To Prof. Supriyo Mohan Sengupta
With regards
Parthasarathi Ghosh

Computers & Geosciences Vol. 18, No. 9, pp. 1195-1211, 1992
Printed in Great Britain. All rights reserved

0098-3004/92 $5.00 + 0.00
Copyright © 1992 Pergamon Press Ltd

ROSE.C—A PROGRAM IN "C" FOR PRODUCING HIGH-QUALITY ROSE DIAGRAMS

T. S. KUTTY and PARTHASARATHI GHOSH
Geological Studies Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta-700 035, India

(Received 10 February 1992; accepted 13 April 1992)

Abstract—ROSE.C is a program written in "C" to draw Rose diagrams of high quality using 24-pin dot-matrix printers. It can handle both unidirectional and bidirectional (axial) data, and also compute and print various statistical parameters such as the direction of the resultant, the circular variance, consistency ratio, and the mean angular deviation. The diagrams produced can be scaled up to a maximum size of 9.3 cm. A number of options, including a selection of nine different patterns to shade the Rose diagram, have been provided. One of them allows the user to control the pattern. The program is designed to run up to 20 datafiles in a batch, each with its own options.

Key Words: Rose diagram, 24-pin dot-matrix printer, Directional data, "C" language.

INTRODUCTION

Scientists may be faced with the analysis of 2-dimensional directional data and their graphic representation. Usually, these types of data are represented by Rose diagrams. ROSE.C, a program for the PC, written in "C" language, is intended to meet the need of the general user in preparing high-quality outputs of Rose diagrams, without having to depend on expensive software. Some programs which use more sophisticated output devices such as plotters, or for use with larger computers are available (Parks, 1974; Charlesworth and others, 1989). ROSE.C, however, is meant for use on 24-pin dot-matrix printers. The diagrams produced by this program are scalable, and this facilitates their direct use in publications (see Figs. 1 and 2).

Usually, the necessity also arises to put a large number of Rose diagrams together in order to compare the patterns in different groups. To meet such needs there is an option to select from nine different patterns with which the diagrams can be shaded, or none.

Another need is to process large number of data sets quickly. ROSE.C can handle up to 20 files in a batch. User's options for each file are entered at the beginning, after which the files are processed one after another in quick succession.

The program also calculates some common statistical measures for each data set. These include the mean angular deviation of Batschelet (1981), the direction of the resultant (or the mean direction), the circular variance, and the consistency ratio.

PROGRAM STEPS

The following are the main steps of the program, and are explained in subsequent sections.

(i) The user is prompted to provide information about the data files and also select some options.
(ii) It initializes all relevant variables using the information provided.
(iii) Reads data from the file—ignoring dips if they are included—and forms an array of class frequencies.
(iv) Calculates the direction of the resultant vector and consistency ratio and sector radii.
(v) Prepares an image of the Rose diagram in the computer memory as per options specified (magnification and shading).
(vi) Sends the image to the printer for the output.

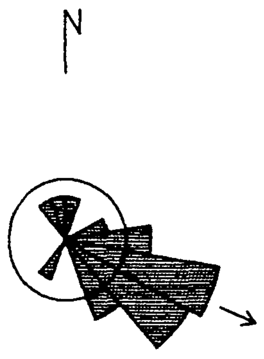
FILE INFORMATION AND USER OPTIONS

A data file for use with ROSE.C must contain only data entries; there is no provision for any comments within it. All data entries must be separated by at least one blank or carriage return; comma, semicolon, etc. should not be used as delimiters. The figures in this paper are based on the data file SAMPLE.DAT given here.

```
120.5  130.5  086    110   140.5  100   155
090    335    341.5  122   098    145   128
123    124.5  115    116   105    014   077
145    105.5  138    110   105.5  105   127
112    136    122.5  215   110    085   115.5
100    135.5
```

When the program is run there is an initial interactive session in which the user will be prompted to provide answers to a number of queries. These queries are listed next along with explanations thereof. The output of Figure 1 was produced by a single run of this program, and is used here to

```
File : sample.dat                    Radius of Circle =1 cms.
Classes =  18                        Observations =   37
Direction of Resultant = 113.8       Mean Angular Deviation = 25.8
Circular Variance =  0.20            Consistency = 79.7
```



```
File : sample.dat                    Radius of Circle =1 cms.
Classes =  18                        Observations =   37
Direction of Resultant = 118.8       Mean Angular Deviation = 22.9
Circular Variance (unadj.) =  0.32   Circular Variance (adj.)=  0.08
Consistency (unadjusted) = 68.0      Consistency (adjusted) = 92.0
```

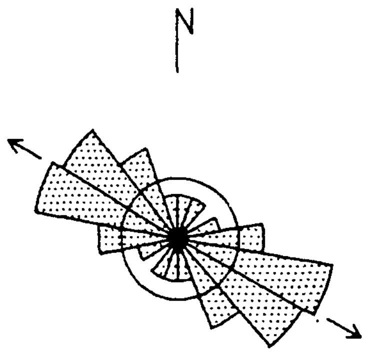


Figure 1. Rose diagram outputs of SAMPLE.DAT (i) as unidirectional data (top) and (ii) as bidirectional data (bottom).

illustrate this interactive session. To produce the output, SAMPLE.DAT was processed twice, first as a unidirectional data set (which it really is), and next as a bidirectional data set. The responses for the queries in running this program are indicated within parentheses at the end of each item. Quote marks do not form part of the response, and are used here merely for clarity.

(i) *Number of data files to be processed* (≤ 20): the program as given here can take up to 20 files in one run. The answer must be a number between 1 and 20. If 0 is given the program will be terminated. (In the example SAMPLE.DAT was processed twice; so, the response was "2".)

The program then prompts user for responses to queries (ii)–(vii) for each file, one file after another.

(ii) *Name of data files* (*include paths where needed*): the response is read as a string and stored in a character array of size 30. Path must be specified if the file is not in the current directory. ("sample.dat" for both files.)

(iii) *Whether the data have only directions or include dips and directions*: the response must be "y" or "n". If dips are included, the first number read in a record will be taken as dip and ignored, and the next will be read as direction. In such situations it therefore is necessary that, for each record, the dips must be the first and then the direction. Also a file should not have a mixture of the two types. (SAMPLE.DAT does not contain dip measurements, so the response was "n".)

(iv) *Whether the data are unidirectional or bidirectional*: response must be "b" or "u". (The dataset first was processed as unidirectional data, so the response for the first file was "u"; the second file was processed as bidirectional data, so the response was "b".)

(v) *The magnification desired*: the response must be a number. The upper limit for the magnification is limited by certain memory considerations explained later and also the nature of the data. Some onscreen hints are supplied. If a large magnification is given, the program will reduce it to the maximum possible value which it calculates at the time of execution. (For both files processed, the magnification used was 10, so that the radius of the circle will be exactly 10 mm.)

(vi) *Selection of pattern for shading the Rose diagram*: there are 10 options to select from, including no shading. Response must be a number. The patterns basically consist of rows of dots or lines, either horizontal or vertical. One option is a dot pattern where the user can specify the spacing between dots (horizontal) and between rows (vertical). [For the first file pattern No. 1 (closely spaced dots) was used, and for the second pattern No. 2 (spaced-out dots) was used.]

(vii) *Whether to draw the resultant direction* (*Y*/*N*): response must be "y" or "n". ("y" was given for both; if "n" was selected the arrow indicating the resultant vector will not be drawn.)

The following queries from (viii) to (x) are collectively for the batch, not individually for each file.

(viii) *Number of classes*: this specifies the number of classes into which the data are to be grouped. The response must be a number less than or equal to 20. (For the example the response given was "18".)

(ix) *Whether to print the statistics* (*Y*/*N*): response must be "y" or "n". If "y" the name of the data file, the radius of the unit circle (reflecting the magnification used) and the various statistical measures will all be printed above the diagram. If "n" these will not be printed. (The response given was "y".)

(x) *Printing to be in draft mode or final mode* (*D*/*F*): response must be "d" or "f". If "f" is pressed, each line will be printed twice so that the lines and dots are darker and clearer. (To produce the output in Fig. 1, "f" (final mode) was selected.)

The diagrams in Figure 2 also were produced by processing SAMPLE.DAT. They show the other ready-made shade patterns available. Note also that the diagrams are smaller than in Figure 1, and were obtained by selecting magnification 5 which gives a

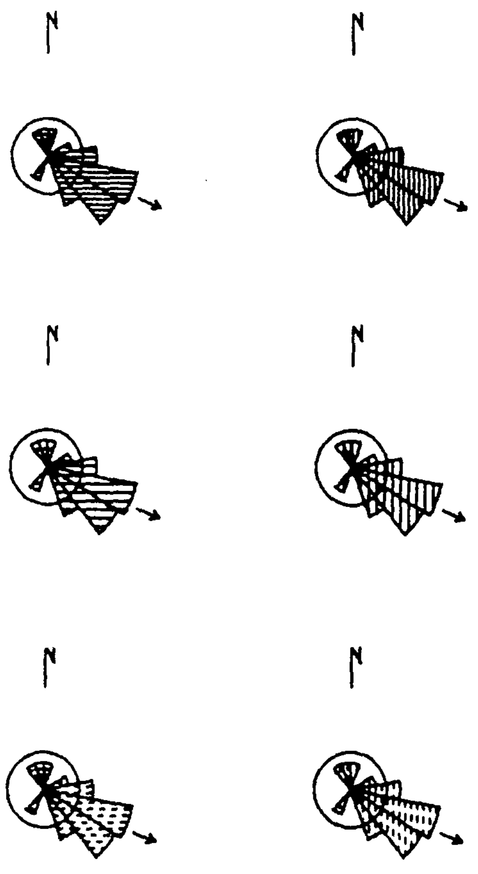


Figure 2. Sample outputs showing some of available shade patterns.

circle with radius 5 mm. In this example the response for query No. (ix), that is printing the statistics, was given as "n" to obtain only the diagrams.

CLASS FREQUENCIES, RESULTANT DIRECTION, AND CONSISTENCY RATIO

Unidirectional data (or simply, directional or vectorial data) can be represented by points lying on a circle; the values have a range from 0 to 360°. For axial or bidirectional data, the observations consist only of one end of the axes (the other can be obtained as reflection through the center); these observations therefore can be represented as a distribution of unidirectional vectors on a semicircle. The treatment of these two types of data are slightly different and are discussed separately.

Unidirectional data

Directional measurements may have values in the range of 0–360°, and the end values refer to the same vector. Directions are by convention measured from the North in a clockwise direction. If we take the E–W axis as the X-axis and the N–S axis as the Y-axis, then any vector OP_i, where O is the origin and P_i is a point with coordinates (x_i, y_i), will have components along X- and Y-axes equal to $\sin\theta_i$ and $\cos\theta_i$. The resultant will have a direction θ such that

$$\sin\theta = \Sigma \sin\theta_i ; \cos\theta = \Sigma \cos\theta_i .$$

We also must take care of boundary conditions, that is when the resultant is exactly along N, E, S, or W direction. The magnitude R of the resultant is given by

$$R = \sqrt{(\Sigma \sin\theta_i)^2 + (\Sigma \cos\theta_i)^2}.$$

If N = the total number of observations, then $R = N$ implies that all data points are in the same direction, and $R = 0$ implies a complete lack of preferred orientation. R/N, therefore is a measure of concentration, and lies between 0 and 1; the consistency ratio is the same measure expressed as percentage (Sengupta and Rao, 1966). $(1 - R/N)$ is known as the circular variance (Mardia, 1972). Another measure of dispersion is the "mean angular deviation" of Batschelet (1981). This is defined as $\sqrt{(1 - R/N)}$, and is measured in radians; it also can be converted and expressed in degrees. The class frequencies are calculated while reading the data (function Read_U and Read_B). The statistical measures are computed within Read_U.

Axial or bidirectional data

The usual way of handling axial data, which has a distribution on a semicircle, is by spreading it out onto a circle, a method developed by Krumbein (1939). This can be achieved by the transformation $\phi = 2\theta$ modulo 360, where θ and ϕ are corresponding angles on the semicircle and the circle respectively. These transformed data (ϕs) are unidirectional, and the vector mean, circular variance, etc. can be calculated as before. To get the resultant direction it is necessary to apply the reverse transformation, that is $\bar{\theta} = \bar{\phi}/2$. The circular variance also will need some adjustment. If V is the circular variance for the axial distribution and V' for the distribution spread onto the circle, then Mardia (1972) suggests the approximation $V = V'/4$; however he also adds that this adjustment usually is not used. For axial data, Batschelet's "mean angular deviation" is modified to $\sqrt{2(1 - R/N)}$, and can be converted and expressed in degrees. It must be noted that the statistical significance of these measures of dispersion for the axial case are somewhat ambiguous.

For the axial data the program first goes into function Read_B where the class frequencies are calculated. The data are read again in Read_U where it is spread first onto the full circle before computing the statistical measures.

The results of the calculations are printed out along with the name of the data file and the magnification used at the beginning before printing the Rose diagram. Figure 1 gives a sample output for unidirectional and bidirectional data sets.

SECTOR RADII

The radius of the sectors are calculated in such a way that the area of each sector is proportional to the frequency of the corresponding class, that is if r_i is the radius of the ith sector, f_i is the frequency of the corresponding class, C is the total number of classes, and N is the total number of observations, then $r_i = \sqrt{(f_i \times C)/N}$. These r_is are stored in the array **ra[ ]**. The sum of the area of all the sectors will then be equal to π, that is the area of a circle with unit radius.

SCALE AND MAGNIFICATION

It is useful to have some sort of scale for the diagrams for help in comparison. For this purpose, we have provided the given circle with unit radius. This circle would represent a hypothetical Rose diagram had the data points been distributed equally in all the sectors.

The Rose diagrams are printed with a resolution of 180 dots per inch (dpi). This value of 1/180th of an inch forms our unit of measurement. So, unless the diagram is magnified before drawing, it will be too small to make sense. A magnification factor therefore has been provided. Note that radius of the circle used for scale also is one unit; therefore the magnification factor also will be the radius of the circle; this is the variable **RC** in the program. The sector radii have to be magnified too; these magnified values are stored in the array **RA[ ]**.

THE MEMORY IMAGE OF THE ROSE DIAGRAM

Suppose that we have a graph paper where the spacing between lines is 1/180th of an inch. The graph divides the total area into a number of cells arranged in rows and columns, and we can refer to any cell uniquely by its row and column number. The arrangement is exactly as a two-dimensional array. Suppose that we superimpose a Rose diagram on this graph sheet. We then define the value of a cell to be 0 or 1 depending on whether any part of the Rose diagram passes through the cell. It is precisely such an array of values that we form in the memory and then pass on to the printer for printing.

In this program **Fig[ ][ ][ ]** is the array that keeps the memory image. It is here a three-dimensional array. This is purely for convenience in addressing cells and communication with the printer.

IMAGE FORMATION

Drawing the Rose diagram involves drawing the sector boundaries, which include an arc of a circle and two lines from the center to the ends of the arc, for each sector with nonzero frequency. Drawing them is done by plotting closely spaced points along them, close enough so that they look continuous. Not that the image formation is done after applying the desired magnification.

The printer moves from the left to right on a line and from the top line to the bottom. We therefore have used the top left position as the origin for our coordinate reference axes, x increasing to the right and y towards the bottom. The angles, however, are measured from the North (top) in a clockwise direction.

If the center of the Rose diagram is at (x_0,y_0) then any point at an angular distance ϕ from the North and at a distance "r" from the center will have coordinates $(x_0 + r \sin \phi, y_0 - r \cos \phi)$. The function **Enterarray()** locates the address of the cell of the image array corresponding to these coordinates and assigns the value 1 to the cell.

The functions **Line()** and **Circle()** determines the points on a line segment, arc, or a circle as specified in their arguments; the **Circle()** function is based on a fast algorithm making little use of trigonometric functions (Neal and Pitteway, 1990).

Filling the sectors with a pattern proceeds as follows. Along a line parallel to the X-axis points are selected at regular intervals (**stepx**) starting from the left boundary of the sector to its right boundary. This procedure is carried out from the top of the sector to its bottom in fixed steps (**stepy**). This procedure is followed for dot patterns and horizontal lines. For vertical lines points are plotted on a line parallel to the Y-axis at intervals **stepy** from the top to the bottom of the sector and this is done from the left of the sector to the right at intervals **stepx**. The function **Patternchoice()** fixes the values for these steps. There is one option which allows the user a selection of **stepx** and **stepy** for a dot pattern. The algorithm is rather long, mainly because of the large number of situations possible. Figure 2 illustrates some of the shading options.

The image also includes a North indicator and an arrow mark to indicate the resultant vector. These are done by the functions **North()** and **Arrow()** (see Figs 1 and 2).

PRINTING THE IMAGE

The function **Printarray()** sends the image of the Rose diagram to the printer with appropriate printer commands. There are mainly three such commands.

(i) ESC "U" 1 This sets the printer on unidirectional printing mode.
(ii) ESC "3" 24 This sets line spacing at 24 dot rows, at 1/180th of an inch per row.
(iii) ESC "*" 39,n1,n2 This puts the printer in triple density graphics mode and sets the horizontal resolution at 180 dpi. (ii) and (iii) ensure the correct aspect ratio. Here, n1 and n2 tells the printer how many bytes of information it is to print in graphics mode. The reader may refer to any printer manual for further details.

The commands are valid for the NEC P5300 24-pin printer. This printer is compatible with Epson LQ1500. For other 24-pin printers the user should consult the printer manual to ensure their validity.

COMMENTS

The program offers an output which, at present, is limited in size to 9.3 cm. If larger diagrams are needed, this limitation can be overcome by increasing the size of the image array **Fig[][][]**; however, this might involve some memory problems. Another significant limitation is the fact that it will not work with an 8-pin printer, in its present form. This will require some modifications of the function **Printarray()**; just changing the commands will not suffice.

It is easy to change the program to make it accept a larger number of files in a batch. All it needs is to change the size of the structure array **name[ ]**.

As it stands the maximum number of classes that the data can be grouped into is 20. To make this larger, although the need seems to be unlikely, all that is required is to change the definition of **SECTORS** to the required number.

A screen display has been omitted. If needed, this can be incorporated readily into the program logic as a separate function.

If the user wants to omit the Unit Circle from the drawing all that he has to do is to comment out the call to the function **Circle()** from the function **Rosedgm()**. The same also can be done for the North indicator.

REFERENCES

Batschelet, E., 1981, Circular statistics in biology: Academic Press, London, 371 p.

Charlesworth, H., Cruden, D., Ramsden, J., and Huang, Q., 1989, ORIENT: an interactive FORTRAN 77 program for processing orientations on a microcomputer: Computers & Geosciences, v. 15, no. 3, p. 275–293.

Krumbein, W. C., 1939, Preferred orientation of pebbles in sedimentary deposits: Jour. Geology, v. 47, no. 7, p. 673–706.

Mardia, K. V., 1972, Statistics of directional data: Academic Press, London, 357 p.

Neal, L. R., and Pitteway, M. L. V., 1990, Yet more circle generators: Computer Jour., v. 33, no. 5, p. 408–411.

Parks, J. M., 1974, Paleocurrent analysis of sedimentary crossbed data with graphic output using three integrated computer programs: Jour. Math. Geology, v. 6, no. 4, p. 353–362.

Sengupta, S., and Rao, J. S., 1966, Statistical analysis of cross-bedding azimuths from the Kamthi Formation around Bheemaram, Pranhita–Godavari Valley: Sankhya: Indian Jour. Stat., ser. B, v. 28, p. 165–174.

APPENDIX

Program Listing

```
/*  ROSE.C  -  A Program to draw Rose Diagrams  */
/*  Processes upto 20 datafiles in a batch */

#include <stdio.h>
#include <math.h>
#define XSIZE 648          /* fixes max. possible size of Rose diagram */
#define LINES XSIZE/24
#define SECTORS 21                                 /* No. of Classes + 1 */
#define RADIAN 0.017453                             /* degree to radian */
#define RADINV 57.29578                             /* radian to degree */
#define TRUE 1
#define FALSE 0
#define MMSTODOT 7.086614                /*  millimetres to dot resol. */
#define DOTTOCMS 0.0141111                         /* dot resol. to cms. */

/*  global variables */
/* ra[] - initial values of sector radii;
   RA[] - magnified sector radii;
   RL[] - sector boundaries tobe drawn;
   a[] - class limits;
   Fig[][][] - image array;
   RC - magnification (also radius of circle);
   resultant - direction of resultant;
   consistency - Consistency Ratio
   rmax - max sector radius (pre-magnification);
   RMAX - after magnification
   figsize - max radius of diagram
      (includes ARROW and RESULTANT indicators)
   x0,y0 - coordinates for the centre of diagram;
   Clas - No. of classes;
   N - No. of observations;
   count - No. of datafiles in the batch;
   draft - 0 for draft mode printing & 1 for final mode
   heading - 0 for not printing & 1 for printing
   maxmag - max. possible magnification
   var - circular variance
   angdev - mean angular deviation */

int      RL[SECTORS], a[SECTORS], count, draft, heading;
int      Clas, x0, y0, N, noofdots;
float    ra[SECTORS], RA[SECTORS];
float    RMAX, RC, resultant, maxmag, rmax;
FILE     *out;
unsigned char Fig[LINES][XSIZE][3];
float    figsize, consistency, var, angdev;
float    adjvar, adjcons;
typedef struct  { char nam[30];
                  float mag;
                  unsigned int dip:1;
                  unsigned int ub:1;
                  unsigned int rsltnt:1;
                  unsigned int pat:4;
                  unsigned int stepx:8;
                  unsigned int stepy:8;
                } NAME;
NAME     name[21], fil;          /* name[] - keeps names or options for */
                           /* each datafile; 'fil'.has the current file */
```

```
void Pause ()
{       puts("Press Any Key to Continue ......");
        getch ();
}                                         /* waits for user to press a key */
void quit ()
{       clrscr ();
        puts ("\n\n\nDo you want to stop the program ?   y / n");
        if ((tolower (getch ())) == 'y' )   exit ();
}

void Initialise ()
{       int      i, j, k;

        for ( i = 0; i <= LINES-1; ++i )
         for ( j = 0; j <= XSIZE-1; ++j )
          for ( k = 0; k <= 2; ++k )
            Fig[i][j][k] = '\0';
        for ( i = 0; i < SECTORS; ++i )
        { ra[i] = 0.0; RA[i] = 0.0; }
          rmax = 0.0; N = 0;
}

void Enterarray (x,y)           /* finds the cell in the image array */
                             /* corresponding to a point & makes it 1 */
unsigned int  x,y;
{       unsigned int Lin, Byte, Row, Bit;

        Lin = y /24; Row = y % 24; Byte = Row/8;
        Bit = 7 - (Row % 8);
        Fig[Lin][x][Byte] |= (1 << Bit);
}

void Line (X,Y,p,q,phi)
int     X,Y,p,q;
float   phi;
{       int      i,x,y;
        float    z1,z2;

        z1 = sin (phi);    z2 = cos (phi);
        for ( i = p; i <= q; ++i )
        { x = X + i * z1 +0.5;
          y = Y - i * z2 +0.5;
          Enterarray (x,y);
}       }

void Arrow (alpha)                                  /* draws an arrowhead */
int alpha;
{       int    i,x1,y1,p,q;
        float  phi;

        printf ("RESULTANT %d\n", alpha);
        phi = alpha * RADIAN;
        p = 13 * RMAX / 12 + 0.5;    q = 16 * RMAX / 12 + 0.5;
        Line (x0,y0,p,q,phi);
        x1 = x0 + q * sin(phi) + 0.5;
        y1 = y0 - q * cos(phi) + 0.5;
        phi = (alpha + 150) * RADIAN;
        q = RMAX / (6 * sqrt (3.)) + 0.5;
        Line (x1,y1,1,q,phi);
        phi = (alpha + 210) * RADIAN;
        Line (x1,y1,1,q,phi);
}

void North  ()                                 /* draws North indicator */
{       int    i,X,Y,x,y,q,k;
        float  b1=150,phi;

        puts("NORTH INDICATOR");
        for ( i = 13*RMAX/12; i <= 18*RMAX/12; ++i )
        {  y = y0 - i;     Enterarray (x0,y);  }
        phi = b1 * RADIAN;
        Y = y0 - 18 * RMAX / 12;
        q = RMAX / (3 * sqrt (3.));
        Line (x0,Y,1,q,phi);
        X = x0 + q * sin(phi);
        Y = Y - q * cos(phi);
        q = RMAX / 6;
        for ( i = 1; i <= q; ++i )
        {  y = Y - i;     Enterarray (X,y);  }
}

void Circle (X,Y,r)
int     X,Y;
float   r;
{       int    x,y;
        float  x1,y1,psi,c,s,w,f,d;

        f = 57.5 / r;     psi = f * RADIAN;
        c = 1 - psi*psi/2;
        s = psi * (1-psi*psi/8);
        printf ("\nUNIT CIRCLE\n");
        x1 = r; y1 = 0.0;
        for ( d = 0.0; d <= 90.0; d += f )
        {  w = y1;
           y1 = y1*c - x1*s;
           x1 = w*s + x1*c;
           x = X + x1;
           y = Y - y1;
           Enterarray (x,y);
           Enterarray (2*x0-x, y);
           Enterarray (x, 2*y0 - y);
           Enterarray (2*x0 - x, 2*y0 - y);
}       }

int Select_Quadrant (theta1,theta2)            /* finds the quadrant */
int  theta1,theta2;                              /* sector falls in  */
{       int   quadrant;

/*  1. 1st quad only ; 2.  1st & 2nd quad */
        if (theta1 < 90)  quadrant = (theta2 <= 90) ? 1 : 2;
/*  3. 2nd quad only ; 4.  2nd & 3rd quad */
        else if (theta1 < 180)
               quadrant = (theta2 <= 180) ? 3 : 4;
/*  5. 3rd quad only ; 6.  3rd & 4th quad */
        else if (theta1 < 270)
               quadrant = (theta2 <= 270) ? 5 : 6;
/*  7. otherwise 4th quad only      */
        else   quadrant = 7;
        return (quadrant);
}

void FillArcArea1 (r,y,t1,t2,q)          /* patternfill area bounded */
int     y,q;                                       /* by arc & line */
float   r,t1,t2;
{       int    k,x,x1a,x2a,i;
        float  v,z;

        z = y - y0;     v = fabs(r*r - z*z); v = sqrt(v);
        switch (q)
        {   case 1: x1a = x0 - z*t1; x2a = x0 + v;    break;
            case 3: x1a = x0 - z*t2; x2a = x0 + v;    break;
            case 4: x1a = x0 - v;    x2a = x0 + v;    break;
            case 5: x1a = x0 - v;    x2a = x0 - z*t1; break;
            case 7: x1a = x0 - v;    x2a = x0 - z*t2;
            default:    break;
        }
        for (k = x1a; k <= x2a; k += fil.stepx)
         for ( i = 1; i <= noofdots; ++i )
         { x = k + i;
           if (x <= x2a)      Enterarray (x,y);
}        }

void FillLineArea1 (y,t1,t2,q)                 /* pattern fill area */
int  y,q;                                  /* bounded by lines only */
float  t1,t2;
{       int    k,x,x11,x21,i;
        float  z;

        z = y - y0;
        switch (q)
        {   case 1: x11 = x0 - z*t1;  x21 = x0 - z*t2;  break;
            case 3:
            case 4:
            case 5: x11 = x0 - z*t2;  x21 = x0 - z*t1;  break;
            case 7: x11 = x0 - z*t1;  x21 = x0 - z*t2;
            default:     break;
        }
        for (k = x11; k <= x21; k += fil.stepx)
         for ( i = 1; i <= noofdots; ++i )
         { x = k + i;
           if (x <= x21)     Enterarray (x,y);
}        }

void Fill1 (r,theta1,theta2)          /* main function for horizontal */
int    theta1,theta2;                                     /* patterns */
float  r;
{       float  beta1, beta2, c1, c2, t1, t2;
        int    q, y1a, y2a, y11, y21, y;

        beta1 = theta1 * RADIAN; beta2 = theta2 * RADIAN;
        c1 = cos (beta1); c2 = cos (beta2);
        if (theta1 == 90) t1 = 99999.9;
        else t1 = (theta1 == 270) ? -99999.9 : tan(beta1);
        if (theta2 == 90) t2 = 99999.9;
        else t2 = (theta2 == 270) ? -99999.9 : tan(beta2);
        q = Select_Quadrant (theta1,theta2);
        switch (q)
        {     case 1: y1a = y0 - r * c1; y2a = y0 - r * c2;
                      for (y = y1a; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,1);
                      y21 = y0;
                      for ( ; y <= y21; y += fil.stepy)
                        FillLineArea1(y,t1,t2,1);  break;
              case 2: y1a = y0 - r*c1; y2a = y0;
                      for (y = y1a; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,1);
                      y2a = y0 - r * c2;
                      for ( ; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,3);  break;
              case 3: y11 = y0; y21 = y0 - r*c1;
                      for (y = y11; y <= y21; y += fil.stepy)
                        FillLineArea1(y,t1,t2,3);
                      y2a = y0 - r*c2;
                      for ( ; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,3);  break;
              case 4: y11 = y0; y21 = y0 - r * c1;
                      for ( y = y11; y <= y21; y += fil.stepy )
                        FillLineArea1(y,t1,t2,4);
                      y2a = y0 + r;
                      for ( ; y <= y2a; y += fil.stepy )
                        FillArcArea1(r,y,t1,t2,4);  break;
              case 5: y11 = y0; y21 = y0 - r*c2;
                      for (y = y11; y <= y21; y += fil.stepy)
                        FillLineArea1(y,t1,t2,5);
                      y2a = y0 - r*c1;
                      for ( ; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,5);  break;
              case 6: y1a = y0 - r*c2; y2a = y0;
                      for (y = y1a; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,7);
                      y2a = y0 - r*c1;
                      for ( ; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,5);
                      break;
              case 7:
                      y1a = y0 - r*c2; y2a = y0 - r*c1;
                      for (y = y1a; y <= y2a; y += fil.stepy)
                        FillArcArea1(r,y,t1,t2,7);
                      y21 = y0;
                      for ( ; y < y21; y += fil.stepy)
                        FillLineArea1(y,t1,t2,7);
                      break;
}       }

void FillArcArea2 (r,x,t1,t2,q)          /* patternfill area bounded */
int     x,q;                                         /* by arc & line */
float   r,t1,t2;
{       int     k, y, y1a, y2a, i;
        float   v,z;
        z = x - x0; v = fabs(r*r - z*z); v = sqrt(v);
        switch (q)
        {   case 1: y1a = y0 - v;      y2a = y0 - z/t2;  break;
            case 2: y1a = y0 - v;      y2a = y0 + v;     break;
            case 3: y1a = y0 - z/t1;   y2a = y0 + v;     break;
            case 4:   break;
            case 5: y1a = y0 - z/t2;   y2a = y0 + v;     break;
            case 6: y1a = y0 - v;      y2a = y0 + v;     break;
            case 7: y1a = y0 - v;      y2a = y0 - z/t1;
            default:  break;
        }
        for (k = y1a; k <= y2a; k += fil.stepy)
         for ( i = 1; i <= noofdots; ++i )
         {  y = k + i;
            if (y <= y2a)      Enterarray (x,y);
}        }

void FillLineArea2 (x,t1,t2,q)          /* pattern fill area bounded */
int     x,q;                                          /* by lines only */
float   t1,t2;
{       int     k, y, y11, y21, i;
        float   z;

        z = x - x0;
        switch (q)
        {     case 1:
              case 2:
              case 3:     y11 = y0 - z/t1;    y21 = y0 - z/t2;
              case 4:          break;
              case 5:
              case 6:
              case 7: y11 = y0 - z/t2;    y21 = y0 - z/t1;
              default:     break;
        }
        for (k = y11; k <= y21; k += fil.stepy)
         for ( i = 1; i <= noofdots; ++i )
         {  y = k + i;
            if (y <= y21)      Enterarray (x,y);
}        }

void Fill2 (r,theta1,theta2)           /* main function for vertical */
int     theta1,theta2;                                 /* patterns */
float   r;
{       float  beta1, beta2, eta, s1, s2, s3, t1, t2, t3;
        int    q, x1a, x2a, x11, x21, x, p;

        beta1 = theta1 * RADIAN; beta2 = theta2 * RADIAN;
        s1 = sin (beta1); s2 = sin (beta2);
        if (theta1 == 0 || theta1 == 180) t1 = 0.00001;
        else if(theta1 == 90 || theta1 == 270) t1 = 9.99999e+99;
        else t1 = tan (beta1);
        if (theta2 == 0 || theta2 == 180) t2 = 0.00001;
        else if(theta2 == 90 || theta2 == 270) t2 = 9.99999e+99;
        else t2 = tan (beta2);
        q = Select_Quadrant (theta1,theta2);
        switch (q)
        { case 1:   x11 = x0;      x21 = x0 + r*s1;
                    for (x = x11; x <= x21; x += fil.stepx)
                      FillLineArea2(x,t1,t2,1);
                    x2a = x0 + r*s2;
                    for ( ; x <= x2a; x += fil.stepx)
                      FillArcArea2(r,x,t1,t2,1);
                    break;
          case 2:   p = theta2 + theta1 - 180;
                    if ( p == 0 )
                    { x11 = x0;     x21 = x0 + r*s1;
                      for (x = x11; x <= x21; x += fil.stepx)
                        FillLineArea2(x,t1,t2,2);
                      x2a = x0 + r;
                      for ( ; x <= x2a; x += fil.stepx)
                        FillArcArea2(r,x,t1,t2,2);
                      break;
                    }
                    else if (p < 0)
                    { eta = (180.0 - theta2) * RADIAN;
                      s3 = sin(eta); t3 = tan(eta);
                      x11 = x0; x21 = x0 + r * s3;
                      for (x = x11; x <= x21; x += fil.stepx)
                        FillLineArea2(x,t3,t2,2);
```

```
                  x2a = x0 + r;
                  for ( ; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t3,t2,2);
                  x1l = x0; x2l = x0 + r*s1;
                  for (x = x1l; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t1,t3,1);
                  x2a = x0 + r*s3;
                  for ( ; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t1,t3,1);
                  break;
                }
                else                                /*  if (p > 0)   */
                { eta = (180 - theta1) * RADIAN;
                  s3 = sin(eta); t3 = tan(eta);
                  x1l = x0; x2l = x0 + r*s1;
                  for (x = x1l; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t1,t3,2);
                  x2a = x0 + r;
                  for ( ; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t1,t3,2);
                  x2l = x0 + r*s2;
                  for (x = x0; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t3,t2,3);
                  x2a = x0 + r*s3;
                  for ( ; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t3,t2,3);
                  break;
                }
     case 3:
                x1l = x0;      x2l = x0 + r*s2;
                for (x = x1l; x <= x2l; x += fil.stepx)
                  FillLineArea2(x,t1,t2,3);
                x2a = x0 + r*s1;
                for ( ; x <= x2a; x += fil.stepx)
                  FillArcArea2(r,x,t1,t2,3);
                break;
     case 4:
                x1a = x0 + (r*s2);      x2a = x0;
                for (x = x1a; x <= x2a; x += fil.stepx)
                  FillArcArea2(r,x,t1,t2,5);
                x2a = x0 + r * s1;
                for ( ; x <= x2a; x += fil.stepx )
                  FillArcArea2(r,x,t1,t2,3);
                break;
     case 5:
                x1a = x0 + (r*s2);   x2a = x0 + r*s1;
                for (x = x1a; x <= x2a; x += fil.stepx)
                  FillArcArea2(r,x,t1,t2,5);
                x2l = x0;
                for ( ; x <= x2l; x += fil.stepx)
                  FillLineArea2(x,t1,t2,5);
                break;
     case 6: p = theta2 + theta1 - 540;
                if ( p == 0 )
                { x1a = x0 - r;   x2a = x0 + (r*s2);
                  for (x = x1a; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t1,t2,6);
                  x2l = x0;
                  for ( ; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t1,t2,6);
                  break;
                }
                else if (p < 0)
                { eta = (540 - theta2) * RADIAN;
                  s3 = sin(eta); t3 = tan(eta);
                  x1a = x0 - r; x2a = x0 + (r*s3);
                  for (x = x1a; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t3,t2,6);
                  x1a = x; x2l = x0;
                  for (x = x1a; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t3,t2,6);
                  x2a = x0 + (r*s1);
                  for (x = x1a; x <= x2a; x += fil.stepx)
                    FillArcArea2(r,x,t1,t3,5);
                  x2l = x0;
                  for ( ; x <= x2l; x += fil.stepx)
                    FillLineArea2(x,t1,t3,5);
```

```
                        break;
                      }
                      else                                 /*  if (p > 0)  */
                      { eta = (540 - theta1) * RADIAN;
                        s3 = sin(eta); t3 = tan(eta);
                        x1a = x0 -r; x2a = x0 + (r*s3);
                        for (x = x1a; x <= x2a; x += fil.stepx)
                          FillArcArea2(r,x,t1,t3,6);
                        x1a = x; x21 = x0;
                        for ( ; x <= x21; x += fil.stepx)
                          FillLineArea2(x,t1,t3,6);
                        x2a = x0 + (r*s2);
                        for ( x = x1a; x <= x2a; x += fil.stepx)
                          FillArcArea2(r,x,t3,t2,7);
                        x21 = x0;
                        for ( ; x <= x21; x += fil.stepx)
                          FillLineArea2(x,t3,t2,7);
                        break;
                      }
          case 7:
                      x1a = x0 + (r*s1); x2a = x0 + (r*s2);
                      for (x = x1a; x <= x2a; x += fil.stepx)
                        FillArcArea2(r,x,t1,t2,7);
                      x21 = x0;
                      for ( ; x < x21; x += fil.stepx)
                        FillLineArea2(x,t1,t2,7);
                      break;
 }        }

 void Arc ()
 {        int    x,y,k;
          float  x1,y1,phi,psi,c,s,w,f,d,theta;

          theta = 360.0 / Clas;
          for ( k = 1; k <= Clas; ++k )
          {  phi = a[k-1] * RADIAN;
             if(RA[k] > 0)
             {  f = 57.5 / RA[k];
                psi = f * RADIAN;
                c = 1 - psi*psi/2;
                s = psi * (1-psi*psi/8);
                printf ("SECTOR  %3d - %3d\n",a[k-1],a[k]);
                y1 = RA[k] * cos(phi); x1 = RA[k] * sin(phi);
                for ( d = 0.0; d <= theta;  d += f )
                { w = y1;
                 y1 = y1*c - x1*s;
                 x1 = w*s + x1*c;
                 x = x0 + x1;
                 y = y0 - y1;
                 Enterarray (x,y);
            }  }
            if (RL[k] > 0)
               Line (x0,y0,1,RL[k],phi);
            if (RA[k] > 0.0)
            { if (fil.pat == 0);
              else if (fil.pat <= 6) Fill1(RA[k],a[k-1],a[k]);
              else      Fill2(RA[k],a[k-1],a[k]);
}         }  }

void Select_Fillpattern (pattern)        /* assigns values for stepx */
int      pattern;                                        /*  & stepy */
{        switch (pattern)
         {  case 1:                               /* Closely spaced dots */
                  fil.stepx = 4; fil.stepy = 4; noofdots = 2;
                  break;
            case 2:                                /* Med. spaced dots */
                  fil.stepx = 8; fil.stepy = 8; noofdots = 2;
                  break;
            case 3:                           /* Choose own dot spacing */
                  noofdots = 2;
                  break;
            case 4:                               /* Close horiz. lines */
                  fil.stepx = 1; fil.stepy = 5; noofdots = 1;
                  break;
            case 5:                         /* Spaced out horiz. lines */
                  fil.stepx = 1; fil.stepy = 8; noofdots = 1;
                  break;
```

```
        case 6:                               /* Broken horiz. lines */
                fil.stepx = 12; fil.stepy = 6; noofdots = 6;
                break;
        case 7:                               /* Close vertical lines */
                fil.stepx = 5; fil.stepy = 1; noofdots = 1;
                break;
        case 8:                               /* Spaced vertical lines */
                fil.stepx = 8; fil.stepy = 1; noofdots = 1;
                break;
        case 9:                               /* Broken vertical lines */
                fil.stepx = 6; fil.stepy = 12; noofdots = 6;
                break;
        default:                                   /* No Fill pattern */
                break;
}       }

void Magnify (m)                              /* scales ra[] into RA[] */
float  m;
{       int   i;
        figsize = 18 * m * rmax / 12. + 1;
        for ( i = 1; i <= Clas; ++i )     RA[i] = ra[i] * m;
        RA[0] = RA[Clas];     RMAX = m * rmax;
        for ( i = 1; i <= Clas; ++i )
           RL[i] = ( RA[i-1] > RA[i] )  ?  RA[i-1]  :  RA[i];
}

float Choose_Magnification ()             /* Option for magnification */
{       float  Mag;
        clrscr ();
        printf ("\n\nCHOOSE  A  MAGNIFICATION\n\n\
               \n Magnification = Radius of the Unit Circle\n\
               \n Max. Possible Size of diagram = 93 mms\n\
               \n Circle to diagram ratio depends on data\n\
               \n A CHOICE OF 10 mms IS USUALLY OK .\n\
               \n Too large a value will be automatically reduced\n\
               \n Give only the number ... \n\n\
               \nCHOICE OF RADIUS (in millimetres) =     ");
        scanf ("%f", &Mag);
        return (Mag * MMSTODOT);
}

void Setprntdefault ()
{
        fprintf (out, "%c%c",27, '\@');
        printf ("Printer has been set to default\n");
        fclose(out);
}

void Printarray ()            /* prints image array in graphics mode */
{       int    i,j,k,l;
        int    n1,n2,line1,line2,p;

        for (i = 0; i <= LINES-1; ++i )
         for ( j = 0; j <= XSIZE-1; ++j )
          for ( k = 0; k <= 2; ++k )
           if (Fig[i][j][k] ==10) Fig[i][j][k] = 11;
           else if (Fig[i][j][k] == 26) Fig[i][j][k] = 27;
        printf ("Printing Rose Diagram.  Wait ....  \n");
        k = x0 + figsize ;
        n2 = k / 256;     n1 = k % 256;
        line1 = (y0 - figsize) / 24;
        line2 = (y0 + figsize) / 24;
        if (line2 > 24) line2 = 24;
        fprintf (out, "%c%c%c",27,'U',1);
        for ( i = line1; i <= line2; ++ i )
        {   l = 0;
Final:      fprintf (out , "\t\t");
            fprintf (out,"%c%c%c%c%c",27,'*',39,n1,n2);
            for ( j = 0; j <= k-1; ++j )
            fprintf (out,"%c%c%c",
                 Fig[i][j][0],Fig[i][j][1],Fig[i][j][2] );
            if (draft)  { fprintf (out, "%c%c%c",27,'3',24);
                 fprintf (out,"\n");  }
            else  { if ((++l)==1)   fprintf (out, "%c%c%c",28,'3',0);
                    else fprintf (out, "%c%c%c",27,'3',24);
                    fprintf (out,"\n");
                    if (l<2) goto Final;  }
        }
        fprintf (out,"\n");
}

int PatternChoice(i)                                          /* Options */
int     i;
{       int    pattern, n1=0, n2=0;

        clrscr();
        printf ("\nSELECT FILLPATTERN \n\n");
        printf ("Enter choice of fill pattern: \
               \n\n\t\t\t0. NONE\n");
        printf ("\t\t\t1. Closely spaced DOTS\n");
        printf ("\t\t\t2. Med. spaced dots\n");
        printf ("\t\t\t3. Choose own dot spacing\n");
        printf ("\t\t\t4. Close HORIZ. LINES\n");
        printf ("\t\t\t5. Spaced out horiz.lines\n");
        printf ("\t\t\t6. Broken horiz. lines\n");
        printf ("\t\t\t7. Close VERTICAL LINES\n");
        printf ("\t\t\t8. Spaced vertical lines\n");
        printf ("\t\t\t9. Broken vertical lines\n\n      ");
        scanf ("%d", &pattern);
        if (pattern == 3)
        { printf ("\n Enter choice of spacing as integer values. \n\n");
          printf (" Note that units here are in 1/180 th of an inch\n\
                 \n 0 spacing is illegal\n\n");
          printf ("   Horizontzlly -  ");
          while (n1 == 0)     scanf ("%d", &n1);
          name[i].stepx = n1;
          printf ("   Vertically -  ");
          while (n2 == 0)     scanf ("%d", &n2);
          name[i].stepy = n2;
        }
        return (pattern);
}

void AdjustRadii()             /* finds sector from sector frequencies */
{       int    i;
        float  z;
        for (i = 1; i <= Clas; ++i )
        {  z = ra[i] * Clas / N;
           ra[i] = sqrt (z) ;
           if ( ra[i] > rmax )   rmax = ra[i];
        }
}

void Fileinfo ()                   /* Prompts for user for information */
{       int    i,j,bidir = 0;
        char   ch, c;
        FILE   *in;
        puts("Enter number of files to be processed ( <= 20 )");
Numb:   scanf ("%d", &count);
        if(count > 20 || count < 0)
        { printf("Number MUST be < 20 ...     "); goto Numb;  }
        else if (count == 0) exit();
        puts("Enter [path\] name of files :\n");
        for ( i = 1; i <= count; ++i )
        { printf(" File  %d.     ",i);
Name:     scanf("%s",name[i].nam);
          in = fopen (name[i].nam, "r");
          if(in == NULL)
          { printf ("\07");
            printf("File:  '%s'  could not be accessed\n", name[i].nam);
            puts("Please check spelling & path  and RETYPE ");
            goto Name;
          }
          fclose (in);
          puts("\n\nRecords include dip values ?     Y / N    ");
/*DIP*/   while((ch=tolower(getch())) != 'y' && ch != 'n');
          name[i].dip = (ch=='y'|| ch=='Y') ? TRUE : FALSE;
          puts("\n\nData Unidirectional / Bidirectional ?   U / B   ");
/*UB*/    while ((ch = tolower (getch())) != 'u' && ch != 'b');
          name[i].ub = (ch == 'u') ? FALSE : TRUE;
          if (name[i].ub ) bidir = 1;
          name[i].mag = Choose_Magnification();
          name[i].pat = PatternChoice(i);
          puts("\n\nPlot resultant vectors ?     Y / N ");
          while ((ch = tolower(getch ())) != 'y' && ch != 'n');
          name[i].rsltnt = (ch == 'y') ? 1 : 0;  clrscr();
        }
        puts("\n\nTHE FOLLOWING OPTIONS WILL BE SAME FOR ALL FILES");
        printf("\n\nNO. OF CLASSES: Must be EVEN if there is axial data\
             \n\nNumber of Classes  :-\n\n\n");
Sect:   scanf("%d",&Clas);
        if(Clas <= 1)
        { printf("\07");
          puts("Number of Classes must be > 1\n\n"); goto Sect;  }
        else if ((Clas % 2) && bidir)
        { printf ("\07");
          puts("Batch includes bidirectional data;\
             \nNumber of Classes MUST BE EVEN \n\n"); goto Sect;  }
        else if (Clas > SECTORS-1)
        { printf("No. of Classes must be < %d\n",SECTORS); goto Sect;  }
        puts("\nPrint Statistical Measures?   Y / N");
        while ((ch = tolower(getch())) != 'y' && ch != 'n');
        heading = (ch == 'y') ? 1 : 0;
        puts("\n\nPrint   Draft / Final mode ?    D / F\n\n\n");
        while ((ch = tolower(getch())) != 'd' && ch != 'f') ;
        draft = (ch == 'd') ? 1 : 0;
}

void Read_B (in)
FILE *in;
{       int    i, n, eof_flag=0;
        float  fdat;
        char   *format1={"%*f%f"}, *format2={"%f"}, *format;

/* Choose appropriate format for uni- or bidirectional data */
        if (fil.dip) format = format1;  else format = format2;
        for (i = 1; eof_flag != EOF; ++i)
        { eof_flag = fscanf(in,format, &fdat);
          if (eof_flag != EOF)
          {             /* Take that end of axis which is in 0 - 180 */
           if (fdat == 360.0) fdat = 0.0;
           else if (fdat >= 180.0) fdat -= 180.0;
           n = Clas / 2;
/* Make Class frequencies for 0 - 180 and duplicate for 180 - 360 */
           for ( i = 1; i <= n; ++i )
           if (fdat < a[i])
           {  ra[i] += 1;
              ra[i+n] += 1;
              break;
        } } }
        rewind (in);              /* Has to be read again in Read_U */
}

void Read_U (in)            /* Read routine for Unidirectional data */
FILE *in;
{       int    i, eof_flag=0, spread;
        float  fdat, phi, R, Rsq, S=0.0, C=0.0 ;
        char   *format1={"%*f%f"}, *format2={"%f"}, *format;

/* Choose appropriate format for uni- or bidirectional data */
        if (fil.dip) format = format1;  else format = format2;
        for (i = 1; eof_flag != EOF; ++i)
        { eof_flag = fscanf(in,format, &fdat);
          if (eof_flag != EOF)
          {  spread = (fil.ub) ? 2 : 1 ;
/* If axial data in 0 - 180 must be spread to 0 - 360 */
           fdat *= spread ;
           if (fdat == 360.0)     fdat = 0.0;
           else if (fdat > 360)  while (fdat > 360)  fdat -= 360;
           N += 1;
           for ( i = 1; i <= Clas; ++i )
/* For axial data class frequencies have already been calculated */
           if (fdat < a[i])
           { if (!fil.ub)     ra[i] += 1;
             phi = fdat * RADIAN;
             C += cos(phi);                      /* Sum of Cosines */
             S += sin(phi);                        /* Sum of Sines */
             break;
        } }  }
        AdjustRadii();
        R = sqrt (S*S + C*C) ;                /* Length of resultant */
        if (fabs(R) < 0.1e-2) { fil.rsltnt = 0; resultant = 0.0; }
        else                        /* Find direction of resultant */
        { resultant = asin (S/R) * RADINV;
          if (resultant == 0)
              resultant = (C > 0.0)  ? resultant     : 180;
          else if(resultant > 0.0)
              resultant = (C >= 0.0) ? resultant     : 180 - resultant;
          else resultant = (C >= 0.0) ? 360+resultant : 180 - resultant;
        }
        if (fil.ub)     resultant /= 2;        /* adjust for spreading */
        consistency = R/N;
        var = 1 - consistency;                   /* circular variance */
        adjvar = var / 4;                      /* adjust for spreading */
        if (fil.ub)  angdev = sqrt (2* adjvar) * RADINV;
        else         angdev = sqrt (var) * RADINV;    /* Mean ang.dev.*/
        consistency *= 100;                      /* Consistency Ratio */
        adjcons = 100 * (1-adjvar);            /* adjust for spreading */
}

void ReadData ()      /* main read routine; calls other read routines */
{       int    i,j;
        FILE   *in;
        char   c;
        puts("      Reading Data ......");
        in = fopen (fil.nam,"r");
        j = 360 / Clas;
        for (i = 0; i <= Clas; ++i)     a[i] = i * j;
        if (fil.ub)  { Read_B(in); Read_U(in); }   else  Read_U(in);
        fclose(in);
}

void Headings ()
{
        fprintf(out,"%c%c%c", 27, 120, 1);
        fprintf(out,"%c%c",27,77);                  /* 12 cpi printing */
        fprintf(out,"%c%c",27,48);                 /* 8 lines per inch */
        fprintf(out,"\n\tFile : %-29s", fil.nam);
        fprintf(out,"Radius of Circle =%.1f cms.\n", RC*DOTTOCMS);
        fprintf(out,"\tClasses = %3d%23c", Clas,' ');
        fprintf(out,"Observations = %4d\n",N);
        if (resultant)
        { fprintf(out,"\tDirection of Resultant = %5.1f%6c",
                      resultant,' ');
          if(fil.ub)
          {
            fprintf(out,"Mean Angular Deviation = %.1f\n", angdev);
            fprintf(out,"\tCircular Variance (unadj.) = %5.2f%2c",
                   var,' ');
            fprintf(out,"Circular Variance (adj.)= %5.2f\n",adjvar);
            fprintf(out,"\tConsistency (unadjusted) = %4.1f%5c",
                   consistency,' ');
            fprintf(out,"Consistency (adjusted) = %4.1f\n",adjcons);
          }
          else
          {
            fprintf(out,"Mean Angular Deviation = %.1f\n", angdev);
            fprintf(out,"\tCircular Variance = %5.2f%11c",var,' ');
            fprintf(out,"Consistency = %.1f\n",consistency );
        } }
        else  fprintf (out,"\n\tLENGTH of Resultant = 0.0\n");
        fprintf(out,"\n\n");
        fprintf(out,"%c%c%c", 27,120,0);
}

void Rosedgm ()                     /* Rose diagram controlling routine */
{       int i,j,k,m;
        char c;

start:  x0 =  y0 = XSIZE /2 ;
        out = fopen ( "LPT1", "w");
        Initialise ();
        ReadData ();
        maxmag = (XSIZE - x0 - 1)*12 / (18*rmax); /* Max.feasible mag.*/
        RC = fil.mag;
        if (RC > maxmag)
        { fprintf(out,"Magnification %.1f too high. Reducing to ",
               (RC/MMSTODOT) );
          printf("Magnification too high.");
          RC = maxmag;
          printf("New Magnification (Max.possible) = %5.1f\n",
               (RC/MMSTODOT) );
          fprintf(out,"%5.1f\n\n\n",RC);
        }
        Magnify (RC);
        if (heading)      Headings ();
        Select_Fillpattern (fil.pat);
        i = resultant+0.5; if(fil.ub) j = i+180;
        if (fil.rsltnt)  { Arrow (i); if (fil.ub) Arrow (j);  }
        North ();
        Arc();
        Circle (x0,y0,RC);
        Printarray ();
        Setprntdefault ();
}

void main ()
{       int    i,n;
        char   c;

begin:  clrscr();
        Fileinfo ();
        for ( i = 1; i <= count; ++ i )
        {  fil = name[i];
           clrscr();
           printf ("file being processed is  %s\n", fil.nam);
           Rosedgm ();
           fclose (out);
/*  Pressing ESC will stop processing of remaining files  */
           if (kbhit() && ((c = getch ()) != '\0' && c == 27)) quit ();
        }
        clrscr();
        puts("\n\n\nAny more files to process ?  y / n");
        c = getch ();
        if (c == 'y' || c == 'Y') goto begin;
}
```