An Introduction to the Piecewise Parabolic Method (PPM) Library I. Interpolation and Hydrodynamics Routines

Paul Ralph Woodward,[†] *B Edgar,*
and S. E. Anderson,[†]

Laboratory for Computational Science and Engineering
Department of Astronomy, School of Physics and Astronomy
116 Church St. SE.
University of Minnesota
Minneapolis, Minnesota 55455

ABSTRACT

This paper is meant to be a practical guide to using the PPM Library for those who already have some familiarity with computational fluid dynamics. It consists of explanations of the various subroutines, along with the FORTRAN code for several basic examples.

1. Introduction

This is meant as an introductory guide to the PPMLIB library routines for interpolation and computational fluid dynamics. These routines consist of of algorithms presented in modular form, adapted for efficient computation on parallel computing platforms. It is based on the Piecewise Parabolic Method (PPM) (Woodward and Colella 1984, Colella and Woodward 1984, Woodward 1986) and incorporates recent improvements developed at the University of Minnesota. It is hoped that this library will allow access to the algorithms to a broad community of investigators, without themselves having to become experts in CFD algorithm development or the intricacies of the modern parallel computing environment. PPMLIB is constructed in a manner analogous to familiar graphics or mathematical libraries. Each algorithm is packaged in a modular form, callable from larger programs.

The Piecewise Parabolic Method was originally developed to simulate the unsteady flows with strong shocks often found in astrophysics, such as in supernova explosions and supersonic jets. It used the ideas of Bram van Leer (1979) to extend the method of Godunov (1959) to a higher formal order of accuracy. Its success comes from the combination of several techniques. Given zone averaged data, a high-order interpolation is used to construct descriptions of the relevant variables everywhere in space, providing accuracy in regions of smooth flow. Monotonicity constraints are applied to ensure that the advection of monotone data produces monotone results.

Several additional features of PPM contribute to its effectiveness. Discontinuity detection algorithms are used to modify the interpolation in zones near discontinuities in order to better represent these structures. This helps to prevent numerical viscosity from smearing discontinuities as they move through the computational grid. Shock detection is also performed so that an appropriate amount of localized numerical dissipation is added only to those zones in shocks.

Multi-dimensional flows are solved by applying a one-dimensional scheme in alternation according to the directional-splitting algorithm of Strang(1968). For example, in two-dimensional Cartesian

‡ Minnesota Supercomputer Institute

geometry, this would require performing a sequence of one-dimensional passes, $x - y - y - x$. This symmetrized order provides second-order accuracy. Most of the routines in PPMLIB are designed to operate on one-dimensional data, which may be subsets of multi-dimensional data. This strategy permits the construction of highly accurate, sophisticated algorithms.

Thus PPM can be broken down into a number of smaller, independent modules. By so doing, relevant pieces of PPM may be incorporated into other codes not requiring the full PPM hydrodynamic algorithm. For example, PPM advection has been incorporated into meteorological codes, and has been used separately at the University of Illinois in an MHD code to advect the magnetic vector potential.

This paper is meant to be a guide to using the individual routines which comprise PPMLIB, consisting of 3 parts. Part 1 introduces some basic concepts relevant to the discussion of the library routines. Also presented here are two programs which create initial data for the interpolation examples. In part 2 are presented the PPMLIB interpolative routines which lie at the heart of PPM. Part 2 includes several examples of how the interpolative routines may be combined to create programs which map quantities from one grid to another. It concludes with a discussion of how the character of PPMLIB interpolations may be customized to suit the user's specific needs.

In part 3, the PPMLIB approach to hydrodynamics is presented, first as Direct Eulerian schemes and also as Lagrangian plus remap formulations. Several complete working examples of PPM schemes are presented, in a graded sequence, to serve both as sources of complete hydrodynamical algorithms and as templates for customized algorithms of the user's own devising.

Since it is meant for those active in computational simulations, the emphasis is placed on hands-on experience. Each section will consist of a brief explanation of the topic, followed by several examples. The reader is strongly urged to construct relevant examples from his own field.

a. Language

The programming language used is FORTRAN, including the extensions for FORTRAN 90. Actual source code will be printed in a fixed width font to distinguish it from the explanatory text. Here follows a short sample program as illustration.

```
      PROGRAM BETTER_THAN_C
      DO I = 1, 10
         PRINT *,' Hello Universe!'
      ENDDO
      END ! PROGRAM BETTER_THAN_C
```

The output from this program is

```
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
  Hello Universe!
```

It is assumed that the reader has some experience in FORTRAN programming and understands the rudiments of text editing, compiling, debugging, and running programs in whatever environments are available.

Many of the examples in this guide create data intended to be examined graphically. In such cases, the output has been designed to be compatible with a plotting package available from Lick Observatory called MONGO. If the reader uses another package, then the output sections of these examples may have to be modified to conform. It is assumed that the reader is familiar with whatever plotting package he has available, and is capable of making such changes.

b. Computational Grid

The PPMLIB routines are designed to operate on data which have been discretized and sampled in some manner on a computational grid. The computational grid is defined by the locations of the boundaries, or *zone interfaces*, delineating the the individual computational zones. Thus each grid zone i is specified by a left and a right zone zone interface, with locations $x_{i-1/2}$ and $x_{i+1/2}$, respectively. A grid is spatially contiguous, so the left interface of zone $i+1$ is the right interface of zone i: $x_{(i+1)-1/2} = x_{i+1/2}$. Thus a grid on N zones is be defined by $N+1$ left zone interfaces, $x_{i-1/2}$. A uniform grid is one on which the zone widths, Δx_i,

$$\Delta x_i = x_{i+1/2} - x_{i-1/2}, \qquad 1.1$$

are equal. On a non-uniform grid, Δx_i vary. A discontinuous grid is a non-uniform grid with an abrupt change in zone widths. For example:

$$\Delta x_i = \begin{cases} 1.0 & \text{if } i < 64 \\ 2.0 & \text{if } i \geq 64 \end{cases}$$

These concepts will be illustrated in the following simple FORTRAN subroutines creating examples of these three kinds of grids are given here. These subroutines will also be used later when illustrating specific uses of some of the PPMLIB interpolative routines.

```
      SUBROUTINE CREATE_UNIFORM_GRID ( XL, X_BEGIN, X_LENGTH, N)
c
C  WHAT               This routine will create a non-uniform computation grid
c                     of N zones beginning at location X_BEGIN and has a
c                     total length X_LENGTH Thus the width of each zone
c                     is X_LENGTH / N
c
c                     The grid is defined by the locations of the lefthand
c                     zone interfaces XL.  Thus zone I would have its left
c                     edge at XL(I) and its right edge at XL(I+1).  Therefore
c                     a grid of N zones requires N+1 XL values.
c
c  DIMENSIONS
c     output         It will be assumed that the dimension of the array XL
c                    is [1:N+1].
c
C  DATA
C    input          Input scalers are:
c                       X_BEGIN:  the starting value of the left edge of the
c                                 grid.  That is XL(1) = X_BEGIN
c                       X_LENGTH: the total length of the grid.
c                                 That is XL(N+1) = X_BEGIN + X_LENGTH
c                       N:        the number of zone in the grid.
C     output        Output arrays are:
c                       XL:       the locations of the lefthand zone interfaces
c
c
c
c
c
      REAL XL(1:N), X_BEGIN, X_LENGTH, DX

      DX = X_LENGTH / FLOAT(N)

      XL(1) = X_BEGIN
      DO I = 2,N+1
         XL(I) = XL(I-1) + DX
      ENDDO

      RETURN
      END  ! SUBROUTINE CREATE_UNIFORM_GRID
```

```
      SUBROUTINE CREATE_NONUNIFORM_GRID ( XL, X_BEGIN, X_LENGTH, N)

c
C  WHAT               This routine will create a non-uniform computation grid
c                     of N zones beginning at location X_BEGIN and which has a
c                     total length X_LENGTH.
c
c                     The grid is defined by the locations of the lefthand
c                     zone interfaces XL.  Thus zone I would have its left
c                     edge at XL(I) and its right edge at XL(I+1).  Therefore
c                     a grid of N zones requires N+1 XL values.
c
c                     A sinusoid is used to vary the zone interface locations
c                     from those of a uniform grid of the same size and
c                     dimensions.
c
c  DIMENSIONS
```

```
c     output        It will be assumed that the dimension of the array XL
c                   is [1:N+1].
c
C  DATA
C    input          Input scalers are:
c                       X_BEGIN:  the starting value of the left edge of the
c                                 grid.  That is XL(1) = X_BEGIN
c                       X_LENGTH: the total length of the grid.
c                                 That is XL(N+1) = X_BEGIN + X_LENGTH
c                       N:        the number of zone in the grid.
C     output        Output arrays are:
c                       XL:       the locations of the lefthand zone interfaces
c
c
c
c
c

      REAL XL(1), X_BEGIN, X_LENGTH, DX, SINFAC, SINAMP

      SINFAC = 6.28318530717958 / FLOAT(N)
      SINAMP = 0.75

      DX = X_LENGTH / FLOAT(N)

      XL(1) = X_BEGIN
      DO I = 2,N+1
         XL(I) = XL(I-1) + DX * ( 1.0 + SINAMP * SIN( SINFAC * FLOAT(I-1) ) )
      ENDDO

      RETURN
      END  ! SUBROUTINE CREATE_NONUNIFORM_GRID
```

```
      SUBROUTINE CREATE_DISCONTINUOUS_GRID ( XL, X_BEGIN, X_LENGTH, N)
c
c
C  WHAT               This routine will create a uniform computation grid
c                     of N zones  has a discontinuity in zone width at
c                     zone N/2.  The grid starts at location X_BEGIN and has
c                     total length X_LENGTH Thus the width of each zone
c                     a is X_LENGTH / N. Grid zones to the left of the
c                     discontinuity are half the size of zones to the right
c                     of the discontinuity.
c
c                     The grid is defined by the locations of the lefthand
c                     zone interfaces XL.  Thus zone I would have its left
c                     edge at XL(I) and its right edge at XL(I+1).  Therefore
c                     a grid of N zones requires N+1 XL values.
c
c  DIMENSIONS
c     output        It will be assumed that the dimension of the array XL
c                   is [1:N+1].
c
C  DATA
C    input          Input scalers are:
c                       X_BEGIN:  the starting value of the left edge of the
c                                 grid.  That is XL(1) = X_BEGIN
c                       X_LENGTH: the total length of the grid.
c                                 That is XL(N+1) = X_BEGIN + X_LENGTH
c                       N:        the number of zone in the grid.
C     output        Output arrays are:
c                       XL:       the locations of the lefthand zone interfaces
c
c
c
c
c
      REAL XL(1), X_BEGIN, X_LENGTH, DX_LEFT, DX_RIGHT, SINFAC, SINAMP
c
      DX_RIGHT = 4.0 * X_LENGTH / FLOAT(3*N)
      DX_LEFT = 2.0 * X_LENGTH / FLOAT(3*N)
      N_BREAK = N/2
c
      XL(1) = X_BEGIN
      DO I = 2,N_BREAK
         XL(I) = XL(I-1) + DX_LEFT
      ENDDO
c
      DO I = N_BREAK+1,N
         XL(I) = XL(I-1) + DX_RIGHT
      ENDDO
c
      RETURN
      END  ! SUBROUTINE CREATE_DISCONTINUOUS_GRID
```

Notice that in all cases, the locations of the left zone interfaces are stored in an array called `XL` and returned to the calling routine. To fully define a grid of N zones, N+1 `XL` values are required. Each grid begins at the location `X_BEGIN`, passed from the calling routine. The ending location of the grid is at location (`X_BEGIN + XLENGTH`), where XLENGTH, the total length of the grid is also passed from the calling routine.

Subroutine CREATE_UNIFORM_GRID consists of N zones of equal width. In the next subroutine CREATE_NONUNIFORM_GRID, a sinusoid is used to smoothly vary the grid widths across the grid. The third subroutine, CREATE_DISCONTINUOUS_GRID, has a sudden change in grid widths in the middle of the grid. The use of one of these routines is illustrated in the program PRINT_GRID which calls one of these subroutines, CREATE_UNIFORM_GRID, to initialize a computational grid beginning at $x_1 = 0.0$, and ending at $x_{N+1} = 1.0$ with 64 zones. It also prints out the locations for examination.

```
      PROGRAM PRINT_GRID
c
C  WHAT               This program creates a grid or mesh of N=64 zones.
c                     The grid begins a 0.0 and has a total length of 1.0,
c                     consisting of N+1 lefthand interface locations.  The
c                     locations of the zone interfaces are then displayed.

      ALLOCATABLE :: XL(:), A_AVG(:)
      PARAMETER ( N = 64 )

      REAL XL(N+1), X_BEGIN, X_LENGTH

      X_BEGIN = 0.0
      X_LENGTH = 1.0

      CALL CREATE_UNIFORM_GRID( XL(1), X_BEGIN, X_LENGTH, N )

      WRITE (6,100)
100   FORMAT('  The computational grid is defined by:',/,
     &  6X,'I',6X,'XL(I)',/,30('_'))

      DO I=1,N+1
        WRITE (6,200) I, XL(I)
200     FORMAT (4X,I4,t10,1PE12.5)
      ENDDO

      END ! PROGRAM PRINT_GRID
```

c. Zone Averages

Many of the routines in PPMLIB operate on *zone averages*. Given a function $a(x)$, then the zone average of $a(x)$ in zone i is

$$a_i = \frac{1}{\Delta x_i} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} a(x) dx. \qquad 1.2$$

The Program CREATE_ZONE_AVERAGES_SINE uses the above grid creation subroutines to creat an initial data set consisting of zone averages of a sine wave.

$$\sin(\alpha x)$$

where the constant α is chosen so that an integral number of sine waves fits on the grid. This is selected interactively, as is the number of grid zones and type of grid. This routine, and the routine which follows will be of use later during the exploration of the PPMLIB interpolative routines.

```
          PROGRAM CREATE_ZONE_AVERAGES_SINE
c
c   This program first creates a computational grid or mesh of N
c   zones .The grid begins at 0.0 and has a total length of 1.0.
c   It consists of N+1 lefthand interface locations.
c
c   Zone averages over this grid are then found by finding the
c   average value of sin(ax) in each zone.  the coefficient a
c   is chosen such that there is an integral number of sine waves
c   on the interval (0,1).
c
c   A file "OUTPUT.DAT" is created which may be used as input for
c   subsequent example programs.

          ALLOCATABLE ::  XL(:), A_AVG(:)

c
c   The number of grid zones is selected interactively
```

```
c
      PRINT *,' Please enter desired number of grid zones (integer):'
      READ *,N
c
c   The number of period is selected interactively
c
      PRINT *,' Kindly enter desired number of periods (integer):'
      READ *,NWAVES
c
c   The type of grid (uniform, non-uniform, discontinuous) is
c                                               selected interactively
      PRINT *,' Please enter -1 for a discontinuous grid'
      PRINT *,'                0 for a non-uniform grid or'
      PRINT *,'               +1 for a uniform grid:'
      READ *,IGRID

      PRINT *,' Thank you. '
      PRINT *,'   You have requested ',NWAVES,
     ^        ' sine waves  over',N,' zones.'

          ALLOCATE (XL(N+1), A_AVG(N))

          X_BEGIN = 0.0
          X_LENGTH = 1.0


      IF (IGRID .EQ. 1 ) THEN
          PRINT *,' on a uniform grid'
          CALL CREATE_UNIFORM_GRID( XL(1), X_BEGIN, X_LENGTH, N )
      ELSE
         IF (IGRID .EQ. 0 ) THEN
             PRINT *,' on a non-uniform grid'
             CALL CREATE_NONUNIFORM_GRID( XL(1), X_BEGIN, X_LENGTH, N )
         ELSE
            IF (IGRID .EQ. -1 ) THEN
                PRINT *,' on a discontinuous-uniform grid'
               CALL CREATE_DISCONTINUOUS_GRID( XL(1), X_BEGIN,
     ^                                                 X_LENGTH, N )
            ELSE
c
c   unknown grid type
               PRINT *,' UNKNOWN GRID TYPE:',IGRID
               CALL ABORT()
            ENDIF
         ENDIF
      ENDIF


c
c   The functional form is sin(alpha*x)
c
c   The average of this function on the interval (x1,x2) is
c
c              -[cos(alpha*x2) - cos(alpha*x1)]/[alpha*(x2-x1)]
c
          ALPHA = 6.28318530717958 * FLOAT(NWAVES)
          DO I = 1,N
             XLEFT = COS( ALPHA*XL(I) )
             XRIGHT = COS( ALPHA*XL(I+1) )
             DX = XL(I+1)-XL(I)

             A_AVG(I) = - (XRIGHT - XLEFT)/(ALPHA*DX)
          ENDDO
          OPEN (UNIT=11, FILE='OUTPUT.AVG')
       write(11,50) N
50    FORMAT(1X,I10)

          WRITE (6,100)
100   FORMAT(6X,'I',6X,'XMID',10X,'A_AVG',/,40('_'))

          DO I=1,N
            XMID = 0.5 * (XL(I+1)+XL(I))
            WRITE (6,200) I, XMID, A_AVG(I)
200     FORMAT (4X,I4,t10,1P,2(E12.5,3x))
            WRITE (11,300)  XL(I), A_AVG(I)
300     FORMAT (4X,1P,2(E12.5,3x))
          ENDDO
            WRITE (11,300)  XL(N+1), A_AVG(N)
            CLOSE(11)

          DEALLOCATE (XL, A_AVG)

          END ! PROGRAM CREATE_ZONE_AVERAGES_SINE
```

This program will create a grid of N zones. The number of zones determined by user input. After the user is queried for the number of periods desired, the user is then given a choice of grid type. Space is allocated for the $N+1$ interface locations, XL and for the N zone averages, A_AVG. The appropriate left hand zone interface locations are then generated, with some checks for valid input. The zone averages are found analytically, after computing the appropriate scale factor. The results are then stored in the file OUTPUT.AVG for future use. An example of the results from the program CREATE_ZONE_AVERAGES_SINE for a single sine wave on a uniform grid of 50 zones is shown in figure

1a. In that figure, the underlying sine wave has been plotted as a dotted line for reference.

A second program, `CREATE_ZONE_AVERAGES_TANH`, will be used to create data sets that mimic discontinuous functions. It finds zone averages of a hyperbolic tangent with the functional form

$$a(x) = \tanh\left[\alpha(x - x_0)\right]. \qquad 1.3$$

In this equation the offset x_0 allows for the centering of the function and α is a scale factor. The value of α is chosen so that most of the variation in *tanh* occurs within a specified number of zones. For this it is useful to note that approximately 90% of the total variation occurs on the range

$$-1.5 \le \alpha(x - x_0) \le 1.5.$$

This range shall be adopted as the discontinuity width. Here, a *discontinuity,* is meant in the operational sense that the numerical grid is not fine enough to resolve well changes in the underlying function.

```
          PROGRAM CREATE_ZONE_AVERAGES_TANH
c
c   This program first creates a computational grid or mesh of N
c   zones .The grid begins at 0.0 and has a total length of 1.0.
c   It consists of N+1 lefthand interface locations.
c
c   Zone averages of a discontinuous-like distribution are created
c   by finding  the average value of tanh(a(x-x0)) in each zone.  the
c   offset x0 is chosen such to ensure that the function is centered,
c   and a is chosen such that 90% of the change in  tanh occurs over
c   a specified number of zones.
c
c   A file "OUTPUT.DAT" is created which may be used as input for
c   subsequent example programs.

          ALLOCATABLE ::  XL(:), A_AVG(:)

c
c   The number of grid zones is selected interactively
c
      PRINT *,' Please enter desired number of grid zones (integer):'
      READ *,N
c
c   The width of the discontinuity in zones
c
      PRINT *,' Kindly enter the approximate width of the '
      PRINT *,' discontinuity in zones'
      READ *,NWIDTH
c
c   The type of grid (uniform, non-uniform, discontinuous) is
c                                           selected interactively
      PRINT *,' Please enter -1 for a discontinuous grid'
      PRINT *,'                0 for a non-uniform grid or'
      PRINT *,'               +1 for a uniform grid:'
      READ *,IGRID


      PRINT *,' Thank you. '
      PRINT *,'   You have requested a discontinuity of approx.',
     ^        NWIDTH,' zones. The total number of grid zones is',N

          ALLOCATE (XL(N+1), A_AVG(N))

          X_BEGIN = 0.0
          X_LENGTH = 1.0


      IF (IGRID .EQ. 1 ) THEN
          PRINT *,' on a uniform grid'
          CALL CREATE_UNIFORM_GRID( XL(1), X_BEGIN, X_LENGTH, N )
      ELSE
         IF (IGRID .EQ. 0 ) THEN
             PRINT *,' on a non-uniform grid'
             CALL CREATE_NONUNIFORM_GRID( XL(1), X_BEGIN, X_LENGTH, N )
         ELSE
            IF (IGRID .EQ. -1 ) THEN
                PRINT *,' on a discontinuous-uniform grid'
               CALL CREATE_DISCONTINUOUS_GRID( XL(1), X_BEGIN,
     ^                                                  X_LENGTH, N )
            ELSE
c
c   unknown grid type
               PRINT *,' UNKNOWN GRID TYPE:',IGRID
               CALL ABORT()
            ENDIF
         ENDIF
      ENDIF


c
```

```
c   The functional form is tanh(alpha*(x-x0))
c
c   The average of this function on the interval (x1,x2) is
c
c              [ln(cosh(alpha*(x2-x0))) -
c                                 ln(cosh(alpha*(x2-x0)))]/[alpha*(x2-x1)]
c
      X0 = 0.5
c
c    90% of the variation of tanh occurs on the range (-1.5:1.5)
c
      NHALF = N/2
      NZL = NWIDTH/2
      NZR = NWIDTH-NZL
      DELTAX = XL(NHALF+NZL) - XL(NHALF-NZR)
      ALPHA = 3.0 / DELTAX
          DO I = 1,N
             XLEFT = ALOG( COSH( ALPHA*(XL(I)-X0) ) )
             XRIGHT = ALOG( COSH( ALPHA*(XL(I+1)-X0) ) )
             DX = XL(I+1)-XL(I)

             A_AVG(I) =  (XRIGHT - XLEFT)/(ALPHA*DX)
          ENDDO
          OPEN (UNIT=11, FILE='OUTPUT.AVG')
       write(11,50) N
50    FORMAT(1X,I10)

          WRITE (6,100)
100   FORMAT(6X,'I',6X,'XMID',10X,'A_AVG',/,40('_'))

          DO I=1,N
            XMID = 0.5 * (XL(I+1)+XL(I))
            WRITE (6,200) I, XMID, A_AVG(I)
200     FORMAT (4X,I4,t10,1P,2(E12.5,3x))
            WRITE (11,300)  XL(I), A_AVG(I)
300     FORMAT (4X,1P,2(E12.5,3x))
          ENDDO
            WRITE (11,300)  XL(N+1), A_AVG(N)
            CLOSE(11)

          DEALLOCATE (XL, A_AVG)

          END ! PROGRAM CREATE_ZONE_AVERAGES_TANH
```

An example from `CREATE_ZONE_AVERAGES_TANH` is shown in figure 1b. The zone averages where computed on a 20 zone non-uniform grid. The transition from -1 to 1 occurs over approximate 8 zones, producing a relatively smooth distribution of zone averages. Again, the underlying function upon which the zone averages were computed is plotted as a dotted line for reference.

Figure 2 shows a sequence of zone average data sets which will prove useful in the next section on interpolation. In this sequence of four sets, the width of the transition decreases from approximately 8 zones to 1 zone by a factor of two between each set. All are created on a 20 zone non-uniform grid. This is a graphic example of the transition of a well resolved function to a poorly resolved function. As will be illustrated later, it is often necessary to modify the interpolations as the data resolution worsens.

d. Boundary Conditions

It is common for results of a calculation at zone i to depend upon the values in neighboring zones. For example, consider this approximation to the averaged values of the second derivative in zone I of some quantity known only as zone averages F_i on a uniform grid:

$$F_i'' = \frac{(F_{i+1} - 2F_i + F_{i-1})}{\Delta x^2}$$

where F_{i+1}, F_i, and F_{i-1} are the zone averages of F in zones $i+1$, i, and $i-1$, resp., and Δx is the (uniform) zone width.

It is apparent that when zone averages are known only on the range [1:N], (as computed in the example above) that it is not possible to determine values for F_1'' and F_N'' due to the lack of information for the non-existant zones 0 and N+1.

A common solution to this quandary is to append the necessary number of boundary zones to each end of the computational grid. These boundary zones (henceforth referred to as *fake zones*) are then filled with values appropriate to the chosen boundary conditions. For example, the following subroutine implements two common boundary types, reflecting and periodic, on an arbitrary number of boundary zones.

```
      SUBROUTINE BOUNDARY ( XL, A, IBDY_TYPE, NZONES, NBDY)
c
c
C  WHAT              This routine implements one of two type of common
c                    boundary conditions: reflecting or periodic.  The
c                    boundary conditions are implemented by filling fake
c                    or boundary zones appended to each end of the arrays
c                    XL and A.  It is assumed that there are NZONES real
c                    (non-boundary) zones in the problem, NBDY fake zones
c                    to set at both the left and right ends, and that
c                    NBDY<=NZONES.
c
c                    The grid is defined by the locations of the lefthand
c                    zone interfaces XL.  Thus zone I would have its left
c                    edge at XL(I) and its right edge at XL(I+1).  Therefore
c                    a grid of N zones requires N+1 XL values.
c
C  DATA
C                  Input arrays are:
c                      XL:  the locations of the lefthand zone interfaces
c                       A:  an array of zone averages
c
C                  Input scalers are:
c                      IBDY_TYPE:  a flag indicating the type of boundary
c                                  conditions to impose:
c                             IBDY_TYPE = 0 -> reflecting
c                             IBDY_TYPE = 1 -> periodic
c                      NZONES:  the number of real (non-boundary) zones
c                                 in the problem.
c                      NBDY:    the number of non-boundary zones to append
c                                 to each end of the arrays.
c
C                  Input arrays are:
c                      XL:  the locations of the lefthand zone interfaces, the
c                            elements (1-NBDY:0) and (NZONES+2:NZONES+NBDY+1)
c                            are filled with appropriate values for the
c                            selected boundary conditions.
c                      AL:  the array of zone averages, the elements (1-NBDY:0)
c                           and (NZONES+1:NZONES+NBDY)
c                            are filled with appropriate values for the
c                            selected boundary conditions.
c  DIMENSIONS
c                   XL [1-NBDY:N+NBDY+1]
c                    A [1-NBDY:N+NBDY]
c
c
c
c
c
c
      REAL XL(1-NBDY:NZONES+NBDY+1)
      REAL  A(1-NBDY:NZONES+NBDY)

      IF ( IBDY_TYPE .NE. 0 ) GOTO 500
c
c   Reflecting boundary conditions
c
c   Left:
        DO I=1-NBDY, 0
           XL(I) = 2.0 * XL(1)  - XL(2 - I)
           A(I) = A(1-I)
        ENDDO
c
c
c   Right:
        DO I = N+1, N+NBDY
           XL(I+1) = 2.0 * XL(N+1)  - XL(2*N+1 - I)
           A(I) = A(2*N+1-I)
        ENDDO

      RETURN
500   CONTINUE
c
      IF ( IBDY_TYPE .NE. 1 ) GOTO 900
c
c   Periodic boundary conditions
c
c   Left:
      DXGRID = XL(NZONES+1) - XL(1)
      DO I=1-NBDY,0
         XL(i) = XL(I+NZONES) - DXGRID
         A(i) = A(I+NZONES)
      ENDDO
c
c   Right:
      DO   I = NZONES+1,NZONES+NBDY
         XL(I+1) = XL(I+1-NZONES) + DXGRID
         A(I) = A(I-NZONES)
      ENDDO
c
      RETURN
900   CONTINUE
c
c   Error, print advisory and terminate program.
       PRINT *,' UNKNOWN BOUNDARY TYPE:',IBDY_TYPE
       CALL ABORT()
```

```
c
      END  ! SUBROUTINE BOUNDARY
```

In the subroutine BOUNDARY, an arbitrary number of fake zones is permitted, signaled by the parameter NBDY. Two arrays are passed to the routine: and array of zone averages A and the array containing the locations of the lefthand zone interfaces XL. It is assumed that the first NBDY and last NBDY zones passed to this routine of each of these arrays are the fake zones which this routine must set. *Existing values (if any) in these fake zones will be overwritten by this subroutine.* The arrays are dimensioned so that the range (1:NZONES) of A and the range (1:NZONES+1) of XL are assumed to contain valid data. The convention will henceforth be adopted that arrays be dimensioned (1-NBDY:NZONES+NBDY[+1]). This explicitly indicates the fake zones within a routine. This is of course merely for the convenience of the programmer. A check is performed to ensure that only a valid boundary type has been requested.

Returning the the example of the second derivative mentioned above, the program SECOND_DER makes use of the boundary subroutine to compute the second derivative.

```
          PROGRAM SECOND_DER
c
c   This program is meant as an example of one way of treating
c   boundary conditions.  Given an array of zone averages A and
c   an array of lefthand zone interface locations XL which
c   define the grid, approximate the average of the 2dn derivative
c   in each zone.  Assume reflecting boundary conditions.
c
c   Zone averages and grid interface locations are read from a file
c   "INPUT.AVG" which was created earlier by either of
c    CREATE_ZONE_AVERAGES_SINE or CREATE_ZONE_AVERAGES_TANH.
c
c   The 2nd derivative is then approximated from
c
c       (A_AVG(I-1) - 2.0*A_AVG(I) + A_AVG(I+1))/DX**2
c
c   This requires NBDY = 1, that is one fake zone on each end of the arrays.
c
c
      ALLOCATABLE :: XL(:), A_AVG(:)
          PARAMETER ( NBDY = 1 )
c
      OPEN (unit = 11, file ='INPUT.AVG')
c  The data consists of N+2 lines.  The first line
c  contains the integer N, the number to zone averages in the data set.
c  This is followed by N lines, each line having two numbers, a value for
c  the location of the left edge of the zone XL, and the zone average for
c  that zone A_AVG.  The lines are assumed to be in sequential order according
c  to XL.  An additional line has only one value, the final left zone edge.  A
c  grid of N zones requires N+1 left zone edge locations. An appropriate data
c  set can be created from the programs CREATE_ZONE_AVERAGES*.


c   Read in the number of zones in the file
            READ (11,*)  N

c   Next, allocate the appropriate space
          ALLOCATE    (XL(1-NBDY:N+1+NBDY))
          ALLOCATE (A_AVG(1-NBDY:N+NBDY))
c
c   Read in the rest of the data
c
      DO I=1,N
         READ (11,*) XL(I),A_AVG(I)
      ENDDO
      READ (11,*) XL(N+1)

c
c   Assume reflecting boundaries:
c
          IBDY_TYPE = 0


          CALL BOUNDARY( XL(1-NBDY), A_AVG(1-NBDY), IBDY_TYPE, N,NBDY )
c
c   We are assuming a uniform grid: DX(I) = constant
c
       DX = XL(2)-XL(1)
       DXSQINV = 1.0 / (DX*DX)
          PRINT *,'       I      XMID           A_AVG(I)        D2AD2X'
          DO I=1,N
            D2AD2X = (A_AVG(I-1) - 2.0*A_AVG(I) + A_AVG(I+1))* DXSQINV
            XMID = 0.5 * (XL(I+1)+XL(I))
            WRITE (6,200) I, XMID, A_AVG(I), D2AD2X
200     FORMAT (4X,I4,t10,1P,3(E12.5,3x))
          ENDDO
```

```
      DEALLOCATE (XL, A_AVG)
      END ! PROGRAM SECOND_DER
```

This program makes use of zone averaged data created previously by either of the programs CREATE_ZONE_AVERAGES_SINE or CREATE_ZONE_AVERAGES_TANH. Either of these programs will create a data file named "OUTPUT.AVG." This file must be renamed "INPUT.AVG" to that the above program may access it. The reader should note that the arrays XL (zone interface locations) and A_AVG (zone averages) are dimensioned with sufficient space to implement the boundary conditions for this problem (one zone, indicated by NBDY), after which the data are read into the appropriate non-fake zone locations from the file INPUT.AVG. The boundary subroutine is then called with an array offset pointing to the first fake zone element. After the fake zones have been set, the calculation may proceed.

Results from this program are shown in figure 3 using data created previously for a discontinuity approximately one zone wide on a non-uniform grid, as shown in figure 2. The second derivative, shown in the bottom half of the plot illustrates that for a discontinuity, the second derivative changes sign, a fact which will be of use later.

This was intended merely as a simple example of how to implement boundary conditions. It must be stressed that the number of fake zones required varies from routine to routine. Also, boundary conditions can become quite complex, depending upon the exact nature of the problem under consideration. The user must expect to and be prepared to provide the routines which will initialize the appropriate fake zones thereby incorporating some of his unique and intimate knowledge of his problem into the calculation.