FINAL REPORT

PRESENTED BY

FRANK J. BLANDO

DEPARTMENT OF MATHEMATICS

TO

DR. W. PERRY

APRIL 21, 1981

ABSTRACT

This report concludes a year of investigative research in artificial intelligence. Artificial intelligence is a field of computer science in which recreating human behavior in machines is the main subject; the field is divided into four areas of research: pattern recognition, natural language, problem solving, and learning and reasoning. The first half of the year of research was spent learning about artificial intelligence in general, then the second half was spent focusing on pattern recognition.

First, pattern recognition was divided into two parts, feature extraction and feature recognition. Three algorithms were implemented on a large computer, the Isodata clustering algorithm, the K-mean clustering algorithm, and the Block-Nilson-Duda algorithm for feature extraction. The theory of these algorithms relies on analytic geometry and common quantities such as the average, the standard deviation, and the distance between two points.

Computers at the TAMU Cyclotron Institute were used to develop a set of subroutines, which are used in energy spectrum analysis, to extract peaks in the data and identify them. The subroutines follow the same procedure as would the human operator; they first approximate the background, subtract the approximation, and then look for peaks in the remaining data. The set of subroutines in appendix C of this paper is the result of the last year of research.

TABLE OF CONTENT

Preamble 1

Clustering algorithms and feature extraction . 2

Feature extraction 3

Clustering algorithms. 4

Isodata algorithm. 5

K-mean algorithm 5

Table 1. 6

Table 2. 7

Motivations. 8

Theory 9

Peak identification. 10

Background approximation 13

Research time-table. 17

Conclusion 17

Bibliography 19

Appendices 20

Isodata clustering program . . . . . . A

Pictures 1 through 7 . . . . B

Subroutine listings. C

PREAMBLE

In May 1980, I presented a proposal for research, in the University Undergraduate Fellow Program, in artificial intelligence. In the proposal, I did not define artificial inteligence; nor did I precisely state the subject of the research, because as a mathematics major I had no previous background in the field. Artificial intelligence was chosen because it is a field in which computer science, applied mathematics and pure mathematics are used. Therefore my first objective was to learn about artificial intelligence and precisely define the subject of my project.

During the first summer session of 1980, I enrolled in English 301 (Technical Writing) and chose artificial intelligence as the topic for my term paper. I became acquainted with the Texas A&M library and the material available on artificial intelligence. Most of my research time was spent on perceptrons(1), and the term paper was entirely on perceptron and hash coding techniques.

During the fall semester of 1980 I audited Dr. D. Friesen's computer science 625 course, a graduate course in artificial intelligence. In the course, artificial intelligence was defined as an entity possessed by a man-made object the action of which are indistinguishable from those of a human in the same situation. We learned that the machine only has to act like a human to possess artificial intelligence; it does not have to reason or decide to act in the same manner as a human would. Hence, only the end result determines possession of artificial intelligence. This definition allows varied approaches to the same problem so a profusion of algorithms and theories results.

Using the text book(2), we performed an overview of the current artificial intelligence research status. We discussed pattern recognition, classification methods, Euclidean description, clustering algorithms, perceptrons, convergence, grammatical inference, feature extraction, heuristics, search algorithm, the resolution problem, problem solving, natural languages, and other techniques. The final exam was replaced by a report, in which I decided to test some different clustering algorithms. I wrote the isodata clustering algorithm(3) in FORTRAN-77 using a VAX-11/780 computer, and used this to get a feel for how such algorithms can be used. The completed report included the test data and a list of bench mark results.

CLUSTERING ALGORITHMS AND FEATURE EXTRACTION

Since in artificial intelligence we are interested in reproducing human behavior in machines, we have to reproduce or simulate the human capability for analysis. The base for human intelligence lies in the human brain's power to analyze, classify, and store events; therefore, for a machine to simulate intelligence it must have the capability to analyze, classify and store information. To analyze a situation, one has to be able to extract relevant and unique facts from the situation, and then classify them according to some known and already classified precedents. This thinking process involves pattern recognition and clustering capabilities. In pattern recognition, the goal is to develop an algorithm which extracts features from a given situation or picture; then, using these features, the algorithm should classify the events according to some known pattern(4).

One of the feature extraction algorithm is the Block-Nilsson-Duda algorithm for feature extraction(5). This algorithm operates on a N dimensional array, which can be a digitized picture or an event representation in an N dimensional Euclidean space. The algorithm assumes the data is in a binary format; any cell in the N dimensional array can assume only one of two values, here we will call them ON and OFF. The program involves a threshold value which is used to decide whether an array possesses a feature or not. The array to be analyzed is "anded" with every feature and then a count of the ON elements in the result is taken. If that count exceeds the threshold, then that array is assigned the feature. The "and" operation is a logical "and", and the result follows the rule:

feature

	AND	ON	OFF
array	ON	ON	OFF
	OFF	OFF	OFF

Also, when an array is recognized to have a feature, and then later on the feature is positively identified, then the algorithm replaces the feature by the result of the "and"; this produces a possible improvement of the program. It should be noted that the threshold value is related to the number of features to be recognized, only N-1 features are required to recognize N objects(6).

Once a feature has been extracted from a context, it is necessary to identify it with respect to some known data. This is where clustering algorithms are used; the unidentified features are represented in an Euclidean space, and then clusters are formed. Most clustering algorithms rely heavily on distance and standard deviation; the Euclidean distance D between 2 points in an N dimensional space is given by:

$$D=\text{Square root}\left(\sum_{I=1}^{N} \text{Square}(X(I)-Y(I)) \right)$$

Where X(I) is the Ith coordinate of the first point, and Y(I) is the Ith coordinate of the second point. The sample mean of a set of N points is given by:

$$\text{Mean} = \left(\sum_{I=1}^{N} X(I) \right) / N$$

And the standard deviation of a set of N points is then given by:

$$\text{Std Dev} = \text{Square root}\left(\left(\sum_{I=1}^{N} \text{Square}(X(I)) - N*\text{Square}(\text{Mean})\right)/(N-1)\right)$$

The first clustering algorithm tested is the isodata clustering algorithm. The algorithm starts by assuming that there are as many clusters as there are points in the space. Then it merges clusters with centers close to one another, and it splits clusters with an internal standard deviation greater than some threshold value. The algorithm repeats this merging and splitting process until the result stays constant or starts oscillating between constant states. Before starting, the algorithm requires a rough estimate on the final number of clusters; the algorithm will converge only if the true number of clusters lies between double the estimate and half the estimate. During the entire process, the clusters are referenced by their centers; once the algorithm is finished each point is assigned to a cluster(7). A listing of the program is provided in appendix A, and an input sample with the generated output is given in table 1.

The next clustering algorithm studied is the K-mean algorithm. Unlike the isodata algorithm, the K-mean

Table 1

Input data for isodata program: 44 points in 2 dimensions

(-1,-1)	(-1,0)	(-1,1)	(0,1)
(0,0)	(0,-1)	(1,-1)	(1,-1)
(1,0)	(1,1)	(4,9)	(4,8)
(4,7)	(5,9)	(5,8)	(5,7)
(5,4)	(5,3)	(5,2)	(5,1)
(6,9)	(6,8)	(6,7)	(6,4)
(6,3)	(6,2)	(6,1)	(7,4)
(7,3)	(7,2)	(7,1)	(8,4)
(8,3)	(8,2)	(8,1)	(10,1)
(10,0)	(10,-1)	(11,1)	(11,0)
(11,-1)	(12,1)	(12,0)	(12,-1)

The input parameters were:

Approximate number of clusters:	5
Standard deviation paramters:	2
Minimum # of points per cluster:	3
Lumping parameter:	3
Number of changes per pass:	30

The result was:

Cluster 1 at (0,0) with:	(0,0)	(0,1)	(1,-1)	(1,1)
	(-1,1)	(0,-1)	(1,0)	(-1,0)
	(-1,-1)			
Cluster 2 at (5,8) with:		(4,9)	(4,8)	(4,7)
		(5,9)	(5,8)	(5,7)
		(6,9)	(6,8)	(6,7)
Cluster 3 at (6.5,2.5) with:		(5,4)	(5,3)	(5,2)
		(5,1)	(6,1)	(6,2)
		(6,3)	(6,4)	(7,1)
		(7,2)	(7,3)	(7,4)
		(8,1)	(8,2)	(8,3)
		(8,4)		
Cluster 4 at (11,0) with:		(10,1)	(10,0)	(10,-1)
		(11,1)	(11,0)	(11,-1)
		(12,1)	(12,0)	(12,-1)

Table 2

The input data was: (0,0) (1,2) (5,2) (1,0)

The produced polynomial coefficients were:

-6 2 2

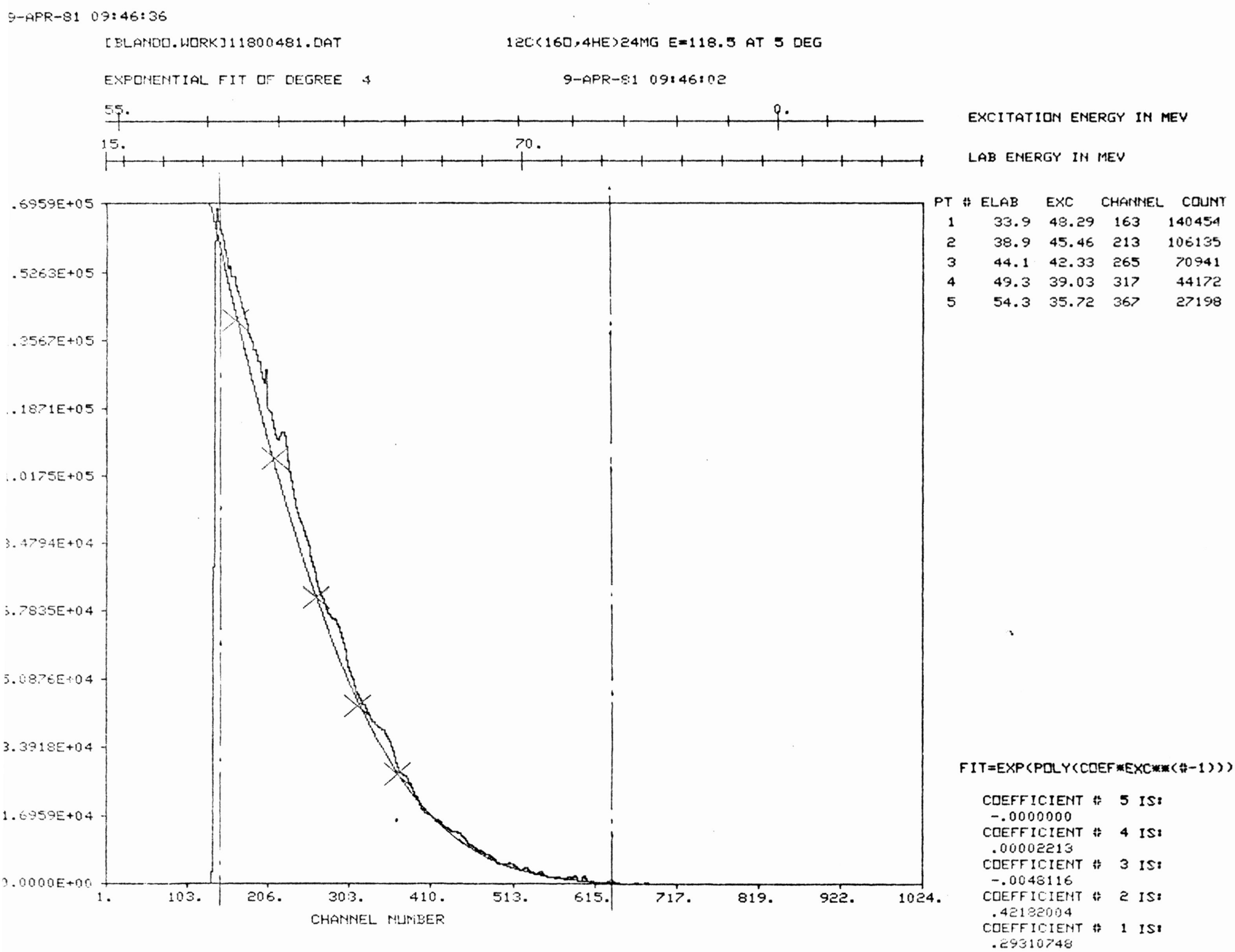


Figure 1

algorithm does not assign each point to a cluster, but it computes the coefficient of a polynomial which graph would separate each cluster. This algorithm is much less complex than the isodata algorithm, and it heavily relies on the inner product of two vectors (dot product). The algorithm starts with all the polynomial coefficients set to zero and then computes the dot product of the vector of coefficients and each individual point. When the dot product is less than or equal to zero, it updates the coefficients by adding the coefficients of the point to the polynomial coefficients. However, since only the equation of a polynomial separating the clusters is produced, this simple algorithm produces a much less useful output than the isodata algorithm. A sample input and the obtained results are shown in table 2. This algorithm can be used in two ways, either the program tries to isolate each cluster in one pass and therefore produces N+1 coefficients to isolate N clusters, or the program separates the data into two regions per pass and therefore requires N-1 passes to isolate N clusters(8).

MOTIVATIONS

After writing the isodata clustering algorithms, I decided to acquire a deeper understanding of clustering techniques and pattern recognition, so I wrote an implementation of the K-mean algorithm, the

Block-Nilsson-Duda algorithm for feature extraction. Then in January 1981 I started to work on a project involving clustering algorithms, feature extraction, and my work at the Cyclotron Institute on heavy ion nuclear physics; I started in September 1980 to work on an analysis program which reads arrays of data created during an experiment at the Cyclotron, and analyzes the features of the data such as peaks corresponding to certain elements at certain energies. The program runs on a VAX-11/780, is interactive, and heavily relies on graphics. The data is stored in vectors made of 1024 entries; then the program manipulates the data under user control and a subtracted spectrum is obtained(9).

The principal goal of the program manipulations is to identify elements by their mass and energy which is shown by the deviation undergone by the beam of particle when subjected to an intence magnetic field. Until now the program relied heavily on the operator, and therefore the results obtained were often different with different operators. My hope was to write a section of code which would perform the subtraction and identification in a less subjective manner. Details on the Cyclotron experiments and the data analysis follows.

THEORY

As in most experimental settings, the collected data obtained at the Cyclotron contains a lot of background noise. Background noise in this case is defined as the result of irrelevant registered events which can be modeled with a random function of the time and an exponential function of the beam energy variable. This background is created by residual particles left after vacuum, by high frequency interferences created by the electronic data acquisition equipment, and by other external unrelated events. A typical collected raw spectrum is shown in figure 1. The relevent parts of the spectrum are between the two dashed lines. In figure 1 is also included the exponential approximation of this background; the program produced this approximation. In figure 2, the background is subtracted and the relevant data shows as peaks. One of the reasons why the background needs to be subtracted is that counts generated by the background far outweigh the important data, and therefore, the peaks only slightly show. This is visible in figure 1 and 2, in figure 1 small notches are visible on the side of the curve, but they are much amplified in figure 2, and are then much more suitable for recognition by the human eye. Once the background approximation is subtracted each peak needs to be identified and correlated with a known particle.

Before proceeding with the code, a definition of a peak had to be developed. A peak is defined as a location where the data contained in the vector achieves a local maximum. The width of a peak is defined as the distance between the two extreme edges of the peak. By the extreme edge of a peak we mean the location in the array where the slope of an N point linear least square fit of the data is equal to zero and after a change of sign becomes greater than one in absolute value. The N points are chosen starting from the peak and then going toward the higher channels on one side and the lower channels on the other side (channel 1 is element 1 of the vector, channel 1024 is element 1024). Everytime the linear least square fit does not satisfy the check for a peak edge, the N points are shifted one channel further from the center; this process is repeated until a right edge and a left edge is found to every peak. The slope of the linear least square regression is obtained using the following formulas:

Given N points,

$$A = \sum_{I=1}^{N} I \qquad B = \sum_{I=1}^{N} I*I$$

$$C = \sum_{I=1}^{N} DATA(I) \qquad D = \sum_{I=1}^{N} (I * DATA(I))$$

Then the slope is given by:

$$Slope = ((N * D) - (A * C)) / ((N * B) - (B * A))$$

Which is obtained by Cramer's rule.

From the previous formula, it is reasonably clear that the location of the edges heavily depends on the number of points chosen for the least square fit. If we choose N to be small, then the local variation at one single point will have more weight than for a larger N; therefore, N can be adjusted depending on the type of peaks we want to identify. A small N will allow us to identify very small peaks; a large N will smooth out small peaks, and only large peaks will be found. An example of this is shown in figure 3 and figure 4; in figure 3, the background is not subtracted, but the N factor is set to 6. The program finds peaks that are hardly visible, but once the peak energy is computed, it is about the same as the energy level of the peak shown in figure 4 which was obtained with a N factor of 30 on a set of data where the background had been subtracted.

However, two problems were encountered. First, if two peaks are within N of one another, then the two peaks will be merged into a single peak. This is shown for peak 4 in figure 5, N was set to 30, the data has a peak at 353 and at 377, the limit between the two peaks is at 370; but the number of events at 370 is larger than at 384 so the two peaks are not separated by the program. Second, when the program tried to find a large number of peaks, some left-over data from the in-between peaks was isolated and displayed as peaks; figure 6 and 7 show this. In figure 6 the first six peaks found are shown; however, some data was left-over between peaks 1, 3, and 5. This left-over data was then singled out by the program as two legitimate peaks, one of these peaks is shown in figure 7. In order to remedy this, the program should not be asked to find too many peaks, or it should eliminate data left between close peaks.

Two methods for eliminating data between close peaks are possible. For the first method, the program should get a background approximation fit which is slightly larger than the between peaks data, thus once this fit is subtracted from the data only the peaks are left, and the vector values are zeros between peaks and therefore no extraneous peaks can be isolated there. Unfortunately, this method implies overestimating slightly the background and therefore could lead to a loss of data; the method would also only work if the data between peaks is a the fairly constant level, otherwise too much data will be truncated. So this method should be used only when we are interested in large peaks. The second method involves setting up a threshold variable and defining the minimum width of a peak. After identifying a peak, the program scans the vector and eliminates any chunk of data which is between two identified peaks less than the threshold value apart.

Since the program can identify peaks before or after the background approximation has been subtracted it is not important for its execution that this approximation be done, but to satisfy the operator a background fit should be made. Two techniques were studied to perform this fit: brute force, and random sampling. Brute force involves making a least square fit of the relevant points, about seven hundred points, and assuming that these points are an approximation of the exponential of a fourth degree polynomial. Once this approximation is done the approximation is lowered by the smallest quantity possible that will prevent that fit from intercepting the plot of the collected data, then this corrected fit is subtracted from the data. Even though this method provides a good approximation of the background, it is impractical to implement because doing a least square fit of seven hundred points involve summing up the values at these points and higher powers of the points. Since in a normal experiment counts of at least 200,000 events and more are common, integer arithmetics would be dangerous to use because the limit on a 32 bit signed integer is about 2,147,483,647. This limit could be reached, and an integer overflow would occur. Thus, real arithmetics would have to be used which would greatly reduce the speed and efficiency of the program, rendering it impractical with present computer technology.

The second method is a derivative of the previous one, but instead of using all the relevant points, a random selection of the points is made. Use of heuristics can however improve the quality of such random sampling and also reduce the number of points necessary for that least square fit. Thus, integer arithmetics is possible and the number of operations required from the computer is kept at a minimum. The heuristics used in picking points is extremly simple; the vector of data is divided into about 10 regions, then an optimal point is chosen from each region. The optimal point is the closest point to the center of the region which is not a local maximum or a local minimum. A point is a local maximum if its count of events is larger than both the one preceding and following it; a point is a local minimum if its count of event is lower than both the preceding and following point. These heuristics are derived from the assumption that the program will only have to process data with the characteristics shown in figure 1; the background can always be approximated by a fourth degree exponential fit, and the raw collected data has its highest count close to its lowest channel.

This program works in a manner similar to the human operator. The points for the least square fit of the background are chosen using a technique that simulates the operator's behavior; the points are chosen only in the relevant part of the spectrum, not-well-behaved points are not chosen for the fit, and only a minimum number of points are chosen. The peaks are defined in a manner that recreates the conditions on the plot as seen by the eye of the operator; a peak occurs when a local maximum is surrounded by a sharp (large linear least square slope) decrease in the number of count, and the edges of the peak are defined as where that sharp drop in event count stops (the slope passes thru zero and then changes sign). However, the program does not totally reproduce the human behavior, the program will always produce the same approximation of the background when presented the same spectrum; whereas a human operator will produce a slightly different approximation each time he is faced with the same spectrum. This is due to exterior factors which are much too complicated to understand and simulate using our computers. In any case the constant behavior of the program is one of its strong points because it reduces the subjectivity of the analysis.

RESEARCH TIME-TABLE

June 80 - August 80	Familiarization with the TAMU library, primary research on perceptron
Sept 80 - Dec 80	Artificial intelligence introductory course. Introduction to clustering algorithms and feature extraction techniques.
Nov 80 - Dec 80	Implementation of the isodata clustering algorithm, the K-mean clustering algorithm, and the Block-Nilson-Duda algorithm for feature extraction.
Dec 80 - Jan 81	Development of the specification and requirement of the spectrum analysis program be developed at the Cyclotron Institute. Work on the interface to the ANALYZE program to which this addition is to be made.
Feb 81 - March 81	Production of the code of the FORTRAN 77 implementation, and debugging of the routines using the VAX-11/780 debugging package. Identification of major flaws in the program logic, and development of remedies.

CONCLUSION

When in April 1980 I proposed to do some research of my own in artificial intelligence, my background in the subject was nonexistent. Therefore, my first task was to acquire some knowledge in the subject; most of my research time was spent taking classes in artificial intelligence and becoming familiar with terms used in that field. During the first part of the year, I focused on learning about artificial intelligence, no time at all was spent on a project; then the second part was spent developing a project which relates the theory to the every-day needs of scientific research in an area of Nuclear Physics.

During the program development, I realized how important it is to understand how a human thinks and deduces a course of action. It became evident that the most important part of writing the program was to breakup the process to be simulated into subtasks and subgoals, in the same manner as a human, until it becomes possible to program each subtask into the computer. The subtask break up should be done in a manner similar to a human doing the same task, and only the final small subtasks can be programmed in a manner differing from the human problem solving process.

The subroutines written at the Cyclotron Institute do not involve any elaborate artificial intelligence techniques, nor do they require a lot of advanced computer hardware. Furthermore, these routines demonstrate that artificial intelligence can be used in a common useful task; even though R2D2 of "Star Wars" fame is still a long time in the future, artificial intelligence can be applied now to improve program performances and understandability.

Bibliography

(1) Minsky, Marvin, "Perceptrons". MIT University Press, Cambridge, Ma. 1969

(2) Hunt, B. Earl, "Artificial Intelligence". Academic press in cognition and perception, N.Y. 1975

(3) J. T. Tou and R. C. Gonzalez, "Pattern recognition principle", Adisson-Wesley pub, Ma

(4) Bellman, Richard, "An introduction to artificial intelligence", Boyb & Fraser, San Fransisco. 1978

(5) Duda, R., & Hart, "Pattern recognitionand scene analysis", Wiley, N.Y. 1973

(6) Bundy, A., "Artificial intelligence, an introductory course", Edinburgh University Press. 1978

(7) Friesen, Dr D., Computer science 625, Texas A&M University, Fall 1980

(8) Becker, Peter W., "Recognition and pattern", 2nd edition, Spring-Verlay Wien, N.Y.

(9) Youngblood, Dave, research group. Texas A&M Cyclotron Institute. 1980

APPENDICES

APPENDIX A

```
      IMPLICIT CHARACTER*1(A),LOGICAL*1(E-F),INTEGER*2(B-D,G-P),
     X REAL*8(Q-Z)
      INTEGER*4 STEP_11,END_OF_LOOP,TOP_OF_LOOP
      CHARACTER*20 FILE_NAME
      INTEGER*2 DATA(0:10,255)
      REAL*8  CLUSTER(0:10,255),DISTANCE(10),STANDARD_DEV(0:10,255),
     X GAMMA,DISTANCES(3,255),LUMPING_PARA,OVER_ALL_AVG
      PARAMETER       ZERO              =0,
     X                FACTOR            =0.5D0,
     X                ONE               =1,
     X                TWO               =2,
     X                THREE             =3,
     X                SIX               =6,
     X                TEN               =10,
     X                TWO_FIFTY_FIVE    =255,
     X                MAXI_VALUE        =1D30
10    FORMAT (A)
      ASSIGN 1000 TO TOP_OF_LOOP
      ASSIGN 11000 TO STEP_11
      ASSIGN 14000 TO END_OF_LOOP
      WRITE (ONE,10) '1'
      WRITE (ONE,10) '$Do you want to enter a data file? '
      READ (ONE,10) ANSWER
      IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') THEN
          WRITE (ONE,10) '$Enter the spacial dimension '
          READ (ONE,*) DIMENSION
          WRITE (ONE,10) ' Just type ©Z to stop'
          COUNT=ONE
          EVER=.TRUE.
          DO WHILE (EVER)
              WRITE (ONE,*) 'Point #',COUNT,':'
              READ (ONE,*,END=9999) (DATA(LENGTH,COUNT),LENGTH=ONE,
     X            DIMENSION)
              COUNT=COUNT+ONE
          END DO
9999      COUNT=COUNT-ONE
          WRITE (ONE,10) '$Should the file be saved on disk? '
          READ (ONE,10) ANSWER
          IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') THEN
100           WRITE (ONE,10) '$Enter the file name '
              READ (ONE,10) FILE_NAME
              OPEN (UNIT=TWO,NAME=FILE_NAME,STATUS='NEW',CARRIAGE
     X            CONTROL='LIST',ERR=100)
              WRITE (TWO,*) DIMENSION,COUNT
              DO LENGTH=ONE,COUNT
                  WRITE (TWO,*) (DATA(COUNTER,LENGTH),COUNTER=ONE,
     X                DIMENSION)
              END DO
              CLOSE (UNIT=TWO,STATUS='KEEP')
          END IF
      ELSE
200       WRITE (ONE,10) '$Enter the name of the file to read '
          READ (ONE,10) FILE_NAME
          OPEN (UNIT=TWO,STATUS='OLD',NAME=FILE_NAME,ERR=200)
```

```
          READ (TWO,*) DIMENSION,COUNT
          DO COUNTER=ONE,COUNT
              READ (TWO,*) (DATA(LENGTH,COUNTER),LENGTH=ONE,
     X                  DIMENSION)
          END DO
      END IF
C
C             STEP 1
C
      NUMBER_CLUSTER=COUNT
      DO LENGTH=ONE,DIMENSION
          DO COUNTER=ONE,COUNT
              CLUSTER(LENGTH,COUNTER)=DATA(LENGTH,COUNTER)
          END DO
      END DO
1000  WRITE (ONE,10) '$Enter the approximate number of clusters '
      READ (ONE,*) NUMBER_K
      WRITE (ONE,10) '$Enter the minimun size of a cluster '
      READ (ONE,*) MINIMUM_SIZE
      WRITE (ONE,10) '$Enter the standard deviation parameter '
      READ (ONE,*) THETA_S
      WRITE (ONE,10) '$Enter the lumping parameter '
      READ (ONE,*) LUMPING_PARA
      WRITE (ONE,10) '$Enter the number of changes per pass '
      READ (ONE,*) NUMBER_L
      WRITE (ONE,10) '$Enter the maximun number of iteration '
      READ (ONE,*) MAXI_ITERATION
      DO ITERATION_COUNT=ONE,MAXI_ITERATION
C
C             STEP 2
C
          WRITE (ONE,*) 'Pass #',ITERATION_COUNT
          DO NUMBER=ONE,COUNT
              WORK=MAXI_VALUE
              DO COUNTER=ONE,NUMBER_CLUSTER
                  SUM=ZERO
                  DO LENGTH=ONE,DIMENSION
                      SUM=SUM+(DATA(LENGTH,NUMBER)-CLUSTER(LENGTH,
     X                    COUNTER))**TWO
                  END DO
                  IF (SUM.LT.WORK) THEN
                      WORK=SQRT(SUM)
                      DATA(ZERO,NUMBER)=COUNTER
                  END IF
              END DO
          END DO
C
C             STEP 3
C
          COUNTER=ONE
          DO WHILE (COUNTER.LE.NUMBER_CLUSTER)
              LENGTH=ZERO
              DO NUMBER=ONE,COUNT
                  IF (DATA(ZERO,NUMBER).EQ.COUNTER) LENGTH=LENGTH+ONE
```

```
            END DO
            IF (LENGTH.LT.MINIMUM_SIZE) THEN
                DO INCREMENT=ONE,COUNT
                    IF (DATA(ZERO,INCREMENT).GT.COUNTER) THEN
                        DATA(ZERO,INCREMENT)=DATA(ZERO,INCREMENT)-ONE
                    ELSE IF (DATA(ZERO,INCREMENT).EQ.COUNTER) THEN
                        WORK=MAXI_VALUE
                        DO LENGTH=ONE,NUMBER_CLUSTER
                            SUM=ZERO
                            DO LOOP_COUNT=ONE,DIMENSION
                                SUM=SUM+(DATA(LOOP_COUNT,INCREMENT)
     X                              -CLUSTER(LOOP_COUNT,LENGTH))**TWO
                            END DO
                            IF (SUM.LT.WORK) THEN
                                WORK=SUM
                                DATA(ZERO,INCREMENT)=LENGTH
                            END IF
                        END DO
                    END IF
                END DO
                DO INCREMENT=COUNTER+ONE,NUMBER_CLUSTER
                    DO LENGTH=ZERO,DIMENSION
                        CLUSTER(LENGTH,INCREMENT-ONE)=CLUSTER(LENGTH
     X                      ,INCREMENT)
                    END DO
                END DO
                NUMBER_CLUSTER=NUMBER_CLUSTER-ONE
            END IF
            COUNTER=COUNTER+ONE
        END DO
C
C           STEP 4
C
        DO COUNTER=ONE,NUMBER_CLUSTER
            DO LENGTH=ONE,DIMENSION
                DISTANCE(LENGTH)=ZERO
            END DO
            WORK=ZERO
            DO LENGTH=ONE,COUNT
                IF (DATA(ZERO,LENGTH).EQ.COUNTER) THEN
                    DO INCREMENT=ONE,DIMENSION
                        DISTANCE(INCREMENT)=DISTANCE(
     X                      INCREMENT)+DATA(INCREMENT,LENGTH)
                    END DO
                    WORK=WORK+ONE
                END IF
            END DO
            DO LENGTH=ONE,DIMENSION
                CLUSTER(LENGTH,COUNTER)=DISTANCE(LENGTH)/WORK
            END DO
        END DO
C
C           STEP 5
C
```

```
      DO COUNTER=ONE,NUMBER_CLUSTER
          SUM=ZERO
          CLUSTER(ZERO,COUNTER)=ZERO
          DO INCREMENT=ONE,COUNT
              IF (DATA(ZERO,INCREMENT).EQ.COUNTER) THEN
                  WORK=ZERO
                  DO LENGTH=ONE,DIMENSION
                      WORK=WORK+(DATA(LENGTH,INCREMENT)-CLUSTER(
     X                    LENGTH,COUNTER))**TWO
                  END DO
                  CLUSTER(ZERO,COUNTER)=CLUSTER(ZERO,COUNTER)+
     X               SQRT(WORK)
                  SUM=SUM+ONE
              END IF
          END DO
          CLUSTER(ZERO,COUNTER)=CLUSTER(ZERO,COUNTER)/SUM
      END DO
C
C         STEP 6
C
      OVER_ALL_AVG=ZERO
      DO COUNTER=ONE,NUMBER_CLUSTER
          OVER_ALL_AVG=OVER_ALL_AVG+CLUSTER(ZERO,COUNTER)
      END DO
      OVER_ALL_AVG=OVER_ALL_AVG/NUMBER_CLUSTER
C
C         STEP 7
C
      IF (ITERATION_COUNT.EQ.MAXI_ITERATION) THEN
          LUMPING_PARA=ZERO
          GO TO STEP 11
      ELSE IF (NUMBER_CLUSTER.LE.NUMBER_K/TWO) THEN
      ELSE IF (ITERATION_COUNT.EQ.ITERATION_COUNT/TWO*TWO.OR.
     X    NUMBER_CLUSTER.GE.TWO*NUMBER_K) THEN
              GO TO STEP 11
      END IF
C
C         STEP 8
C
      DO COUNTER=ONE,NUMBER_CLUSTER
          DO INCREMENT=ZERO,DIMENSION
              STANDARD_DEV(INCREMENT,COUNTER)=ZERO
          END DO
          SUM=ZERO
          DO INCREMENT=ONE,COUNT
              IF (DATA(ZERO,INCREMENT).EQ.COUNTER) THEN
                  DO LENGTH=ONE,DIMENSION
                      STANDARD_DEV(LENGTH,COUNTER)=STANDARD_DEV
     X                    (LENGTH,COUNTER)+(DATA(LENGTH,INCREMENT
     X                    )-CLUSTER(LENGTH,COUNTER))**TWO
                  END DO
                  SUM=SUM+ONE
              END IF
          END DO
```

```
                DO INCREMENT=ONE,DIMENSION
                    STANDARD_DEV(INCREMENT,COUNTER)=SQRT(STANDARD_DEV(
     X                  INCREMENT,COUNTER)/SUM)
                END DO
            END DO
C
C               STEP 9
C
            DO COUNTER=ONE,NUMBER_CLUSTER
                SUM=ZERO
                DO LENGTH=ONE,DIMENSION
                    IF (STANDARD_DEV(LENGTH,COUNTER).GT.SUM) THEN
                        SUM=STANDARD_DEV(LENGTH,COUNTER)
                        STANDARD_DEV(ZERO,COUNTER)=LENGTH
                    END IF
                END DO
            END DO
C
C               STEP 10
C
            FLAG_FOR_SPLIT=.FALSE.
            DO COUNTER=ONE,NUMBER_CLUSTER
                POINTER=STANDARD_DEV(ZERO,COUNTER)
                NUMBER=ZERO
                DO LENGTH=ONE,COUNT
                    IF (DATA(ZERO,LENGTH).EQ.COUNTER) NUMBER=NUMBER+ONE
                END DO
                IF (STANDARD_DEV(POINTER,COUNTER).GT.THETA_S.AND.
     X              (NUMBER_CLUSTER.LE.NUMBER_K/TWO.OR.(CLUSTER(ZERO,
     X              COUNTER).GT.OVER_ALL_AVG.AND.NUMBER.GT.TWO*(MINIMUM
     X              _SIZE+ONE)))) THEN
                        GAMMA=FACTOR*STANDARD_DEV(POINTER,COUNTER)
                        NUMBER_CLUSTER=NUMBER_CLUSTER+ONE
                        DO LENGTH=ZERO,DIMENSION
                            CLUSTER(LENGTH,NUMBER_CLUSTER)=CLUSTER(
     X                          LENGTH,COUNTER)
                        END DO
                        CLUSTER(POINTER,NUMBER_CLUSTER)=CLUSTER(POINTER,
     X                      NUMBER_CLUSTER)-GAMMA
                        CLUSTER(POINTER,COUNTER)=GAMMA+CLUSTER(POINTER,
     X                      COUNTER)
                        FLAG_FOR_SPLIT=.TRUE.
                END IF
            END DO
            IF (FLAG_FOR_SPLIT) GO TO END_OF_LOOP
C
C               STEP 11
C
11000       POINTER=ZERO
            DO COUNTER=ONE,TWO_FIFTY_FIVE
                DISTANCES(ONE,COUNTER)=MAXI_VALUE
            END DO
            DO COUNTER=ONE,NUMBER_CLUSTER-ONE
                DO LENGTH=COUNTER+ONE,NUMBER_CLUSTER
```

```
                SUM=ZERO
                DO INCREMENT=ONE,DIMENSION
                    SUM=SUM+(CLUSTER(INCREMENT,COUNTER)-CLUSTER(
     X                  INCREMENT,LENGTH))**TWO
                END DO
                IF (POINTER.LT.TWO_FIFTY_FIVE) THEN
                    POINTER=POINTER+ONE
                    DISTANCES(ONE,POINTER)=SQRT(SUM)
                    DISTANCES(TWO,POINTER)=COUNTER
                    DISTANCES(THREE,POINTER)=LENGTH
                ELSE
                    CALL MERGE (DISTANCES,SUM,COUNTER,LENGTH)
                END IF
            END DO
        END DO
C
C           STEP 12
C
        CALL SORT (DISTANCES)
        COUNTER=ONE
        DO WHILE (COUNTER.LE.POINTER)
            IF (DISTANCES(ONE,COUNTER).GE.LUMPING_PARA) THEN
                POINTER=COUNTER-ONE
            ELSE
                COUNTER=COUNTER+ONE
            END IF
        END DO
        POINTER=MIN(POINTER,NUMBER_L)
C
C           STEP 13
C
        DO COUNTER=ONE,POINTER
            IF (DISTANCES(TWO,COUNTER).NE.ZERO.AND.DISTANCES(THREE,
     X          COUNTER).NE.ZERO) THEN
                    INCREMENT=DISTANCES(TWO,COUNTER)
                    LENGTH=DISTANCES(THREE,COUNTER)
                    COUNT_1=ZERO
                    COUNT_2=ZERO
                    DO LOOP_COUNT=ONE,DIMENSION
                        DISTANCE(LOOP_COUNT)=ZERO
                    END DO
                    DO LOOP_COUNT=ONE,COUNT
                        IF (DATA(ZERO,LOOP_COUNT).EQ.INCREMENT) THEN
                            COUNT_1=COUNT_1+ONE
                            DATA(ZERO,LOOP_COUNT)=LENGTH
                            DO NUMBER=ONE,DIMENSION
                                DISTANCE(NUMBER)=DISTANCE(NUMBER)+
     X                              DATA(NUMBER,LOOP_COUNT)
                            END DO
                        ELSE IF (DATA(ZERO,LOOP_COUNT).EQ.LENGTH) THEN
                            COUNT_2=COUNT_2+ONE
                        END IF
                    END DO
                    DO LOOP_COUNT=ONE,POINTER
```

```
                            IF ((DISTANCES(TWO,LOOP_COUNT).EQ.INCREMENT
     X                          ).OR.(DISTANCES(TWO,LOOP_COUNT).EQ.LENGTH
     X                          )) THEN
                                    DISTANCES(TWO,LOOP_COUNT)=ZERO
                            ELSE IF ((DISTANCES(THREE,LOOP_COUNT).EQ.
     X                          LENGTH).OR.(DISTANCES(THREE,LOOP_COUNT)
     X                          .EQ.INCREMENT)) THEN
                                    DISTANCES(THREE,LOOP_COUNT)=ZERO
                            END IF
                        END DO
                        DO NUMBER=ONE,DIMENSION
                            CLUSTER(NUMBER,LENGTH)=((CLUSTER(NUMBER,
     X                          LENGTH)*COUNT_2)+DISTANCE(NUMBER))/(
     X                          COUNT_1+COUNT_2)
                        END DO
                        NUMBER_CLUSTER=NUMBER_CLUSTER-ONE
                        DO LOOP_COUNT=INCREMENT,NUMBER_CLUSTER
                            DO NUMBER=ZERO,DIMENSION
                                CLUSTER(NUMBER,LOOP_COUNT)=CLUSTER(
     X                              NUMBER,LOOP_COUNT+ONE)
                            END DO
                        END DO
                END IF
            END DO
C
C               STEP 14
C
14000       WRITE (ONE,*) NUMBER_CLUSTER,' distinct clusters now'
            WRITE (ONE,10) '$Do you want to see the clusters? '
            READ (ONE,10) ANSWER
            IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') THEN
                DO NUMBER=ONE,NUMBER_CLUSTER
                    WRITE (ONE,*) (CLUSTER(LENGTH,NUMBER),LENGTH=ZERO,
     X                  DIMENSION)
                END DO
            END IF
            WRITE (ONE,10) '$Do you want to see the points? '
            READ (ONE,10) ANSWER
            IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') THEN
                DO NUMBER=ONE,COUNT
                    WRITE (ONE,*) (DATA(LENGTH,NUMBER),LENGTH=ZERO,
     X                  DIMENSION)
                END DO
            END IF
            WRITE (ONE,10) '$Do you want to go back to the top? '
            READ (ONE,10) ANSWER
            IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') GO TO TOP_OF_LOOP
            WRITE (ONE,10) '$Do you want to stop? '
            READ (ONE,10) ANSWER
            IF (ANSWER.EQ.'Y'.OR.ANSWER.EQ.'y') STOP ' '
        END DO
        END
```

APPENDIX B

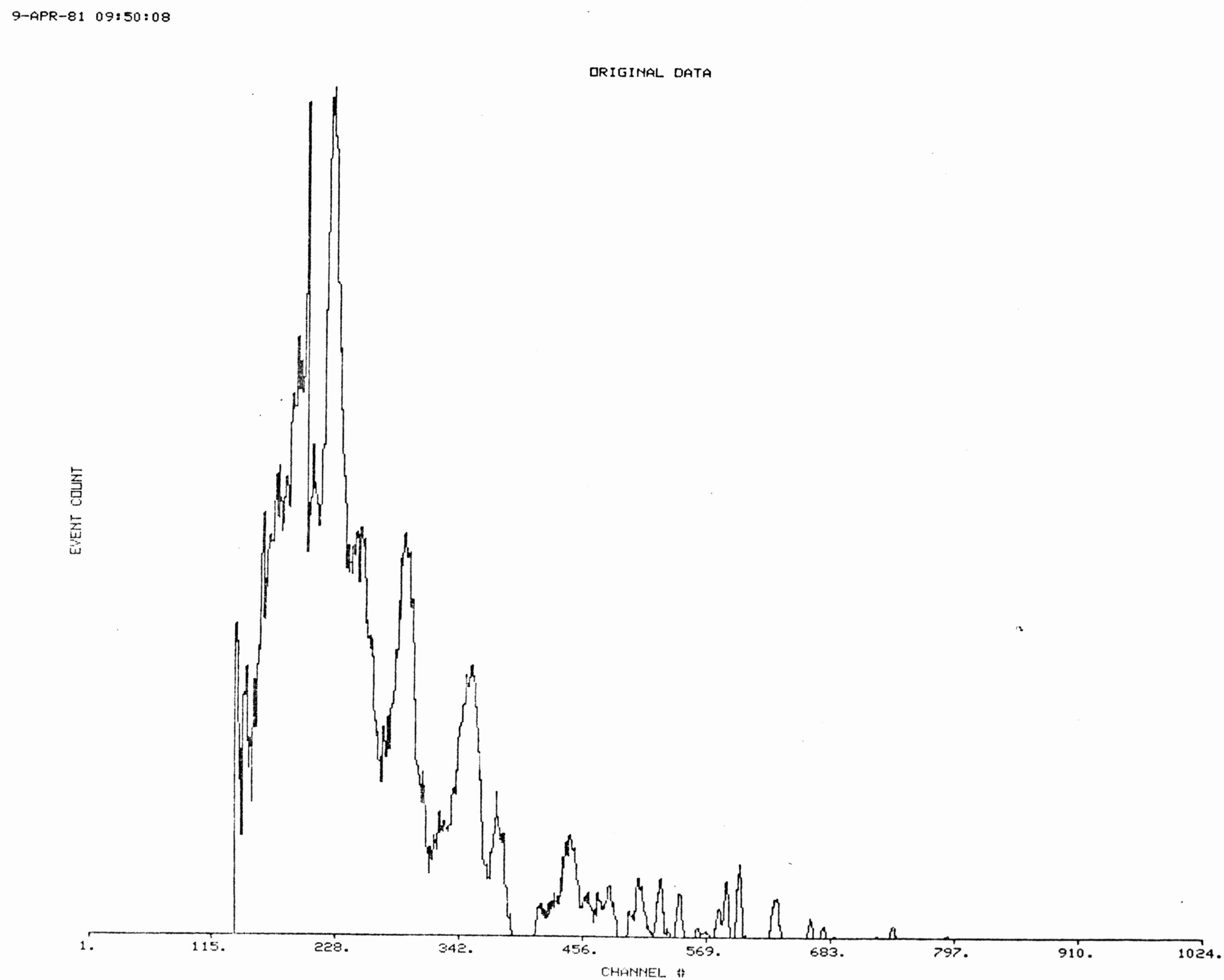

9-APR-81 09:50:08
ORIGINAL DATA
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 2

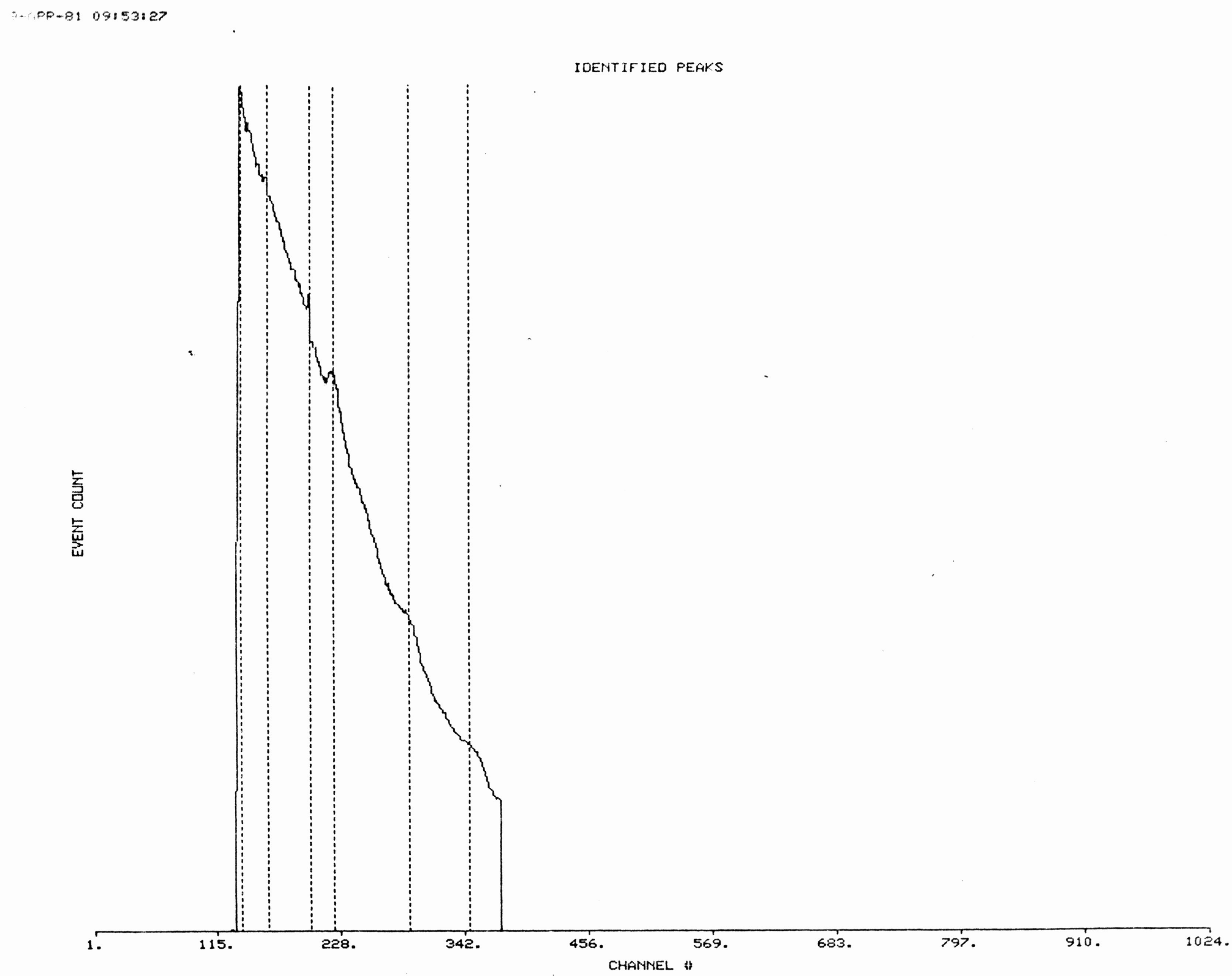

APR-81 09:53:27
IDENTIFIED PEAKS
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 3

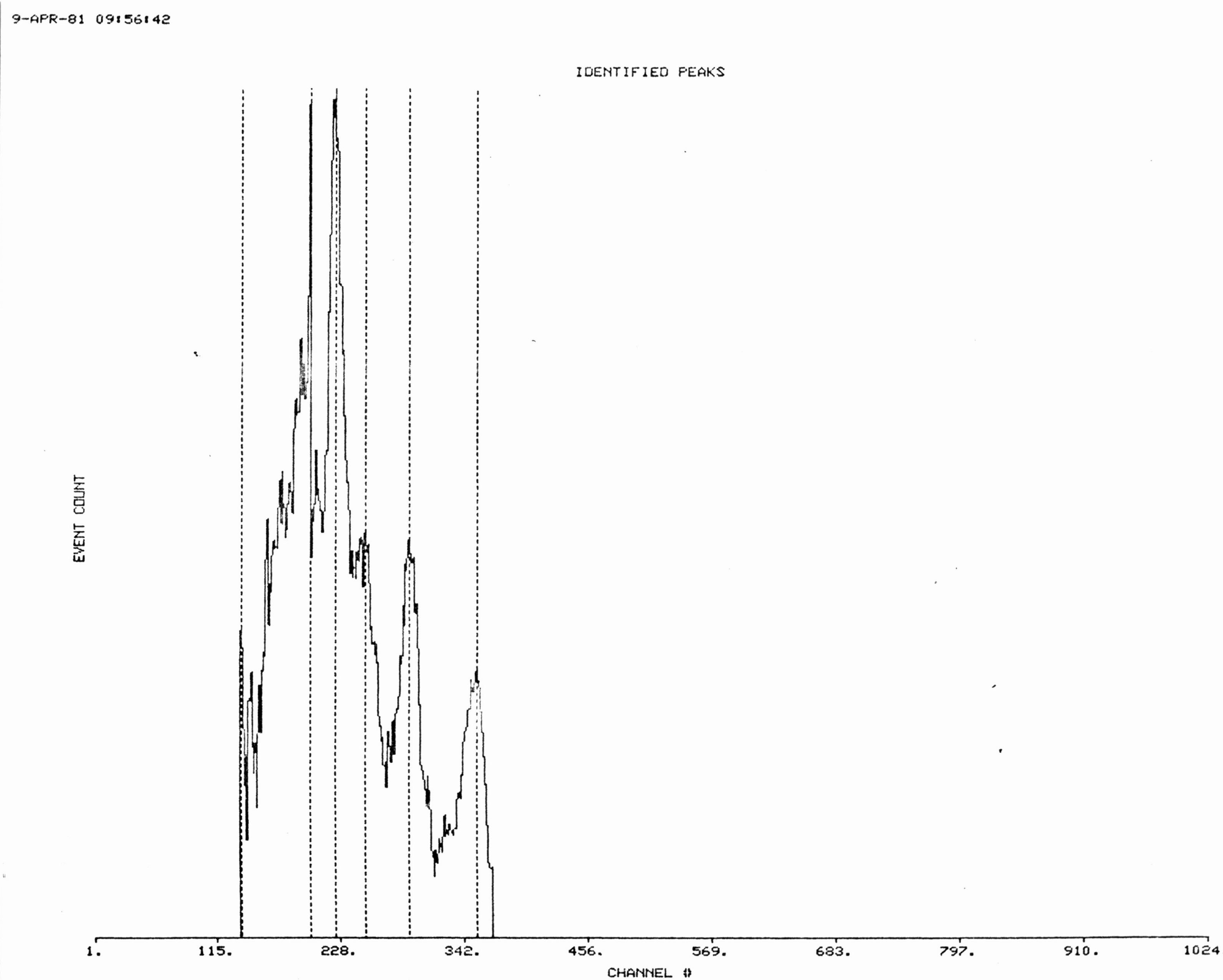
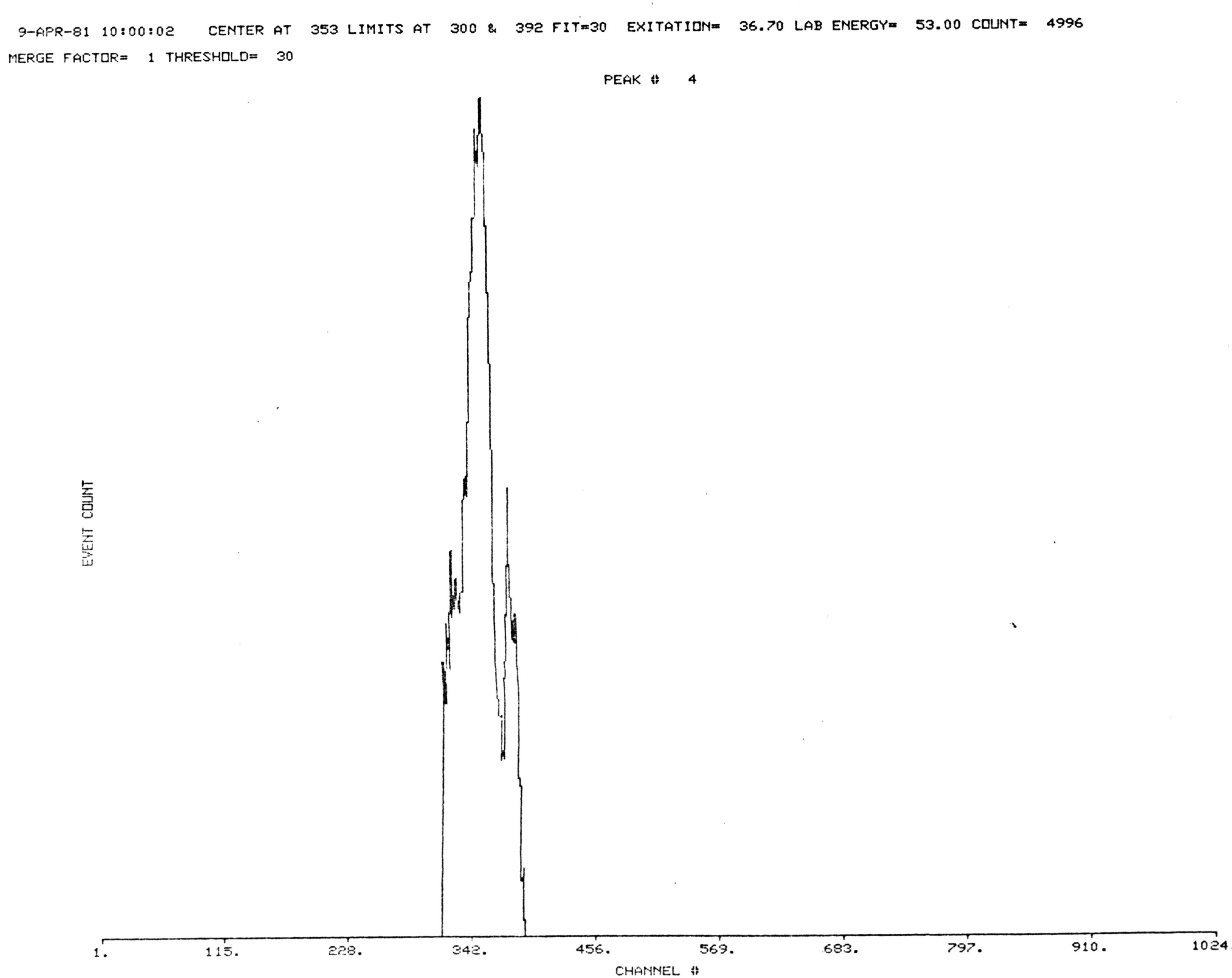
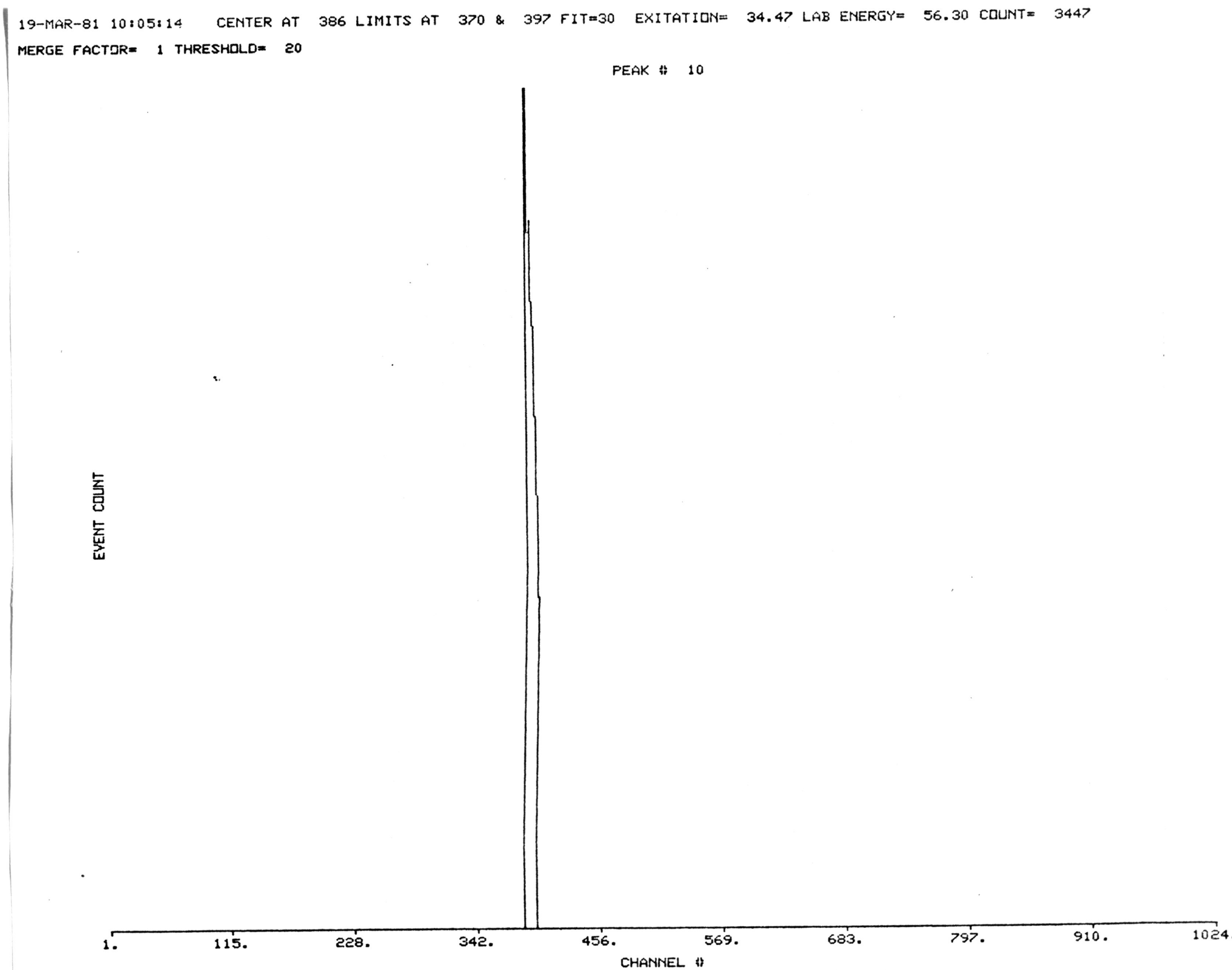

9-APR-81 09:56:42
IDENTIFIED PEAKS
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 4



9-APR-81 10:00:02 CENTER AT 353 LIMITS AT 300 & 392 FIT=30 EXITATION= 36.70 LAB ENERGY= 53.00 COUNT= 4996
MERGE FACTOR= 1 THRESHOLD= 30
PEAK # 4
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 5



19-MAR-81 10:05:14 CENTER AT 386 LIMITS AT 370 & 397 FIT=30 EXITATION= 34.47 LAB ENERGY= 56.30 COUNT= 3447
MERGE FACTOR= 1 THRESHOLD= 20
PEAK # 10
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 6

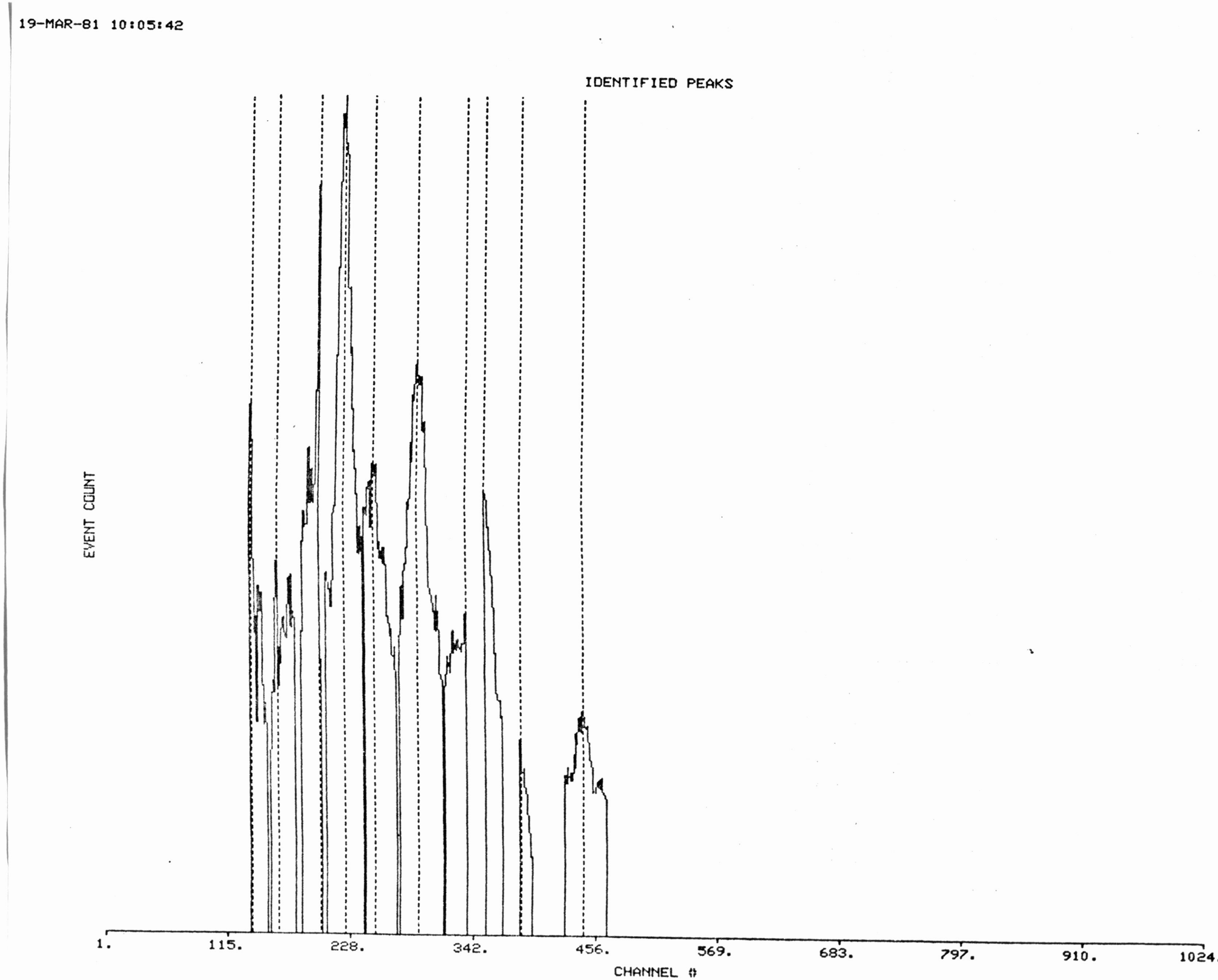

19-MAR-81 10:05:42
IDENTIFIED PEAKS
EVENT COUNT
1.
115.
228.
342.
456.
569.
683.
797.
910.
1024.
CHANNEL #

Figure 7

APPENDIX C

```
C
C       The following subroutines was developed by Frank J. Blando, on the
C       Cyclotron Institute VAX-11/780 computers.  It should be linked with
C       the program ANALYZE on the Ken Nagatani Group directory.
C
C       The subroutine AI is a peak finder subroutine.  The parameters passed
C       consist of an INTEGER*4 array of 1024 members and a run number.  The
C       subroutine will plot the array, and scan it for peaks.  The subroutine
C       requires that a CRT type terminal be assigned to the fortran unit 3.
C       From this CRT, the subroutine will ask for 3 parameters:
C               a) A fit paramter, telling the number of points that should
C                  be used for computing the extrapolation slope
C               b) A threshold value, when fewer points are left between 2 peak
C                  this data will not be considered for peak analysis.
C               c) A merge factor, the program will work with the average of
C                  this factor many points.  Used to smooth out sharp peaks.
C
        SUBROUTINE AI (DATA,RUN)
        INTEGER DATA(1024),GARBAGE(1024),PEAKS(1024),LEFT
     X          ,RIGHT,WIDTH,CENTER,COUNT,NUMBER,GET_MAXIMUM
     X          ,PEAK(1024),RUN,POINTER,MERGE,COMMON,THRESHOLD
        REAL*4 CENTERS(100)
        REAL*16 RIGHT_SLOPE,LEFT_SLOPE,FACTOR,SLOPE
        LOGICAL*1 YES,NO,WANT_MORE,INVALID,RESPONCE
        CHARACTER*50 ANSWER
        EXTERNAL LIB$EMULATE
        COMMON /WHERE/ POINTER
        PARAMETER                   ZERO=                   0,
     X                              ZEROS=                  0.,
     X                              ONE=                    1,
     X                              ONES=                   1.0Q0,
     X                              TWO=                    2,
     X                              DASHED=                 3,
     X                              CRT=                    3,
     X                              PLOTTER=                4,
     X                              PRINTER=                6,
     X                              TEN=                    10,
     X                              FIFTY=                  50,
     X                              NINETY_NINE=            99,
     X                              ONE_THIRTY=             130,
     X                              TOP=                    700,
     X                              SEVEN_THIRTY=           730,
     X                              SEVEN_FIFTY=            750,
     X                              TEN_TWENTY_FOUR=        1024,
     X                              LARGE_MAX=              +1.0Q4100,
     X                              LARGE_MIN=              -1.0Q4100
10      FORMAT (A)
20      FORMAT ('PEAK #',I4)
30      FORMAT ('CENTER AT ',I4,' LIMITS AT ',I4,' & ',I4,' FIT=',I2)
40      FORMAT ('MERGE FACTOR=',I3,' THRESHOLD=',I4)
        WRITE (CRT,10)'$How many points per fit? '
        READ (CRT,*) NUMBER
        WRITE (CRT,10) '$Enter the between peak threshold '
        READ (CRT,*) THRESHOLD
```

```
      CALL GET_ANSWER ('$Do you want to merge points? ',RESPONCE
     X        ,ANSWER)
      YES=.TRUE.
      NO=.FALSE.
      CALL LIB$ESTABLISH (LIB$EMULATE)
      DO I=ONE,TEN_TWENTY_FOUR
          PEAKS(I)=ZERO
          GARBAGE(I)=DATA(I)
      END DO
      IF (WANT_MORE(ANSWER)) THEN
140       WRITE (CRT,10) '$How many points at a time? '
          READ (CRT,*) MERGE
          IF  (MERGE.LT.TWO.OR.MERGE.GT.FIFTY) GO TO 140
          DO I=ONE,TEN_TWENTY_FOUR-MERGE,MERGE
              COMMON=ZERO
              DO J=ZERO,MERGE-ONE
                  COMMON=COMMON+GARBAGE(J+I)
              END DO
              COMMON=COMMON/MERGE
              DO J=ZERO,MERGE-ONE
                  GARBAGE(J+I)=COMMON
              END DO
          END DO
          COUNT=TEN_TWENTY_FOUR/MERGE*MERGE
          IF (COUNT.NE.TEN_TWENTY_FOUR) THEN
              COMMON=ZERO
              DO I=COUNT+ONE,TEN_TWENTY_FOUR
                  COMMON=GARBAGE(I)+COMMON
              END DO
              COMMON=COMMON/(TEN_TWENTY_FOUR-COUNT)
              DO I=COUNT+ONE,TEN_TWENTY_FOUR
                  GARBAGE(I)=COMMON
              END DO
          END IF
      ELSE
          MERGE=ONE
      END IF
      COUNT=ZERO
      CALL TKP_ERASE
      CALL GET_DISPLAY (GARBAGE)
      CALL TKP_TSEND
      FACTOR=QEXT(TEN_TWENTY_FOUR)/QEXT(GARBAGE(GET_MAXIMUM(GARBAGE)))
      POINTER=ZERO
      DO WHILE (WANT_MORE(ANSWER).AND.COUNT.LE.NINETY_NINE)
          COUNT=COUNT+ONE
          CENTER=GET_MAXIMUM (GARBAGE)
          RIGHT=CENTER+ONE
          RIGHT_SLOPE=LARGE_MIN
          WRITE (PRINTER,*) 'PEAK #',COUNT
          DO WHILE (RIGHT_SLOPE.LE.-ONES.AND.RIGHT.LE.TEN_TWENTY_FOUR)
              RIGHT_SLOPE=SLOPE(GARBAGE,RIGHT,YES,NUMBER)*FACTOR
              RIGHT=RIGHT+ONE
          END DO
          RIGHT=RIGHT+NUMBER/TWO
```

```
      IF (RIGHT.GT.TEN_TWENTY_FOUR) RIGHT=TEN_TWENTY_FOUR
      CALL GET_LIMIT (GARBAGE,RIGHT,NUMBER)
      LEFT=CENTER-ONE
      LEFT_SLOPE=LARGE_MIN
      DO WHILE (LEFT_SLOPE.LE.-ONES.AND.LEFT.GE.ONE)
          LEFT_SLOPE=SLOPE(GARBAGE,LEFT,NO,NUMBER)*FACTOR
          LEFT=LEFT-ONE
      END DO
      LEFT=LEFT-NUMBER/TWO
      IF (LEFT.LT.ONE) LEFT=ONE
      CALL GET_LIMIT (GARBAGE,LEFT,NUMBER)
      CENTERS(COUNT)=REAL(CENTER)
      CALL TKP_MOVEA (CENTERS(COUNT),ZEROS)
      CALL TKP_DASHR (ZEROS,REAL(GARBAGE(CENTER)),TWO)
      CALL TKP_TSEND
      CALL GET_ANSWER ('$Do you want an identification plot? ',
X         RESPONCE,ANSWER)
      IF (WANT_MORE(ANSWER)) THEN
          DO I=ONE,LEFT-ONE
              PEAK(I)=ZERO
          END DO
          DO I=LEFT,RIGHT
              PEAK(I)=GARBAGE(I)
          END DO
          DO I=RIGHT,TEN_TWENTY_FOUR
              PEAK(I)=ZERO
          END DO
          CALL PLOT (PEAK)
          ENCODE (TEN,20,ANSWER(ONE:TEN)) COUNT
          CALL PXP_TITLE ('CHANNEL #','EVENT COUNT',ANSWER(ONE:
X             TEN))
          ENCODE (FIFTY,30,ANSWER) CENTER,LEFT,RIGHT,NUMBER
          CALL PXP_LABEL (ANSWER,ONE_THIRTY,SEVEN_FIFTY,ONE)
          CALL GET_VALUE (ANSWER,CENTER,DATA,RUN)
          ENCODE (FIFTY,40,ANSWER) MERGE,THRESHOLD
          CALL PXP_LABEL (ANSWER,ZERO,SEVEN_THIRTY,ONE)
          CALL GET_DATE_TIME (PLOTTER)
      END IF
      DO I=LEFT,RIGHT
          PEAKS(I)=DATA(I)
          GARBAGE(I)=ZERO
      END DO
      CALL BETWEEN_CHECK (GARBAGE,RIGHT,LEFT,THRESHOLD)
      IF (COUNT.LE.NINETY_NINE) THEN
          CALL GET_ANSWER ('$Do you want to keep on going? ',
X         RESPONCE,ANSWER)
      END IF
  END DO
  CALL GET_ANSWER ('$Do you want a hard copy? ',RESPONCE,ANSWER)
  IF (WANT_MORE(ANSWER)) THEN
      CALL PLOT (DATA)
      CALL PXP_TITLE ('CHANNEL #','EVENT COUNT','ORIGINAL DATA')
      CALL GET_DATE_TIME (PLOTTER)
      CALL PLOT (PEAKS)
```

```
    DO I=ONE,COUNT
        CALL PXP_SCALE (CENTERS(I),ZEROS,IX,IY)
        CALL PXP_VECTOR (IX,IY,IX,TOP,DASHED)
    END DO
    CALL PXP_TITLE ('CHANNEL #','EVENT COUNT','IDENTIFIED PEAKS')
    CALL GET_DATE_TIME (PLOTTER)
    CALL PLOT (GARBAGE)
    CALL PXP_TITLE ('CHANNEL #','EVENT COUNT','DATA LEFT OVER')
    CALL GET_DATE_TIME (PLOTTER)
END IF
END
```

```
C
C       This function inputs an array of event counts and identifies 10 points
C       to be used in a quartic exponential approximation fit.  The subroutine
C       assumes that not all counts are zero in the array (ie the array has a
C       non-zero maximum point somewhere.  The coordinates of the points are
C       returned into the INTEGER arrays CENTER_x and CENTER_Y.
C
        SUBROUTINE SUBTRACT (ARRAY,RUN,CENTER_X,CENTER_Y)
        INTEGER ARRAY(1024),MAXIMUM_X,MAXIMUM_Y,GET_MAXIMUM
     X          ,COUNT,RUN,LEFT,RIGHT,MIDDLE,CENTER_X(40)
     X          ,CENTER_Y(40),END_POINT
        LOGICAL*1 MORE_TO_GO
        EQUIVALENCE (MORE_TO_GO,ANSWER)
        PARAMETER               ZERO=                   0,
     X                          ONE=                    1,
     X                          TWO=                    2,
     X                          CRT=                    3,
     X                          FIVE=                   5,
     X                          NINE=                   9,
     X                          TEN=                    10,
     X                          HUNDRED=                100,
     X                          TEN_TWENTY_FOUR=        1024
        MAXIMUM_X=GET_MAXIMUM(ARRAY)
        MAXIMUM_Y=ARRAY(MAXIMUM_X)
        FACTOR=.95
        END_POINT=TEN_TWENTY_FOUR
        COUNT=MAXIMUM_Y/HUNDRED
        DO WHILE (ARRAY(END_POINT).LT.COUNT)
            END_POINT=END_POINT-ONE
        END DO
        INCREMENT=(END_POINT-MAXIMUM_X-ONE)/NINE
        DO COUNT=ONE,NINE
            MIDDLE=MAXIMUM_X+INCREMENT*COUNT-INCREMENT/TWO
            IF (ARRAY(MIDDLE).GE.ARRAY(MIDDLE+ONE).AND.
     X          ARRAY(MIDDLE).LE.ARRAY(MIDDLE-ONE)) THEN
                CENTER_X(COUNT)=MIDDLE
                CENTER_Y(COUNT)=ARRAY(MIDDLE)*FACTOR+ONE
                CALL XXX (MIDDLE,CENTER_Y(COUNT))
            ELSE
                MORE_TO_GO=.TRUE.
                RIGHT=MIDDLE+ONE
                LEFT=MIDDLE-ONE
                DO WHILE (LEFT.GT.MAXIMUM_X.AND.RIGHT.LE.TEN_TWENTY_
     X              FOUR.AND.MORE_TO_GO)
                    IF (ARRAY(RIGHT).LE.ARRAY(RIGHT-ONE).AND.
     X                  ARRAY(RIGHT).GE.ARRAY(RIGHT+ONE)) THEN
                        MORE_TO_GO=.FALSE.
                        CENTER_X(COUNT)=RIGHT
                        CENTER_Y(COUNT)=ARRAY(RIGHT)*FACTOR+ONE
                        CALL XXX (RIGHT,CENTER_Y(COUNT))
                    ELSE IF (ARRAY(LEFT).LE.ARRAY(LEFT-ONE).AND.
     X                  ARRAY(LEFT).GE.ARRAY(LEFT+ONE)) THEN
                        MORE_TO_GO=.FALSE.
                        CENTER_X(COUNT)=LEFT
```

```
                CENTER_Y(COUNT)=ARRAY(LEFT)*FACTOR+ONE
                CALL XXX (LEFT,CENTER_Y(COUNT))
            END IF
            LEFT=LEFT-ONE
            RIGHT=RIGHT+ONE
        END DO
        IF (MORE_TO_GO) STOP 'Unable to use Heuristics!'
    END IF
END DO
END
```