C.1. Input Files

The program `P-TERM.EXE` is a program that was compiled using Watcom C/C^{++} v.10 to work in a DOS environment. `P-TERM.EXE` computes approximations to some terms of the Poisson integral (equation 25, see §4.3) using experimental, wing-body junction flow, LDV velocity data. The calculations performed by this program were used to analyze the Poisson Ratio (equation 56) that is discussed in §4.3. Input to `P-TERM.EXE` is controlled through the header file `BL-DATA.H`. The program doesn't accept any command line arguments. Therefore, the source code must be recompiled in order to change the input information. The calculations were performed using two data sets. The control of which data set was used was handled using a C-compiler directive, `#define`, at the beginning of `P-TERM.C`. The directive `#define LOW_RE` causes the program to use the velocity data of Ölçmen and Simpson (1996, Re_θ = 7300(2-D), 5940(3-D)). The directive `#define HIGH_RE` causes the program to use the velocity data of Ölçmen *et al.* (1998, Re_θ = 23400(2-D), 23200(3-D)).

C.2. Output files

`P-TERM.EXE` creates 10 output files– one from 2-D velocity data and 9 from each velocity measurement station around the wing-body junction. Each output file contains 12 columns. Each column is a y^+-profile of a particular variable. Table C.1 shows the contents of each column of the output file.

For 2-D flow, $W = 0$. Therefore, $\partial W^+/\partial y^+$ (column 8, see table C.1) and v^+ ($\partial W^+/\partial y^+$) (column 11, see table C.1) should be identically zero. However, these terms were non-zero when calculated using the experimental data, due to the experimental uncertainty of the velocity data. Therefore, the output file contains non-zero values in columns 8 and 11 even when 2-D flow velocity data was used in the calculation. However, the quantity in column 12 (see table C.1) was

adjusted so as not to include non-zero *W*-velocity data. When 2-D flow velocity data is used in the calculation, column 12 contains

$$\left[\sqrt{\overline{\left(v^{+}\right)^{2}}}\left(\frac{\partial U^{+}}{\partial y^{+}}\right)+\left(\frac{\partial}{\partial y^{+}}\sqrt{\overline{\left(v^{+}\right)^{2}}}\right)\left(\frac{\partial V^{+}}{\partial y^{+}}\right)\right]^{2}$$

When 3-D, wing-body junction flow velocity data is used in the calculation, columns 4-12 each contain the term shown in table C.1 normalized by its value in a 2-D flow at comparable Re_{θ}. For example, when 3-D, wing-body junction flow velocity data is used in the calculation, column 4 contains

$$\frac{\left[\sqrt{\overline{\left(v^{+}\right)^{2}}}\right]_{\text{3-D}}}{\left[\sqrt{\overline{\left(v^{+}\right)^{2}}}\right]_{\text{2-D}}}$$

All derivatives are calculated using a 5-point parabola fit to the experimental data.

Column Number	Column Label	Variable contained
1	"yplus"	y^+
2	"y"	y-coordinate, units = meters
3	"ydstar"	y/δ^*
4	"v"	$\sqrt{\overline{\left(v^+\right)^2}}$
5	"vdy"	$\frac{\partial}{\partial y^+}\sqrt{\overline{\left(v^+\right)^2}}$
6	"dUdy"	$\frac{\partial U^+}{\partial y^+}$
7	"dVdy"	$\frac{\partial V^+}{\partial y^+}$
8	"dWdy"	$\frac{\partial W^+}{\partial y^+}$
9	"Uterm"	$\sqrt{\overline{\left(v^+\right)^2}}\left(\frac{\partial U^+}{\partial y^+}\right)$
10	"Vterm"	$\left(\frac{\partial}{\partial y^+}\sqrt{\overline{\left(v^+\right)^2}}\right)\left(\frac{\partial V^+}{\partial y^+}\right)$
11	"Wterm"	$\sqrt{\overline{\left(v^+\right)^2}}\left(\frac{\partial W^+}{\partial y^+}\right)$
12	"SumSq"	$\left[\sqrt{\overline{\left(v^+\right)^2}}\left(\frac{\partial U^+}{\partial y^+}+\frac{\partial W^+}{\partial y^+}\right)+\left(\frac{\partial}{\partial y^+}\sqrt{\overline{\left(v^+\right)^2}}\right)\left(\frac{\partial V^+}{\partial y^+}\right)\right]^2$

Table C.1. Contents of `P-TERM.EXE` output file.

C.3. Program Listings

The source code for P-TERM.EXE is split up into 8 files:

P-TERM.C	Main functions of the program
BL-DATA.H	Header file that contains the path and name of the LDV velocity data files and values for boundary layer parameters such as U_e, δ, δ^*, etc.
VELDATA.H VELDATA.C	These two files contain functions used to read the LDV data files
PARA-FIT.H PARA-FIT.C	These two files contain functions that fit a polynomial of order 2 or less to two data arrays using a least-squares fit
MYFILE.H MYFILE.C	These two files contain utility functions used for file I/O

P-TERM.C

```
/*************************************************************************
*   THIS PROGRAM CALCULATES SOME TERMS OF THE POISSON EQN AND WRITES        *
*   THEM TO FILE.  VALUES AT 3D STATIONS ARE NORMALIZED BY CORRESPONDING    *
*   2D VALUES.  NOTE THAT ALTHOUGH dW/dy IS CALCULATED FOR THE 2D FLOW.  IT *
*   SHOULD BE ZERO.                                                         *
*                                                                           *
*   TO RUN LOW RE(THETA)   #define LOW_RE                                  *
*                                                                           *
*   TO RUN HIGH RE(THETA)  #define HIGH_RE                                 *
*                                                                           *
*  MODIFIED 5/4/99 TO USE YPLUS-WALL-COORDINATES VELOCITY FILES             *
*                                                                           *
*************************************************************************/
#define LOW_RE

#include "myfile.h"
#include "veldata.h"
#include "bl-data.h"
#include "para-fit.h"
#include <stdio.h>
#include <string.h>
#include <math.h>

#define ERR_LOW_Y       2
#define ERR_HIGH_Y      3
#define ERR_BAD_DERIV   4

void main(void)
{  FileInfo InF, OutF ;
   int      yloc, err, st ;
```

```
   double   dimY,
            v,      v2D[NUM_Y_LOCS],
            dvdy,   dvdy2D[NUM_Y_LOCS],
            dUdy,   dUdy2D[NUM_Y_LOCS],
            dVdy,   dVdy2D[NUM_Y_LOCS],
            dWdy,   Wterm,
            Uterm,  Uterm2D[NUM_Y_LOCS],
            Vterm,  Vterm2D[NUM_Y_LOCS],
            SumSq,  SumSq2D[NUM_Y_LOCS] ;
//
//  2D STATION - POISSON TERMS ARE STORED IN ORDER TO NORMALIZE POISSON TERMS
//      FOR 3D STATIONS
//
    //
    //  OPEN DATA FILE AND READ DATA
    //
   strcpy(InF.name, DataFname[0]) ;
   OpenFile (&InF, "rt") ;
   ZeroAllVelData() ;
   ReadVelData(InF.ptr) ;
   CloseFile(&InF) ;
    //
    //  CONVERT Ys TO METERS AND TAKE SQUARE ROOT OF NORMAL STRESSES
    //
//   CorrectYs() ;
   SqrtNormStresses() ;
    //
    //  OPEN OUTPUT FILE AND WRITE COLUMN LABELS
    //
   strcpy(OutF.name, OutFname[0]);
   OpenFile(&OutF, "wt");

   fprintf (OutF.ptr,"    \"yplus\"        \"y\"    ") ;
   fprintf (OutF.ptr,"   \"ydstar\"        \"v\"    ");
   fprintf (OutF.ptr,"     \"vdy\"       \"dUdy\"   ") ;
   fprintf (OutF.ptr,"    \"dVdy\"       \"dWdy\"   ") ;
   fprintf (OutF.ptr,"    \"Uterm\"      \"Vterm\"  ") ;
   fprintf (OutF.ptr,"    \"Wterm\"      \"SumSq\"\n") ;
    //
    //  FOR EACH y+
    //
   for (yloc=0; yloc<NUM_Y_LOCS; yloc++)
    {
        //
        //  CALCULATE POISSON TERMS
        //
            //  y+, y(m), y/d*
            //
      fprintf (OutF.ptr, "%13.4le", yplus[yloc]);
      dimY = yplus[yloc] ;
      fprintf (OutF.ptr, "%13.4le", dimY*nu[0]/utau[0]) ;
      fprintf (OutF.ptr, "%13.4le", dimY*nu[0]/utau[0]/delstar[0]) ;
            //
            //  v
            //
      err = CalcTerm(&v2D[yloc], dimY, &vv, 0) ;
      if (err)
       { printf (" ERR : yloc = %i, v2D CalcTerm returned %i\n", yloc, err);
         return ;
       }
      fprintf (OutF.ptr,"%13.4le", v2D[yloc]) ;
            //
```

```
            //  dv/dy
            //
        err = CalcTerm(&dvdy2D[yloc], dimY, &vv, 1) ;
        if (err)
         { printf (" ERR : yloc = %i, dvdy2D CalcTerm returned %i\n",yloc,err);
           return ;
         }
        fprintf (OutF.ptr,"%13.4le", dvdy2D[yloc]) ;
            //
            //  dU/dy
            //
        err = CalcTerm(&dUdy2D[yloc], dimY, &U, 1) ;
        if (err)
         { printf (" ERR : yloc = %i, dUdy2D CalcTerm returned %i\n",yloc,err);
           return ;
         }
        fprintf (OutF.ptr,"%13.4le", dUdy2D[yloc]) ;
            //
            //  dV/dy
            //
        err = CalcTerm(&dVdy2D[yloc], dimY, &V, 1) ;
        if (err)
         { printf (" ERR : yloc = %i, dVdy2D CalcTerm returned %i\n",yloc,err);
           return ;
         }
        fprintf (OutF.ptr,"%13.4le", dVdy2D[yloc]) ;
            //
            //  dW/dy
            //
        err = CalcTerm(&dWdy, dimY, &W, 1) ;
        if (err)
         { printf (" ERR : yloc = %i, dWdy2D CalcTerm returned %i\n",yloc,err);
           return ;
         }
        fprintf (OutF.ptr,"%13.4le", dWdy) ;
            //
            //  Uterm = (v)(dU/dy)
            //
        Uterm2D[yloc] = v2D[yloc]*dUdy2D[yloc] ;
        fprintf (OutF.ptr, "%13.4le", Uterm2D[yloc]) ;
            //
            //  Vterm = (dv/dy)(dV/dy)
            //
        Vterm2D[yloc] = dvdy2D[yloc]*dVdy2D[yloc] ;
        fprintf (OutF.ptr, "%13.4le", Vterm2D[yloc]) ;
            //
            //  Wterm = (v)(dW/dy)
            //
        Wterm = v2D[yloc]*dWdy ;
        fprintf (OutF.ptr, "%13.4le", Wterm) ;
            //
            //  SumSq = [ (v)(dU/dy) + (dv/dy)(dV/dy) ]^2
            //
        SumSq2D[yloc] = Uterm2D[yloc] + Vterm2D[yloc] ;
        SumSq2D[yloc] *= SumSq2D[yloc] ;
        fprintf (OutF.ptr,"%13.4le\n", SumSq2D[yloc]);
        fflush(OutF.ptr) ;
      }
    CloseFile (&OutF) ;
//
//  FOR EACH 3D STATION
//
```

```
   for (st=1; st<NUM_STATIONS; st++)
    {
        //
        //  OPEN DATA FILE AND READ DATA
        //
       strcpy(InF.name, DataFname[st]) ;
       OpenFile (&InF, "rt") ;
       ZeroAllVelData() ;
       ReadVelData(InF.ptr) ;
       CloseFile(&InF) ;
        //
        //  CONVERT Ys TO METERS AND TAKE SQUARE ROOT OF NORMAL STRESSES
        //
//#ifdef LOW_RE
//       if (st>7)        //  USE IF STATEMENT FOR LOW RE(THETA)
//#endif
//         CorrectYs() ;
       SqrtNormStresses() ;
        //
        //  OPEN OUTPUT FILE AND WRITE COLUMN LABELS
        //
       strcpy(OutF.name, OutFname[st]);
       OpenFile(&OutF, "wt");

       fprintf (OutF.ptr,"    \"yplus\"        \"y\"    ") ;
       fprintf (OutF.ptr,"   \"ydstar\"       \"v\"    ");
       fprintf (OutF.ptr,"     \"vdy\"       \"dUdy\"   ") ;
       fprintf (OutF.ptr,"    \"dVdy\"       \"dWdy\"   ") ;
       fprintf (OutF.ptr,"    \"Uterm\"      \"Vterm\"  ") ;
       fprintf (OutF.ptr,"    \"Wterm\"      \"SumSq\"\n") ;
        //
        //  FOR EACH yplus
        //
       for (yloc=0; yloc<NUM_Y_LOCS; yloc++)
        {
            //
            //  CALCULATE POISSON TERMS
            //
                //  y+, y(m), y/d*
                //
          fprintf (OutF.ptr, "%13.4le", yplus[yloc]);
          dimY = yplus[yloc] ;
          fprintf (OutF.ptr, "%13.4le", dimY*nu[st]/utau[st]) ;
          fprintf (OutF.ptr, "%13.4le", dimY*nu[st]/utau[st]/delstar[st]) ;
                //
                //  v
                //
          err = CalcTerm(&v, dimY, &vv, 0) ;
          if (err)
           { printf (" ERR : yloc = %i, v CalcTerm returned %i\n", yloc, err);
             return ;
           }
          fprintf (OutF.ptr,"%13.4le", v) ;
                //
                //  dv/dy
                //
          err = CalcTerm(&dvdy, dimY, &vv, 1) ;
          if (err)
           { printf (" ERR : yloc = %i, dvdy CalcTerm returned
%i\n",yloc,err);
             return ;
           }
```

```
          fprintf (OutF.ptr,"%13.4le", dvdy) ;
                //
                //  dU/dy
                //
          err = CalcTerm(&dUdy, dimY, &U, 1) ;
          if (err)
           { printf (" ERR : yloc = %i, dUdy CalcTerm returned
%i\n",yloc,err);
             return ;
           }
          fprintf (OutF.ptr,"%13.4le", dUdy) ;
                //
                //  dV/dy
                //
          err = CalcTerm(&dVdy, dimY, &V, 1) ;
          if (err)
           { printf (" ERR : yloc = %i, dVdy CalcTerm returned
%i\n",yloc,err);
             return ;
           }
          fprintf (OutF.ptr,"%13.4le", dVdy) ;
                //
                //  dW/dy
                //
          err = CalcTerm(&dWdy, dimY, &W, 1) ;
          if (err)
           { printf (" ERR : yloc = %i, dWdy CalcTerm returned
%i\n",yloc,err);
             return ;
           }
          fprintf (OutF.ptr,"%13.4le", dWdy) ;
                //
                //  Uterm = (v)(dU/dy)
                //
          Uterm = v*dUdy ;
          fprintf (OutF.ptr, "%13.4le", Uterm / Uterm2D[yloc]) ;
                //
                //  Vterm = (dv/dy)(dV/dy)
                //
          Vterm = dvdy*dVdy ;
          fprintf (OutF.ptr, "%13.4le", Vterm / Vterm2D[yloc]) ;
                //
                //  Wterm = (v)(dW/dy)
                //
          Wterm = v*dWdy ;
          fprintf (OutF.ptr, "%13.4le", Wterm) ;
                //
                //  SumSq = [ (v)(dU/dy) + (dv/dy)(dV/dy) + (v)(dW/dy) ]^2
                //
          SumSq = Uterm + Vterm + Wterm ;
          SumSq *= SumSq / SumSq2D[yloc] ;
          fprintf (OutF.ptr,"%13.4le\n", SumSq);
          fflush(OutF.ptr) ;
        }
       CloseFile (&OutF) ;
    }
   printf ("\n PROGRAM COMPLETED\n") ;
}
```

```
/*************************************************************************
*   THIS FUNCTION REPLACES THE NORMAL STRESSES WITH THEIR SQUARE ROOT       *
*************************************************************************/
void SqrtNormStresses(void)
{ int i ;

  for (i=0; i<uu.num; i++)  uu.dat[i] = sqrt(uu.dat[i]) ;
  for (i=0; i<vv.num; i++)  vv.dat[i] = sqrt(vv.dat[i]) ;
  for (i=0; i<ww.num; i++)  ww.dat[i] = sqrt(ww.dat[i]) ;
}


/*************************************************************************
*   THIS FUNCTION CONVERTS THE Y VALUES FROM MICROMETERS TO METERS          *
*************************************************************************/
void CorrectYs(void)
{ int i ;

  for (i=0; i<U.num; i++)   U.y[i] *= 1e-6 ;
  for (i=0; i<V.num; i++)   V.y[i] *= 1e-6 ;
  for (i=0; i<W.num; i++)   W.y[i] *= 1e-6 ;
  for (i=0; i<uu.num; i++)  uu.y[i] *= 1e-6 ;
  for (i=0; i<vv.num; i++)  vv.y[i] *= 1e-6 ;
  for (i=0; i<ww.num; i++)  ww.y[i] *= 1e-6 ;
  for (i=0; i<uv.num; i++)  uv.y[i] *= 1e-6 ;
  for (i=0; i<uw.num; i++)  uw.y[i] *= 1e-6 ;
  for (i=0; i<vw.num; i++)  vw.y[i] *= 1e-6 ;
}


/*************************************************************************
*   THIS FUNCTION INTERPOLATES A VALUE OR ITS DERIVATIVE USING A 5 POINT    *
*       PARABOLA FIT GIVEN A "KEY" Y VALUE                                  *
*************************************************************************/
int CalcTerm (double *term, double KeyValue, VelData *D, int Deriv)
{ int       r=0, StartRow, err ;
  double    A[3] ;

  while ((KeyValue > D->y[r]) && (r < D->num)) r++ ;

  if (r == 0)  return ERR_LOW_Y ;
  if (r == D->num)  return ERR_HIGH_Y ;

  StartRow = r - 2 ;
  if (StartRow < 0) StartRow = 0 ;
  if (StartRow > D->num-5) StartRow = D->num - 5 ;

  err = ParaFit(5, &(D->y[StartRow]), &(D->dat[StartRow]), A) ;
  if (err == ERR_UNDEF_COEFF)  return err ;

  switch (Deriv)
   { case 0 : *term = A[2]*KeyValue*KeyValue + A[1]*KeyValue + A[0] ; break ;
     case 1 : *term = 2.0*A[2]*KeyValue + A[1] ; break ;
//     case 2 : *term = 2.0*A[2] ; break ;
     default : return ERR_BAD_DERIV ;
   }

  return 0 ;
}
```

BL-DATA.H

```
#ifndef BL_DATA_H_
#define BL_DATA_H_
//
//  THESE ARE THE NON-DIMENSIONAL Y VALUES FOR THE OUTPUT PROFILE
//
#define NUM_Y_LOCS  14
  const double yplus[NUM_Y_LOCS] =
                  { 10.0, 20.0, 30.0, 40.0, 50.0, 62.8, 69.1, 75.4,
                    81.7, 88.0, 94.2, 100.5, 106.8, 113.1 };

//
//  THESE ARE THE FILE NAMES OF DATA AND OUTPUT FILES
//
#define NUM_STATIONS    10

/*************************************************************************
*   CONTROL OF VARIABLE INITIALIZATION (LOW RE(THETA) OR HIGH RE(THETA)) IS *
*       THROUGH #define LOW_RE STATEMENT AT BEGINNING OF p-term.c          *
*              ^^^^^^^^^^^^^                                              *
*************************************************************************/

#ifdef  LOW_RE
        //  RE(THETA) = 5940

  const char   DataFname[NUM_STATIONS][80] =
                {"E:\\LDV-BL\\RE7000\\2DDER\\TC\\YPLUS\\2D1.UT",
                 "E:\\LDV-BL\\RE7000\\ST1\\WC\\YPLUS\\ST11.UT",
                 "E:\\LDV-BL\\RE7000\\ST2\\WC\\YPLUS\\ST21.UT",
                 "E:\\LDV-BL\\RE7000\\ST3\\WC\\YPLUS\\ST31.UT",
                 "E:\\LDV-BL\\RE7000\\ST4\\WC\\YPLUS\\ST41.UT",
                 "E:\\LDV-BL\\RE7000\\ST5\\WC\\YPLUS\\ST51.UT",
                 "E:\\LDV-BL\\RE7000\\ST6\\WC\\YPLUS\\ST61.UT",
                 "E:\\LDV-BL\\RE7000\\ST7\\WC\\YPLUS\\ST71.UT",
                 "E:\\LDV-BL\\RE7000\\STATION8\\WC\\YPLUS\\S81.UT",
                 "E:\\LDV-BL\\RE7000\\STATION9\\WC\\YPLUS\\S912.UT" } ;

  const char   OutFname[NUM_STATIONS][80] =
                {"E:\\LDV-BL\\RE7000\\POISSON\\LO2D-P.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS1-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS2-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS3-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS4-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS5-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS6-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS7-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS8-PN.DAT",
                 "E:\\LDV-BL\\RE7000\\POISSON\\LOS9-PN.DAT" } ;

#else
        //  RE(THETA) = 23200

  const char   DataFname[NUM_STATIONS][80] =
                {"E:\\LDV-BL\\RE23000\\WC\\2D61.UT",
                 "E:\\LDV-BL\\RE23000\\WC\\S1H12.UT2",
                 "E:\\LDV-BL\\RE23000\\WC\\S2H15.UT2",
                 "E:\\LDV-BL\\RE23000\\WC\\S3H12.UT",
                 "E:\\LDV-BL\\RE23000\\WC\\S4H13.UT",
                 "E:\\LDV-BL\\RE23000\\WC\\ST5HL1.UT2",
```

```
                "E:\\LDV-BL\\RE23000\\WC\\S6H13.UT",
                "E:\\LDV-BL\\RE23000\\WC\\S7H12.UT",
                "E:\\LDV-BL\\RE23000\\WC\\S8H12.UT",
                "E:\\LDV-BL\\RE23000\\WC\\S9H12.UT" } ;

  const char   OutFname[NUM_STATIONS][80] =
                {"E:\\LDV-BL\\RE23000\\POISSON\\HI2D-P.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS1-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS2-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS3-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS4-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS5-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS6-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS7-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS8-PN.DAT",
                 "E:\\LDV-BL\\RE23000\\POISSON\\HIS9-PN.DAT" } ;
#endif
//
//  THESE ARE BOUNDARY LAYER VARIABLES USED TO NON-DIMENSIONALIZE
//
#ifdef LOW_RE
        //  RE(THETA) = 5940

   double nu[NUM_STATIONS] =
                { 1.67e-5, 1.65e-5, 1.65e-5, 1.65e-5, 1.65e-5,
                  1.67e-5, 1.68e-5, 1.68e-5, 1.67e-5, 1.68e-5 } ;
   double utau[NUM_STATIONS] =
                { 0.980, 0.864, 0.865, 0.957, 1.110,
                  1.150, 1.160, 1.200, 1.024, 1.011 } ;
   double Ue[NUM_STATIONS] =
                { 27.1, 24.9, 24.8, 25.3, 27.3,
                  29.5, 30.5, 31.0, 30.9, 30.5 } ;
   double del[NUM_STATIONS] =
                { 39.1e-3, 39.2e-3, 40.2e-3, 39.3e-3, 39.0e-3,
                  39.6e-3, 39.2e-3, 38.8e-3, 38.4e-3, 40.7e-3 } ;
   double delstar[NUM_STATIONS] =
                { 6.20e-3, 6.90e-3, 7.54e-3, 6.86e-3, 5.53e-3,
                  5.37e-3, 5.24e-3, 5.20e-3, 5.08e-3, 5.68e-3 } ;
#else
        //  RE(THETA) = 23200

   double nu[NUM_STATIONS] =
                { 1.68e-5, 1.67e-5, 1.67e-5, 1.67e-5, 1.67e-5,
                  1.68e-5, 1.67e-5, 1.66e-5, 1.69e-5, 1.67e-5 } ;
   double utau[NUM_STATIONS] =
                { 1.030, 0.910, 0.916, 1.094, 1.240,
                  1.208, 1.208, 1.304, 1.128, 1.104 } ;
   double Ue[NUM_STATIONS] =
                { 31.31, 28.31, 28.71, 29.03, 31.14,
                  33.02, 34.66, 35.48, 35.17, 34.30 } ;
   double del[NUM_STATIONS] =
                { 13.42e-2, 13.64e-2, 13.51e-2, 13.68e-2, 13.19e-2,
                  12.33e-2, 12.86e-2, 12.90e-2, 13.35e-2, 13.46e-2 } ;
   double delstar[NUM_STATIONS] =
                { 15.80e-3, 22.78e-3, 18.37e-3, 16.84e-3, 17.28e-3,
                  13.72e-3, 17.17e-3, 12.87e-3, 12.99e-3, 13.51e-3 } ;
#endif
#endif
```

VELDATA.H

```
#ifndef _VELDATA_H_
#define _VELDATA_H_

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

#define MAX_V_ROWS  40

typedef struct
 { int      num ;
   double   y[MAX_V_ROWS] ;
   double   dat[MAX_V_ROWS] ;
 }
VelData ;

extern VelData U, V, W, uu, vv, ww, uv, uw, vw ;

extern int  ReadVelData (FILE *Fptr) ;
extern void ZeroAllVelData (void) ;
extern void ZeroVelData (VelData *D) ;
extern int  ReadField (FILE *Fptr, double CurrY, VelData *D) ;

#endif
```

VELDATA.C

```
/*************************************************************************
*   THIS FILE CONTAINS FUNCTIONS TO READ VELOCITY PROFILE DATA FILES.  IT   *
*       PROPERLY HANDLES ENCOUNTERING "M" WHICH DENOTES A MISSING DATA      *
*       POINT                                                               *
*************************************************************************/

#include "veldata.h"

VelData U, V, W, uu, vv, ww, uv, uw, vw ;

int ReadVelData (FILE *Fptr)
{ double    CurrY ;
  int       err, i ;

  for (i=0; i<MAX_V_ROWS; i++)
   { //
     // Column 1 = y
     //
     err = fscanf(Fptr, "%lg", &CurrY);
     if (err == EOF) return 0;

     ReadField (Fptr, CurrY, &U) ;
     ReadField (Fptr, CurrY, &V) ;
     ReadField (Fptr, CurrY, &W) ;
     ReadField (Fptr, CurrY, &uu) ;
     ReadField (Fptr, CurrY, &vv) ;
     ReadField (Fptr, CurrY, &ww) ;
     ReadField (Fptr, CurrY, &uv) ;
```

```
     ReadField (Fptr, CurrY, &uw) ;
     ReadField (Fptr, CurrY, &vw) ;
   }
  return 0 ;
}


void ZeroAllVelData (void)
{ ZeroVelData (&U) ;
  ZeroVelData (&V) ;
  ZeroVelData (&W) ;
  ZeroVelData (&uu) ;
  ZeroVelData (&vv) ;
  ZeroVelData (&ww) ;
  ZeroVelData (&uv) ;
  ZeroVelData (&uw) ;
  ZeroVelData (&vw) ;
}


void ZeroVelData (VelData *D)
{ int i ;

  D->num = 0 ;
  for (i=0; i<MAX_V_ROWS; i++)
   { D->y[i]   = 0.0 ;
     D->dat[i] = 0.0 ;
}  }


int ReadField (FILE *Fptr, double CurrY, VelData *D)
{ char strfield[50] ;

  fscanf (Fptr, "%*[^0123456789EeM.+-]%[0123456789EeM.+-]", strfield) ;
  if (strcmp(strfield, "M"))
   { D->y[D->num] = CurrY ;
     D->dat[D->num] = atof(strfield) ;
     D->num++ ;
   }

  return 0 ;
}
```

PARA-FIT.H

```
#ifndef _PARA_FIT_H_
#define _PARA_FIT_H_
/*************************************************************************
*   HEADER FILE FOR para-fit.c WHICH FITS A PARABOLA TO THE GIVEN X AND Y   *
*       ARRAYS USING A LEAST SQUARES FIT                                    *
*************************************************************************/

#define SMALL           1e-30
#define ERR_UNDEF_COEFF 1

extern int    ParaFit (int Npts, double *X, double *Y, double *A) ;
extern double Deter33 (double A[3][3]) ;

#endif
```

PARA-FIT.C

```
/*************************************************************************
*   THIS FUNCTION FITS A PARABOLA TO THE GIVEN X AND Y ARRAYS USING A LEAST *
*       SQUARES FIT                                                       *
*************************************************************************/

#include "para-fit.h"
#include <stdio.h>

int ParaFit (int Npts, double *X, double *Y, double *A)
{ int       i, r, c, cc ;
  double    C[3][3], TempMat[3][3], D[3], CoeffDet ;

  for (r=0; r<3; r++)
   { D[r] = 0.0 ;
     for (c=0; c<3; c++)
      C[r][c] = 0.0 ;
   }

  for (i=0; i<Npts; i++)
   { C[0][1] += X[i] ;
     C[0][2] += X[i]*X[i] ;
     C[1][2] += X[i]*X[i]*X[i] ;
     C[2][2] += X[i]*X[i]*X[i]*X[i] ;
        D[0] += Y[i] ;
        D[1] += Y[i]*X[i] ;
        D[2] += Y[i]*X[i]*X[i] ;
   }
  C[0][0] = (double)Npts ;
  C[1][0] = C[0][1] ;
  C[1][1] = C[2][0] = C[0][2] ;
  C[2][1] = C[1][2] ;

  CoeffDet = Deter33(C) ;
  if (CoeffDet < SMALL) return ERR_UNDEF_COEFF ;

  for (cc=0; cc<3; cc++)
   { for (r=0; r<3; r++)
      { for (c=0; c<3; c++)
         TempMat[r][c] = C[r][c] ;
        TempMat[r][cc] = D[r] ;
      }

     A[cc]  = Deter33(TempMat) ;
     A[cc] /= CoeffDet ;
   }

  return 0 ;
}
```

```
/*************************************************************************
*   THIS FUNCTION COMPUTES THE DETERMINANT OF A 3X3 MATRIX               *
*************************************************************************/
double Deter33 (double A[3][3])
{ double   det ;

  det  = A[0][0] * A[1][1] * A[2][2] ;
  det += A[0][1] * A[1][2] * A[2][0] ;
  det += A[2][1] * A[1][0] * A[0][2] ;
  det -= A[0][2] * A[1][1] * A[2][0] ;
  det -= A[1][0] * A[0][1] * A[2][2] ;
  det -= A[2][1] * A[1][2] * A[0][0] ;

  return det ;
}
```

MYFILE.H

```
#ifndef _MYFILE_H_
#define _MYFILE_H_

#include <stdio.h>
#include <string.h>
#include <errno.h>

typedef struct
 { FILE             *ptr ;
   char             name[80] ;
 }
FileInfo ;

extern int  OpenFile (FileInfo *F, char *mode) ;
extern void CloseFile (FileInfo *F) ;

#endif
```

MYFILE.C

```
#include "myfile.h"

/*************************************************************************
*   THIS FUNCTIONS OPENS A FILE AND PRINTS STATUS INFO TO THE SCREEN      *
*************************************************************************/
int OpenFile (FileInfo *F, char *mode)
{
 printf (" Openning %s ... ", F->name) ;
 if ((F->ptr=fopen (F->name, mode)) == NULL)
  { printf ("\n  ERROR %i : %s\n", errno, strerror(errno)) ;
    return errno ;
  }
 printf ("O.K.\n") ;
 return 0 ;
}


void CloseFile (FileInfo *F)
{
  fclose (F->ptr) ;
}
```