CHAPTER
ONE

Problem solving in structural geology

1.1 OBJECTIVES OF STRUCTURAL ANALYSIS

In structural analysis, a fundamental objective is to describe as accurately as possible the geological structures in which we are interested. Commonly, we want to quantify three types of observations.

Orientations are the angles that describe how a line or plane is positioned in space. We commonly use either *strike* and *true dip* or true dip and *dip direction* to define planes, and *trend* and *plunge* for the orientations of lines (Fig. 1.1). The trend of the true dip is always at 90° to the strike, but the true dip is not the only angle that we can measure between the plane and the horizontal. An *apparent dip* is any angle between the plane and the horizontal that is not measured perpendicular to strike. For example, the angle labeled "plunge" in Figure 1.1 is also an apparent dip because line A lies in the gray plane. Strike, dip direction, and trend are all horizontal azimuths, usually measured with respect to the geographic north pole of the Earth. Dip and plunge are vertical angles measured downwards from the horizontal. Where a line lies in an inclined plane, we also use a measure known as the *rake* or the *pitch*, which is the angle between the strike direction and the line. There are few things more fundamental to structural geology than the accurate description of these quantities.

Whereas orientations are described using angles only, *magnitudes* describe how big, or small, the quantity of interest is. Magnitudes are, essentially, dimensions and thus have units of length, area, or volume. Some examples of magnitudes include the amplitude of a fold, the thickness of a bed, the length of a stretched cobble in a deformed conglomerate, the area of rupture during an earthquake, or the width of a vein. With magnitudes, size matters, whereas with orientations it does not.

The third type of observation compares both orientation and magnitude of something at two different times. The difference between an initial and a final state is known as *deformation*. Determining deformation involves measuring the feature in the final state and making

Cambridge University Press
978-1-107-01200-4 — Structural Geology Algorithms
Richard W. Allmendinger , Nestor Cardozo , Donald M. Fisher
Excerpt
More Information

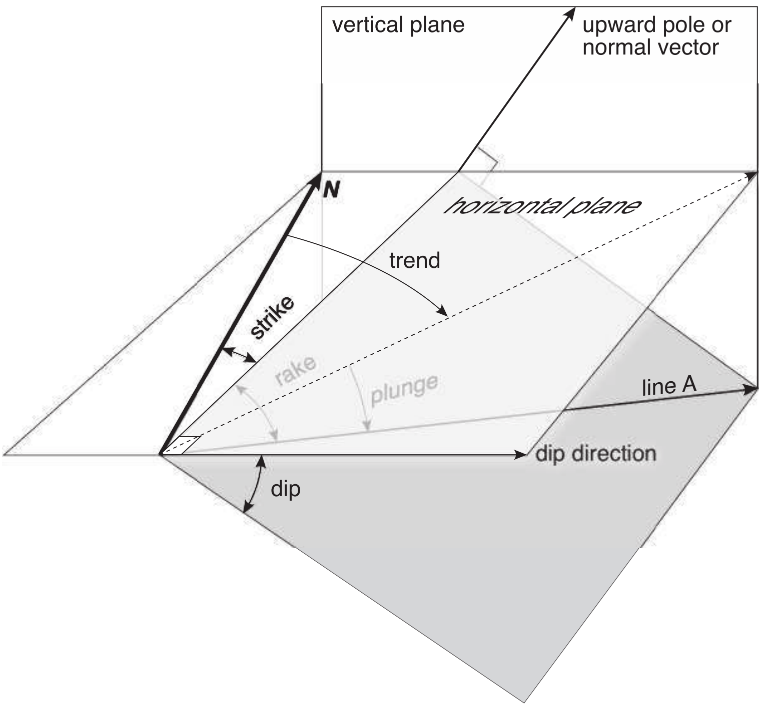


Figure 1.1 Three-dimensional perspective diagram showing the definition of typical structural geology terms. Strike and dip give the orientation of the gray plane with respect to geographic north (N) and the horizontal. Trend and plunge describe the orientation of line A. Because line A lies within the gray plane, we can specify the rake, the angle that the line makes with respect to the strike of the plane. The pole or normal vector is perpendicular to the plane. Note that because dip and plunge are measured from the horizontal, there is an implicit sign convention that down is positive and up negative.

inferences about its size, position, and orientation in the initial state. Deformation is commonly broken down into translation, rotation, and strain (or distortion) and each can be analyzed separately (Fig. 1.2), although when strains are large the sequence in which those effects are analyzed is important.

To determine orientations, magnitudes, or deformations, we need to make measurements. All measurements have some degree of *uncertainty*: is the length of that deformed cobble 10.0 or 10.3 cm? Is the strike of bedding on the limb of a fold 047° or 052°? In structural geology, the measurements that we make of natural, inherently irregular objects usually have a high degree of uncertainty. Typically, uncertainties, or *errors*, are estimated by making multiple measurements and averaging the result. However, we often want to calculate a quantity based on measurements of different quantities. *Error propagation* allows us to attach meaningful uncertainties to calculated quantities; this important operation is the subject of Chapter 12.

A complete structural analysis, of course, involves much more than just orientations, magnitudes, and deformations. These quantities tell us the “what” but not the “why.” They may tell us that the rocks surrounding pyrite grains and curved pressure shadows suffered a rotation of 37° and a stretch of 2, but they tell us nothing about why the deformation occurred nor, for example, why the rocks surrounding the pyrite changed shape continuously whereas the pyrite itself did not deform at all. Nor does the fact that a thrust belt was shortened

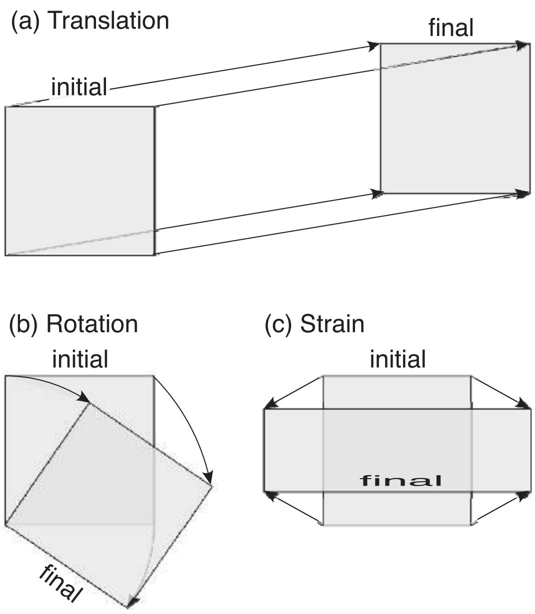


Figure 1.2 The three components of deformation – (a) translation, (b) rotation, and (c) strain – all require the comparison of an initial and a final state.

horizontally by 50% tell us anything about why the thrust belt formed in the first place. This complete understanding of structures is beyond the scope of this book, but the reader should never lose sight of the fact that accurate description based on measurements and their errors is just one aspect of a modern structural analysis.

1.2 ORTHOGRAPHIC PROJECTION AND PLANE TRIGONOMETRY

The methods we use to describe structures serve another purpose besides just providing an answer to a problem: They help us visualize complex, three-dimensional forms, thereby giving us a better intuitive understanding. Thus, many structural methods are graphical in nature, or are simple plane trigonometry solutions that have been derived from graphical constructions. Maps and cross sections constitute some of our most basic ways of graphically representing structural data and interpretations. Simpler graphical constructions using folding lines, front, side, and top views, etc. help us to visualize structures in three dimensions (Fig. 1.3). Until the 1980s, most structural geologists did not have knowledge of, or access to, the computing power needed to analyze complicated structural problems in any way except via graphical methods. Graphical methods, including spherical projection, were necessary to reduce complex three-dimensional geometries to two-dimensional sheets of paper.

Beginning structural geology students typically learn two types of graphical constructions: *orthographic* and *spherical* projections. In orthographic projection, one views the simple three-dimensional geometries as if they formed the sides of a box. Because one can only measure true angles with a protractor when looking perpendicularly down on the surface in which they occur, the sides of the box have to be unfolded before one can measure the angles of interest.

Consider the problem depicted in Figure 1.3: The gray plane has a strike, a true dip, δ, measured in a direction perpendicular to the strike, and an apparent dip, α, in a different direction. If one knows two out of the three quantities – the strike, true dip, and apparent dip – one can determine the third quantity. In orthographic projection, the true dip direction and the apparent dip direction are used as *folding lines*; they are literally like the creases on an unfolded

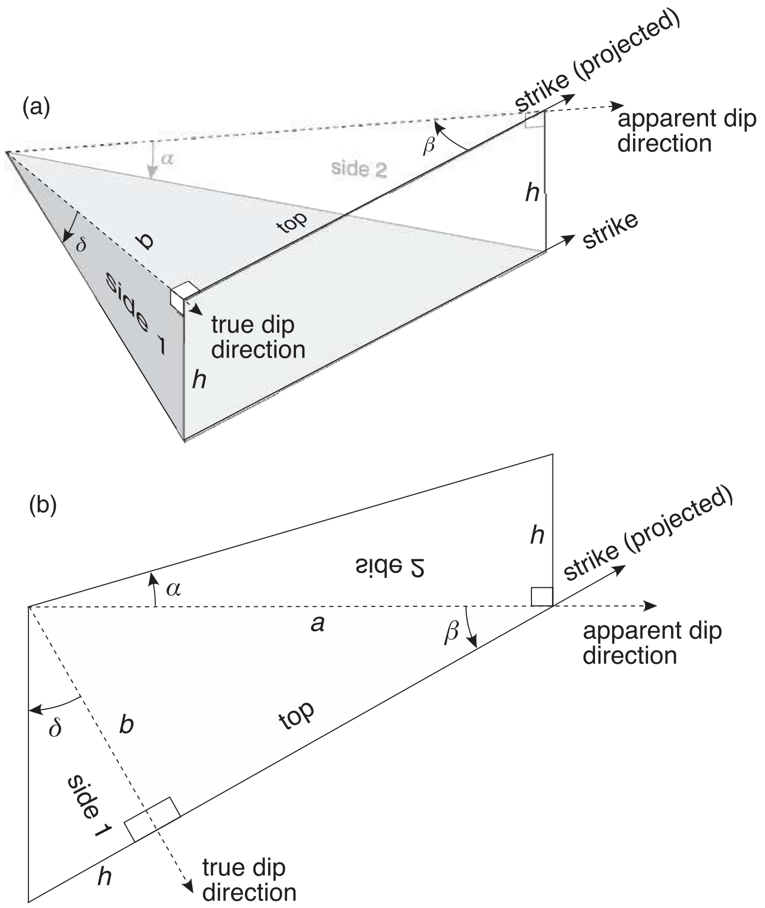


Figure 1.3 (a) Block diagram and (b) orthographic projection illustrating a graphical approach to the apparent dip problem. The dashed lines corresponding to the true and apparent dip directions are folding lines along which sides 1 and 2 have been folded up to lie in the same plane as the top of the block. h is the height of the block, which is the same everywhere along the strike line. δ, α, and β are the true dip, apparent dip, and angle between the strike and apparent dip directions, respectively.

cardboard box. By folding up the sides so that the top and the two sides all lie in the same plane, one can simply measure with a protractor whichever angle is needed.

The orthographic projection also provides the geometry necessary for deriving a simple trigonometric relationship that allows us to solve for the angle of interest by introducing a new angle from the top of the block (Fig. 1.3b): the angle between the strike and the apparent dip direction, β. Edge b of the top of the block is equal to

$$b = \frac{h}{\tan\delta}$$

The edge between the top and side 2, a is

$$a = \frac{b}{\sin\beta} = \frac{h}{\tan\delta\sin\beta}$$

And, from side 2 we get

$$a = \frac{h}{\tan \alpha}$$

Thus, using *plane trigonometry*, we can write the equation for the apparent dip:

$$\tan \delta \sin \beta = \tan \alpha \tag{1.1}$$

where δ is the true dip, β the angle between the strike and the apparent dip direction, and α the apparent dip. Plane trigonometry works very well for simple problems but is more cumbersome, or more likely impossible, for more complex problems.

A different approach, which has the flexibility to handle more difficult computations, is spherical trigonometry. To visualize this situation, imagine that the plane in which we are interested intersects the lower half of a sphere (Fig. 1.4) rather than a box. In general, with power comes complexity, and spherical trigonometry is no exception. To calculate the apparent

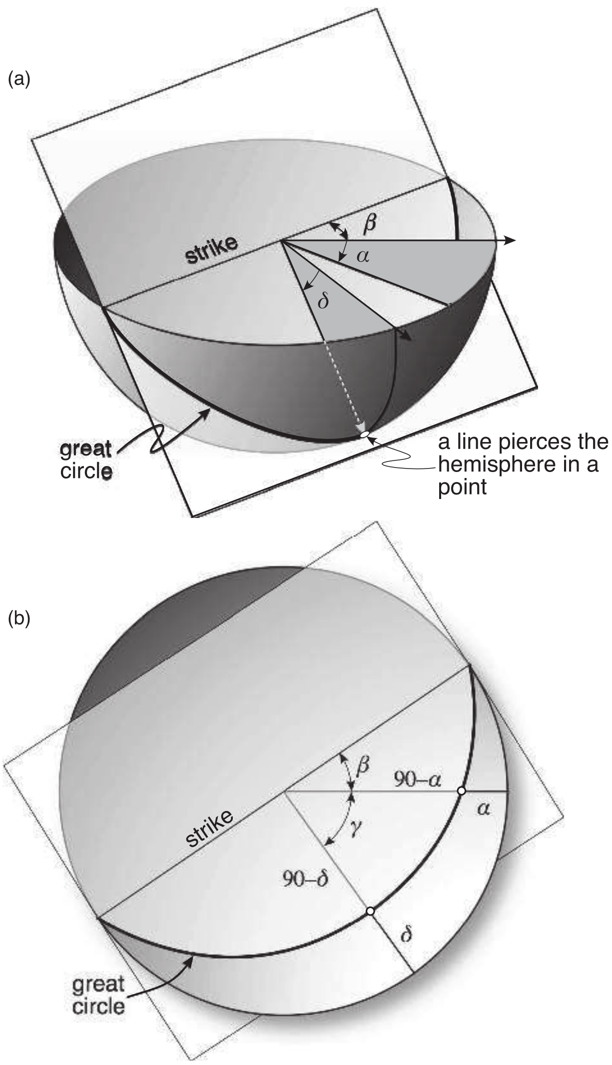


Figure 1.4 (a) Perspective view of a plane intersecting the lower half of a sphere. The angular relations are the same as those shown in Figure 1.3. The intersection of a sphere with any plane that goes through its center is a great circle. (b) Same geometry as in (a) but viewed from directly overhead as if one were looking down into the bowl of the lower hemisphere. View (b) was constructed using a stereographic projection. γ is the angle between true and apparent dip directions and other symbols are as in Figure 1.3.

dip, one must realize that, for the right spherical triangle shown (Fig. 1.4b), we know two angles (γ, which is the difference between the true and apparent dip directions, and the angle 90 because it is a true dip) and the included side (90 – the true dip δ). Thus, we can calculate the other side of the triangle (90 – the apparent dip α) from the following equation:

$$\cos\gamma = \tan(90 - \delta) * \cot(90 - \alpha) \tag{1.2}$$

A problem with both trigonometric methods is that one must guard against a multitude of special cases such as taking the tangent of 90°, the sign changes associated with sine and cosine functions, etc. On a more basic level, they give one little insight into the physical nature of what it is we are trying to determine. For most people, they are merely formulas associated with a complex geometric construction. And, the mathematical solution to this problem bears no obvious relation to other, more complicated problems we might wish to solve in structural geology.

1.3 SOLVING PROBLEMS BY COMPUTATION

One of the primary purposes of this book is to show you how to solve problems in structural geology by computation. There are many reasons for this emphasis: As a practicing geologist, you will use computer programs written by other people most of your professional life, so you should know how those programs work. Furthermore, computation is an important skill for any modern research scientist and allows you to solve problems that others cannot. Most importantly, the language of computation is linear algebra, and linear algebra is fundamental to developing a complete understanding of structures and continuum mechanics.

There are lots of different choices of computer platform and language that one could make. Perhaps simplest would be the humble spreadsheet program. In fact, many of the calculations that we ask you to do early in the book can easily be done in a spreadsheet program without even using its programming language (Visual Basic in the case of the popular program Excel). However, when you get to more complicated programs, spreadsheets are inadequate. Most commercial software these days is written in C, C++, or a variety of other platforms. In those programs, implementing the interface – that is, the windows, menus, drawing, dialog boxes, and so on – commonly takes up 95% or more of the lines of code. In this book, however, we want you to focus on the scientific algorithms rather than the interface.

Thus, we have chosen to illustrate this approach using the commercial software package, MATLAB®. Many universities now teach computer science and scientific computing using MATLAB, and many research geologists use MATLAB as their computing platform of choice. Because MATLAB is an interpreted language, it removes much of the fussiness of traditional compiled languages such as FORTRAN, Pascal, and C among a myriad of others. MATLAB also allows you to get results conveniently without worrying about the interface. You will be introduced to MATLAB in the next section, so we wanted to say a few general words about programming and syntax here.

First, programming languages, including spreadsheets and MATLAB, do trigonometric calculations in radians, not degrees. The relationship between radians and degrees is

$$\begin{aligned} 1 \text{ radian} &= \frac{180^\circ}{\pi} = 57.295\,779\,513\,1^\circ \\ 1^\circ &= \frac{\pi}{180} = 0.017\,453\,292\,5 \text{ radians} \end{aligned} \tag{1.3}$$

The four points of the compass – N, E, S, and W – can be defined in radians quite easily:

$$\begin{aligned}
&\textit{North}\ 0^\circ = 0 \text{ radians} = 360^\circ = 2\pi \text{ radians}\\
&\textit{East} \quad = 90^\circ = \frac{\pi}{2}\text{radians}\\
&\textit{South} \quad = 180^\circ = \pi \text{ radians}\\
&\textit{West} \quad = 270^\circ = \frac{3\pi}{2}\text{radians}
\end{aligned} \tag{1.4}$$

Second, any good computer code should have explanatory comments that tell the reader what the program is doing and why. Comments are for humans and are totally ignored by the computer. In all computer languages, a special character precedes comments; in MATLAB, that character is `%`, the percent character. We have tried to use comments liberally in this book to help you understand what is going on in the functions we provide.

In all computer programs, the things to be calculated are held in *variables*. Variables can hold a single number, but they can also hold more complicated groups of numbers called *arrays*. The best way to think about arrays is that they represent a list of related data (in one dimension) or a table of related data in two dimensions. Mathematically, arrays are matrices. When one has their data in an array, repetitive calculations can be made very easily via what are known as *loops*. Let's say we need to add together 25 random numbers. We could write

```
x1 + x2 + x3 + x4 + x5 + x6 + x7 + ··· + x22 + x23 + x24 + x25
```

Alternatively, one can do this calculation using an array and a loop:

```
x = randn(1,25); %x is an array of 25 random numbers
Sum = 0; %Initialize a variable to hold the sum of the array elements
for i=1:25 %Start of the loop. i starts at 1 and ends at 25
    Sum = Sum + x(i); %Add the current value x(i) to Sum
end %End of the loop
```

We will see later on in the book that the arrays and loops are what make the marriage of computing and linear algebra so seamless. Though the above example is trivial, arrays and loops will really help when we get to something like a tensor transformation that involves nine equations with nine terms each!

In computer programs, we can also select at run-time which operations or block of code are executed. We do this through the `if` control statement. Suppose we want to add the even but subtract the odd elements of array **x**. We can do this by modifying the loop above as follows:

```
for i=1:25 %Start of the loop. i starts at 1 and ends at 25
    if rem(i,2) == 0 %Start if statement. If remainder i/2=zero (i.e., even)
        Sum = Sum + x(i); %Add even element to Sum
    else %Else if odd element
        Sum = Sum - x(i); %Subtract odd element from Sum
    end %End of if statement
end %End of the loop
```

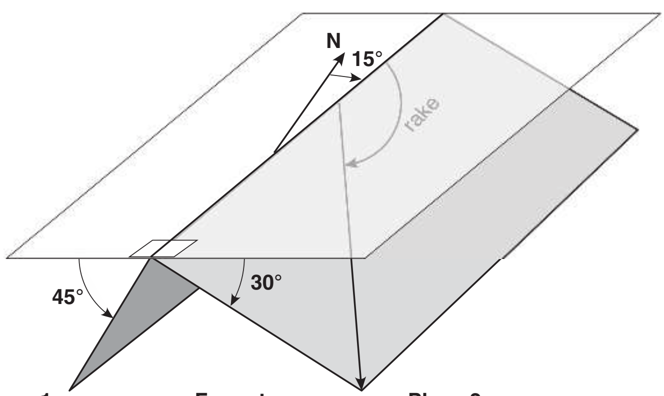
Finally (for now), many multi-step calculations are repeatedly used in a variety of contexts. Just as the tangent is used in both Equations 1.1 and 1.2, you can imagine more complicated calculations being used multiple times with different values. All programming languages have a variety of built-in functions, including trigonometric functions. The above code snippets use two such built-in functions: **`randn`**, which assigns random numbers to the array **`x`**, and **`rem`**, which determines the remainder of a division by an integer. Programming makes it easy to write your code in modular snippets that can be reused. You will see multiple examples in this book where one chunk of code, called a *function* in MATLAB and a function or *subroutine* in other languages, calls another chunk of code. Table 1.1 lists all of the MATLAB functions written especially for this book and shows which functions call, or are called by, other functions. All the functions follow the MATLAB help syntax. To get information about one of the functions, for example function **`StCoordLine`**, just type in MATLAB: `help StCoordLine`

1.4 SPHERICAL PROJECTIONS

The image in Figure 1.4b is known as a *spherical projection*, which is an elegant way of representing angular relationships on a sphere on a two-dimensional piece of paper. It should not be surprising that spherical projections are closely related to map projections, with the exception that in structural geology we use the lower hemisphere, as shown in Figure 1.4, whereas map projections use the upper hemisphere. Spherical projections are one of the most published types of plots in structural geology. They are used to carry out angular calculations such as rotations, apparent dip problems, and so on, as well as to present orientation data in papers and reports. Visualizing "stereonets," as they are commonly called, is one of the most important tasks a structural geology student can learn.

1.4.1 Data formats in spherical coordinates

Before diving in to stereonets, however, we need to examine briefly how orientations are generally specified in spherical coordinates (Fig. 1.5). In North America, planes are commonly recorded using their strike and dip. But, the strike can correspond to either of two



Plane 1	Format	Plane 2
N 15 E, 45 W	Quadrant	N 15 E, 30 E
015, 45 W	Azimuth	015, 30 E
195, 45	Right-hand Rule	015, 30
285, 45	Dip direction & Dip	105, 30

Figure 1.5 Common data formats for two planes that share the same strike but dip in opposite directions. Plane 1 is dark gray and plane 2 light gray. We do not recommend the quadrant format!

Cambridge University Press
978-1-107-01200-4 — Structural Geology Algorithms
Richard W. Allmendinger , Nestor Cardozo , Donald M. Fisher
Excerpt
More Information

Chapter	Function	Description	Called by	Call
1	`StCoordLine`	Coordinates of a line in an equal angle or equal area stereonet of unit radius	`GreatCircle, SmallCircle, Bingham, PTAxes`	Zer
1	`ZeroTwoPi`	Constrains azimuth to lie between 0 and 2 radians	`StCoordLine, CartToSph, Pole, SmallCircle, GeogrToView, Bingham, InfStrain`	
2	`SphToCart`	Converts from spherical to Cartesian coordinates	`CalcMV, Angles, Pole, Rotate, GeogrToView, Bingham, Cauchy, DirCosAxes, PTAxes`	
2	`CartToSph`	Converts from Cartesian to spherical coordinates	`CalcMV, Angles, Pole, Rotate, GeogrToView, Bingham, PrincipalStress, ShearOnPlane, InfStrain, PTAxes, FinStrain`	Zer
2	`CalcMV`	Calculates the mean vector for a given series of lines		Sph
2	`Angles`	Calculates the angles between two lines, between two planes, etc.		Sph
2	`Pole`	Returns the pole to a plane or the plane that correspond to a pole	`Angles, GreatCircle, Stereonet`	Zer Ca
3	`DownPlunge`	Constructs the down plunge projection of a bed		
3	`Rotate`	Rotates a line by performing a coordinate transformation	`GreatCircle, SmallCircle`	Sph
3	`GreatCircle`	Computes the great circle path of a plane in an equal angle or equal area stereonet of unit radius	`Stereonet, Bingham, PTAxes`	StC
3	`SmallCircle`	Computes the paths of a small circle defined by its axis and cone angle, for an equal angle or equal area stereonet of unit radius	`Stereonet`	Zer Ro
3	`GeogrToView`	Transforms a line from NED to view direction	`Stereonet`	Zer C
3	`Stereonet`	Plots an equal angle or equal area stereonet of unit radius in any view direction	`Bingham, PTAxes`	Pol Sm
4	`MultMatrix`	Multiplies two conformable matrices		
4	`Transpose`	Calculates the transpose of a matrix		
4	`CalcCofac`	Calculates all of the cofactor elements for a 3×3 matrix	`Determinant`	
4	`Determinant`	Calculates the determinant and cofactors for a 3×3 matrix	`Invert`	Cal

Cambridge University Press
978-1-107-01200-4 — Structural Geology Algorithms
Richard W. Allmendinger , Nestor Cardozo , Donald M. Fisher
Excerpt
More Information

Chapter	Function	Description	Called by	Calls
4	`Invert`	Calculates the inverse of a 3 × 3 matrix		`Dete`
5	`Bingham`	Calculates and plots a cylindrical best fit to a pole distribution		`Zero` `St` `Gr`
6	`Cauchy`	Computes the tractions on an arbitrarily oriented plane	`ShearOnPlane`	`DirC`
6	`DirCosAxes`	Calculates the direction cosines of a right-handed, Cartesian coordinate system of any orientation	`Cauchy, PrincipalStress TransformStress`	`SphT`
6	`TransformStress`	Transforms a stress tensor from old to new coordinates		`DirC`
6	`PrincipalStress`	Calculates the principal stresses and their orientations	`ShearOnPlane`	`DirC`
6	`ShearOnPlane`	Calculates the direction and magnitudes of the normal and shear tractions on an arbitrarily oriented plane		`Prin` `Ca`
8	`InfStrain`	Computes infinitesimal strain from an input displacement gradient tensor	`GridStrain`	`Cart`
8	`PTAxes`	Computes the P and T axes from the orientation of fault planes and their slip vectors		`SphT` `Gr`
8	`GridStrain`	Computes the infinitesimal strain of a network of stations with displacements in x and y		`InfS`
9	`FinStrain`	Computes finite strain from an input displacement gradient tensor		`Cart`
10	`PureShear`	Computes displacement paths and progressive finite strain history for pure shear		
10	`SimpleShear`	Computes displacement paths and progressive finite strain history for simple shear		
10	`GeneralShear`	Computes displacement paths and progressive finite strain history for general shear		
10	`Fibers`	Determines the incremental and finite strain history of a fiber in a pressure shadow		
11	`FaultBendFold`	Plots the evolution of a simple step, Mode I fault-bend fold		`Supp`
11	`SuppeEquation`	Equation 11.8 for fault-bend folding	`FaultBendFold, FaultBendFold Growth`	
11	`SimilarFold`	Plots the evolution of a similar fold		
11	`FixedAxisFPF`	Plots the evolution of a simple step, fixed axis fault-propagation fold		
11	`ParallelFPF`			`Supp`

