NONLINEAR ELECTRONICS (NOEL) PACKAGE 5: CANONICAL PIECEWISE-LINEAR MODELING

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/32

21 April 1986

NONLINEAR ELECTRONICS (NOEL) PACKAGE 5:

CANONICAL PIECEWISE-LINEAR MODELING

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/32

21 April 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

NONLINEAR ELECTRONICS (NOEL) PACKAGE 5:

CANONICAL PIECEWISE-LINEAR MODELING

by

An-Chang Deng and Leon O. Chua

Memorandum No. UCB/ERL M86/32

21 April 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

NOEL PACKAGE 5 : CANONICAL PIECEWISE-LINEAR MODELING†

An-Chang Deng and Leon O. Chua

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, CA 94720

ABSTRACT

The programs in this package construct the canonical piecewise-linear model

$$y = a + bx + \sum_{i=1}^{\sigma} c_i \mid x - \beta_i \mid$$

for a 2-terminal device; or

$$y_1 = a_1 + b_{11}x_1 + b_{12}x_2 + \sum_{i=1}^{\sigma} c_{1i} \mid \alpha_i x_1 - x_2 + \gamma_i \mid$$

$$y_2 = a_2 + b_{21}x_1 + b_{22}x_2 + \sum_{i=1}^{\sigma} c_{2i} \mid \alpha_i x_1 - x_2 + \gamma_i \mid$$

for a 3-terminal or 2-port device. The coefficients of each model are optimally chosen such that the approximation error between the measured data and the canonical piecewise-linear model is minimized.

March 26, 1986

† Research supported by the Office of Naval Research under Contract N00014-76-C-0572, and by the National Science Foundation under Grant ECS-8313278.

NOEL PACKAGE 5 : CANONICAL PIECEWISE-LINEAR MODELING

1. Introduction

To take advantage of the remarkable computational efficiency of the canonical piecewise-linear approach (dc analysis in package 6 and transient analysis in package 7)[1,2], we need a preliminary step to construct the canonical piecewise-linear model for each nonlinear device; namely

$$y = a + bx + \sum_{i=1}^{\sigma} c_i \mid x - \beta_i \mid \tag{1}$$

for a 2-terminal device; or

$$y_1 = a_1 + b_{11}x_1 + b_{12}x_2 + \sum_{i=1}^{\sigma} c_{1i} \mid \alpha_i x_1 - x_2 + \gamma_i \mid \tag{2a}$$

$$y_2 = a_2 + b_{21}x_1 + b_{22}x_2 + \sum_{i=1}^{\sigma} c_{2i} \mid \alpha_i x_1 - x_2 + \gamma_i \mid \tag{2b}$$

for a 3-terminal or 2-port device. A unified parameter optimization algorithm for constructing such models has been presented in [3] and is to be implemented by this software package.

Starting from the measurement of the terminal v-i characteristic of the device, we fit these measured data into the compact global piecewise-linear function Eqs.(1) or (2) for an optimal canonical piecewise-linear representation with the optimal parameters a , b , c_i , β_i in Eq.(1), or a_1 , a_2 , b_{11} , b_{12} , b_{21} , b_{22} , c_{1i} , c_{2i} , α_i , γ_i in Eq.(2), such that the approximation error between the canonical piecewise-linear model and the measured data is minimized. In this way, the internal physical phenomena of the device is not required as long as the terminal voltage and current can be accurately measured. Three distinct programs are included in this package; namely, **pwlmod** for constructing the 1-dimensional canonical piecewise-linear model Eq.(1); **pwlmod2** for 2-dimensional canonical piecewise-linear function Eq.(2); **pwlmod1** for a two dimension to one dimension mapping $f : \mathbf{R}^2 \rightarrow \mathbf{R}^1$, such as the single equation Eq.(2a) or Eq.(2b).

2. Algorithm

All the following equation numbers are referring to the equations in the report[3].

2.1. 1-dimensional canonical piecewise-linear modeling

Step 1.

Choose an initial set of breakpoints

$$z_2^{(k)} = \{\beta_1^{(k)}, \beta_2^{(k)}, \cdots, \beta_\sigma^{(k)}\}$$

Set k=0.

Step 2.

Construct the matrix A in Eq.(2.6) and solve Eq.(2.12) for the optimal parameters $z_1^*(z_2^{(k)})$ w.r.t. the set of breakpoints $z_2^{(k)}$.

Step 3.

Find the line search direction $s^{(k)}$ by Eqs.(2.15)-(2.21).

Step 4.

Perform the line search along $s^{(k)}$ to find $\alpha^{(k)}$ for the minimization problem of Eq.(2.26).

Step 5.

Compute the approximation error $E^{(k)}$ estimated at the minimum along $s^{(k)}$.

Step 6.

If $E^{(k)}$ is within the acceptable range then stop;
else increment k to k+1 and go to Step 2.

2.2. 2-dimensional canonical piecewise-linear modeling

Step 1.

Choose an initial set of partition boundaries $z_3^{(k)}$; k=0.

Step 2.

Solve Eq.(3.12) to Eq.(3.17) for $z_1^*(z_3^{(k)})$ and $z_2^*(z_3^{(k)})$.

Step 3.

Find the line search direction $s^{(k)}$ by Eqs.(3.20)-(3.26).

Step 4.

Perform the line search along $s^{(k)}$ to find the minimum in this line search direction.

Step 5.

Compute the approximation error $E(z^{(k)})$ by Eq.(3.8). If the error is within acceptable range, then stop the optimization process; else increment k to k+1 and go to Step 2 for a new line search direction.

2.3. 2-dimension to 1-dimension canonical piecewise-linear modeling

Procedures are the same as those of Section 2.2, except the optimization is corresponding to a single canonical piecewise-linear equation which is either Eq.(3.6a) or Eq.(3.6b).

3. User's Instruction

3.1. 1-dimensional canonical piecewise-linear modeling

Step 1.

Create a data file "xx...x.dat", which consists of m lines for m data points, and each line characterizes a data point with the following format :

xx...x yy...y ww...w

where *xx...x* , *yy...y* and *ww...w* are the ASCII codes for the x-coordinate, y-coordinate, and the weighting factor of the data point respectively.

Steps 2-5 are combined as a batch process and are executed by the command

getpwl xx...x

Step 2.

Type the command

pwlmod xx...x

to find the 1-dimensional canonical piecewise-linear model Eq.(1) which optimally fits the data points in "xx...x.dat". It proceeds with the following interactive procedures :

Enter the number of breakpoints

Enter an integer σ for a $(\sigma + 1)$-segment piecewise-linear partition. The piecewise-linear approximation gets better for larger σ.

Enter the initial set of breakpoints

Enter σ real numbers for the initial breakpoints $\beta_1^{(0)}$, $\beta_2^{(0)}$,...., $\beta_\sigma^{(0)}$.

approximation error = ...

.....

.....

A sequence of approximation errors are printed, each represents the approximation error of a canonical piecewise-linear model partitioned by the set of breakpoints $z_2^{(k)} + \alpha_i s^{(k)}$, for i=1,2,..., which is located along the line search direction $s^{(k)}$ for each i. The approximation error should decrease to a minimum, it then prints the canonical piecewise-linear model parameters corresponding to this minimum, and prompts a message

continue? y/n

Enter '**y**' to restart another iteration process for next line search direction $s^{(k+1)}$; otherwise enter '**n**' to stop the optimization process. The canonical piecewise-linear model found at this point is then printed to a file "xx...x.c" which lists the C source code of the optimal canonical piecewise-linear equation.

Step 3.

Compile the C source code "xx...x.c".

Step 4.

Link the object code "xx...x.o" with the graphic routines for drawing the canonical piecewise-linear model.

Step 5.

Type the command

pwldraw xx...x

to plot the canonical piecewise-linear model.

3.2. 2-dimensional canonical piecewise-linear modeling

Step 1.

Create a data file "xx...x.dat", which consists of m lines for m data points, and each line characterizes a data point with the following format :

$x_1x_1 \cdots x_1 \quad x_2x_2 \cdots x_2 \quad y_1y_1 \cdots y_1 \quad y_2y_2 \cdots y_2 \quad ww...w$

where $x_1x_1...x_1$, $x_2x_2 \cdots x_2$ are respectively the x_1 and x_2 coordinates of the data point in the R^2 domain space; $y_1y_1 \cdots y_1$, $y_2y_2 \cdots y_2$ are respectively the y_1 and y_2 coordinates in the R^2 range space; and w is the weighting factor of the data point.

Steps 2-5 are combined as a batch process and are executed by the command

getpwl2 xx...x

Enter the number of partition boundaries

Enter an integer σ for σ partition lines in the 2-dimensional plane.

Enter the initial partition boundaries

Enter σ sets of real numbers (m_1, t_1), (m_2, t_2), ..., (m_σ, t_σ) for the initial partition boundaries

$$m_i x_1 - x_2 + t_i = 0$$

for i=1,2,...,σ. Choose large m_i and t_i in case a vertical line is encountered.

approximation error = ...

.....

.....

A sequence of approximation errors are printed , each represents the approximation error of a canonical piecewise-linear model partitioned by the set of partition boundaries $z_3^{(k)} + \alpha_i s^{(k)}$, for i=1,2,..., which is located along the line search direction $s^{(k)}$ for each i. The approximation error should decrease to a minimum and it then prints the canonical piecewise-linear model parameters corresponding to this minimum, and prompts a message

continue? y/n

Enter 'y' to restart another iteration process for next line search direction $s^{(k+1)}$; otherwise enter 'n' to stop the optimization process. The 2-dimensional canonical piecewise-linear model found at this point is then printed to a file 'xx...x.c" which lists the C source code of the optimal canonical piecewise-linear equation.

Step 3.

Compile the source code "xx...x.c".

Step 4.

Link the object code "xx...x.o" with the graphic routines for drawing the canonical piecewise-linear model.

Step 5.

Type the command

pwldraw2 xx...x

to plot the canonical piecewise-linear model on the monitor with the following options :

Enter the option number

1 : y1 vs x1 for fixed x2

2 : y1 vs x2 for fixed x1

3 : y2 vs x1 for fixed x2

4 : y2 vs x2 for fixed x1

Enter the option number for various types of families of 1-dimensional piecewise-linear curves.

3.3. 2-dimension to 1-dimension canonical piecewise-linear modeling

Same as the procedures in Section 3.2, except the data file exhibits the following format :

$x_1 x_1 \cdots x_1 \quad x_2 x_2 \cdots x_2 \quad y_1 y_1 \cdots y_1 \quad ww...w$

The modeling process is executed by the batch command

getpwl1 xx...x

4. Output Format

4.1. 1-dimensional canonical piecewise-linear modeling

Both the data points and the 1-dimensional curve of the computed canonical piecewise-linear model are plotted in the color monitor for convenience of comparison. In addition to this graphic display, the optimal canonical piecewise-linear model equation is also shown in the file "xx...x.c" which is written as a C function.

4.2. 2-dimensional canonical piecewise-linear modeling

The family of the 1-dimensional piecewise-linear curves y_1(or y_2) = $f\ (x_1\ ,\ x_2)\ |_{\ x_2=p_j}$ or y_1(or y_2)=$f\ (x_1\ ,\ x_2)\ |_{\ x_1=p_j}$ are plotted for various values of p_j, j=1,2,..,k. Similar to the 1-dimensional case, the optimal canonical piecewise-linear model equation is written to the file "xx...x.c" as a C function.

5. Examples

Example 1 : in file "ex1.dat"

A pn junction diode (Fig.1).

Example 2 : in file "ex2.dat"

An MOSFET i_D vs (v_{GS} , v_{DS}) characteristic (Fig.2).

Example 3 : in file "ex3.dat"

A 2-dimensional piecewise-linear function (Fig.3).

6. Diagnosis

1. PWLMOD DATA_FILE

   Bad command line for 1-dimensional canonical piecewise-linear modeling, the correct one should be

   pwlmod xx...x

   where "xx...x.dat" is the filename for the data file.

2. PWLMOD1 DATA_FILE

   Bad command line for 2-dimension to 1-dimension canonical piecewise-linear modeling, the correct one should be

   pwlmod1 xx...x

   where "xx...x.dat" is the filename for the data file.

3. PWLMOD2 DATA_FILE

   Bad command line for 2-dimensional canonical piecewise-linear modeling, the correct one should be

   pwlmod2 xx...x

   where "xx...x.dat" is the filename for the data file.

4. CAN'T OPEN THE DATA FILE

   The data file "xx...x.dat" doesn't exist.

5. SINGULAR MATRIX

   The matrix A in Eq.(2.6) or Eq.(3.13) is singular.

6. IMPROPER BOUNDARY

   The partition boundary is outside the data region.

References

[1] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 6 : canonical piecewise-linear DC analysis," ERL Memo., UCB/ERL M86, University of California, Berkeley, 1986.

[2] A.C.Deng and L.O.Chua, "NOnlinear ELectronics package 7 : canonical piecewise-linear transient analysis," ERL Memo., UCB/ERL M86, University of California, Berkeley, 1986.

[3] L.O.Chua and A.C.Deng, "Canonical piecewise-linear modeling," ERL Memo., UCB/ERL M85/35, University of California, Berkeley, Apr. 26, 1986.

4.000e-01	4.802e-08	1.0
4.030e-01	5.390e-08	1.0
4.060e-01	6.049e-08	1.0
4.090e-01	6.789e-08	1.0
4.120e-01	7.619e-08	1.0
4.150e-01	8.551e-08	1.0
4.180e-01	9.597e-08	1.0
4.210e-01	1.077e-07	1.0
4.240e-01	1.209e-07	1.0
4.270e-01	1.357e-07	1.0
4.300e-01	1.523e-07	1.0
4.330e-01	1.709e-07	1.0
4.360e-01	1.918e-07	1.0
4.390e-01	2.152e-07	1.0
4.420e-01	2.415e-07	1.0
4.450e-01	2.711e-07	1.0
4.480e-01	3.042e-07	1.0
4.510e-01	3.415e-07	1.0
4.540e-01	3.832e-07	1.0
4.570e-01	4.301e-07	1.0
4.600e-01	4.827e-07	1.0
4.630e-01	5.417e-07	1.0
4.660e-01	6.080e-07	1.0
4.690e-01	6.823e-07	1.0
4.720e-01	7.658e-07	1.0
4.750e-01	8.595e-07	1.0
4.780e-01	9.646e-07	1.0
4.810e-01	1.083e-06	1.0
4.840e-01	1.215e-06	1.0
4.870e-01	1.364e-06	1.0
4.900e-01	1.530e-06	1.0
4.930e-01	1.717e-06	1.0
4.960e-01	1.928e-06	1.0
4.990e-01	2.163e-06	1.0
5.020e-01	2.428e-06	1.0
5.050e-01	2.725e-06	1.0
5.080e-01	3.058e-06	1.0
5.110e-01	3.432e-06	1.0
5.140e-01	3.852e-06	1.0
5.170e-01	4.323e-06	1.0
5.200e-01	4.852e-06	1.0
5.230e-01	5.445e-06	1.0
5.260e-01	6.111e-06	1.0
5.290e-01	6.858e-06	1.0
5.320e-01	7.697e-06	1.0
5.350e-01	8.639e-06	1.0
5.380e-01	9.695e-06	1.0
5.410e-01	1.088e-05	1.0
5.440e-01	1.221e-05	1.0
5.470e-01	1.371e-05	1.0
5.500e-01	1.538e-05	1.0
5.530e-01	1.726e-05	1.0
5.560e-01	1.937e-05	1.0
5.590e-01	2.174e-05	1.0
5.620e-01	2.440e-05	1.0
5.650e-01	2.739e-05	1.0

5.680e-01	3.074e-05	1.0
5.710e-01	3.450e-05	1.0
5.740e-01	3.872e-05	1.0
5.770e-01	4.345e-05	1.0
5.800e-01	4.876e-05	1.0
5.830e-01	5.473e-05	1.0
5.860e-01	6.142e-05	1.0
5.890e-01	6.894e-05	1.0
5.920e-01	7.737e-05	1.0
5.950e-01	8.683e-05	1.0
5.980e-01	9.745e-05	1.0
6.010e-01	1.094e-04	1.0
6.040e-01	1.227e-04	1.0
6.070e-01	1.378e-04	1.0
6.100e-01	1.546e-04	1.0
6.130e-01	1.735e-04	1.0
6.160e-01	1.947e-04	1.0
6.190e-01	2.185e-04	1.0
6.220e-01	2.453e-04	1.0
6.250e-01	2.753e-04	1.0
6.280e-01	3.089e-04	1.0
6.310e-01	3.467e-04	1.0
6.340e-01	3.891e-04	1.0
6.370e-01	4.367e-04	1.0
6.400e-01	4.901e-04	1.0
6.430e-01	5.501e-04	1.0
6.460e-01	6.174e-04	1.0
6.490e-01	6.929e-04	1.0
6.520e-01	7.776e-04	1.0
6.550e-01	8.727e-04	1.0
6.580e-01	9.795e-04	1.0
6.610e-01	1.099e-03	1.0
6.640e-01	1.234e-03	1.0
6.670e-01	1.385e-03	1.0
6.700e-01	1.554e-03	1.0
6.730e-01	1.744e-03	1.0
6.760e-01	1.957e-03	1.0
6.790e-01	2.197e-03	1.0
6.820e-01	2.465e-03	1.0
6.850e-01	2.767e-03	1.0
6.880e-01	3.105e-03	1.0
6.910e-01	3.485e-03	1.0
6.940e-01	3.911e-03	1.0
6.970e-01	4.390e-03	1.0
7.000e-01	4.927e-03	1.0

-1.000e+00	-1.000e+00	0.000e+00	1.100e+01
-6.000e-01	-1.000e+00	0.000e+00	2.000e+01
-2.000e-01	-1.000e+00	0.000e+00	2.000e+01
2.000e-01	-1.000e+00	0.000e+00	2.000e+01
6.000e-01	-1.000e+00	0.000e+00	2.000e+01
1.000e+00	-1.000e+00	0.000e+00	1.100e+01
1.400e+00	-1.000e+00	-4.500e+01	1.100e+01
1.800e+00	-1.000e+00	-6.500e+01	1.100e+01
2.200e+00	-1.000e+00	-8.500e+01	1.100e+01
2.600e+00	-1.000e+00	-1.050e+02	1.100e+01
3.000e+00	-1.000e+00	-1.250e+02	1.100e+01
3.400e+00	-1.000e+00	-1.450e+02	1.100e+01
3.800e+00	-1.000e+00	-1.650e+02	1.100e+01
4.200e+00	-1.000e+00	-1.850e+02	1.100e+01
4.600e+00	-1.000e+00	-2.050e+02	1.100e+01
5.000e+00	-1.000e+00	-2.250e+02	1.100e+01
-1.000e+00	-6.000e-01	0.000e+00	1.100e+01
-6.000e-01	-6.000e-01	0.000e+00	1.100e+01
-2.000e-01	-6.000e-01	0.000e+00	2.000e+01
2.000e-01	-6.000e-01	0.000e+00	2.000e+01
6.000e-01	-6.000e-01	0.000e+00	2.000e+01
1.000e+00	-6.000e-01	0.000e+00	2.000e+01
1.400e+00	-6.000e-01	-2.100e+01	1.100e+01
1.800e+00	-6.000e-01	-3.300e+01	1.100e+01
2.200e+00	-6.000e-01	-4.500e+01	1.100e+01
2.600e+00	-6.000e-01	-5.700e+01	1.100e+01
3.000e+00	-6.000e-01	-6.900e+01	1.100e+01
3.400e+00	-6.000e-01	-8.100e+01	1.100e+01
3.800e+00	-6.000e-01	-9.300e+01	1.100e+01
4.200e+00	-6.000e-01	-1.050e+02	1.100e+01
4.600e+00	-6.000e-01	-1.170e+02	1.100e+01
5.000e+00	-6.000e-01	-1.290e+02	1.100e+01
-1.000e+00	-2.000e-01	0.000e+00	1.100e+01
-6.000e-01	-2.000e-01	0.000e+00	1.100e+01
-2.000e-01	-2.000e-01	0.000e+00	1.100e+01
2.000e-01	-2.000e-01	0.000e+00	2.000e+01
6.000e-01	-2.000e-01	0.000e+00	2.000e+01
1.000e+00	-2.000e-01	0.000e+00	2.000e+01
1.400e+00	-2.000e-01	-5.000e+00	2.000e+01
1.800e+00	-2.000e-01	-9.000e+00	1.100e+01
2.200e+00	-2.000e-01	-1.300e+01	1.100e+01
2.600e+00	-2.000e-01	-1.700e+01	1.100e+01
3.000e+00	-2.000e-01	-2.100e+01	1.100e+01
3.400e+00	-2.000e-01	-2.500e+01	1.100e+01
3.800e+00	-2.000e-01	-2.900e+01	1.100e+01
4.200e+00	-2.000e-01	-3.300e+01	1.100e+01
4.600e+00	-2.000e-01	-3.700e+01	1.100e+01
5.000e+00	-2.000e-01	-4.100e+01	1.100e+01
-1.000e+00	2.000e-01	0.000e+00	1.100e+01
-6.000e-01	2.000e-01	0.000e+00	1.100e+01
-2.000e-01	2.000e-01	0.000e+00	1.100e+01
2.000e-01	2.000e-01	0.000e+00	1.100e+01
6.000e-01	2.000e-01	0.000e+00	2.000e+01
1.000e+00	2.000e-01	0.000e+00	2.000e+01
1.400e+00	2.000e-01	3.000e+00	2.000e+01
1.800e+00	2.000e-01	7.000e+00	2.000e+01

2.200e+00	2.000e-01	1.100e+01	1.100e+01
2.600e+00	2.000e-01	1.500e+01	1.100e+01
3.000e+00	2.000e-01	1.900e+01	1.100e+01
3.400e+00	2.000e-01	2.300e+01	1.100e+01
3.800e+00	2.000e-01	2.700e+01	1.100e+01
4.200e+00	2.000e-01	3.100e+01	1.100e+01
4.600e+00	2.000e-01	3.500e+01	1.100e+01
5.000e+00	2.000e-01	3.900e+01	1.100e+01
-1.000e+00	6.000e-01	0.000e+00	1.100e+01
-6.000e-01	6.000e-01	0.000e+00	1.100e+01
-2.000e-01	6.000e-01	0.000e+00	1.100e+01
2.000e-01	6.000e-01	0.000e+00	1.100e+01
6.000e-01	6.000e-01	0.000e+00	1.100e+01
1.000e+00	6.000e-01	0.000e+00	2.000e+01
1.400e+00	6.000e-01	4.016e+00	2.000e+01
1.800e+00	6.000e-01	1.500e+01	2.000e+01
2.200e+00	6.000e-01	2.700e+01	2.000e+01
2.600e+00	6.000e-01	3.900e+01	1.100e+01
3.000e+00	6.000e-01	5.100e+01	1.100e+01
3.400e+00	6.000e-01	6.300e+01	1.100e+01
3.800e+00	6.000e-01	7.500e+01	1.100e+01
4.200e+00	6.000e-01	8.700e+01	1.100e+01
4.600e+00	6.000e-01	9.900e+01	1.100e+01
5.000e+00	6.000e-01	1.110e+02	1.100e+01
-1.000e+00	1.000e+00	0.000e+00	1.000e+00
-6.000e-01	1.000e+00	0.000e+00	1.000e+00
-2.000e-01	1.000e+00	0.000e+00	1.000e+00
2.000e-01	1.000e+00	0.000e+00	1.000e+00
6.000e-01	1.000e+00	0.000e+00	1.000e+00
1.000e+00	1.000e+00	0.000e+00	1.000e+00
1.400e+00	1.000e+00	4.048e+00	1.000e+01
1.800e+00	1.000e+00	1.606e+01	1.000e+01
2.200e+00	1.000e+00	3.500e+01	1.000e+01
2.600e+00	1.000e+00	5.500e+01	1.000e+01
3.000e+00	1.000e+00	7.500e+01	1.000e+00
3.400e+00	1.000e+00	9.500e+01	1.000e+00
3.800e+00	1.000e+00	1.150e+02	1.000e+00
4.200e+00	1.000e+00	1.350e+02	1.000e+00
4.600e+00	1.000e+00	1.550e+02	1.000e+00
5.000e+00	1.000e+00	1.750e+02	1.000e+00
-1.000e+00	1.400e+00	0.000e+00	1.000e+00
-6.000e-01	1.400e+00	0.000e+00	1.000e+00
-2.000e-01	1.400e+00	0.000e+00	1.000e+00
2.000e-01	1.400e+00	0.000e+00	1.000e+00
6.000e-01	1.400e+00	0.000e+00	1.000e+00
1.000e+00	1.400e+00	0.000e+00	1.000e+00
1.400e+00	1.400e+00	4.080e+00	1.000e+00
1.800e+00	1.400e+00	1.619e+01	1.000e+01
2.200e+00	1.400e+00	3.614e+01	1.000e+01
2.600e+00	1.400e+00	6.300e+01	1.000e+01
3.000e+00	1.400e+00	9.100e+01	1.000e+01
3.400e+00	1.400e+00	1.190e+02	1.000e+00
3.800e+00	1.400e+00	1.470e+02	1.000e+00
4.200e+00	1.400e+00	1.750e+02	1.000e+00
4.600e+00	1.400e+00	2.030e+02	1.000e+00
5.000e+00	1.400e+00	2.310e+02	1.000e+00

-1.000e+00	1.800e+00	0.000e+00	1.000e+00
-6.000e-01	1.800e+00	0.000e+00	1.000e+00
-2.000e-01	1.800e+00	0.000e+00	1.000e+00
2.000e-01	1.800e+00	0.000e+00	1.000e+00
6.000e-01	1.800e+00	0.000e+00	1.000e+00
1.000e+00	1.800e+00	0.000e+00	1.000e+00
1.400e+00	1.800e+00	4.112e+00	1.000e+00
1.800e+00	1.800e+00	1.632e+01	1.000e+00
2.200e+00	1.800e+00	3.643e+01	1.000e+01
2.600e+00	1.800e+00	6.426e+01	1.000e+01
3.000e+00	1.800e+00	9.900e+01	1.000e+01
3.400e+00	1.800e+00	1.350e+02	1.000e+01
3.800e+00	1.800e+00	1.710e+02	1.000e+00
4.200e+00	1.800e+00	2.070e+02	1.000e+00
4.600e+00	1.800e+00	2.430e+02	1.000e+00
5.000e+00	1.800e+00	2.790e+02	1.000e+00
-1.000e+00	2.200e+00	0.000e+00	1.000e+00
-6.000e-01	2.200e+00	0.000e+00	1.000e+00
-2.000e-01	2.200e+00	0.000e+00	1.000e+00
2.000e-01	2.200e+00	0.000e+00	1.000e+00
6.000e-01	2.200e+00	0.000e+00	1.000e+00
1.000e+00	2.200e+00	0.000e+00	1.000e+00
1.400e+00	2.200e+00	4.144e+00	1.000e+00
1.800e+00	2.200e+00	1.645e+01	1.000e+00
2.200e+00	2.200e+00	3.672e+01	1.000e+00
2.600e+00	2.200e+00	6.477e+01	1.000e+01
3.000e+00	2.200e+00	1.004e+02	1.000e+01
3.400e+00	2.200e+00	1.430e+02	1.000e+01
3.800e+00	2.200e+00	1.870e+02	1.000e+01
4.200e+00	2.200e+00	2.310e+02	1.000e+00
4.600e+00	2.200e+00	2.750e+02	1.000e+00
5.000e+00	2.200e+00	3.190e+02	1.000e+00
-1.000e+00	2.600e+00	0.000e+00	1.000e+00
-6.000e-01	2.600e+00	0.000e+00	1.000e+00
-2.000e-01	2.600e+00	0.000e+00	1.000e+00
2.000e-01	2.600e+00	0.000e+00	1.000e+00
6.000e-01	2.600e+00	0.000e+00	1.000e+00
1.000e+00	2.600e+00	0.000e+00	1.000e+00
1.400e+00	2.600e+00	4.176e+00	1.000e+00
1.800e+00	2.600e+00	1.658e+01	1.000e+00
2.200e+00	2.600e+00	3.701e+01	1.000e+00
2.600e+00	2.600e+00	6.528e+01	1.000e+00
3.000e+00	2.600e+00	1.012e+02	1.000e+01
3.400e+00	2.600e+00	1.446e+02	1.000e+01
3.800e+00	2.600e+00	1.950e+02	1.000e+01
4.200e+00	2.600e+00	2.470e+02	1.000e+01
4.600e+00	2.600e+00	2.990e+02	1.000e+00
5.000e+00	2.600e+00	3.510e+02	1.000e+00
-1.000e+00	3.000e+00	0.000e+00	1.000e+00
-6.000e-01	3.000e+00	0.000e+00	1.000e+00
-2.000e-01	3.000e+00	0.000e+00	1.000e+00
2.000e-01	3.000e+00	0.000e+00	1.000e+00
6.000e-01	3.000e+00	0.000e+00	1.000e+00
1.000e+00	3.000e+00	0.000e+00	1.000e+00
1.400e+00	3.000e+00	4.208e+00	1.000e+00
1.800e+00	3.000e+00	1.670e+01	1.000e+00

2.200e+00	3.000e+00	3.730e+01	1.000e+00
2.600e+00	3.000e+00	6.579e+01	1.000e+00
3.000e+00	3.000e+00	1.020e+02	1.000e+00
3.400e+00	3.000e+00	1.457e+02	1.000e+01
3.800e+00	3.000e+00	1.968e+02	1.000e+01
4.200e+00	3.000e+00	2.550e+02	1.000e+01
4.600e+00	3.000e+00	3.150e+02	1.000e+01
5.000e+00	3.000e+00	3.750e+02	1.000e+00
-1.000e+00	3.400e+00	0.000e+00	1.000e+00
-6.000e-01	3.400e+00	0.000e+00	1.000e+00
-2.000e-01	3.400e+00	0.000e+00	1.000e+00
2.000e-01	3.400e+00	0.000e+00	1.000e+00
6.000e-01	3.400e+00	0.000e+00	1.000e+00
1.000e+00	3.400e+00	0.000e+00	1.000e+00
1.400e+00	3.400e+00	4.240e+00	1.000e+00
1.800e+00	3.400e+00	1.683e+01	1.000e+00
2.200e+00	3.400e+00	3.758e+01	1.000e+00
2.600e+00	3.400e+00	6.630e+01	1.000e+00
3.000e+00	3.400e+00	1.028e+02	1.000e+00
3.400e+00	3.400e+00	1.469e+02	1.000e+00
3.800e+00	3.400e+00	1.984e+02	1.000e+01
4.200e+00	3.400e+00	2.570e+02	1.000e+01
4.600e+00	3.400e+00	3.230e+02	1.000e+01
5.000e+00	3.400e+00	3.910e+02	1.000e+01
-1.000e+00	3.800e+00	0.000e+00	1.000e+00
-6.000e-01	3.800e+00	0.000e+00	1.000e+00
-2.000e-01	3.800e+00	0.000e+00	1.000e+00
2.000e-01	3.800e+00	0.000e+00	1.000e+00
6.000e-01	3.800e+00	0.000e+00	1.000e+00
1.000e+00	3.800e+00	0.000e+00	1.000e+00
1.400e+00	3.800e+00	4.272e+00	1.000e+00
1.800e+00	3.800e+00	1.696e+01	1.000e+00
2.200e+00	3.800e+00	3.787e+01	1.000e+00
2.600e+00	3.800e+00	6.682e+01	1.000e+00
3.000e+00	3.800e+00	1.036e+02	1.000e+00
3.400e+00	3.800e+00	1.480e+02	1.000e+00
3.800e+00	3.800e+00	1.999e+02	1.000e+00
4.200e+00	3.800e+00	2.591e+02	1.000e+01
4.600e+00	3.800e+00	3.253e+02	1.000e+01
5.000e+00	3.800e+00	3.990e+02	1.000e+01
-1.000e+00	4.200e+00	0.000e+00	1.000e+00
-6.000e-01	4.200e+00	0.000e+00	1.000e+00
-2.000e-01	4.200e+00	0.000e+00	1.000e+00
2.000e-01	4.200e+00	0.000e+00	1.000e+00
6.000e-01	4.200e+00	0.000e+00	1.000e+00
1.000e+00	4.200e+00	0.000e+00	1.000e+00
1.400e+00	4.200e+00	4.304e+00	1.000e+00
1.800e+00	4.200e+00	1.709e+01	1.000e+00
2.200e+00	4.200e+00	3.816e+01	1.000e+00
2.600e+00	4.200e+00	6.733e+01	1.000e+00
3.000e+00	4.200e+00	1.044e+02	1.000e+00
3.400e+00	4.200e+00	1.492e+02	1.000e+00
3.800e+00	4.200e+00	2.015e+02	1.000e+00
4.200e+00	4.200e+00	2.611e+02	1.000e+00
4.600e+00	4.200e+00	3.279e+02	1.000e+01
5.000e+00	4.200e+00	4.016e+02	1.000e+01

-1.000e+00	4.600e+00	0.000e+00	1.000e+00
-6.000e-01	4.600e+00	0.000e+00	1.000e+00
-2.000e-01	4.600e+00	0.000e+00	1.000e+00
2.000e-01	4.600e+00	0.000e+00	1.000e+00
6.000e-01	4.600e+00	0.000e+00	1.000e+00
1.000e+00	4.600e+00	0.000e+00	1.000e+00
1.400e+00	4.600e+00	4.336e+00	1.000e+00
1.800e+00	4.600e+00	1.722e+01	1.000e+00
2.200e+00	4.600e+00	3.845e+01	1.000e+00
2.600e+00	4.600e+00	6.784e+01	1.000e+00
3.000e+00	4.600e+00	1.052e+02	1.000e+00
3.400e+00	4.600e+00	1.503e+02	1.000e+00
3.800e+00	4.600e+00	2.031e+02	1.000e+00
4.200e+00	4.600e+00	2.632e+02	1.000e+00
4.600e+00	4.600e+00	3.305e+02	1.000e+00
5.000e+00	4.600e+00	4.048e+02	1.000e+01
-1.000e+00	5.000e+00	0.000e+00	1.000e+00
-6.000e-01	5.000e+00	0.000e+00	1.000e+00
-2.000e-01	5.000e+00	0.000e+00	1.000e+00
2.000e-01	5.000e+00	0.000e+00	1.000e+00
6.000e-01	5.000e+00	0.000e+00	1.000e+00
1.000e+00	5.000e+00	0.000e+00	1.000e+00
1.400e+00	5.000e+00	4.368e+00	1.000e+00
1.800e+00	5.000e+00	1.734e+01	1.000e+00
2.200e+00	5.000e+00	3.874e+01	1.000e+00
2.600e+00	5.000e+00	6.835e+01	1.000e+00
3.000e+00	5.000e+00	1.060e+02	1.000e+00
3.400e+00	5.000e+00	1.515e+02	1.000e+00
3.800e+00	5.000e+00	2.046e+02	1.000e+00
4.200e+00	5.000e+00	2.652e+02	1.000e+00
4.600e+00	5.000e+00	3.331e+02	1.000e+00
5.000e+00	5.000e+00	4.080e+02	1.000e+00

0.000e+00	0.000e+00	-7.900e+01	7.900e+01	1.00
0.000e+00	1.000e+00	-7.300e+01	7.300e+01	1.00
0.000e+00	2.000e+00	-6.700e+01	6.700e+01	1.00
0.000e+00	3.000e+00	-6.100e+01	6.100e+01	1.00
0.000e+00	4.000e+00	-5.500e+01	5.500e+01	1.00
0.000e+00	5.000e+00	-4.900e+01	4.900e+01	1.00
0.000e+00	6.000e+00	-4.300e+01	4.300e+01	1.00
0.000e+00	7.000e+00	-3.700e+01	3.700e+01	1.00
0.000e+00	8.000e+00	-3.100e+01	3.100e+01	1.00
0.000e+00	9.000e+00	-2.500e+01	2.500e+01	1.00
0.000e+00	1.000e+01	-1.900e+01	1.900e+01	1.00
0.000e+00	1.100e+01	-1.300e+01	1.300e+01	1.00
0.000e+00	1.200e+01	-7.000e+00	7.000e+00	1.00
0.000e+00	1.300e+01	-1.000e+00	1.000e+00	1.00
0.000e+00	1.400e+01	5.000e+00	-5.000e+00	1.00
0.000e+00	1.500e+01	1.100e+01	-1.100e+01	1.00
1.000e+00	0.000e+00	-6.800e+01	6.800e+01	1.00
1.000e+00	1.000e+00	-7.000e+01	7.000e+01	1.00
1.000e+00	2.000e+00	-6.400e+01	6.400e+01	1.00
1.000e+00	3.000e+00	-5.800e+01	5.800e+01	1.00
1.000e+00	4.000e+00	-5.200e+01	5.200e+01	1.00
1.000e+00	5.000e+00	-4.600e+01	4.600e+01	1.00
1.000e+00	6.000e+00	-4.000e+01	4.000e+01	1.00
1.000e+00	7.000e+00	-3.400e+01	3.400e+01	1.00
1.000e+00	8.000e+00	-2.800e+01	2.800e+01	1.00
1.000e+00	9.000e+00	-2.200e+01	2.200e+01	1.00
1.000e+00	1.000e+01	-1.600e+01	1.600e+01	1.00
1.000e+00	1.100e+01	-1.000e+01	1.000e+01	1.00
1.000e+00	1.200e+01	-4.000e+00	4.000e+00	1.00
1.000e+00	1.300e+01	2.000e+00	-2.000e+00	1.00
1.000e+00	1.400e+01	8.000e+00	-8.000e+00	1.00
1.000e+00	1.500e+01	1.400e+01	-1.400e+01	1.00
2.000e+00	0.000e+00	-5.700e+01	5.700e+01	1.00
2.000e+00	1.000e+00	-5.900e+01	5.900e+01	1.00
2.000e+00	2.000e+00	-6.100e+01	6.100e+01	1.00
2.000e+00	3.000e+00	-5.500e+01	5.500e+01	1.00
2.000e+00	4.000e+00	-4.900e+01	4.900e+01	1.00
2.000e+00	5.000e+00	-4.300e+01	4.300e+01	1.00
2.000e+00	6.000e+00	-3.700e+01	3.700e+01	1.00
2.000e+00	7.000e+00	-3.100e+01	3.100e+01	1.00
2.000e+00	8.000e+00	-2.500e+01	2.500e+01	1.00
2.000e+00	9.000e+00	-1.900e+01	1.900e+01	1.00
2.000e+00	1.000e+01	-1.300e+01	1.300e+01	1.00
2.000e+00	1.100e+01	-7.000e+00	7.000e+00	1.00
2.000e+00	1.200e+01	-1.000e+00	1.000e+00	1.00
2.000e+00	1.300e+01	5.000e+00	-5.000e+00	1.00
2.000e+00	1.400e+01	1.100e+01	-1.100e+01	1.00
2.000e+00	1.500e+01	7.000e+00	-7.000e+00	1.00
3.000e+00	0.000e+00	-4.600e+01	4.600e+01	1.00
3.000e+00	1.000e+00	-4.800e+01	4.800e+01	1.00
3.000e+00	2.000e+00	-5.000e+01	5.000e+01	1.00
3.000e+00	3.000e+00	-5.200e+01	5.200e+01	1.00
3.000e+00	4.000e+00	-4.600e+01	4.600e+01	1.00
3.000e+00	5.000e+00	-4.000e+01	4.000e+01	1.00
3.000e+00	6.000e+00	-3.400e+01	3.400e+01	1.00
3.000e+00	7.000e+00	-2.800e+01	2.800e+01	1.00

3.000e+00	8.000e+00	-2.200e+01	2.200e+01	1.00
3.000e+00	9.000e+00	-1.600e+01	1.600e+01	1.00
3.000e+00	1.000e+01	-1.000e+01	1.000e+01	1.00
3.000e+00	1.100e+01	-4.000e+00	4.000e+00	1.00
3.000e+00	1.200e+01	2.000e+00	-2.000e+00	1.00
3.000e+00	1.300e+01	8.000e+00	-8.000e+00	1.00
3.000e+00	1.400e+01	4.000e+00	-4.000e+00	1.00
3.000e+00	1.500e+01	0.000e+00	0.000e+00	1.00
4.000e+00	0.000e+00	-3.500e+01	3.500e+01	1.00
4.000e+00	1.000e+00	-3.700e+01	3.700e+01	1.00
4.000e+00	2.000e+00	-3.900e+01	3.900e+01	1.00
4.000e+00	3.000e+00	-4.100e+01	4.100e+01	1.00
4.000e+00	4.000e+00	-4.300e+01	4.300e+01	1.00
4.000e+00	5.000e+00	-3.700e+01	3.700e+01	1.00
4.000e+00	6.000e+00	-3.100e+01	3.100e+01	1.00
4.000e+00	7.000e+00	-2.500e+01	2.500e+01	1.00
4.000e+00	8.000e+00	-1.900e+01	1.900e+01	1.00
4.000e+00	9.000e+00	-1.300e+01	1.300e+01	1.00
4.000e+00	1.000e+01	-7.000e+00	7.000e+00	1.00
4.000e+00	1.100e+01	-1.000e+00	1.000e+00	1.00
4.000e+00	1.200e+01	5.000e+00	-5.000e+00	1.00
4.000e+00	1.300e+01	1.000e+00	-1.000e+00	1.00
4.000e+00	1.400e+01	-3.000e+00	3.000e+00	1.00
4.000e+00	1.500e+01	-7.000e+00	7.000e+00	1.00
5.000e+00	0.000e+00	-2.400e+01	2.400e+01	1.00
5.000e+00	1.000e+00	-2.600e+01	2.600e+01	1.00
5.000e+00	2.000e+00	-2.800e+01	2.800e+01	1.00
5.000e+00	3.000e+00	-3.000e+01	3.000e+01	1.00
5.000e+00	4.000e+00	-3.200e+01	3.200e+01	1.00
5.000e+00	5.000e+00	-3.400e+01	3.400e+01	1.00
5.000e+00	6.000e+00	-2.800e+01	2.800e+01	1.00
5.000e+00	7.000e+00	-2.200e+01	2.200e+01	1.00
5.000e+00	8.000e+00	-1.600e+01	1.600e+01	1.00
5.000e+00	9.000e+00	-1.000e+01	1.000e+01	1.00
5.000e+00	1.000e+01	-4.000e+00	4.000e+00	1.00
5.000e+00	1.100e+01	2.000e+00	-2.000e+00	1.00
5.000e+00	1.200e+01	-2.000e+00	2.000e+00	1.00
5.000e+00	1.300e+01	-6.000e+00	6.000e+00	1.00
5.000e+00	1.400e+01	-1.000e+01	1.000e+01	1.00
5.000e+00	1.500e+01	-1.400e+01	1.400e+01	1.00
6.000e+00	0.000e+00	-1.300e+01	1.300e+01	1.00
6.000e+00	1.000e+00	-1.500e+01	1.500e+01	1.00
6.000e+00	2.000e+00	-1.700e+01	1.700e+01	1.00
6.000e+00	3.000e+00	-1.900e+01	1.900e+01	1.00
6.000e+00	4.000e+00	-2.100e+01	2.100e+01	1.00
6.000e+00	5.000e+00	-2.300e+01	2.300e+01	1.00
6.000e+00	6.000e+00	-2.500e+01	2.500e+01	1.00
6.000e+00	7.000e+00	-1.900e+01	1.900e+01	1.00
6.000e+00	8.000e+00	-1.300e+01	1.300e+01	1.00
6.000e+00	9.000e+00	-7.000e+00	7.000e+00	1.00
6.000e+00	1.000e+01	-1.000e+00	1.000e+00	1.00
6.000e+00	1.100e+01	-5.000e+00	5.000e+00	1.00
6.000e+00	1.200e+01	-9.000e+00	9.000e+00	1.00
6.000e+00	1.300e+01	-1.300e+01	1.300e+01	1.00
6.000e+00	1.400e+01	-1.700e+01	1.700e+01	1.00
6.000e+00	1.500e+01	-2.100e+01	2.100e+01	1.00

7.000e+00	0.000e+00	-2.000e+00	2.000e+00	1.00
7.000e+00	1.000e+00	-4.000e+00	4.000e+00	1.00
7.000e+00	2.000e+00	-6.000e+00	6.000e+00	1.00
7.000e+00	3.000e+00	-8.000e+00	8.000e+00	1.00
7.000e+00	4.000e+00	-1.000e+01	1.000e+01	1.00
7.000e+00	5.000e+00	-1.200e+01	1.200e+01	1.00
7.000e+00	6.000e+00	-1.400e+01	1.400e+01	1.00
7.000e+00	7.000e+00	-1.600e+01	1.600e+01	1.00
7.000e+00	8.000e+00	-1.000e+01	1.000e+01	1.00
7.000e+00	9.000e+00	-4.000e+00	4.000e+00	1.00
7.000e+00	1.000e+01	-8.000e+00	8.000e+00	1.00
7.000e+00	1.100e+01	-1.200e+01	1.200e+01	1.00
7.000e+00	1.200e+01	-1.600e+01	1.600e+01	1.00
7.000e+00	1.300e+01	-2.000e+01	2.000e+01	1.00
7.000e+00	1.400e+01	-2.400e+01	2.400e+01	1.00
7.000e+00	1.500e+01	-2.800e+01	2.800e+01	1.00
8.000e+00	0.000e+00	9.000e+00	-9.000e+00	1.00
8.000e+00	1.000e+00	7.000e+00	-7.000e+00	1.00
8.000e+00	2.000e+00	5.000e+00	-5.000e+00	1.00
8.000e+00	3.000e+00	3.000e+00	-3.000e+00	1.00
8.000e+00	4.000e+00	1.000e+00	-1.000e+00	1.00
8.000e+00	5.000e+00	-1.000e+00	1.000e+00	1.00
8.000e+00	6.000e+00	-3.000e+00	3.000e+00	1.00
8.000e+00	7.000e+00	-5.000e+00	5.000e+00	1.00
8.000e+00	8.000e+00	-7.000e+00	7.000e+00	1.00
8.000e+00	9.000e+00	-1.100e+01	1.100e+01	1.00
8.000e+00	1.000e+01	-1.500e+01	1.500e+01	1.00
8.000e+00	1.100e+01	-1.900e+01	1.900e+01	1.00
8.000e+00	1.200e+01	-2.300e+01	2.300e+01	1.00
8.000e+00	1.300e+01	-2.700e+01	2.700e+01	1.00
8.000e+00	1.400e+01	-3.100e+01	3.100e+01	1.00
8.000e+00	1.500e+01	-3.500e+01	3.500e+01	1.00
9.000e+00	0.000e+00	2.000e+01	-2.000e+01	1.00
9.000e+00	1.000e+00	1.800e+01	-1.800e+01	1.00
9.000e+00	2.000e+00	1.600e+01	-1.600e+01	1.00
9.000e+00	3.000e+00	1.400e+01	-1.400e+01	1.00
9.000e+00	4.000e+00	1.200e+01	-1.200e+01	1.00
9.000e+00	5.000e+00	1.000e+01	-1.000e+01	1.00
9.000e+00	6.000e+00	8.000e+00	-8.000e+00	1.00
9.000e+00	7.000e+00	6.000e+00	-6.000e+00	1.00
9.000e+00	8.000e+00	-6.000e+00	6.000e+00	1.00
9.000e+00	9.000e+00	-1.800e+01	1.800e+01	1.00
9.000e+00	1.000e+01	-2.200e+01	2.200e+01	1.00
9.000e+00	1.100e+01	-2.600e+01	2.600e+01	1.00
9.000e+00	1.200e+01	-3.000e+01	3.000e+01	1.00
9.000e+00	1.300e+01	-3.400e+01	3.400e+01	1.00
9.000e+00	1.400e+01	-3.800e+01	3.800e+01	1.00
9.000e+00	1.500e+01	-4.200e+01	4.200e+01	1.00
1.000e+01	0.000e+00	3.100e+01	-3.100e+01	1.00
1.000e+01	1.000e+00	2.900e+01	-2.900e+01	1.00
1.000e+01	2.000e+00	2.700e+01	-2.700e+01	1.00
1.000e+01	3.000e+00	2.500e+01	-2.500e+01	1.00
1.000e+01	4.000e+00	2.300e+01	-2.300e+01	1.00
1.000e+01	5.000e+00	2.100e+01	-2.100e+01	1.00
1.000e+01	6.000e+00	1.900e+01	-1.900e+01	1.00
1.000e+01	7.000e+00	7.000e+00	-7.000e+00	1.00

1.000e+01	8.000e+00	-5.000e+00	5.000e+00	1.00
1.000e+01	9.000e+00	-1.700e+01	1.700e+01	1.00
1.000e+01	1.000e+01	-2.900e+01	2.900e+01	1.00
1.000e+01	1.100e+01	-3.300e+01	3.300e+01	1.00
1.000e+01	1.200e+01	-3.700e+01	3.700e+01	1.00
1.000e+01	1.300e+01	-4.100e+01	4.100e+01	1.00
1.000e+01	1.400e+01	-4.500e+01	4.500e+01	1.00
1.000e+01	1.500e+01	-4.900e+01	4.900e+01	1.00
1.100e+01	0.000e+00	4.200e+01	-4.200e+01	1.00
1.100e+01	1.000e+00	4.000e+01	-4.000e+01	1.00
1.100e+01	2.000e+00	3.800e+01	-3.800e+01	1.00
1.100e+01	3.000e+00	3.600e+01	-3.600e+01	1.00
1.100e+01	4.000e+00	3.400e+01	-3.400e+01	1.00
1.100e+01	5.000e+00	3.200e+01	-3.200e+01	1.00
1.100e+01	6.000e+00	2.000e+01	-2.000e+01	1.00
1.100e+01	7.000e+00	8.000e+00	-8.000e+00	1.00
1.100e+01	8.000e+00	-4.000e+00	4.000e+00	1.00
1.100e+01	9.000e+00	-1.600e+01	1.600e+01	1.00
1.100e+01	1.000e+01	-2.800e+01	2.800e+01	1.00
1.100e+01	1.100e+01	-4.000e+01	4.000e+01	1.00
1.100e+01	1.200e+01	-4.400e+01	4.400e+01	1.00
1.100e+01	1.300e+01	-4.800e+01	4.800e+01	1.00
1.100e+01	1.400e+01	-5.200e+01	5.200e+01	1.00
1.100e+01	1.500e+01	-5.600e+01	5.600e+01	1.00
1.200e+01	0.000e+00	5.300e+01	-5.300e+01	1.00
1.200e+01	1.000e+00	5.100e+01	-5.100e+01	1.00
1.200e+01	2.000e+00	4.900e+01	-4.900e+01	1.00
1.200e+01	3.000e+00	4.700e+01	-4.700e+01	1.00
1.200e+01	4.000e+00	4.500e+01	-4.500e+01	1.00
1.200e+01	5.000e+00	3.300e+01	-3.300e+01	1.00
1.200e+01	6.000e+00	2.100e+01	-2.100e+01	1.00
1.200e+01	7.000e+00	9.000e+00	-9.000e+00	1.00
1.200e+01	8.000e+00	-3.000e+00	3.000e+00	1.00
1.200e+01	9.000e+00	-1.500e+01	1.500e+01	1.00
1.200e+01	1.000e+01	-2.700e+01	2.700e+01	1.00
1.200e+01	1.100e+01	-3.900e+01	3.900e+01	1.00
1.200e+01	1.200e+01	-5.100e+01	5.100e+01	1.00
1.200e+01	1.300e+01	-5.500e+01	5.500e+01	1.00
1.200e+01	1.400e+01	-5.900e+01	5.900e+01	1.00
1.200e+01	1.500e+01	-6.300e+01	6.300e+01	1.00
1.300e+01	0.000e+00	6.400e+01	-6.400e+01	1.00
1.300e+01	1.000e+00	6.200e+01	-6.200e+01	1.00
1.300e+01	2.000e+00	6.000e+01	-6.000e+01	1.00
1.300e+01	3.000e+00	5.800e+01	-5.800e+01	1.00
1.300e+01	4.000e+00	4.600e+01	-4.600e+01	1.00
1.300e+01	5.000e+00	3.400e+01	-3.400e+01	1.00
1.300e+01	6.000e+00	2.200e+01	-2.200e+01	1.00
1.300e+01	7.000e+00	1.000e+01	-1.000e+01	1.00
1.300e+01	8.000e+00	-2.000e+00	2.000e+00	1.00
1.300e+01	9.000e+00	-1.400e+01	1.400e+01	1.00
1.300e+01	1.000e+01	-2.600e+01	2.600e+01	1.00
1.300e+01	1.100e+01	-3.800e+01	3.800e+01	1.00
1.300e+01	1.200e+01	-5.000e+01	5.000e+01	1.00
1.300e+01	1.300e+01	-6.200e+01	6.200e+01	1.00
1.300e+01	1.400e+01	-6.600e+01	6.600e+01	1.00
1.300e+01	1.500e+01	-7.000e+01	7.000e+01	1.00

1.400e+01	0.000e+00	7.500e+01	-7.500e+01	1.00
1.400e+01	1.000e+00	7.300e+01	-7.300e+01	1.00
1.400e+01	2.000e+00	7.100e+01	-7.100e+01	1.00
1.400e+01	3.000e+00	5.900e+01	-5.900e+01	1.00
1.400e+01	4.000e+00	4.700e+01	-4.700e+01	1.00
1.400e+01	5.000e+00	3.500e+01	-3.500e+01	1.00
1.400e+01	6.000e+00	2.300e+01	-2.300e+01	1.00
1.400e+01	7.000e+00	1.100e+01	-1.100e+01	1.00
1.400e+01	8.000e+00	-1.000e+00	1.000e+00	1.00
1.400e+01	9.000e+00	-1.300e+01	1.300e+01	1.00
1.400e+01	1.000e+01	-2.500e+01	2.500e+01	1.00
1.400e+01	1.100e+01	-3.700e+01	3.700e+01	1.00
1.400e+01	1.200e+01	-4.900e+01	4.900e+01	1.00
1.400e+01	1.300e+01	-6.100e+01	6.100e+01	1.00
1.400e+01	1.400e+01	-7.300e+01	7.300e+01	1.00
1.400e+01	1.500e+01	-7.700e+01	7.700e+01	1.00
1.500e+01	0.000e+00	8.600e+01	-8.600e+01	1.00
1.500e+01	1.000e+00	8.400e+01	-8.400e+01	1.00
1.500e+01	2.000e+00	7.200e+01	-7.200e+01	1.00
1.500e+01	3.000e+00	6.000e+01	-6.000e+01	1.00
1.500e+01	4.000e+00	4.800e+01	-4.800e+01	1.00
1.500e+01	5.000e+00	3.600e+01	-3.600e+01	1.00
1.500e+01	6.000e+00	2.400e+01	-2.400e+01	1.00
1.500e+01	7.000e+00	1.200e+01	-1.200e+01	1.00
1.500e+01	8.000e+00	0.000e+00	0.000e+00	1.00
1.500e+01	9.000e+00	-1.200e+01	1.200e+01	1.00
1.500e+01	1.000e+01	-2.400e+01	2.400e+01	1.00
1.500e+01	1.100e+01	-3.600e+01	3.600e+01	1.00
1.500e+01	1.200e+01	-4.800e+01	4.800e+01	1.00
1.500e+01	1.300e+01	-6.000e+01	6.000e+01	1.00
1.500e+01	1.400e+01	-7.200e+01	7.200e+01	1.00
1.500e+01	1.500e+01	-8.400e+01	8.400e+01	1.00

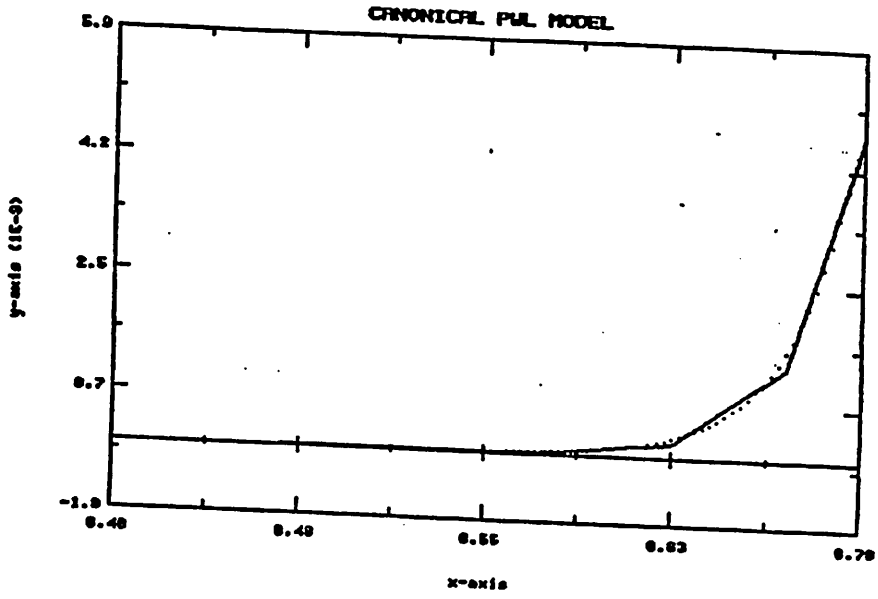

CANONICAL PWL MODEL
5.0
4.2
2.5
0.7
-1.0
y-axis (1E-0)
0.48
0.49
0.55
0.63
0.70
x-axis

Fig.1

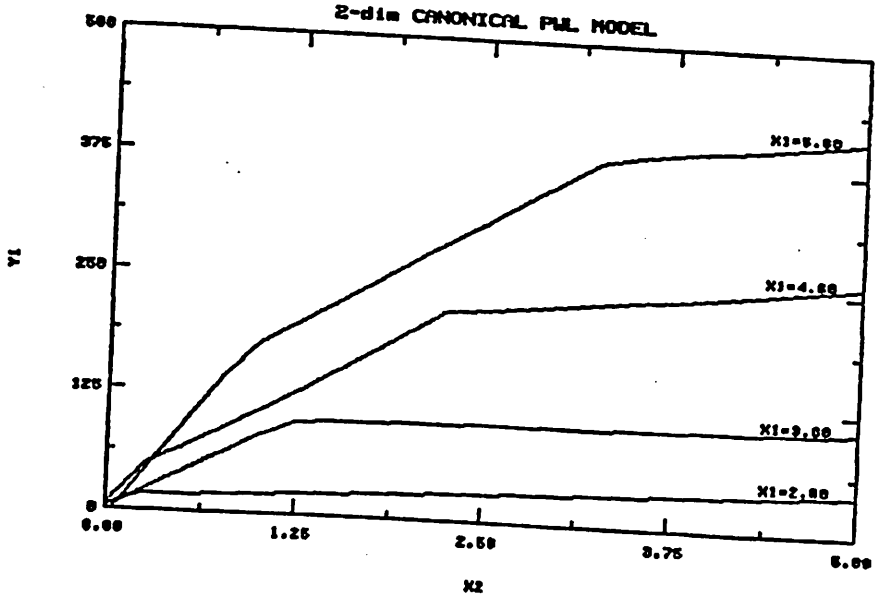
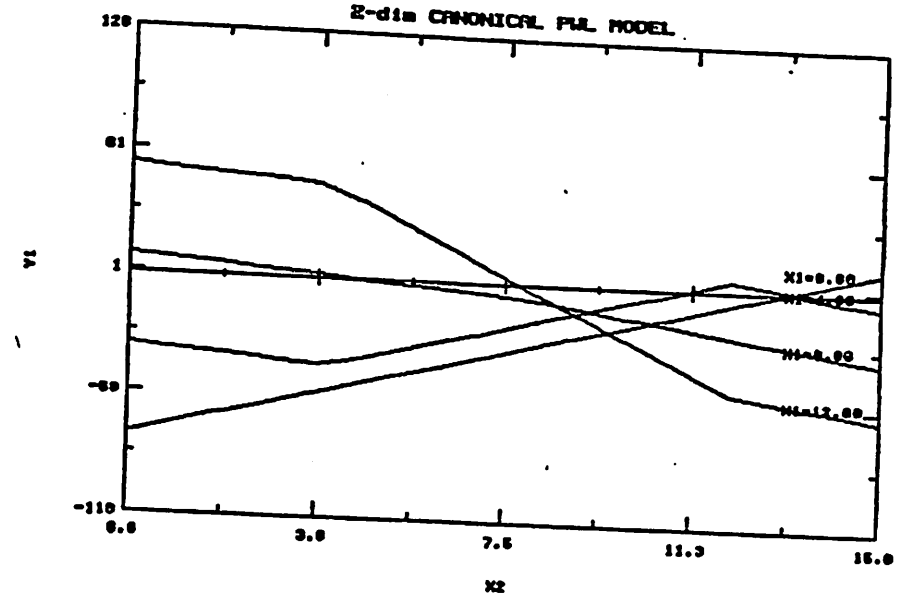
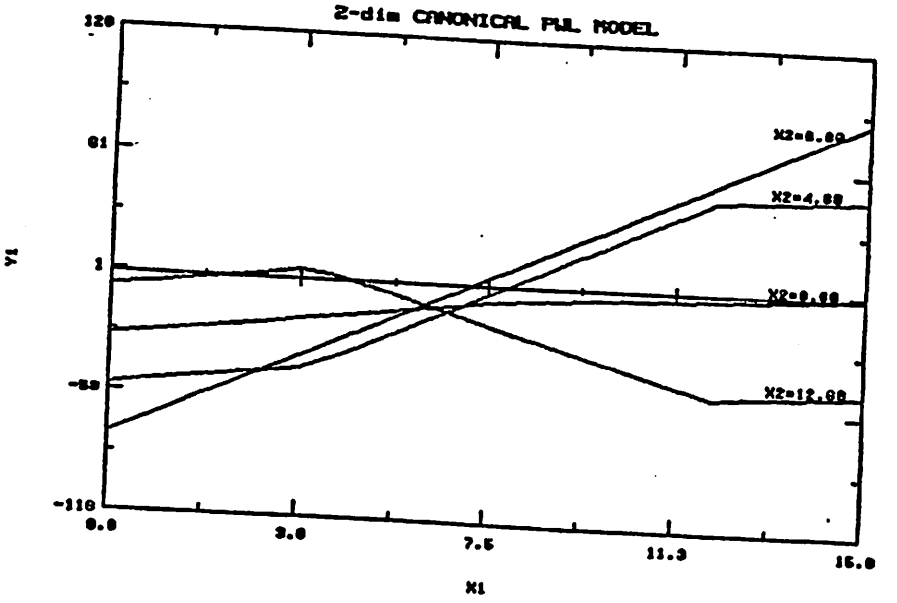

2-dim CANONICAL PWL MODEL
500
375
250
125
0
Y1
X1=5.00
X1=4.00
X1=3.00
X1=2.00
0.00
1.25
2.50
3.75
5.00
X2

Fig.2


2-dim CANONICAL PWL MODEL
120
61
1
-59
-119
0.0
3.8
7.5
11.3
15.0
Y1
X2
X1=0.00
X1=4.00
X1=8.00
X1=12.00

(a)


2-dim CANONICAL PWL MODEL
120
61
1
-59
-119
0.0
3.8
7.5
11.3
15.0
Y1
X1
X2=0.00
X2=4.00
X2=8.00
X2=12.00

(b)

Fig, 3

APPENDIX : SOURCE CODE LISTINGS

1. **1-dimensional canonical piecewise-linear modeling**
   pwln1.c, pwln2.c, pwln3.c, readt.c, aclib.c, pwlngf.c

2. **2-dimensional canonical piecewise-linear modeling**
   mod1.c, mod2.c, mod3.c, readt.c, aclib.c, modgf.c

```
double LABEL_SHIFT=0.5,NAME_SHIFT=1.5;
int N_LBLS=5,N_TICKS=2,TICK_LENGTH=5,SIG_FIGS=3,LABEL_SIDE=1;
int SIZE=1,BOX=1,LABELS=1,TICKS=1,AXES=1;
int TI=1;
double OFFSET=0.5;
```

```
#include <stdio.h>

int m,n;
double *x1,*y1,*r1,*b1,*p,error,*w,lb,ub;

/*****************************************************************************/
/* Canonical piecewise-linear modeling : 1-dimensional function.             */
/*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    FILE *fp,*gp,*fopen();
    char line[30];

    sprintf(line,"%s.dat",*++argv);
    if ((fp=fopen(line,"r")) == NULL)
    {
        printf("CAN'T OPEN THE DATA FILE %s\n",*argv);
        exit();
    }
    pwl_model(fp);
    sprintf(line,"%s.c",*argv);
    print_pwl(line);
    fclose(fp);
}

/*****************************************************************************/
/* Follow the algorithm in Section 2 of the paper "Canonical Piecewise-      */
/* Linear Modeling" to find the optimal canonical piecewise-linear model     */
/* for 2-terminal element which matches the measured data points with        */
/* minimal approximation error.                                              */
/*****************************************************************************/

pwl_model(fp)
FILE *fp;
{
    char ch[2];
    int i,j=0;

    /* get the data points and the initial partition on breakpoints */
    get_data(fp);

    /* find the optimal canonical pwl model w.r.t. */
    /* the initial partition                       */
    if (opt_p(p,&error)==-1)    /* breakpoint(s) out of data region */
        exit_message("IMPROPER BOUNDARIES");

    /* adjust the partition on breakpoints to */
    /* get the optimal canonical pwl model    */
    ch[0]='y';
    while (ch[0]=='y')
    {
        /* try to get out of the control of a local minimum */
        if (j>=2)
```

```
        {
            printf("reaching a local minimum\n");
            printf("please enter a new set of breakpoints\n");
            for (i=0;i<n;i++)
                scanf("%lf",p+i);
            j=0;
            if (opt_p(p,&error) == -1)
                exit_message("IMPROPER BOUNDARIES");
        }

        /* perform a line search and adjust the position of breakpoint(s) */
        if (l_search()==-1)
            j++;

        /* print the canonical pwl model */
        printf("\n*** partition boundaries ***\n");
        for (i=0;i<n;i++)
            printf("x = %.3e\n",p[i]);
        printf("a=%.3e\tb=%.3e\n",b1[0],b1[1]);
        for (i=0;i<n;i++)
            printf("c%d=%.3e\n",i+1,b1[i+2]);
        printf("continue? y/n\n");
        scanf("%1s",ch);
    }
}

/*****************************************************************************/
/* Get the measured data and the initial breakpoint(s) from an input file. */
/*****************************************************************************/

get_data(fp)
FILE *fp;
{
    char *calloc();
    int i;

    /* read the measured data points */
    read_data(fp);

    printf("enter the number of breakpoints\n");
    scanf("%d",&n);

    /* dynamic allocation for the variables */
    v_alloc();

    printf("enter the initial set of breakpoints\n");
    for (i=0;i<n;i++)
        scanf("%lf",p+i);
}

/*****************************************************************************/
/* Dynamic allocation for the variables.                                     */
/*****************************************************************************/

v_alloc()
{
```

```
    char *calloc();

    r1=(double *)calloc(m,sizeof(double));
    b1=(double *)calloc(n+2,sizeof(double));
    p=(double *)calloc(n,sizeof(double));
}

/*****************************************************************************/
/* Perform the line search to adjust the position of breakpoint(s) for       */
/* reducing the error between the measured data points and the canonical     */
/* pwl model.                                                                */
/*****************************************************************************/

l_search()
{
    char *calloc();
    int i,*ipvt,red=0,j=0;
    double t=1,*f,*g,*zq,rcond,max=0,fabs(),ratio=1;

    f=(double *)calloc(n*n,sizeof(double));
    g=(double *)calloc(n,sizeof(double));
    ipvt=(int *)calloc(n,sizeof(int));
    zq=(double *)calloc(n,sizeof(double));

    /* construct the matrix Y of Eq.(2.20) and the vector g of Eq.(2.15) */
    get_g_F(p,b1,r1,f,g);

    /* g is in the most descent direction */
    for (i=0;i<n;i++)
        g[i] = -g[i];

    /* find the line search direction inv(Y)*g */
    sgeco(f,n,ipvt,&rcond,zq);
    sgesl(f,n,ipvt,g,0);

    /* max = norm-1 of the vector g */
    for (i=0;i<n;i++)
        if (fabs(g[i])>max)
            max=fabs(g[i]);

    /* adjust the breakpoint position until a minimum is reached */
    /* along the line search direction                           */
    while (t>0.1 && fabs(ratio))>1e-3)
    {
        j++;
        if (new_p(&red,&t,g,max+1,&ratio)==-1)
            break;
    }

    cfree(f);
    cfree(g);
    cfree(ipvt);
    cfree(zq);
    if (j<3)
        return(-1);
    else
```

```
        return(0);
}

/*************************************************************************/
/* get a new position of breakpoint(s).                                  */
/*************************************************************************/

new_p(red,t,s,max,ratio)
int *red;
double *s,*t,max,*ratio;
{
    char *calloc();
    int i;
    double err,*px;

    px=(double *)calloc(n,sizeof(double));
    for (i=0;i<n;i++)
    {
        px[i]=p[i];
        p[i] += *t*s[i]/max;
    }
    if (opt_p(p,&err)==-1)
        *ratio=1;
    else
        *ratio=(err-error)/error;
    printf("approximation error = %.3e\n",err);

    /* approximation error is increasing */
    if (*ratio>0)
    {
        if (*ratio<0.01)
        {
            cfree(px);
            return(-1);
        }
        *t = *t/4;      /* reduce the increment size in line search */
        *red=1;
        for (i=0;i<n;i++)
            p[i]=px[i];
    }

    /* approximation error is decreasing */
    else
    {
        if (*ratio)-0.1 && *red==0)     /* double the increment size */
            *t=2*(*t);
        error=err;
    }
    cfree(px);
    return(0);
}

/*************************************************************************/
/* When get stuck in a local minimum, adjust the breakpoint position along */
/* a line direction until the approximation error begins to drop and may   */
/* possibly get out the control region of the previous local minimum.      */
```

```
/*****************************************************************************/

getout()
{
    char *calloc();
    int i,k=0,t=0;
    double ratio,*px,*ss,err,errx;

    px=(double *)calloc(n,sizeof(double));
    ss=(double *)calloc(n,sizeof(double));

    /* line direction */
    for (i=0;i<n;i++)
        ss[i]=(ub-p[i])/25;

    while (t++<=20)
    {
        if (t==20)
        {
            if (k==n)
                k=0;
            ss[k]=(lb-p[k])/25;
            k++;
            t=1;
        }
        for (i=0;i<n;i++)
            px[i]=p[i]+t*ss[i];
        printf("px[0]=%.3e\tpx[1]=%.3e\tpx[2]=%.3e\n",px[0],px[1],px[2]);
        opt_p(px,&err);
        if (t>1)
            ratio=(errx-err)/err;
        else
            ratio=(error-err)/error;
        printf("t=%d\terr=%.3e\tratio=%.3e\n",t,err,ratio);
        if (ratio>0.05)
        {
            error=err;
            for (i=0;i<n;i++)
                p[i]=px[i];
            break;
        }
        errx=err;
    }
    cfree(px);
    cfree(ss);
}
```

```
#include <stdio.h>

extern int n,m;
extern double *x1,*y1,*r1,*b1,*w;
extern double lb,ub;

/*****************************************************************************/
/* Find the parameters for the optimal canonical piecewise-linear model      */
/* w.r.t. a given set of breakpoints.                                        */
/*****************************************************************************/

opt_p(p,err)
double *p,*err;
{
    char *calloc();
    int *ipvt,i;
    double *a,*zq,rcond,get_err();

    /* check whether the breakpoint is out of the data region */
    for (i=0;i<n;i++)
        if (p[i]>=ub || p[i]<=lb)
        {
            printf("the breakpoint %.3e is out of bound; enter a new one\n",
                    p[i]);
            scanf("%lf",p+i);
        }

    ipvt=(int *)calloc(n+2,sizeof(int));
    zq=(double *)calloc(n+2,sizeof(double));
    a=(double *)calloc((n+2)*(n+2),sizeof(double));

    /* construct the matrix B=A*W*A', where A and W are */
    /* defined in Eqs.(2.6) and (2.7) respectively      */
    get_A(p,a);

    /* construct the vector b=A*W*y, where y and W are */
    /* defined in Eqs.(2.11) and (2.7) respectively    */
    get_b(p,b1,y1);

    /* find z=inv(B)*b for optimal parameters */
    sgeco(a,n+2,ipvt,&rcond,zq);
    if (rcond<1.0e-10)
    {
        printf("\nSINGULAR MATRIX\n");
        for (i=0;i<n;i++)
            printf("p[%d]=%.3e\t",i,p[i]);
        printf("\n");
        return(-1);
    }
    sgesl(a,n+2,ipvt,b1,0);

    /* estimate the approximation error */
    *err=get_err(p,b1,y1,r1);

    cfree(a);
    cfree(ipvt);
```

```
    cfree(zq);
    return(0);
}

/*****************************************************************************/
/* Construct the matrix B=A*W*A', where A is defined in Eq.(2.6) and W is  */
/* the weighting factor specified by the user.                               */
/*****************************************************************************/

get_A(p,a)
double *p,*a;
{
    int i,j,k;
    double fabs();

    for (k=0;k<m;k++)
    {
        a[0]+=1/w[k];
        a[1]+=x1[k]/w[k];
    }
    for (i=2;i<n+2;i++)
        for (k=0;k<m;k++)
            a[i]+=fabs(x1[k]-p[i-2])/w[k];
    for (k=0;k<m;k++)
        a[n+3]+=x1[k]*x1[k]/w[k];
    for (i=n+4;i<2*n+4;i++)
        for (k=0;k<m;k++)
            a[i]+=x1[k]*fabs(x1[k]-p[i-n-4])/w[k];
    for (i=2;i<n+2;i++)
        for (j=i;j<n+2;j++)
            for (k=0;k<m;k++)
                a[i*(n+2)+j]+=fabs(x1[k]-p[i-2])*fabs(x1[k]-p[j-2])/w[k];
    for (i=1;i<n+2;i++)
        for (j=0;j<i;j++)
            a[i*(n+2)+j]=a[j*(n+2)+i];
}

/*****************************************************************************/
/* Construct the vector b=A*W*y, where A and W are defined in Eqs.(2.6)    */
/* and (2.7), y is the data vector and is defined in Eq.(2.11).            */
/*****************************************************************************/

get_b(p,b,z)
double *p,*b,*z;
{
    int i,k;
    double fabs();

    for (i=0;i<n+2;i++)
        b[i]=0.0;
    for (k=0;k<m;k++)
    {
        b[0]+=z[k]/w[k];
        b[1]+=x1[k]*z[k]/w[k];
    }
    for (i=2;i<n+2;i++)
```

```
        for (k=0;k<m;k++)
            b[i]+=z[k]*fabs(x1[k]-p[i-2])/w[k];
}

/*****************************************************************************/
/* Follow Eq.(2.2) to estimate the approximation error between the measured*/
/* data and the canonical piecewise-linear model.                            */
/*****************************************************************************/

double
get_err(p,b,y,r)
double *p,*b,*y,*r;
{
    int i,k;
    double err=0,fabs();

    for (k=0;k<m;k++)
    {
        r[k]=0;
        r[k]=b[0]+b[1]*x1[k]-y[k];
        for (i=0;i<n;i++)
            r[k]+=b[i+2]*fabs(x1[k]-p[i]);
        r[k]=r[k]/w[k];
        err+=r[k]*r[k];
    }
    return(err);
}

/*****************************************************************************/
/* Find the 1st order derivative g and the 2nd order derivative Y of the   */
/* approximation error w.r.t. the breakpoint position; g is found by       */
/* Eq.(2.15) and represented by the 1-dim array g, Y is found by Eq.(2.20) */
/* and represented by the 1-dim array f.                                   */
/*****************************************************************************/

get_g_F(p,b,r,f,g)
double *p,*b,*r,*f,*g;
{
    int i,j,k;

    for (i=0;i<n;i++)
    {
        for (k=0;k<m;k++)
            if (x1[k]-p[i]<0)
                g[i]+=2*b[i+2]*r[k]/w[k];
            else
                g[i]-=2*b[i+2]*r[k]/w[k];
        for (j=i;j<n;j++)
            for (k=0;k<m;k++)
            {
                if ((x1[k]-p[i])*(x1[k]-p[j])>0)
                {
                    f[i*n+j]+=2*b[i+2]*b[j+2]/w[k];
                    f[j*n+i]=f[i*n+j];
                }
                else
```

```
                {
                    f[i*n+j]-=2*b[i+2]*b[j+2]/w[k];
                    f[j*n+i]=f[i*n+j];
                }
            }
    }
}
```

```
#include <stdio.h>

extern int m,n;
extern double lb,ub,*b1,*p;

/*****************************************************************************/
/* Print the optimal canonical piecewise-linear model into the file     */
/* "xx...x.c", where "xx...x.dat" is the filename of the data file.     */
/*****************************************************************************/

print_pwl(line)
char *line;
{
    int i;
    FILE *fopen(),*fp;

    if ((fp=fopen(line,"w"))==NULL)
    {
        printf("CAN'T OPEN %s\n",line);
        exit();
    }
    /* print the main program */
    fprintf(fp,"main(argc,argv)\n");
    fprintf(fp,"int argc;\n");
    fprintf(fp,"char *argv[];\n");
    fprintf(fp,"{\n    pwl_draw(argc,argv);\n}\n\n");

    /* print the canonical pwl function */
    fprintf(fp,"double pwl(x)\n");
    fprintf(fp,"double x;\n");
    fprintf(fp,"{\n");
    fprintf(fp,"    double z,fabs();\n\n");
    fprintf(fp,"    z= %.3e+%.3e*x;\n",b1[0],b1[1]);
    for (i=0;i<n;i++)
        fprintf(fp,"    z+= %.3e*fabs(x-%.3e);\n",b1[i+2],p[i]);
    fprintf(fp,"    return(z);\n");
    fprintf(fp,"}\n");
    fclose(fp);
}
```

```
#include <stdio.h>

double lc,uc;
extern int n,m;
extern double *x1,*y1,*w,lb,ub;

/*****************************************************************************/
/* Read the data file and assign the weighting factor for each data point. */
/*****************************************************************************/

read_data(fp)
FILE *fp;
{
    char line[81],*calloc();
    int i=0,j;
    double fabs();

    /* temperatory allocation of the data points and the weighting factor */
    x1=(double *)calloc(1000,sizeof(double));
    y1=(double *)calloc(1000,sizeof(double));
    w=(double *)calloc(1000,sizeof(double));

    /* read each data point and the weighting factor */
    i=0;
    lb = 10000;
    ub = -10000;
    lc = 10000;
    uc = -10000;
    while (fgets(line,80,fp) != NULL)
    {
        sscanf(line,"%lf%lf%lf",x1+i,y1+i,w+i);

        /* find the lower and the upper bound */
        if (lb > x1[i])
            lb=x1[i];
        if (ub < x1[i])
            ub=x1[i];
        if (lc > y1[i])
            lc=y1[i];
        if (uc < y1[i])
            uc=y1[i];

        i++;
    }
    m=i;

    /* re-allocation after the exact number of data points is known */
    rallocd(&x1,m,1000);
    rallocd(&y1,m,1000);
    rallocd(&w,m,1000);
}
```

```
#include <stdio.h>
#include "/usr/include/local/graf.h"
#include "gf.h"

int m,n;
double *x1,*y1,*w,lb,ub;
extern double lc,uc;

/*************************************************************************/
/* Draw family of curves of MOS transistor : I_ds vs V_ds parametrized  */
/* by V_gs.                                                              */
/*************************************************************************/

pwl_draw(argc,argv)
int argc;
char *argv[];
{
    GRAF *gp,*graf_open();
    FILE *fopen(),*fp;
    double ymin,ymax;
    char x_name[30],y_name[30],title[30],line[20];
    double x,y,z;
    double pwl();
    int i,j,k;

    if (argc != 2)
        exit_message("PWLDRAW DATAFILE");
    sprintf(line,"%s.dat",*++argv);
    if ((fp=fopen(line,"r")) == NULL)
    {
        printf("CAN'T OPEN THE DATA FILE %s\n",line);
        exit();
    }
    read_data(fp);

    ymin=lc-(uc-lc)*0.2;
    ymax=uc+(uc-lc)*0.2;

    /* assign coordinate titles */
    sprintf(x_name,"x-axis");
    sprintf(y_name,"y-axis");
    sprintf(title,"CANONICAL PWL MODEL");

    mgiasngp(0,0);      /* assign the graphic processor */
    gp=graf_open();

    /* draw the graphic coordinate box */
    setup_graf(lb,ub,ymin,ymax,x_name,y_name,title,gp);

    mgihue(5);

    k=m/5;
    y=pwl(lb);
    graf_move(lb,y,gp);
    for (i=0;i<k;i++)
    {
```

```
        x=lb+5*(i+1)*(ub-lb)/(m-1);
        y=pwl(x);
        graf_draw(x,y,gp);
    }

    mgihue(3);
    for (i=0;i<m;i++)
        graf_point(x1[i],y1[i],gp);

    graf_close(gp);
    mgideagp();
    fclose(fp);
}

/*************************************************************************/
/* Draw the graphic coordinate box.                                      */
/*************************************************************************/

setup_graf(xmin,xmax,ymin,ymax,x_name,y_name,title,gp)
double xmin,xmax,ymin,ymax;
char x_name[],y_name[],title[];
GRAF *gp;
{
    if (gp==NULL)
    {
        printf("gp=NULL\n");
        exit();
    }
    define_colors();
    mgihue(1);
    set_screen(80,600,140,500,gp);
    set_real(xmin,xmax,ymin,ymax,gp);
    set_x_axis(N_LBLS,N_TICKS,TICK_LENGTH,SIG_FIGS,LABEL_SIDE,LABEL_SHIFT,
    x_name,NAME_SHIFT,gp);
    set_y_axis(N_LBLS,N_TICKS,TICK_LENGTH,SIG_FIGS,LABEL_SIDE,LABEL_SHIFT,
    y_name,NAME_SHIFT,gp);
    set_title(title,SIZE,OFFSET,gp);
    TI=1;
    draw_bounds(BOX,LABELS,TICKS,AXES,TI,gp);
}

/*************************************************************************/
/*************************************************************************/

define_colors()
{
    mgipln(31);
    mgiclearpln(0,-1,0);
    mgicm(1,0xf0f0f0L);
    mgicm(2,0x00f0a0L);
    mgicm(3,0xf0f008L);
    mgicm(4,0xf008f0L);
    mgicm(5,0xf00000L);
    mgicm(6,0xa0b005L);
    mgicm(7,0x00f000L);
    mgicm(8,0x0000f0L);
```

```
#include <stdio.h>

/*****************************************************************************/
/* Convert a real number expression terminated by a unit character to a      */
/* real number with double precision.                                        */
/*****************************************************************************/

double stof(s)

char *s;        /* input string expression */
{
    char *d;    /* number field expression */
    char ch;    /* unit character */
    char *calloc();
    double x,atof();

    d=calloc(strlen(s)+1,sizeof(char));
    ch = *(s+strlen(s)-1);      /* extract the last character */
    if (ch<'0' || ch>'9')       /* if it is a unit character */
    {
        strcpy(d,s);
        strdel(d,strlen(s)-1,1);        /* extract the number field */
        x=atof(d);
        switch(ch)
        {
            case 'K' : x=x*1e3;   break;        /* Kilo */
            case 'M' : x=x*1e6;   break;        /* Mega */
            case 'G' : x=x*1e9;   break;        /* Giga */
            case 'T' : x=x*1e12;  break;        /* Tera */
            case 'm' : x=x*1e-3;  break;        /* milli */
            case 'u' : x=x*1e-6;  break;        /* micro */
            case 'n' : x=x*1e-9;  break;        /* naro */
            case 'p' : x=x*1e-12; break;        /* pico */
            case 'f' : x=x*1e-15; break;        /* femptl */
            default  : {
                           printf("UNDEFINED UNIT CHARACTER %c\n",ch);
                           exit();
                       }
                       break;
        }
    }
    else x=atof(s);
    return(x);
}

/*****************************************************************************/
/* Give the sign of a real number x; 1 if x)=0 and -1 if x=0.                */
/*****************************************************************************/

sgn(x)

double x;
{
    if (x)=0) return(1);
    else return(-1);
}
```

```
/*************************************************************************/
/* Switch two real number.                                               */
/*************************************************************************/

switch_d(a,b)

double *a,*b;
{
    double c;

    c = *a;
    *a = *b;
    *b = c;
}

/*************************************************************************/
/* Switch two integer number.                                            */
/*************************************************************************/

switch_i(a,b)

int *a,*b;
{
    int c;

    c = *a;
    *a = *b;
    *b = c;
}

/*************************************************************************/
/* Find the i-th word w from the string s.                               */
/*************************************************************************/

find_word(s,w,i)

char *s,*w;
int i;
{
    int k=1,m=0,n=0,j;
    char *ps;

    j=strlen(s)+1;
    ps=s;       /* starting position in the input string s */
    while (*s==' ' && m++<j)    /* delete leading blanks in s */
        s++;
    if (m<j)
    {
        while (k<i && n++<j)
        {
            /* search the starting position for the i-th word */
            if (*s==' ' && *(s+1)!=' ')
                k++;
            s++;
        }
```

```
        while (*s!=' ' && *s!='\t' && *s!='\0' && n<j)
            *w++ = *s++;        /* copy the i-th word to w */
        *w='\0';
    }
    if (m>=j || n>=j)
    {
        printf("FAIL TO FIND A WORD IN %s\n",ps);
        exit();
    }
}

/*************************************************************************/
/* Find the position of the string t within the string s; -1 is returned  */
/* if t is not found within s.                                             */
/*************************************************************************/

find_index(t,s)
char *s,*t;
{
    int i,j,k;

    for (i=0;s[i]!='\0';i++)
    {
        for (j=i,k=0;t[k]!='\0' && s[j]==t[k];j++,k++);
        if (t[k]=='\0')
            return(i);
    }
    return(-1);
}

/*************************************************************************/
/* Delete n characters from the n1-th position of string s.                */
/*************************************************************************/

strdel(s,n1,n)
char *s;
int n1,n;
{
    int i,j,k=0;

    j=strlen(s)+1;
    for (i=0;k<=j,s[i+n1+n]!='\0';k++,i++)
        s[i+n1]=s[i+n1+n];
    if (k>j)
    {
        printf("ERROR IN STRDEL IN STRING\n");
        printf("%s\n",s);
        exit();
    }
    else s[i+n1]='\0';
}

/*************************************************************************/
/* Get rid of unnecessary spaces in the string.                            */
/*************************************************************************/
```

```
sqeez(s)
char *s;
{
    char *sx,*tx,*t,*calloc();
    int i=0,k;

    sx=s;
    k=strlen(s);
    t=calloc(k+1,sizeof(int));
    tx=t;
    while (*sx != '\0' && i++<k)
    {
        if (*sx != ' ' && *sx != '\t' && *sx != '\n')
            *tx++ = *sx++;
        else sx++;
    }
    *tx='\0';
    if (i>k)
        exit_message("ERROR IN SQEEZ");
    else
        strcpy(s,t);
}

/*****************************************************************************/
/* Print a message and exit.                                                 */
/*****************************************************************************/

exit_message(message)
char *message;
{
    printf("%s\n",message);
    exit();
}

/*****************************************************************************/
/* allocate space for integer.                                               */
/*****************************************************************************/

calloci(n,pt,s)
int n,**pt;
char *s;
{
    char *calloc();
    int *p;

    p =(int *) calloc(n,sizeof(int));
    if (p == NULL)
    {
        printf("CAN'T ALLOCATE SPACE FOR %s\n",s);
        exit();
    }
    else
        *pt = p;
}

/*****************************************************************************/
```

```
/* Allocate space for double.                                                 */
/*****************************************************************************/

callocd(n,pt,s)
int n;
double **pt;
char *s;
{
    char *calloc();
    double *p;

    p = (double *) calloc(n,sizeof(double));
    if ( p == NULL )
    {
        printf("CAN'T ALLOCATE SPACE FOR %s\n",s);
        exit();
    }
    else
        *pt = p;
}

/*****************************************************************************/
/* Re-allocate oldsize of integers to a new area with newsize spaces.        */
/*****************************************************************************/

ralloci(ip,newsize,oldsize)
int **ip,newsize,oldsize;
{
    char *calloc();
    int i,size,*pt;

    /* allocate a new space */
    pt = *ip;
    if ((*ip= (int *) calloc(newsize,sizeof(int)))==NULL)
        exit_message("CAN'T RE_ALLOCATE");

    if (newsize<oldsize)
        size=newsize;
    else
        size=oldsize;

    /* move the data to the new area */
    for (i=0;i<size;i++)
        (*ip)[i]=pt[i];

    cfree(pt);
}

/*****************************************************************************/
/* Re-allocate oldsize of doubles to a new area with newsize spaces.         */
/*****************************************************************************/

rallocd(dp,newsize,oldsize)
int newsize,oldsize;
double **dp;
{
```

```
    char *calloc();
    int i,size;
    double *pt;

    /* allocate the a new space */
    pt = *dp;
    if ((*dp= (double *) calloc(newsize,sizeof(double)))==NULL)
        exit_message("CAN'T RE_ALLOCATE");

    if (newsize<oldsize) size=newsize;
    else size=oldsize;

    /* move the data to the new area */
    for (i=0;i<size;i++)
        (*dp)[i]=pt[i];

    cfree(pt);
}

/*****************************************************************************/
/* Transform a double precision number to its ASCII code.                    */
/*****************************************************************************/

f_to_ax(x,d)
double x;
char d[];
{
    int i;
    char e[20],e1[8],e2[10];
    double fabs();

    sprintf(e,"%.5e",fabs(x));  /* e = ASCII code of |x| */
    strncpy(e1,e,7);            /* e1 = number field of e */
    e1[7]='\0';

    /* delete unnecessary '0' in e1 */
    for (i=6;i>0;i--)
    {
        if (e1[i]=='0' || e1[i]=='.') e1[i]='\0';
        else break;
    }

    strcpy(e2,e+7);             /* e2 = exponent of e */

    /* delete unnecessary '0' in e2 */
    i=0;
    while (i++<2 && e2[2]=='0')
        strdel(e2,2,1);
    if (strlen(e2)==2) e2[0]='\0';
    if (e2[1]=='+') strdel(e2,1,1);

    /* concate e1 (number) with e2 (exponent) */
    if (x<0)
        sprintf(d,"-%s%s",e1,e2);
    else
        sprintf(d,"%s%s",e1,e2);
```

```
#include <stdio.h>

int m,n;
double *x1,*x2,*y1,*r1,*b1,*p,*q,error,*w;

/*****************************************************************************/
/* Canonical piecewise-linear modeling : 2-dimensional function.             */
/*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    FILE *fp,*fopen();
    char line[30];

    /* open the data file */
    if (argc != 2)
        exit_message("PWLMOD2 DATA_FILE");
    sprintf(line,"%s.dat",*++argv);
    if ((fp=fopen(line,"r")) == NULL)
    {
        printf("CAN'T OPEN THE DATA FILE\n");
        exit();
    }

    /* canonical pwl modeling */
    pwl_model(fp);

    fclose(fp);

    /* print the optimal canonical pwl model equation */
    /* to the output file xx...x.c                    */
    sprintf(line,"%s.c",*argv);
    print_pwl(line);
}

/*****************************************************************************/
/* Follow the formulae of Section 3 of the paper (ERL Memo M85/35) to find */
/* the optimal canonical piecewise-linear model for 2-port (or 3-terminal) */
/* elements.                                                                 */
/*****************************************************************************/

pwl_model(fp)
FILE *fp;
{
    char ch[2];
    int i;

    /* read the data and get the initial partition boundaries */
    get_data(fp);

    /* find the optimal canonical pwl model w.r.t. the initial partition */
    if (opt_p(p,q,&error)==-1)
        exit_message("IMPROPER BOUNDARIES");
```

```
    /* adjust the partition boundaries to get the optimal canonical */
    /* 2-dimensional canonical pwl model                            */
    ch[0]='y';
    while (ch[0]=='y')
    {
        /* perform the line search */
        l_search();

        /* print the optimal canonical pwl model */
        printf("\n*** partition boundaries ***\n");
        for (i=0;i<n;i++)
            printf("%.2e*x - y + %.2e = 0 \n",p[i],q[i]);
        printf("a=%.3e\tb11=%.3e\tb12=%.3e\n",b1[0],b1[1],b1[2]);
        for (i=0;i<n;i++)
            printf("c1%d=%.3e\n",i+1,b1[i+3]);
        printf("continue? y/n\n");
        scanf("%1s",ch);

        /* try to get out of a local minimum */
        if (ch[0]=='z')
        {
            getout();
            ch[0]='y';
        }
    }
}

/*****************************************************************************/
/* Read the input data points and the initial partition boundariies.         */
/*****************************************************************************/

get_data(fp)
FILE *fp;
{
    char *calloc();
    int i;

    /* read the measured data */
    read_data(fp);

    printf("enter the number of partition boundaries\n");
    scanf("%d",&n);
    v_alloc();
    printf("enter the initial set of partition boundaries\n");
    for (i=0;i<n;i++)
        scanf("%lf%lf",p+i,q+i);
}

/*****************************************************************************/
/* Dynamic allocation for the variables.                                     */
/*****************************************************************************/

v_alloc()
{
    char *calloc();
```

```
    r1=(double *)calloc(m,sizeof(double));
    b1=(double *)calloc(n+3,sizeof(double));
    p=(double *)calloc(n,sizeof(double));
    q=(double *)calloc(n,sizeof(double));
}

/*****************************************************************************/
/* Perform the line search on the partition boundaries for reducing the      */
/* approximation error.                                                      */
/*****************************************************************************/

l_search()
{
    char *calloc();
    int i,*ipvt,red=0;
    double t=1,*f,*g,*zq,rcond,max=0,fabs(),ratio=1;

    f=(double *)calloc(4*n*n,sizeof(double));
    g=(double *)calloc(2*n,sizeof(double));
    ipvt=(int *)calloc(2*n,sizeof(int));
    zq=(double *)calloc(2*n,sizeof(double));

    /* get the matrix Y and the vector g in Eqs.(2.15-2.16) */
    get_g_F(p,q,b1,r1,f,g);

    /* find the line search direction inv(Y)*g */
    for (i=0;i<2*n;i++)
        g[i] = -g[i];
    sgeco(f,2*n,ipvt,&rcond,zq);
    sgesl(f,2*n,ipvt,g,0);

    /* find the infinite norm */
    for (i=0;i<2*n;i++)
        if (fabs(g[i])>max)
            max=fabs(g[i]);

    /* get a new partition along the line direction */
    while (t>0.1 && fabs(ratio)>1e-3)
        new_p(&red,&t,g,max+1,&ratio);

    cfree(f);
    cfree(g);
    cfree(ipvt);
    cfree(zq);
}

/*****************************************************************************/
/* Adjust the partition boundaries along the line search direction.          */
/*****************************************************************************/

new_p(red,t,s,max,ratio)
int *red;
double *s,*t,max,*ratio;
{
    char *calloc();
    int i;
```

```
    double err,*px,*qx,rat,fabs();

    px=(double *)calloc(2*n,sizeof(double));
    qx=(double *)calloc(2*n,sizeof(double));
    for (i=0;i<n;i++)
    {
        px[i]=p[i];
        qx[i]=q[i];
        p[i]+= *t*s[i]/max;
        q[i]+= *t*s[i+n]/max;
    }
    rat = *ratio;
    if (opt_p(p,q,&err)==-1)
        *ratio=1;
    else
        *ratio=(err-error)/error;
    printf("approximation error = %.3e\n",err);
    rat=(rat-*ratio)/rat;

    /* reduce the increment size if the error is increasing */
    if (*ratio>0)
    {
        *t = *t/4;
        *red=1;
        for (i=0;i<n;i++)
        {
            p[i]=px[i];
            q[i]=qx[i];
        }
    }
    else
    {
        /* increase the increment size if the error decreases */
        if ((*ratio<-0.02 || fabs(rat)<0.1)  && *red==0)
            *t=2*(*t);
        error=err;
    }
    cfree(px);
    cfree(qx);
}

/*************************************************************************/
/* Adjust the partition boundaries until the approximation error gets out  */
/* of the range of a local minimum.                                        */
/*************************************************************************/

getout()
{
    char *calloc();
    int i,t=0;
    double *px,*qx,*ss,err,errx,ratio;

    px=(double *)calloc(n,sizeof(double));
    qx=(double *)calloc(n,sizeof(double));
    ss=(double *)calloc(n,sizeof(double));
```

```
    /* find the increment size */
    for (i=0;i<n;i++)
        if (-1*q[i]/p[i]>3.5)
            ss[i]=(5-q[i])/25;
        else
            ss[i] = -1*(5*p[i]+q[i])/25;

    /* get a new partition */
    while (t++<=10)
    {
        for (i=0;i<n;i++)
            qx[i]=q[i]+t*ss[i];
        opt_p(p,qx,&err);
        if (t>1)
            ratio=(errx-err)/err;
        else
            ratio=(error-err)/error;
        printf("approximation error = %.3e\n",err);
        if (ratio>0.02)
        {
            error=err;
            for (i=0;i<n;i++)
                q[i]=qx[i];
            break;
        }
        errx=err;
    }
    for (i=0;i<n;i++)
        q[i]=qx[i];
    cfree(ss);
    cfree(px);
    cfree(qx);
}
```

```
#include <stdio.h>

extern int n,m;
extern double *x1,*x2,*y1,*r1,*b1,*w;

/*****************************************************************************/
/* Find the optimal parameters of a canonical piecewise-linear model w.r.t.*/
/* a set of partition boundaries.                                            */
/*****************************************************************************/

opt_p(p,q,err)
double *p,*q,*err;
{
    char *calloc();
    int *ipvt;
    double *a,*zq,rcond,get_err();

    ipvt=(int *)calloc(n+3,sizeof(int));
    zq=(double *)calloc(n+3,sizeof(double));
    a=(double *)calloc((n+3)*(n+3),sizeof(double));

    /* Find the matrix B=A*W*A' in Eq.(3.17) */
    get_A(p,q,a);

    /* find the vector b=A*W*y in Eq.(3.17) */
    get_b(p,q,b1,y1);

    /* solve B*z=b for the optimal canonical pwl parameters */
    sgeco(a,n+3,ipvt,&rcond,zq);
    if (rcond<1.0e-10)
        return(-1);
    sgesl(a,n+3,ipvt,b1,0);

    /* find the estimation error */
    *err=get_err(p,q,b1,y1,r1);

    cfree(a);
    cfree(ipvt);
    cfree(zq);
    return(0);
}

/*****************************************************************************/
/* Construct the matrix B=A*W*A', where A is defined in Eq.(3.13).           */
/*****************************************************************************/

get_A(p,q,a)
double *p,*q,*a;
{
    int i,j,k;
    double fabs();

    for (k=0;k<m;k++)
    {
        a[0]+=w[k];
        a[1]+=x1[k]*w[k];
```

```
        a[2]+=x2[k]*w[k];
    }
    for (i=3;i<n+3;i++)
        for (k=0;k<m;k++)
            a[i]+=fabs(p[i-3]*x1[k]-x2[k]+q[i-3])*w[k];
    for (k=0;k<m;k++)
    {
        a[n+4]+=x1[k]*x1[k]*w[k];
        a[n+5]+=x1[k]*x2[k]*w[k];
    }
    for (i=n+6;i<2*n+6;i++)
        for (k=0;k<m;k++)
            a[i]+=x1[k]*fabs(p[i-n-6]*x1[k]-x2[k]+q[i-n-6])*w[k];
    for (k=0;k<m;k++)
        a[2*n+8]+=x2[k]*x2[k]*w[k];
    for (i=2*n+9;i<3*n+9;i++)
        for (k=0;k<m;k++)
            a[i]+=x2[k]*fabs(p[i-2*n-9]*x1[k]-x2[k]+q[i-2*n-9])*w[k];
    for (i=3;i<n+3;i++)
        for (j=i;j<n+3;j++)
            for (k=0;k<m;k++)
                a[i*(n+3)+j]+=fabs(p[i-3]*x1[k]-x2[k]+q[i-3])*
                              fabs(p[j-3]*x1[k]-x2[k]+q[j-3])*w[k];
    for (i=1;i<n+3;i++)
        for (j=0;j<i;j++)
            a[i*(n+3)+j]=a[j*(n+3)+i];
}

/*****************************************************************************/
/* Construct the vector b=A*W*y, where A and y are defined in Eqs.(3.13)     */
/* and (3.16) respectively.                                                  */
/*****************************************************************************/

get_b(p,q,b,z)
double *p,*q,*b,*z;
{
    int i,k;
    double fabs();

    for (i=0;i<n+3;i++)
        b[i]=0.0;
    for (k=0;k<m;k++)
    {
        b[0]+=z[k]*w[k];
        b[1]+=x1[k]*z[k]*w[k];
        b[2]+=x2[k]*z[k]*w[k];
    }
    for (i=3;i<n+3;i++)
        for (k=0;k<m;k++)
            b[i]+=z[k]*fabs(p[i-3]*x1[k]-x2[k]+q[i-3])*w[k];
}

/*****************************************************************************/
/* Estimate the approximation error w.r.t. a canonical piecewise-linear      */
/* model.                                                                    */
/*****************************************************************************/
```

```
double
get_err(p,q,b,y,r)
double *p,*q,*b,*y,*r;
{
    int i,k;
    double err=0,fabs();

    for (k=0;k<m;k++)
    {
        r[k]=0;
        r[k]=b[0]+b[1]*x1[k]+b[2]*x2[k]-y[k];
        for (i=0;i<n;i++)
            r[k]+=b[i+3]*fabs(p[i]*x1[k]-x2[k]+q[i]);
        r[k]=r[k]*w[k];
        err+=r[k]*r[k];
    }
    return(err);
}

/*****************************************************************************/
/* Find the matrix Y in Eq.(3.22) (represented by the 1-dim array f) and     */
/* the vector g in Eq.(3.21) (represented by the 1-dim array g).             */
/*****************************************************************************/

get_g_F(p,q,b,r,f,g)
double *p,*q,*b,*r,*f,*g;
{
    int i,j,k;

    for (i=0;i<n;i++)
    {
        for (k=0;k<m;k++)
        {
            if (p[i]*x1[k]-x2[k]+q[i]>=0)
            {
                g[i]+=2*b[i+3]*x1[k]*r[k]*w[k];
                g[i+n]+=2*b[i+3]*r[k]*w[k];
            }
            else
            {
                g[i]-=2*b[i+3]*x1[k]*r[k]*w[k];
                g[i+n]-=2*b[i+3]*r[k]*w[k];
            }
        }
        for (j=0;j<n;j++)
            for (k=0;k<m;k++)
            {
                if ((p[i]*x1[k]-x2[k]+q[i])*(p[j]*x1[k]-x2[k]+q[j])>0)
                {
                    f[i*2*n+j]+=2*x1[k]*x1[k]*b[i+3]*b[j+3]*w[k];
                    f[i*2*n+n+j]+=2*x1[k]*b[i+3]*b[j+3]*w[k];
                    f[(n+j)*2*n+i]=f[i*2*n+n+j];
                    f[(n+i)*2*n+n+j]+=2*b[i+3]*b[j+3]*w[k];
                }
                else
```

```
                {
                    f[i*2*n+j]-=2*x1[k]*x1[k]*b[i+3]*b[j+3]*w[k];
                    f[i*2*n+n+j]-=2*x1[k]*b[i+3]*b[j+3]*w[k];
                    f[(n+j)*2*n+i]=f[i*2*n+n+j];
                    f[(n+i)*2*n+n+j]-=2*b[i+3]*b[j+3]*w[k];
                }
            }
        }
}
```

```
#include <stdio.h>

extern int n;
extern double *b1,*p,*q;

/*****************************************************************************/
/* Print the optimal canonical piecewise-linear model into the file     */
/* "pwl.c".                                                                  */
/*****************************************************************************/

print_pwl(line)
char *line;
{
    int i;
    FILE *fp,*fopen();

    if ((fp=fopen(line,"w")) == NULL)
    {
        printf("CAN'T OPEN %s\n",line);
        exit();
    }

    /* print the main program */
    fprintf(fp,"main(argc,argv)\n");
    fprintf(fp,"int argc;\n");
    fprintf(fp,"char *argv[];\n");
    fprintf(fp,"{\n    pwl_draw1(argc,argv);\n}\n\n");

    /* print the canonical pwl function */
    fprintf(fp,"double pwl(x,y)\n");
    fprintf(fp,"double x,y;\n");
    fprintf(fp,"{\n");
    fprintf(fp,"    double z,fabs();\n\n");
    fprintf(fp,"    z= %.3e+%.3e*x+%.3e*y;\n",b1[0],b1[1],b1[2]);
    for (i=0;i<n;i++)
        fprintf(fp,"    z+= %.3e*fabs(%.3e*x-y+%.3e);\n",b1[i+3],p[i],q[i]);
    fprintf(fp,"    return(z);\n");
    fprintf(fp,"}\n");
}
```

```
#include <stdio.h>

extern int n,m;
extern double *x1,*x2,*y1,*w;
double x11,xu1,x12,xu2,y11,yu1;

/*****************************************************************************/
/* Read the data file and assign the weighting factor for each data point. */
/*****************************************************************************/

read_data(fp)
FILE *fp;
{
    char line[81],*calloc();
    int i=0,j;
    double fabs();

    /* temperatory allocation of the data points and the weighting factor */
    x1=(double *)calloc(1000,sizeof(double));
    x2=(double *)calloc(1000,sizeof(double));
    y1=(double *)calloc(1000,sizeof(double));
    w=(double *)calloc(1000,sizeof(double));

    /* read each data point and the weighting factor */
    i=0;
    x11 = 10000;
    xu1 = -10000;
    x12 = 10000;
    xu2 = -10000;
    y11 = 10000;
    yu1 = -10000;
    while (fgets(line,80,fp) != NULL)
    {
        sscanf(line,"%lf%lf%lf%lf",x1+i,x2+i,y1+i,w+i);
        if (x11 > x1[i])
            x11 = x1[i];
        if (xu1 < x1[i])
            xu1 = x1[i];
        if (x12 > x2[i])
            x12 = x2[i];
        if (xu2 < x2[i])
            xu2 = x2[i];
        if (y11 > y1[i])
            y11 = y1[i];
        if (yu1 < y1[i])
            yu1 = y1[i];
        i++;
    }
    m=i;

    /* re-allocation after the exact number of data points is known */
    rallocd(&x1,m,1000);
    rallocd(&x2,m,1000);
    rallocd(&y1,m,1000);
    rallocd(&w,m,1000);
}
```

```
#include <stdio.h>
#include "/usr/include/local/graf.h"
#include "gf.h"

int n,m,nc,opt;
double *x1,*x2,*y1,*w,*p;
extern double xl1,xu1,xl2,xu2,yl1,yu1;

/*************************************************************************/
/* Draw family of curves of the canonical pwl model y1=f(x1,x2), for     */
/* y1 vs x1 with fixed values of x2, or y1 vs x2 with fixed values of    */
/* x1.                                                                   */
/*************************************************************************/

pwl_draw1(argc,argv)
int argc;
char *argv[];
{
    GRAF *gp,*graf_open();
    FILE *fp;
    double ymin=0,ymax=500;
    char line[30],x_name[30],y_name[30],title[40];

    /* readn the data points */
    open_data(argc,argv,&fp);
    read_data(fp);

    /* input the graph information */
    get_info();

    /* assign coordinate titles */
    if (opt == 1)
        sprintf(x_name,"X1");
    else
        sprintf(x_name,"X2");
    sprintf(y_name,"Y1");
    sprintf(title,"2-dim CANONICAL PWL MODEL");

    mgiasngp(0,0);      /* assign the graphic processor */
    gp=graf_open();

    /* draw the graphic coordinate box */
    ymin=yl1-(yu1-yl1)*0.2;
    ymax=yu1+(yu1-yl1)*0.2;
    setup_graf(xl2,xu2,ymin,ymax,x_name,y_name,title,gp);

    draw_pwl(gp);

    graf_close(gp);
    mgideagp();
}

/*************************************************************************/
/* Draw the 2-dimensional canonical pwl model equation in terms of       */
/* family of curves; either y1 vs x1 with fixed x2, or y1 vs x2 with     */
/* fixed x1.                                                             */
/*************************************************************************/
```

```
/*************************************************************************/

draw_pwl(gp)
GRAF *gp;
{
    char is[10];
    int i,j;
    double x,y,pwl();

    /* draw nc curves */
    for (i=0;i<nc;i++)
    {
        mgihue(i+2);
        /* draw y1 vs x1 with fixed x2 */
        if (opt == 1)
        {
            sprintf(is,"X2=%4.2f",p[i]);
            graf_move(xl1,pwl(xl1,p[i]),gp);
            for (j=0;j<=20;j++)
            {
                x=xl1+j*(xu1-xl1)/20;
                y=pwl(x,p[i]);
                graf_draw(x,y,gp);
            }
        }
        else
        {
            /* draw y1 vs x2 with fixed x1 */
            sprintf(is,"X1=%4.2f",p[i]);
            graf_move(xl2,pwl(p[i],xl2),gp);
            for (j=0;j<=20;j++)
            {
                x=xl2+j*(xu2-xl2)/20;
                y=pwl(p[i],x);
                graf_draw(x,y,gp);
            }
        }

        /* draw the x1 or x2 value on top of the corresponding curve */
        mgimode(1);
        mgixy(-40,0);
        mgigfs(0,0,0,is);
        mgimode(0);
    }
}

/*************************************************************************/
/* Input the information for drawing the graph.                          */
/*************************************************************************/

get_info()
{
    char *calloc();
    int i;

    printf("enter the number of curves to be plotted\n");
```

```
    scanf("%d",&nc);
    p=(double *)calloc(nc,sizeof(double));
    printf("enter the option number\n");
    printf("option 1 : y vs x1 for fixed x2\n");
    printf("option 2 : y vs x2 for fixed x1\n");
    scanf("%d",&opt);
    for (i=0;i<nc;i++)
    {
        printf("the %d-th parameter value = ");
        scanf("%lf",p+i);
    }
    printf("\n");
}

/*****************************************************************************/
/* Open the data file "xx...x.dat".                                          */
/*****************************************************************************/

open_data(argc,argv,fp)
int argc;
char *argv[];
FILE **fp;
{
    FILE *fopen();
    char line[20];

    if (argc != 2)
        exit_message("PWLDRAW2 DATA_FILE");
    sprintf(line,"%s.dat",*++argv);
    if ((*fp=fopen(line,"r")) == NULL)
    {
        printf("CAN'T OPEN THE DATA FILE %s\n",line);
        exit();
    }
}

/*****************************************************************************/
/* Draw the graphic coordinate box.                                          */
/*****************************************************************************/

setup_graf(xmin,xmax,ymin,ymax,x_name,y_name,title,gp)
double xmin,xmax,ymin,ymax;
char x_name[],y_name[],title[];
GRAF *gp;
{
    if (gp==NULL)
    {
        printf("gp=NULL\n");
        exit();
    }
    define_colors();
    mgihue(1);
    set_screen(80,600,140,500,gp);
    set_real(xmin,xmax,ymin,ymax,gp);
    set_x_axis(N_LBLS,N_TICKS,TICK_LENGTH,SIG_FIGS,LABEL_SIDE,LABEL_SHIFT,
    x_name,NAME_SHIFT,gp);
```

```
    set_y_axis(N_LBLS,N_TICKS,TICK_LENGTH,SIG_FIGS,LABEL_SIDE,LABEL_SHIFT,
    y_name,NAME_SHIFT,gp);
    set_title(title,SIZE,OFFSET,gp);
    TI=1;
    draw_bounds(BOX,LABELS,TICKS,AXES,TI,gp);
}

/*****************************************************************************/
/*****************************************************************************/

define_colors()
{
    mgipln(31);
    mgiclearpln(0,-1,0);
    mgicm(1,0xf0f0f0L);
    mgicm(2,0x00f0a0L);
    mgicm(3,0xf0f008L);
    mgicm(4,0xf008f0L);
    mgicm(5,0xf00000L);
    mgicm(6,0xa0b005L);
    mgicm(7,0x00f000L);
    mgicm(8,0x0000f0L);
}
```