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.

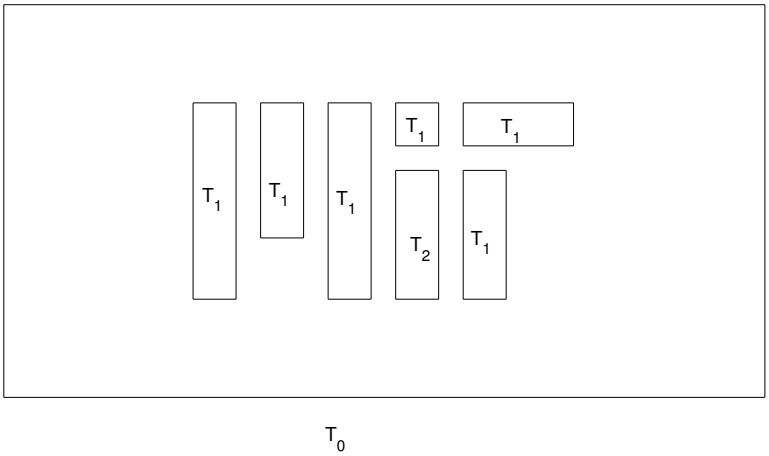


Figure 12.1: Heat transfer problem in a complex domain. Temperature at outer boundary is maintained at T_0. Temperature of all of the internal rectangles except one are maintained at T_1, while the remaining internal rectangle is maintained at T_2.

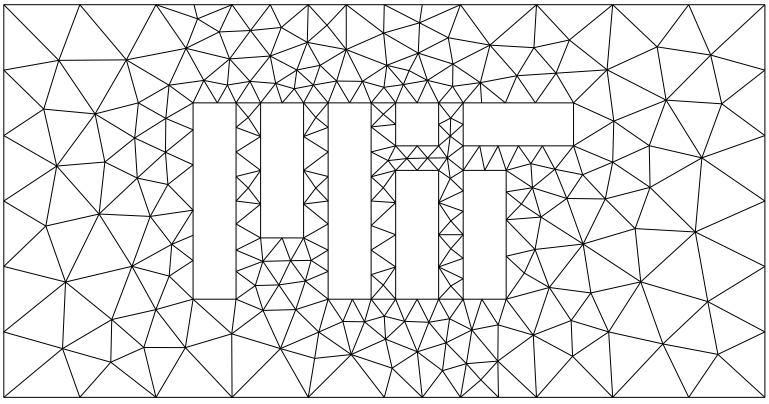


Figure 12.2: Mesh for heat transfer problem in a complex domain.

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).

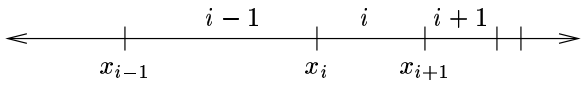


Figure 12.3: Mesh and notation for one-dimensional finite element method.

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

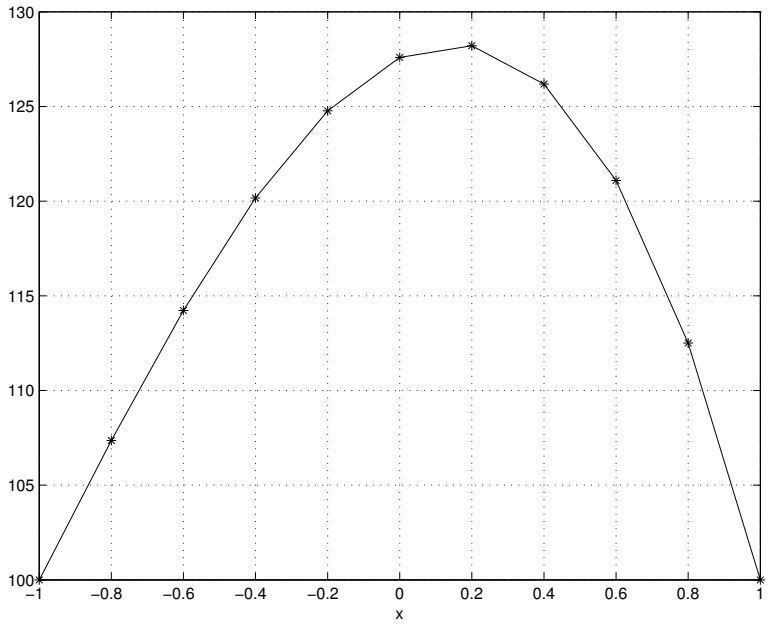


Figure 12.4: A linear element solution on a mesh with constant element size, $\Delta x_i = 0.2$.

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,

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

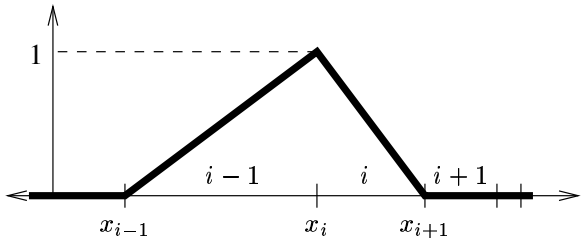


Figure 12.5: $\phi_i(x)$ for linear elements using a nodal basis.

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 12.5. 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 12.6. 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, $\Delta x_{j-1} = \Delta x_j$*).*

12.5 Boundary Conditions

Coming in next installment.

Example 12.2 (FEM implementation 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=0
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 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,

$$\int_{-1}^{+1} \xi^i \, d\xi = \begin{cases} 0 & \text{if } i = \text{odd} \\ \frac{2}{i+1} & \text{if } i = \text{even} \end{cases}$$

$$\Rightarrow \int_{-1}^{+1} \sum_{i=1}^{M} c_i \xi^i \, d\xi = 2\left(c_0 + \frac{1}{3}c_2 + \frac{1}{5}c_4 + \cdots\right). \tag{12.7}$$

12.6.1 $N_q = 1$ 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,

$$\int_{-1}^{+1} g(\xi) \, d\xi \approx \alpha_1 g(\xi_1), \qquad \alpha_1 = 2, \quad \xi_1 = 0.$$

12.6.2 $N_q = 2$ 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 $\phi_1(\xi)$.

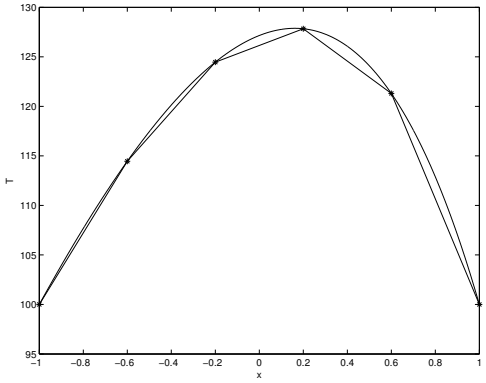
The following is a Matlab script that uses Gaussian quadrature to evaluate the forcing integral and solve the problem described in Example 11.1. The number of points being used is set at the beginning of the script. Results for both 1-point and 2-point quadrature are shown in Figures 12.7 and 12.8 for 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,
```

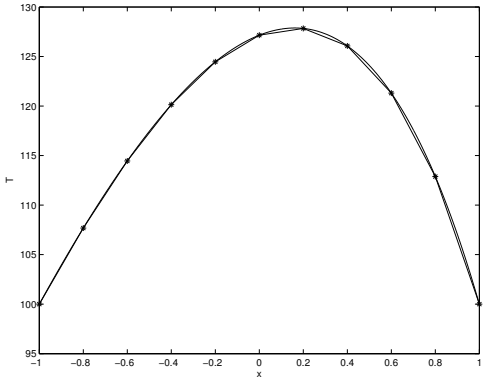
```
kn1 = ii;
kn2 = ii+1;

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=0
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.
```

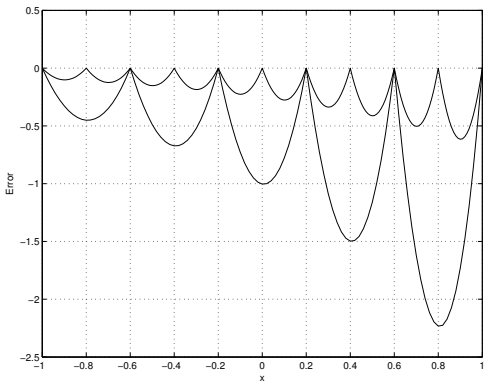
```
verr(1) = v(1) - Te(1);
h = x(2)-x(1);
for i = 2:Npt-1,
  xxi = xe(i);
  Ti  = Te(i);
  j = floor((xxi-xe(1))/h) + 1;
  x0 = x(j);
  x1 = x(j+1);
  v0 = v(j);
  v1 = v(j+1);
  xi = 2*(xxi - x0)/(x1-x0)-1;  % This gives xi between +/-1
  vi = 0.5*(1-xi)*v0 + 0.5*(1+xi)*v1;
  verr(i) = vi - Ti;
end
verr(Npt) = v(Ne+1) - Te(Npt);

figure(2);
plot(xe,verr);
xlabel('x');
ylabel('Error');
```



(a) $N = 5$ elements

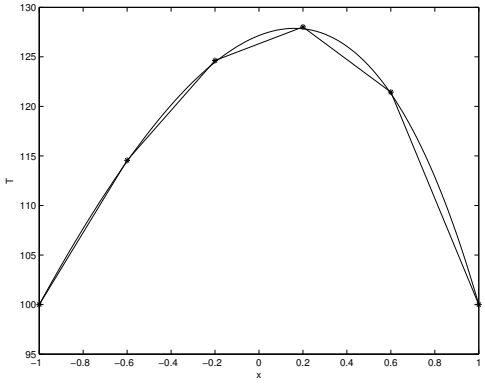


(b) $N = 10$ elements

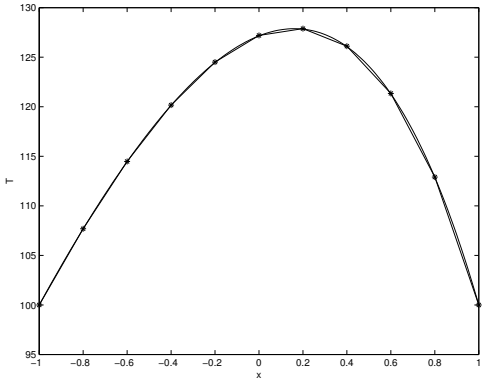


(c) Error

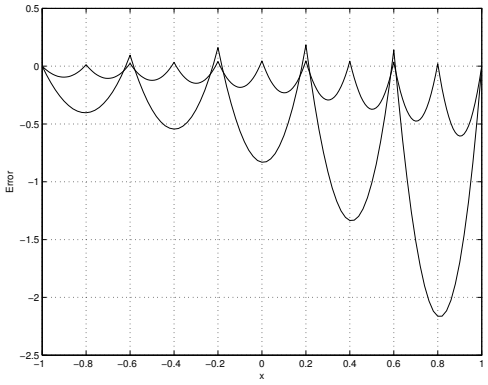
Figure 12.6: Comparison of finite element solution to exact solution.



(a) $N = 5$ elements

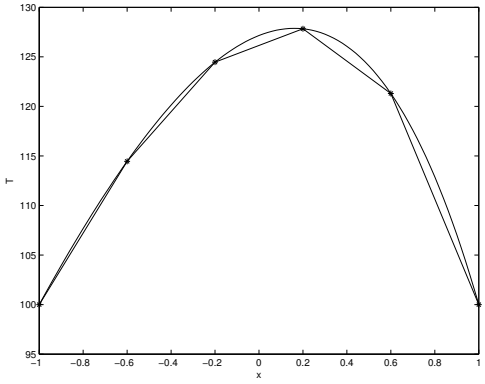


(b) $N = 10$ elements

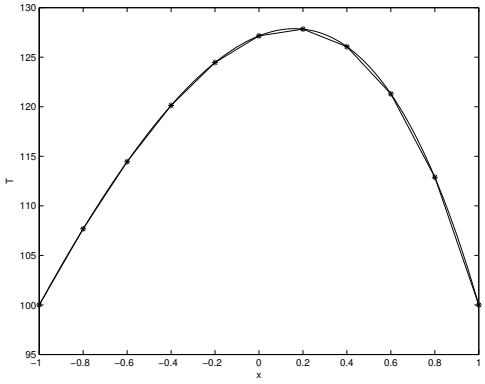


(c) Error

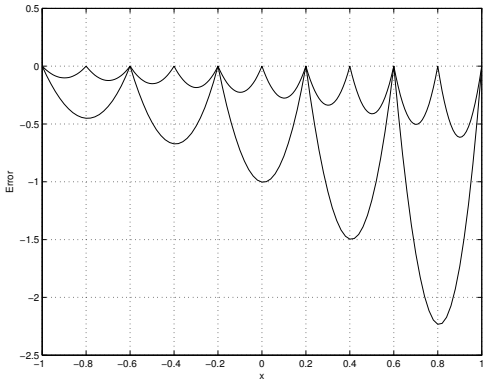
Figure 12.7: Comparison of finite element solution using $N_q = 1$ point Gaussian quadrature to exact solution.



(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.