```
        DUM = (VOLTS**2-A(CAL_CODE))/B(CAL_CODE);
        IF DUM<0
        THEN
            FPS = 0;
        ELSE
            FPS = DUM**(1/EXPONENT);
        END;
    WHEN (2)
    DO;
        DUM = B(CAL_CODE)*B(CAL_CODE)-4*C(CAL_CODE)*
                                    (A(CAL_CODE)-VOLTS**2);
        IF DUM<0
        THEN
            FPS = 0;
        ELSE
            DO;
                DUM = (-B(CAL_CODE) + SQRT(DUM))/(2*C(CAL_CODE));
                FPS = DUM*DUM;
                END;
        END;
    OTHERWISE
        IF (CAL_CODE>2 & CAL_CODE<7)
        THEN DO;
            V1 = 0;
            E4 = VOLTS**2;
            D1 = 1;
            DO UNTIL (D1<0.0000001 | SLOPE<0);
                DO K1 = 1 TO 10000 UNTIL (W3>E4);
                    FPS = V1 + D1*K1;
                    SELECT (CAL_CODE);
                        WHEN (3)
                            W3 = A(CAL_CODE) + B(CAL_CODE)*FPS**0.5
                                    + C(CAL_CODE)*FPS + D(CAL_CODE)*FPS**1.5;
                        WHEN (4)
                            W3 = A(CAL_CODE) + B(CAL_CODE)*FPS**0.5
                                    + C(CAL_CODE)*FPS + D(CAL_CODE)*FPS**1.5
                                    + D(CAL_CODE)*FPS*FPS;
                        WHEN (5)
                            W3 = A(CAL_CODE) + B(CAL_CODE)*FPS**0.5
                                    + C(CAL_CODE)*FPS**1.5;
                        WHEN (6)
                            W3 = A(CAL_CODE) + B(CAL_CODE)*FPS**0.5
                                    + C(CAL_CODE)*FPS**1.5
                                    + D(CAL_CODE)*FPS**2.5;
                        OTHERWISE;
                        END;
                    END;
                V1 = FPS-2*D1;
                IF V1<0;
                THEN
                    V1 = 0;
                ELSE;
```

```
                    D1 = D1/10;
                    V5 = FPS**0.5;
                    SELECT (CAL_CODE);
                        WHEN (3)
                            SLOPE = 0.5*B(3)/V5 + C(3) + 1.5*D(CAL_CODE)*V5;
                        WHEN (4)
                            SLOPE = 0.5*B(4)/V5 + C(4) + 1.5*D(4)*V5 + 2*E(4)*FPS;
                        WHEN (5)
                            SLOPE = 0.5*B(5)/V5 + 1.5*C(5)*V5;
                        WHEN (6)
                            SLOPE = 0.5*B(6)/V5 + 1.5*C(6)*V5 + 2.5*V5**3;
                        OTHERWISE;
                        END;
                    END;
                END;
            END VELOCITY;

/*    TURBULENCE -  A CALCULATION OF TURB. INTENSITY AND SENSITIVITY          */
/*                  OF THE HOT-WIRE, 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;

/*  CALIBRATION_PROG  -  A CALCULATION OF CALIBRATION FUNCTION FOR A    */
/*                       HOT-WIRE ANEMOMETER, PROCEDURE                 */
/*                                                                      */
/*    + + + + + + + + + + + INPUT FORMAT - STREAM + + + + + + + + + + + */
/*                                                                      */
/*      1.   CAL_CODE, NEW_DATA (BIT FLAG), PRT_TABLE, CORR_FLAG,       */
/*           VEL_CORR (OPTIONAL)                                        */
/*               CAL_CODE DETERMINES FUNCTION:                          */
/*                                   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        */
/*               NEW_DATA DETERMINES DATA USED:                         */
/*                                   IF '1' INPUT NEW CALIBRATION DATA  */
/*                                   IF '0' USE DATA CURRENTLY IN STORAGE*/
/*               PRT_TABLE:                                             */
/*                                   IF '1' PRINT CAL. DATA TABLE       */
/*                                   IF '0' DO NOT PRINT TABLE          */
/*               CORR_FLAG:                                             */
/*                                   IF '1' VELOCITY CORRECTION APPLIED */
/*                                   IF '0' NO VELOCITY CORRECTION APPLIED*/
/*               VEL_CORR:     IF CORR_FLAG = '1' THIS FACTOR MUST BE   */
/*                             SUPPLIED. (TRUE VELOCITY)*(VEL_CORR) IS  */
/*                             THE VELOCITY THE HOT-WIRE SENSES         */
/*                                                                      */
/*      1.1  EXPONENT                                                   */
/*               INPUT ONLY IF EXPONENT LAW CALIBRATION                 */
/*                                                                      */
/*      1.2  IF INPUT NEW CALIBRATION DATA THEN THE FOLLOWING IS SUPPLIED*/
/*                                                                      */
/*      1.2.1    DATE (CHAR(8), AS 09/11/80),                           */
/*               CAL# (CALIBRATION RUN NUMBER FOR THAT DAY),            */
/*               POINT# (NUMBER OF DATA POINTS),                        */
```

```
/*                TEMP(ATMOS. TEMP., DEGREES FAHRENHEIT),                  */
/*                PRES(ATMOS. BARO. PRESSURE, INCHES MERCURY),             */
/*                PROBE# (CHAR(4), EG. "1P"),                              */
/*                MANO_FLAG (BIT FLAG)                                     */
/*                     IF MANO_FLAG = '1' THEN VEL. IS COMPUTED FROM A     */
/*                          MANOMETER PRESSURE READING                     */
/*                     IF MANO_FLAG = '0' THEN VEL. IS GIVEN DIRECTLY      */
/*                                                                         */
/*      1.2.1.1   IF MANO_FLAG = '1' THEN THIS INPUT IS REQUIRED           */
/*                PITOT (BIT FLAG)                                         */
/*                     IF PITOT = '1' THEN PITOT-STATIC PROBE DATA FOR VEL.*/
/*                     IF PITOT = '0' THEN CHAMBER PRESSURE FROM HOT-WIRE  */
/*                          CALIBRATION RIG FOR VEL.                       */
/*                                                                         */
/*      1.2.2     VEL (CAN BE PRESSURE), MEAN_VOLTAGE, RMS_VOLTAGE         */
/*                     REPEAT AS NEEDED FOR EACH CALIBRATION DATA POINT.   */
/*                     IF MANOFLAG = '1' THEN MANOMETER READING IS INPUT   */
/*                          IN PLACE OF VELOCITY, FOR CONVERSION BY THE    */
/*                          PROGRAM.                                       */
/*   + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + */
/*                                                                         */
/*     CALLS:    POLYNOMIAL, PROCEDURE                                     */
/*               KING_SERIES, PROCEDURE                                    */
/*               PRINT_STATISTICS, PROCEDURE                               */
/*   ************************************************************          */

CALIBRATE_PROG:
     PROC;
        VEL_CORR = 1.000000;
        GET LIST (CAL_CODE, NEW_DATA, PRT_TABLE, CORR_FLAG);
        IF CORR_FLAG
        THEN
            GET LIST (VEL_CORR);
        ELSE;
        IF CAL_CODE = 1
        THEN
            GET LIST (EXPONENT);
        ELSE;
        IF NEW_DATA
        THEN DO;
            GET LIST (DATE,CAL#,POINT#,TEMP,PRES,PROBE#,MANO_FLAG);
            IF MANO_FLAG
            THEN
                GET LIST(PITOT);
            ELSE;
            DO INDEX = 1 TO POINT#;
                GET LIST (VEL(INDEX),
                          MEAN_VOLTAGE(INDEX),
                          RMS_VOLTAGE(INDEX));
                END;
            IF MANO_FLAG
            THEN
```

```
                DO INDEX = 1 TO POINT#;
                    CALL MANVEL;
                    END;
                ELSE;
                END;
            ELSE;
            IF CAL_CODE<5
            THEN DO;
                IF CAL_CODE>1
                THEN
                    EXPONENT = 0.5
                ELSE;
                CALL POLYNOMIAL;
                END;
            ELSE
                CALL KING_SERIES;
            CALL PRINT_STATISTICS;
            END CALIBRATE_PROG;

/*    MANVEL-  A PROCEDURE TO CALCULATE VELOCITY FROM MANOMETER          */
/*             READING                                                   */
/*                                                                       */
/*      ENTER WITH INDEX, VEL(INDEX) = MANOMETER READING, TEMP, PRES,    */
/*             PITOT (BIT FLAG)                                          */
/*                                                                       */
/*      EXIT WITH VEL(INDEX) = VELOCITY (FPS)                            */
/*    ***************************************************************    */

MANVEL:
    PROC;
        IF PITOT
        THEN
            VEL(INDEX) = 3.1525*SQRT((460 + TEMP)*VEL(INDEX)/PRES);
        ELSE
            VEL(INDEX) = 15.895*SQRT((460 + TEMP)*VEL(INDEX)/PRES);
        END MANVEL;

/*    POLYNOMIAL-  A PROCEDURE TO PERFORM LEAST SQUARES FIT OF POLY-     */
/*                 NOMIAL FUNCTION OF SQRT. VEL. AND VOLTS SQUARED       */
/*                                                                       */
/*      ENTER WITH DATA IN VEL(X), MEAN_VOLTAGE(X), CAL_CODE,            */
/*             VEL_CORR, AND POINT#                                      */
/*                                                                       */
/*      EXIT WITH FUNCTION COEFFICIENTS IN A(CAL_CODE) THRU E(CAL_CODE)  */
/*                                                                       */
/*      CALLS:   MAT_INVERT, PROCEDURE                                   */
/*               MAT_MULT, PROCEDURE                                     */
/*    ***************************************************************    */

POLYNOMIAL:
    PROC;
        ORDER = CAL_CODE;
```

```
        DA = 0;
        DO I = 1 TO ORDER + 1;
            DO J = 1 TO POINT#;
                FPS = VEL(J)*VEL_CORR;
                SELECT (I);
                    WHEN (1) DUM = 1;
                    WHEN (2) DUM = FPS**EXPONENT;
                    WHEN (3) DUM = FPS;
                    WHEN (4) DUM = FPS**1.5;
                    WHEN (5) DUM = FPS**2;
                    OTHERWISE;
                    END;
                DA(I) = DA(I) + DUM*MEAN_VOLTAGE(J)**2;
                END;
            END;
        FA = 0;
        DO I = 1 TO ORDER + 1;
            DO J = 1 TO ORDER + 1;
                SUM = 0;
                DO K = 1 TO POINT#;
                    FPS = VEL(K)*VEL_CORR;
                    SELECT (I);
                        WHEN (1) DUM1 = 1;
                        WHEN (2) DUM1 = FPS**EXPONENT;
                        WHEN (3) DUM1 = FPS;
                        WHEN (4) DUM1 = FPS**1.5;
                        WHEN (5) DUM1 = FPS**2;
                        OTHERWISE;
                        END;
                    SELECT (J);
                        WHEN (1) DUM2 = 1;
                        WHEN (2) DUM2 = FPS**EXPONENT;
                        WHEN (3) DUM2 = FPS;
                        WHEN (4) DUM2 = FPS**1.5;
                        WHEN (5) DUM2 = FPS**2;
                        OTHERWISE;
                        END;
                    FA(I,J) = FA(I,J) + DUM1*DUM2;
                    END;
                END;
            END;
        CALL MAT_INVERT;
        CALL MAT_MULT;
        A(CAL_CODE) = CA(1);
        B(CAL_CODE) = CA(2);
        C(CAL_CODE) = CA(3);
        D(CAL_CODE) = CA(4);
        E(CAL_CODE) = CA(5);
        END POLYNOMIAL;

/*  KING_SERIES  -  A PROCEDURE TO PERFORM LEAST SQUARES FIT OF      */
/*                  KING SERIES FUNCTIONS FOR HOT-WIRES              */
```

```
/*                                                                    */
/*      ENTER WITH CAL_CODE, VEL_CORR, AND DATA IN VEL(X) AND         */
/*              MEAN_VOLTAGE(X)                                       */
/*                                                                    */
/*      EXIT WITH FUNCTION COEFFICIENTS IN A(CAL_CODE) THRU D(CAL_CODE) */
/*                                                                    */
/*      CALLS:    MAT_INVERT, PROCEDURE                               */
/*                MAT_MULT, PROCEDURE                                 */
/*   ************************************************************     */

KING_SERIES:
    PROC;
        ORDER = CAL_CODE-3;
        DA = 0;
        DO I = 1 TO ORDER + 1;
            DO J = 1 TO POINT#;
                FPS = VEL(J)*VEL_CORR;
                SELECT (I);
                    WHEN (1) DUM = 1;
                    WHEN (2) DUM = FPS**0.5;
                    WHEN (1) DUM = FPS**1.5;
                    WHEN (1) DUM = FPS**2.5;
                    OTHERWISE;
                    END;
                DA(I) = DA(I) + DUM*MEAN_VOLTAGE(J)**2;
                END;
            END;
        FA = 0;
        DO I = 1 TO ORDER + 1;
            DO J = 1 TO ORDER + 1;
                SUM = 0;
                DO K = 1 TO POINT#;
                    FPS = VEL(K)*VEL_CORR;
                    SELECT (I);
                        WHEN (1) DUM1 = 1;
                        WHEN (2) DUM1 = FPS**0.5;
                        WHEN (3) DUM1 = FPS**1.5;
                        WHEN (4) DUM1 = FPS**2.5;
                        OTHERWISE;
                        END;
                    SELECT (J);
                        WHEN (1) DUM2 = 1;
                        WHEN (2) DUM2 = FPS**0.5;
                        WHEN (3) DUM2 = FPS**1.5;
                        WHEN (4) DUM2 = FPS**2.5;
                        OTHERWISE;
                        END;
                    FA(I,J) = FA(I,J) + DUM1*DUM2;
                    END;
                END;
            END;
        CALL MAT_INVERT;
```

```
        CALL MAT_MULT;
        A(CAL_CODE) = CA(1);
        B(CAL_CODE) = CA(2);
        C(CAL_CODE) = CA(3);
        D(CAL_CODE) = CA(4);
        END KING_SERIES;

/*    MAT_MULT  -  A PROCEDURE SPECIALIZED TO MULTIPLY TWO MATRICES      */
/*                 FORMED ACCORDING TO LEAST SQUARES METHODS, TO       */
/*                 OBTAIN THE COEFFICIENT MATRIX OF THE FUNCTION       */
/*                                                                     */
/*      ENTER WITH FA(ORDER + 1,ORDER + 1), DA(ORDER + 1), ORDER       */
/*                                                                     */
/*      EXIT WITH CA(ORDER + 1), THE COEFFICIENT MATRIX                */
/*    ***************************************************************  */

MAT_MULT:
    PROC;
        CA = 0;
        DO I = 1 TO ORDER + 1;
            DO K = 1 TO ORDER + 1;
                CA(I) = CA(I) + FA(I,K)*DA(K);
                END;
            END;
        END MAT_MULT;

/*    MAT_INVERT  -  A PROCEDURE TO INVERT A MATRIX IN PLACE USING THE   */
/*                   GAUSS-JORDAN METHOD                                 */
/*                                                                       */
/*      ENTER WITH FA(ORDER,ORDER), ORDER                                */
/*                                                                       */
/*      EXIT WITH FA(ORDER,ORDER) INVERTED                               */
/*    *****************************************************************  */

MAT_INVERT:
    PROC;
        DO K = 1 TO ORDER + 1;
            DO J = 1 TO ORDER + 1;
                IF J ¬= K
                THEN
                    FA(K,J) = FA(K,J)/FA(K,K);
                ELSE;
                END;
            FA(K,K) = 1/FA(K,K);
            DO I = 1 TO ORDER + 1;
                IF I ¬= K
                THEN DO J = 1 TO ORDER + 1;
                    IF J ¬= K
                    THEN
                        FA(I,J) = FA(I,J)-FA(K,J)*FA(I,K);
                    ELSE;
                    END;
```

```
            ELSE;
            END;
        DO I = 1 TO ORDER + 1;
            IF I ¬ = K
            THEN
                FA(I,K) = -FA(I,K)*FA(K,K);
            ELSE;
            END;
        END;
    END MAT_INVERT;

/*  PRINT_STATISTICS    - A PROCEDURE TO OUTPUT RESULTS OF CALIBRATION    */
/*                        FIT                                             */
/*                                                                        */
/*  + + + + + + + + + + + + + + OUTPUT FORMAT + + + + + + + + + + + + + + */
/*                                                                        */
/*                        CALIBRATION METHOD                              */
/*                                                                        */
/*     CAL. DATE     CAL. #      POINT #     PROBE #                      */
/*                                                                        */
/*     TEMPERATURE               PRESSURE                                 */
/*                                                                        */
/*                          CONSTANTS                                     */
/*     A                    B                        C                   */
/*                                                                        */
/*     D                    E                        EXPONENT            */
/*                                                                        */
/*     STANDARD DEVIATION              CORRECTION FACTOR                  */
/*                                                                        */
/*      (TABLE)                                                           */
/*   TRUE VEL.    CAL. VEL.     MEAN VOLT.   RMS VOLT.   TURB. INT.   SENS. */
/*  + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + */
/*                                                                        */
/*     CALLS:   STANDARD_DEVIATION, PROCEDURE                             */
/*              VELOCITY, PROCEDURE                                       */
/*              TURBULENCE, PROCEDURE                                     */
/*  ************************************************************          */

PRINT_STATISTICS:
    PROC;
        CALL STANDARD_DEVIATION;   /* RETURNS SIGMA */
        PUT PAGE;
        IF NEW_DATA
        THEN DO;
            PUT SKIP(3) EDIT   ('CALIBRATION DATE','CALIBRATION NUMBER',
                               '# OF DATA POINTS','PROBE #')
                               (COL(10),A,COL(30),F(4),COL(50),F(4),COL(70),A);
            PUT SKIP EDIT      (DATE,CAL#,POINT#,PROBE#)
                               (COL(10),A,COL(30),F(4),COL(50),F(4),COL(70),A);
            PUT SKIP(2) EDIT   ('TEMP., DEG. F.','PRESSURE, IN. HG')
                               (COL(10),A,COL(50),A)
            PUT SKIP EDIT      (TEMP,PRES)  (COL(10),F(5,1),COL(50),F(8,3));
```

```
    END;
ELSE;
PUT SKIP;
SELECT (CAL_CODE);
    WHEN (1) PUT LIST (' ','EXPONENT LAW');
    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 SERIES');
    WHEN (6) PUT LIST (' ','THIRD ORDER KING SERIES');
    OTHERWISE PUT LIST (' ','******  CODE NOT VALID  ******');
    END;
PUT SKIP(3) LIST (' ','CONSTANTS');
PUT SKIP(2) EDIT    ('A','B','C')
                    (COL(10),A,COL(30),A,COL(50),A);
PUT SKIP EDIT       (A(CAL_CODE),B(CAL_CODE),C(CAL_CODE))
                    (COL(10),E(16,7,8),COL(30),E(16,7,8),COL(50),
                    E(16,7,8));
PUT SKIP(2) EDIT ('D','E')  (COL(10),A,COL(30),A);
IF CAL_CODE = 1
THEN
    PUT EDIT ('EXPONENT')  (COL(50),A);
ELSE;
PUT SKIP EDIT       (D(CAL_CODE),E(CAL_CODE))
                    (COL(10),E(16,7,8),COL(30),E(16,7,8));
IF CAL_CODE = 1
THEN
    PUT EDIT (EXPONENT)  (COL(50),F(9,6));
ELSE;
PUT SKIP(2) EDIT    ('STANDARD DEVIATION (FPS)','CORRECTION FACTOR')
                    (COL(10),A,COL(40),A);
PUT SKIP EDIT       (SIGMA,VELCORR)  (COL(10),F(9,4),COL(45),F(9,6));
IF PRT_TABLE
THEN DO;
    PUT SKIP(3) EDIT    ('TRUE VEL., FPS',
                         'CAL. BEL., FPS',
                         'MEAN VOLTAGE',
                         'RMS VOLTAGE',
                         'SENS., FPS/VOLT')
                         (COL(10),A,COL(26),A,COL(42),A,
                         COL(56),A,COL(70),A,COL(84),A);
    PUT SKIP;
    DO INDEX = 1 TO POINT#;
        VOLTS = MEAN_VOLTAGE(INDEX);
        RMS_VOLTS = RMS_VOLTAGE(INDEX);
        CALL TURBULENCE;
        IF CORR_FLAG
        THEN DO;
            FPS = FPS*(1/VEL_CORR);
            IF FPS>0
            THEN
                TI = 100*RMS_VOLTS*SENSITIVITY/FPS;
```

```
                    ELSE
                        TI = 0;
                    END;
                ELSE;
                PUT EDIT        (VEL(INDEX),
                                 FPS,
                                 MEAN_VOLTAGE(INDEX),
                                 RMS_VOLTAGE(NDEX),
                                 TI,
                                 SENSITIVITY)
                                 COL(12),F(8,2),
                                 COL(28),F(8,2),
                                 COL(44),F(6,3),
                                 COL(56),F(10,6),
                                 COL(72),F(7,2),
                                 COL(86),F(8,3));
                END;
            END;
        ELSE;
        END PRINT_STATISTICS;

/*   STANDARD_DEVIATION -  A PROCEDURE TO FIND THE STANDARD DEVIATION   */
/*                         OF A CALIBRATION FIT                         */
/*                                                                      */
/*     ENTER WITH CAL_CODE, MEAN_VOLTAGE(X), VEL(X), CORR_FLAG, VEL_CORR */
/*                                                                      */
/*     EXIT WITH SIGMA (STANDARD DEVIATION, FPS)                        */
/*                                                                      */
/*     CALLS:    VELOCITY, PROCEDURE                                    */
/*   ********************************************************           */

STANDARD_DEVIATION:
    PROC;
        SUM = 0;
        DO I = 1 TO POINT#;
            VOLTS = MEAN_VOLTAGE(I);
            CALL VELOCITY;
            IF CORR_FLAG
            THEN
                FPS = FPS*(1/VEL_CORR);
            ELSE;
            SUM = SUM + (VEL(I)-FPS)**2;
            END;
        SIGMA = SQRT(SUM/(POINT#-1));
        END STANDARD_DEVIATION;

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

DECLARE
    CALIBRATE BIT(1),
```

```
      CONSTANT_FLAG BIT(1),
      CORR_FLAG BIT(1),
      EOF BIT(1),M
      MANO_FLAG BIT(1),
      NEW_DATA BIT(1),
      PITOT BIT(1),
      PRT_TABLE BIT(1);

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

DECLARE
      A(10) FLOAT DEC(16),
      B(10) FLOAT DEC(16),
      C(10) FLOAT DEC(16),
      CA(5) FLOAT DEC(16),
      CAL_CODE FIXED DEC(2),
      CAL# FIXED DEC(2),
      D(10) FLOAT DEC(16),
      DA(5) FLOAT DEC(16),
      DATE CHAR(8),
      DUM FLOAT DEC(16),
      DUM1 FLOAT DEC(16),
      DUM2 FLOAT DEC(16),
      D1 FLOAT DEC(16),
      E(10) FLOAT DEC(16),
      EXPONENT FLOAT DEC(16),
      E4 FLOAT DEC(16),
      FA(5,5) FLOAT DEC(16),
      FPS FLOAT DEC(16),
      I FIXED DEC(3),
      INDEX FIXED DEC(3),
      I1 FIXED DEC(3),
      I2 FIXED DEC(3),
      J FIXED DEC(3),
      K FIXED DEC(3),
      K1FIXED DEC(8),
      MEAN_VOLTAGE(40) FLOAT DEC(16),
      ORDER FIXED DEC(2),
      POINT# FIXED DEC(2),
      PRES FLOAT DEC(8),
      PROBE# CHAR(4),
      RMS_VOLTS FLOAT DEC(8),
      RMS_VOLTAGE(40) FLOAT DEC(8),
      S FLOAT DEC(16),
      SENSITIVITY FLOAT DEC(8),
      SIGMA FLOAT DEC(8),
      SLOPE FLOAT DEC(6),
      SUM FLOAT DEC(16),
      TEMP FLOAT DEC(8),
      TI FLOAT DEC(8),
```

```
VEL(40) FLOAT DEC(16),
VEL_CORR FLOAT DEC(8),
VOLTS FLOAT DEC(16),
V1 FLOAT DEC(16),
V5 FLOAT DEC(16),
W3 FLOAT DEC(16),
Z FLOAT DEC(16);

END GCAL;
```