Help

```
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*  To evaluate the cubic spline interpolant at a specifie
    d point

  XB : (input) point at which interpolation is required
  N : (input) Number of points in the table
  X : (input) Array of length N, containing the abscissas
  F : (input) Array of length N, containing the function
    values at X[I]
  C : (input) Array of length 3*N containing the spline
    coefficients
    which should have been calculated using SPLINE
  DFB : (output) First derivative of spline at x=XB
  DDFB : (output) Second derivative of spline at x=XB
  IER : (output) error parameter, IER=0 if execution is su
    ccessful
    IER=24 implies XB is outside the range of table on
    higher side
    IER=25 implies XB is outside the range of table on
    lower side
    IER=201 implies N<2
  SPLEVL will be the interpolated value at x=XB

  Required functions : None
*/

double splevl(double xb, long n, double x[], double f[],
    double **c,
    double *dfb, double *ddfb, int *ier)

{
  long igh,nigh, mid;
```

```
  double r1,dx;
  static long low=-1;

  if(n<2) {*ier=201; return 0.0;}

  *ier=0;
/*  If the previous value of LOW is inadmissible, set the
    range to (0,N-1) */
  if(low<0 || low>=n-1) {low=0; igh=n-1;}
  else igh=low+1;

  while((xb<x[low] && xb<x[igh]) || (xb>x[low] && xb>x[ig
    h])) {
/*  Extend the range */
    if((xb>x[low]) == (x[n-1]>x[0])) {
/*  Extend the range on higher side */
      if(igh >= n-1) {*ier=24; low=n-2; break;}
      else {
        nigh=igh+2*(igh-low); if(n-1 < nigh) nigh=n-1;
        low=igh; igh=nigh;
      }
    }

    else {
/*  Extend the range on lower side */
      if(low <= 0) {*ier=25; igh=low+1; break;}
      else {
        nigh=low;
        low=low-2*(igh-low); if(low<0) low=0;
        igh=nigh;
      }
    }
  }


/*  Once the point is bracketed between two tabular points
     locate it by bisection */
  while((igh-low > 1) && (xb != x[low])) {
    mid=(low+igh)/2;
    if((xb<= x[mid]) == (xb<= x[low])) low=mid;
    else igh=mid;
```

```
  }

  dx=xb-x[low];
  r1=((c[low][2]*dx+c[low][1])*dx+c[low][0])*dx+f[low];
  *dfb=(3.0*c[low][2]*dx+2.*c[low][1])*dx+c[low][0];
  *ddfb=6.*c[low][2]*dx+2.*c[low][1];
  return r1;
}



/*  To calculate coefficients of cubic spline interpolati
    on with
    not-a-knot boundary conditions

  X : (input) Array of length N containing x values
  F : (input) Array of length N containing values of
    function at X[I]
    F[I] is the tabulated function value at X[I].
  N : (input) Length of table X, F
  C : (output) Array of length 3*N containing the spline
    coefficients

  Error status is returned by the value of the function SPLINE.
    0 value implies successful execution
    201 implies that N<2

  Required functions : None
*/


int spline(double x[], double f[], long n, double **c)

{
  long i,j;
  double g, c1, cn, div12, div01;

  if(n<2) return 201;
  else if(n == 2) {
/*  Use linear interpolation */
    c[0][0]=(f[1]-f[0])/(x[1]-x[0]);
```

```
    c[0][1]=0.0;
    c[0][2]=0.0;
    return 0;
  }
  else if(n == 3) {
/*  Use quadratic interpolation */
    div01=(f[1]-f[0])/(x[1]-x[0]);
    div12=(f[2]-f[1])/(x[2]-x[1]);
    c[0][2]=0.0;
    c[1][2]=0.0;
    c[0][1]=(div12-div01)/(x[2]-x[0]);
    c[1][1]=c[0][1];
    c[0][0]=div01+c[0][1]*(x[0]-x[1]);
    c[1][0]=div12+c[0][1]*(x[1]-x[2]);
          return 0;
  }
  else {
/*  Use cubic splines

  Setting up the coefficients of tridiagonal matrix */
    c[n-1][2]=(f[n-1]-f[n-2])/(x[n-1]-x[n-2]);
    for(i=n-2; i>=1; --i) {
      c[i][2]=(f[i]-f[i-1])/(x[i]-x[i-1]);
      c[i][1]=2.*(x[i+1]-x[i-1]);
/*  The right hand sides */
      c[i][0]=3.*(c[i][2]*(x[i+1]-x[i])+c[i+1][2]*(x[i]-
    x[i-1]));
    }

/*  The not-a-knot boundary conditions */
    c1=x[2]-x[0];
    c[0][1]=x[2]-x[1];
    c[0][0]=c[1][2]*c[0][1]*(2.*c1+x[1]-x[0])+c[2][2]*(x[
    1]-x[0])*(x[1]-x[0]);
    c[0][0]=c[0][0]/c1;
    cn=x[n-1]-x[n-3];
    c[n-1][1]=x[n-2]-x[n-3];
    c[n-1][0]=c[n-1][2]*c[n-1][1]*(2.*cn+x[n-1]-x[n-2]);
    c[n-1][0]=(c[n-1][0]+c[n-2][2]*(x[n-1]-x[n-2])*(x[n-1
    ]-x[n-2]))/cn;
/*  Solving the equation by Gaussian elimination */
```

```
    g=(x[2]-x[1])/c[0][1];
    c[1][1]=c[1][1]-g*c1;
    c[1][0]=c[1][0]-g*c[0][0];
    for(j=1; j<n-2; ++j) {
      g=(x[j+2]-x[j+1])/c[j][1];
      c[j+1][1]=c[j+1][1]-g*(x[j]-x[j-1]);
      c[j+1][0]=c[j+1][0]-g*c[j][0];
    }
    g=cn/c[n-2][1];
    c[n-1][1]=c[n-1][1]-g*(x[n-2]-x[n-3]);
    c[n-1][0]=c[n-1][0]-g*c[n-2][0];


/*  The back-substitution */
    c[n-1][0]=c[n-1][0]/c[n-1][1];
    for(i=n-2; i>=1; --i) c[i][0]=(c[i][0]-c[i+1][0]*(x[
    i]-x[i-1]))/c[i][1];
    c[0][0]=(c[0][0]-c[1][0]*c1)/c[0][1];

/*  Calculating the coefficients of cubic spline */
    for(i=0; i<n-1; ++i) {
      c[i][1]=(3.*c[i+1][2]-2.*c[i][0]-c[i+1][0])/(x[i+1
    ]-x[i]);
      c[i][2]=(c[i][0]+c[i+1][0]-2.*c[i+1][2])/((x[i+1]-
    x[i])*(x[i+1]-x[i]));
    }
/*  Set the coefficients for interval beyond X(N) using
    continuity
  of second derivative, although they may not be used. */
    c[n-1][1]=c[n-1][1]+3*(x[n-1]-x[n-2])*c[n-2][2];
    c[n-1][2]=0.0;
    return 0;
  }
}




/*  To draw a smooth curve passing through a set of data
    points
      using cubic spline interpolation
```

```
  NTAB : (input) Number of points in the table
  X : (input) Array of length NTAB containing X values
  F : (input) Array of length NTAB containing function val
    ues at X[I]
  C : (output) Array of length 3*NTAB which will contain
    the spline coefficients
  NP : (input) Number of points at which interpolation is
    to be calculated
  XP : (output) Array of length NP containing the x value
    s at
           NP uniformly spaced points for use in plotting
  FP : (output) Array of length NP containing interpolated
    function values at XP[I]

  Error status is returned by the value of the function SM
    OOTH.
    0 value implies successful execution
    202 implies NP<=1
    other values may be set by SPLINE

  Arrays XP and FP can be used to draw a smooth curve thro
    ugh the
  tabulated points.

  Required functions : SPLINE, SPLEVL
*/

int smooth(long ntab, double x[], double f[], double **c,
    int np, double xp[], double fp[])
{
  int i,ier;
  double dx,dfb,ddfb;

  i=spline(x,f,ntab,c);
  if(i>100) return i;

  if(np <= 1) return 202;

  dx=(x[ntab-1]-x[0])/(np-1);
  for(i=0; i<np; ++i) {
```

```
    xp[i]=x[0]+dx*i;
    fp[i]=splevl(xp[i], ntab, x, f, c, &dfb, &ddfb, &ier)
    ;
  }
  return 0;
}

int smoothmod(long ntab, double x[], double f[], double **
    c, int np, double xp[], double fp[])
{
  int i,ier;
  double dfb,ddfb;

  i=spline(x,f,ntab,c);
  if(i>100) return i;

  if(np <= 1) return 202;

  //dx=(x[ntab-1]-x[0])/(np-1);
  for(i=0; i<np; ++i) {
  //  xp[i]=x[0]+dx*i;
    fp[i]=splevl(xp[i], ntab, x, f, c, &dfb, &ddfb, &ier)
    ;
  }
  return 0;
}

double smoothscalar(long ntab, double x[], double f[],
    double **c, double xp)
{
  int i,ier ;
  double fp, dfb,ddfb;

  i=spline(x,f,ntab,c);
  if(i>100) return i;

  fp=splevl(xp, ntab, x, f, c, &dfb, &ddfb, &ier);

  return fp;
}
```

```
#endif //PremiaCurrentVersion
```

References