The Finite Element Method

Computer Lab 1

Introduction

Differential equations are of vast interest in science and engineering since they provide mathematical models to a wide range of real world phenomenons, such as heat transfer, electromagnetic waves, and the weather, for instance. Thus it is very appealing to try to gain insight into the world around us simply by solving various differential equations. Unfortunately solutions to differential equations can rarely be expressed by closed formulas, but they can often be approximated rather simply. The finite element method is a general technique for computing numerical (i.e., approximate) solutions to differential equations. In this computer session we shall study the derivation and implementation of a finite element method a two point boundary value problem, namely,

$$-u'' = f, \quad a < x < b, \tag{1}$$

$$u(a) = u(b) = 0, \tag{2}$$

where u is the sought solution and f a given function.

Weak or Variational Form

A finite element method is always derived from the weak or variational formulation of the problem at hand. To this end, let us define the vector space

$$H_0^1 = \{v : \|v\|^2 + \|v'\|^2 < \infty,\ v(a) = v(b) = 0\}. \tag{3}$$

Multiplying $f = -u''$ by a function $v \in H_0^1$ and integrating by parts gives

$$\int_a^b fv\,dx = \int_a^b -u''v \tag{4}$$

$$= -[u'v]_a^b + \int_a^b u'v'\,dx \tag{5}$$

$$= -u'(b)v(b) + u'(a)v(a) + \int_a^b u'v'\,dx. \tag{6}$$

Since $v(a) = v(b) = 0$ the boundary terms drop out and we are left with

$$\int_a^b fv\,dx = \int_a^b u'v'\,dx. \tag{7}$$

Hence, the weak or variational form of (1) reads: Find $u \in H_0^1$, such that

$$\int_a^b u'v'\,dx = \int_a^b fv\,dx, \tag{8}$$

for all $v \in H_0^1$.

Finite Element Approximation

The space H_0^1 is very big and contains many functions and it is therefore just as hard to find a function $u \in H_0^1$ which satisfies the variational equation (8) as it is to solve the original problem (1). The approximation of the finite element method is therefore to look for a solution approximation u_h within a small (i.e., finite dimensional) subspace V_h of H_0^1 typically consisting of piecewise polynomials.

Thus let V_h be the space of all continuous piecewise linear functions, which vanish at the end points a and b, on a uniform partition $a = x_0 < x_1 < x_2 < \ldots < x_N = b$ of the interval $a \leq x \leq b$ into N subintervals of equal length h. Moreover let $\{\varphi_j\}$ be the set of hat basis functions of V_h associated with the $N-1$ nodes x_j, $j = 1, 2 \ldots, N-1$, such that

$$\varphi_i(x_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{9}$$

Note especially that there are no hat functions φ_0 and φ_N since the functions of V_h must vanish at the end points x_0 and x_N. Otherwise would not V_h be a subspace of H_0^1.

The finite element approximation of (8) thus reads: Find $u_h \in V_h$ such that

$$\int_a^b u_h' v' \, dx = \int_a^b f v \, dx, \tag{10}$$

for all $v \in V_h$.

It can be shown that (10) is equivalent to the $N-1$ equations

$$\int_a^b u_h' \varphi_i' \, dx = \int_a^b f \varphi_i \, dx, \quad i = 1, 2, \ldots, N-1, \tag{11}$$

because if (10) is satisfied for each basis function φ_j separately then it is also satisfied for any linear combination of them, that is, any arbitrary function $v \in V_h$.

Expanding u_h as a linear combination of hat functions we make the ansatz

$$u_h = \sum_{j=1}^{N-1} \xi_j \varphi_j(x), \tag{12}$$

where ξ_j, $j = 1, 2, \ldots, N-1$ are $N-1$ coefficients, the nodal values of u_h, to be determined.

Substituting (12) into (10) yields

$$\sum_{j=1}^{N-1} \xi_j \int_a^b \varphi_j' \varphi_i' \, dx = \int_a^b f \varphi_i \, dx \quad i = 1, 2, \ldots, N-1, \tag{13}$$

which is a $(N-1) \times (N-1)$ system of equations for ξ_j. In matrix form we write

$$A\xi = b, \tag{14}$$

where A is a $(N-1) \times (N-1)$ matrix, the so-called stiffness matrix, with entries

$$A_{i,j} = \int_a^b \varphi_i'(x) \, \varphi_j'(x) \, dx, \quad i, j = 1, 2, \ldots, N-1, \tag{15}$$

$\xi = (\xi_1, \xi_2, \ldots, \xi_{N-1})^T$ is a $(N-1)$ vector containing the unknown coefficients ξ_j, $j = 1, 2, \ldots, N-1$, and b is a $(N-1)$ vector, the so-called load vector, with entries

$$b_i = \int_a^b f(x)\varphi_i(x)\,dx, \quad i = 1, 2, \ldots, N-1. \tag{16}$$

Computer Implementation

Assembling the stiffness matrix and load vector

The explicit expression for a hat function $\varphi_i(x)$ is given by

$$\varphi_i(x) = \begin{cases} (x - x_{i-1})/h, & \text{if } x_{i-1} \leq x \leq x_i, \\ (x_{i+1} - x)/h, & \text{if } x_i \leq x \leq x_{i+1}, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

Hence the derivative φ_i' is either $+1/h$, $-1/h$, or 0 depending on the interval.

Using (17) it is straightforward to calculate the entries of the stiffness matrix. For $|i-j| > 1$, we have $A_{i,j} = 0$, since φ_i and φ_j lack overlapping support. However, if $i = j$, then

$$\int_a^b \varphi_i'^2\,dx = \int_{x_{i-1}}^{x_i} \varphi_i'^2\,dx + \int_{x_i}^{x_{i+1}} \varphi_i'^2\,dx \tag{18}$$

$$= \int_{x_i}^{x_{i+1}} \frac{1}{h^2}\,dx + \int_{x_i}^{x_{i+1}} \frac{(-1)^2}{h^2}\,dx = \frac{2}{h}. \tag{19}$$

where we have used that $x_i - x_{i-1} = x_{i+1} - x_i = h$. Further, if $j = i+1$, then

$$A_{i,i+1} = \int_a^b \varphi_i'\varphi_{i+1}'\,dx = \int_{x_i}^{x_{i+1}} \varphi_i'\varphi_{i+1}'\,dx \tag{20}$$

$$= \int_{x_i}^{x_{i+1}} \frac{(-1)}{h} \cdot \frac{1}{h}\,dx = -\frac{1}{h}. \tag{21}$$

Changing i to $i-1$ we also have $A_{i-1,i} = -1/h$.

Thus the stiffness matrix looks like:

$$A = \frac{1}{h} \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}. \tag{22}$$

The entries b_i of the load vector must often be evaluated using quadrature, since they involve the function f which can be hard to integrate analytically. For example, using the trapezoidal rule one obtains the approximate load vector entries

$$b_i = \int_a^b f\varphi_i \, dx = \int_{x_{i-1}}^{x_{i+1}} f\varphi_i \, dx \approx f(x_i)h. \tag{23}$$

Assembly. From a computational point it is advantageous to rewrite (14) as

$$\frac{1}{h} \begin{bmatrix} 10^6 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 10^6 \end{bmatrix} \begin{bmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \vdots \\ \xi_{N-1} \\ \xi_N \end{bmatrix} = h \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) \end{bmatrix}, \tag{24}$$

that is, to extend the original $(N-1) \times (N-1)$ system of equations into a new $(N+1) \times (N+1)$ system of equations by adding additional equations for the nodal values $\xi_0 = u_h(x_0)$ and $\xi_N = u_h(x_N)$. Recalling the boundary conditions $u(a) = u(b) = 0$ it is clear that we want $\xi_0 = \xi_N = 0$. This is accomplished by setting the diagonal entries $A_{0,0}$ and $A_{N+1,N+1}$ to a big number, say 10^6. The entries b_0 and b_N of the extended load vector are superfluous and can be chosen arbitrarily.

The system of equations (24) can be computed using the standard assembly technique. Thus the stiffness matrix is successively computed by looping over the elements, for each element adding the element stiffness matrix, viz.,

$$\frac{1}{h}\begin{bmatrix} 1 & -1 & & & \\ -1 & 1 & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} + \frac{1}{h}\begin{bmatrix} & & & & \\ & 1 & -1 & & \\ & -1 & 1 & & \\ & & & & \\ & & & & \end{bmatrix} + \ldots + \frac{1}{h}\begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & 1 & -1 \\ & & & -1 & 1 \end{bmatrix}.$$

Each element matrix is found by performing the integration (15) over a single element.

A routine for assembling the stiffness matrix is listed below.

```
function A=my_stiffness_matrix_assembler(x)
%
% Returns the assembled stiffness matrix A.
% Input is a vector x of node coords.
%
N = length(x) - 1;   % number of elements
A = zeros(N+1, N+1); % initialize stiffnes matrix to zero
for i = 1:N          % loop over elements
  h = x(i+1) - x(i); % element length
  n = [i i+1];       % nodes
  A(n,n) = A(n,n) + [1 -1; -1 1]/h; % assemble element stiffness
end
A(1,1) = 1.e+6; % adjust for BC
A(N+1,N+1) = 1.e+6;
```

The load vector is computed analogously.

```
function B=my_load_vector_assembler(x)
%
% Returns the assembled load vector b.
% Input is a vector x of node coords.
%
N = length(x) - 1; B = zeros(N+1, 1); for i = 1:N
```

```
  h = x(i+1) - x(i);
  n = [i i+1];
  B(n) = B(n) + [f(x(i)); f(x(i+1))]*h/2;
end
```

Note that these algorithms also work when the node point are not equidistributed.

Putting it all together we get the main routine:

```
function my_first_fem_solver()
a = 0 % left end point of interval
b = 1 % right
N = 5 % number of intervals
h = (b-a)/N % mesh size
x = a:h:b; % node coords
A=my_stiffness_matrix_assembler(x);
B=my_load_vector_assembler(x);
xi = A\B; % solve system of equations
plot(x,xi) % plot solution
```

Discrete 1D Laplacian

Consider the residual of the approximation defined as

$$R(u_h) = f + u_h'', \tag{25}$$

where u_h is the finite element approximation of the exact solution of (1) and f is given right hand side. The residual is usually used in a posteriori error estimates and mesh adaptations. While using a space of piecewise linear polynomials V_h with basis functions defined in (17) we notice that the second derivative or Laplacian of u_h is zero.

In practice it will be convenient to use a discrete analog of the Laplacian Δ, and let us denote it as Δ_h. The discrete Laplacian is computed by projecting the Laplacian onto V_h, i.e. $\Delta_h u_h = P_h \Delta u_h$, where P_h denotes the L_2

projection onto V_h. For given $u_h \in V_h$ find $\Delta_h u \in V_h$ such that

$$\int_a^b \Delta_h u_h v dx = -\int_a^b u_h' v' dx, \quad \forall v \in V_h. \tag{26}$$

As it was shown in (12), for some nodal values ζ_j, $j = 1, 2, \ldots, N-1$ the discrete Laplacian can be viewed as

$$\Delta_h u_h = \sum_{j=1}^{N-1} \zeta_j \varphi_j(x). \tag{27}$$

Now, insert (27) and (12) into the variational formulation (26) we get the following linear system

$$M\zeta = -A\xi, \tag{28}$$

where ξ is the nodal values of u_h and M is the mass matrix with entities

$$M_{ij} = \int_a^b \varphi_j(x)\varphi_i(x)dx, \quad i, j = 1, 2, \ldots, N-1.$$

Once the mass matrix is computed, the nodal values of the discrete Laplacian can be found as

$$\zeta = -M^{-1}A\xi. \tag{29}$$

Problem 1. Implement the finite element solver outlined above. Test your code by solving $-u'' = 2$, $0 < x < 1$, $u(0) = u(1) = 0$. Use uniform meshes with $h = 1/2$, $1/4$, $1/16$ and $1/256$. Compare with the exact solution $u = x(1-x)$.

Problem 2. Modify your code so that it can handle the inhomogeneous boundary condition $u(1) = g$. Test your code by solving $-u'' = 0$, $0 < x < 1$, $u(0) = 0$, $u(1) = 7$.

Problem 3. Investigate the convergence rate of the finite element approximation u_h. Set up a problem for which the exact solution u is known and study how the error $e = u - u_h$ behaves when the mesh size h decreases. In particular, use the energy norm $\| \cdot \|_E$ to measure the error e. The energy norm depends on your choice of problem, but for the model problem (1) it is defined by

$$\|w\|_E^2 = \int_a^b (w')^2\,dx.$$

To compute $\|e\|_E$ note that $\|e\|_E^2 = \|u'\|^2 - \|u_h'\|^2$ due to orthogonality. Further, the term $\|u_h'\|^2$ may be numerically evaluated by the dot product `xi'*A*xi`.

Problem 4. If you compare equation (22) and (24) two new lines have been added in the matrix. This can actually be avoided. Assume the in general the boundary conditions are given as $u(a) = g_1$ and $u(b) = g_2$, where g_1 and g_2 are given boundary values. Since we know the values on the boundary we enforce the system (24) such that it gives the boundary values: $\xi_0 = g_1$ and $\xi_N = g_2$. This can be done for example as follows: replace the first and last rows of the matrix A by the corresponding rows of the identity matrix. Then replace the first and last entities of the right hand side with their corresponding boundary values, i.e. $b_0 = g_1$ and $b_N = g_2$. Solve the system with this way and compare this solution to the one you got in Problem 1. Use the same right hand side $f = 2$. Compare the condition number (`cond` in Matlab) of the matrices in the two computations. Which version is most efficient to use if the linear system is solved using an iterative solver?

Problem 5. Consider the equation $-u'' = \exp(-1000\,(x-0.5)^2)$ in $-1 < x < 1$, $u(-1) = u(1) = 0$. Write a function that assembles the mass matrix and compute the discrete Laplacian $\Delta_h u$ using the algorithm discussed in above. For this you can copy your code `my_stiffness_matrix_assembler.m` into a new file and modify it. Then, compute the residual of the finite element approximation. Test your code with different mesh resolutions and plot the residual for all of them. Does it decrease as mesh is refined?