```
# include <stdlib.h>
# include <stdio.h>
# include <time.h>

int main ( void );
void assemble ( double adiag[], double aleft[], double arite[], double f[],
  double h[], int indx[], int nl, int node[], int nu, int nquad, int nsub,
  double ul, double ur, double xn[], double xquad[] );
double ff ( double x );
void geometry ( double h[], int ibc, int indx[], int nl, int node[], int nsub,
  int *nu, double xl, double xn[], double xquad[], double xr );
void init ( int *ibc, int *nquad, double *ul, double *ur, double *xl,
  double *xr );
void output ( double f[], int ibc, int indx[], int nsub, int nu, double ul,
  double ur, double xn[] );
void phi ( int il, double x, double *phii, double *phiix, double xleft,
  double xrite );
double pp ( double x );
void prsys ( double adiag[], double aleft[], double arite[], double f[],
  int nu );
double qq ( double x );
void solve ( double adiag[], double aleft[], double arite[], double f[],
  int nu );
void timestamp ( void );

/******************************************************************************/

int main ( void )

/******************************************************************************/
/*
  Purpose:

    MAIN is the main program for FEM1D.

  Discussion:

    FEM1D solves a one dimensional ODE using the finite element method.

    The differential equation solved is

      - d/dX (P dU/dX) + Q U  =  F

    The finite-element method uses piecewise linear basis functions.

    Here U is an unknown scalar function of X defined on the
    interval [XL,XR], and P, Q and F are given functions of X.

    The values of U or U' at XL and XR are also specified.

    The interval [XL,XR] is "meshed" with NSUB+1 points,

    XN(0) = XL, XN(1)=XL+H, XN(2)=XL+2*H, ..., XN(NSUB)=XR.

    This creates NSUB subintervals, with interval number 1
    having endpoints XN(0) and XN(1), and so on up to interval
    NSUB, which has endpoints XN(NSUB-1) and XN(NSUB).

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:
```

```
    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    double ADIAG(NU), the "diagonal" coefficients.  That is, ADIAG(I) is
    the coefficient of the I-th unknown in the I-th equation.

    double ALEFT(NU), the "left hand" coefficients.  That is, ALEFT(I)
    is the coefficient of the (I-1)-th unknown in the I-th equation.
    There is no value in ALEFT(1), since the first equation
    does not refer to a "0-th" unknown.

    double ARITE(NU).
    ARITE(I) is the "right hand" coefficient of the I-th
    equation in the linear system.  ARITE(I) is the coefficient
    of the (I+1)-th unknown in the I-th equation.  There is
    no value in ARITE(NU) because the NU-th equation does not
    refer to an "NU+1"-th unknown.

    double F(NU).
    ASSEMBLE stores into F the right hand side of the linear
    equations.
    SOLVE replaces those values of F by the solution of the
    linear equations.

    double H(NSUB)
    H(I) is the length of subinterval I.  This code uses
    equal spacing for all the subintervals.

    int IBC.
    IBC declares what the boundary conditions are.
    1, at the left endpoint, U has the value UL,
       at the right endpoint, U' has the value UR.
    2, at the left endpoint, U' has the value UL,
       at the right endpoint, U has the value UR.
    3, at the left endpoint, U has the value UL,
       and at the right endpoint, U has the value UR.
    4, at the left endpoint, U' has the value UL,
       at the right endpoint U' has the value UR.

    int INDX[NSUB+1].
    For a node I, INDX(I) is the index of the unknown
    associated with node I.
    If INDX(I) is equal to -1, then no unknown is associated
    with the node, because a boundary condition fixing the
    value of U has been applied at the node instead.
    Unknowns are numbered beginning with 1.
    If IBC is 2 or 4, then there is an unknown value of U
    at node 0, which will be unknown number 1.  Otherwise,
    unknown number 1 will be associated with node 1.
    If IBC is 1 or 4, then there is an unknown value of U
    at node NSUB, which will be unknown NSUB or NSUB+1,
    depending on whether there was an unknown at node 0.

    int NL.
    The number of basis functions used in a single
    subinterval.  (NL-1) is the degree of the polynomials
    used.  For this code, NL is fixed at 2, meaning that
    piecewise linear functions are used as the basis.

    int NODE[NL*NSUB].
```

```
    For each subinterval I:
    NODE[0+I*2] is the number of the left node, and
    NODE[1+I*2] is the number of the right node.

    int NQUAD.
    The number of quadrature points used in a subinterval.
    This code uses NQUAD = 1.

    int NSUB.
    The number of subintervals into which the interval [XL,XR] is broken.

    int NU.
    NU is the number of unknowns in the linear system.
    Depending on the value of IBC, there will be NSUB-1,
    NSUB, or NSUB+1 unknown values, which are the coefficients
    of basis functions.

    double UL.
    If IBC is 1 or 3, UL is the value that U is required
    to have at X = XL.
    If IBC is 2 or 4, UL is the value that U' is required
    to have at X = XL.

    double UR.
    If IBC is 2 or 3, UR is the value that U is required
    to have at X = XR.
    If IBC is 1 or 4, UR is the value that U' is required
    to have at X = XR.

    double XL.
    XL is the left endpoint of the interval over which the
    differential equation is being solved.

    double XN(0:NSUB).
    XN(I) is the location of the I-th node.  XN(0) is XL,
    and XN(NSUB) is XR.

    double XQUAD(NSUB)
    XQUAD(I) is the location of the single quadrature point
    in interval I.

    double XR.
    XR is the right endpoint of the interval over which the
    differential equation is being solved.
*/
{
# define NSUB 80000
# define NL 20

  double adiag[NSUB+1];
  double aleft[NSUB+1];
  double arite[NSUB+1];
  double f[NSUB+1];
  double h[NSUB];
  int ibc;
  int indx[NSUB+1];
  int node[NL*NSUB];
  int nquad;
  int nu;
  double ul;
  double ur;
  double xl;
  double xn[NSUB+1];
  double xquad[NSUB];
  double xr;
```

```
  timestamp ( );

  printf ( "\n" );
  printf ( "FEM1D\n" );
  printf ( "  C version\n" );
  printf ( "\n" );
  printf ( "  Solve the two-point boundary value problem\n" );
  printf ( "\n" );
  printf ( "  - d/dX (P dU/dX) + Q U  =  F\n" );
  printf ( "\n" );
  printf ( "  on the interval [XL,XR], specifying\n" );
  printf ( "  the value of U or U' at each end.\n" );
  printf ( "\n" );
  printf ( "  The interval [XL,XR] is broken into NSUB = %d subintervals\n", NSUB );
  printf ( "  Number of basis functions per element is NL = %d\n", NL );
/*
  Initialize the data.
*/
  init ( &ibc, &nquad, &ul, &ur, &xl, &xr );
/*
  Compute the geometric quantities.
*/
  geometry ( h, ibc, indx, NL, node, NSUB, &nu, xl, xn, xquad, xr );
/*
  Assemble the linear system.
*/
  assemble ( adiag, aleft, arite, f, h, indx, NL, node, nu, nquad,
    NSUB, ul, ur, xn, xquad );
/*
  Print out the linear system.
*/
  prsys ( adiag, aleft, arite, f, nu );
/*
  Solve the linear system.
*/
  solve ( adiag, aleft, arite, f, nu );
/*
  Print out the solution.
*/
  output ( f, ibc, indx, NSUB, nu, ul, ur, xn );
/*
  Terminate.
*/
  printf ( "\n" );
  printf ( "FEM1D:\n" );
  printf ( "  Normal end of execution.\n" );

  printf ( "\n" );
  timestamp ( );

  return 0;
# undef NL
# undef NSUB
}
/******************************************************************************/

void assemble ( double adiag[], double aleft[], double arite[], double f[],
  double h[], int indx[], int nl, int node[], int nu, int nquad, int nsub,
  double ul, double ur, double xn[], double xquad[] )

/******************************************************************************/
/*
  Purpose:
```

```
    ASSEMBLE assembles the matrix and right-hand-side of the linear system.

  Discussion:

    The linear system has the form:

      K * C = F

    that is to be solved for the coefficients C.

    Numerical integration is used to compute the entries of K and F.

    Note that a 1 point quadrature rule, which is sometimes used to
    assemble the matrix and right hand side, is just barely accurate
    enough for simple problems.  If you want better results, you
    should use a quadrature rule that is more accurate.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Output, double ADIAG(NU), the "diagonal" coefficients.  That is,
    ADIAG(I) is the coefficient of the I-th unknown in the I-th equation.

    Output, double ALEFT(NU), the "left hand" coefficients.  That is,
    ALEFT(I) is the coefficient of the (I-1)-th unknown in the I-th equation.
    There is no value in ALEFT(1), since the first equation
    does not refer to a "0-th" unknown.

    Output, double ARITE(NU).
    ARITE(I) is the "right hand" coefficient of the I-th
    equation in the linear system.  ARITE(I) is the coefficient
    of the (I+1)-th unknown in the I-th equation.  There is
    no value in ARITE(NU) because the NU-th equation does not
    refer to an "NU+1"-th unknown.

    Output, double F(NU).
    ASSEMBLE stores into F the right hand side of the linear
    equations.
    SOLVE replaces those values of F by the solution of the
    linear equations.

    Input, double H(NSUB)
    H(I) is the length of subinterval I.  This code uses
    equal spacing for all the subintervals.

    Input, int INDX[NSUB+1].
    For a node I, INDX(I) is the index of the unknown
    associated with node I.
    If INDX(I) is equal to -1, then no unknown is associated
    with the node, because a boundary condition fixing the
    value of U has been applied at the node instead.
    Unknowns are numbered beginning with 1.
    If IBC is 2 or 4, then there is an unknown value of U
    at node 0, which will be unknown number 1.  Otherwise,
```

```
    unknown number 1 will be associated with node 1.
    If IBC is 1 or 4, then there is an unknown value of U
    at node NSUB, which will be unknown NSUB or NSUB+1,
    depending on whether there was an unknown at node 0.

    Input, int NL.
    The number of basis functions used in a single
    subinterval.  (NL-1) is the degree of the polynomials
    used.  For this code, NL is fixed at 2, meaning that
    piecewise linear functions are used as the basis.

    Input, int NODE[NL*NSUB].
    For each subinterval I:
    NODE[0+I*2] is the number of the left node, and
    NODE[1+I*2] is the number of the right node.

    Input, int NU.
    NU is the number of unknowns in the linear system.
    Depending on the value of IBC, there will be NSUB-1,
    NSUB, or NSUB+1 unknown values, which are the coefficients
    of basis functions.

    Input, int NQUAD.
    The number of quadrature points used in a subinterval.
    This code uses NQUAD = 1.

    Input, int NSUB.
    The number of subintervals into which the interval [XL,XR] is broken.

    Input, double UL.
    If IBC is 1 or 3, UL is the value that U is required
    to have at X = XL.
    If IBC is 2 or 4, UL is the value that U' is required
    to have at X = XL.

    Input, double UR.
    If IBC is 2 or 3, UR is the value that U is required
    to have at X = XR.
    If IBC is 1 or 4, UR is the value that U' is required
    to have at X = XR.

    Input, double XL.
    XL is the left endpoint of the interval over which the
    differential equation is being solved.

    Input, double XR.
    XR is the right endpoint of the interval over which the
    differential equation is being solved.
*/
{
  double aij;
  double he;
  int i;
  int ie;
  int ig;
  int il;
  int iq;
  int iu;
  int jg;
  int jl;
  int ju;
  double phii;
  double phiix;
  double phij;
  double phijx;
```

```
  double x;
  double xleft;
  double xquade;
  double xrite;
/*
  Zero out the arrays that hold the coefficients of the matrix
  and the right hand side.
*/
  for ( i = 0; i < nu; i++ )
  {
    f[i] = 0.0;
  }
  for ( i = 0; i < nu; i++ )
  {
    adiag[i] = 0.0;
  }
  for ( i = 0; i < nu; i++ )
  {
    aleft[i] = 0.0;
  }
  for ( i = 0; i < nu; i++ )
  {
    arite[i] = 0.0;
  }
/*
  For interval number IE,
*/
  for ( ie = 0; ie < nsub; ie++ )
  {
    he = h[ie];
    xleft = xn[node[0+ie*2]];
    xrite = xn[node[1+ie*2]];
/*
  consider each quadrature point IQ,
*/
    for ( iq = 0; iq < nquad; iq++ )
    {
      xquade = xquad[ie];
/*
  and evaluate the integrals associated with the basis functions
  for the left, and for the right nodes.
*/
      for ( il = 1; il <= nl; il++ )
      {
        ig = node[il-1+ie*2];
        iu = indx[ig] - 1;

        if ( 0 <= iu )
        {
          phi ( il, xquade, &phii, &phiix, xleft, xrite );
          f[iu] = f[iu] + he * ff ( xquade ) * phii;
/*
  Take care of boundary nodes at which U' was specified.
*/
          if ( ig == 0 )
          {
            x = 0.0;
            f[iu] = f[iu] - pp ( x ) * ul;
          }
          else if ( ig == nsub )
          {
            x = 1.0;
            f[iu] = f[iu] + pp ( x ) * ur;
          }
/*
```

```
  Evaluate the integrals that take a product of the basis
  function times itself, or times the other basis function
  that is nonzero in this interval.
*/
          for ( jl = 1; jl <= nl; jl++ )
          {
            jg = node[jl-1+ie*2];
            ju = indx[jg] - 1;

            phi ( jl, xquade, &phij, &phijx, xleft, xrite );

            aij = he * ( pp ( xquade ) * phiix * phijx
                       + qq ( xquade ) * phii  * phij   );
/*
  If there is no variable associated with the node, then it's
  a specified boundary value, so we multiply the coefficient
  times the specified boundary value and subtract it from the
  right hand side.
*/
            if ( ju < 0 )
            {
              if ( jg == 0 )
              {
                f[iu] = f[iu] - aij * ul;
              }
              else if ( jg == nsub )
              {
                f[iu] = f[iu] - aij * ur;
              }
            }
/*
  Otherwise, we add the coefficient we've just computed to the
  diagonal, or left or right entries of row IU of the matrix.
*/
            else
            {
              if ( iu == ju )
              {
                adiag[iu] = adiag[iu] + aij;
              }
              else if ( ju < iu )
              {
                aleft[iu] = aleft[iu] + aij;
              }
              else
              {
                arite[iu] = arite[iu] + aij;
              }
            }
          }
        }
      }
    }
  }
  return;
}
/******************************************************************************/

double ff ( double x )

/******************************************************************************/
/*
  Purpose:

    FF evaluates the right hand side function.
```

```
  Discussion:

    This routine evaluates the function F(X) in the differential equation.

      -d/dx (p du/dx) + q u  =  f

    at the point X.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    John Burkardt

  Parameters:

    Input, double X, the argument of the function.

    Output, double FF, the value of the function.
*/
{
  double value;

  value = 0.0;

  return value;
}
/******************************************************************************/

void geometry ( double h[], int ibc, int indx[], int nl, int node[], int nsub,
  int *nu, double xl, double xn[], double xquad[], double xr )

/******************************************************************************/
/*
  Purpose:

    GEOMETRY sets up the geometry for the interval [XL,XR].

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Output, double H(NSUB)
    H(I) is the length of subinterval I.  This code uses
    equal spacing for all the subintervals.

    Input, int IBC.
    IBC declares what the boundary conditions are.
    1, at the left endpoint, U has the value UL,
       at the right endpoint, U' has the value UR.
    2, at the left endpoint, U' has the value UL,
       at the right endpoint, U has the value UR.
```

```
    3, at the left endpoint, U has the value UL,
       and at the right endpoint, U has the value UR.
    4, at the left endpoint, U' has the value UL,
       at the right endpoint U' has the value UR.

    Output, int INDX[NSUB+1].
    For a node I, INDX(I) is the index of the unknown
    associated with node I.
    If INDX(I) is equal to -1, then no unknown is associated
    with the node, because a boundary condition fixing the
    value of U has been applied at the node instead.
    Unknowns are numbered beginning with 1.
    If IBC is 2 or 4, then there is an unknown value of U
    at node 0, which will be unknown number 1.  Otherwise,
    unknown number 1 will be associated with node 1.
    If IBC is 1 or 4, then there is an unknown value of U
    at node NSUB, which will be unknown NSUB or NSUB+1,
    depending on whether there was an unknown at node 0.

    Input, int NL.
    The number of basis functions used in a single
    subinterval.  (NL-1) is the degree of the polynomials
    used.  For this code, NL is fixed at 2, meaning that
    piecewise linear functions are used as the basis.

    Output, int NODE[NL*NSUB].
    For each subinterval I:
    NODE[0+I*2] is the number of the left node, and
    NODE[1+I*2] is the number of the right node.

    Input, int NSUB.
    The number of subintervals into which the interval [XL,XR] is broken.

    Output, int *NU.
    NU is the number of unknowns in the linear system.
    Depending on the value of IBC, there will be NSUB-1,
    NSUB, or NSUB+1 unknown values, which are the coefficients
    of basis functions.

    Input, double XL.
    XL is the left endpoint of the interval over which the
    differential equation is being solved.

    Output, double XN(0:NSUB).
    XN(I) is the location of the I-th node.  XN(0) is XL,
    and XN(NSUB) is XR.

    Output, double XQUAD(NSUB)
    XQUAD(I) is the location of the single quadrature point
    in interval I.

    Input, double XR.
    XR is the right endpoint of the interval over which the
    differential equation is being solved.
*/
{
  int i;
/*
  Set the value of XN, the locations of the nodes.
*/
  printf ( "\n" );
  printf ( "  Node      Location\n" );
  printf ( "\n" );
  for ( i = 0; i <= nsub; i++ )
  {
```

```
    xn[i]  =  ( ( double ) ( nsub - i ) * xl
              + ( double )          i   * xr )
              / ( double ) ( nsub );
    printf ( "  %8d  %14f \n", i, xn[i] );
  }
/*
  Set the lengths of each subinterval.
*/
  printf ( "\n" );
  printf ( "Subint    Length\n" );
  printf ( "\n" );
  for ( i = 0; i < nsub; i++ )
  {
    h[i] = xn[i+1] - xn[i];
    printf ( "  %8d  %14f\n", i+1, h[i] );
  }
/*
  Set the quadrature points, each of which is the midpoint
  of its subinterval.
*/
  printf ( "\n" );
  printf ( "Subint    Quadrature point\n" );
  printf ( "\n" );
  for ( i = 0; i < nsub; i++ )
  {
    xquad[i] = 0.5 * ( xn[i] + xn[i+1] );
    printf ( "  %8d  %14f\n", i+1, xquad[i] );
  }
/*
  Set the value of NODE, which records, for each interval,
  the node numbers at the left and right.
*/
  printf ( "\n" );
  printf ( "Subint  Left Node  Right Node\n" );
  printf ( "\n" );
  for ( i = 0; i < nsub; i++ )
  {
    node[0+i*2] = i;
    node[1+i*2] = i + 1;
    printf ( "  %8d  %8d  %8d\n", i+1, node[0+i*2], node[1+i*2] );
  }
/*
  Starting with node 0, see if an unknown is associated with
  the node.  If so, give it an index.
*/
  *nu = 0;
/*
  Handle first node.
*/
  i = 0;
  if ( ibc == 1 || ibc == 3 )
  {
    indx[i] = -1;
  }
  else
  {
    *nu = *nu + 1;
    indx[i] = *nu;
  }
/*
  Handle nodes 1 through nsub-1
*/
  for ( i = 1; i < nsub; i++ )
  {
    *nu = *nu + 1;
```

```
    indx[i] = *nu;
  }
/*
  Handle the last node.
/*/
  i = nsub;

  if ( ibc == 2 || ibc == 3 )
  {
    indx[i] = -1;
  }
  else
  {
    *nu = *nu + 1;
    indx[i] = *nu;
  }

  printf ( "\n" );
  printf ( "  Number of unknowns NU = %8d\n", *nu );
  printf ( "\n" );
  printf ( "  Node  Unknown\n" );
  printf ( "\n" );
  for ( i = 0; i <= nsub; i++ )
  {
    printf ( "  %8d  %8d\n", i, indx[i] );
  }

  return;
}
/******************************************************************************/

void init ( int *ibc, int *nquad, double *ul, double *ur, double *xl,
  double *xr )

/******************************************************************************/
/*
  Purpose:

    INIT assigns values to variables which define the problem.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Output, int *IBC.
    IBC declares what the boundary conditions are.
    1, at the left endpoint, U has the value UL,
       at the right endpoint, U' has the value UR.
    2, at the left endpoint, U' has the value UL,
       at the right endpoint, U has the value UR.
    3, at the left endpoint, U has the value UL,
       and at the right endpoint, U has the value UR.
    4, at the left endpoint, U' has the value UL,
       at the right endpoint U' has the value UR.
```

```
    Output, int *NQUAD.
    The number of quadrature points used in a subinterval.
    This code uses NQUAD = 1.

    Output, double *UL.
    If IBC is 1 or 3, UL is the value that U is required
    to have at X = XL.
    If IBC is 2 or 4, UL is the value that U' is required
    to have at X = XL.

    Output, double *UR.
    If IBC is 2 or 3, UR is the value that U is required
    to have at X = XR.
    If IBC is 1 or 4, UR is the value that U' is required
    to have at X = XR.

    Output, double *XL.
    XL is the left endpoint of the interval over which the
    differential equation is being solved.

    Output, double *XR.
    XR is the right endpoint of the interval over which the
    differential equation is being solved.
*/
{
/*
  IBC declares what the boundary conditions are.
*/
  *ibc = 1;
/*
  NQUAD is the number of quadrature points per subinterval.
  The program as currently written cannot handle any value for
  NQUAD except 1.
*/
  *nquad = 1;
/*
  Set the values of U or U' at the endpoints.
*/
  *ul = 0.0;
  *ur = 1.0;
/*
  Define the location of the endpoints of the interval.
*/
  *xl = 0.0;
  *xr = 1.0;
/*
  Print out the values that have been set.
*/
  printf ( "\n" );
  printf ( "  The equation is to be solved for\n" );
  printf ( "  X greater than XL = %f\n", *xl );
  printf ( "  and less than XR = %f\n", *xr );
  printf ( "\n" );
  printf ( "  The boundary conditions are:\n" );
  printf ( "\n" );

  if ( *ibc == 1 || *ibc == 3 )
  {
    printf ( "  At X = XL, U = %f\n", *ul );
  }
  else
  {
    printf ( "  At X = XL, U' = %f\n", *ul );
  }
```

```
  if ( *ibc == 2 || *ibc == 3 )
  {
    printf ( "  At X = XR, U = %f\n", *ur );
  }
  else
  {
    printf ( "  At X = XR, U' = %f\n", *ur );
  }

  printf ( "\n" );
  printf ( "  Number of quadrature points per element is %d\n", *nquad );

  return;
}
/******************************************************************************/

void output ( double f[], int ibc, int indx[], int nsub, int nu, double ul,
  double ur, double xn[] )

/******************************************************************************/
/*
  Purpose:

    OUTPUT prints out the computed solution.

  Discussion:

    We simply print out the solution vector F, except that, for
    certain boundary conditions, we are going to have to get the
    value of the solution at XL or XR by using the specified
    boundary value.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Input, double F(NU).
    ASSEMBLE stores into F the right hand side of the linear
    equations.
    SOLVE replaces those values of F by the solution of the
    linear equations.

    Input, int IBC.
    IBC declares what the boundary conditions are.
    1, at the left endpoint, U has the value UL,
       at the right endpoint, U' has the value UR.
    2, at the left endpoint, U' has the value UL,
       at the right endpoint, U has the value UR.
    3, at the left endpoint, U has the value UL,
       and at the right endpoint, U has the value UR.
    4, at the left endpoint, U' has the value UL,
       at the right endpoint U' has the value UR.

    Input, int INDX[NSUB+1].
    For a node I, INDX(I) is the index of the unknown
```

```
    associated with node I.
    If INDX(I) is equal to -1, then no unknown is associated
    with the node, because a boundary condition fixing the
    value of U has been applied at the node instead.
    Unknowns are numbered beginning with 1.
    If IBC is 2 or 4, then there is an unknown value of U
    at node 0, which will be unknown number 1.  Otherwise,
    unknown number 1 will be associated with node 1.
    If IBC is 1 or 4, then there is an unknown value of U
    at node NSUB, which will be unknown NSUB or NSUB+1,
    depending on whether there was an unknown at node 0.

    Input, int NSUB.
    The number of subintervals into which the interval [XL,XR] is broken.

    Input, int NU.
    NU is the number of unknowns in the linear system.
    Depending on the value of IBC, there will be NSUB-1,
    NSUB, or NSUB+1 unknown values, which are the coefficients
    of basis functions.

    Input, double UL.
    If IBC is 1 or 3, UL is the value that U is required
    to have at X = XL.
    If IBC is 2 or 4, UL is the value that U' is required
    to have at X = XL.

    Input, double UR.
    If IBC is 2 or 3, UR is the value that U is required
    to have at X = XR.
    If IBC is 1 or 4, UR is the value that U' is required
    to have at X = XR.

    Input, double XN(0:NSUB).
    XN(I) is the location of the I-th node.  XN(0) is XL,
    and XN(NSUB) is XR.
*/
{
  int i;
  double u;

  printf ( "\n" );
  printf ( "  Computed solution coefficients:\n" );
  printf ( "\n" );
  printf ( "  Node    X(I)        U(X(I))\n" );
  printf ( "\n" );

  for ( i = 0; i <= nsub; i++ )
  {
/*
  If we're at the first node, check the boundary condition.
*/
    if ( i == 0 )
    {
      if ( ibc == 1 || ibc == 3 )
      {
        u = ul;
      }
      else
      {
        u = f[indx[i]-1];
      }
    }
/*
  If we're at the last node, check the boundary condition.
```

```
*/
    else if ( i == nsub )
    {
      if ( ibc == 2 || ibc == 3 )
      {
        u = ur;
      }
      else
      {
        u = f[indx[i]-1];
      }
    }
/*
  Any other node, we're sure the value is stored in F.
*/
    else
    {
      u = f[indx[i]-1];
    }

    printf ( "  %8d  %8f  %14f\n", i, xn[i], u );
  }

  return;
}
/******************************************************************************/

void phi ( int il, double x, double *phii, double *phiix, double xleft,
  double xrite )

/******************************************************************************/
/*
  Purpose:

    PHI evaluates a linear basis function and its derivative.

  Discussion:

    The evaluation is done at a point X in an interval [XLEFT,XRITE].

    In this interval, there are just two nonzero basis functions.
    The first basis function is a line which is 1 at the left
    endpoint and 0 at the right.  The second basis function is 0 at
    the left endpoint and 1 at the right.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Input, int IL, the index of the basis function.
    1, the function which is 1 at XLEFT and 0 at XRITE.
    2, the function which is 0 at XLEFT and 1 at XRITE.

    Input, double X, the evaluation point.
```

```
    Output, double *PHII, *PHIIX, the value of the
    basis function and its derivative at X.

    Input, double XLEFT, XRITE, the left and right
    endpoints of the interval.
*/
{
  if ( xleft <= x && x <= xrite )
  {
    if ( il == 1 )
    {
      *phii = ( xrite - x ) / ( xrite - xleft );
      *phiix =         -1.0 / ( xrite - xleft );
    }
    else
    {
      *phii = ( x - xleft ) / ( xrite - xleft );
      *phiix = 1.0          / ( xrite - xleft );
    }
  }
/*
  If X is outside of the interval, just set everything to 0.
*/
  else
  {
    *phii  = 0.0;
    *phiix = 0.0;
  }

  return;
}
/******************************************************************************/

double pp ( double x )

/******************************************************************************/
/*
  Purpose:

    PP evaluates the function P in the differential equation.

  Discussion:

    The function P appears in the differential equation as;

      - d/dx (p du/dx) + q u  =  f

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    John Burkardt

  Parameters:

    Input, double X, the argument of the function.

    Output, double PP, the value of the function.
*/
```

```
{
  double value;

  value = 1.0;

  return value;
}
/******************************************************************************/

void prsys ( double adiag[], double aleft[], double arite[], double f[],
  int nu )

/******************************************************************************/
/*
  Purpose:

    PRSYS prints out the tridiagonal linear system.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameter:

    Input, double ADIAG(NU), the "diagonal" coefficients.  That is,
    ADIAG(I) is the coefficient of the I-th unknown in the I-th equation.

    Input, double ALEFT(NU), the "left hand" coefficients.  That is, ALEFT(I)
    is the coefficient of the (I-1)-th unknown in the I-th equation.
    There is no value in ALEFT(1), since the first equation
    does not refer to a "0-th" unknown.

    Input, double ARITE(NU).
    ARITE(I) is the "right hand" coefficient of the I-th
    equation in the linear system.  ARITE(I) is the coefficient
    of the (I+1)-th unknown in the I-th equation.  There is
    no value in ARITE(NU) because the NU-th equation does not
    refer to an "NU+1"-th unknown.

    Input, double F(NU).
    ASSEMBLE stores into F the right hand side of the linear
    equations.
    SOLVE replaces those values of F by the solution of the
    linear equations.

    Input, int NU.
    NU is the number of unknowns in the linear system.
    Depending on the value of IBC, there will be NSUB-1,
    NSUB, or NSUB+1 unknown values, which are the coefficients
    of basis functions.
*/
{
  int i;

  printf ( "\n" );
  printf ( "Printout of tridiagonal linear system:\n" );
  printf ( "\n" );
```

```
  printf ( "Equation  ALEFT  ADIAG  ARITE  RHS\n" );
  printf ( "\n" );

  for ( i = 0; i < nu; i++ )
  {
    printf ( "  %8d  %14f  %14f  %14f  %14f\n",
      i + 1, aleft[i], adiag[i], arite[i], f[i] );
  }

  return;
}
/******************************************************************************/

double qq ( double x )

/******************************************************************************/
/*
  Purpose:

    QQ evaluates the function Q in the differential equation.

  Discussion:

    The function Q appears in the differential equation as:

      - d/dx (p du/dx) + q u  =  f

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    John Burkardt

  Parameters:

    Input, double X, the argument of the function.

    Output, double QQ, the value of the function.
*/
{
  double value;

  value = 0.0;

  return value;
}
/******************************************************************************/

void solve ( double adiag[], double aleft[], double arite[], double f[],
  int nu )

/******************************************************************************/
/*
  Purpose:

    SOLVE solves a tridiagonal matrix system of the form A*x = b.

  Licensing:
```

```
    This code is distributed under the GNU LGPL license.

  Modified:

    29 May 2009

  Author:

    C version by John Burkardt

  Parameters:

    Input/output, double ADIAG(NU), ALEFT(NU), ARITE(NU).
    On input, ADIAG, ALEFT, and ARITE contain the diagonal,
    left and right entries of the equations.
    On output, ADIAG and ARITE have been changed in order
    to compute the solution.
    Note that for the first equation, there is no ALEFT
    coefficient, and for the last, there is no ARITE.
    So there is no need to store a value in ALEFT(1), nor
    in ARITE(NU).

    Input/output, double F(NU).
    On input, F contains the right hand side of the linear
    system to be solved.
    On output, F contains the solution of the linear system.

    Input, int NU, the number of equations to be solved.
*/
{
  int i;
/*
  Carry out Gauss elimination on the matrix, saving information
  needed for the backsolve.
*/
  arite[0] = arite[0] / adiag[0];

  for ( i = 1; i < nu - 1; i++ )
  {
    adiag[i] = adiag[i] - aleft[i] * arite[i-1];
    arite[i] = arite[i] / adiag[i];
  }
  adiag[nu-1] = adiag[nu-1] - aleft[nu-1] * arite[nu-2];
/*
  Carry out the same elimination steps on F that were done to the
  matrix.
*/
  f[0] = f[0] / adiag[0];
  for ( i = 1; i < nu; i++ )
  {
    f[i] = ( f[i] - aleft[i] * f[i-1] ) / adiag[i];
  }
/*
  And now carry out the steps of "back substitution".
*/
  for ( i = nu - 2; 0 <= i; i-- )
  {
    f[i] = f[i] - arite[i] * f[i+1];
  }

  return;
}
/******************************************************************************/

void timestamp ( void )
```

```
/******************************************************************************/
/*
  Purpose:

    TIMESTAMP prints the current YMDHMS date as a time stamp.

  Example:

    31 May 2001 09:45:54 AM

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    24 September 2003

  Author:

    John Burkardt

  Parameters:

    None
*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  printf ( "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```