AM466: Finite Element Method

Solutions of Homework 3

1. (15 marks) Derive explicit expressions for cubic Lagrange shape functions and sketch them for the master element $[-1, 1]$. Illustrate the form of the basis functions produced by such shape functions for a mesh consisting of two elements.

   Solution. The explicit expressions for cubic Lagrange shape functions are

$$
\begin{aligned}
p_1(\xi) &= \frac{(\xi-\xi_2)(\xi-\xi_3)(\xi-\xi_4)}{(\xi_1-\xi_2)(\xi_1-\xi_3)(\xi_1-\xi_4)}, \\
p_2(\xi) &= \frac{(\xi-\xi_1)(\xi-\xi_3)(\xi-\xi_4)}{(\xi_2-\xi_1)(\xi_2-\xi_3)(\xi_2-\xi_4)}, \\
p_3(\xi) &= \frac{(\xi-\xi_1)(\xi-\xi_2)(\xi-\xi_4)}{(\xi_3-\xi_1)(\xi_3-\xi_2)(\xi_3-\xi_4)}, \\
p_4(\xi) &= \frac{(\xi-\xi_1)(\xi-\xi_2)(\xi-\xi_3)}{(\xi_4-\xi_1)(\xi_4-\xi_2)(\xi_4-\xi_3)}.
\end{aligned}
$$

   They are sketched in Figure 1.

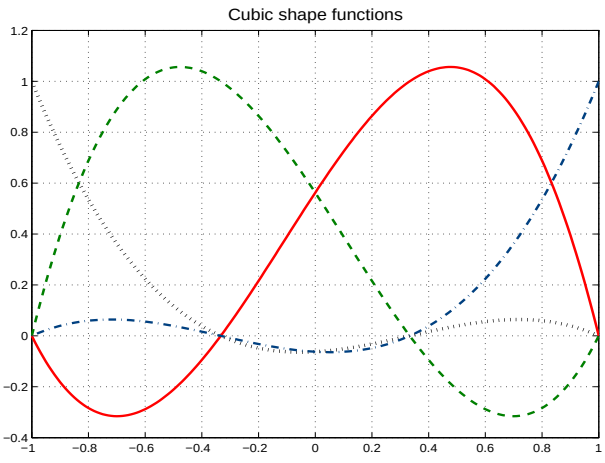

Figure 1: Cubic shape functions.

The basis functions produced by such shape functions for a mesh consisting of two elements are as follows. Figure 2 was provided generously by Mr. Rod Locke, and Figure 3 generously by Mr. Xiaomin Li. Many other students also responded to my request quickly. I greatly appreciate their help and generosity.

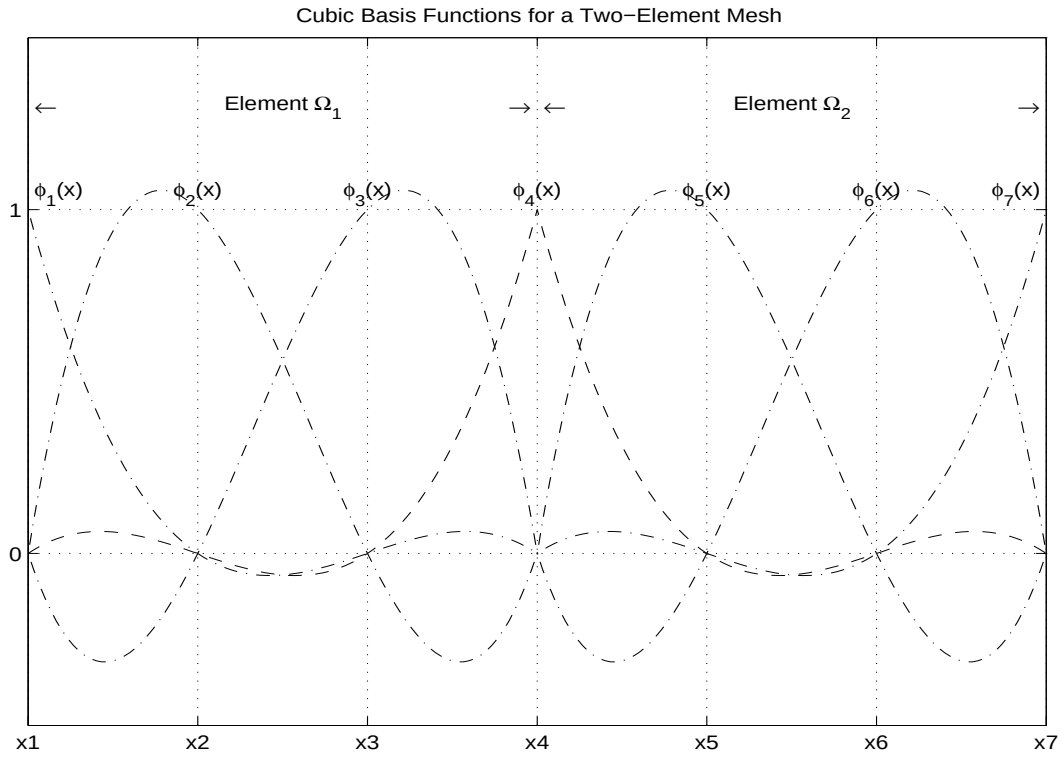

Figure 2: Cubic basis functions.

2. (35 marks) Use the cubic shape functions to revise **ode2.m** in Chapter 4 and present your revised program and the code of cubic shape functions. Then test your revised code with following boundary value problems:

   (a) The Dirichlet problem

$$y'' + y' + y - 6x - 3x^2 - x^3 = 0, \quad 1 < x < 2, \tag{1}$$
$$y(1) = 1, \quad y(2) = 8. \tag{2}$$

   (b) The Neumann problem

$$y'' + y' + y - 6x - 3x^2 - x^3 = 0, \quad 1 < x < 2, \tag{3}$$
$$y'(1) = 3, \quad y'(2) = 12. \tag{4}$$

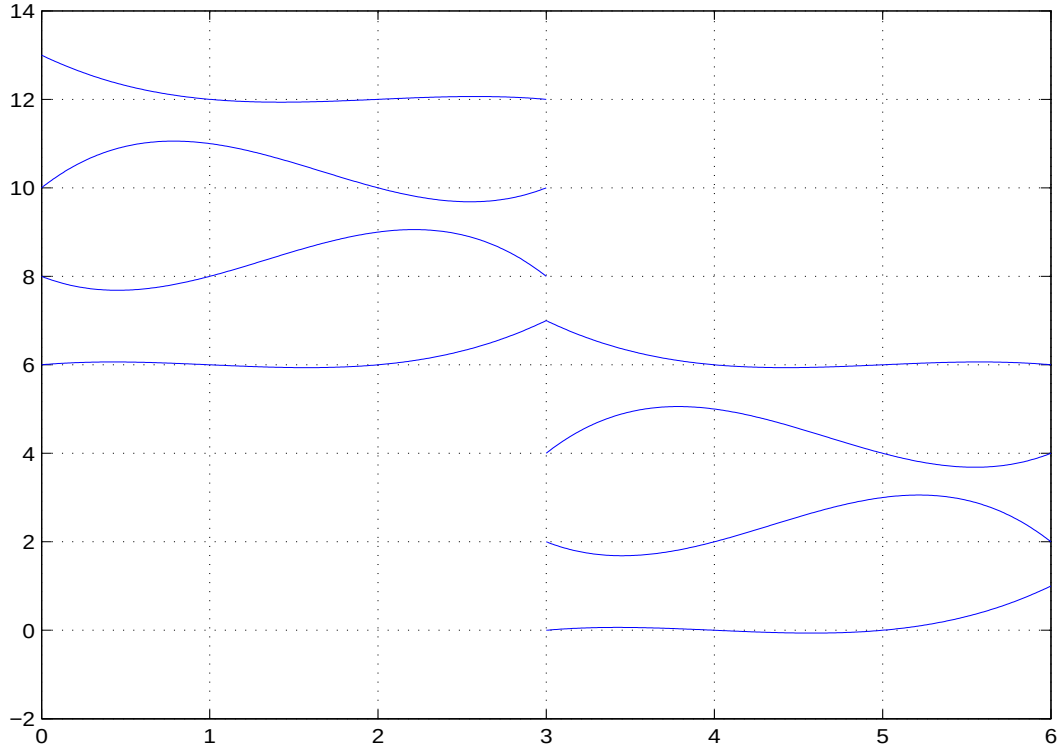

Figure 3: Cubic basis functions.

Both problems have the exact solution $y = x^3$. Present your mesh data file and coefficient file. Plot both the exact solution and your piece-wise cubic approximations. Do your piece-wise cubic approximations give this exact solution?

Solution. The revised **ode2.m** is as follows:

```
%===============
%  PROGRAM ode2cubicElement
%===============

      clear
%     ----------------
%     INPUT DATA
%     ----------------
      load cubicMESH.dat   -ASCII
      load QUAD   -ASCII

%     ----------------
```

```
%     Define MESH Data
%     ----------------
      NUMNP= cubicMESH(1,1);
      NUMEL= (NUMNP -1)/3;
      for i=1:NUMNP;
        XORD(i) = cubicMESH(i+1,1);
        NPBC(i) = cubicMESH(i+1,2);
        Y(i)    = cubicMESH(i+1,3);
        Q(i)    = cubicMESH(i+1,4);
      end

%     ----------------
%     Define QUAD Data
%     ----------------
      NQPTS = QUAD(1,1);
      for i=1:NQPTS
        GPTS(i)=QUAD(i+1,1);
        GWTS(i)=QUAD(i+1,2);
      end

%     ---------------------------
%     Initialize stiffness matrix
%     ---------------------------
      for I=1:NUMNP
        for J=1:7
          SK(I,J)=0.0;
        end
      end

%     --------------------------------
%     FORMATION OF STIFFNESS MATRIX
%      and
%     RIGHT HAND SIDE
%     --------------------------------
      IB=7;
      for I=1:NUMEL
%       ---------------------------
%       Initialize Element Variables
%       ---------------------------
        Xa = XORD(3*I-2);
        Xb = XORD(3*I+1);
```

```
        RL=Xb-Xa;
        DxDu=RL/2.0;
        DuDx=2.0/RL;

        for J=1:4
          for K=1:4
            S1(J,K)=0.0;
            S2(J,K)=0.0;
            S3(J,K)=0.0;
          end
          Qe(J)=0.0;
        end

%       -------------------------
%       Begin Gaussian Quadrature
%       -------------------------
        for J=1:NQPTS
          u = GPTS(J);
%         --------------------
%         Global coordinate of
%         current Gauss point
%         --------------------
          Xg = (Xa+Xb)/2  + (RL/2.0)*u;

%         ---------------------------
%         INCLUDE COEF.m
%         Defines: AX, BX, CX, and DX
%         ---------------------------
          COEF

%         -----------------------------
%         Gradient of shape functions at
%         current Gauss point
%         -----------------------------
          DNDX(1) = cubicSF(1,1,u)*DuDx;
          DNDX(2) = cubicSF(1,2,u)*DuDx;
          DNDX(3) = cubicSF(1,3,u)*DuDx;
          DNDX(4) = cubicSF(1,4,u)*DuDx;

%         ---------------------------
%         Element stiffness matrices
```

```
%         ---------------------------
          Wt = GWTS(J);
          for K=1:4
            for L=1:4
              S1(K,L)=S1(K,L) +Wt*DNDX(K)*AX*DNDX(L)*DxDu;
              S2(K,L)=S2(K,L) +Wt*cubicSF(0,K,u)*BX*DNDX(L)*DxDu;
              S3(K,L)=S3(K,L) +Wt*cubicSF(0,K,u)*CX*cubicSF(0,L,u)*DxDu;
            end
            Qe(K)=Qe(K) + Wt*cubicSF(0,K,u)*DX*DxDu;
          end
        end
%       --------- Quadrature now complete

%       --------------------------
%       Assemble element matrices
%       into global matrix
%       --------------------------
        K1=3*(I-1);
        L0=4;
        for K=1:4
          L0=L0-1;
          K1=K1+1;
          L1=L0;
          for L=1:4
            L1=L1+1;
            SK(K1,L1)=SK(K1,L1) +S1(K,L)-S2(K,L)-S3(K,L);
          end
          Q(K1) =  Q(K1) + Qe(K);
        end
      end
%     ---- Global Matrices are assembled

%     -------------------------
%     BOUNDARY CONDITIONS
%     -------------------------
      B = 0;
      for I=1:NUMNP
        for J=1:7
          B = B+abs(SK(I,J));
        end
      end
```

```
      B = B*(1.0E+04);
      for I=1:NUMNP
        if NPBC(I) == 1
          SK(I,4)=B;
          Q(I)=Y(I)*B;
        end
      end
%     -------------------------
%     CALL EQUATION SOLVER
%     -------------------------
      Y =  nGAUSS(SK,Q,NUMNP,IB);

%     ----------------------
%     OUTPUT DATA
%     ----------------------
      save Y Y -ASCII

%     plot the solution
      hold on
      for I=1:NUMEL
          L_I=XORD(3*I+1)-XORD(3*I-2);
          h_I= L_I/10;
          x_I= XORD(3*I-2):h_I:XORD(3*I+1);
          u_I= (2*x_I-XORD(3*I+1)-XORD(3*I-2))/L_I;
          y_I =Y(3*I-2)* cubicSF(0,1,u_I)+Y(3*I-1)* cubicSF(0,2,u_I)...
              +Y(3*I)* cubicSF(0,3,u_I)+Y(3*I+1)* cubicSF(0,4,u_I);
          app=plot(x_I,y_I, '-r', 'linewidth', 2);
          grid on;
      end
      title(' Results from ode2cubicElement.m')
      xlabel(' Independent variable')
      ylabel(' Dependent variable')

      x = min(XORD):0.01:max(XORD);
      y = x.^3;
      exact=plot(x,y, '-.k', 'linewidth', 2);
      legend([app exact], 'Approximate solution', 'Exact solution');

      hold off;
```

The code of the cubic shape functions are

```
     function s  = cubicSF(D,node,u)
%-------------------------------------------------------------
%    Shape Functions
%       D = derivative
%         = 0 function itself
%         = 1 first derivative of function
%    node = nodal point number (1 or 2)
%       u = local coordinate
%-------------------------------------------------------------
       u_1=-1;
       u_2=-1/3;
       u_3=1/3;
       u_4=1;
       if D == 0   %  Calculate shape function
           if node == 1
              s = (u-u_2).*(u-u_3).*(u-u_4)./((u_1-u_2).*(u_1-u_3).*(u_1-u_4));
           elseif node == 2
              s =(u-u_1).*(u-u_3).*(u-u_4)./((u_2-u_1).*(u_2-u_3).*(u_2-u_4));
           elseif node == 3
              s = (u-u_1).*(u-u_2).*(u-u_4)./((u_3-u_1).*(u_3-u_2).*(u_3-u_4));
           elseif node == 4
              s = (u-u_1).*(u-u_2).*(u-u_3)./((u_4-u_1).*(u_4-u_2).*(u_4-u_3));
           else
                error('Error #1 in SF function')
           end

       elseif D == 1   %  Calculate derivative shape function
          if node == 1
              s = ( (u-u_3).*(u-u_4)+(u-u_2).*(u-u_4)+(u-u_2).*(u-u_3) )...
                  ./((u_1-u_2).*(u_1-u_3).*(u_1-u_4));
           elseif node == 2
              s = ( (u-u_3).*(u-u_4) +(u-u_1).*(u-u_4)+(u-u_1).*(u-u_3) )...
                  ./((u_2-u_1).*(u_2-u_3).*(u_2-u_4));
           elseif node == 3
              s =( (u-u_2).*(u-u_4)+(u-u_1).*(u-u_4)+(u-u_1).*(u-u_2)) ...
                  ./((u_3-u_1).*(u_3-u_2).*(u_3-u_4));
           elseif node == 4
              s =( (u-u_2).*(u-u_3)+(u-u_1).*(u-u_3)+(u-u_1).*(u-u_2) )...
                  ./((u_4-u_1).*(u_4-u_2).*(u_4-u_3));
          else
```

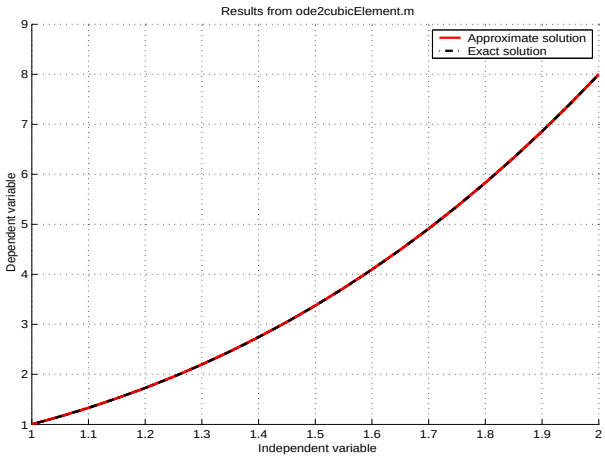

Figure 4: Exact and approximate solution of the Dirichelt problem.

```
            error('Error #2 in SF function');
      end
   else
      error('Error #3 in SF function')
   end
```

(a) The mesh data file for the Dirichlet problem is

```
%  MESH data file
%-------------------------------------------
% NUMNP  + 3 dummy numbers
%-------------------------------------------
   7       0     0     0
%-------------------------------------------
% XORD  NPBC     U     Q
%-------------------------------------------
   1       1     1     0
   1.2     0     0     0
   1.4     0     0     0
   1.5     0     0     0
   1.7     0     0     0
   1.9     0     0     0
   2       1     8     0
```

The coefficient INCLUDE file is

```
%=============================================
%    COEF.m  include file
%
%    Coefficients for ode2cubicElement.m
%=============================================

      AX = 1;
      BX = 1;
      CX = 1;
      DX = -6*Xg-3*Xg^2-Xg^3;
```

The exact solution and piece-wise cubic approximation are plotted in Figure 4, which shows that they are equal since the cubic approximation contains the exact solution.

(b) The mesh data file for the Neumann problem is

```
   %  MESH data file
%-------------------------------------------
% NUMNP  + 3 dummy numbers
%-------------------------------------------
   7       0     0     0
%-------------------------------------------
% XORD  NPBC     U     Q
%-------------------------------------------
   1       0     0     -3
   1.2     0     0     0
   1.4     0     0     0
   1.5     0     0     0
   1.7     0     0     0
   1.9     0     0     0
   2       0     0     12
```

The exact solution and piece-wise cubic approximation are plotted in Figure 5, which shows that they are equal since the cubic approximation contains the exact solution.

3. (10 marks) Run the linear finite element code **ode2.m** to solve the above Dirichlet problem and compare this result with your cubic finite element approximation. What conclusions can you draw?

   Solution. The linear finite element and cubic finite element approximations are plotted in Figure 6. This example shows that as the degree of polynomials increases, the accuracy of the approximate solution increases. However, there is a price to pay for such accuracy. As the degree increases, the bandwidth of the resulting stiffness matrix

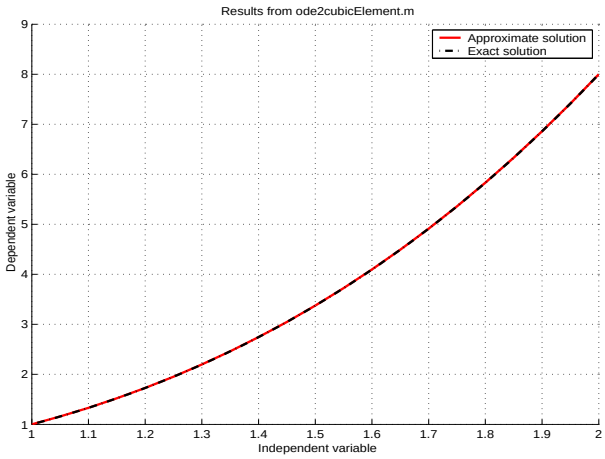

Figure 5: Exact and approximate solution of the Neumann problem.

increases and thus, in general, so does the computational effort required in solving the final system of equations.

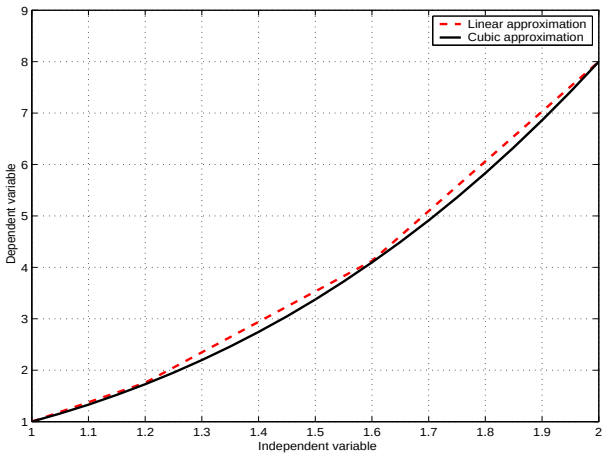

Figure 6: The linear and cubic approximations of the Dirichlet problem.