APPENDIX C - COMPUTER PROGRAMS

PROGRAM 'BLSOL'

BLSOL is a PL/I procedure which finds momentum and displacement thickness of a boundary layer from hot-wire traverse data. Several basic velocity profile parameters are computed, as well as an estimate of the coefficient of friction at the wall using the universal law of the wall for flat plates as given by Schlichting[17]. The friction coefficient is also computed by the method set forth by Clauser[20].

Many different types of hot-wire calibration methods are supported, including those methods set forth in section V.D. General flow of the program can be read from the program listing in Figure 63.

The main procedure BLSOL acts to input environmental factors such as temperature and barometric pressure, and the raw data of height count of the traverse mechanism, time average voltage reading of the anemometer, and RMS voltage output of the anemometer. Some elementary calculation provides density and viscosity of the air (viscosity is provided by Sutherland's formula). BLSOL then expects control cards which determine if a calculation of boundary layer data is to be performed and control the printout of boundary layer data in table form. These actions are performed by other procedures and brief descriptions of them follow.

The first procedure called by BLSOL is BL_DATA. This procedure is the heart of the program. It expects an integer input (CAL_CODE) which determines the type of hot-wire calibration function and the coefficients of the function. The input flag CORR_FLAG signifies the input of a velocity correction factor to account for the alignment of the hot wire probe to the mean flow (see Webster[53] and Comte-Bellot[58]). The velocity and turbulence level at each data point is then calculated, with a wall correction imposed which assumes a standard five-micron wire. A correction function patterned after that given by Wills[61] is used. This is the piecewise linear function

$$E^2_{corr} = E^2_{meas} + C_1\left(\frac{b}{a}\right) - C_2 \qquad (36)$$

a = Diameter of hot-wire

b = Distance of hot-wire from wall

E = The voltage reading from the hot-wire

b/a	C_1	C_2
120-60	0.0005	0.06
60-40	0.00125	0.105
40-20	0.0035	0.195

This function is embodied in the procedure WALL_CORR. The recalculation of a set of data by different methods is facilitated by specifying the calibration method and any corrections after the input of raw data (mean and rms voltage, probe height).

After calculation of data point velocity and turbulence, the Reynold's number based on distance along the surface in the direction of flow is calculated. A crude estimate of surface friction coefficient is computed (PT1_CF) which is based on the assumption that the first data point from the surface is in the laminar sub-layer. Trapezoidal integration is used to obtain momentum and displacement thickness of the boundary layer. These are used to calculate the shape factor H and Reynolds Number based on displacement thickness.

An application of the law of the wall for turbulent boundary layers is used to provide a good estimate of surface friction coefficient (if the boundary layer is turbulent, of course). If the boundary layer is not turbulent in nature, but is laminar, the initial estimate (PT1_CF) will actually be more exact. The variable PT1_CF is used for an initial estimate of friction velocity, which in turn is used to transform the height above the plate into the coordinate y^+. According to Schlichting the universal law of the wall will be closely followed for y^+ from 20 to 500 in zero pressure gradient boundary layers. The

program determines the data points which fall in this category, and then calculates the mean coefficient of friction associated with those boundary layer data points. In order to determine the accuracy of the estimate of the surface friction, the standard deviation of the point friction coefficients is also computed (FRICTION_ST_DEV).

As an extra feature of the program, two quantities are computed which were first put forward by Clauser. Clauser claimed that two quantities, Δ and G, could be used to describe the behavior of turbulent boundary layers in pressure gradients. These numbers would be constant for a boundary layer under the influence of a constant pressure gradient. He experimentally derived the values for the zero pressure gradient case as $\Delta = 3.6$ and $G = 6.1$. Since the universal law of the wall is well accepted, it is interesting to contrast the values of Δ and G obtained from that law with the values found by Clauser.

Figure 63 - BL_SOL computer program listing

```
/*   BLSOL  -  A MAIN PROCEDURE TO SOLVE MOMENTUM AND DISPLACEMENT   */
/*             THICKNESS OF A BOUNDARY LAYER AND OTHER BASIC PARA-    */
/*             METERS, AS WELL AS ESTIMATE THE COEFFICIENT OF         */
/*             FRICTION FOR TURBULENT BOUNDARY LAYERS BY THE UNI-     */
/*             VERSAL LAW OF THE WALL FOR FLAT PLATES AS GIVEN BY     */
/*             SCHLICHTING.                                           */
/*   ********************************************************         */

BLSOL:
    PROC OPTIONS (MAIN);
        DO UNTIL (TEMP_F<0);
            GET LIST (POINT#,
                      TOP#,
                      LE_DISTANCE,
                      ZERO_CT_HEIGHT,
                      TEMP_F,
                      BAR_PRES);
            DO I = 1 TO POINT#;
                GET LIST (HEIGHT(I),
                          MEAN_VOLTAGE(I),
                          RMS_VOLTAGE(I));
                END;
            DENSITY = 0.0412*BAR_PRES/(460 + TEMP_F);
            VISCOSITY = 2.2137E-8*(460 + TEMP_F)**1.5/(658.23 + TEMP_F);
            DO I = 1 TO POINT#;
                HEIGHT(I) = HEIGHT(I)*INCH_PER_CT + ZERO_CT_HEIGHT;
                END;
            DO UNTIL (¬BL_PROCESS_FLAG);
                GET LIST (BL_PROCESS_FLAG);
                IF BL_PROCESS_FLAG
                THEN DO;
                    CALL BL_DATA;
                    GET LIST (TABLE_FLAG);
                    CALL PRINT_DATA;
                    END;
                ELSE;
                END;
            END;


/*   BL_DATA -  A PROCEDURE TO SOLVE THE BOUNDARY LAYER PARAMETERS   */
/*              THICKNESS OF A BOUNDARY LAYER AND OTHER BASIC PARA-  */
/*              AND COEFFICIENT OF FRICTION USING THE LAW OF THE     */
/*              WALL FOR FLAT PLATES AS GIVEN BY SCHLICHTING         */
/*   ********************************************************       */

BL_DATA:
    PROC;
        GET LIST (CAL_CODE);
```

```
GET LIST (A(CAL_CODE),
          B(CAL_CODE),
          C(CAL_CODE),
          D(CAL_CODE),
          E(CAL_CODE),
          EXPONENT);
GET LIST (CORR_FLAG);
IF CORR_FLAG
THEN
    GET LIST(VEL_CORR);
ELSE;
DO I = 1 TO POINT#;
    VOLTS = MEAN_VOLTAGE(I);
    RMS_VOLTS = RMS_VOLTAGE(I);
    IF HEIGHT(I)<0.025
    THEN
        CALL WALL_CORR;
    ELSE;
    CALL TURBULENCE;
    IF CORRFLAG
    THEN DO;
        FPS = FPS*(1/VEL_CORR);
        IF FPS>0
        THEN
            TI = 100*RMS_VOLTS*SENSITIVITY/FPS;
        ELSE;
        END;
    VEL(I) = FPS;
    TURB(I) = TI;
    END;
REX = VEL(TOP#)*DENSITY*LE_DISTANCE/(VISCOSITY*INCH_PER_FT);
MAX_VEL = VEL(TOP#);
PT1_CF = 2*VISCOSITY*VEL(1)*INCH_PER_FT/(DENSITY*HEIGHT(1)
                                          *MAX_VEL**2);
OLD_DISP_DER = 0;
OLD_MOM_DER = 0;
DISP_THK_SUM = 0;
MOM_THK_SUM = 0;
POINT_FRICTION_COEF = -1;
DO I = 1 TO TOP#;
    IF I = 1
    THEN
        DELTA_Y = HEIGHT(I);
    ELSE
        DELTA_Y = HEIGHT(I)-HEIGHT(I-1);
    MOM_DER = MAX_VEL-VEL(I);
    MOM_THK_SUM = MOM_THK_SUM + DELTA_Y*(OLD_MOM_DER
                                          + MOM_DER)/2;
    OLD_MOM_DER = MOM_DER;
    DISP_DER = VEL(I)*(MAX_VEL-VEL(I));
    DISP_THK_SUM = DISP_THK_SUM + DELTA_Y*(OLD_DISP_DER + DISP_DER)/2;
    OLD_DISP_DER = DISP_DER;
```

```
        END;
    DISP_THK = DISP_THK_SUM/(MAX_VEL**2);
    MOM_THK = MOM_THK_SUM/MAX_VEL;
    REL = MAX_VEL*DENSITY/VISCOSITY;
    H = MOM_THK/DISP_THK;
    RED = MAX_VEL*DISP_THK*DENSITY/(VISCOSITY*INCH_PER_FT);

/*  INITIAL STAB AT FRICTION COEF          */

    COEF_FRICTION = PT1_CF;
    FRICTION_VEL = MAX_VEL/SQRT(2/COEF_FRICTION);

/*  ELIMINATE POOR POINTS    */

    DO J = 1 TO 2;
        DO I = 1 TO TOP#;
            Y_PLUS = HEIGHT(I)*FRICTION_VEL*DENSITY/(VISCOSITY*INCH_PER_FT);
            IF Y_PLUS<20
            THEN
                LAW_WALL_BOT# = I + 1;
            ELSE;
            IF Y_PLUS< = 500
            THEN
                LAW_WALL_TOP# = I;
            ELSE;
            END;
        FRICTION_POINT# = LAW_WALL_TOP#-LAW_WALL_BOT# + 1;

/*  FIND AVERAGE FRICTION COEF.      */

        FRICTION_SUM = 0;
        DO I = LAW_WALL_BOT# TO LAW_WALL_TOP#;
            REY = HEIGHT(I)*MAX_VEL*DENSITY/(VISCOSITY*INCH_PER_FT);
            VEL_RATIO = VEL(I)/MAX_VEL;
            CALL DATA_POINT_FRICTION;
            FRICTION_SUM = FRICTION_SUM + POINT_FRICTION_COEF(I);
            END;
        IF FRICTION_POINT#>0
        THEN
            AVG_FRICTION_COEF = FRICTION_SUM/FRICTION_POINT#;
        ELSE
            AVG_FRICTION_COEF = 10;
        IF J = 1 & AVG_FRICTION_COEF>0
        THEN
            FRICTION_VEL = MAX_VEL/SQRT(2/AVG_FRICTION_COEF);
        ELSE;
        END;

/*  FIND STANDARD DEV. OF FRICTION COEF.     */

    FRICTION_SUM = 0;
    DO I = LAW_WALL_BOT# TO LAW_WALL_TOP#;
```

```
            FRICTION_SUM = FRICTION_SUM + (POINT_FRICTION_COEF(I)
                                          -AVG_FRICTION_COEF)**2;
            END;
        IF FRICTION_POINT# > 1
        THEN
            FRICTION_ST_DEV = SQRT(FRICTION_SUM/(FRICTION_POINT#-1));
        ELSE
            FRICTION_ST_DEV = -1;
        IF AVG_FRICTION_COEF < = 0
        THEN
            AVG_FRICTION_VEL = -1;
        ELSE
            AVG_FRICTION_VEL = MAX_VEL/SQRT(2/AVG_FRICTION_COEF);

/*  FIND CLAUSER'S VARIABLES  */

        IF AVG_FRICTION_COEF > 0
        THEN DO;
            DELTA = MOM_THK/SQRT(AVG_FRICTION_COEF/2);
            G = (1-1/H)/SQRT(AVG_FRICTION_COEF/2);
            END;
        ELSE DO;
            G = -1;
            DELTA = -1;
            END;
    END BL_DATA;

/*  DATA_POINT_FRICTION  - A PROCEDURE TO FIND THE COEFFICIENT OF      */
/*                         FRICTION CORRESPONDING TO THE HEIGHT        */
/*                         REYNOLDS NUMBER AND THE VELOCITY RATIO      */
/*                         (POINT VELOCITY OVER FREE STREAM VELO-      */
/*                         CITY).  USES THE LAW FOF THE WALL FOR       */
/*                         FLAT PLATES GIVEN BY SCHLICHTING.           */
/*  ********************************************************           */

DATA_POINT_FRICTION:
    PROC;
        DELTA_CF = 0.001;
        CF = 0;
        STEP = 0;
        DO K5 = 1 TO 10000 UNTIL (DELTA_CF < 0.000001);
            CF = CF + DELTA_CF;
            ERROR = VEL_RATIO*SQRT(2/CF)-5.85*LOG10(REY*SQRT(CF/2))-5.56;
            IF ERROR < 0
            THEN DO;
                CF = CF-DELTA_CF;
                DELTA_CF = DELTA_CF/10;
                END;
            POINT_FRICTION_COEF(I) = CF;
        END DATA_POINT_FRICTION;

/*  WALL_CORR -  A PROCEDURE TO CORRECT THE HOT-WIRE ANEMOMETER        */
```

```
/*                  READING NEAR THE SURFACE.  A STANDARD 5 MICRON WIRE   */
/*                  IS ASSUMED.  THE VOLTAGE READING IS CORRECTED WITH    */
/*                  A PIECEWISE LINEARIZED FUNCTION RELATED TO THE        */
/*                  COMPLEX FUNCTION CORRECTION GIVEN BY ANALYSIS.        */
/*   ************************************************************         */

WALL_CORR:
     PROC;
        E_SQUARE = VOLTS**2;
        B_A = HEIGHT(I)/0.0001969;
        IF B_A<120 & B_A>60
        THEN
           E_SQUARE = E_SQUARE + 0.0005*B_A-0.06
        ELSE
           IF B_A< = 60 & B_A>40
           THEN
              E_SQUARE = E_SQUARE + 0.00125*B_A-0.105
           ELSE
              IF B_A<40
              THEN
                 E_SQUARE = E_SQUARE + 0.0035*B_A-0.195;
              ELSE;
        VOLTS = SQRT(E_SQUARE);
        MEAN_VOLTAGE(I) = VOLTS;
     END WALL_CORR;

/*   VELOCITY  -  A CALIBRATION OF VELOCITY FROM HOT-WIRE MEAN             */
/*                VOLTAGE                                                 */
/*                                                                        */
/*                ENTER WITH CAL_CODE, VOLTS, AND CALIBRATION CONSTANTS   */
/*                                                                        */
/*                EXIT WITH VELOCITY IN FPS                               */
/*   ************************************************************         */

VELOCITY:
     PROC;
        SELECT (CALCODE);
           WHEN (1) DO;
              DUM = (VOLTS**2-A(1))/B(1);
              IF DUM<0
              THEN
                 FPS = 0;
              ELSE
                 FPS = DUM**(1/EXPONENT);
              END;
           WHEN (2) DO;
              DUM = B(2)*B(2)-4*C(2)*(A(2)-VOLTS**2);
              IF DUM<0
              THEN
                 FPS = 0;
              ELSE DO;
                 DUM = (-B(2) + SQRT(DUM))/(2*C(2));
```

```
                FPS = DUM*DUM;
                END;
            END;
        OTHERWISE
            IF (CALCODE>2 & CALCODE<7)
            THEN DO;
                V1 = 0;
                E4 = VOLTS**2;
                D1 = 1;

/*  TURBULENCE -  A CALCULATION OF TURB. INTENSITY AND SENSITIVITY      */
/*                OF A HOT-WIRE ANEMOMETER, PROCEDURE                   */
/*                                                                      */
/*    ENTER WITH CAL_CODE, VOLTS (MEAN VOLTAGE), AND RMS_VOLTS          */
/*                                                                      */
/*    EXIT WITH SENSITIVITY (FPS/VOLT), TI (TURB. INTENSITY, %)         */
/*                                                                      */
/*    CALLS:  VELOCITY, PROCEDURE                                       */
/*  ********************************************************************/

TURBULENCE:
    PROC;
        CALL VELOCITY;
        IF FPS>0.01
        THEN
            SELECT (CAL_CODE);
                WHEN (1) DO;
                    DUM = (VOLTS**2-A(1))/b(1);
                    IF DUM<0.0001
                    THEN
                        TI = 0;
                    ELSE
                        SENSITIVITY = 2*VOLTS*(DUM**(1/EXPONENT-1))/
                                      (EXPONENT*B(1));
                    END;
                WHEN (2)
                    SENSITIVITY = 2*VOLTS/(0.5*B(2)*FPS**-0.5 + C(2));
                WHEN (3)
                    SENSITIVITY = 2*VOLTS/(0.5*B(3)*FPS**-0.5 + C(3)
                                  + 1.5*D(3)*FPS**0.5);
                WHEN (4)
                    SENSITIVITY = 2*VOLTS/(0.5*B(4)*FPS**-0.5 + C(4)
                                  + 1.5*D(4)*FPS**0.5 + 2*E(4)*FPS);
                WHEN (5)
                    SENSITIVITY = 2*VOLTS/(0.5*B(5)*FPS**-0.5
                                  + 1.5*C(5)*FPS**0.5);
                WHEN (6)
                    SENSITIVITY = 2*VOLTS/(0.5*B(6)*FPS**-0.5
                                  + 1.5*C(6)*FPS**0.5
                                  + 2.5*D(6)*FPS**1.5);
                OTHERWISE;
            END;
```

```
        ELSE
            SENSITIVITY = 0;
        IF FPS>0
        THEN
            TI = RMS_VOLTS*SENSITIVITY*100/FPS;
        ELSE
            TI = 0;
    END TURBULENCE;
/*  PRINT_DATA  -  A PROCEDURE TO PRINT OUT THE RESULTS OF DATA REDUC-  */
/*                 TION OF FREE-STREAM PARAMETERS, BOUNDARY LAYER      */
/*                 PARAMETERS AND FRICTION COEFFICIENT.  ALSO CALLS A  */
/*                 TABLE PRINTING OUTPUT ROUTINE TO GIVE PARTICULAR    */
/*                 TABLES OF DATA POINTS IN THE BOUNDARY LAYER.        */
/*  ***************************************************************    */

PRINT_DATA:
    PROC;
        PUT PAGE LINE(10);
        SELECT (CAL_CODE);
            WHEN (1) PUT EDIT ('EXPONENT LAW  EXPONENT = ',EXPONENT)
                                (COL(10),A(25),F(5,2));
            WHEN (2) PUT EDIT ('SECOND ORDER POLY. IN SQRT. VEL.')
                                (COL(10),A);
            WHEN (3) PUT EDIT ('THIRD ORDER POLY. IN SQRT. VEL.')
                                (COL(10),A);
            WHEN (4) PUT EDIT ('FOURTH ORDER POLY. IN SQRT. VEL.')
                                (COL(10),A);
            WHEN (5) PUT EDIT ('SECOND ORDER KING DERIVATION')
                                (COL(10),A);
            WHEN (6) PUT EDIT ('THIRD ORDER KING DERIVATION')
                                (COL(10),A);
            OTHERWISE;
            END;
        PUT SKIP (3) EDIT   ('FREE-STREAM VEL., FT/S',
                             'TEMP., DEG. F',
                             'BAR. PRESSURE, INCHES HG',
                             'REYNOLDS NO. PER FOOT')
                             (COL(6),F(8,3),COL(26),F(7),COL(46),F(8,4),
                             COL(71),F(14,6));
        PUT SKIP EDIT       (MAX_VEL,
                             TEMP_F,
                             BAR_PRES,
                             REL)
                             (COL(6),F(8,3),COL(26),F(7),COL(46),F(8,4),
                             COL(71),E(14,6));
        PUT SKIP (2) EDIT   ('LENGTH REYNOLDS NO.',
                             'DENSITY, SLUG/CUBIC FT',
                             'VISCOSITY, SLUG-FT/S',
                             'DIST. FROM LE, IN.')
                             (COL(2),A,COL(23),A,COL(47),A,COL(70),A);
        PUT SKIP EDIT       (REX,
                             DENSITY,
```

```
                                VISCOSITY,
                                LE_DISTANCE)
                                (COL(4),E(16,8),CPOL(27),F(10,7),COL(47),
                                E(16,8),COL(72),F(10,4));
            PUT SKIP (2) EDIT   ('DISP. THICKNESS, IN.',
                                'MOM. THICKNESS, IN.',
                                'H',
                                'DISP. THKNS. REYNOLDS NO.')
                                (COL(2),A,COL(25),A,COL(47),A,COL(59),A);
            PUT SKIP EDIT       (DISP_THK,
                                MOM_THK,
                                H,
                                RED)
                                (COL(2),E(16,8),COL(27),E(16,8),
                                COL(46),F(10,6),COL(60),E(16,8));
            PUT SKIP (3) EDIT   ('FRICTION POINT',
                                'FIRST PT. OF EST.',
                                'CLAUSER CF EST.',
                                'FRICTION VEL. EST')
                                (COL(2),A,COL(20),A,COL(40),A,COL(60),A);
            COEF_FRICTION = 2*(MOM_THK/3.6)**2;
            PUT SKIP EDIT       (FRICTION_POINT#,
                                PT1_CF,
                                COEF_FRICTION,
                                FRICTION_VEL)
                                (COL(6),F(3),COL(20),E(14,6),COL(40),E(14,6),
                                COL(60),E(14,6));
            PUT SKIP (2) EDIT   ('AVG. FRICTION COEF.',
                                'FRICTION COEF. STANDARD DEV.',
                                'AVG. FRICTION VEL., FT/S')
                                (COL(2),A,COL(25),A,COL(59),A);
            PUT SKIP EDIT       (AVG_FRICTION_COEF,
                                FRICTION_ST_DEV,
                                AVG_FRICTION_VEL)
                                (COL(4),E(14,6),COL(27),E(14,6),COL(62),E(14,6));
            PUT SKIP (2) EDIT   ('DELTA','G')   (COL(2),A,COL(25),A);
            PUT SKIP EDIT       (DELTA,G)   (COL(6),E(14,6),COL(28),E(14,6));
            DO WHILE (TABLE_FLAG);
                GET LIST (TABLE_CODE);
                CALL TABLE;
                GET LIST (TABLE_FLAG);
                END;
            END PRINT_DATA;

/*    TABLE  -  A PROCEDURE TO OUTPUT A PRINTED TABLE OF POINTS TAKEN      */
/*              IN THE BOUNDARY LAYER, REDUCED ACCORDING TO VARIOUS        */
/*              SCHEMES                                                    */
/*    ***********************************************************          */

TABLE:
    PROC;
        PUT PAGE;
```

```
SELECT (TABLE_CODE);
    WHEN (1)
        PUT LIST    ('POINT NO.',
                     'HEIGHT, INCHES',
                     'VELOCITY, FT/2',
                     'TURB. INT., %');
    WHEN (2)
        PUT LIST    ('POINT NO.',
                     'Y-THETA',
                     'U PRIME',
                     'U MINUS');
    WHEN (3)
        PUT EDIT    ('POINT NO.',
                     'Y PLUS',
                     'Y REYNOLDS NO.',
                     'U PLUS',
                     'PT. FRICTION COEF.')
                     (COL(2),A,COL(12),A,COL(32),A,
                     COL(52),A,COL(72),A);
    WHEN (4)
        PUT LIST    ('POINT NO.',
                     'Y MINUS',
                     'U MINUS',
                     'TURB. INT., %');
    OTHERWISE;
    END;
PUT SKIP;
DO I = 1 TO POINT#;
    SELECT (TABLE_CODE);
        WHEN (1) DO;
            PUT SKIP LIST    (I,
                              HEIGHT(I),
                              VEL(I),
                              TURB(I));
            END;
        WHEN (2) DO;
            YTHETA = HEIGHT(I)/DISP_THK;
            UPRIME = VEL(I)/MAX_VEL;
            UMINUS = (VEL(I)-MAX_VEL)/AVG_FRICTION_VEL;
            PUT SKIP LIST    (I,
                              YTHETA,
                              UPRIME,
                              UMINUS);
            END;
        WHEN (3) DO;
            YPLUS = HEIGHT(I)*AVG_FRICTION_VEL*DENSITY/(VISCOSITY
                                        *INCH_PER_FT;
            REY = HEIGHT(I)*MAX_VEL*DENSITY/(VISCOSITY*INCH_PER_FT);
            UPLUS = VEL(I)/AVG_FRICTION_VEL;
            IF POINT_FRICTION_COEF(I)>0
            THEN
                CF = POINT_FRICTION_COEF(I);
```

```
                    ELSE
                        CF = 0;
                    PUT SKIP EDIT    (I,
                                      YPLUS,
                                      REY
                                      UPLUS,
                                      CF)
                                      (COL(2),F(5),COL(12),E(14,7),CPOL(32),
                                      E(14,7),COL(52),E(14,7),COL(72),E(14,7));
                    END;
                WHEN (3) DO;
                    YMINUS = HEIGHT(I)/DELTA;
                    UMINUS = (VEL(I)-MAX_VEL)/AVG_FRICTION_VEL;
                    PUT SKIP LIST    (I,
                                      YMINUS,
                                      UMINUS,
                                      TURB(I));
                    END;
                OTHERWISE;
                END;
            END;
        END TABLE;

/*   ******************   */
/*   BUILT IN FUNCTIONS   */
/*   ******************   */

DECLARE
    LOG10 BUILTIN,
    SQRT BUILTIN;

/*   ******************   */
/*   PROGRAM CONSTANTS    */
/*   ******************   */

DECLARE
    INCH_PER_CT FLOAT DEC(8) INIT(1.046E-3),
    INCH_PER_FT FLOAT DEC(8) INIT(12.);

/*   ******************   */
/*   PROGRAM FLAGS        */
/*   ******************   */

DECLARE
    BL_PROCESS_FLAG BIT(1),
    CORR_FLAG BIT(1),
    TABLE_FLAG BIT(1);

/*   ******************   */
/*   PROGRAM VARIABLES    */
/*   ******************   */
```

```
DECLARE
    A(10) FLOAT DEC(10),
    AVG_FRICTION_COEF FLOAT DEC(8),
    AVG_FRICTION_VEL FLOAT DEC(8),
    B(10) FLOAT DEC(10),
    B_A FLOAT DEC(8),
    BAR_PRES FLOAT DEC(8),
    C(10) FLOAT DEC(10),
    CAL_CODE FIXED BIN(15),
    CF FLOAT DEC(8),
    COEF_FRICTION FLOAT DEC(8),
    D(10) FLOAT DEC(10),
    DELTA FLOAT DEC(8),
    DELTA_CF FLOAT DEC(8),
    DELTA_Y FLOAT DEC(8),
    DENSITY FLOAT DEC(8),
    DISP_DER FLOAT DEC(10),
    DISP_THK FLOAT DEC(8),
    DISP_THK_SUM FLOAT DEC(10),
    DUM FLOAT DEC(8),
    D1 FLOAT DEC(10),
    E(10) FLOAT DEC(10),
    E_SQUARE FLOAT DEC(8),
    ERROR FLOAT DEC(8),
    EXPONENT FLOAT DEC(8),
    E4 FLOAT DEC(8),
    FPS FLOAT DEC(8),
    FRICTION_POINT# FIXED BIN(15),
    FRICTION_ST_DEV FLOAT DEC(8),
    FRICTION_SUM FLOAT DEC(10),
    FRICTION_VEL FLOAT DEC(8),
    G FLOAT DEC(8),
    H FLOAT DEC(10),
    HEIGHT(150) FLOAT DEC(8),
    I FIXED BIN(15),
    J FIXED BIN(15),
    K1 FIXED BIN(15),
    K5 FIXED BIN(15),
    LAW_WALL_BOT# FIXED BIN(15),
    LAW_WALL_TOP# FIXED BIN(15),
    LE_DISTANCE FLOAT DEC(8),
    MAX_VEL FLOAT DEC(8),
    MEAN_VOLTAGE(150) FLOAT DEC(8),
    MOM_DER FLOAT DEC(10),
    MOM_THK FLOAT DEC(8),
    MOM_THK_SUM FLOAT DEC(10),
    OLD_DISP_DER FLOAT DEC(10),
    OLD_MOM_DER FLOAT DEC(10),
    POINT_FRICTION_COEF(150) FLOAT DEC(8),
    POINT# FIXED BIN(15),
    PT1_CF FLOAT DEC(8),
    RED FLOAT DEC(8),
```

```
REL FLOAT DEC(8),
REX FLOAT DEC(8),
REY FLOAT DEC(8),
RMS_VOLTS FLOAT DEC(8),
RMS_VOLTAGE(150) FLOAT DEC(8),
SENSITIVITY FLOAT DEC(8),
SLOPE FLOAT DEC(8),
STEP FIXED BIN(15),
TABLE_CODE FIXED BIN(15),
TEMP_F FLOAT DEC(8),
TI FLOAT DEC(8),
TOP# FIXED BIN(15),
TURB(150) FLOAT DEC(8),
UMINUS FLOAT DEC(8),
UPLUS FLOAT DEC(8),
UPRIME FLOAT DEC(8),
VEL(150) FLOAT DEC(8),
VEL_CORR FLOAT DEC(6),
VEL_RATIO FLOAT DEC(8),
VISCOSITY FLOAT DEC(8),
VOLTS FLOAT DEC(8),
V1 FLOAT DEC(8),
V5 FLOAT DEC(8),
W3 FLOAT DEC(8),
YPLUS FLOAT DEC(8),
YMINUS FLOAT DEC(8),
YPLUS FLOAT DEC(8),
YTHETA FLOAT DEC(8),
ZERO_CT_HEIGHT FLOAT DEC(8);

END BLSOL;
```

PROGRAM 'GCAL'

GCAL is a PL/I procedure which takes hot-wire probe calibration data and fits specific calibration functions to that data. The calibration function supported by the program are:

Exponent Law

Second Order Polynomial in V^2

Third Order Polynomial in V^2

Fourth Order Polynomial in V^2

Second Order King Series

Third Order King Series

Corrections for the influence of the probe stem and orientation on the hot wire can also be applied in the calibration solution. The different functions are fitted to the data using least squares techniques, and the standard deviation of the data from the fitted curve is computed. The program also has the capability of solving velocity and turbulence for a set of data given a previous calibration fit, or functional coefficients as part of the input stream.

Extensive details of the program operation are provided by comments in the program itself in Figure 66.

Figure 66 - GCAL computer program listing

```
/*   GCAL -  A GENERAL CALIBRATION AND DATA REDUCTION PROGRAM FOR      */
/*           HOTWIRES                                                 */
/*                                                                    */
/*        THIS PROGRAM ACCEPTS DATA SETS OF HOT-WIRE CALIBRRATION     */
/*   DATA OR HOT-WIRE READING TO BE REDUCED.  SEVERAL CALIBRATION     */
/*   METHODS ARE SUPPORTED.                                           */
/*        THE MAIN PROGRAM ONLY READS A BIT FLAG TO CHOOSE CALI-      */
/*   BRATION OR REDUCTION.  SEE PROCEDURE LISTINGS FOR FURTHER DETAIL */
/*        CALIBRATION METHODS SUPPORTED ARE:                          */
/*                    1 - EXPONENT LAW                                */
/*                    2 - SECOND ORDER SQRT. VEL.                     */
/*                    3 - THIRD ORDER SQRT. VEL.                      */
/*                    4 - FOURTH ORDER SQRT. VEL.                     */
/*                    5 - SECOND ORDER KING SERIES                    */
/*                    6 - THIRD ORDER KING SERIES                     */
/*                                                                    */
/*                + + + + + + + + INPUT - STREAM + + + + + + + +      */
/*        1. CALIBRATE - BIT FLAG                                     */
/*                    IF '1' RUN CALIBRATION FIT                      */
/*                    IF '0' RUN DATA REDUCTION                       */
/*                                                                    */
/*                + + + + + + + + + + + + + + + + + + + + + + + +     */
/*                                                                    */
/*        CALLS:      CALIBRATE_PROG, PROCEDURE                       */
/*                    INTERPRET_DATA, PROCEDURE                       */
/*                                                                    */
/*   ************************************************************     */

GCAL:
    PROC OPTIONS(MAIN);
        A = 0;
        B = 0;
        C = 0;
        D = 0;
        E = 0;
        EOF = '0'B;
        ON ENDFILE (SYSIN) EOF = '1'B;
        DO UNTIL (EOF);
            GET LIST (CALIBRATE);
            IF CALIBRATE
            THEN
                CALL CALIBRATE_PROG;
            ELSE
                CALL INTERPRETDATA;
            END;

/*   INTERPRET_DATA  -  A HOT-WIRE DATA REDUCTION PROCEDURE  *******   */
/*                                                                    */
/*        REDUCTION IS ACCOMPLISHED BY A VARIETY OF CALIBRATION FIT   */
```

```
/*    FUNCTIONS.  THE FUNCTION CONSTANTS CAN BE THE RESULT OF A PREVIOUS  */
/*    FIT OR NEW CONSTANTS MAY BE SPECIFIED.                              */
/*                                                                        */
/*    + + + + + + + + + + + INPUT FORMAT - STREAM + + + + + + + + + + + + */
/*                                                                        */
/*    1. CONSTANT_FLAG (BIT FLAG), CAL_CODE, CORR_FLAG                    */
/*                                                                        */
/*          CONSTANT_FLAG:  IF '1' USE PREVIOUSLY CALCULATED CONSTANTS    */
/*                          IF '0' THEN READ IN CONSTANTS                 */
/*                                                                        */
/*          CAL_CODE:       1 - EXPONENT LAW                              */
/*                          2 - SECOND ORDER SQRT VEL.                    */
/*                          3 - THIRD ORDER SQRT. VEL.                    */
/*                                                                        */
/*    1.1 IF CONSTANT_FLAG = '0' (READ IN CONSTANTS) INPUT CONSTANTS      */
/*          A THRU E AND EXPONENT. IF CONSTANT OR EXPONENT NOT USED       */
/*          INPUT ZERO IN ITS PLACE.                                      */
/*                                                                        */
/*    1.2 IF CORR_FLAG = '1' AND CONSTANT_FLAG = '1' (VELOCITY CORRECTION */
/*          APPLIED AND NEW CONSTANTS) INPUT VEL_CORR                     */
/*                                                                        */
/*    2. INPUT EACH DATA POINT: MEAN VOLTAGE, RMS VOLTAGE.                */
/*                                                                        */
/*    + + + + + + + + + + + + + OUTPUT FORMAT + + + + + + + + + + + + +   */
/*                                                                        */
/*                              CALIBRATION METHOD                        */
/*    (TABLE)                                                             */
/*    VOLTAGE     RMS VOLTAGE     VELOCITY     TURBULENCE INTENSITY       */
/*                                                                        */
/*    + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + */
/*                                                                        */
/*    CALLS:    VELOCITY, PROCEDURE                                       */
/*              TURBULENCE, PROCEDURE                                     */
/*                                                                        */
/*    ********************************************************          */

INTERPRET_DATA:
     PROC;
        VEL_CORR = 1.00000;
        GET LIST (CONSTANT_FLAG,CAL_CODE,CORR_FLAG);
        IF ¬CONSTANT_FLAG
        THEN DO;
            GET LIST    (A(CAL_CODE),
                         B(CAL_CODE),
                         C(CAL_CODE),
                         D(CAL_CODE),
                         E(CAL_CODE),
                         EXPONENT);
            IF CORR_FLAG
            THEN
                GET LIST(VEL_CORR);
            ELSE;
```

```
            END;
        ELSE;
        PUT PAGE LINE(5) LIST (' ','DATA REDUCTION');
        PUT SKIP(2);
        SELECT (CAL_CODE);
            WHEN (1) PUT LIST (' ','EXPONENT LAW'II(7)' 'II'EXP = ',
                                                    EXPONENT);
            WHEN (2) PUT LIST (' ',' ','SECOND ORDER SQRT VELOCITY');
            WHEN (3) PUT LIST (' ',' ','THIRD ORDER SQRT VELOCITY');
            WHEN (4) PUT LIST (' ',' ','FOURTH ORDER SQRT VELOCITY');
            WHEN (5) PUT LIST (' ',' ',
                            'SECOND ORDER KING''S LAW DERIVATION');
            WHEN (6) PUT LIST (' ',' ',
                            'THIRD ORDER KING''S LAW DERIVATION');
            OTHERWISE PUT LIST ('***** CODE NOT VALID *****');
            END;
        IF CORR_FLAG
        THEN
            PUT SKIP (2) LIST (' ','VEL. CORRECTION FACTOR = ',VEL_CORR);
        ELSE;
        PUT SKIP (2) LIST ('VOLTAGE','RMS VOLTAGE','VELOCITY, FPS',
                            'TURBULENCE INTENSITY, %');
        PUT SKIP;
        DO UNTIL (VOLTS = 0);
            GET LIST (VOLTS,RMS_VOLTS);
            CALL TURBULENCE;
            IF CORR_FLAG
            THEN
                FPS = FPS*(1/VEL_CORR);
            ELSE;
            IF FPS>0
            THEN
                TI = 100*RMS_VOLTS*SENSITIVITY/FPS;
            ELSE
                TI = 0;
            PUT SKIP LIST (VOLTS,RMS_VOLTS,RPS,TI);
            END;
        END INTERPRET_DATA;

/*    VELOCITY  -  A CALCULATION OF VELOCITY FROM HOT-WIRE MEAN VOLTAGE   */
/*                                                                        */
/*      ENTER WITH CALCODE, VOLTS, AND CALIBRATION CONSTANTS ( A TO E)    */
/*                                                                        */
/*      EXIT WITH FPS (VELOCITY)                                          */
/*                                                                        */
/*    ******************************************************************  */

VELOCITY:
    PROC;
        SELECT (CAL_CODE);
            WHEN (1)
            DO;
```