システム・ソフトウェア開発

方向性データを変換するソフトウェアFDConverterの開発

山路 敦*・大原修平*

FDConverter: A Software for the Conversion of Orientation Data

Atsushi YAMAJI* and Shuhei OHARA*

Abstract : We have developed a software to convert orientation data for fault-striation analysis. The software specializes in dealing with the strike and dip of the format, e.g., N30E50E, that is common among Japanese geologists. Each fault data consist of the strike and dip of fault plane, orientation of striation, and sense of shear. If the orientation of striation is measured by its azimuth and plunge, the measurement error often shifts the orientation from the fault plane. In that case, the software calculate the angular misfit between the orientation from the fault plane. The result can be saved as a text file for later use. The software is useful not only for fault analysis but also other purposes, such as paleo-current analysis.

Key words : Orientation data, Data conversion, Fault striation analysis

1. はじめに

フィールドでわれわれは、断層のスリップベクトルや古流向といった、方向にかかわるデータを数多く取得する。そうしたデータも、近年は計算機で処理することが多くなったが、その場合N29E69Sというような文字列データを、計算機が処理しやすい数値データに変換する必要がある。筆者らが小断層解析のpre-processorとして開発したソフトウェアFDConverterは、そうした際にひろく利用され得るものと思い、ここに公表することにした。断層だけでなく、変成岩のリニエーションや堆積構造など、色々な種類の方向性データを変換するときにも使える。そうした目的では、すでにShorrock and Lisle (1998) によるソフトがある。しかし、欧米で普及している走向傾斜の記載様式がわが国のそれと違うため、例えばN29E69Sというようなデータフォーマットのままでは、かれらのソフトは処理できない。そこで、ソフトウェアを独自に作製することにしたわけである。本ソフトウェアはフリーソフトとして公開するつもりである。興味をもたれた方は、筆者までご連絡いただきたい。

2. ソフトウェアの機能概要

このソフトウェアは、FDConverterという名前で、元来は断層データ (Fault Data) を処理するために造られたもの

である。わが国においては、あまり行われなかったことだが、海外で行われている小断層解析では断層面に残された断層運動時のひっかき傷である断層擦痕 (striation or striae) の方向をかならず記載する。そこでこのソフトでは、断層面と擦痕の方向をあわせて処理できるようになっている。付加的部分をのぞいた核になるのは、断層面と擦痕のように、面的構造とそのうえにのる線적構造の方向をあらわす方向を文字列データとして受け取って、数値データとして出力する機能である。第1図のようなテキストファイルを入力し、変換結果と変換時のエラーメッセージとを別のテキストファイルに出力する。小断層解析以外でも、面的構造と線적構造との同様の組合せからなるデータについて、このソフトを利用することができる。

面的構造については、たとえば面的構造の方向をN30E20Eというような露頭で記載されえる通りの文字列として受け取って走向と傾斜の数値に分解し、その方位角 (azimuth) と面の傾斜またはプランジ (plunge) の値を出力する。方位角とは第2図のように真北をゼロとして、時計まわりにはかった角である。また、プランジとは線적構造が水平面となす角のことである。出力する面の方位は、第2図のD (最大傾斜方向)、R (right-hand ruleによる走向方向)、L (left-hand ruleによる走向方向) のいずれにするか、選ぶことができる。

線적構造の方向を露頭で記載するときには、面的構造の

2001年1月15日受付; 2001年2月24日受理

* 京都大学大学院理学研究科地球惑星科学専攻 Division of Earth and Planetary Sciences,
Kyoto University, Kyoto 606-8512, Japan. E-mail: ayamaji@ip.media.kyoto-u.ac.jp

断層擦痕のような線的な構造の方向を表すには、方位角とプランジの組を使う方法と、断層面の走向傾斜とレイクの組を使う方法との、2つの記載法があるが(第2図)、本ソフトウェアはどちらの形式でも受け付ける。

方位角・プランジを入力する場合は、間を半角の/ (スラッシュ) で区切る。方位角は0から360までの整数とする。プランジは0から90までの整数である。第2図の例ならば、88/22である。

露頭状況によっては、方位角とプランジのうち片方しか測れないことがある。その場合でも、断層面の方向がわかっていれば、もう一方は計算で求めることができる。そこで片方しか指定しなくとも、このソフトは受け付ける。

方位角 (s_{az}) のみが測られた場合は、AN88Eのように記述する。ここで第1文字のAは、方位角のみであることを表している。つづくN88Eは真北から東に向かって88°ということである。同様に、AS18Eというような真南を基準とする記述も許す。これは、方位角が $180^\circ - 18^\circ = 162^\circ$ の方向である。

プランジ (s_{pl}) しか記載していない場合も許容するが、その場合は、P22Nのように記述する。第1文字のPはplungeのみのデータであることをあらわす。プランジが等しくても、第2図のように、●と○の2方向がありうるので、どちらであるか最後の文字で指定する。P22Nの場合、北寄りである●の方向ということである。

レイクで擦痕の方向を記載した場合は、R22Nのように

記述する。第1文字のRは、rakeの頭文字、最後のNは、P22Nのときと同じで、●と○の2方向のうち北寄りの方という意味である。

断層変位のセンスは、正断層 (normal) ・逆断層 (reverse) ・右ずれ (dextral) ・左ずれ (sinistral) の頭文字をとって、N, R, D, Sのいずれかの文字とする。

小断層データ以外に、面的構造とその上にある線的構造の方向をデータ変換したいという場合にも、このソフトウェアを利用することができる。例えば古流向なら、断層面の走向傾斜のかわりに層理面のそれを記入し、擦痕の方向のかわりに古流向のそれを記入したデータファイルを入力すればよい。その場合、変位のセンスは意味がないが、N, R, D, Sのうちから任意の文字を選んで記入しておけばよい。それら以外の文字でも問題はないが、その場合はRemark欄にエラーメッセージが出力される。第1b図の例では、礫のimbricationやrippleマークから古流向を推定したなら、その旨を頭文字I, Rなどで表している。

3.2 使用方法

今回作成したソフトウェアは、FDConverter.exeという名前の実行ファイルになっている。それを通常のWindows用ソフトと同様に起動する。そしてFileメニューからOpenを選ぶと、変換すべきデータのはいったテキストファイルを指定するダイアログが開く。データファイルを指定すると

Original Data		Reformatted Data						
	Fault Plane	Striae	Dip Direction		Striae Direction		Sense	Remark
			az	pl	az	pl		
1	N87W54S	117/29	183.0	54.0	117.0	29.0	N	Misfit=0.2 # ○○地点
2	N47E85E	224/26	137.0	85.0	224.0	26.0	D	Misfit=0.5
3	N19W69W	88/49	251.0	69.0	88.0	49.0	R	Misfit=58.9
4	N79E80N	R31W	349.0	80.0	265.0	30.5	N	
5	N56E55N	AN68W	326.0	55.0	292.0	49.8	R	
6	N46E79N	P20W	316.0	79.0	230.1	20.0	D	
7	N63W77S	R34E	207.0	77.0	125.6	33.0	N	# 何々層
8	N69W68N	A40W	21.0	68.0	?	?	N	
9	N87E93S	P3W	?	?	?	?	D	
10	N50WV	R9W	40.0	90.0	310.0	9.0	D	
11			?	?	?	?	N	
12	N88W61S	R90W	182.0	61.0	2.0	61.0	N	
13	N85E67N	P53E	355.0	67.0	50.7	53.0	N	

sample_data_1.txt Number of Fault = 59

第3図 実行画面。第1図のデータを入力し、変換結果が表示されたところ。

変換が実行される。第1 a図のファイルを変換した結果が、第3図に示されている。ウィンドウに表示された一覧表において、Fault Planeの列とStriaeの列とSenseの列は、データファイルにあった上記の3つの列がそのまま表示されている。Reformatted Dataの下には4列あって、そのうち始めの2列は断層面の方向を、後の2列は擦痕の方位角・プランジをそれぞれ表す。すべて単位は、degreeである。変換不能のデータを与えた場合、数値のかわりに疑問符が表示される。最後のRemarkの列には、データファイルにあった注釈が表示される。ただし、注釈のみの行は、無視され、ウィンドウに現れない。Sense欄に変換不能の文字があれば、エラーメッセージが、やはりRemarkの欄に記入される。

断層擦痕は測定誤差のため、必ずしも断層面と平行にはならない。そこでその場合は、何度ずれるかを計算し、「Misfit=何度」という具合にRemark欄に表示する。また、その場合は、擦痕の方位角・プランジの測定値の示す方向をまず計算し、それを断層面上に射影し、その方位角・プランジを表示する。第1, 2, 3番のデータがその例である。

入力する角度はすべて整数だが、上記のような射影操作によって結果は実数になる。しかし、露頭における測定には 1° あるいはそれ以上の誤差があるので、多くの桁数で結果を出力しても意味がない。そこで、OptionメニューのInteger or Floatというサブメニューから、結果を整数で表示するか小数点以下第1位まで表示するか、選択できるようになっている。デフォルトでは、後者である。

断層面の方向の変換結果は、3通りの表示法から選ぶことができる。最大傾斜方向の方位角・プランジ（第2図の f_{az} と f_{pl} ）がデフォルトの選択肢である。ほかには、OptionメニューからRight-Hand RuleとLeft-Hand Ruleが選べる。前者は、腕を走向と平行にのばし、右手を断層面と平行に伏せ、親指を最大傾斜方向に向けたときの人差し指の方位を走向の方位とする（Barnes, 1991）。これは、第2図のRの方向であり、 $f_{az} + 90^\circ$ の方位角である。その場合のplunge欄は、面の傾斜が記入される。Left-Hand Ruleによる走向の方位は逆に、第2図のLの方向、すなわち $f_{az} - 90^\circ$ の方位である。ただし、Right-Hand Ruleには逆向きの定義もあるので注意すべきである（Marshak and Mitra, 1988）。

変換結果を保存したければ、FileメニューからSaveを選べば、保存すべきファイルの名前を指定するダイアログが開き、保存できる。出力ファイルはテキストファイルになる。保存するときに、ファイルの種類としてテキストファイルと断層スリップデータ（Fault-slip data）のいずれかを選ぶことができる。前者がデフォルトである。

3.3 出力データ形式

出力ファイルはテキストファイルで、ファイルの種類がテキストファイルか断層スリップデータかでその内容が異なる。前者の場合、出力ファイル名に拡張子TXTが添付さ

れる。内容は実行画面の一覧表そのままである。後者の種類を選択すると、ファイルの拡張子がFDTとなり、一覧表のReformatted DataとSenseの列のみが保存される。いずれにせよ出力ファイルの各行において、項目間はTABで区切られている。結果はテキストファイルだから、表計算ソフトやエディターで編集することができる。

3.4 使用上の注意事項

入力ファイルは、空白区切りのテキストファイルである。面的構造と線的構造を表す文字列の境界を、空白文字として認識するようになっているので、データをタブやコンマで区切ったファイルでは処理できず、認識できなかった行にエラーメッセージが表示される。

必要とする資源について次に述べる。入力ファイルの行数は、使用する計算機のメモリーで上限が規制される。このソフトウェアは、行数に応じてメモリーが動的に確保される仕組みになっているからである。注釈には文字数の制限がないので、消費するメモリーの量は注釈の文字数にも依存する。注釈無しの場合について、データ数とメモリー量との関係を示したのが第4図である。フィールドで実際に得られる現実的なデータ数はたかだか数百～数千であろう。その場合に必要なメモリーは5 Mbyteあまりである。

データ処理で時間を食うのは、ファイルの読み込みと実際の変換作業である。データ数が1万のファイルをPentium

第4図 データ数と消費する計算機メモリーとの関係。データを読み込んでない状態で、4.8Mbyteを要する。グラフの横軸は、波線の右側で対数目盛。

166MHzのパソコンで処理する場合、要する時間は20秒ほどである。

Windows OSが日本語版でなくとも、メニュー項目など基本的な部分ではArialフォントを使っているの、同フォントをインストールしたパソコンであれば英数字については、問題なく処理できる。ただし日本語の注釈を表示するため、変換結果の一覧表(第3図)でMSゴシックフォントを使っている。したがって、そこでは英数字と日本語フォントの他は、文字化けする。

3.5 使用例

第1a図の断層スリップデータを読み込んだ結果が、第3図に示されている。実行画面には処理結果が一覧表示されている。表の第1列は、断層の番号である。第1～7番の断層の走向傾斜と擦痕の方向をステレオネットにプロットして第5図に示す。

第1番の断層は、擦痕の方向を方位角とプランジで記載したので、断層面となす角がMisfit = 0.2°であることがRemark欄に示されている。この欄の#記号以下は、入力ファイルにあった注釈である。第3番の断層も、擦痕の方向が方位角とプランジで記載されているが、断層面とのmisfitが58.9°もある(第5図)。このことは、この断層を記載した際に、なんらかの間違いがあったことを示唆している。

第8, 9, 11番の断層は、入力文字列に不正な部分がある場合の例である。実行画面の一覧表では、解釈できなかった旨、疑問符が表示されている。第8番の場合は、擦痕の方

向がAN40WかAS40Wか、第2文字のNあるいはSが無いために判断できないというエラーである。第9番は、断層面の傾斜が90°を超えているので、エラーになった。また第11番は、変位のセンスのみで、断層と擦痕の方向が入力されなかったためにエラーになっている。

5. ソースコードの概要

プログラミングには、Panasonic製サブノートパソコンLet's note AL-N2T516 (CPU: MMX Pentium, 166 MHz, メモリ: 64 MB)を用いて、Borland (株)の開発ソフトウェアDelphi ver. 4で作成した。言語はObject Pascalである。

本ソフトウェアのソースコードは千行あまりの長さであるが、その多くはGUIの制御のためのもので、データ変換の核心部は全体の2/3程度である。以下、ソースコードの概要のみについて説明する。データファイルの読み込みから結果の保存までの処理の概要を、第6図にフローチャートでしめす。チャートの右側には、当該処理を受け持つ関数または手続き名が示されている。それらの名前はすべてTFormMainではじまっているが、それはTFormMainというクラスに属することを示している。付録には、ソースコードを示す。

データ変換のルーチン群は、2つに大別できる。文字列データから数値を抽出するものと、走向・傾斜・レイクなどの数値から方位角とプランジを計算するものである。

5.1 文字列データの処理

文字列データの処理部は、断層面の走向傾斜と擦痕の方向という2つの処理に分けられる(第6b図)。データファイルの行数を任意とするため、まず任意行数の文字列データを格納するためのコンテナであるRichEditにファイルの内容をロードする(procedure TFormMain.Open1Click)。RichEditの行数にもとづいて、function TFormMain.decompose_dataにおいて変換結果を格納する構造体の動的配列DataGridにメモリを割り当てる。この関数は、データの各行をスキャンし、データ間の空白を認識する。そして、断層面の走向傾斜・擦痕の方向・変位のセンス・注釈を文字列として切り出し、構造体の配列DataGridに格納する。次にfunction TFormMain.reformatからfunction TFormMain.fplane_is_OKを呼んで断層面の方位角・プランジを実数型データに変換し、配列DataGridに格納する。さらにfunction TFormMain.reformatからfunction TFormMain.striae_is_OKを呼んで、擦痕の方位角・プランジを計算し、同じくDataGridに格納する。それらの処理では、データの形式によって、第5c, d図に示す関数がはたらく。

第5図 断層面の走向傾斜と擦痕の方向を表す文字列(第1a図の1番目までのデータ)を下半球ステレオネットにプロットした図。大円が断層面を、小さな黒丸が擦痕の方向を表す。後者に添えられた白抜きの数字は、第1a図の第何行目のデータかを表す。第3のデータは、入力した擦痕の方向が、断層面と58.9°も離れている。

第6図 処理の概要をしめすフローチャート。(a) 全体の流れ。(b) 断層面の方向と擦痕の方向を表す文字列の抽出部分。(c) 断層面の走向傾斜を表す文字列の処理部。(d) 擦痕の方向を表す文字列の処理部。

5.2 断層擦痕の方向の数値処理

5.2.1 方位角/プランジが指定された場合

擦痕は断層面の上にはのっているはずだが、測定誤差のため、指定された方位角/プランジでは断層面と平行にならないことが普通である。そこで、指定された擦痕の方位角/プランジを単位ベクトルとして、それを断層面に正射投影する。そうして得られる面上のベクトルと、はじめの単位ベクトルとの角度差を計算し、構造体Tdataの misfit に格納

する。これらの処理は、function TFormMain.striae_is_azpl がfunction TFormMain.project_vector_onto_planeを呼び出して実行する。

直交デカルト座標系O-123を第7図のように定義する。擦痕の方向を表す単位ベクトルを $\vec{s} = (s_1, s_2, s_3)^T$ 、断層面に対する単位法線ベクトルを $\vec{n} = (n_1, n_2, n_3)^T$ とする。ただし、前者は下半球を、後者は上半球を向くものとする。また、単位行列をIとおく。するとベクトルの断層面

第7図 ソースコード中で使用する座標系。直交デカルト座標O-123の第1座標軸と第2座標軸を、それぞれ東と北に向け、第3座標軸を鉛直上向きとする。(a) オイラー角 θ , ϕ , ψ によって定義される、原点を共有する2つの座標系O-123とO-1'2'3'のあいだの関係。(b) 球座標を使う場合の余緯度 (col) と経度 (lon)。

への射影操作は、射影子 ($\mathbf{I} - \mathbf{N}$) を当該ベクトルの左側から掛け合わせるにより実行される (山路, 2000, p. 261)。ただし行列 $\mathbf{N}$ は、単位ベクトル $\vec{n}$ の成分から次のように構成される行列である：

$$N_{ij} = n_i n_j \quad (1)$$

射影の結果は ($\mathbf{I} - \mathbf{N}$) $\vec{s}$ なので、これを単位ベクトルになおして $\vec{s}$ と内積をとり、さらにその逆余弦をとれば misfit 角がえられる。

5.2.2 方位角またはプランジが指定された場合

これらの場合は、第2図の直角球面三角形LSAに球面三角法の公式を適用すれば、指定されなかった方の角が求まる。すなわち、球面三角形LSAは $\angle A = 90^\circ$ なので、式

$$\tan s_{pl} = \sin d \tan f_{pl} \quad (2)$$

が使えるわけである (中村, 1957, p. 131)。ただし d は、水

平面上の2点LとAのあいだの角距離である。

5.2.3 レイクが指定された場合

レイクは第3図の角 r である。断層面を第6a図の第1'2'座標平面に等しいとし、さらに第1'座標軸が擦痕と平行であるとする。断層面の方位角とプランジからオイラー角の θ と ϕ が決まり、レイクから ψ が決定できる。すなわち、 $\theta = f_{pl}$, $\psi = -r$ である。 ϕ は、擦痕の方向が right-hand rule の走向方向に近いのか left-hand rule の走向方向に近いかで $f_{az} + 90^\circ$ と $f_{az} - 90^\circ$ の場合に分かれる。原点を共有する2つの座標系O-123とO-1'2'3'のあいだの座標変換は、直交行列

$$\mathbf{Q} = \begin{pmatrix} \cos\psi \cos\phi - \cos\theta \sin\phi \sin\psi & -\sin\psi \cos\phi - \cos\theta \sin\phi \cos\psi & \sin\theta \sin\phi & \cos\psi \sin\phi + \cos\theta \cos\phi \sin\psi & \sin\psi \sin\theta \\ -\sin\psi \sin\phi + \cos\theta \cos\phi \cos\psi & \cos\psi \sin\theta & -\sin\theta \cos\phi & \end{pmatrix} \quad (3)$$

によって表される。求めたいのは、第1'座標軸の方向であるが、それに平行な単位ベクトルの成分は、 $\mathbf{Q}$ の第1行ベクトルひとしい (山路, 2000, p. 256)。そこでその指す方向の方位角とプランジを計算すればよい。ただし、ほかで作成した直交行列生成関数を流用したので、ソースコード中では、 $\mathbf{R} = \mathbf{Q}^{-1}$ の第1列ベクトルを使用している。

6. まとめ

面的構造とその上に載る線構造の方向を文字列として入力し、それらの方位角とプランジを数値に変換して、テキストファイルとして出力するソフトウェアFDConverterを開発した。元来これは小断層解析の pre-processor として開発したものだが、他の方向性データにも利用できる。例えば断層の走向傾斜とスリップ方向のかわりに、地層の走向傾斜と古流向のデータを変換したり、変成岩の片理の走向傾斜と、その上に載る引き伸ばし線構造の方向を変換するといった利用法がある。

引用文献

- Barnes, J. (1991) *Basic Geological Mapping*, 2nd ed., Open Univ. Press, Milton Keynes, 119 p.
 狩野謙一・村田明広 (1998) 構造地質学. 朝倉書店, 298 p.
 Marshak, S. and Mitra, G. (1988) *Basic methods of structural geology*. Prentice-Hall, Inc., 446 p.
 中村芳彦 (1957) 三角法 (新数学シリーズ4). 培風館, 166 p.

- Shorrock, C. and Lisle, R. J. (1998) DATACON: a QuickBASIC program to reformat orientation data from faults. *Comp. Geosci.*, vol. 24, pp. 585-590.
- 山路 敦 (2000) 理論テクトニクス入門：構造地質学からのアプローチ. 朝倉書店, 287 p.

要 旨

方向性データを変換するソフトウェアFDConverterの開発

：山路 敦・大原修平

断層面の走向傾斜についての文字列データから面の方位角とプランジを計算し、さらに断層擦痕の方向に関する文字列データからその方向の方位角とプランジを計算するソフトウェアFDConverterを作成した。断層に関するデータ以外にも、面的構造とその上に載る線の構造を記載するフィールドデータから、海外等で利用されているデータ形式の数値データに変換する手続きを、このソフトウェアを用いて実行することができる。

キーワード：方向性データ，データ変換，小断層解析

付録A メインプログラム (Fconv.pas)

```

unit Fconv;
|
| Main program of the project FDConverter.
|
| Version 1 January, 1999 (Ohara, S.)
|
| Version 2.0 January, 2001 (Yamaji, A.)
| The version published in Geoinformatics (Journal of Japan Society of
| Geoinformatics) .
|
|
| interface
|
| uses
| Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
| ImgList, Menus, ToolWin, ComCtrls, Grids, ExtCtrls, StdCtrls, Math, ShellAPI;
|
| const
|   MaxNumberOfData = 32767;
|   DefaultRowCount = 21;
|   NoMisfit = -1.0E+20;
|   CARRIAGE_RETURN_AND_LINE_FEED = #13#10;
|
| type
|   TCharSet = set of char; // N, E, S, W and |sense| are set when Form created
|
|   Tmatrix = array[1..3,1..3] of single;
|   Tvector = array[1..3] of single;
|
|   Tspherical = record // spherical coordinates
|     col, lon : single; // degrees
|   end;
|
|   Tdata = record
|     fplane, striae: string; // orientation of fault plane and striae
|     faz, fpl : single; // azimuth and plunge of fault plane
|     saz, spl : single; // azimuth and plunge of striae
|     sense : string; // sense of shear
|     rem : string; // remark
|     fOK : boolean; // validity of format of fplane
|     sOK : boolean; // validity of format of striae
|     linenum : smallint; // line number of DataList
|   end;
|
|   TFormMain = class (TForm)
|     MainMenu: TMainMenu;
|     File: TMenuItem;
|     Open1: TMenuItem;
|     Save1: TMenuItem;
|     N1: TMenuItem;
|     Exit1: TMenuItem;
|     ImageList1: TImageList;
|     Panel_base: TPanel;
|     Panel_original_data: TPanel;
|     StringGrid1: TStringGrid;
|     Panel_reformatted_data: TPanel;
|     Panel4: TPanel;
|     Panel5: TPanel;
|     OpenDialog1: TOpenDialog;
|     StatusBar1: TStatusBar;
|     ToolBar1: TToolBar;
|     ToolButton1: TToolButton;
|     ToolButton2: TToolButton;
|     Edit1: TMenuItem;
|     OptionDigit: TMenuItem;
|     SaveDialog1: TSaveDialog;
|     OptionPrec: TMenuItem;
|     OptionFO: TMenuItem;
|     Help2: TMenuItem;
|     OptionAbout: TMenuItem;
|     Integer1: TMenuItem;
|     Float1: TMenuItem;
|     OptionDO: TMenuItem;
|     OptionRHR: TMenuItem;
|     OptionLHR: TMenuItem;
|     ProgressBar1: TProgressBar;
|     procedure FormCreate (Sender: TObject);
|     procedure Open1Click (Sender: TObject);
|     procedure FormResize (Sender: TObject);
|     procedure Exit1Click (Sender: TObject);
|     procedure Save1Click (Sender: TObject);
|     procedure ToolButton2Click (Sender: TObject);
|     procedure FormDestroy (Sender: TObject);
|     procedure OptionAboutClick (Sender: TObject);
|     procedure Integer1Click (Sender: TObject);
|     procedure Float1Click (Sender: TObject);
|     procedure OptionDOClick (Sender: TObject);
|     procedure OptionRHRClick (Sender: TObject);
|     procedure OptionLHRClick (Sender: TObject);
|   private
|     | The dynamic array DataGrid contains the data shown in the StringGrid1.
|     | The array is allocated and deallocated in the procedures FormCreate () and
|     | FormDestroy (), respectively. The size of the array is reallocated in the
|     | procedure decompose_data ().
|     DataGrid : array of Tdata;
|
|   CharSet_NESW : TCharSet;
|   NumberOfData : smallint;
|   DataList : TStringList;
|
|   procedure WMDropFiles (var Message: TWMDropFiles); // for drag-and-drop files
|   message WM_DROPFILES; // for drag-and-drop files
|
|   procedure clear_StringGrid1 ();
|   procedure decompose_data ();
|   procedure reformat ();
|   procedure tabulate_data ();
|   procedure spherical_to_azpl (const p: Tspherical; var az, pl: single);
|   procedure horizontal_fault (const idat: cardinal);
|   function fplane_is_OK (const bedding: string; var az, pl: single): boolean;
|   function process_NS (const bedding: string; var az, pl: single): boolean;
|   function process_EW (const bedding: string; var az, pl: single): boolean;
|   function process_N (const bedding: string; var az, pl: single): boolean;
|   function striae_is_OK (const striae: string;
|     const faz, fpl: single; var az, pl: single;
|     var misfit: single): boolean;
|   function striae_is_azpl (const striae: string;
|     const faz, fpl: single; var az, pl: single;
|     var misfit: single): boolean;
|   function striae_israke_only (const striae: string;
|     const faz, fpl: single; var az, pl: single;
|     var misfit: single): boolean;
|   function striae_isazmth_only (const striae: string;
|     const faz, fpl: single; var az, pl: single;
|     var misfit: single): boolean;
|   function striae_isplunge_only (const striae: string;
|     const faz, fpl: single; var az, pl: single;
|     var misfit: single): boolean;
|   function sense_is_OK (const sense: string): boolean;
|   function matrix_times_vector (const a: Tmatrix; const x: Tvector): Tvector;
|   function orthogonal_tensor (const theta, phi, psi: single): Tmatrix;
|   function azpl_to_unit_vector (const az, pl: single): Tvector;
|   function azpl_to_spherical (const az, pl: single): Tspherical;
|   function unit_vector_to_spherical (const v: Tvector): Tspherical;
|   function dotprod (const a, b: Tvector): single;
|   function normalize_vector (const x: Tvector): Tvector;
|   function project_vector_onto_plane (const v: Tvector; const n: Tvector): Tvector;
|   function azimuthal_distance (const x, y: single): single;
|   function amod (const x, y: single): single;
|   function roundf (const x: single; const digit: byte): single;
|   public
|   protected
|     procedure CreateWnd (); override; // for drag-and-drop files
|   end; TFormMain;
|
| var
|   FormMain: TFormMain;
|
| implementation
| uses About;
|
| {$R *.DFM}
|
| //-----
| // This and following CreateWnd () procedures accept a file that is specified
| // through a drag-and-drop action.
|
| procedure TFormMain.WMDropFiles (var Message: TWMDropFiles);
| var
|   FileIndex, FileCount : Cardinal;
|   FileNameBuffer : PChar;
|   FileNameLength : Cardinal;
| begin
|   try
|     FileCount := DragQueryFile (Message.Drop, $FFFFFFF, nil, 0);
|     if FileCount > 1 then begin
|       ShowMessage ( 'Plural files were dropped.' + CARRIAGE_RETURN_AND_LINE_FEED
|         + 'This application accept only one file at a time.' );
|     end else begin
|       FileNameLength := DragQueryFile (Message.Drop, 0, nil, 0);
|       GetMem ( FileNameBuffer, FileNameLength+1 );
|       try
|         DragQueryFile (Message.Drop, FileIndex, FileNameBuffer, FileNameLength+1);
|
|         ProgressBar1.Position := 1;
|       try
|         DataList.LoadFromFile ( FileNameBuffer );
|       except
|         on EOpenError do begin
|           ShowMessage ( 'Cannot open the file '
|             + ExtractFileName (FileNameBuffer) );
|           ProgressBar1.Position := 0;
|           Exit;
|         end; |on|
|       end; |try|
|     if DataList.Count < MaxNumberOfData then begin
|       ProgressBar1.Position := 2;
|       decompose_data (); ProgressBar1.Position := 3;
|       reformat (); ProgressBar1.Position := 4;
|       tabulate_data (); ProgressBar1.Position := 0;
|       StatusBar1.Panels[0].Text := ExtractFileName (FileNameBuffer);
|       DataList.Clear;
|     end;
|   end;
| end;

```

```

end else begin
  ShowMessage ('The data file contains too many data. ');
end; if
finally
  FreeMem ( FileNameBuffer );
end; if
finally
  DragFinish (Message.Drop);
end; if
Message.Result := 0;
end; procedure TFormMain.WNDropFiles

//-----
// This procedure works with WNDropFiles () to accept drag-and-drop action.

procedure TFormMain.CreateWnd ();
begin
  inherited CreateWnd;
  if HandleAllocated then DragAcceptFiles ( Handle, TRUE );
end; procedure TFormMain.CreateWnd

//-----
procedure TFormMain.FormCreate (Sender: TObject);
begin
  CharSet_NESW := ['N', 'E', 'S', 'W'];

  DataList := TStringList.Create;
  DataList.Clear;
  SetLength (DataGrid, 20);
  clear_StringGrid1 ();
  with StringGrid1 do begin // first row of the grid
    RowCount := DefaultRowCount; // define in the const section of this unit
    Cells[1,0] := 'Fault Plane';
    Cells[2,0] := 'Striae';
    Cells[3,0] := 'az';
    Cells[4,0] := 'pl';
    Cells[5,0] := 'az';
    Cells[6,0] := 'pl';
    Cells[7,0] := 'Sense';
    Cells[8,0] := 'Remark';
  end; with
  with ProgressBar1 do begin // put ProgressBar1 on StatusBar1
    Parent := StatusBar1;
    Top := 5;
    Left := 360;
    Visible := True;
    Max := 5;
  end; with
end; procedure

//-----
procedure TFormMain.FormDestroy (Sender: TObject);
begin
  Finalize (DataGrid);
  DataList.Free;
end; procedure

//-----
procedure TFormMain.Exit1Click (Sender: TObject);
begin
  Application.Terminate;
end; procedure

//-----
procedure TFormMain.clear_StringGrid1 ();
var
  ic : byte;
  ir : cardinal;
begin
  with StringGrid1 do begin
    for ir := 1 to RowCount do Cells[0,ir] := IntToStr (ir);
    for ir := 1 to RowCount do
      for ic := 1 to ColCount-1 do Cells[ir,ic] := '';
    end; with
    DataList.Clear;
  end; procedure

//-----
procedure TFormMain.Open1Click (Sender: TObject);
begin
  ProgressBar1.Position := 0;
  if OpenFileDialog1.Execute then begin
    ProgressBar1.Position := 1;
    try
      DataList.LoadFromFile (OpenDialog1.FileName);
    except
      on EOpenError do begin
        ShowMessage ('Cannot open the file '
          + ExtractFileName (OpenDialog1.FileName) );
        ProgressBar1.Position := 0;
        Exit;
      end; if
    end; if
    if DataList.Count < MaxNumberofData then begin
      ProgressBar1.Position := 2;
      decompose_data (); ProgressBar1.Position := 3;

      reformat (); ProgressBar1.Position := 4;
      tabulate_data (); ProgressBar1.Position := 0;

      StatusBar1.Panels[0].Text := ExtractFileName (OpenDialog1.FileName);
      DataList.Clear;
    end else begin
      ShowMessage ('The data file contains too many data. ');
    end; if
    if OpenFileDialog1.Execute
  end; procedure

//-----
procedure TFormMain.FormResize (Sender: TObject);
var i : smallint;
begin
  with StringGrid1 do begin
    ColWidths[0] := 30; i := ColWidths[0];
    ColWidths[1] := 80; i := i + ColWidths[1];
    ColWidths[2] := 60; i := i + ColWidths[2];
    ColWidths[3] := 45; i := i + ColWidths[3];
    ColWidths[4] := 40; i := i + ColWidths[4];
    ColWidths[5] := 45; i := i + ColWidths[5];
    ColWidths[6] := 40; i := i + ColWidths[6];
    ColWidths[7] := 38; i := i + ColWidths[7];
    ColWidths[8] := max ( FormMain.Width - i, 500 );
  end; with
end; procedure

//-----
procedure TFormMain.tabulate_data ();
var
  iRow : cardinal;
  digit : byte;
  aztemp : single;
begin
  if Integer1.Checked then digit := 0;
  if Float1.Checked then digit := 1;

  with StringGrid1 do begin
    for iRow := 1 to NumberofData do begin
      Cells[1,iRow] := DataGrid[iRow].fplane;
      Cells[2,iRow] := DataGrid[iRow].striae;

      if DataGrid[iRow].fok then begin
        aztemp := DataGrid[iRow].faz;
        if OptionRHR.Checked then aztemp := amod ( aztemp+90, 360 );
        if OptionLHR.Checked then aztemp := amod ( aztemp-90, 360 );
        Cells[3,iRow] := FloatToStrF ( roundf (aztemp, digit), fFixed, 6, digit );
        Cells[4,iRow] := FloatToStrF ( roundf (DataGrid[iRow].fpl, digit), fFixed, 6, digit );
      end
      else begin
        Cells[3,iRow] := '?';
        Cells[4,iRow] := '?';
      end; if

      if DataGrid[iRow].sok then begin
        Cells[5,iRow] := FloatToStrF ( roundf (DataGrid[iRow].saz, digit), fFixed, 6, digit );
        Cells[6,iRow] := FloatToStrF ( roundf (DataGrid[iRow].spl, digit), fFixed, 6, digit );
      end
      else begin
        Cells[5,iRow] := '?';
        Cells[6,iRow] := '?';
      end; if

      Cells[7,iRow] := DataGrid[iRow].sense;
      Cells[8,iRow] := DataGrid[iRow].rem;
    end; for
  end; with
end; procedure

//-----
procedure TFormMain.Save1Click (Sender: TObject);
const TAB = char (9);
var
  f : TextFile;
  iRow : cardinal;
begin
  SaveDialog1.DefaultExt := 'txt';
  SaveDialog1.Options := [ofHideReadOnly, ofOverwritePrompt, ofExtensionDifferent];

  if SaveDialog1.Execute then begin
    try
      AssignFile ( f, SaveDialog1.FileName );
      Rewrite ( f );
      if LowerCase (ExtractFileExt (SaveDialog1.FileName)) = '.fdt' then
        for iRow := 1 to StringGrid1.RowCount - 1 do begin
          Writeln ( f, StringGrid1.Cells[1,iRow] + ' '
            + StringGrid1.Cells[4,iRow] + ' '
            + StringGrid1.Cells[5,iRow] + ' '
            + StringGrid1.Cells[6,iRow] + ' '
            + StringGrid1.Cells[7,iRow] );
        end; for
      else
        for iRow := 1 to StringGrid1.RowCount - 1 do begin
          Writeln ( f, StringGrid1.Cells[1,iRow] + TAB
            + StringGrid1.Cells[2,iRow] + TAB
            + StringGrid1.Cells[3,iRow] + TAB
            + StringGrid1.Cells[4,iRow] + TAB
            + StringGrid1.Cells[5,iRow] + TAB
            + StringGrid1.Cells[6,iRow] + TAB

```

```

        + StringGrid1.Cells[7,iRow] + TAB
        + StringGrid1.Cells[8,iRow] );
    end; {for and if}
    finally
        CloseFile (f);
    end; {try}
end; {if}
end; {procedure}

//-----
procedure TFormMain.ToolButton2Click (Sender: TObject);
begin
    SaveClick (Sender);
end; {procedure}

//-----
procedure TFormMain.OptionAboutClick (Sender: TObject);
begin
    FormAbout.Visible := True;
end; {procedure}

//-----
procedure TFormMain.Integer1Click (Sender: TObject);
var iRow : cardinal;
begin
    Integer1.Checked := True;
    Float1.Checked := False;
    for iRow := 1 to StringGrid1.RowCount - 1 do StringGrid1.Cells[8,iRow] := '';
    if DataList.Count > 0 then tabulate_data ();
end; {procedure}

//-----
procedure TFormMain.Float1Click (Sender: TObject);
var iRow : cardinal;
begin
    Integer1.Checked := False;
    Float1.Checked := True;
    for iRow := 1 to StringGrid1.RowCount - 1 do StringGrid1.Cells[8,iRow] := '';
    if DataList.Count > 0 then tabulate_data ();
end; {procedure}

//-----
procedure TFormMain.OptionDDClick (Sender: TObject);
var iRow : cardinal;
begin
    OptionDD.Checked := True;
    OptionRHR.Checked := False;
    OptionLHR.Checked := False;
    for iRow := 1 to StringGrid1.RowCount - 1 do StringGrid1.Cells[8,iRow] := '';
    if DataList.Count > 0 then tabulate_data ();
end; {procedure}

//-----
procedure TFormMain.OptionRHRClick (Sender: TObject);
var iRow : cardinal;
begin
    OptionDD.Checked := False;
    OptionRHR.Checked := True;
    OptionLHR.Checked := False;
    for iRow := 1 to StringGrid1.RowCount - 1 do StringGrid1.Cells[8,iRow] := '';
    if DataList.Count > 0 then tabulate_data ();
end; {procedure}

//-----
procedure TFormMain.OptionLHRClick (Sender: TObject);
var iRow : cardinal;
begin
    OptionDD.Checked := False;
    OptionRHR.Checked := False;
    OptionLHR.Checked := True;
    for iRow := 1 to StringGrid1.RowCount - 1 do StringGrid1.Cells[8,iRow] := '';
    if DataList.Count > 0 then tabulate_data ();
end; {procedure}

//-----
// Purpose : decompose lines of datafile into orientation of fault plane,
//           striae, sense of shear, and comment.

procedure TFormMain.decompose_data ();
var
    line : cardinal;
    idat : cardinal;
    buf : string;
    bedding : string;
    striae : string;
    sense : string;
    comment : string;
    pos : smallint; // position of character in string
begin
    idat := 0;
    StringGrid1.RowCount := DataList.Count + 1;

    Finalize (DataGrid);
    SetLength (DataGrid, DataList.Count + 1);

    for line := 0 to DataList.Count - 1 do begin
        buf := DataList.Strings[line];
        if buf = '' then continue;
        buf := trim (buf);
        if buf[1]='#' then continue;
        pos := AnsiPos ('#', buf);
        if pos>0 then begin // the line has a comment part
            comment := Copy (buf, pos, Length (buf) -pos+1);
            buf := Copy (buf, 1, pos-1);
        end
        else begin
            comment := '';
        end;
        if
            pos := AnsiPos (' ', buf);
            bedding := trim (Copy (buf, 1, pos));
            buf := trim (Copy (buf, pos, Length (buf) -pos+1));
            pos := AnsiPos (' ', buf);
            striae := trim (Copy (buf, 1, pos));
            sense := trim (Copy (buf, pos, Length (buf) -pos+1));

            idat := idat + 1;
            StringGrid1.Cells[0,idat] := IntToStr (idat);
            DataGrid[idat].fplane := bedding;
            DataGrid[idat].striae := striae;
            DataGrid[idat].sense := sense;
            DataGrid[idat].rem := comment;
            DataGrid[idat].linenum := line;
            NumberOfData := idat;
        end; {for line}

        StatusBar1.Panels[1].Text := 'Number of Fault = ' + IntToStr (idat);
    end; {procedure}

//-----
procedure TFormMain.horizontal_fault (const idat: cardinal);
var
    saz, spl : single;
    misfit : single;
begin
    DataGrid[idat].fOK := True;
    DataGrid[idat].faz := 0;
    DataGrid[idat].fpl := 0;

    if not striae_is_azmth_only (DataGrid[idat].striae, 0, 0, saz, spl, misfit)
    then begin
        DataGrid[idat].sOK := False;
        DataGrid[idat].rem := '! Error in striae ' + DataGrid[idat].rem;
        Exit;
    end;
    DataGrid[idat].sOK := True;
    DataGrid[idat].saz := saz;
    DataGrid[idat].spl := spl;
    {faz is defined parallel to saz for horizontal fault}
    if OptionDD.Checked then DataGrid[idat].faz := saz
    else if OptionRHR.Checked then DataGrid[idat].faz := amod (saz+90+720, 360)
    else {OptionLHR.Checked} DataGrid[idat].faz := amod (saz-90+720, 360);

    if (UpperCase (DataGrid[idat].sense) <> 'N') or
       (UpperCase (DataGrid[idat].sense) <> 'R') then begin
        DataGrid[idat].rem := '! Error in sense ' + DataGrid[idat].rem;
    end; {if sense_is_OK}
end; {procedure}

//-----
procedure TFormMain.reformat ();
var
    idat : cardinal;
    misfit : single;
    digit : byte;
begin
    if Integer1.Checked then digit := 0;
    if Float1.Checked then digit := 1;

    for idat := 1 to NumberOfData do begin
        if DataGrid[idat].fplane = 'H' then begin
            horizontal_fault (idat);
            continue;
        end; {if horizontal fault}

        try
            if not sense_is_OK (DataGrid[idat].sense) then begin
                DataGrid[idat].rem := '! Error in sense ' + DataGrid[idat].rem;
            end; {if sense_is_OK}

            if fplane_is_OK (DataGrid[idat].fplane,
                           DataGrid[idat].faz, DataGrid[idat].fpl) then
                DataGrid[idat].fOK := True
            else begin
                DataGrid[idat].fOK := False;
                continue;
            end; {if}

            if striae_is_OK (DataGrid[idat].striae,
                           DataGrid[idat].faz, DataGrid[idat].fpl,
                           DataGrid[idat].saz, DataGrid[idat].spl, misfit) then begin
                if misfit >= 0 then
                    DataGrid[idat].rem := 'Misfit='
                    + FloatToStrF (roundf (misfit,digit), fFixed, 6, digit) + ' '
                    + DataGrid[idat].rem;
            end;
        end;
    end;
end;

```

```

    DataGrid[idat].sOK := True;
  end
else begin
  DataGrid[idat].sOK := False;
end; {if}
except
  on EAccessViolation do {nothing};
end; {try}
end; {for i}
end; {procedure}

//-----
function TFormMain.striae_is_OK ( const striae : string;
                                const faz, fpl : single;
                                var az, pl : single;
                                var misfit : single ) : boolean;
begin
  if length (striae) = 0 then begin
    result := False;
    Exit;
  end; {if}

  if AnsiPos ('/', striae) > 0 then begin // e.g., 238/34
    result := striae_is_azpl (striae, faz, fpl, az, pl, misfit);
    Exit;
  end; {if}

  if striae[1] = 'R' then begin // e.g., R23W
    result := striae_is_rake_only (striae, faz, fpl, az, pl, misfit);
    Exit;
  end; {if}

  if striae[1] = 'A' then begin // e.g., AN23E
    result := striae_is_azmth_only (striae, faz, fpl, az, pl, misfit);
    Exit;
  end; {if}

  if striae[1] = 'P' then begin // e.g., P33S, P4W
    result := striae_is_plnge_only (striae, faz, fpl, az, pl, misfit);
    Exit;
  end; {if}

  result := False;
end; {function}

//-----
function TFormMain.striae_is_plnge_only ( const striae : string;
                                          const faz, fpl : single;
                                          var az, pl : single;
                                          var misfit : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
  direc : char;
  er : integer;
begin // e.g., P45W, P5S
  misfit := NoMisfit; // frag

  val ( Copy (striae, 2, length (striae) - 2), pl, er );
  direc := striae[length (striae)];

  if er < 0 then goto End_of_this_function_and_return_FALSE;
  if not ( direc in CharSet_NESW ) then goto End_of_this_function_and_return_FALSE;

  try
    az := arcsin ( tan (DegToRad (pl)) / tan (DegToRad (fpl)) );
    az := RadToDeg (az);
    if ( 0 <= faz ) and ( faz < 90 ) then // dipping to NE quadrant
      case direc of
        'W', 'N' : az := faz - 90 + az;
        'E', 'S' : az := faz + 90 - az;
      end {case}
    else if ( 90 <= faz ) and ( faz < 180 ) then // dipping to SE quadrant
      case direc of
        'N', 'E' : az := faz - 90 + az;
        'S', 'W' : az := faz + 90 - az;
      end {case}
    else if ( 180 <= faz ) and ( faz < 270 ) then // dipping to SW quadrant
      case direc of
        'S', 'E' : az := faz - 90 + az;
        'N', 'W' : az := faz + 90 - az;
      end {case}
    else begin // dipping to NW quadrant
      case direc of
        'W', 'S' : az := faz - 90 + az;
        'E', 'N' : az := faz + 90 - az;
      end {case}
    end; {if}
    result := True;
  except
    on EInvalidOp do result := False;
  end; {try}

  az := amod (az + 720, 360);
  Exit;

End_of_this_function_and_return_FALSE;
result := False;
end; {function}

//-----
function TFormMain.striae_is_azmth_only ( const striae : string;
                                          const faz, fpl : single;
                                          var az, pl : single;
                                          var misfit : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
  direc1 : char; // N or S
  direc2 : char; // E or W
  er : integer;
  angulardistance : single;
begin // e.g., AN23E, AN2E, AS26W, A40W
  misfit := NoMisfit;

  val ( Copy (striae, 3, length (striae) - 3), az, er );
  direc1 := striae[2];
  direc2 := striae[length (striae)];

  if er < 0 then goto End_of_this_function_and_return_FALSE;
  if not (direc1 in CharSet_NESW) then goto End_of_this_function_and_return_FALSE;
  if not (direc2 in CharSet_NESW) then goto End_of_this_function_and_return_FALSE;

  case direc1 of
    'N': case direc2 of
      'E': {do nothing};
      'W': az := 360 - az;
    else goto End_of_this_function_and_return_FALSE;
    end; {case direc2}
    'S': case direc2 of
      'E': az := 180 - az;
      'W': az := 180 + az;
    else goto End_of_this_function_and_return_FALSE;
    end; {case direc2}
  else goto End_of_this_function_and_return_FALSE;
end; {case direc1}

{determine plunge}
angulardistance := azimuthal_distance (faz+90, az);
if angulardistance > 90 then
  angulardistance := azimuthal_distance (faz-90, az);
pl := RadToDeg ( arctan ( sin (DegToRad (angulardistance)) * tan (DegToRad (fpl)) ) );

result := True;
Exit;

End_of_this_function_and_return_FALSE;
result := False;
end; {function}

//-----
function TFormMain.striae_is_azpl ( const striae : string;
                                   const faz, fpl : single;
                                   var az, pl : single;
                                   var misfit : single ) : boolean;
var
  pos : integer;
  pole : Tvector; // upper hemisphere
  striv : Tvector; // parallel to striae
  projected_striv : Tvector;
begin
  pos := AnsiPos ('/', striae);
  az := StrToFloat ( Copy (striae, 1, pos-1) );
  pl := StrToFloat ( Copy (striae, pos+1, length (striae) - pos) );
  if pl > 90 then result := False;
  if pl < 0 then result := False;

  {determine misfit angle}
  striv := azpl_to_unit_vector (az, pl);
  pole := azpl_to_unit_vector (faz, -90+fpl);
  projected_striv := normalize_vector (
    project_vector_onto_plane ( striv, pole )
  );
  misfit := RadToDeg ( arccos (dotprod (striv, projected_striv)) );

  result := True;
end; {function}

//-----
function TFormMain.striae_is_rake_only ( const striae : string;
                                          const faz, fpl : single;
                                          var az, pl : single;
                                          var misfit : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
  rake : single;
  direc : char;
  er : integer; // error flag
  strike : Tspherical; // strike.lon = phi of Euler angle
  R : Tmatrix; // inverse of Q that maps coef.s from 0123 to 01'2'3'
  one_dash : Tvector;
begin
  misfit := NoMisfit; // flag
  {decompose striae into blocks}
  if IsCharAlpha (striae[3]) then begin // striae is one-digit number, e.g., R8W
    val ( striae[2], rake, er );
    direc := striae[3];
  end else begin // striae is two-digit number, e.g., R23W
    val ( Copy (striae, 2, 2), rake, er );
    direc := striae[4];
  end; {if}

```

```

if er > 0 then goto End_of_this_function_and_return_FALSE;
if rake>90 then goto End_of_this_function_and_return_FALSE;
if rake<0 then goto End_of_this_function_and_return_FALSE;
if not (direc in CharSet_NESW)
    then goto End_of_this_function_and_return_FALSE;

if (0<=faz) and (faz<90) then // dipping to NE quadrant
    if (direc = 'N') or (direc = 'W') then
        strike := azpl_to_spherical ( faz-90, 0 )
    else begin
        strike := azpl_to_spherical ( faz+90, 0 )
    end; ifl;
if (90<=faz) and (faz<180) then // dipping to SE quadrant
    if (direc = 'N') or (direc = 'E') then
        strike := azpl_to_spherical ( faz-90, 0 )
    else begin
        strike := azpl_to_spherical ( faz+90, 0 )
    end; ifl;
if (180<=faz) and (faz<270) then // dipping to SW quadrant
    if (direc = 'S') or (direc = 'E') then
        strike := azpl_to_spherical ( faz-90, 0 )
    else begin
        strike := azpl_to_spherical ( faz+90, 0 )
    end; ifl;
if (270<=faz) and (faz<360) then // dipping to NW quadrant
    if (direc = 'S') or (direc = 'W') then
        strike := azpl_to_spherical ( faz+90, 0 )
    else begin
        strike := azpl_to_spherical ( faz-90, 0 )
    end; ifl;

//determine the direction of the base of 1' axis
R := orthogonal_tensor ( fpl, // theta of Euler angle
    strike.lon, // phi
    -rake // psi
);
one_dash[1] := R[1,1];
one_dash[2] := R[2,1];
one_dash[3] := R[3,1];
spherical_to_azpl ( unit_vector_to_spherical (one_dash) , az, pl );
az := amod ( az, 360.0 );

result := True;
Exit;

End_of_this_function_and_return_FALSE:
result := False;
end; ifunction!

//-----
function TFormMain.sense_is_OK ( const sense : string ) : boolean;
const CharSet_SENSE = {'N', 'R', 'S', 'D'}; CharSet_SENSE : TCharSet;
begin
    if sense[1] in CharSet_SENSE then
        result := True
    else begin
        result := False;
    end;
end; ifunction!

//-----
function TFormMain.fplane_is_OK ( const bedding : string;
    var az, pl : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
    bd : string; // bedding
    ns : byte; // length of string
begin
    bd := UpperCase ( bedding );
    ns := length ( bd );
    if ns = 0 then begin // bd is empty
        result := False;
        exit;
    end; ifl;

    if Copy ( bd,1,2 ) = 'NS' then begin // e.g., NS73E, NS4W
        result := process_NS ( bd, az, pl );
        Exit;
    end; ifl;

    if Copy ( bd,1,2 ) = 'EW' then begin // e.g., EW73N, EW8S
        result := process_EW ( bd, az, pl );
        Exit;
    end; ifl;

    result := process_N ( bd, az, pl );
end; ifunction!

//-----
function TFormMain.process_NS ( const bedding : string;
    var az, pl : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
    bd : string;
    ns : byte;
    buf : string;
    errorcode : integer;
begin // e.g., NS73E, NS4W
    bd := bedding;
    ns := length ( bd );
    if
        Copy ( bd,ns,1 ) = 'E' then az := 90
        else if Copy ( bd,ns,1 ) = 'W' then az := 270
        else
            goto End_of_this_function_and_return_FALSE;
    buf := Copy ( bd,3,ns-3 );
    if (length ( buf ) > 3)
    or (length ( buf ) = 0) then goto End_of_this_function_and_return_FALSE;
    val ( buf, pl, errorcode );
    if errorcode>0 then goto End_of_this_function_and_return_FALSE;
    if (pl<0) or (pl>180) then goto End_of_this_function_and_return_FALSE;
    result := True;
    Exit;

End_of_this_function_and_return_FALSE:
result := False;
end; ifunction!

//-----
function TFormMain.process_EW ( const bedding : string;
    var az, pl : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
    bd : string;
    ns : byte;
    buf : string;
    errorcode : integer;
begin
    bd := bedding;
    ns := length ( bd );
    if
        Copy ( bd,ns,1 ) = 'N' then az := 0
        else if Copy ( bd,ns,1 ) = 'S' then az := 180
        else goto End_of_this_function_and_return_FALSE;
    buf := Copy ( bd,3,ns-3 );
    if (length ( buf ) > 3) or (length ( buf ) = 0) then
        goto End_of_this_function_and_return_FALSE;
    val ( buf, pl, errorcode );
    if errorcode>0 then goto End_of_this_function_and_return_FALSE;
    if (pl<0) or (pl>180) then goto End_of_this_function_and_return_FALSE;
    result := True;
    Exit;

End_of_this_function_and_return_FALSE:
result := False;
end; ifunction!

//-----
function TFormMain.process_N ( const bedding : string;
    var az, pl : single ) : boolean;
label End_of_this_function_and_return_FALSE;
var
    bd : string;
    strike, dip : string;
    direc1 : char;
    direc2 : string; // N, E, S, W
    ns : integer;
    er : integer;
begin
    er := 0; direc1 := '0'; // initialization

    bd := bedding;
    ns := length ( bd );
    if ns>10 then // abnormally long string
        goto End_of_this_function_and_return_FALSE;

    try
        //decompose strike part
        if IsCharAlpha ( bd[3] ) then begin // strike is a one-digit number
            strike := bd[2];
            direc1 := bd[3];
            bd := Copy ( bd,4,ns-3 );
        end else begin // strike is a two-digit number
            strike := Copy ( bd,2,2 );
            direc1 := bd[4];
            bd := Copy ( bd,5,ns-4 );
        end; ifl;
        //decompose dip part
        ns := length ( bd );
        if IsCharAlpha ( bd[2] ) then begin // dip is a one-digit number
            dip := bd[1];
            direc2 := Copy ( bd,2,ns-1 ); // dip is a two-digit number
        end else begin
            dip := Copy ( bd,1,2 );
            direc2 := Copy ( bd,3,ns-2 );
        end; ifl;
        if bd[ns] = 'V' then begin
            dip := '90';
            direc2 := 'V';
        end; ifl;
    except
        on EAccessViolation do er := 1;
    end; iftry!
    if er = 1 then goto End_of_this_function_and_return_FALSE;

    val ( strike, az, er );
    if er > 0 then goto End_of_this_function_and_return_FALSE;

    if direc2='V' then begin
        pl := 90;
        case direc1 of
            'E': az := 270 + az;
            'W': az := 90 - az;
        end;
    end;
end; ifunction!

```

```

else goto End_of_this_function_and_return_FALSE;
end; [case]
result := True;
Exit;
end; [if]

val ( dip, pl, er );
if er > 0 then goto End_of_this_function_and_return_FALSE;
if (az<0) or (az>90) then goto End_of_this_function_and_return_FALSE;
if (pl<0) or (pl>180) then goto End_of_this_function_and_return_FALSE;

case direc1 of
'E': begin
  if (direc2='S') or (direc2='SE') or (direc2='E') then
    az := 90 + az // dipping to SE quadrant
  else
    if (direc2='N') or (direc2='NW') or (direc2='W') then
      az := 270 + az // dipping to NW quadrant
    else
      goto End_of_this_function_and_return_FALSE; end of if-block
    end;
  'W': begin
    if (direc2='S') or (direc2='SW') or (direc2='W') then
      az := 270 - az // dipping to SW quadrant
    else
      if (direc2='N') or (direc2='NE') or (direc2='E') then
        az := 90 - az // dipping to NE quadrant
      else
        goto End_of_this_function_and_return_FALSE; end of if-block
      end;
    else
      goto End_of_this_function_and_return_FALSE;
    end; [case]

az := amod ( az + 720, 360 );
result := True;
Exit;

End_of_this_function_and_return_FALSE;
result := False;
end; [function]

//-----
function TFormMain.matrix_times_vector (const a: Tmatrix; const x: Tvector) : Tvector;
begin
  result[1] := a[1,1]*x[1] + a[1,2]*x[2] + a[1,3]*x[3];
  result[2] := a[2,1]*x[1] + a[2,2]*x[2] + a[2,3]*x[3];
  result[3] := a[3,1]*x[1] + a[3,2]*x[2] + a[3,3]*x[3];
end; [function]

//-----
function TFormMain.orthogonal_tensor ( const theta, phi, psi: single ) : Tmatrix;
var
  sin_tht, cos_tht : single; // cosine and sine
  sin_phi, cos_phi : single; // of Euler
  sin_psi, cos_psi : single; // angles
begin
  sin_tht := Sin ( DegToRad (theta) ); cos_tht := Cos ( DegToRad (theta) );
  sin_phi := Sin ( DegToRad (phi) ); cos_phi := Cos ( DegToRad (phi) );
  sin_psi := Sin ( DegToRad (psi) ); cos_psi := Cos ( DegToRad (psi) );

  result[1,1] := cos_psi * cos_phi - sin_psi * cos_tht * sin_phi;
  result[2,1] := cos_psi * sin_phi + sin_psi * cos_tht * cos_phi;
  result[3,1] := sin_psi * sin_tht;

  result[1,2] := -sin_psi * cos_phi - cos_psi * cos_tht * sin_phi;
  result[2,2] := -sin_psi * sin_phi + cos_psi * cos_tht * cos_phi;
  result[3,2] := cos_psi * sin_tht;

  result[1,3] := sin_tht * sin_phi;
  result[2,3] := -sin_tht * cos_phi;
  result[3,3] := cos_tht;
end; [function]

//-----
function TFormMain.axpl_to_unit_vector (const az, pl: single) : Tvector;
var x : Tspherical; // spherical coordinates
begin
  x := axpl_to_spherical (az, pl);
  x.col := DegToRad ( x.col ); // colatitude z-axis points to zenith
  x.lon := DegToRad ( x.lon ); // longitude x-axis points to east
  result[1] := sin (x.col) * cos (x.lon);
  result[2] := sin (x.col) * sin (x.lon);
  result[3] := cos (x.col);
end; [function]

//-----
function TFormMain.axpl_to_spherical (const az, pl: single) : Tspherical;
begin
  result.col := 90 + pl; // colatitude z-axis points to zenith
  result.lon := 90 - az; // longitude x-axis points to east
end; [function]

//-----
function TFormMain.unit_vector_to_spherical (const v: Tvector) : Tspherical;
const eps = 1.0E-10;
var r : single;
begin
  r := sqrt ( sqr (v[1]) + sqr (v[2]) + sqr (v[3]) );
  if r < eps then begin // v is zero vector
    result.col := -1.0;
    result.lon := -1.0;
  end else begin // not zero vector
    result.col := RadToDeg ( arccos ( v[3] / r ) );
    result.lon := RadToDeg ( arctan2 ( v[2], v[1] ) );
  end; [function]

//-----
procedure TFormMain.spherical_to_axpl (const p: Tspherical; var az, pl: single);
begin
  az := 90 - p.lon;
  pl := p.col - 90;
end; [procedure]

//-----
function TFormMain.dotprod (const a, b: Tvector) : single;
begin
  result := a[1]*b[1] + a[2]*b[2] + a[3]*b[3];
end; [function]

//-----
function TFormMain.normalize_vector (const x: Tvector) : Tvector;
var r : single;
begin
  r := sqrt (dotprod (x,x));
  if r < 1.0E-8 then begin
    result[1] := 0; result[2] := 0; result[3] := 0;
  end else begin
    result[1] := x[1]/r;
    result[2] := x[2]/r;
    result[3] := x[3]/r;
  end; [if]
end; [function]

//-----
function TFormMain.project_vector_onto_plane (const v : Tvector; // object vector
const n : Tvector // normal to plane
) : Tvector;
var
  projector : Tmatrix;
  unitvector : Tvector;
  i, j : byte;
begin
  unitvector := normalize_vector ( n );
  for i := 1 to 3 do begin
    for j := 1 to 3 do projector[i,j] := - unitvector[i] * unitvector[j];
    projector[i,i] := 1.0 + projector[i,i];
  end; [for i]
  result := matrix_times_vector ( projector, v );
end; [function]

//-----
// Purpose : Returns the remainder when the first argument is divided by the
// second argument.
function TFormMain.amod (const x, y: single) : single;
begin
  result := x + 2 * y;
  result := result - trunc ( result / y ) * y;
end; [function]

//-----
// Purpose : round off figures
function TFormMain.roundf (const x: single; const digit: byte) : single;
var dd : extended;
begin
  dd := power (10, digit);
  result := round ( dd * x ) / dd;
end; [function]

//-----
// Purpose : Calculate angular distance of orientations of which azimuths are
// specified.
function TFormMain.azimuthal_distance (const x, y: single) : single; // degrees
begin
  result := abs (x-y);
  if result > 180 then result := 360 - result;
end; [function]

//-----
end.

```

付録B バージョン情報表示/非表示制御ルーチン (About.pas)

```
unit About;
|
|   Event handler for the about box of FDCConverter.
|   1st version : January 2001 (Yamaji, A.)
|
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

type
  TFormAbout = class (TForm)
    Panell: TPanel;
    Image1: TImage;
    Button_OK: TButton;
    Memo1: TMemo;
    procedure Button_OKClick (Sender: TObject) ;
  private
  public
  end;

var
  FormAbout: TFormAbout;

implementation

{$R *.DPM}

procedure TFormAbout.Button_OKClick (Sender: TObject) ;
begin
  FormAbout.Visible := False;
end; {procedure}

end.
15
```