CHAPTER

THREE

Transformations of coordinate axes and vectors

3.1 WHAT ARE TRANSFORMATIONS AND WHY ARE THEY IMPORTANT?

The word "transformation" looks imposing and mathematical but it is, in fact, quite a simple thing that we do commonly without thinking about it. Whenever we change coordinate systems, we do a coordinate transformation. Suppose we submit some samples of fossils and their locations in latitude, longitude, and elevation to a paleontologist for identification. The paleontologist writes back with the instructions that the locations in eastings and northings (i.e., UTM coordinates), not latitude and longitude, are required. Thus, a coordinate transformation is needed. This doesn't make us very happy because the change requires a long calculation that would be tedious to do by hand! In this chapter, we are interested only in transformations that can be precisely described mathematically, but one should realize that coordinate transformations are a very common thing. Basically, coordinate transformations are just another way of looking at the same thing.[1] In the above example, the specific numbers used to describe the location in the two coordinate systems are different but the physical location where the samples were collected has not changed. The change in numbers simply represents a change in the coordinate system not a change in the position or fundamental magnitude of the thing being described.

The concept of a transformation is very important and one with incredible power for a wide variety of structural applications. It is commonly necessary to look at a problem from two different points of view. For example, when studying continental drift (Fig. 3.1a), at least two different coordinate systems are commonly required, one in present-day geographic space and one attached to the continent at some time in the past when it was in a different place and orientation on the globe. Or, take the case of analysis of a fault (Fig. 3.1b). To understand what is

[1] Later in the book, we will use the word "transformation" in a different context to refer to changes brought about by deformation that takes place between an initial and a final state.

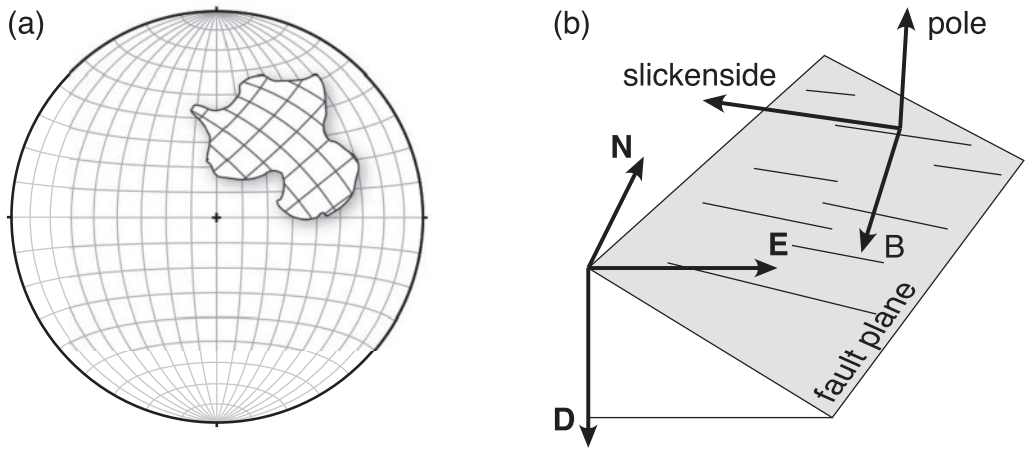


Figure 3.1 Examples of coordinate transformations in geology. (a) Continental drift; the continent has a coordinate system marked on it that corresponds to some time in the past. (b) A fault plane with two different coordinate systems.

going on from the perspective of the fault we need one coordinate system attached to the fault (e.g., with one axis perpendicular to the fault plane and another parallel to the slickensides on the fault). However, we also want to relate this to our more familiar geographic coordinate system; a transformation allows us to do that.

There is, however, an even more elemental reason for the importance of transformations. As intimated above, real, physical properties do not change when they are transformed from one coordinate system to another. As we will see in Chapter 5, this statement will be turned around to form the definition of an entire class of entities known as tensors. For right now, though, it is sufficient to be aware that the same thing can be described from many different viewpoints. Because the nature of something does not change when it is transformed, if we know its coordinates in one system we should be able to calculate its coordinates in any other system. This logic assumes that we know how the two coordinate systems are related to each other and that is our starting point.

3.2 TRANSFORMATION OF AXES

Before we can talk about transforming objects, we must consider the transformation that describes a change from one coordinate system to another. We will address only the change from one rectangular Cartesian coordinate system to another, which means the transformation is from one set of mutually perpendicular axes to another. As we will see in Section 3.2.3, this orthogonality makes our life very much easier.

3.2.1 Two-dimensional change in axes

The simplest type of transformation that you can think of is a two-dimensional shift or translation of axes without any rotation. Basically, we just establish the origin at a different place; it is simple to write equations that relate one set of axes to another. In the case of Figure 3.2,

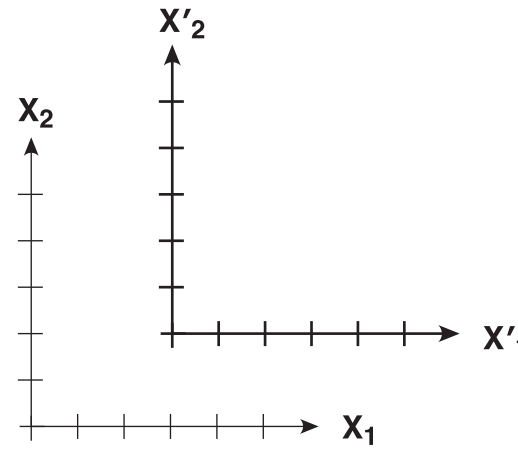


Figure 3.2 Translation of axes. The new axes are primed.

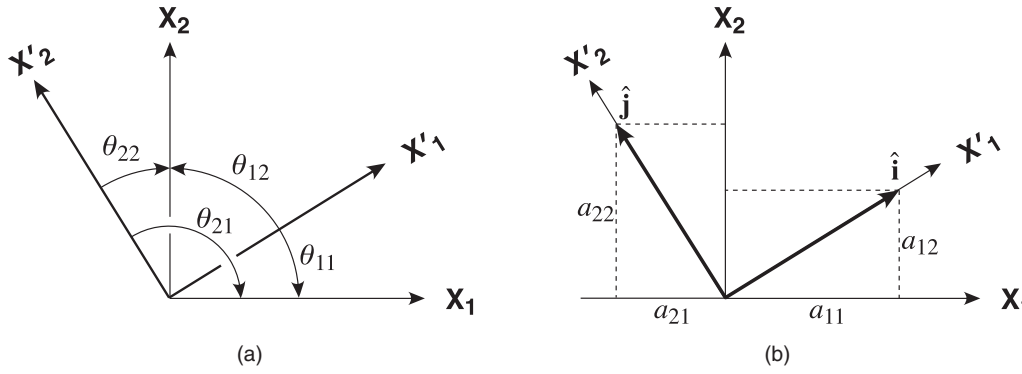


Figure 3.3 Rotation of axes in two dimensions. New axes are primed. (a) Shows the four angles, θ_{11}, θ_{12}, θ_{21}, and θ_{22}, that define the coordinate transformation. (b) Same transformation, but expressed in terms of base vectors and their direction cosines, a_{11}, a_{12}, a_{21}, and a_{22}.

$$\mathbf{X}'_1 = \mathbf{X}_1 - 3 \text{ and } \mathbf{X}'_2 = \mathbf{X}_2 - 2 \qquad \text{(new in terms of old)}$$

and

$$\mathbf{X}_1 = \mathbf{X}'_1 + 3 \text{ and } \mathbf{X}_2 = \mathbf{X}'_2 + 2 \qquad \text{(old in terms of new)}$$

We will come back to this example when we get to deformation (Chapter 7). Although this provides a useful starting point, it really doesn't provide any new information and therefore is not of great interest in our study of vectors. You can probably visualize that a vector will make the same angles with the axes in both the new and the old coordinates and, furthermore, the components of the vector will have the same magnitude in both coordinate systems. We have not really learned anything, yet.

More interesting is the case of rotation of a coordinate system. From Figure 3.3a you can see that, in two dimensions, there are four angles that define the transformation. Rather than give all of these a different letter, they are distinguished by double subscripts. As you can see by close inspection of the figure, the choice of subscripts is not arbitrary. The convention is that

the first subscript refers to the new (i.e., primed) axis, whereas the second subscript refers to the old (unprimed) axis. Thus, θ_{21} indicates the angle between the new, $\mathbf{X}'_2$ axis and the old, $\mathbf{X}_1$ axis.

Although there are, clearly, four angles, one can intuitively see that they are not all independent of each other. In fact, in two dimensions, we need only specify one of the four and the rest can be calculated from the first one. For example, $\theta_{11} = 90^\circ - \theta_{12}$, $\theta_{21} = 90^\circ + \theta_{22}$, etc. If we represent the axes by their base vectors, then you can see that the projection of the new axis onto the old axis is equal to the cosine of the angle between the two axes (Fig. 3.3b). For that reason, the relations between the two coordinate systems are commonly given in terms of the direction cosines between them: $a_{11} = \cos\theta_{11}$, $a_{12} = \cos\theta_{12}$, $a_{21} = \cos\theta_{21}$, and $a_{22} = \cos\theta_{22}$. By a simple application of the Pythagorean Theorem (see Fig. 2.4) and recalling that the length of a unit vector is 1 (i.e., $|\hat{i}| = 1$ in Fig. 3.3b), you can see that

$$a_{11}^2 + a_{12}^2 = 1 \text{ and } a_{21}^2 + a_{22}^2 = 1 \tag{3.1}$$

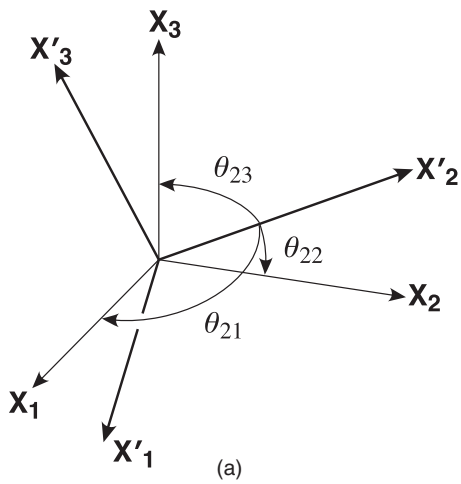
Furthermore, recall that the dot product of two perpendicular vectors is equal to zero (because the cosine of 90° is zero). Therefore, the dot product of the base vectors in the new system (Fig. 3.3) gives us a third constraint:

$$a_{11}a_{21} + a_{12}a_{22} = 0 \tag{3.2}$$

We have three equations, 3.1 and 3.2, and four unknowns, so only one of the direction cosines is independent. If you understand this two-dimensional case, extension to three dimensions is obvious.

3.2.2 Three-dimensional change in axes: The transformation matrix

The relations in three dimensions logically follow from those in two dimensions. There are three old axes and three new ones; hence, there will be nine angles that completely define the coordinate transformation (Fig. 3.4a). As before, we use double subscripts to identify the angles



Old axes

New axes	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$
$\mathbf{X}'_1$	a_{11}	a_{12}	a_{13}
$\mathbf{X}'_2$	a_{21}	a_{22}	a_{23}
$\mathbf{X}'_3$	a_{31}	a_{32}	a_{33}

(b)

Figure 3.4 (a) A general, three-dimensional coordinate transformation. The new axes are primed; the old axes are in black. Only three of the nine possible angles are shown. (b) Graphic device for remembering how the subscripts of the direction cosines relate to the new and the old axes.

and their direction cosines, with the first subscript referring to the new axis and the second to the old axis (Fig. 3.4b). As before, not all nine of these angles are independent. Just visually, you can see that, given θ_{22} and θ_{23}, the third angle, θ_{21}, is fixed. Intuitively, you may be able to see that, to completely constrain the transformation, only one other angle between any of the other two new axes and any of the old axes is needed.

The array of direction cosines in Figure 3.4b is known as the *transformation matrix*. It is commonly written:

$$a_{ij} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tag{3.3}$$

The way we have written Equation 3.3 uses some notation that we have not seen much of up to this point. We will see much more of matrices and indicial notation in the next chapter.

3.2.3 The orthogonality relations

Earlier, we began developing some general equations, 3.1 and 3.2, that described how the angles (or really their direction cosines) relate to one another. The development in three dimensions is an extension of the previous derivation. In three dimensions, the length of any vector is the square root of the sum of the squares of its three components (Eq. 2.3). If that vector is a unit vector, then the sum of the squares of the direction cosines will be equal to one. We showed in Section 3.2.1 that the components of the transformation matrix are nothing more than the direction cosines of the base vectors of the new coordinate system in the old coordinate system. Therefore, we can write the following three equations, which give the lengths (squared) of the three base vectors of the new coordinate system:

$$\begin{aligned} a_{11}^2 + a_{12}^2 + a_{13}^2 &= 1 \\ a_{21}^2 + a_{22}^2 + a_{23}^2 &= 1 \\ a_{31}^2 + a_{32}^2 + a_{33}^2 &= 1 \end{aligned} \tag{3.4}$$

Likewise, as stated above, the dot product of two perpendicular vectors is zero. Because each of the three base vectors of the new coordinate system is perpendicular to the others, we can write three additional equations:

$$\begin{aligned} a_{21}a_{31} + a_{22}a_{32} + a_{23}a_{33} &= 0 \\ a_{31}a_{11} + a_{32}a_{12} + a_{33}a_{13} &= 0 \\ a_{11}a_{21} + a_{12}a_{22} + a_{13}a_{23} &= 0 \end{aligned} \tag{3.5}$$

Equations 3.4 and 3.5 collectively form what are known as the *orthogonality relations*. Now, in three dimensions, we have six equations and nine unknowns (i.e., the nine direction cosines). This proves quantitatively what we already knew intuitively: There are only three independent direction cosines in the transformation matrix.

3.3 TRANSFORMATION OF VECTORS

Now that we have put the transformation of axes to rest, we'll look at something more practical: the transformation of vectors. As before, we'll start in two dimensions, where it is easier to get a feeling for the geometry. The two-dimensional transformation equations are derived by projecting the old components of the vector, v_1 and v_2, onto the new axes, $\mathbf{X}_1'$ and $\mathbf{X}_2'$. In Figure 3.5b,

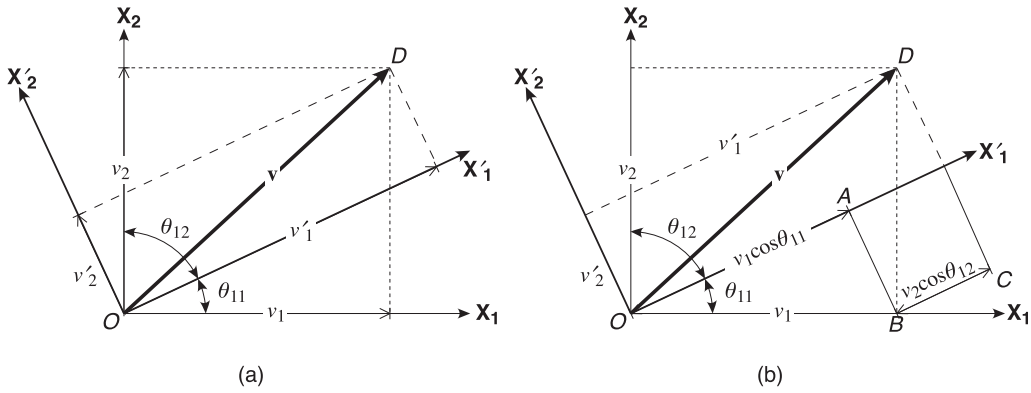


Figure 3.5 Transformation of vector **v** in two dimensions. (a) The components of the vector in the old coordinate system are v_1 and v_2; in the new coordinate system, the coordinates are v'_1 and v'_2. (b) Shows the geometry for deriving the v'_1 component of transformation equation (3.5) from triangles *OAB* and *BCD*.

you can see that v'_1 will be equal to the sum of line segments *OA* and *BC*, which can be calculated from the trigonometry of triangles *OAB* and *BCD* (Fig. 3.5b). We get

$$v'_1 = v_1 \cos\theta_{11} + v_2 \cos\theta_{12}$$

or, in terms of the direction cosines of the transformation matrix (and including without proof the equation for v'_2),

$$\begin{aligned} v'_1 &= v_1 a_{11} + v_2 a_{12} \\ v'_2 &= v_1 a_{21} + v_2 a_{22} \end{aligned} \tag{3.6}$$

Note that the above equations give the new components of the vector in terms of the old. By projecting the new components, v'_1 and v'_2, onto the old axes, $\mathbf{X_1}$ and $\mathbf{X_2}$, you can make the same geometric arguments and derive the reverse transformation, which is the old in terms of the new:

$$\begin{aligned} v_1 &= v'_1 a_{11} + v'_2 a_{21} \\ v_2 &= v'_1 a_{12} + v'_2 a_{22} \end{aligned} \tag{3.7}$$

There are some subtle, but important, changes between Equations 3.6 and 3.7. First, in the latter the primed components are on the right-hand side. Less obvious, but no less important, the positions of a_{12} and a_{21} have been switched or *transposed*. One of the nice things about vector algebra is that it is extremely symmetrical and logical!

Figure 3.6 shows a three-dimensional vector transformation. As before, notice that only the coordinates change, not the fundamental length or orientation of the vector, itself. Thus in Figure 3.6, $v_1 \neq v'_1$, $v_2 \neq v'_2$, and $v_3 \neq v'_3$ but the vector is just as long and points the same way in both coordinate systems.

The geometry in three dimensions is really the same as in two, only harder to visualize. Think about decomposing the vector into its three components parallel to the old axes, and then transforming those components along with the old axes into the new coordinate system. Thought of this way, the transformation of any vector is analogous to transforming the base vectors of the axes themselves, except that the components are not unit vectors. Three equations describe the three-dimensional vector transformation:

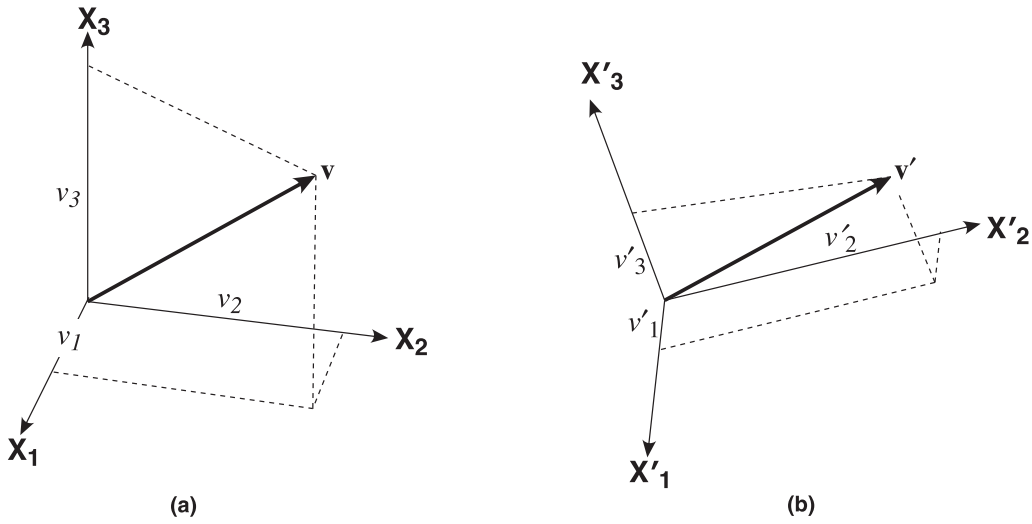


Figure 3.6 Vector **v** in two different coordinate systems. Note that the length and orientation of **v** on the page has not changed; only the axes have changed.

$$\begin{aligned} v'_1 &= a_{11}v_1 + a_{12}v_2 + a_{13}v_3 \\ v'_2 &= a_{21}v_1 + a_{22}v_2 + a_{23}v_3 \\ v'_3 &= a_{31}v_1 + a_{32}v_2 + a_{33}v_3 \end{aligned} \tag{3.8}$$

and the reverse transformation (old in terms of new):

$$\begin{aligned} v_1 &= a_{11}v'_1 + a_{21}v'_2 + a_{31}v'_3 \\ v_2 &= a_{12}v'_1 + a_{22}v'_2 + a_{32}v'_3 \\ v_3 &= a_{13}v'_1 + a_{23}v'_2 + a_{33}v'_3 \end{aligned} \tag{3.9}$$

Note that we have reversed the order of a and v in these equations from the earlier Equations 3.6 and 3.7, but that is perfectly okay because all terms are scalars. If you examine carefully Equations 3.8 and 3.9, it looks as though we have "flipped" the transformation matrix so that the rows are now columns and vice versa. Mathematically, "flipping" a matrix is known as taking the *transpose*, as we will see in a later chapter.

You can transform the coordinates of a point in space using the same equations that you would for vectors (3.8 and 3.9). That's because any point can be thought of as being connected to the origin of the coordinate system by a vector known as a *position vector*. The components of the vector are the same as the coordinates of the point. We will use this concept below in Section 3.4.1.

3.4 EXAMPLES OF TRANSFORMATIONS IN STRUCTURAL GEOLOGY

Generally, we give little thought to the fact that some of our most commonplace structural problems involve transformations of the type described in the previous section. That is because we are taught to do them with laborious manual methods, like orthographic projections, or on a stereonet. In this section, we will see how to solve two such problems using the methods developed in this chapter.

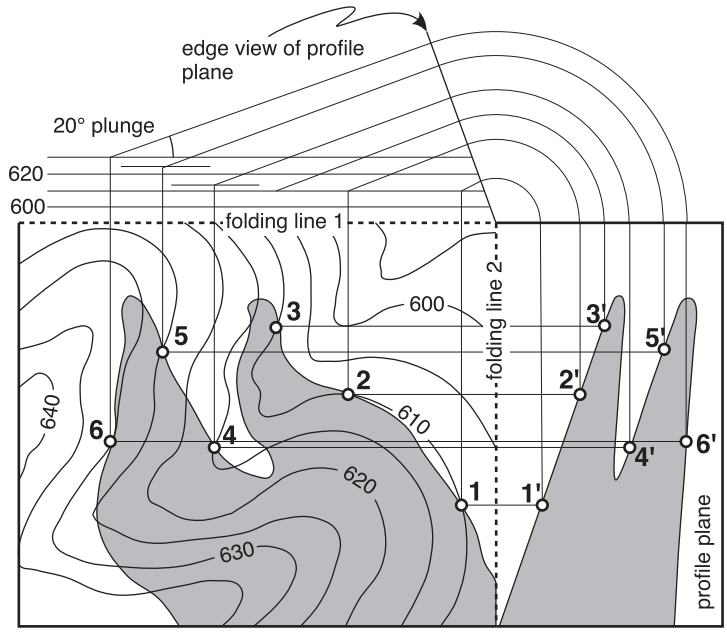


Figure 3.7 Graphical construction for drawing a down-plunge projection in a region with topography. The fold in this example plunges 20° east. The projection of six control points from the map onto the projection is shown. Modified from Ragan (2009, p. 461).

3.4.1 Down-plunge projection

To get the best sense of the "true" geometry of a cylindrical fold, geologists usually construct a *profile view* of the fold, a cross section of the fold perpendicular to the fold axis. When folds are cylindrical, all the points along all of their surfaces can be projected parallel to the fold axis onto this profile plane. This task is complicated by two facts: First, the surface of the Earth is irregular with hills and valleys and, second, folds commonly plunge oblique to the ground surface. The graphical method taught in virtually all structural geology lab manuals employs orthographic projection (Fig. 3.7). One chooses a horizontal folding line that is perpendicular to the fold axis and another that is horizontal and in the same plane as the fold axis. Then by swinging arcs with a compass and carefully drawing parallel straight lines you can construct the profile.

The construction is made more tedious by the fact that a separate folding line is needed for each elevation of the control points used. There is ample opportunity for error in the construction of the parallel lines as well as interpolation between widely separated control points.

There is, however, a different way of making a down-plunge projection that applies the methods we have seen earlier in this chapter. Specifically, we determine the geographic coordinates for a series of points along each bedding surface; that is, we digitize the bedding surfaces. Then we transform those points into the fold coordinate system (Fig. 3.8). All of this can be done on a computer much more rapidly than is possible by hand. Because we are dealing with geographic coordinates, our old, right-handed coordinate system will be $\mathbf{X_1}$ = east, $\mathbf{X_2}$ = north, and $\mathbf{X_3}$ = up (or elevation); our new, right-handed system will be as shown in Figure 3.8, with $\mathbf{X'_3}$ coinciding with the fold axis.

Now, we need to determine the transformation matrix in terms of the orientation (trend and plunge) of the fold axis. The angular relations are given in Figure 3.9. Some of the angles are

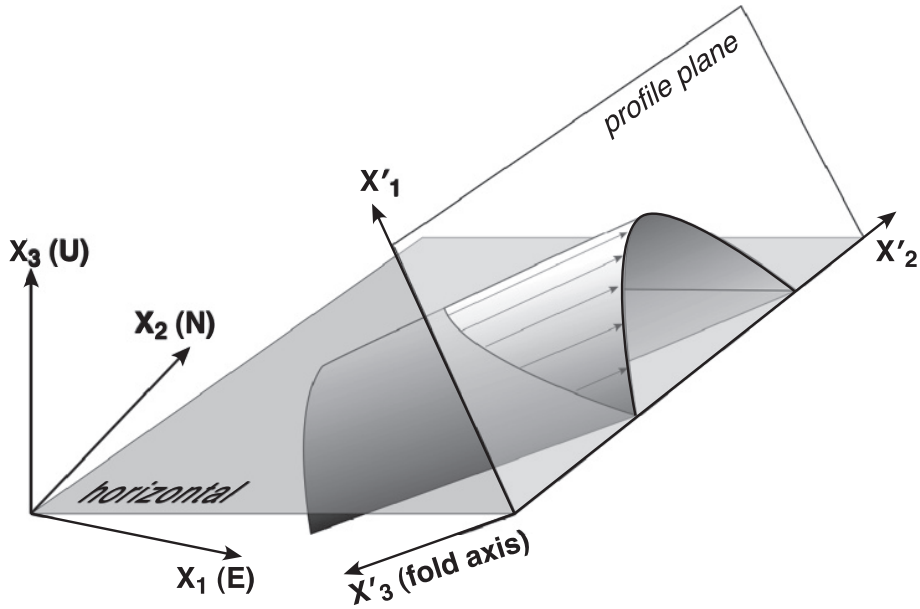
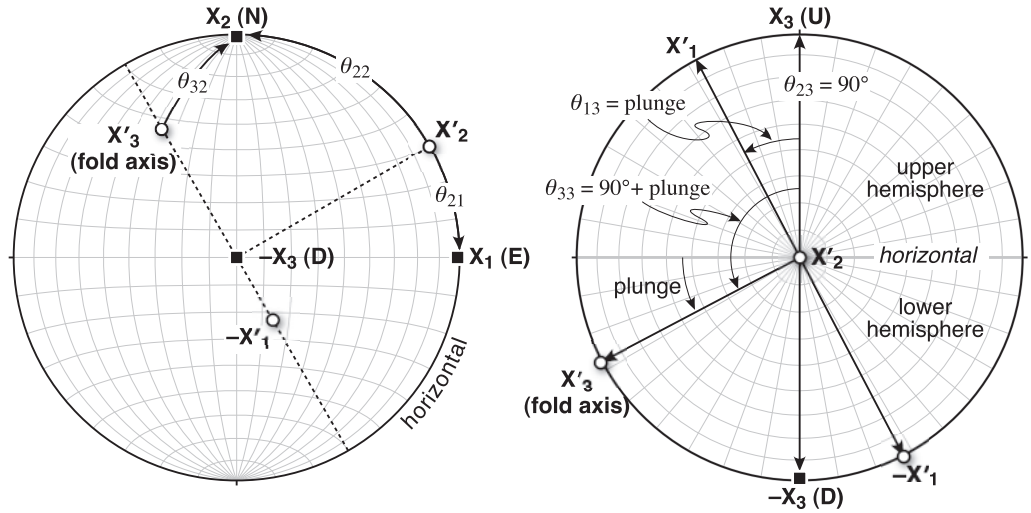


Figure 3.8 The down-plunge projection, showing the relation between its graphical construction and the right-hand coordinate system we will use below. The true profile plane is the plane that contains the $\mathbf{X}'_1$ and $\mathbf{X}'_2$ axes. The $\mathbf{X}'_2$ axis corresponds to a folding line in the orthographic projection technique shown in Figure 3.7.



(a) Horizontal (map) view (b) Vertical view

Figure 3.9 (a) Equal area lower hemisphere projection showing the angular relations between the two sets of axes in the down-plunge projection problem. ENU is the old coordinate system and the axes defined by the fold axis ($\mathbf{X}'_3$) are the new coordinate system. Several of the angles that define the coordinate transformation (θ_{21}, θ_{22}, etc.) are shown. (b) The same coordinate transformation viewed in a vertical plane that contains the trend and plunge of the fold axis.

obvious, as in the case of all of the new axes with respect to the old $\mathbf{X_3}$ axis. For example, from Figure 3.9b it is clear that the angle between the new $\mathbf{X'_3}$ and the old $\mathbf{X_3}$ axes is equal to the fold axis plunge plus 90°. The angle between the $\mathbf{X'_1}$ and $\mathbf{X_3}$ axes is equal to the fold axis plunge, itself, and the angle between $\mathbf{X'_2}$ and $\mathbf{X_3}$ is just 90°. Thus, in terms of the direction cosines we can write

$$a_{13} = \cos(plunge)$$
$$a_{23} = \cos(90) = 0$$
$$a_{33} = \cos(90 + plunge) = -\sin(plunge)$$

Notice that, because all of these are with respect to one old axis, they are not all independent. If we can determine one more angle, we could use the orthogonality relations to calculate the rest. In fact, it will be easier to determine all of the angles directly in this example. The angles that $\mathbf{X'_2}$ makes with the other two old axes are in a horizontal plane (Fig. 3.8) and therefore are just a function of the trend of the fold axis. Angle $(\mathbf{X'_2X_1}) = 360° - \text{trend}$, and $(\mathbf{X'_2X_2}) = \text{trend} - 270°$. This will give us two more direction cosines:

$$a_{21} = \cos(360 - trend) = \cos(trend)$$
$$a_{22} = \cos(trend - 270) = -\sin(trend)$$

The final direction cosines can be determined if we recall that they are nothing more than the direction cosines of the fold axis and its perpendicular ($\mathbf{X'_1}$) in an east-north-up coordinate system. Thus, we can use the relations in Table 2.1 and modify them for the change in coordinate system. The direction cosines with respect to north and east will not change because cos(–*plunge*) = cos(*plunge*). The cosine with respect to up will be equal to the –sin (*plunge*). Thus,

$$a_{31} = \sin(trend)\cos(plunge)$$
$$a_{32} = \cos(trend)\cos(plunge)$$

and the remaining direction cosines for the $\mathbf{X'_1}$ axis can be calculated by projecting its negative into the lower hemisphere and then multiplying by –1:

$$a_{11} = -\sin(trend + 180)\cos(90 - plunge) = \sin(trend)\sin(plunge)$$
$$a_{12} = -\cos(trend + 180)\cos(90 - plunge) = \cos(trend)\sin(plunge)$$

Thus, we can combine all of the above equations and write out the transformation in shorthand form, as in Equation 3.3:

$$a_{ij} = \begin{pmatrix} \sin(trend)\sin(plunge) & \cos(trend)\sin(plunge) & \cos(plunge) \\ \cos(trend) & -\sin(trend) & 0 \\ \sin(trend)\cos(plunge) & \cos(trend)\cos(plunge) & -\sin(plunge) \end{pmatrix} \tag{3.10}$$

Now, to accomplish the down-plunge projection, substitute the direction cosines from Equation 3.10 into Equations 3.8 and coordinates in the new coordinate system can be calculated. In the actual projection, the v'_3 component is ignored because everything will be projected onto the $\mathbf{X'_1X'_2}$ plane. After that, it's just a matter of connecting the dots! The following MATLAB® function **`DownPlunge`** does the transformations for the down-plunge projection of a bed but it does not plot or connect the dots!

```
function dpbedseg = DownPlunge(bedseg,trd,plg)
%DownPlunge constructs the down plunge projection of a bed
%
% [dpbedseg] = DownPlunge(bedseg,trd,plg) constructs the down plunge
% projection of a bed from the X1 (East), X2 (North),
```

```
% and X3 (Up) coordinates of points on the bed (bedseg) and the
% trend (trd) and plunge (plg) of the fold axis
%
% The array bedseg is a two-dimensional array of size npoints x 3
% which holds npoints on the digitized bed, each point defined by
% 3 coordinates: X1 = East, X2 = North, X3 = Up
%
% NOTE: Trend and plunge of fold axis should be entered in radians

%Number of points in bed
nvtex = size(bedseg,1);

%Allocate some arrays
a=zeros(3,3);
dpbedseg = zeros(size(bedseg));

%Calculate the transformation matrix a(i,j). The convention is that
%the first index refers to the new axis and the second to the old axis.
%The new coordinate system is with X3' parallel to the fold axis, X1'
%perpendicular to the fold axis and in the same vertical plane, and
%X2' perpendicular to the fold axis and parallel to the horizontal. See
%equation 3.10
a(1,1) = sin(trd)*sin(plg);
a(1,2) = cos(trd)*sin(plg);
a(1,3) = cos(plg);
a(2,1) = cos(trd);
a(2,2) = -sin(trd);
a(2,3) = 0.0;
a(3,1) = sin(trd)*cos(plg);
a(3,2) = cos(trd)*cos(plg);
a(3,3) = -sin(plg);

%The east, north, up coordinates of each point to be rotated already define
%the coordinates of vectors. Thus we don't need to convert them to
%direction cosines (and don't want to either because they are not unit vectors)
%The following nested do-loops perform the coordinate transformation on the
%bed. The details of this algorithm are described in Chapter 4
for nv = 1:nvtex
    for i = 1:3
        dpbedseg(nv,i) = 0.0;
        for j = 1:3
            dpbedseg(nv,i) = a(i,j)*bedseg(nv,j) + dpbedseg(nv,i);
        end
    end
end
end
```

For example, say you want to construct the down-plunge projection of the contact between the white and gray units in Figure 3.7. Digitize the contact, and in a text editor make a file with

the east, north, up coordinates of points on the contact, one point per line (coordinate entries can be separated by commas or spaces). Save this file as bedseg.txt. Now type in MATLAB:

```
load bedseg.txt; %Load bed
dpbedseg = DownPlunge(bedseg,90*pi/180,20*pi/180);% Down plunge projection
plot(dpbedseg(:,2),dpbedseg(:,1), 'k-'); %Plot bed
axis equal; %Make plot axes equal
```

You will get a chance to try this on a real structure in the exercises at the end of the chapter!

3.4.2 Rotation of orientation data

There are few operations more basic to structural geology than rotations. Unfolding lineations, paleomagnetic fold tests, and converting data measured on a thin section to its original geographic orientation all require rotations. The stereonet is a convenient graphic device for accomplishing rotations about a horizontal axis, but rotations about an inclined axis are more difficult. That is because points (lines) being rotated trace out small circles centered on the rotation axis. A stereonet only shows small circles centered on the horizontal. It can be done, but it is tedious.

A rotation is nothing more than a transformation of coordinate system and vectors. When we unfold linear elements, we are transforming from a geographic coordinate system to one pinned to bedding (or layering). Therefore, we should be able to use the mathematics developed in this chapter to determine the equations necessary to accomplish a general rotation about any axis in space. As before, we need to determine the transformation matrix that will allow us to transform the vectors representing our orientation measurements. The rotation axis is commonly specified by its trend and plunge, and the magnitude of rotation is given as an angle that is positive if the rotation is clockwise about the given axis (the old right-hand rule, again). The tricky part here is that the rotation axis does not generally coincide with the axes of either the new or the old coordinate system (unlike the previous example where the fold axis did define one of the new axes).

Ultimately we want to calculate the direction cosines for the transformation from the old axes to their new equivalents, rotated about the given rotation axis. Here we give the derivation for just one of the direction cosines, a_{22}; you can derive the rest yourself! In Figure 3.10, notice that, during the rotation, the $\mathbf{X_2}$ axis tracks along a small circle centered on the rotation axis. The size of the circle, or in three dimensions the half-apical angle of the cone, is equal to the angle between the rotation axis and $\mathbf{X_2}$, β. The angle between the new axis, $\mathbf{X'_2}$, and the rotation axis will also be β. Although, the points track along a small circle, the angle that we want to calculate is that between the new and old axes, θ_{22}, which is measured along a great circle (Fig. 3.10).

The simplest way to solve this problem is to use the law of cosines for spherical triangles. Notice that ω is the angle included between the two equal sides of the β–β–θ_{22} triangle (Fig. 3.10). Thus the appropriate formula to use is

$$\cos c = \cos a \cos b + \sin a \sin b \cos C$$

where $c = \theta_{22}$, $a = b = \beta$, and $C = \omega$. Substituting and rearranging, we get

$$\begin{aligned} a_{22} = \cos\theta_{22} &= \cos\beta\cos\beta + \sin\beta\sin\beta\cos\omega \\ &= \cos^2\beta + (1 - \cos^2\beta)\cos\omega \\ &= \cos\omega + \cos^2\beta(1 - \cos\omega) \end{aligned} \tag{3.11a}$$

By the same reasoning the direction cosines for $\mathbf{X_1}$–$\mathbf{X'_1}$ and $\mathbf{X_3}$–$\mathbf{X'_3}$ are

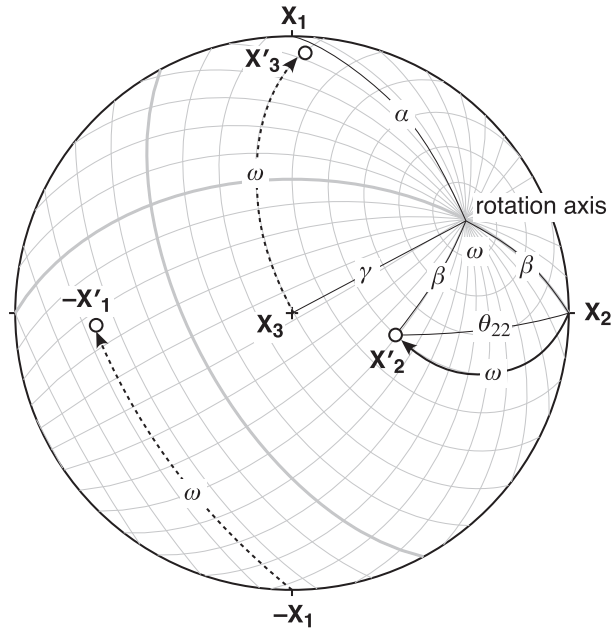


Figure 3.10 Lower hemisphere projection showing the geometry and angles involved in a general rotation about a plunging axis. The new axes are indicated by open circles and primed labels. The angle β is the angle between the rotation axis and the $\mathbf{X_2}$ (east) axis, ω is the magnitude of the rotation, and θ_{22} is the angle between the old $\mathbf{X_2}$ and the new $\mathbf{X'_2}$ axes.

$$\begin{aligned} a_{11} &= \cos\omega + \cos^2\alpha(1-\cos\omega) \\ a_{33} &= \cos\omega + \cos^2\gamma(1-\cos\omega) \end{aligned} \tag{3.11b}$$

We now have three equations and three independent unknowns. Therefore the remaining direction cosines can be calculated from the orthogonality relations or you can go through the somewhat more involved geometric derivation. They are given below without proof:

$$\begin{aligned} a_{12} &= -\cos\gamma\sin\omega + \cos\alpha\cos\beta(1-\cos\omega) \\ a_{13} &= \cos\beta\sin\omega + \cos\alpha\cos\gamma(1-\cos\omega) \\ a_{21} &= \cos\gamma\sin\omega + \cos\beta\cos\alpha(1-\cos\omega) \\ a_{23} &= -\cos\alpha\sin\omega + \cos\beta\cos\gamma(1-\cos\omega) \\ a_{31} &= -\cos\beta\sin\omega + \cos\gamma\cos\alpha(1-\cos\omega) \\ a_{32} &= \cos\alpha\sin\omega + \cos\gamma\cos\beta(1-\cos\omega) \end{aligned} \tag{3.11c}$$

These equations give the direction cosines of the transformation matrix in terms of the direction cosines of the rotation axis and the magnitude of the rotation. All that is needed to accomplish a general rotation is to convert the trend and plunge of the rotation axis into direction cosines, and then use the transformation matrix in Equations 3.11 in the vector transformation Equations 3.8. Here is a MATLAB function, **`Rotate`**, to do a rotation about an arbitrary axis:

```
function [rtrd,rplg] = Rotate(raz,rdip,rot,trd,plg, ans0)
%Rotate rotates a line by performing a coordinate transformation on
%vectors. The algorithm was originally written by Randall A. Marrett
%
%   USE: [rtrd,rplg] = Rotate(raz,rdip,rot,trd,plg,ans0)
%
```

```
%   raz = trend of rotation axis
%   rdip = plunge of rotation axis
%   rot = magnitude of rotation
%   trd = trend of the vector to be rotated
%   plg = plunge of the vector to be rotated
%   ans0 = A character indicating whether the line to be rotated is an axis
%   (ans0 = 'a') or a vector (ans0 = 'v')
%
%   NOTE: All angles are in radians
%
%   Rotate uses functions SphToCart and CartToSph

%Allocate some arrays
a = zeros(3,3); %Transformation matrix
pole = zeros(1,3); %Direction cosines of rotation axis
plotr = zeros(1,3); %Direction cosines of rotated vector
temp = zeros(1,3); %Direction cosines of unrotated vector

%Convert rotation axis to direction cosines. Note that the convention here
%is X1 = North, X2 = East, X3 = Down
[pole(1) pole(2) pole(3)] = SphToCart(raz,rdip,0);

% Calculate the transformation matrix
x = 1.0 - cos(rot);
sinRot = sin(rot); %Just reduces the number of calculations
cosRot = cos(rot);
a(1,1) = cosRot + pole(1)*pole(1)*x;
a(1,2) = -pole(3)*sinRot + pole(1)*pole(2)*x;
a(1,3) = pole(2)*sinRot + pole(1)*pole(3)*x;
a(2,1) = pole(3)*sinRot + pole(2)*pole(1)*x;
a(2,2) = cosRot + pole(2)*pole(2)*x;
a(2,3) = -pole(1)*sinRot + pole(2)*pole(3)*x;
a(3,1) = -pole(2)*sinRot + pole(3)*pole(1)*x;
a(3,2) = pole(1)*sinRot + pole(3)*pole(2)*x;
a(3,3) = cosRot + pole(3)*pole(3)*x;

%Convert trend and plunge of vector to be rotated into direction cosines
[temp(1) temp(2) temp(3)] = SphToCart(trd,plg,0);

%The following nested loops perform the coordinate transformation
for i = 1:3
    plotr(i) = 0.0;
    for j = 1:3
        plotr(i) = a(i,j)*temp(j) + plotr(i);
    end
end

%Convert to lower hemisphere projection if data are axes (ans0 = 'a')
```

```
if plotr(3) < 0.0 && ans0 == 'a'
    plotr(1) = -plotr(1);
    plotr(2) = -plotr(2);
    plotr(3) = -plotr(3);
end

%Convert from direction cosines back to trend and plunge
[rtrd,rplg]=CartToSph(plotr(1),plotr(2),plotr(3));
end
```

3.4.3 Graphical aside: Plotting great and small circles as a pole rotation

The transformation matrix we derived in the previous problem provides us with a simple and elegant way to draw great and small circles on any sort of spherical projection. The basic problem is, how to come up with a series of equally spaced points in the projection (lines in three dimensions) that one can connect with line segments to form the great or small circle. To solve this problem, we consider the pole to the great circle, or the axis of the conic section that defines the small circle, to be the rotation axis. Any vector perpendicular to the pole to the plane will, when rotated around the pole, trace out a plane that will intersect the projection sphere as a great circle. Likewise any vector that makes an angle of less than 90° will trace out a cone, which intersects the projection sphere as a small circle.

Thus, to make a program to draw great or small circles, you must first calculate the direction cosines of the pole to the plane or the center (axis) of the small circle. Then, pick a vector that lies somewhere on the great or small circle. If you are plotting a great circle, it is most convenient to choose the point where the circle intersects the primitive (i.e., the edge) of the projection. One of the main reasons for using a right-hand-rule format for specifying strike azimuths is that that vector will automatically trace out a lower hemisphere great circle when rotated 180° clockwise about the pole (a positive rotation). For small circles, you will probably want to choose the vector that has the minimum plunge (i.e., the vector with the same trend as the small circle axis and a plunge equal to the plunge of the axis minus the half apical angle of the small circle), unless the small circle intersects the edge of the stereographic projection, in which case the intersection is where you want to start.

From there, it is just a matter of rotating the vector a fixed increment and then drawing a line segment between the new and the old positions of the vector as projected on the net. This procedure is repeated until the total number of rotation increments equals 180° for a great circle or 360° for a small circle. On most computer screens, the resolution is such that 20 rotations in 9° increments (or something similar) will produce a reasonably smooth great circle. Smaller increments are time consuming and may actually produce a rougher great circle. The following MATLAB functions, **`GreatCircle`** and **`SmallCircle`**, use rotations to calculate the traces of great and small circles in equal area and equal angle projections:

```
function path = GreatCircle(strike,dip,sttype)
%GreatCircle computes the great circle path of a plane in an equal angle
%or equal area stereonet of unit radius
%
%   USE: path = GreatCircle(strike,dip,sttype)
%
%   strike = strike of plane
```

```
%   dip = dip of plane
%   sttype = type of stereonet. 0 for equal angle and 1 for equal area
%   path = vector with x and y coordinates of points in great circle path
%
%   NOTE: strike and dip should be entered in radians.
%
%   GreatCircle uses functions StCoordLine, Pole and Rotate

%Compute the pole to the plane. This will be the axis of rotation to make
%the great circle
[trda,plga] = Pole(strike,dip,1);

%Now pick a line at the intersection of the great circle with the primitive
%of the stereonet
trd = strike;
plg = 0.0;

%To make the great circle, rotate the line 180 degrees in increments
%of 1 degree
rot=(0:1:180)*pi/180;
path = zeros(size(rot,2),2);
for i = 1:size(rot,2)
    %Avoid joining ends of path
    if rot(i) == pi
        rot(i) = rot(i)*0.9999;
    end
    %Rotate line
    [rtrd,rplg] = Rotate(trda,plga,rot(i),trd,plg,'a');
    %Calculate stereonet coordinates of rotated line and add to great
    %circle path
    [path(i,1),path(i,2)] = StCoordLine(rtrd,rplg,sttype);
end
end

function [path1,path2,np1,np2] = SmallCircle(trda,plga,coneAngle,sttype)
%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
%
%   USE: [path1,path2,np1,np2] = SmallCircle(trda,plga,coneAngle,sttype)
%
%   trda = trend of axis
%   plga = plunge of axis
%   coneAngle = cone angle
%   sttype = type of stereonet. 0 for equal angle and 1 for equal area
%   path1 and path2 are vectors with the x and y coordinates of the points
%   in the small circle paths
%   np1 and np2 are the number of points in path1 and path2,
%   respectively
%
```

```
%   NOTE: All angles should be in radians
%
%   SmallCircle uses functions ZeroTwoPi, StCoordLine and Rotate

%Find where to start the small circle
if (plga - coneAngle) >= 0.0
    trd = trda;
    plg = plga - coneAngle;
else
    if plga == pi/2.0
        plga = plga * 0.9999;
    end
    angle = acos(cos(coneAngle)/cos(plga));
    trd = ZeroTwoPi(trda+angle);
    plg = 0.0;
end

%To make the small circle, rotate the starting line 360 degrees in
%increments of 1 degree
rot=(0:1:360)*pi/180;
path1 = zeros(size(rot,2),2);
path2 = zeros(size(rot,2),2);
np1 = 0; np2 = 0;
for i = 1:size(rot,2)
    %Rotate line: Notice that here the line is considered as a vector
    [rtrd,rplg] = Rotate(trda,plga,rot(i),trd,plg,'v');
    % Add to the right path
    % If plunge of rotated line is positive add to first path
    if rplg >= 0.0
        np1 = np1 + 1;
        %Calculate stereonet coordinates and add to path
        [path1(np1,1),path1(np1,2)] = StCoordLine(rtrd,rplg,sttype);
    %If plunge of rotated line is negative add to second path
    else
        np2 = np2 + 1;
        %Calculate stereonet coordinates and add to path
        [path2(np2,1),path2(np2,2)] = StCoordLine(rtrd,rplg,sttype);
    end
end
end
```

Normally, stereonets are presented with the primitive equal to the horizontal (i.e., looking straight down). However, it is often convenient to construct a stereonet in another orientation. For example, one may want to plot data in the plane of a cross section (a view direction that is horizontal and perpendicular to the trend of the cross section), or in the down-plunge view of a cylindrical fold (a view direction parallel to the fold axis). The MATLAB function **`GeogrToView`** below enables one to calculate a stereonet looking in any direction, by transforming any point in the stereonet from **NED** coordinates to view direction coordinates.

```
function [rtrd,rplg] = GeogrToView(trd,plg,trdv,plgv)
%GeogrToView transforms a line from NED to View Direction
%coordinates
%
%   USE: [rtrd,rplg] = Geogr To View(trd,plg,trdv,plgv)
%
%   trd = trend of line
%   plg = plunge of line
%   trdv = trend of view direction
%   plgv = plunge of view direction
%   rtrd and rplg are the new trend and plunge of the line in the view
%   direction.
%
%   NOTE: Input/Output angles are in radians
%
%   GeogrToView uses functions ZeroTwoPi, SphToCart and CartToSph

%Some constants
east = pi/2.0;

% Make transformation matrix between NED and View Direction
a = zeros(3,3);
[a(3,1),a(3,2),a(3,3)] = SphToCart(trdv,plgv,0);
temp1 = trdv + east;
temp2 = 0.0;
[a(2,1),a(2,2),a(2,3)] = SphToCart(temp1,temp2,0);
temp1 = trdv;
temp2 = plgv - east;
[a(1,1),a(1,2),a(1,3)] = SphToCart(temp1,temp2,0);

% Direction cosines of line
dirCos = zeros(1,3);
[dirCos(1),dirCos(2),dirCos(3)] = SphToCart(trd,plg,0);
% Transform line
nDirCos = zeros(1,3);
for i=1:3
    nDirCos(i) = a(i,1)*dirCos(1) + a(i,2)*dirCos(2)+ a(i,3)*dirCos(3);
end

% Compute line from new direction cosines
[rtrd,rplg] = CartToSph(nDirCos(1),nDirCos(2),nDirCos(3));

% Take care of negative plunges
if rplg < 0.0
    rtrd = ZeroTwoPi(rtrd+pi);
    rplg = -rplg;
end
end
```

Now we put all of the previous routines together in a function, **Stereonet**, that plots an equal area or equal angle stereonet in any view direction you want. This code is very short and efficient because it calls several of the previous functions in this chapter and Chapters 1 and 2.

```
function [] = Stereonet(trdv,plgv,intrad,sttype)
%Stereonet plots an equal angle or equal area stereonet of unit radius
%in any view direction
%
%   USE: Stereonet(trdv,plgv,intrad,stttype)
%
%   trdv = trend of view direction
%   plgv = plunge of view direction
%   intrad = interval in radians between great or small circles
%   sttype = An integer indicating the type of stereonet. 0 for equal angle,
%   and 1 for equal area
%
%   NOTE: All angles should be entered in radians
%
%   Example: To plot an equal area stereonet at 10 deg intervals in a
%   default view direction type:
%
%   Stereonet(0,90*pi/180,10*pi/180,1);
%
%   To plot the same stereonet but with a view direction of say: 235/42,
%   type:
%
%   Stereonet(235*pi/180,42*pi/180,10*pi/180,1);
%
%   Stereonet uses functions Pole, GeogrToView, SmallCircle and GreatCircle

% Some constants
east = pi/2.0;
west = 3.0*east;

% Plot stereonet reference circle
r = 1.0; % radius of stereonet
TH = (0:1:360)*pi/180; % polar angle, range 2 pi, 1 degree increment
[X,Y] = pol2cart(TH,r); % cartesian coordinates of reference circle
plot(X,Y,'k'); % plot reference circle
axis ([-1 1 -1 1]); % size of stereonet
axis equal; axis off; % equal axes, no axes
hold on; % hold plot

% Number of small circles
nCircles = pi/(intrad*2.0);
% Small circles
% Start at the North
trd = 0.0;
plg = 0.0;
```

```
% If view direction is not the default (trd=0,plg=90), transform line to
% view direction
if trdv ~= 0.0 || plgv ~= east
    [trd,plg] = GeogrToView(trd,plg,trdv,plgv);
end
% Plot small circles
for i = 1:nCircles
    coneAngle = i*intrad;
    [path1,path2,np1,np2] = SmallCircle(trd,plg,cone Angle,sttype);
    plot(path1(1:np1,1),path1(1:np1,2),'b');
    if np2 > 0
        plot(path2(1:np2,1),path2(1:np2,2),'b');
    end
end

% Great circles
for i = 0:nCircles*2
    %Western half
    if i <= nCircles
        % Pole of great circle
        trd = west;
        plg = i*intrad;
    %Eastern half
    else
        % Pole of great circle
        trd = east;
        plg = (i-nCircles)*intrad;
    end
    % If pole is vertical, shift it a little bit
    if plg == east
        plg = plg * 0.9999;
    end
    % If view direction is not the default (trd=0,plg=90), transform line to
    % view direction
    if trdv ~= 0.0 || plgv ~= east
        [trd,plg] = GeogrToView(trd,plg,trdv,plgv);
    end
    % Compute plane from pole
    [strike,dip] = Pole(trd,plg,0);
    % Plot great circle
    path = GreatCircle(strike,dip,sttype);
    plot(path(:,1),path(:,2),'b');
end
hold off; %release plot
end
```

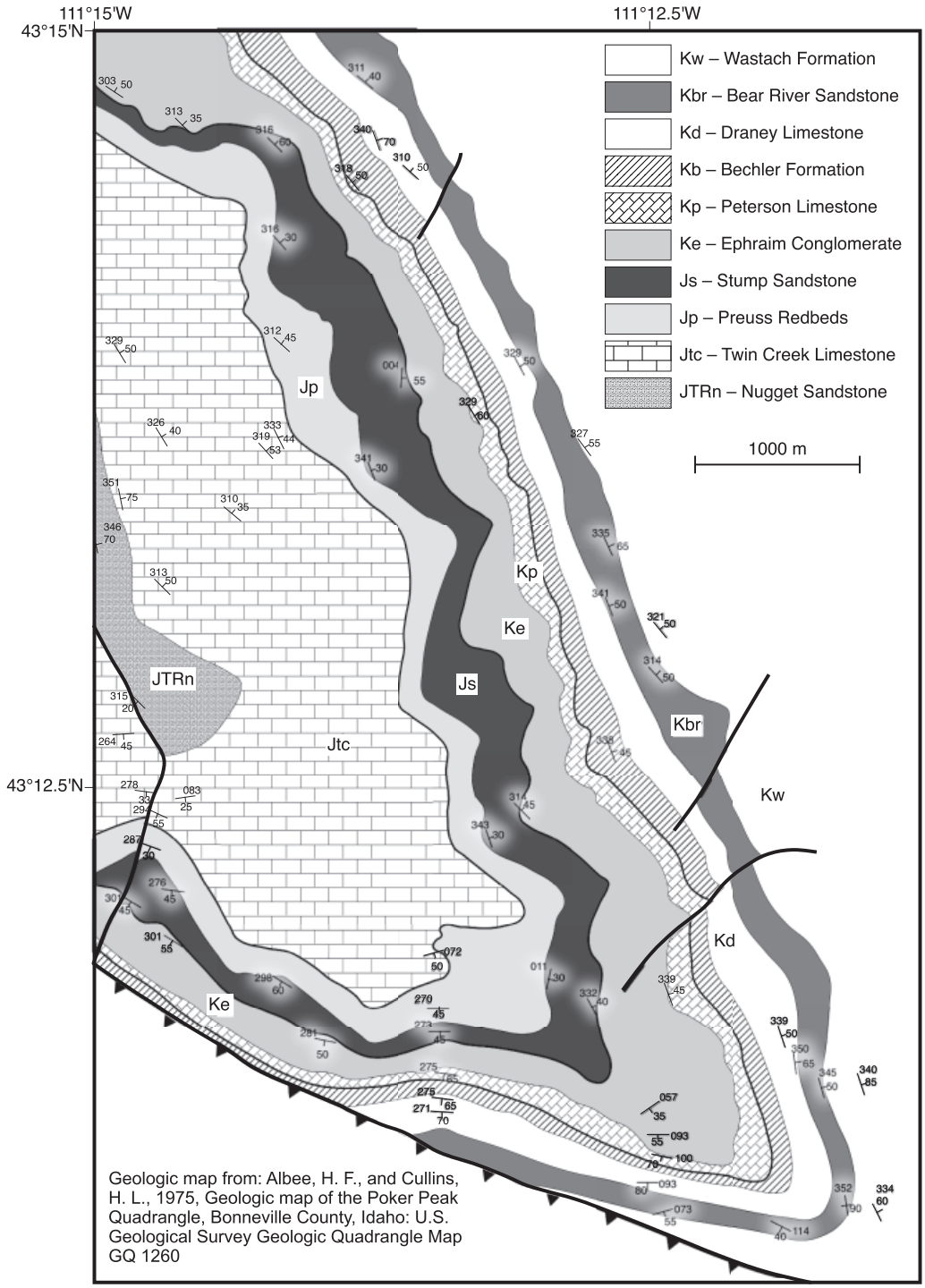


Figure 3.11 Simplified geologic map of the Big Elk anticline in southeastern Idaho, to accompany Exercise 8.

3.5 EXERCISES

1. Derive the equation for component a_{21} of the transformation matrix for the case of a general rotation.
2. Derive the transformation matrix of Equation 3.10, but this time as a vector transformation (Eq. 3.8) between the $\mathbf{X_1}$ = north, $\mathbf{X_2}$ = east, $\mathbf{X_3}$ = up coordinate system and the fold axis based $\mathbf{X'_1}$–$\mathbf{X'_2}$–$\mathbf{X'_3}$ coordinate system.
3. Derive the transformation matrix for a down-plunge projection in the right-handed coordinate system, $\mathbf{X_1}$ = south, $\mathbf{X_2}$ = west, $\mathbf{X_3}$ = down.
4. Evaluate the problem of construction of a vertical section of a plunging cylindrical fold. Can this problem be carried out as a transformation of coordinates and points? If so, derive the transformation matrix; if not precisely state why not.
5. Construct the down-plunge projection of the contact between the gray and white units in Figure 3.7, using the MATLAB function **`DownPlunge`**.
6. Using the function **`Stereonet`**, plot equal area stereonets with 10° grid interval, and the following view directions: 123/42, 032/57, 245/21, 321/49.
7. Plot in MATLAB the following lines and planes in equal area stereonets with 10° grid interval, and view directions 000/90 and 214/56. Lines = 212/23, 014/56, 321/53. Planes = 211/24, 035/67, 238/76. Hint: Use functions **`StCoordLine`** (Chapter 1), **`GreatCircle`**, **`GeogrToView`**, and **`Stereonet`** (this chapter).
8. Figure 3.11 is a geologic map of the Big Elk anticline, located in the Mesozoic thrust belt in southeastern Idaho, United States (Albee and Cullins, 1975).
   a. The trend and plunge of the fold axis is 125/26. In Chapter 5, we will return to this example once you have learned how to calculate a best-fitting fold axis.
   b. Supplementary data file "Problem 3.8" contains the digitized contacts (east, north, up) of the top of the Jurassic Twin Creek Limestone (Jtc), the Jurassic Stump Sandstone (Js), and the Cretaceous Peterson Limestone (Kp). Using the equations and functions (e.g., **`DownPlunge`**) developed in this chapter, construct a down-plunge section of the Big Elk anticline.
   c. The Idaho–Wyoming thrust belt in which this structure occurs thrusts from west to east. What is the vergence (i.e., asymmetry) of the Big Elk anticline and does it agree with the general direction of thrusting? Do you note anything unusual about the sequence between Jtc and Js? This sequence contains the Preuss redbeds, which are known to contain evaporate minerals. Can you draw any conclusions with this additional information?