Numerical Solution to Differential Equations using Matlab: Part 2: Finite Element Methods

Niall Madden

21 March 2012 (Rev 12.03.24)

1 Outline

Today we want to develop some Matlab code to implement a simple finite element methods for a one-dimensional problem:

$$-u'' + p(x)u' + r(x)u = f(x) \quad x \in \Omega := (0,1), \qquad (1)$$

and with Dirichlet boundary conditions. The code given below is for the simple case:

1. $p \equiv 0$ (this is often called the convective term, especially in computational fluid dynamics).
2. The reaction term, r, is constant;
3. Linear basis functions for the finite dimensional space. However, it's written in such a way so that it is not difficult to use higher-order polynomials.
4. The boundary conditions are of Dirichlet type. That is, we specify the value of the solution on the boundary. Alternatively, one could specify the first derivative, called a *Neumann condition*. Or some combination, giving a *Robin condition*.
5. The boundary conditions are homogeneous, i.e, $u(0) = u(1) = 0$. However, it is very easy to adjust the code to change this.
6. The mesh is uniform. But is not too challenging to extend this code so that it works for arbitrary meshes.
7. The method is implemented as a script file. It would be better to implement this as a function.
8. The error is estimated in the L_2-norm. To date, our mathematical analysis has been for energy norms: $\|u - u^h\|_B := \sqrt{B(u-u^h, u-u^h)}$.
9. The function `Integrate` is not given. (Addendum: I've now posted a simple integrator to the webpage. It uses 2-point Gaussian quadrature.).

We denote the mesh as $\Omega^N := \{x_0 < x_1 < \cdots < x_N\}$. As with finite difference methods, we denote the mesh width as $h_i = x_i - x_{i-1}$. Since here the mesh is uniform, we can write $h = h_i$.

2 The code

This section contains the code. You can download it from http://tinyurl.com/845fnul

2.1 Generic code comments

Your code should begin with a short description of what the program does and why it was written.

```
%% Solve1dFeLinear.m  21/03/2012
% Written by Niall Madden for the course on
% Mathematics of the Finite Element Method.
% An implementation of a Finite Element Method
% for a one-dimensional problem on a uniform
% mesh, with linear basis elements.
% The problem (in strong form) is:
%      -u'' + r u=f(x) on Omega=(0,1)
% u(0)=u(1)=0;
% See also Integrate.m
```

2.2 Problem specific information

The problem we'll solve is a simplified variant on (1):

$$-u'' + ru = 4e^{-x} \text{ on } (0,1), \quad u(0) = u(1) = 0. \qquad (2)$$

The exact solution is readily available, and is given in the code.

```
fprintf('A Finite Element implementation for:');
fprintf('\n    -u'' + r u=f(x) on (0,1) \n');
fprintf('using linear basis elements \n');
r=1/4; % r is constant
f=@(x)(4*exp(-x));
u=@(x)16/3*((exp(x/2).*(exp(-1)-exp(-1/2)) - ...
  exp(-x/2).*(exp(-1)-exp(1/2)))./(exp(1/2)- ...
  exp(-1/2))- exp(-x));
```

2.3 Information on the method and mesh

To find a numerical solution to (2) we need to specify the particular finite element discretization we'll be using: that is we need to choose a mesh, and some basis functions for our space. Here we are going to use a uniform mesh, so it is specified just by giving N, the number of intervals on the mesh.

```
N=2^5;
h = 1/N; % Assuming mesh is uniform
x = linspace(0,1,N+1)'; % mesh is a column vector
```

We'll use piecewise linear basis functions, and define them on a reference element $[0, h]$:

$$\phi_1(x) = 1 - \frac{x}{h} \qquad \phi_2(x) = \frac{x}{h}.$$

They are stored in a call array. I'm indexing from 1, rather than 0, since that is what Matlab supports. Of course we also need the derivatives. Although they are constant in this case, I'm representing them as functions. This is because you might like to use higher-order elements.

```
phi = {@(x)( 1-x/h );
       @(x)( x/h );};
Dphi = {@(x)( x*0-1/h );
       @(x)( x*0+1/h ); };
```

2.4 Now construct the element matrices

Every interval supports two basis functions. That means that each contributes to two rows and to two columns of the matrix. So we'll represent these contributions as a 2×2 matrix. I use `E2` to represent the contribution from second-order term, and `E0` the contribution from the zero-order term. That is,

$$E2_{ij} = \int_{x_{i-1}}^{x_i} \phi_j'\phi_i' dx. \qquad E0_{ij} = \int_{x_{i-1}}^{x_i} \phi_j\phi_i dx.$$

```
E2 = zeros(2,2);
E0 = zeros(2,2);
for i=1:2
  for j=1:2
    E2(i,j) = Integrate(@(X)(Dphi{j}(X).*...
      Dphi{i}(X)),0,h);
    E0(i,j) = Integrate(@(X)(phi{j}(X).*...
      phi{i}(X)),0,h);
  end
end
```

Note that I have not included the reaction term, r in this construct. This is because I want to use the `E0` matrix for forming the right-hand side, as is explained below.

2.5 And then build the linear system

Now we construct the linear system. Here, for purposes of exposition, I'm building the matrix one interval at a time. This is very slow, but easier to understand than using a more complicated `sparse` construct. I'm a little lazy in how the right-hand side is formed. Instead of computing either the exact integral (f, ϕ_i), or a quadrature approximation to that, I use the piecewise linear interpolant to f. That is, the ith term in the right-hand side is

$$(f, \phi_i) \approx \Big(\sum_{j=0}^{N} f(x_j)\phi_j(x), \phi_i(x) \Big) = \sum_{j=i-1}^{i+1} f(x_j)\big(\phi_j, \phi_i\big).$$

```
A = sparse(N+1, N+1);
F = zeros(N+1,1);
for i=1:N
  A(i:i+1,i:i+1) = A(i:i+1,i:i+1) + E2 + r*E0;
  F(i:i+1) = F(i:i+1) + E0*f(x(i:i+1));
end
```

2.6 Solve and display

The differential equation has homogeneous boundary conditions. They are enforced explicitly as follows:

```
U = [0; A(2:end-1, 2:end-1)\F(2:end-1); 0];
```

2.7 Finally, compute the error

As mentioned in Section 1, this code does not compute the error correctly, instead the L_2 error is given. You should adjust the code to estimate the error in the energy norm.

```
L2Error=0;
Diff = @(X)(u(X) - interp1(x,U,X));
for i=1:N
  L2Error = L2Error + ...
    Integrate(@(X)(Diff(X).^2), x(i),x(i+1));
end
L2Error = sqrt(L2Error);
fprintf('N=%6d, L2Error=%8.3d\n', N, L2Error);
```