Computers & Geosciences Vol. 17, No. 6, pp. 801–811, 1991
Printed in Great Britain

0098-3004/91 $3.00 + 0.00
Pergamon Press plc

SPLIT: A TURBO-C PROGRAM FOR THE GRAPHICAL REPRESENTATION AND SEPARATION OF FAULT-SLIP DATA SETS

DIETER KREJCI[1] and CARL RICHTER[2]
[1]Institut für Geologie, Universität Tübingen, D-7400 Tübingen, Germany and
[2]Department of Geological Sciences, University of Michigan, Ann Arbor, MI 48109, U.S.A.

(Received 9 January 1991; revised 20 February 1991)

Abstract—This paper presents a computer program (language: C) for IBM or compatible personal computers for the graphical representation and the manual separation of data consisting of a fault plane, the striation, and the relative sense of movement of the hanging block. Usually, brittle structures are the result of multiphase deformational processes in varying stress fields that are difficult to distinguish in the outcrop. With SPLIT the data are plotted on the computer screen using the representation method of R. Hoeppener and they can be separated manually into different populations using the cursor keys, which can be written subsequently into separate data files. The obtained files are suitable for an individual calculation and representation of the stress fields using other programs.

Key Words: Brittle tectonics, Paleostress analysis, Fault-slip data, Hoeppener method.

INTRODUCTION

The analysis of brittle data (fault plane, striation, and sense of movement) allows a determination of the principal directions of the paleostress field ($\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3$) which caused the faulting. Numerous algorithms and programs of the graphical representation and stress tensor calculation of these data exist (e.g. Angelier and Goguel, 1979; Etchecopar, Vasseur, and Daignieres, 1981; Lisle, 1988; Huang, 1989). Our approach is a straightforward graphical computer procedure that uses no complex mathematical analysis and that is suitable for a quick interactive treatment of raw data sets.

The analysis of paleostress uses measurements acquired from fault planes with both slickensides and sense of movement. The relationships between the state of stress and the movement on a surface are described by Arthaud (1969), Angelier and Mechler (1977), Angelier (1979, 1984), and Aleksandrowski (1985) among others. Individual fault planes and striations derived from the investigated outcrops usually are presented in equal area, lower hemisphere projections using the method of Angelier (1984) or the method of Hoeppener (1955). The latter combines the pole of the fault plane with an arrow that indicates the direction of movement of the hanging block. The Angelier method on the other hand combines the great circle of the fault plane with the direction of movement of the hanging block. However, when the number of individual great circles is large, the Angelier plots tend to be confusing. For this reason we prefer the method of Hoeppener giving a clearer diagram also for large data sets. Data measured in outcrop may not have formed in the same tectonic stress regime. The separation of different geometrical groups of data can be carried out using numerical separation methods of striae into different populations (Etchecopar, Vasseur, and Daignieres, 1981; Armijo, Carey, and Cisternas, 1982). In many applications, however, it is preferable to carry out interactive selection of the data before subsequent numerical analysis.

Our approach uses data sets consisting of the dip direction and dip of the fault plane, the azimuth and plunge of the striations, and the sense of movement of the hanging wall. These data are plotted into lower hemisphere, equal area stereograms using the method of Hoeppener (1955) and manually split into different populations using the computer screen. This approach is useful for deciphering complex fault systems and to determine possible variations in the orientation of the paleostress field. In addition, the program is useful for the rapid identification of erroneous data input or field measurements. The criteria for the selection, however, are subjective which has both advantages and disadvantages compared to numerical methods. It is emphasized that a meaningful selection can be carried out only if field criteria similar to overprinting relationships of striae are taken into consideration.

INPUT FILES AND PROGRAM ALGORITHM

First, the equations to determine the Hoeppener plot coordinates for the graphical representation are given followed by the routines for selection and separation of the data on the computer screen. The listing of the complete program is added in the Appendix.

Input text files (ASCII) contain the dip direction and the angle of dip of the fault planes and the corresponding direction and plunge of striations, e.g.

plane	striae	sense	symbol (optional)
133 86	45 16	+	1
157 57	86 24	–	1
22 87	101 17	0	3

The sense of movement in our convention is "–" for the relative down movement of the hanging block (not only normal faults, but also oblique and strike slip faults with a down component are included), "+" when relative up, and "0" if ambiguous relative to the lower block or if the sense was not detected in the field. The input file optionally may contain a sixth column with a number representing the plot symbol of the fault plane. This is added to identify individual data or groups of data on the computer screen. The definition of the symbols is omitted in our source code in the Appendix. However, the full code, obtainable on disk contains this feature. If other conventions or file formats are preferred, the routine "char readinput" in the source code (Appendix) should be modified.

GRAPHICAL REPRESENTATION

The coordinate system we use is defined by the origin (0,0) in the upper left-hand corner of the screen for a convenient operation with the Turbo-C graphics library. The y-axis points down and the x-axis to the right (Fig. 1).

Two steps are necessary to create a Hoeppener plot from the data: (1) determine the plot coordinates of the fault plane and (2) a more involved procedure, calculate the arrow. As an auxiliary construction this includes the calculation of the plane containing the normal of the fault plane and the striation (great circle) and the check of the conditions that make the arrow point up or down.

(1) The equations to obtain the x,y coordinates of the polar coordinates (ϕ,θ) of the normal of the fault plane in a lower hemisphere equal-area stereogram are (e.g. Wallbrecher, 1986):

$$\begin{aligned} x &= Mx + \sin[\phi * \sqrt{2} * R * \sin(\theta/2)] \\ y &= My - \cos[\phi * \sqrt{2} * R * \sin(\theta/2)] \end{aligned} \qquad (1)$$

where (Mx,My) is the center of the lower hemisphere projection plane in the reference frame and R the selected radius (Fig. 1).

(2) The sense of movement of the hanging wall is indicated by a line, or an arrow, on the great circle that contains the striation and the normal to the fault plane (Hoeppener, 1955). The head of the arrow (x',y' in Fig. 1) points into the relative direction of movement of the hanging block. In Figure 1, for example, the fault system is characterized by a pronounced horizontal dextral component and a slight upwards movement of the hanging block.

The plot coordinates of the auxiliary point (x',y') that marks the end of the hanging wall arrow are obtained from the following equations:

$$\begin{aligned} x' &= \cos(mpl + \text{dif}) * Rbc + xbc \\ y' &= \sin(mpl + \text{dif}) * Rbc + ybc \end{aligned} \qquad (2)$$

with the slope $mpl = (y - ybc)/(x - xbc)$ and with dif $= \pm 2\tan^{-1}(L/Rbc)$, where L is the length of the arrow, xbc and ybc the x,y coordinates of the great circle center, that is defined by the normal of the fault plane and the direction of the striation. Rbc is the radius of the great circle. The numerical determinations of the center and radius of great circles are not straightforward and because of its length are not described in this context (cf. Appendix: double

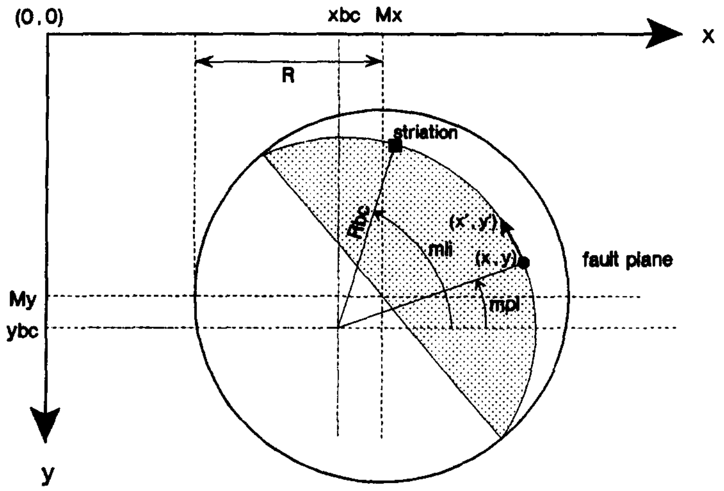

Figure 1. Principle of plotting brittle tectonic data sets into lower hemisphere equal-area stereogram. Center of projection is at (Mx,My). Radius of hemisphere is R; (xbc,ybc) is center of great circle with radius Rbc containing pole to fault plane (x,y) with striation. Great circle parameters are evaluated for determination of (x',y') which is auxiliary point marking end of arrow.

Table 1. Determination of sign of "dif": mpl and mli are slopes of pole of fault plane or striation referred to center of great circle (cf. Fig. 1)

	sense	conditions	dif
mpl < mli	+	mli - mpl ≥ π	+
	+	mli - mpl < π	-
	-	mli - mpl ≥ π	-
	-	mli - mpl < π	+
mpl > mli	+	mpl - mli ≥ π	-
	+	mpl - mli < π	+
	-	mpl - mli ≥ π	+
	-	mpl - mli < π	-

great circle coordinates; see Wallbrecher, 1986 for a deduction). The sign of the term "dif" is a function of both the sense of movement and the relative orientation of the fault plane to the striation on the projection net. The conditions that have to be checked to determine the sign of the term "dif" are summarized in Table 1.

The normal of the fault plane (x,y) together with the connecting line between (x,y) and (x',y') is an unambiguous representation of the total slip system. If the slip direction of the hanging block is vague, that is if no field data about the direction of movement of the hanging block are obtainable, the slip system is represented by the pole of the fault plane connecting the auxiliary point (x',y') calculated with a positive sign of the term dif with a second auxiliary point $(x1',y1')$ calculated with a negative sign of dif. This results in a short line following the trace of the great circle on both sides of the pole.

GRAPHICAL IDENTIFICATION AND SEPARATION OF DATA ON THE COMPUTER SCREEN

The calculated (x,y) plot coordinates of the fault plane are stored in an array and used to mark the pole of the fault plane on the computer screen with an arrow. The arrow is determined by the line connecting the point (x,y) with $(x - A,y)$, where A is the length of the arrow. Using the left and right arrow keys of the computer's cursor block, the arrow on the computer screen can be moved to the next or the previous fault-slip set indicated. The continuous number of the marked fault-slip set in the selected file is provided in the upper left of the screen and can be used to identify individual data of the total array. It is for example, possible to determine where the representative point of set number 7 is, to compare its position with field observations, and, if necessary, to remove the set from the data file.

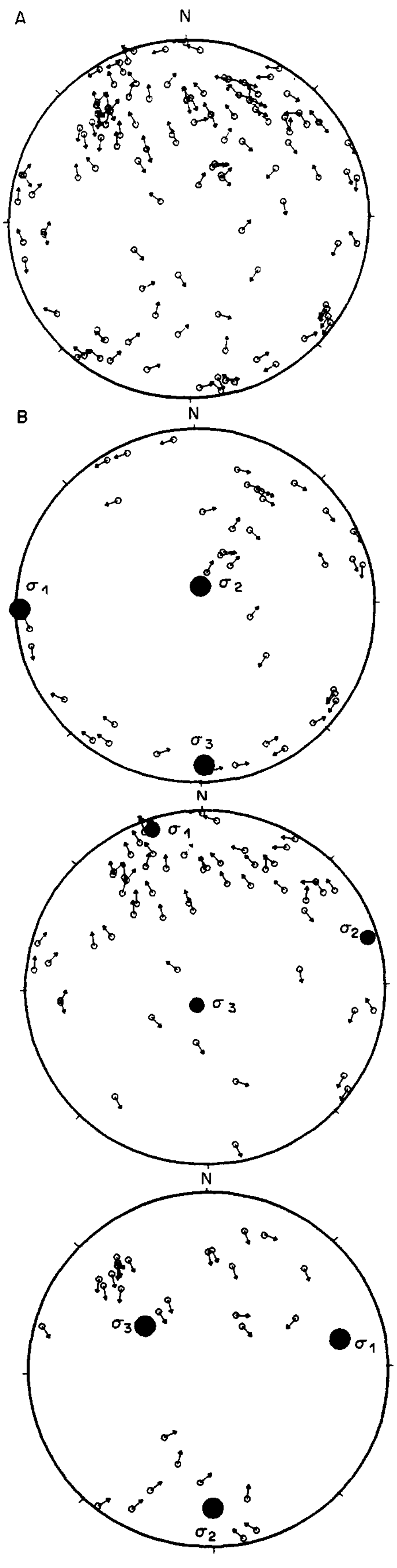

Figure 2. A—Hoeppener plot of raw fault-striation data from Villnöß line, Southern Alps; B—results of manual separation of raw data into different groups based on field relationships. Principal stress axes were calculated separately for each group.

To select a data set to be separated into a new data file the "<" key has to be pressed before entering a number (1, 2, 3, or 4) to symbolize the name of the selected file. After leaving the program the user is asked to enter filenames instead of the digits (up to four different files are possible) and the data sets are written into the selected files.

AN EXAMPLE FOR THE APPLICATION OF SPLIT

We developed SPLIT during the evaluation of brittle data sets from major faults of the Southalpine unit of the Eastern Alps (Ring and Richter, 1992). This example is on data obtained from the Villnöß line (Villnöß Valley, Südtirol, N. Italy). The Neogene-aged Villnöß line represents the northern backthrust of the Dolomite pop-up structure described by Doglioni (1987). The fault is a major N-directed reverse fault, with total vertical displacement on the order of 1000 m.

The raw data from the Villnöß line are presented in Figure 2A.The slickensides and striae are the result of polyphase brittle tectonic activities with different stress field orientations. The determination of the paleostress directions from the data requires that a relative age be assigned to each measured slickenside-striation set. In practice this is rarely possible for each measurement. Most of the data lack proper age control. Consequently, they had to be separated using field relationships to identify correctly paleostress fields. Overprinting criteria of striae overgrowth allowed us to distinguish between three events at some places in the outcrop:

(1) A dextral strike-slip component,
(2) An about N–S oriented reverse fault component,
(3) An oblique or normal faulting.

The raw data were combined into different populations according to these criteria. The results of our manual separation with SPLIT are shown in Figure 2B. Using an inverse method described by Angelier and Mechler (1977) and Angelier (1979) we calculated for each group separately the principal stress directions (Fig. 2B). The example shows strike slip movements with a marked reverse fault component. A normal fault component with a vertical σ_1, which was observed usually in other outcrops, is represented by an oblique slip faulting. Faulting was initiated during a pre-early Miocene (late Cretaceous–Paleogene?) approx. E–W directed event (Doglioni, 1985); however, later movements reactivated the existing fault system (for a full discussion of the stress history see Ring and Richter, 1992). This method allows a more reasonable estimate of paleostress determination than does a purely numerical approach because the cleaning of the data is based on field relationships.

SOFTWARE AND HARDWARE REQUIREMENTS

The program is written in the computer language C. We use the Borland Turbo-C compiler (Version 2.0). The source code of the graphic routines is Turbo-C specific and cannot be compiled with any other C-compiler without changing the graphic routines. However, modifications adjusted to individual requirements, for example special input file arrangements, may easily be realized in the source code. The program requires an IBM (or compatible) XT, AT 286, or AT 386 with a monochrome or color graphics adapter (Hercules, EGA, or VGA).

SUMMARY

The problem of the separation of fault slip data into different populations is discussed in this paper, which may represent different episodes of deformation or changes in the stress regime. The data are derived from field observations, and the program provides a graphical interface for the display and manipulation of the data.

(1) SPLIT is an interactive program to display and select brittle tectonic data sets as used in paleostress investigations.
(2) Data are presented graphically using the method of Hoeppener (1955) and combined into different files using the cursor keys. In addition, a simple identification of an individual data set is possible on the screen with the help of the number of the set in the data-array. The use of different plot symbols for single or groups of data make their rapid identification on the computer screen possible. This approach allows to get a quick overview of the data, their manipulation and elimination of input or measurement errors.
(3) In the program source code the individual required input and output file formats can easily be modified.

AVAILABILITY

The source code and an executable version of the program together with an example, are available on a 5.25 or 3.5 in. disk from one of the authors. SPLIT can be obtained by submitting a check for $5.00, to cover costs of duplication and mailing.

Acknowledgments—We thank Ben A. van der Pluijm for a critical review and major improvements of the paper. C.R. enjoyed discussions and field work with Uwe Ring. Financial support from the Land Baden-Württemberg (Germany) and the German Science Foundation (DFG grant Ri576/1-1) is gratefully acknowledged.

REFERENCES

Aleksandrowski, P., 1985, Graphical determination of principal stress directions for slickenside lineation populations: an attempt to modify Arthaud's method: Jour. Struct. Geology, v. 7, no. 1, p. 73–82.

Angelier, J., 1979, Determination of the mean principal stress for a given fault population: Tectonophysics, v. 56, no. 3/4, p. T17–T26.

Angelier, J., 1984, Tectonic analysis of fault slip data sets: Jour. Geophys. Res., v. 89, no. B7, p. 5835–5848.

Angelier, J., and Goguel, J., 1979, Sur une méthode simple de détermination des axes principaux des constraintes pour une population de failles: C. R. Acad. Sci. Paris, v. 288, Sér. D, p. 307–310.

Angelier, J., and Mechler, P., 1977, Sur une méthode graphique de recherche des constraintes principales également utilisable en tectonique et en seismologie: la méthode des dièdres droits: Bull. Soc. Géol. France, v. 21, no. 6, p. 1309–1318.

Angelier, J., Colletta, B., and Anderson, E. R., 1985, Neogene paleostress changes in the Basin and Range: a case study at Hoover Dam, Nevada–Arizona: Geol. Soc. America Bull., v. 96, no. 3, p. 347–361.

Armijo, R., Carey, E., and Cisternas, A., 1982, The inverse problem in microtectonics and the separation of tectonic phases: Tectonophysics, v. 82, no. 1, p. 145–160.

Arthaud, F., 1969, Méthode de détermination graphique des directions de racourcissement, d'allongement et intermédiaire d'une population de failles: Bull. Soc. Géol. Fr., v. 11, no. 7, p. 729–737.

Doglioni, C., 1985, The overthrusts in the Dolomites: ramp-flat systems: Eclogae Geol. Helv., v. 78, no. 2, p. 335–350.

Doglioni, C., 1987, Tectonics of the Dolomites (Southern Alps, Northern Italy): Jour. Struct. Geology, v. 9, no. 2, p. 181–193.

Etchecopar, A., Vasseur, G., and Daignieres, M., 1981, An inverse problem in microtectonics for the determination of stress tensors from fault striation analysis: Jour. Struct. Geology, v. 3, no. 1, p. 51–65.

Hoeppener, R., 1955, Tektonik im Schiefergebirge: Geol. Rundschau, v. 44, p. 26–58.

Huang, Q., 1989, Modal and vectorial analysis for determination of stress axes associated with fault slip data: Jour. Math. Geology, v. 21, no. 5, p. 543–558.

Lisle, R. J., 1988, ROMSA: a BASIC program for paleostress analysis using fault-striation data: Computers & Geosciences, v. 14, no. 2, p. 255–259.

Ring, U., and Richter, C., 1992, The evolution of Southalpine unit east of the Giudicarie line since the Cambrian: repeated subsidence and tectonism: Geological Mag., in press.

Wallbrecher, E., 1986, Gefügekundliche Arbeitsmethoden: Enke Verlag, Stuttgart, 208 p.

APPENDIX

SPLIT.C Program Listing

```
/*-----------------------------------------------------------------*/
/*  APPENDIX 1: Listing of SPLIT.C            (Krejci & Richter, 1991) */
/*                                                                 */
/*  Use TurboC Version 2.0  Compiler -> model: large               */
/*                          Linker   -> Default libraries: on      */
/*                          Linker   -> Graphics library : on      */
/*                          Debug    -> Source debugging : off     */
/*  Note: graphic drivers (*.bgi) must be registered at graphics.lib! */
/*-----------------------------------------------------------------*/

#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <graphics.h>

#define MAXNSAMPLES     1000
#define PI              3.14159
#define RAD             (3.14159/180)
#define SYM_SIZE        4

double (*azi_planar)[],(*dip_planar)[],(*azi_linear)[],(*dip_linear)[],
       (*azi_planar_sav)[],(*dip_planar_sav)[],(*azi_linear_sav)[],(*dip_linear_sav)[],
       (* azical)[],(* dipcal)[],(* xval)[],(* yval)[],(*yval_sav)[],aspectratio;
char   sense[MAXNSAMPLES][2], plus_[]={"+"}, minus[]={"-"}, null_[]={"0"};
int    sym[MAXNSAMPLES], maxx, maxy, error, radius, graphdriver, graphmode,
       nitems, index[MAXNSAMPLES], index[MAXNSAMPLES],maximum;
void   *arrow; void *circ;  /* here other symbols may be defined */
/*..............................................................*/
void create_arrow()
 { int picsize;
   if(graphdriver==HERCMONO)
     {
       picsize=imagesize(0,0,16,16); arrow=malloc(picsize);
       line(0,7,11,7); line(0,8,12,8); line(0,9,11,9); line(7,4,7,12); line(8,5,8,11);
       line(9,6,9,10); line(7,4,12,8); line(7,12,12,8); getimage(0,0,16,16,arrow);
     }
    else
     { setcolor(RED);
       picsize=imagesize(0,0,10,10); arrow=malloc(picsize);
       line(0,4,0,6); line(0,4,9,4); line(0,5,10,5); line(0,6,9,6);
       line(8,3,8,5); line(9,4,9,6); line(7,2,7,8);  line(7,2,10,5);
       line(7,8,10,5); setcolor(BLUE); getimage(0,0,10,10,arrow);
     }
 }
/*...............................................................*/
void create_circ()       /* in a similar way other symbols may be defined */
 { int picsize;
   picsize=imagesize(0,0,SYM_SIZE,SYM_SIZE); circ=malloc(picsize);
   circle(2,2,2); getimage(0,0,SYM_SIZE,SYM_SIZE,circ);
 }
/*...............................................................*/
void initialize()
 {
 int errflag=0; graphdriver = DETECT;
 errflag |= registerbgidriver(EGAVGA_driver); errflag |= registerbgidriver(Herc_driver);
 if (!errflag)
  {  printf("Graphics System Error: %s\n",grapherrormsg(error)); getch(); exit(1); }
 else
  { initgraph(&graphdriver,&graphmode,"");
    maxx = getmaxx(); maxy = getmaxy();
    if(graphdriver==HERCMONO) { radius=(int) maxy/2; aspectratio=0.667; }
          else              { radius=(int) maxy/2 * 0.75; aspectratio=1.0; }
 }
}
/*...............................................................*/
void _circle_()
{ /* Ellipse is necessary for Hercules */
  ellipse(maxx/2,maxy/2,0,360,radius,radius*aspectratio);
  line(maxx/2-radius, maxy/2, maxx/2+radius, maxy/2);
  line(maxx/2, maxy/2-radius*aspectratio, maxx/2, maxy/2+radius*aspectratio); }
/*...............................................................*/
double rad2sin (double in)  { return(sin(in*PI/180));  }
double rad2cos (double in)  { return(cos(in*PI/180));  }
double rad2tan (double in)  { return(tan(in*PI/180));  }
double rad2atan (double in) { return(atan(in)*180/PI); }
double deg2rad (double in)  { return(in*PI/180);       }
/*...............................................................*/
char readinput(char *inputfname,
             double azi_planar[], double dip_planar[],
             double azi_linear[], double dip_linear[])
 { FILE *inputfile; char line[100]; int i,k; inputfile=fopen(inputfname,"rt");
  if (inputfile==NULL) { printf("Cannot open data file!\n");
                         printf("Press any key...\n"); getch(); exit(1); }
   for (i=0,k=0;!feof(inputfile) && fgets(line,100,inputfile) && i<MAXNSAMPLES; i++,k++)
     sscanf(line," %lf %lf %lf %lf %s %d ",&azi_planar[i],&dip_planar[i],
                         &azi_linear[i],&dip_linear[i],&sense[i],&sym[i]);
  fclose(inputfile); nitems=k; return(0);
 }
/*...............................................................*/
double plot_symbol(double xval[], double yval[])
 { int k;
    for(k=0;k<nitems;k++)
    {
     if(sym[k] > 7) sym[k]-= 8; switch(sym[k])
      {
      case 0: putimage((int)xval[k]-SYM_SIZE/2,(int) yval[k]-SYM_SIZE/2, circ,XOR_PUT);
      break;
   /* case 1: if different symbols were defined, they have to be listed here */
      }
    }
return(0);
}
/*...............................................................*/
double calculate_point(double azi_planar[],double dip_planar[],
                     double xval[],double yval[],double yval_sav[])
{ int i,k; double l;
 for (i=0;i<nitems;i++) azi_planar[i]+=180;
```

```
  for (k=0;k<nitems;k++)
    { l=sqrt(2)*radius*rad2sin(dip_planar[k]/2);
      xval[k]     = maxx/2 + (rad2sin(azi_planar[k]) * l);
      yval[k]     = maxy/2 - (rad2cos(azi_planar[k]) * l) * aspectratio;
      yval_sav[k] = maxy/2 - (rad2cos(azi_planar[k]) * l);
    }
 return(0);
 }
 /*................................................................*/
 double greatcircle_coordinates(double azi_planar[],double dip_planar[],
                                double azi_linear[],double dip_linear[],
                                double azical[],double dipcal[])
 { int i;
   double px, py, pz, pp, intazi, intdip;
   for(i=0;i<nitems;i++)
   {
     azi_planar[i] = 180 + azi_planar[i];  dip_planar[i] = 90 - dip_planar[i];
     if (azi_planar[i] > 360) azi_planar[i]-=360;
     azi_planar[i] = RAD * azi_planar[i]; dip_planar[i] = RAD * dip_planar[i];
     azi_linear[i] = RAD * azi_linear[i]; dip_linear[i] = RAD * dip_linear[i];
     px = sin(azi_planar[i]) * cos(dip_planar[i]) * sin(dip_linear[i])
         -sin(dip_planar[i]) * sin(azi_linear[i]) * cos(dip_linear[i]);
     py = sin(dip_planar[i]) * cos(azi_linear[i]) * cos(dip_linear[i])
         -cos(azi_planar[i]) * cos(dip_planar[i]) * sin(dip_linear[i]);
     pz = cos(azi_planar[i]) * cos(dip_planar[i]) * sin(azi_linear[i])
         * cos(dip_linear[i]) - sin(azi_planar[i]) * cos(dip_planar[i])
         * cos(azi_linear[i]) * cos(dip_linear[i]);
    if (pz < 0) { px = -px; py = -py; pz = -pz;}
    pp = pz / sqrt ( pow(px,2) + pow(py,2) + pow(pz,2));
    intdip = atan (pp / (sqrt (1-pow(pp,2))) );
    if ( px == 0 && py > 0) intazi = PI/2; if ( px == 0 && py < 0) intazi = 3*PI/2;
    if (px !=0) intazi = atan(py/px);
    dipcal[i] = (intdip/RAD+0.5); azical[i] = (intazi/RAD+0.5);
    if (px < 0) azical[i] = azical[i] + 180;
    if (px > 0 && py < 0)  azical[i] = azical[i] + 360;  azical[i] = azical[i] + 180;
    if (azical[i] > 360) azical[i] = azical[i] - 360;    dipcal[i] = 90 - dipcal[i];
 }
return(0);
}
/*................................................................*/
void free_storage(void)
 {  if (azi_planar     ) free(azi_planar     ); if (dip_planar     ) free(dip_planar     );
    if (azi_planar_sav) free(azi_planar_sav); if (dip_planar_sav) free(dip_planar_sav);
    if (azi_linear_sav) free(azi_linear_sav); if (dip_linear_sav) free(dip_linear_sav);
    if (azi_linear     ) free(azi_linear     ); if (dip_linear     ) free(dip_linear     );
    if (azical         ) free(azical         ); if (dipcal         ) free(dipcal         );
    if (xval           ) free(xval           ); if (yval           ) free(yval           );
    if (yval_sav       ) free(yval_sav       );
 }
/*................................................................*/
void *qcalloc(size_t n,size_t size)
 {
  void *p=calloc(n,size); if(p) return(p); printf("Memory error!\n");
   printf("Press any key...\n"); getch();  exit(-1); return(0);
 }
/*................................................................*/
char write_output(int item, char *outputfname,
                  double azi_planar_sav[], double dip_planar_sav[],
                  double azi_linear_sav[], double dip_linear_sav[])
 { FILE *outputfile; int i;
   outputfile=fopen(outputfname,"wt");
   if (outputfile==NULL)
               { printf("Cannot open output file!\n");
                 printf("Press any key...\n"); getch(); exit(1); }
   for (i=0;i<nitems;i++)
   { if(index[i]==item)
     fprintf(outputfile," %3.0lf %3.0lf %3.0lf %3.0lf %s %d\n",
     azi_planar_sav[i],dip_planar_sav[i],azi_linear_sav[i],dip_linear_sav[i],
     sense[i],sym[i]);
   } fclose(outputfile); return(0);
 }
/*................................................................*/
```

```
double create_vector(double azical[],double dipcal[],
                  double azi_linear[], double dip_linear[],
                  double xval[], double yval[], double yval_sav[])
{ int i;
  double dist, rad, radius_bc, xbc, ybc, l, li_xval, li_yval,
         delta_x,delta_y,m_pl,dif,x_,y_,length=5,ybc_a,
         x_01, y_01, x_02, y_02, dif_01, dif_02,
         delta_x_li,delta_y_li,m_li;
 for(i=0;i<nitems;i++)
 { /* calculate x/y values of striation */
   if(dipcal[i] > 88.5) dipcal[i] = 88.5; dip_linear[i]=90-dip_linear[i];
    l=sqrt(2) * radius * rad2sin(dip_linear[i]/2);
     li_xval = maxx/2 + (rad2sin(azi_linear[i]) * l);
     li_yval = maxy/2 - (rad2cos(azi_linear[i]) * l); azical[i]+=180;
          dist= ( radius /(2*sqrt(2))) * (rad2sin(dipcal[i])/
                (rad2sin((90-dipcal[i])/2)));
          rad   = radius * sqrt(2) * rad2sin((90-dipcal[i])/2);
          radius_bc=dist+rad;
          xbc   = maxx/2 + dist * rad2cos(azical[i]-90);
          ybc   = maxy/2 + dist * rad2sin(azical[i]-90);
          ybc_a = maxy/2 + dist * rad2sin(azical[i]-90) * aspectratio;
          delta_x=xval[i]-xbc;  delta_y= yval_sav[i]-ybc;
          if(xval[i] > xbc)  m_pl=atan( delta_y/delta_x );
                else          m_pl=atan(delta_y/delta_x)+PI/(4/3);
          delta_x_li = li_xval-xbc;  delta_y_li = li_yval-ybc;
          if(li_xval > xbc)  m_li=atan( delta_y_li/delta_x_li );
               else          m_li=atan( delta_y_li/delta_x_li)+PI/(4/3);
          if(m_li < 0) m_li += 2*PI;    if(m_pl < 0) m_pl += 2*PI;
          if(m_pl < m_li)
           {
            if(stricmp(plus_,sense[i])==NULL)
             {  if( (m_li - m_pl) >= PI)
                  dif=+2*atan(length/radius_bc); else dif=- 2*atan(length/radius_bc);
             }  /* plus */
            if(stricmp(minus,sense[i])==NULL)
             {  if( (m_li - m_pl) >= PI)
                  dif= - 2*atan(length/radius_bc);  else
                  dif= + 2*atan(length/radius_bc);
             }  /* minus */
           }
           else
           {
            if(stricmp(plus_,sense[i])==NULL)
             {  if( (m_pl-m_li) >= PI)
                  dif= - 2*atan(length/radius_bc); else dif= + 2*atan(length/radius_bc);
             }  /* plus */
            if(stricmp(minus,sense[i])==NULL)
             { if( (m_pl-m_li) >= PI)
                dif= + 2*atan(length/radius_bc); else dif= - 2*atan(length/radius_bc);
             }  /* minus */
           }
           setcolor(14); if(stricmp(null_,sense[i])==NULL)
          { if(xval[i] > xbc)
          m_pl=atan(delta_y/delta_x);  else
          m_pl=atan(delta_y/delta_x) + PI/(4/3);
          dif_01= + 2*atan(length/radius_bc);  dif_02= - 2*atan(length/radius_bc);
          x_01=cos(m_pl+dif_01)*radius_bc+xbc;
          y_01=sin(m_pl+dif_01)*radius_bc*aspectratio+ybc_a;
          x_02=cos(m_pl+dif_02)*radius_bc+xbc;
          y_02=sin(m_pl+dif_02)*radius_bc*aspectratio+ybc_a;
          line(xval[i],yval[i],x_01,y_01); line(xval[i],yval[i],x_02,y_02);
          }
          /* plus and minus */
          else
          {
          x_=cos(m_pl+dif) * radius_bc + xbc;
          y_=sin(m_pl+dif) * radius_bc * aspectratio + ybc_a;
          line(xval[i],yval[i],x_, y_);
          }
        } /* for */
 setcolor(1);
return(0);
}
```

```
/*..................................................................*/
void create_index(int count)
{  char ch1;
  ch1=getch(); switch(ch1)
   {
    case '1': if(index[count]!=1)
               {
               if(graphdriver==HERCMONO) setcolor(0); else setcolor(7);
               outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"2");
               outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"3");
               outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"4");
               setcolor(1);
               }
               outtextxy((int) (*xval)[count]+3,(int) (*yval)[count]-3,"1");
               index[count]=1;  break;
    case '2': if(index[count]!=2)
              {
              if(graphdriver==HERCMONO) setcolor(0); else setcolor(7);
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"1");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"3");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"4");
              setcolor(1);
              }
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"2");
              index[count]=2;  break;
    case '3': if(index[count]!=3)
              {
              if(graphdriver==HERCMONO) setcolor(0); else setcolor(7);
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"1");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"2");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"4");
              setcolor(1);
              }
              outtextxy((int) (*xval)[count]+3,(int) (*yval)[count]-3,"3");
              index[count]=3;  break;
    case '4': if(index[count]!=4)
              {
              if(graphdriver==HERCMONO) setcolor(0); else setcolor(7);
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"1");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"2");
              outtextxy((int) (*xval)[count]+3,(int)(*yval)[count]-3,"3");
              setcolor(1);
              }
              outtextxy((int) (*xval)[count]+3,(int) (*yval)[count]-3,"4");
              index[count]=4;  break;
      }  /* switch */
}
/*..................................................................*/
void findmax()
{ int i;
  maximum=index[0]; for (i=1;i<nitems;i++) maximum=max(maximum,index[i]);
}
/*..................................................................*/
main(int argc,char *argv[])
{
int count,i,left,mid;
char ch, buf[40], out[12][12];
if (argc<2) {     /* information text for the program usage */
printf("Usage is: <split.exe> <work file (*.*)> \n");
printf("          Program asks for <output file(s) (*.*)>\n\n");
printf("Work file (six columns):\n");
printf("  Rows contain: 1. Dip direction of fault plane (max. 1000)\n");
printf("                2. Dip angle of fault plane (max. 1000)\n");
printf("                3. Strike of striation (max. 1000)\n");
printf("                4. Dip angle of striation (max. 1000)\n");
printf("                5. Direction of movement of the hanging block:\n");
printf("                   (+ up; - down; 0 not detected)\n");
printf("                6. Plot symbol (0 is default)\n");
printf(" Press '<'  and 1, 2, 3, or 4 to mark data set and write into\n");
printf("            different files!\n\n");
printf(" Copyright (c) by Dieter Krejci and Carl Richter 1991.\n\n");
exit(-1); }
 atexit(free_storage);  /* Clear all at exit */
```

```
/* Memory allocation */
  azi_planar     = qcalloc(MAXNSAMPLES,sizeof(double));
  dip_planar     = qcalloc(MAXNSAMPLES,sizeof(double));
  azi_planar_sav = qcalloc(MAXNSAMPLES,sizeof(double));
  dip_planar_sav = qcalloc(MAXNSAMPLES,sizeof(double));
  azi_linear_sav = qcalloc(MAXNSAMPLES,sizeof(double));
  dip_linear_sav = qcalloc(MAXNSAMPLES,sizeof(double));
  azi_linear     = qcalloc(MAXNSAMPLES,sizeof(double));
  dip_linear     = qcalloc(MAXNSAMPLES,sizeof(double));
  azical         = qcalloc(MAXNSAMPLES,sizeof(double));
  dipcal         = qcalloc(MAXNSAMPLES,sizeof(double));
  xval           = qcalloc(MAXNSAMPLES,sizeof(double));
  yval           = qcalloc(MAXNSAMPLES,sizeof(double));
  yval_sav       = qcalloc(MAXNSAMPLES,sizeof(double));
/* Read data from work file */
  readinput(argv[1],(double *) azi_planar, (double *)dip_planar,
                    (double *) azi_linear, (double *)dip_linear);
  for(i=0;i<nitems;i++)
   { (*azi_planar_sav)[i]=(*azi_planar)[i]; (*dip_planar_sav)[i]=(*dip_planar)[i];
     (*azi_linear_sav)[i]=(*azi_linear)[i]; (*dip_linear_sav)[i]=(*dip_linear)[i]; }

/* Calculate azi and dip values from a plane and a linear */
 greatcircle_coordinates((double *) azi_planar, (double *) dip_planar,
                         (double *) azi_linear, (double *) dip_linear,
                         (double *) azical    , (double *) dipcal     );
/* Begin of graphics */
 initialize();
 if(graphdriver==HERCMONO)  { left=16; mid=8; } else { left=10; mid=5; }
 setbkcolor(LIGHTGRAY); setcolor(BLUE);
 create_arrow(); cleardevice(); create_circ(); cleardevice();
  /* here all defined symbols have to be initialized */
 _circle_();  /* Plot circle */
  for(i=0;i<nitems;i++)
  { (*azi_planar)[i] = (*azi_planar_sav)[i]; (*dip_planar)[i] = (*dip_planar_sav)[i]; }
/* Calculate X and Y Values of points */
 calculate_point((double *) azi_planar,(double *) dip_planar,
                 (double *) xval,(double *) yval,(double*) yval_sav);
/* Plot symbols */
 plot_symbol( (double *) xval,(double *) yval); count=0;
 outtextxy(26,10," -> Number"); ultoa( count+1,buf,10); outtextxy(2,10,buf);
 outtextxy(2,maxy-20,"Esc -> to abort program");
 putimage((int)(*xval)[count]-left,(int)(*yval)[count]-mid,arrow,XOR_PUT);

/* Calculate and plot Hoeppener arrows */
for(i=0;i<nitems;i++)
 { (*azi_linear)[i] = (*azi_linear_sav)[i];
   (*dip_linear)[i]    = (*dip_linear_sav)[i];    }
create_vector( (double *) azical, (double *) dipcal,
             (double *) azi_linear, (double *) dip_linear,
             (double *) xval, (double *) yval, (double *) yval_sav);
/* Move arrow and select file number for new output */
 while( (ch=getch(), ch!=27) )
    { switch(ch)
      {
      case 'K': if(count>=1)      /* left cursor */
                { count-=1;
                  putimage((int)(*xval)[count+1]-left,
                           (int)(*yval)[count+1]-mid,arrow,XOR_PUT);
                  putimage((int)(*xval)[count]-left,
                           (int)(*yval)[count]-mid,arrow,XOR_PUT); } break;
      case 'M': if(count>=0 && count < nitems-1)   /* right cursor */
                { count+=1;
                  putimage((int)(*xval)[count-1]-left,
                           (int)(*yval)[count-1]-mid,arrow,XOR_PUT);
                  putimage((int)(*xval)[count]-left,
                           (int)(*yval)[count]-mid,arrow,XOR_PUT); } break;
      case '<' : /* select file and plot number of file on screen */
                 create_index(count); break;
      }
 /* Write number of selected data set */
 setviewport(0,0,25,80,0); clearviewport(); ultoa( count+1,buf,10); outtextxy(2,10,buf);
  } /* while */
```

```
closegraph(); /* end of graphics */
findmax();
if(maximum > 0) /* Write data to new files */
  {
   for(i=0;i<maximum;i++) { printf(" %d. File name? ",i+1); gets(out[i]); }
    for(i=0;i<maximum;i++)
      write_output(i+1,out[i],(double *) azi_planar_sav, (double *) dip_planar_sav,
                              (double *) azi_linear_sav, (double *) dip_linear_sav);
}
return(0);
}
```