');hold on;
plot(xp,up); hold off;
xlabel('x');
ylabel('u');

figure(2);
plot(xp,vp-up);
xlabel('x');
ylabel('Error');
```

In-class Discussion 14.2 (Behavior of Quadratic FEM) *The results in Figures 14.2 and 14.3 will be discussed in class.*

14.3 Hierarchical Basis for Quadratic Elements

In this section, we consider a different basis for quadratic polynomials. Since we want the solution to be continuous from element-to-element, we will still specify that a_1 and a_3 are the values at the end of the elements (i.e. at the nodes),

$$a_1 = \tilde{T}(-1), \qquad a_3 = \tilde{T}(1).$$

However, we will no longer associate a_2 with the midpoint value of the temperature. Instead, let the additional constraint be that a_2 is the value of the second derivative in the middle of

the reference element, specifically,

$$a_2 = \tilde{T}_{\xi\xi}(0).$$

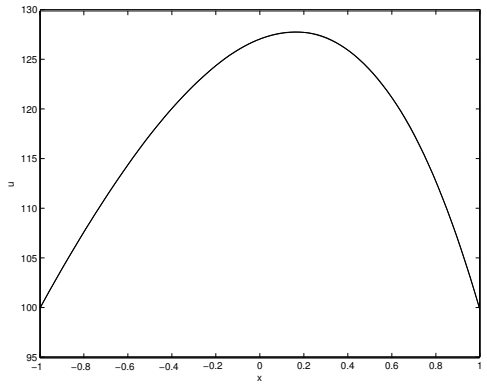
These three constraints lead to the following conditions on the $\phi_i(\xi)$:

$$\begin{aligned}
&\phi_1(-1) = 1, \quad \phi_{1\xi\xi}(0) = 0, \quad \phi_1(1) = 0. \\
&\phi_2(-1) = 0, \quad \phi_{2\xi\xi}(0) = 1, \quad \phi_2(1) = 0. \\
&\phi_3(-1) = 0, \quad \phi_{3\xi\xi}(0) = 0, \quad \phi_3(1) = 1.
\end{aligned}$$

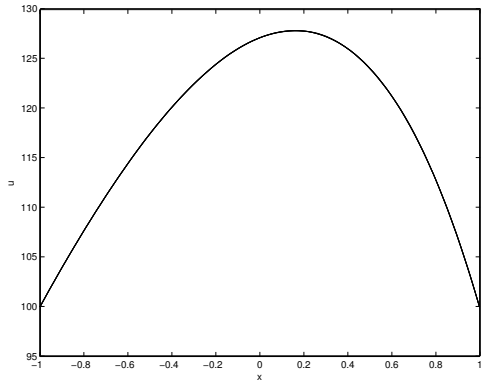
Applying these constraints and solving for the quadratic $\phi_i(\xi)$ gives,

$$\begin{aligned}
\phi_1(\xi) &= \frac{1}{2}(1-\xi), \\
\phi_2(\xi) &= \frac{1}{2}\left(\xi^2 - 1\right), \\
\phi_3(\xi) &= \frac{1}{2}(1+\xi).
\end{aligned}$$

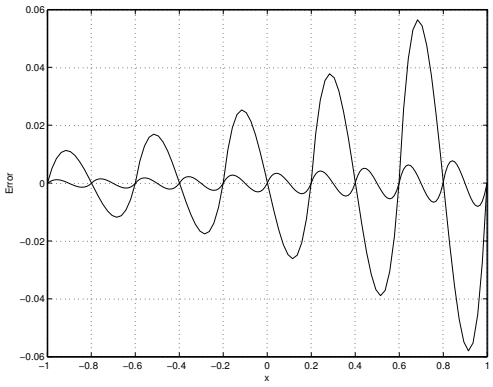
This basis is known as a hierarchical basis because the quadratic basis functions are the usual linear basis functions (ϕ_1 and ϕ_3) with an additional function (ϕ_2) that brings the quadratic contribution into the approximate solution. In other words, the basis is hierarchical because the basis for a linear-varying solution is a subset of the basis for the quadratic solution. The plots of ϕ_1, ϕ_2, and ϕ_3 are shown in Figure 14.4.



(a) $N = 5$ elements

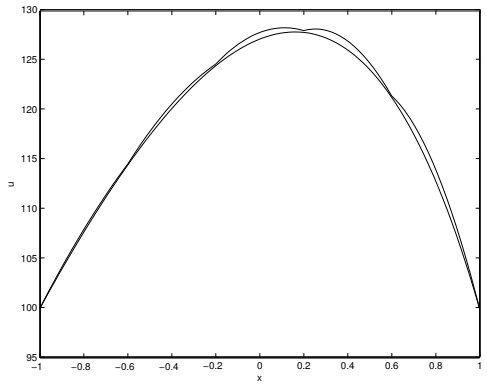


(b) $N = 10$ elements

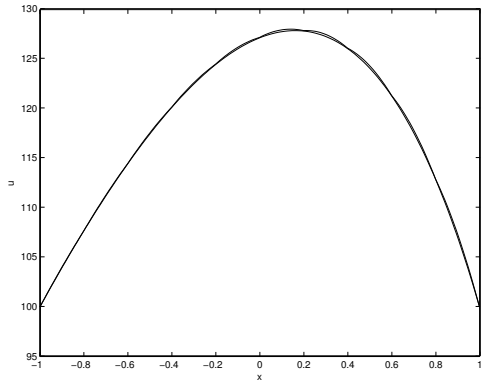


(c) Error

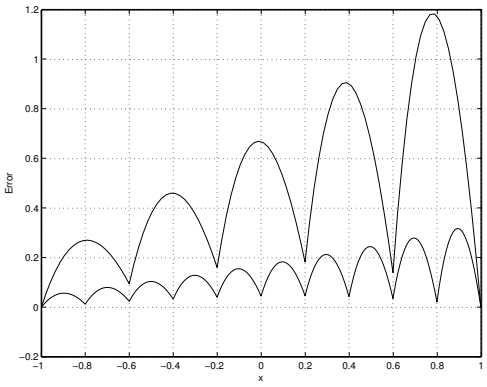
Figure 14.2: Comparison of quadratic finite element solution using 3 point Gaussian quadrature on forcing function.



(a) $N = 5$ elements



(b) $N = 10$ elements



(c) Error

Figure 14.3: Comparison of quadratic finite element solution using 1 point Gaussian quadrature on forcing function.

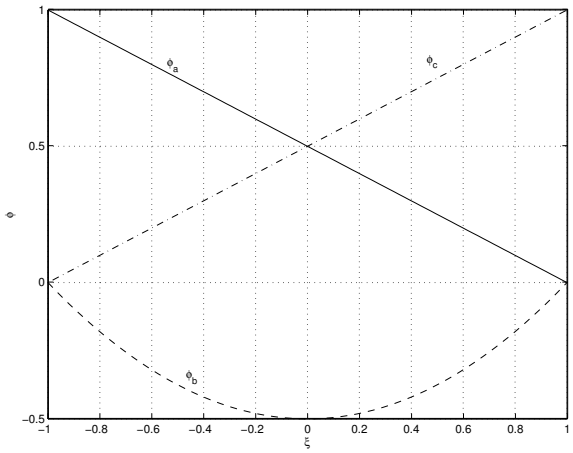
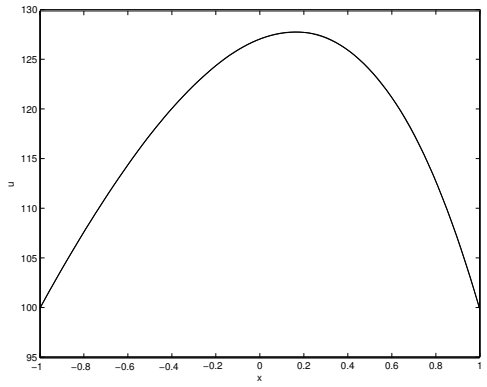
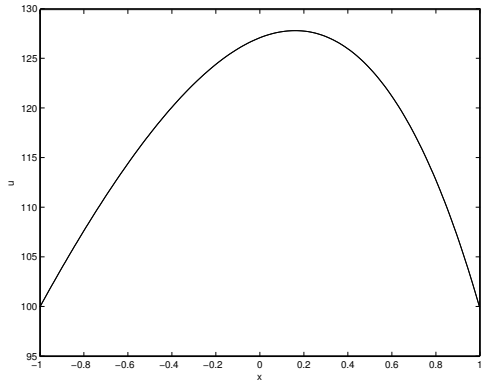


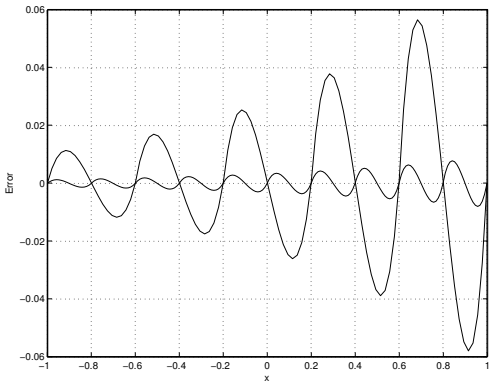
Figure 14.4: Plots of quadratic hierarchical basis functions.



(a) $N = 5$ elements



(b) $N = 10$ elements



(c) Error

Figure 14.5: Comparison of quadratic finite element solution with hierarchical basis using 3 point Gaussian quadrature on forcing function.

Lecture 15

Introduction to the Monte Carlo Method

In this lecture, we begin our exploration of probabilistic methods, i.e. numerical methods which are used to quantify the impact of uncertainty. In particular, we will focus on the Monte Carlo method as it is the most common probabilistic method and is the foundation for many others.

To make our discussion concrete, we will consider a simplified model for the heat transfer through a cooled turbine blade as shown in Figure 15.1. A one-dimensional model of the heat transfer along the dashed line is,

$$\dot{q} = h_{gas}\left(T_{gas} - T_{TBC}\right), \tag{15.1}$$

$$\dot{q} = \frac{k_{TBC}}{L_{TBC}}\left(T_{TBC} - T_{mh}\right), \tag{15.2}$$

$$\dot{q} = \frac{k_m}{L_m}\left(T_{mh} - T_{mc}\right), \tag{15.3}$$

$$\dot{q} = h_{cool}\left(T_{mc} - T_{cool}\right). \tag{15.4}$$

Then, given the values of h_{gas}, T_{gas}, k_{TBC}, L_{TBC}, k_m, L_m, h_{cool}, and T_{cool}, we can solve these

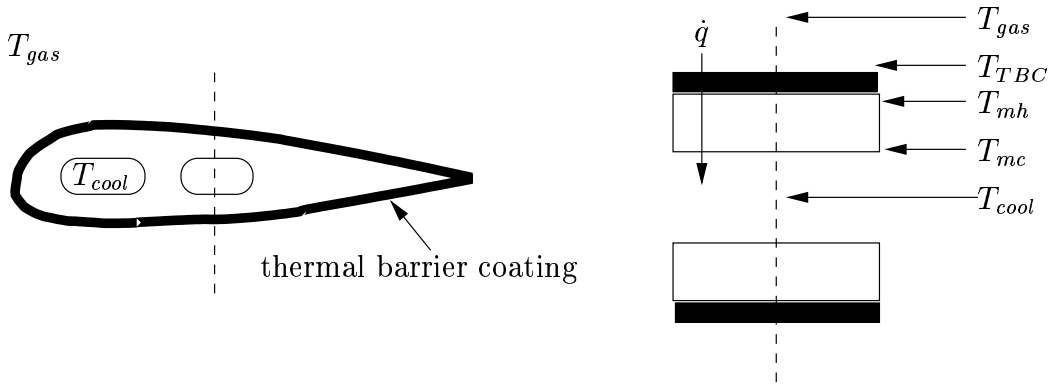


Figure 15.1: Turbine blade heat transfer example

four equations to determine, T_{TBC}, T_{mh}, T_{mc}, and $\dot{q}$. In the design of cooled turbine blades, a key parameter is the hot-side metal temperature, T_{mh}, because as this temperature increases, the durability (usable life) of a blade decreases. Thus, the goal of the heat transfer design is to maintain the metal temperatures at an acceptablly low value while minimizing cost.

Example 15.1 (Nominal Analysis of Turbine Blade Heat Transfer) *A typical, deterministic analysis of the turbine blade heat transfer problem would assume that all of the input parameters are at their nominal, i.e. design-intent, values. Suppose the design-intent values were,*

$$h_{gas} = 3000\,W/(m^2K), \qquad h_{cool} = 1000\,W/(m^2K),$$
$$T_{gas} = 1500\,K, \qquad T_{cool} = 600\,K,$$
$$k_{TBC} = 1\,W/(mK), \qquad k_m = 20\,W/(mK),$$
$$L_{TBC} = 0.0005\,m, \qquad L_m = 0.003\,m.$$

The results of this design-intent analysis give,

$$T_{TBC} = 1348.7\,K, \quad T_{mh} = 1121.8\,K, \quad T_{mc} = 1053.8\,K, \quad \dot{q} = 453780\,W/m^2.$$

Due to manufacturing variability, the parameters of the actual manufactured blades are not exactly the design intent values but rather are distributed. For example, due to the difficulty of applying the thermal barrier coating on the outside of the blades, the thickness of the thermal barrier coating is variable. The role of probabilistic methods is to quantify the impact of this type of variability on properties of interest (e.g. the hot-side metal temperature). The results of the probabilistic analysis can take many forms depending on the specific application. In the example of the turbine blade where the hot-side metal temperature is critical, the following information might be desired from a probabilistic analysis:

- The distribution of T_{mh} that would be observed in the population of manufactured blades.
- The probability that T_{mh} is above some critical value (indicating the blade's life will be unacceptable low).
- Instead of determining the entire distribution of T_{mh}, sometimes knowing the mean value, $\mu_{T_{mh}}$, is sufficient.
- To have some indication of the variability of T_{mh} without requiring accurate estimation of the entire distribution, the standard deviation, $\sigma_{T_{mh}}$, can be used.

The Monte Carlo method is based on the idea of taking a small, randomly-drawn sample from a population and estimating the desired outputs from this sample. For the outputs described above, this would involve:

- Replacing the distribution of T_{mh} that would be observed over the entire population of manufactured blades with the distribution (i.e. histogram) of T_{mh} observed in the random sample.

- Replacing the probability that T_{mh} is above a critical value for the entire population of manufactured blades with the fraction of blades in the random sample that have T_{mh} greater than the critical value.
- Replacing the mean value of T_{mh} for the entire population with the mean value of the random sample.
- Replacing the standard deviation of T_{mh} for the entire population with the standard deviation of the random sample.

Since this exactly what is done in the field of statistics, the analysis of the Monte Carlo method is a direct application of statistics.

In summary, the Monte Carlo method involves essentially three steps:

1. Generate a random sample of the input parameters according to the (assumed) distributions of the inputs.
2. Analyze (deterministically) each set of inputs in the sample.
3. Estimate the desired probabilistic outputs, and the uncertainty in these outputs, using the random sample.

15.1 Monte Carlo Method for Uniform Distributions

To demonstrate the Monte Carlo method in more detail, let's consider the specific case where the thermal barrier coating in the previous turbine blade example is known to be uniformly distributed from $0.00025\,m < L_{TBC} < 0.00075\,m$ as shown from the probability distribution function (PDF) of L_{TBC} in Figure 15.2.

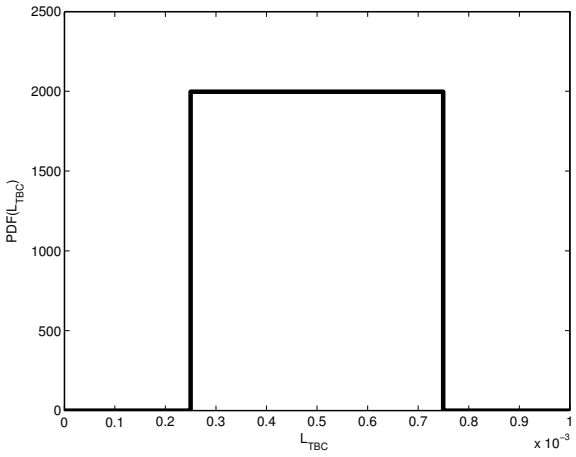


Figure 15.2: Probability distribution function (PDF) of L_{TBC} uniformly-distributed from $0.00025\,m < L_{TBC} < 0.00075\,m$.

The first step is to generate a random sample of L_{TBC}. The basic approach relies on the ability to generate random numbers which are uniformly distributed from 0 to 1. This type of

functionality exists within many different scientific programming environments or languages. In Matlab, the command **rand** provides this capability. Then, using the uniform distribution from 0 to 1, a uniform distribution of L_{TBC} over the desired range can be created,

$$\mathbf{L}_{TBC} = 0.00025 + 0.0005\mathbf{u},$$

where $\mathbf{u}$ is a random variable uniformly distributed from 0 to 1. This approach is used to create the samples shown as histograms in Figure 15.3 for samples of size $N = 100$, 1000, and 10000. For the smaller sample size (specifically $N = 100$), the fact that the sample was drawn from a uniform distribution is not readily apparent. However, as the number of samples increases, the uniform distribution becomes more evident. Clearly, the sample size will have a direct impact on the accuracy of the probabilistic estimates in the Monte Carlo method.

The following is a Matlab script that implements the Monte Carlo method for this uniform distribution of L_{TBC}. The distributions of T_{mh} shown in Figure 15.4 correspond to the L_{TBC} distributions shown in Figure 15.3 and were generated with this script.

```
clear all;

% Nominal values of input parameters

hgas  = 3000;    % TBC-gas heat transfer coef. (W/(m^2 K))
Tgas  = 1500;    % Mixed gas temperature (K)
ktbc  = 1;       % TBC   thermal conduct. (W/mK)
km    = 20;      % Metal thermal conduct. (W/mK)
Lm    = 0.003;   % Metal thickness (m)
hcool = 1000;    % Coolant-metal heat transfer coef. (W/(m^2 K))
Tcool =  600;    % Coolant temperature (K)

% Number of Monte Carlo trials
N = 100;
Ltbc = zeros(N,1);
Tmh  = zeros(N,1);

for n = 1:N,

  % generate Ltbc values using a uniform distribution

  Ltbc(n) = 0.00025 + 0.0005*rand;

  % Solve heat transfer problem

  [Ttbc, Tmh(n), Tmc, q] = blade1D(hgas, Tgas, ...
   ktbc, Ltbc(n), ...
   km, Lm, ...
```

```
   hcool, Tcool);

end

figure(1);
hist(Ltbc,20);
xlabel('L_{tbc} (m)');
figure(2);
hist(Tmh,20);
xlabel('T_{mh} (K)');
```

15.2 Monte Carlo Method for Non-Uniform Distributions

Input variability can be distributed in many ways beyond the simple uniform distribution considered above. In this section, we discuss a common approach used to implement the Monte Carlo method for non-uniform distributions. However, we note that for many of the most common distribution types, random number generators widely available. For example, in Matlab, the function **randn** returns random numbers that are normally distributed with a mean of zero and a standard deviation of one. In Matlab's Statistics Toolbox, many other distribution types are available (see the documentation for the **random** function for details).

In these notes, we will discuss the inversion method for generating random numbers with non-uniform distributions. While other methods exist for generating random numbers, they are often based on the inversion method. The basic principle of the inversion method is to utilize the inverse of the cumulative distribution function (CDF) to transform a uniform distribution to a desired distribution. Recall that the CDF is defined as the integral of the PDF,

$$F(x) = \int_{-\infty}^{x} f(\xi)\, d\xi,$$

and that the CDF is related to probability by,

$$F(x) \equiv P\left\{\mathbf{x} \leq x\right\}.$$

That is, the probability of the random variable $\mathbf{x} \leq x$ is the CDF evaluated at x. As shown in Figure 15.5, $F(x)$ ranges from 0 to 1. The inversion method for generating random numbers of an arbitrary distribution consists of the following two steps:

1. Generate a random number, u, from a uniform distribution between 0 and 1.

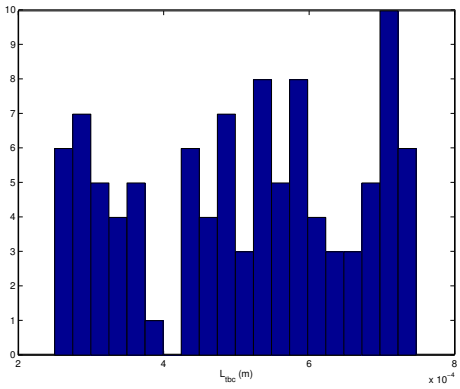
2. Given u, find the value of x at which $u = F(x)$. In otherwords, invert $F(x)$ such that, $x = F^{-1}(u)$.

15.2.1 Triangular Distributions

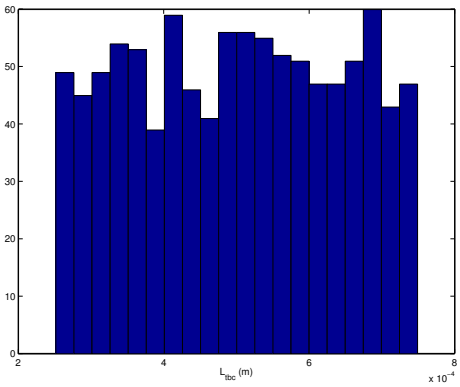
This was discussed in class. Hand-written notes were distributed.

15.2.2 Empirical Distributions

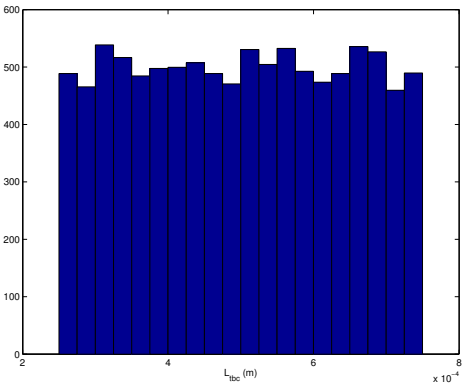
This was discussed in class.



(a) $N = 100$

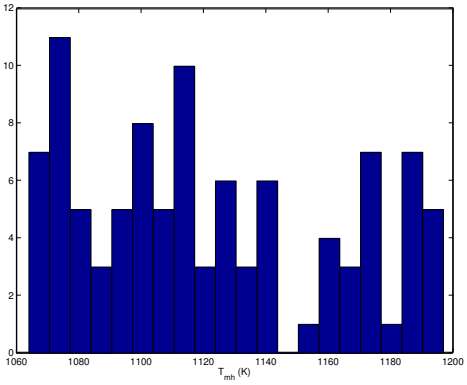


(b) $N = 1000$

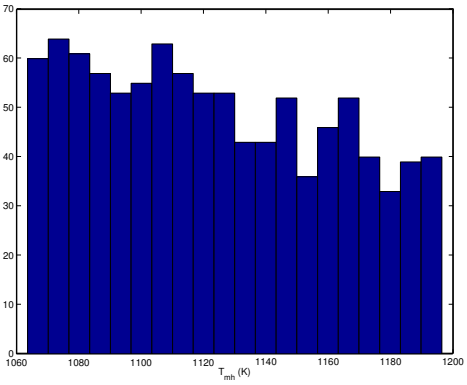


(c) $N = 10000$

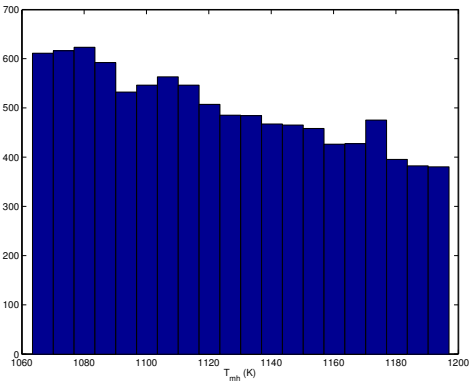
Figure 15.3: Distribution of a random sample from a uniformly-distributed L_{TBC} for a sample size of $N = 100$, 1000, and 10000.



(a) $N = 100$



(b) $N = 1000$



(c) $N = 10000$

Figure 15.4: Distribution of a T_{mh} from a uniformly-distributed L_{TBC} for a sample size of $N = 100$, 1000, and 10000.

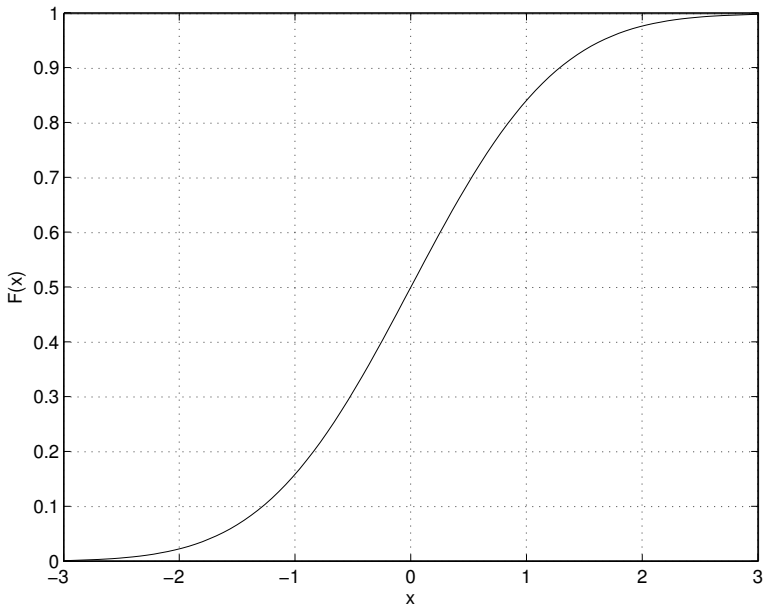


Figure 15.5: Cumulative distribution function (CDF) of a normally distributed variables with zero mean and unit variance.

Lecture 16

Error Estimates for the Monte Carlo Method

These notes address the accuracy of Monte Carlo methods in estimating probabilistic outputs. We will begin with errors when estimating the expected (i.e. mean) value, and then move on to other quantities such as the variance and probability.

16.1 Mean

Let the output of interest be labelled y, e.g. $y = T_{mh}$ in the previous turbine blade heat transfer problem. For a Monte Carlo simulation of sample size, N, label the individual values from each trial be labelled, y_i where $i = 1$ to N. In this section, we will consider the error made when estimating the expected value, also known as the mean, of y when using a sample of size N. This Monte Carlo error estimate is in fact a direct application of the field of statistics. If the distribution of y is $f(y)$, then the expected value of y is,

$$\mu_y \equiv E(y) = \int_{-\infty}^{+\infty} y f(y) dy.$$

Using the N trials, a reasonable **estimator** for μ_y would be,

$$\overline{y} \equiv \frac{1}{N} \sum_{i=1}^{N} y_i.$$

In-class Discussion 16.1 (Mean of T_{mh} from Monte Carlo) *The variability of the sample mean of T_{mh} will be demonstrated in class using various sample sizes.*

As seen in the In-class Discussion 16.1, the sample mean, $\overline{T_{mh}}$ varies from Monte Carlo simulation-to-simulation. However, as the sample size N increased, the variability of the sample means decreased. One question which might be asked is: on average how accurate is $\overline{y}$ as an estimate of μ_y? To see this, take the expectation of $\overline{y} - \mu_y$:

$$E(\overline{y} - \mu_y) \quad = \quad E(\overline{y}) - \mu_y,$$

$$
\begin{aligned}
&= E\left(\frac{1}{N}\sum_{i=1}^{N} y_i\right) - \mu_y, \\
&= \frac{1}{N}\sum_{i=1}^{N} E(y_i) - \mu_y.
\end{aligned}
$$

Since the y_i's occur from a random sampling of the inputs when using the Monte Carlo method, then $E(y_i) = \mu_y$, thus:

$$E(\overline{y} - \mu_y) = \frac{1}{N} N\mu_y - \mu_y = 0.$$

This result shows that on average, the error in using $\overline{y}$ to approximate μ_y is zero. When an estimator gives an expected error of zero, it is called an **unbiased estimator**.

To quantify the variability in $\overline{y}$, we use the variance of $\overline{y} - \mu_y$:

$$
\begin{aligned}
E\left[(\overline{y} - \mu_y)^2\right] &= E\left[\left(\left(\frac{1}{N}\sum_{i=1}^{N} y_i\right) - \mu_y\right)^2\right], \\
&= E\left[\left(\frac{1}{N}\sum_{i=1}^{N} (y_i - \mu_y)\right)^2\right], \\
&= E\left[\frac{1}{N^2}(y_1 - \mu_y + y_2 - \mu_y + \cdots)(y_1 - \mu_y + y_2 - \mu_y + \cdots)\right], \\
&= E\left[\frac{1}{N^2}\left\{(y_1 - \mu_y)^2 + (y_2 - \mu_y)^2 + \cdots + 2(y_1 - \mu_y)(y_2 - \mu_y) + \cdots\right\}\right].
\end{aligned}
$$

Because the Monte Carlo method draws independent, random samples, then the following two conditions hold,

$$E\left[(y_i - \mu_y)^2\right] = E\left[(y - \mu_y)^2\right] \equiv \sigma_y^2.$$
$$E\left[(y_i - \mu_y)(y_i - \mu_y)\right] = 0.$$

Thus, the variance of the mean estimate is,

$$E\left[(\overline{y} - \mu_y)^2\right] = \frac{\sigma_y^2}{N}.$$

Summarizing, we have found that,

$$\mu_{\overline{y}} \equiv E(\overline{y}) = \mu_y, \qquad \sigma_{\overline{y}}^2 \equiv E[(\overline{y} - \mu_y)^2] = \frac{\sigma_y^2}{N}.$$

Note, the quantity, $\sigma_{\overline{y}}$ is known as the **standard error** of the estimator. Thus, the standard error decreases with the square root of the sample size, $\sqrt{N}$. In other words, to decrease the variability in the estimate by a factor of 10 requires a factor of 100 increase in the sample size.

In-class Discussion 16.2 (Distribution of $\overline{T_{mh}}$) *In-class we will discuss how $\overline{T_{mh}}$ is distributed from Monte Carlo simulation-to-simulation.*

For large sample size N, the central limit theorem can be applied to approximate the distribution of $\overline{y}$. Specifically, the central limit theorem says for large N, the distribution of $\overline{y}$ will approach a normal distribution with mean μ_y and variation $\sigma_y/\sqrt{N}$:

$$f(\overline{y}) \to N(\mu_{\overline{y}}, \sigma_{\overline{y}}) = N(\mu_y, \sigma_y/\sqrt{N}).$$

We can now use this to make some very precise statements about the error in $\overline{y}$. Suppose, for example, that we want to have 95% confidence on the possible values for μ_y. Since the normal distribution has approximately 95% of its values within ± 2 standard deviations of the mean, then we know that,

$$P\left\{-2\frac{\sigma_y}{\sqrt{N}} \le \bar{y} - \mu_y \le 2\frac{\sigma_y}{\sqrt{N}}\right\} \approx 0.95.$$

If even higher confidence is wanted, then a wider range of error must be accepted. For example, a 99% confidence interval occurs at ± 3 standard deviations for a normal distribution. Thus,

$$P\left\{-3\frac{\sigma_y}{\sqrt{N}} \le \bar{y} - \mu_y \le 3\frac{\sigma_y}{\sqrt{N}}\right\} \approx 0.99.$$

Unfortunately, in a practical situation, we cannot actually calculate the above error estimates or confidence intervals because they depend on σ_y and we do not know σ_y. So, we typically use an estimate of σ_y. In particular, an unbiased estimate of σ_y^2 is,

$$s_y^2 \equiv \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})^2. \tag{16.1}$$

So, the usual practice is to replace σ_y by s_y in the various error estimates. Note, this does introduce additional uncertainty in the quality of the estimate and for small sample sizes this could be significant.

16.2 Other Estimators and Standard Errors

16.2.1 Probability

Often, Monte Carlo simulations are used to estimate the probability of an event occurring. For example, in the turbine blade example, we might be interested in the probability that the hot metal temperature exceeds a critical value. Generically, suppose that the event of interest is A. Then, an estimate of $P\{A\}$ is the fraction of times the event A occurs out of the total number of trials,

$$\hat{p}(A) = \frac{N_A}{N},$$

where N_A is the number of times A occurred in the Monte Carlo simulation of sample size N. $\hat{p}(A)$ is actually an unbiased estimate of $P\{A\}$. To see this, define a function $I(A_i)$

which equals 1 if event A occurred on the i-th trial, and equals zero if A did not occur. For example, if the event A is defined as $y > y_{limit}$, $I(A_i)$ would be defined as,

$$I(A_i) = I(y_i > y_{limit}) = \begin{cases} 1 & \text{if } y_i > y_{limit}, \\ 0 & \text{if } y_i \leq y_{limit}. \end{cases}$$

Using this definition, the number of times which A occurred can be written,

$$N_A = \sum_{i=1}^{N} I(A_i). \tag{16.2}$$

Finding the expectation of N_A gives,

$$\begin{aligned} E[N_A] &= E\left[\sum_{i=1}^{N} I(A_i)\right], \\ &= \sum_{i=1}^{N} E\left[I(A_i)\right]. \end{aligned}$$

Since we assume that the Monte Carlo trials are drawn at random and independently from each other, then $E[I(A_i)] = P\{A\}$. Thus,

$$E[N_A] = NP\{A\}.$$

Finally, using this result it is easy to show that,

$$E[\hat{p}(A)] = \frac{E[N_A]}{N} = P\{A\}.$$

We can also use Equation (16.2) in combination with the central limit theorem to show that $hatp(A)$ is normally distributed for large N with mean $P\{A\}$ and standard error,

$$\sigma_{\hat{p}} = \sqrt{\frac{P\{A\}\left(1 - P\{A\}\right)}{N}}.$$

In-class Discussion 16.3 (Low Probability Estimation with Monte Carlo)

16.2.2 Variance

The variance of y is given the symbol, σ_y^2, and is defined as,

$$\sigma_y^2 = E[(y - \mu_y)^2]. \tag{16.3}$$

As noted in Equation (16.1), an unbiased estimator of σ_y^2 is s_y^2, that is:

$$E[s_y^2] = \sigma_y^2.$$

Note, you should try proving this result.

To quantify the uncertainty in this estimator, we would like to determine the standard error,

$$\sigma_{s_y^2} \equiv \left\{ E\left[(s_y^2 - \sigma_y^2)^2 \right] \right\}^{1/2}.$$

Unfortunately, this standard error is not known for general distributions of y. However, if y has a normal distribution, then,

$$\sigma_{s_y^2} = \frac{\sigma_y^2}{\sqrt{N/2}}. \tag{16.4}$$

Under the assumption of y being normally distributed, the distribution of s_y^2 is also related to the chi-squared distribution. Specifically, $(N-1)s_y^2/\sigma_y^2$ has a chi-square distribution with $N-1$ degrees of freedom. Note that the requirement that y be normally distributed is much more restrictive than the requirements for the mean error estimates to hold. For the mean error estimates, the standard error, $\sigma_{\overline{y}} = \sigma_y/\sqrt{N}$, is exact regardless of the distribution of y. The application of the central limit theorem which gives that $\overline{y}$ is normally distributed only requires that the number of samples is large but does constrain the distribution of y itself (beyond requiring that $f(y)$ is continuous).

16.2.3 Standard Deviation

Typically, the standard deviation of y is estimated using s_y, i.e. the square root of the variance estimator. This estimate, however, is biased,

$$E[s_y] \neq \sigma_y.$$

The standard error for this estimate is only known exactly when y is normally distributed. In that case,

$$\sigma_{s_y} \equiv \left\{ E[(s_y - \sigma_y)^2] \right\}^{1/2} = \frac{\sigma_y}{\sqrt{2N}}.$$

16.3 Bootstrapping

The standard errors and confidence intervals for a variety of estimators often rely upon assumptions on the underlying output values, y. For example, the standard error for the variance as given in Equation 16.4 requires the assumption that y is distributed normally. Clearly, in most applications that will not be the case. In this situation, an alternative

method is required if confidence intervals are desired. One such method is known as bootstrapping.

As an example, let's consider the Monte Carlo estimation of the 50-percentile value of a population. From a Monte Carlo sample, the 50-percentile value of the population could be estimated as the median of the sample. If we wanted to determine the standard error associated with using the median of sample size N as an estimate for the 50-percentile value, one possibility would be to run multiple Monte Carlo's of sample size N and estimate the variability of the estimator directly. Suppose the a total of M Monte Carlo samples are run (each of size N), then the total number of simulations would be $M \times N$. Labelling θ_i as the estimator (i.e. the median) from the i-th Monte Carlo sample, the distribution of θ can be determined and confidence intervals can be built using the M values of θ_i. This, however, could be very expensive since $M \times N$ simulations are required.

Bootstrapping avoids the need for $M \times N$ simulations by resampling from a single Monte Carlo simulation. After a single Monte Carlo simulation, N simulations have been performed and the results of the simulations are all equally likely to have occurred (since they were drawn in a random, independent manner). A bootstrap resampling is performed by drawing with uniform probability from this original Monte Carlo sample. Specifically, if we wanted to generate M bootstrap samples of size N:

1. Perform an initial Monte Carlo sample of size N to produce y_i with $i = 1$ to N. From this sample, calculate the desired estimator $\theta_1 = \theta(y_i)$.

2. Resample the values of y_i assuming uniform probability of the events (i.e. each y_i has a probability of $1/N$). Since the same event may occur a different number of times in this resample and in the original sample, this will generate a new sample, $\hat{y}_i$. From this resample, calculate the desired estimator, $\theta_j = \theta(\hat{y}_i)$.

3. Perform the resampling in Step 2 a total of $M - 1$ times. In all, this will produce M values of the estimator, θ_1 through θ_M.

4. From the M values of θ_i, confidence intervals can be determined.

Appendix A

Summary of Multi-Step Methods

This appendix contains a summary of the most common multi-step methods. Multi-stage methods (i.e. Runge-Kutta methods) are discussed in Lecture 6.

Recall from Definition 3.1, the generic form of an s-step multi-step method is,

$$v^{n+1} + \sum_{i=1}^{s} \alpha_i v^{n+1-i} = \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i}.$$

A multi-step method with $\beta_0 = 0$ is known as an **explicit** method since in this case the new value v^{n+1} can be determined as an explicit function of known values (i.e. from v^i and f_i with $i \leq n$). A multi-step method with $\beta_0 \neq 0$ is known as an **implicit** method since in this case the new value v^{n+1} is an implicit function of itself through the forcing function, $f^{n+1} = f(v^{n+1}, t^{n+1})$.

A.1 Adams-Bashforth Methods

Adams-Bashforth methods are explicit methods of the form,

$$v^{n+1} - v^n = \Delta t \sum_{i=1}^{s} \beta_i f^{n+1-i}.$$

Thus, the basic time derivative approximation remains the same for all p (i.e. du/dt is approximated by $(v^{n+1} - v^n)/Dt$) and the higher-order accuracy is achieved by using more values of f. The coefficients for the first through fourth order methods are given in Table A.1. The first-order Adams-Bashforth is forward Euler.

The stability boundary for these methods are shown in Figure A.1. As the order of accuracy increases, the stability regions become smaller. Note, this is the opposite of Runge-Kutta methods for which the size of the stability regions increases with increased accuracy (see Lecture 6).

A.2 Adams-Moulton Methods

Adams-Moulton methods are implicit methods of the form,

$$v^{n+1} - v^n = \Delta t \sum_{i=0}^{s} \beta_i f^{n+1-i}.$$

These methods use the same time derivative approximation as the Adams-Bashforth methods, however they include the $n+1$ value of f. The coefficients for the first through fourth order methods are given in Table A.2.

The stability boundary for these methods are shown in Figure A.2. While the stability regions are larger than the Adams-Bashforth methods, for $p > 2$ the methods have bounded stability regions. Thus, they will not be appropriate for stiff problems.

A.3 Backward Differentiation Methods

Backwards differentiation methods were described in detail in Section 5.4. For completeness, we have included much of the same information in this appendix. Backwards differentiation methods are implicit methods of the form,

$$v^{n+1} + \sum_{i=1}^{s} v^{n+1-i} = \Delta t \beta_0 f^{n+1}.$$

The coefficients for the first through fourth order methods are given in Table A.3.

The stability boundary for these methods are shown in Figure A.3. As can be seen, all of these methods are stable everywhere on the negative real axis, and are mostly stable in the left-half plane in general. Even up to $p = 4$, the stability regions are unbounded. Thus, backwards differentiation work well for stiff problems in which stong damping is present.

p	β_1	β_2	β_3	β_4
1	1			
2	$\frac{3}{2}$	$-\frac{1}{2}$		
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$

Table A.1: Coefficients for Adams-Bashforth methods (these methods are explicit so $\beta_0 = 0$). Note: the $p = 1$ method is the forward Euler method.

p	β_0	β_1	β_2	β_3
1	1			
2	$\frac{1}{2}$	$\frac{1}{2}$		
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$

Table A.2: Coefficients for Adams-Moulton methods. Note: the $p = 1$ method is the backward Euler method, and the $p = 2$ method is the Trapezoidal method.

p	α_1	α_2	α_3	α_4	β_0
1	-1				1
2	$-\frac{4}{3}$	$\frac{1}{3}$			$\frac{2}{3}$
3	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$		$\frac{6}{11}$
4	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$	$\frac{12}{25}$

Table A.3: Coefficients for backward differentiation methods. Note: the $p = 1$ method is the backward Euler method.

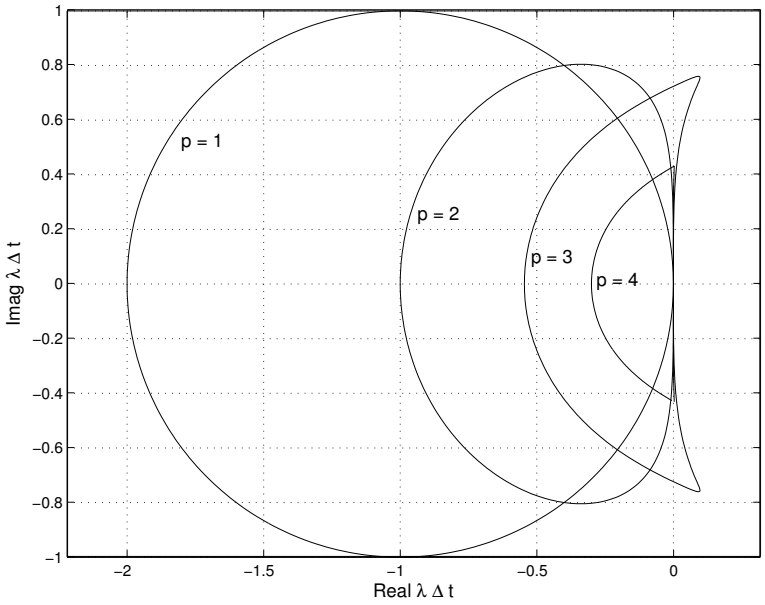


Figure A.1: Adams-Bashforth stability regions for $p = 1$ through $p = 4$ methods. Note: interior of contours is stable region.

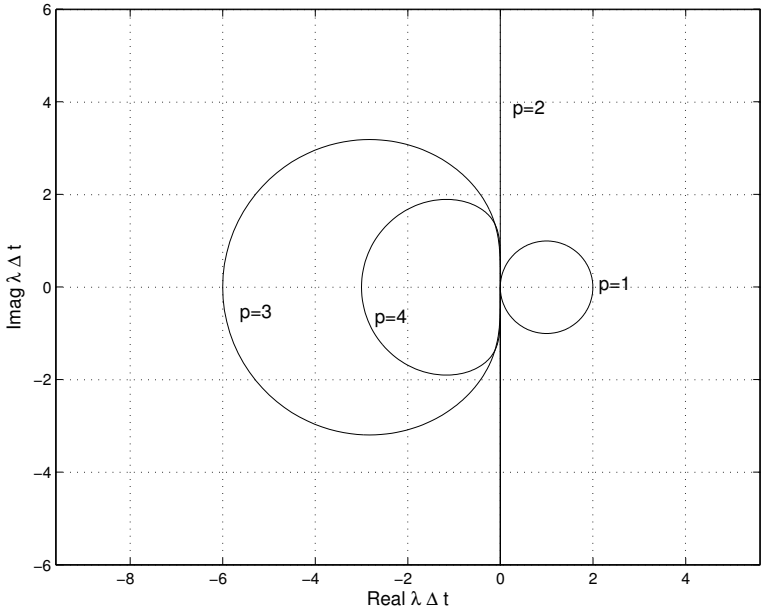


Figure A.2: Adams-Moulton stability regions for $p = 1$ through $p = 4$ methods. Note: $p = 1$ is stable outside of contour, the $p = 2$ integrator is stable in the left-half plane, and $p \geq 3$ are stable inside their respective contours.

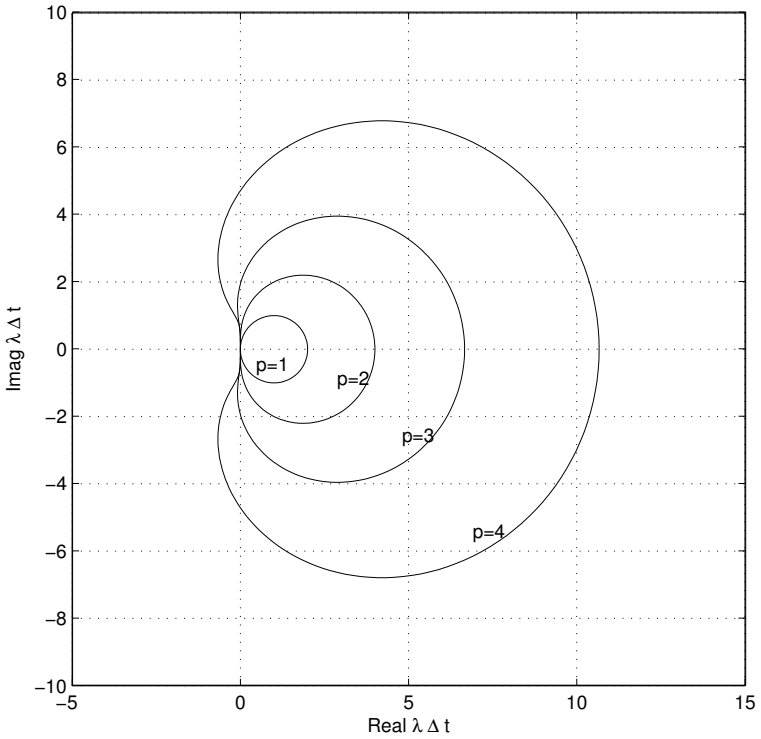


Figure A.3: Backwards differentiation stability regions for $p = 1$ through $p = 4$ methods. Note: interior of contours is unstable region.

Appendix B

Review of Probability and Statistics

The purpose of this appendix is to provide a quick review of some basic concepts in probability.

B.1 Outcomes and Events

Consider an experiment or activity which will be performed several times. Each time the experiment is performed, the outcome ζ can be recorded. An event A is set of outcomes for which certain conditions have been met. An elementary event consists of only a single outcome.

Example: Consider the inspection of rotor blades in the turbine of a jet engine. Suppose that the total number of rotor blades in the engine is N. The outcome of a single inspection is recorded as the number of blades that must be replaced due to damage. Thus, the outcomes are the set of non-negative integers, $\{0, 1, 2, ..., N\}$. If the number of blades replaced in a single engine is greater than some number, say 5, than this may indicate more significant damage has occurred and the engine will need to be inspected more thoroughly. In this situation, we would be interested in the event where the number of replaced blades is $\{6, 7, 8, ..., N\}$. This is not an elementary event since it consists of more than one outcome. However, we would also be interested in the situation where no blades are replaced. In this case, the event consists of a single outcome (i.e. 0) and therefore is an elementary event.

B.2 The Meaning of Probability

Given an event, A, the probability of the event is $P\{A\}$. Probabilities are assumed to satisfy the following properties:

- $P\{A\} \geq 0$.
- If and only if the event is certain to occur, then $P\{A\} = 1$.
- Given two mutually exclusive events, A and B, then $P\{A + B\} = P\{A\} + P\{B\}$.

B.3 Random Variables

The utility of probability theory is in describing the behavior of random variables. In the simplest terms, a random variable can be defined as a variable or parameter whose values depend on the particular outcome of the experimental trial. Thus, random variables will be a function of the outcome. We will use boldface letters to denote a random variable, for example, $\mathbf{x}$. It is understood that the value of $\mathbf{x}$ depends on the outcome which has occurred, i.e. $\mathbf{x} = \mathbf{x}(\zeta)$.

Example: We will continue with the rotor blade inspection example. A very simple example of a random variable would be the number of blades which are replaced for a particular inspection. In this case, the random variable is just the outcome itself! Specifically,

$$\mathbf{N}(\zeta) = \zeta.$$

Now, let's try something a little more complicated. The airline is concerned about the cost of the inspection and repair of its engines. Not including the cost of replacing any blades, simply performing the inspection costs the airlines C_I dollars due to employee salaries (labor). The cost to replace a single blade is C_B (including the cost of the new blade and the labor). Also, if the number of replacements is greater than 5, the cost rises dramatically since a more thorough inspection must be performed. We will model this as an increment C_D. Since the cost of the inspection and repair depends on the outcome of the inspection (and the outcome is random), clearly the cost of the inspection is a random variable. Specifically,

$$\mathbf{C}(\zeta) = \begin{cases} C_I + C_B\zeta, & \text{for } 0 \leq \zeta \leq 5, \\ C_I + C_B\zeta + C_D, & \text{for } 6 \leq \zeta \leq N. \end{cases}$$

B.4 Probability density functions (PDF)

We are often concerned with probabilities of parameters which are real numbers (i.e. which have infinitely many values). In this case, a probability density function (PDF) is used to describe the probability of the parameter being in some range. In particular, given a random variable $\mathbf{x}$, the probability that $a \leq \mathbf{x} \leq b$ is,

$$P\{a \leq x \leq b\} = \int_a^b f(x)dx,$$

where $f(x)$ is the PDF of $\mathbf{x}$.

A common (and probably the simplest) distribution is the uniform distribution. In this case, we assume the probability density is constant within some range and zero outside of this range,

$$\textbf{Uniform: } f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0, & \text{otherwise.} \end{cases}$$

Other distribution types are described in Section B.9

B.5 Expected value and mean

Given the PDF, $f(x)$ of a random variable $\mathbf{x}$, the expectated value is defined as,

$$E\{\mathbf{x}\} \equiv \int_{-\infty}^{+\infty} x f(x)\, dx.$$

The expected value of $\mathbf{x}$ is also known as the mean value. We will also use the symbol μ_x for the expected value of $\mathbf{x}$.

B.6 Variance and standard deviation

The variance of $\mathbf{x}$ is defined as,

$$\sigma_x^2 \equiv \int_{-\infty}^{+\infty} (x - \mu_x)^2 f(x)\, dx.$$

The value σ_x is known as the standard deviation of $\mathbf{x}$. The variance is a measure of the variability in $\mathbf{x}$ about its mean value. A frequently used relationship exists between the mean and variance,

$$\sigma_x^2 = E\left\{\mathbf{x}^2\right\} - \mu_x^2.$$

Try proving this!

B.7 Cumulative distribution functions (CDF)

Cumulative distribution function (CDF) of $\mathbf{x}$ is defined as the probability that $\mathbf{x} \leq x$. Specifically,

$$F(x) \equiv P\{\mathbf{x} \leq x\}.$$

From the basic assumptions of probability, it can be shown that $F(-\infty) = 0$ (i.e. the probability of $\mathbf{x}$ becoming infinite is zero) and $F(+\infty) = 1$ (i.e. the probability of $\mathbf{x}$ being less than infinity is one). The CDF and PDF of $\mathbf{x}$ are related as follows,

$$F(a) = \int_{-\infty}^{a} f(x)\, dx$$

Thus, we can show,

$$F(b) - F(a) = \int_{a}^{b} f(x)\, dx.$$

Furthermore, this implies that

$$f = \frac{dF}{dx}.$$

B.8 Percentiles

The u percentile of $\mathbf{x}$ is the smallest number x_u such that,

$$u = P\{\mathbf{x} \leq x_u\} = F(x_u).$$

Note, since u is a probability, its range is $0 \leq u \leq 1$.

B.9 Common distribution types

B.9.1 Normal distribution

The normal (or Gaussian) distribution is,

$$f(x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-(x-\mu_x)^2/2\sigma_x^2}.$$

We will use the common notation $\mathbf{x} = N(\mu; \sigma)$ to indicate that $\mathbf{x}$ is a normally-distributed random variable with mean μ and standard deviation σ.