Chapter 17. Crustal strain as measured using GPS geodesy

17.1 Introduction

I begin this chapter by acknowledging that my treatment of this material follows that of Allmendinger, Cardozo and Fisher (2012), to whom I owe an intellectual debt for writing such a clear text.

The purpose of this chapter is to bring you to the point at which you can estimate the 2-dimensional instantaneous strain rate between a set of 3 geodetic GPS stations. This entails working on concepts as well as code.

17.2 Strain

One-dimensional strain

The idea of one-dimensional strain is the starting point for discussions of strain in physics, engineering, and structural geology courses, so it might be familiar to you. If so, just play along so that we can build our understanding from the ground up.

You have a lovely strip of elastic material like a broken rubber band. This is a very special elastic strip -- the sort that one dreams of in a physics class, along with frictionless surfaces and massless elephants. Your elastic strip has a perfectly linear-elastic rheology, so that it lengthens in exact proportion to the amont of force you exert to pull on it. And we can use the idea of force rather than stress because our imaginary elastic strip is infinitely thin. So displacement is a linear function of force for our 1-D experiment. With all of those limitations in mind, let's move on.

Before you elongate the elastic strip, you paint a thin white band around it in a location that will serve as the origin of our 1-dimensional coordinate system; that is, its coordinate is defined as $\{0\}$. You move down the strip a little bit to your right and paint a thin blue band, and a little ways further down the strip you paint a thin red band. The blue band has an initial or original coordinate of x_{bo}, and the red band has an original coordinate of x_{ro}.

Next, we stretch the rubber strip so that the distance from the origin to the new location of the thin red band (x_{rn}) is twice the distance before stretching (x_{ro}). That is,

$$x_{rn} = 2\, x_{ro}$$

Thanks to all of the restrictions we placed on the properties of the elastic strip, we can reliably predict the coordinates of the blue band after stretching.

$$x_{bn} = 2\, x_{bo}$$

In fact, we can specify the new location of any point ρ along the elastic strip ($x_{\rho n}$) given its original location ($x_{\rho o}$).

$$x_{\rho n} = 2\, x_{\rho o}$$

The process of expressing a new location in terms of an initial or original location is an example of a Green transformation. We use the word "transformation" in the sense of "coordinate transformation" -- determining the coordinates of a point relative to a different coordinate system.

The number "2" in this example is called the deformation gradient or slope, which makes some sense because our elastic strip has a *linear* elastic rheology. Digging a little deeper, the difference between the initial distance between any two points ρ and β on the strip, $(x_{\rho o} - x_{\beta o}) = \Delta x_o$, is proportional to the difference between the new distance between those same two points, $(x_{\rho n} - x_{\beta n}) = \Delta x_n$. The quotient of the final length divided by the initial length is called the *stretch* (S) by structural geologists, so

$$S = \frac{\text{final distance}}{\text{initial distance}} = \frac{(x_{\rho n} - x_{\beta n})}{(x_{\rho o} - x_{\beta o})} = \frac{\Delta x_n}{\Delta x_o} = \lim \frac{\partial x_n}{\partial x_o} = 2$$

Hence, the stretch in our 1-D example is the same as the deformation gradient. We can say that the deformation gradient is homogeneous in our example because it is constant for any pair of points along the linear-elastic strip. Although it was not strictly necessary in this example, the deformation gradient was expressed in terms of partial derivatives using the ∂ symbol because we are trying to build a toolbox that will allow us to account for deformation in 2 and 3 dimensions.

Similarly, the original location can be expressed in terms of the new location using a Cauchy transformation, so the inverse stretch (S^{-1}) is

$$S^{-1} = \frac{\text{initial distance}}{\text{final distance}} = \frac{(x_{\rho o} - x_{\beta o})}{(x_{\rho n} - x_{\beta n})} = \frac{\Delta x_o}{\Delta x_n} = \lim \frac{\partial x_o}{\partial x_n} = 0.5$$

Another way to approach the analysis is to consider the displacement vectors for the red and blue markers. The displacement vectors connecting the initial to the final positions of the red and blue markers are

$$\boldsymbol{u}_r = x_{rn} - x_{ro}$$

and

$$\boldsymbol{u}_b = x_{bn} - x_{bo}$$

A displacement vector from the original location to the final location is called a *Lagrangian* displacement vector. The gradient or slope of Lagrangian displacement vector for a 1-D experiment is the same as the extension (e) that you might be familiar with from structural geology

$$e = \frac{(\text{final length}) - (\text{initial length})}{\text{initial length}}$$

In this instance,

$$e = \frac{\text{difference between the lengths of two displacement vectors}}{\text{difference between the lengths of two initial-location vectors}}$$

The difference between the lengths of two displacement vectors is

$$\Delta \boldsymbol{u}_L = \boldsymbol{u}_r - \boldsymbol{u}_b = (x_{rn} - x_{ro}) - (x_{bn} - x_{bo})$$

and the difference between the lengths of two initial-location vectors is

$$\Delta x_o = \boldsymbol{x}_{ro} - \boldsymbol{x}_{bo}$$

so

$$e = \frac{(\boldsymbol{u}_r - \boldsymbol{u}_b)}{(\boldsymbol{x}_{ro} - \boldsymbol{x}_{bo})} = \frac{\Delta \boldsymbol{u}_L}{\Delta x_o} = \lim \frac{\partial \boldsymbol{u}_L}{\partial x_o}$$

The inverse of this process, in which we consider the displacement vectors relative to the final location vectors, gives the *Eulerian* displacement vector, and its gradient is the inverse of the extension.

$$e^{-1} = \frac{\text{difference between the lengths of two displacement vectors}}{\text{difference between the lengths of two final}-\text{location vectors}}$$

$$e^{-1} = \frac{(\boldsymbol{u}_r - \boldsymbol{u}_b)}{(\boldsymbol{x}_{rn} - \boldsymbol{x}_{bn})} = \frac{\Delta \boldsymbol{u}}{\Delta x_n} = \lim \frac{\partial \boldsymbol{u}}{\partial x_n}$$

The stretch and the extension are scalar components of tensor quantities.

Three-dimensional strain

If the mental image we worked with in the previous section was that of an elastic string, elongated (or shortened) in one dimension, we are now going to build on that experience to think of a continuous elastic sheet that can be deformed in 3 dimensions.

Before we press on into 2- or 3-dimensional strain, we need to add to our understanding of matrices and, in particular, the matrix representation of tensors. In this square matrix,

$$\begin{pmatrix} \mathrm{A} & 0 & 0 \\ 0 & \mathrm{B} & 0 \\ 0 & 0 & \mathrm{C} \end{pmatrix}$$

the part of the matrix that has all of the capital letters (A, B, C) is called the *diagonal* or *axis* of the matrix. It is called a *square* matrix because it has the same number of rows as columns. Values that are in the positions occuppied by 0s are said to be *off-axis* terms. If the values above the diagonal are equal to the values below and directly across the diagonal, like this

$$\begin{pmatrix} \mathrm{A} & \mathrm{d} & \mathrm{e} \\ \mathrm{d} & \mathrm{B} & \mathrm{f} \\ \mathrm{e} & \mathrm{f} & \mathrm{C} \end{pmatrix}$$

the matrix is said to be a *symmetric* matrix. If values across the diagonal from each other have the same magnitude but different sign, like

$$\begin{pmatrix} \mathrm{A} & \mathrm{d} & \mathrm{e} \\ -\mathrm{d} & \mathrm{B} & \mathrm{f} \\ -\mathrm{e} & -\mathrm{f} & \mathrm{C} \end{pmatrix}$$

the matrix is said to be an *antisymmetric* matrix. An *asymmetric* matrix, like

$$\begin{pmatrix} \mathrm{A} & \mathrm{d} & \mathrm{e} \\ \mathrm{g} & \mathrm{B} & \mathrm{h} \\ \mathrm{n} & -\mathrm{f} & \mathrm{C} \end{pmatrix}$$

lacks at least some of the symmetries we have just mentioned. Finally, if we define a matrix

$$M = \begin{pmatrix} \mathrm{A} & \mathrm{d} & \mathrm{e} \\ \mathrm{g} & \mathrm{B} & \mathrm{h} \\ \mathrm{n} & -\mathrm{f} & \mathrm{C} \end{pmatrix},$$

the *transpose* of matrix M is represented by M^T and is

$$M^T = \begin{pmatrix} \mathrm{A} & \mathrm{g} & \mathrm{n} \\ \mathrm{d} & \mathrm{B} & -\mathrm{f} \\ \mathrm{e} & \mathrm{h} & \mathrm{C} \end{pmatrix},$$

so the values along the diagonal are unchanged, but the values across the diagonal from each other are swapped.

The deformation and displacement gradients that we previously explored for 1-D strain can be expanded into their 3-D counterparts. The following equations relate to the displacement of a point with initial coordinates $\{x_o, y_o, z_o\}$ to final coordinates $\{x_n, y_n, z_n\}$, and vice versa. The Green transformation in 3-D is

$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = \begin{pmatrix} \frac{\partial x_n}{\partial x_o} & \frac{\partial x_n}{\partial y_o} & \frac{\partial x_n}{\partial z_o} \\ \frac{\partial y_n}{\partial x_o} & \frac{\partial y_n}{\partial y_o} & \frac{\partial y_n}{\partial z_o} \\ \frac{\partial z_n}{\partial x_o} & \frac{\partial z_n}{\partial y_o} & \frac{\partial z_n}{\partial z_o} \end{pmatrix} \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix},$$

and the Cauchy transformation is

$$\begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} = \begin{pmatrix} \frac{\partial x_o}{\partial x_n} & \frac{\partial x_o}{\partial y_n} & \frac{\partial x_o}{\partial z_n} \\ \frac{\partial y_o}{\partial x_n} & \frac{\partial y_o}{\partial y_n} & \frac{\partial y_o}{\partial z_n} \\ \frac{\partial z_o}{\partial x_n} & \frac{\partial z_o}{\partial y_n} & \frac{\partial z_o}{\partial z_n} \end{pmatrix} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}.$$

The Lagrange displacement equation is

$$\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \frac{\partial u_x}{\partial x_o} & \frac{\partial u_x}{\partial y_o} & \frac{\partial u_x}{\partial z_o} \\ \frac{\partial u_y}{\partial x_o} & \frac{\partial u_y}{\partial y_o} & \frac{\partial u_y}{\partial z_o} \\ \frac{\partial u_z}{\partial x_o} & \frac{\partial u_z}{\partial y_o} & \frac{\partial u_z}{\partial z_o} \end{pmatrix} \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix},$$

and the Euler displacement equation is

$$\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \frac{\partial u_x}{\partial x_n} & \frac{\partial u_x}{\partial y_n} & \frac{\partial u_x}{\partial z_n} \\ \frac{\partial u_y}{\partial x_n} & \frac{\partial u_y}{\partial y_n} & \frac{\partial u_y}{\partial z_n} \\ \frac{\partial u_3}{\partial x_n} & \frac{\partial u_3}{\partial y_n} & \frac{\partial u_3}{\partial z_n} \end{pmatrix} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}.$$

We make the simplifying assumption that the Lagrangian and Eulerian displacement gradients are approximately equal; that is,

$$\begin{pmatrix} \frac{\partial u_x}{\partial x_o} & \frac{\partial u_x}{\partial y_o} & \frac{\partial u_x}{\partial z_o} \\ \frac{\partial u_y}{\partial x_o} & \frac{\partial u_y}{\partial y_o} & \frac{\partial u_y}{\partial z_o} \\ \frac{\partial u_z}{\partial x_o} & \frac{\partial u_z}{\partial y_o} & \frac{\partial u_z}{\partial z_o} \end{pmatrix} \approx \begin{pmatrix} \frac{\partial u_x}{\partial x_n} & \frac{\partial u_x}{\partial y_n} & \frac{\partial u_x}{\partial z_n} \\ \frac{\partial u_y}{\partial x_n} & \frac{\partial u_y}{\partial y_n} & \frac{\partial u_y}{\partial z_n} \\ \frac{\partial u_3}{\partial x_n} & \frac{\partial u_3}{\partial y_n} & \frac{\partial u_3}{\partial z_n} \end{pmatrix}$$

Hence, when strains are infinitesimal, the difference between the displacement gradients in the initial and final states is not important. The matrix of extensions in the Lagrangian displacement matrix looks like

$$\begin{pmatrix} \frac{\partial u_x}{\partial x_o} & \frac{\partial u_x}{\partial y_o} & \frac{\partial u_x}{\partial z_o} \\ \frac{\partial u_y}{\partial x_o} & \frac{\partial u_y}{\partial y_o} & \frac{\partial u_y}{\partial z_o} \\ \frac{\partial u_z}{\partial x_o} & \frac{\partial u_z}{\partial y_o} & \frac{\partial u_z}{\partial z_o} \end{pmatrix} = \begin{pmatrix} e_{xx} & e_{xy} & e_{xz} \\ e_{yx} & e_{yy} & e_{yz} \\ e_{zx} & e_{zy} & e_{zz} \end{pmatrix}$$

The elements of the matrix along the diagonal (e_{xx}, e_{yy}, e_{zz}) are equal to the extensions along the x, y and z axes, respectively. So the diagonal terms are related to changes in length in each of three orthogonal directions.

The remaining off-diagonal components are related to angular changes. The displacement gradient tensor is an asymmetric tensor that represents both distortion (change in shape) and rotation. A tensor that represented rotation only would be antisymmetric; that is, $e_{xy} = -e_{yx}$, $e_{xz} = -e_{zx}$ and $e_{yz} = -e_{zy}$. The asymmetry in the displacement gradient tensor arises because of rotations and infinitesimal distortion.

Imagine a vector with finite length (that is, it is not infinitesimally short) that is initially parallel to the x axis and is rotated through an angle θ counter-clockwise around the z axis toward the y axis during deformation. Then

$$\tan\theta = \frac{\Delta u_y}{\Delta x_o + \Delta u_x}$$

Given the condition of infinitesimal strain, we know that $\Delta u_x \ll \Delta x_o$; that is, the component of the displacement along the x axis is much smaller than the original length of the vector along the x axis. Consequently,

$$\tan\theta \approx \frac{\Delta u_y}{\Delta x_o}$$

and angle θ is quite small. The tangent of very small angles is approximately equal to the angle itself, expressed in radian measure, so

$$\tan\theta \approx \theta \approx \frac{\Delta u_y}{\Delta x_o} = e_{yx}$$

Component e_{yx} measures the counter-clockwise (positive) rotation, around the z axis, of a vector that is initially parallel to the x axis toward the y axis during deformation. Similarly, given a vector that is initially parallel to the y axis, component e_{xy} is approximately equal to the angle that the vector rotates *clockwise,* around the z axis, toward the x axis during deformation.

Thus far in our discussion, I have tried to keep the symbols I use as simple and uncomplimented as possible. It is more typical to use tensor notation, sometimes called Einstein notation, to express relationships in a very compact manner. Tensor notation uses subscript letters (typically *i*, *j* and *k*) to indicate coordinate directions or combinations of coordinate directions. The details of how to unpack tensor notation are explained in the appendix. For clarity in this introductory presentation, we will leave the expressions unpacked and explicitly use the older *x, y, z* terminology for coordinate axes; however, we will employ the index subscript notation in some variable names introduced below.

An asymmetric tensor can be decomposed into the sum of a symmetric tensor and an antisymmetric tensor. The asymmetric Lagrangian displacement tensor e_{ij} is the sum of the symmetric infinitesimal strain tensor ε_{ij} and the antisymmetric rotation tensor Ω_{ij}.

$$\varepsilon_{ij} = \begin{pmatrix} e_{xx} & \frac{(e_{xy}+e_{yx})}{2} & \frac{(e_{xz}+e_{zx})}{2} \\ \frac{(e_{yx}+e_{xy})}{2} & e_{yy} & \frac{(e_{yz}+e_{zy})}{2} \\ \frac{(e_{zx}+e_{xz})}{2} & \frac{(e_{zy}+e_{yz})}{2} & e_{zz} \end{pmatrix}$$

$$\Omega_{ij} = \begin{pmatrix} 0 & \frac{(e_{xy}-e_{yx})}{2} & \frac{(e_{xz}-e_{zx})}{2} \\ \frac{(e_{yx}-e_{xy})}{2} & 0 & \frac{(e_{yz}-e_{zy})}{2} \\ \frac{(e_{zx}-e_{xz})}{2} & \frac{(e_{zy}-e_{yz})}{2} & 0 \end{pmatrix}$$

The antisymmetric rotation tensor Ω_{ij} is also known as an axial vector. The axial vector is directed along the rotational axis and the length of the axial vector is equal to the angle of rotation expressed in radian measure. The Cartesian coordinates of the axial vector are given by $\{r_x, r_y, r_z\}$, where

$$r_x = \frac{-(\Omega_{yz} - \Omega_{zy})}{2}$$

$$r_y = \frac{-(\Omega_{xz} - \Omega_{zx})}{2}$$

$$r_z = \frac{-(\Omega_{xy} - \Omega_{yx})}{2}$$

and the angle of rotation in radians is the length of the axial vector,

$$|\mathrm{r}| = \sqrt{r_x^2 + r_y^2 + r_z^2}$$

Example

Problem. Given the following displacement gradient tensor (e_{ij}), calculate the strain tensor (ε_{ij}) and the rotation matrix (Ω_{ij}), and the magnitudes and orientations of the principal axes.

$$e_{ij} = \begin{pmatrix} e_{xx} & e_{xy} & e_{xz} \\ e_{yx} & e_{yy} & e_{yz} \\ e_{zx} & e_{zy} & e_{zz} \end{pmatrix} = \begin{pmatrix} 3 & 3 & 2 \\ 9 & 8 & 1 \\ 6 & -1 & 5 \end{pmatrix}$$

Answer

The strain tensor is given by

$$\varepsilon_{ij} = \begin{pmatrix} e_{xx} & \frac{(e_{xy}+e_{yx})}{2} & \frac{(e_{xz}+e_{zx})}{2} \\ \frac{(e_{yx}+e_{xy})}{2} & e_{yy} & \frac{(e_{yz}+e_{zy})}{2} \\ \frac{(e_{zx}+e_{xz})}{2} & \frac{(e_{zy}+e_{yz})}{2} & e_{zz} \end{pmatrix} = \begin{pmatrix} 3 & \frac{(3+9)}{2} & \frac{(2+6)}{2} \\ \frac{(9+3)}{2} & 8 & \frac{(1+(-1))}{2} \\ \frac{(6+2)}{2} & \frac{((-1)+1)}{2} & 5 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 4 \\ 6 & 8 & 0 \\ 4 & 0 & 5 \end{pmatrix}$$

and the rotation tensor is

$$\Omega_{ij} = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} & \Omega_{xz} \\ \boldsymbol{\Omega_{yx}} & \Omega_{yy} & \Omega_{yz} \\ \Omega_{zx} & \Omega_{zy} & \Omega_{zz} \end{pmatrix} = \begin{pmatrix} 0 & \frac{(e_{xy}-e_{yx})}{2} & \frac{(e_{xz}-e_{zx})}{2} \\ \frac{(e_{yx}-e_{xy})}{2} & 0 & \frac{(e_{yz}-e_{zy})}{2} \\ \frac{(e_{zx}-e_{xz})}{2} & \frac{(e_{zy}-e_{yz})}{2} & 0 \end{pmatrix} = \begin{pmatrix} 0 & \frac{(3-9)}{2} & \frac{(2-6)}{2} \\ \frac{(9-3)}{2} & 0 & \frac{(1-(-1))}{2} \\ \frac{(6-2)}{2} & \frac{((-1)-1)}{2} & 0 \end{pmatrix} = \begin{pmatrix} 0 & -3 & -2 \\ 3 & 0 & 1 \\ 2 & -1 & 0 \end{pmatrix}$$

The antisymmetric rotation tensor (a.k.a., the axial vector) Ω_{ij} is directed along the rotational axis, and the length of the axial vector is equal to the angle of rotation expressed in radian measure. The Cartesian coordinates of the axial vector are given by $\{r_x, r_y, r_z\}$, where

$$r_x = \frac{-(\Omega_{yz} - \Omega_{zy})}{2} = \frac{-(1-(-1))}{2} = -1$$

$$r_y = \frac{-(\Omega_{xz} - \Omega_{zx})}{2} = \frac{-((-2)-2)}{2} = 2$$

$$r_z = \frac{-(\Omega_{xy} - \Omega_{yx})}{2} = \frac{-((-3)-3)}{2} = 3$$

and the angle of rotation in radians is

$$|\mathbf{r}| = \sqrt{r_x^2 + r_y^2 + r_z^2} = \sqrt{(-1)^2 + 2^2 + 3^2} = \sqrt{14} = 3.74166 \text{ radians}$$

Using built-in *Mathematica* functions, we can determine the eigenvectors and eigenvalues. Eigenvectors are vectors that coincide with the principal strain axes, and eigenvalues are the magnitudes of the principal strains.

17.3 2-D horizontal strain rate from GPS velocity data

Introduction

The purpose of this section is to determine the horizontal strain in a triangular area between three non-colinear GPS stations, given their initial locations as well as their north-south and east-west velocities. This code is based largely on the explanations published in Allmendinger, Cardozo and Fisher (2012) and Cardozo and Allmendinger (2009).

A stand-alone Macintosh application called SSPX performs this same analysis in a more comprehensive manner, including estimates of uncertainty. SSPX is available for academic or research use for free via Nestor

Cardozo’s website at http://homepage.mac.com/nfcd/work /programs.html.

Input data

GPS data from the EarthScope Plate Boundary Observatory is managed by UNAVCO (http://www.unavco.org/) and is available online for free at http://pbo.unavco.org/data. The full public data holdings of UNAVCO are available via their “Data Archive Interface Version 2” at http://facility.unavco.org/data/dai2/app/dai2.html#.

I am going to search for data generated by one of the Plate Boundary Observatory’s permanent GPS stations near Lake Tahoe along the California-Nevada border. If I don’t know which station I want to learn about, I can go to the interactive PBO map (http://pbo.unavco.org/network/gps) and zoom in on an area of interest. I find a green marker dot in my area of interest, indicating a station that is functioning normally, and click on it for some initial information. The dot I chose is associated with station P150 (Martis Creek CN2008) located near Kings Beach on the north side of Lake Tahoe.

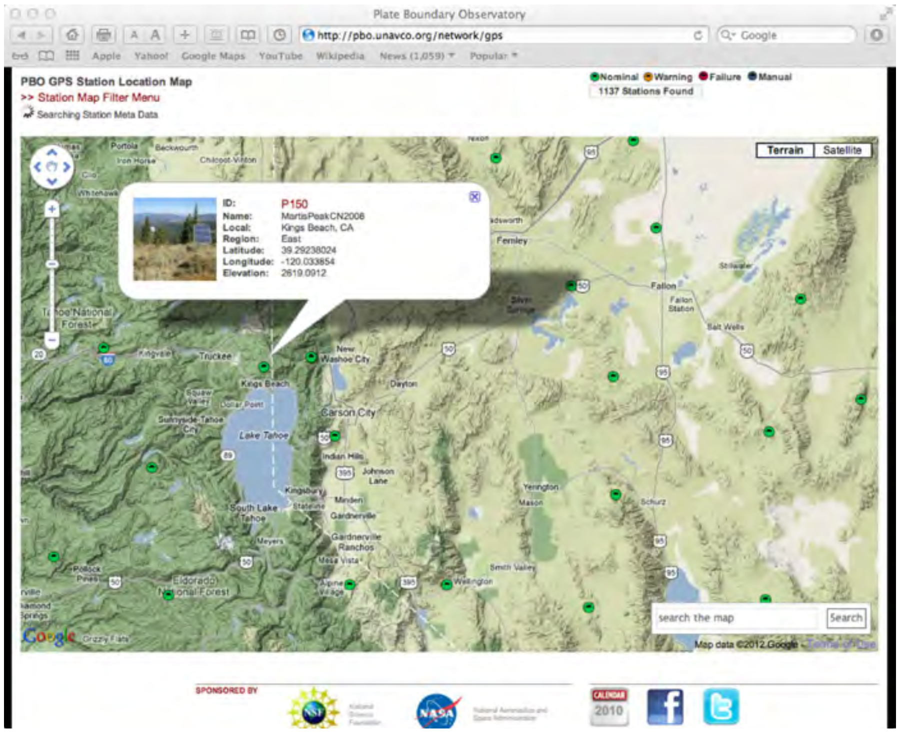
Plate Boundary Observatory
http://pbo.unavco.org/network/gps
PBO GPS Station Location Map
>> Station Map Filter Menu
Searching Station Meta Data
Nominal Warning Failure Manual
1137 Stations Found
Terrain
Satellite
Truckee
Kings Beach
Lake Tahoe
Carson City
South Lake Tahoe
Fallon
search the map
Search
SPONSORED BY

ID: P150
Name: MartisPeakCN2008
Local: Kings Beach, CA
Region: East
Latitude: 39.29238024
Longitude: -120.033854
Elevation: 2619.0912

Figure 17-1. Interactive Plate Boundary Observatory station site viewer, zoomed to the area around Lake Tahoe. Inset window provides some data and a clickable link for more data about site P150. From http://pbo.unavco.org/network/gps.

Clicking on the dot gives me a little bit of name and location information, and a clickable link to more information at http://pbo.unavco.org/station/overview/P150.

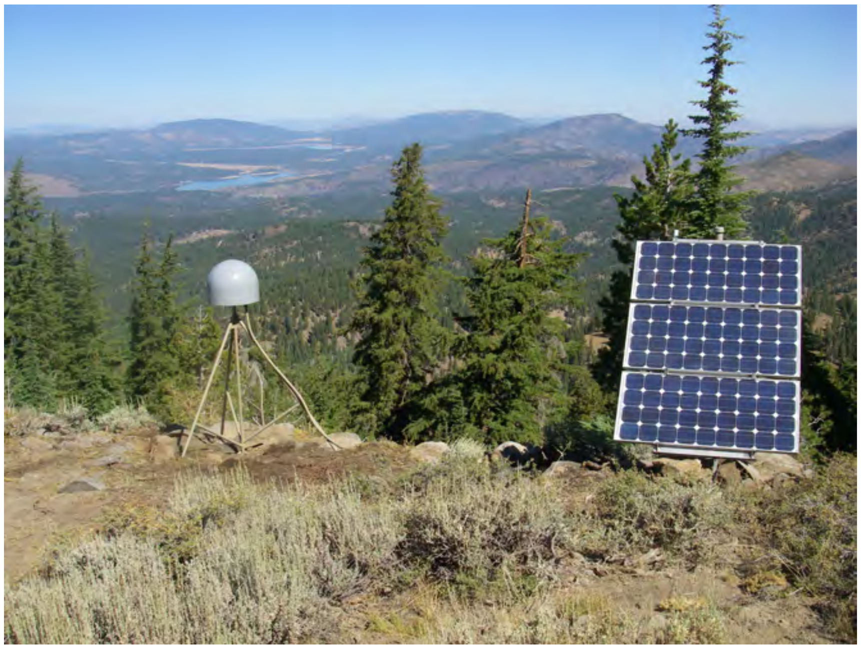

Figure 17-2. Plate Boundary Observatory Station 150 just north of Lake Tahoe. From http://pbo.unavco.org/station/photos/P150/.

____

The overview page provides me with the location of the site in the Stable North American Reference Frame (SNARF), WGS84 coordinate system: latitude 39.292380598° and longitude -120.033853482° at an elevation of 2619.0828 meters above the WGS84 datum. To compute the corresponding location in Universal Transverse Mercator (utm) projection, I used an online conversion utility (*e.g.*, http://www.uwgb.edu/dutchs/usefuldata/ConvertUTMNoOZ.HTM or http://home.hiwaay.net/~taylorc/toolbox/geography/geoutm.html; to learn more about the conversion, go to http://www.uwgb.edu/dutchs/usefuldata/utmformulas.htm) . I can also click on the "Data Products" tab and access a variety of other data, including velocity data. At http://pbo.unavco.org/ index.php/station/data/P150, I found detrended time-series plots that indicated that P150 has an average instantaneous N-S velocity of 5.97±0.04 mm/yr (a positive value means moving toward north), an E-W velocity of -11.07 ±0.03 mm/yr (-ve means moving toward west), and an up-down velocity of -0.20±0.07 mm/yr (-ve means moving down).

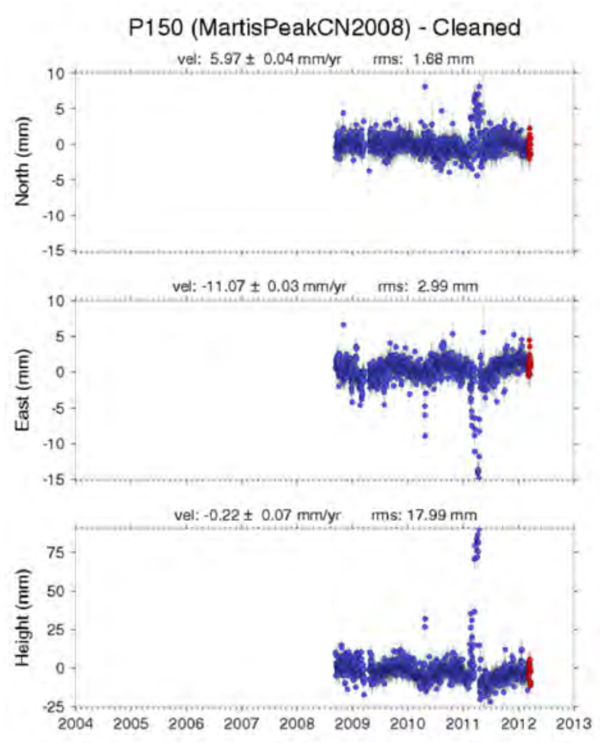

Figure 17-3. Static plot of cleaned and detrended time-series plots with interpreted velocities relative to the Stable North American Reference Frame (SNARF) from PBO GPS station P150. Accessed 25 March 2012 via http://pbo.unavco.org/index.php/station/data/P150.

For convenience, we compile the data in a 9 x 3 matrix in which each row corresponds to a different GPS station and the columns contain the following data types : 1 = site number, 2 = longitude, 3 = latitude, 4 = elevation (m, ellipsoid), 5 = UTM X coordinate, 6 = Y coordinate, 7 = N velocity (m/yr), 8 = E velocity, 9 = up velocity.
The order in which the station data are presented in this matrix (i.e., which station corresponds to record 1, 2 or 3) is arbitrary.

In[1]:=

inputData =

$$\begin{pmatrix} 146 & -120.537284 & 39.337459 & 2347.844 & 712\,247.260 & 4\,357\,118.219 & 0.00715 & -0.01025 & -0.0018 \\ 149 & -120.104975 & 39.60212988 & 2634.6887 & 748\,566.261 & 4\,387\,604.030 & 0.00555 & -0.00933 & -0.0016 \\ 150 & -120.03385482 & 39.292380598 & 2619.0828 & 755\,806.266 & 4\,353\,418.463 & 0.00597 & -0.01107 & -0.0002 \end{pmatrix};$$

General explanation of the method

We would like to use data from GPS arrays to define the average instantaneous strain rate in a triangular area between three GPS stations. We know the initial location (x_o, y_o), as well as the east-west instantaneous velocity (u_x) and the north-south instantaneous velocity (u_y), of each GPS station. We do not know the elements of the deformation gradient tensor (e_{xx}, e_{xy}, e_{yx}, e_{yy}) or the coordinates of the translation vector (t_x, t_y). The underlying relationships in matrix form is

$$\begin{pmatrix} {}^1u_x \\ {}^1u_y \\ {}^2u_x \\ {}^2u_y \\ {}^3u_x \\ {}^3u_y \end{pmatrix} = \begin{pmatrix} 1 & 0 & {}^1x_o & {}^1y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^1x_o & {}^1y_o \\ 1 & 0 & {}^2x_o & {}^2y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^2x_o & {}^2y_o \\ 1 & 0 & {}^3x_o & {}^3y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^3x_o & {}^3y_o \end{pmatrix} \begin{pmatrix} t_x \\ t_y \\ e_{xx} \\ e_{xy} \\ e_{yx} \\ e_{yy} \end{pmatrix}$$

Following the explanation in Allmendinger and others (2012, p. 156), the matrix equation above has the form of

$$\mathbf{y} = \mathbf{M}\mathbf{x}$$

where **y** is a 1x6 matrix of known quantities (the instantaneous displacement/velocity vectors), **x** is a 1x6 matrix of unknown quantities (the translation vector and the deformation gradient tensor), and **M** is a 6x6 matrix of known values including zeros, ones and the location vector coordinates of the three GPS stations. We need to rearrange this matrix equation so that all of the known quantities are collected on one side and the unknowns are on the other side of the equation. This requires us to compute the inverse of **M**

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$$

Mathematica has a built-in function to invert matrices, which we will employ in the form Inverse[**M**] = $\mathbf{M}^{-1}$. Allmendinger and others (2012) state that for perfectly constrained cases like this notebook was designed to handle, in which data from just three non-colinear GPS stations are used, the matrix can be inverted using LU decomposition. *Mathematica* has a built-in process for LU decomposition via the function **`LUDecomposition`**[*m*].

What if we want to analyze velocity data from more than three GPS stations? A more generalized form of the 2-D matrices provided above is given by

$$\begin{pmatrix} {}^1u_x \\ {}^1u_y \\ {}^2u_x \\ {}^2u_y \\ {}^3u_x \\ {}^3u_y \\ \vdots \\ {}^nu_x \\ {}^nu_y \end{pmatrix} = \begin{pmatrix} 1 & 0 & {}^1x_o & {}^1y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^1x_o & {}^1y_o \\ 1 & 0 & {}^2x_o & {}^2y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^2x_o & {}^2y_o \\ 1 & 0 & {}^3x_o & {}^3y_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^3x_o & {}^3y_o \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & {}^nx_o & {}^ny_o & 0 & 0 \\ 0 & 1 & 0 & 0 & {}^nx_o & {}^ny_o \end{pmatrix} \begin{pmatrix} t_x \\ t_y \\ e_{xx} \\ e_{xy} \\ e_{yx} \\ e_{yy} \end{pmatrix}$$

Here, the $2n\times6$ matrix (where $n>3$) is the **M** matrix that we must invert to compute values for the 6 unknowns on the right side of the equation. For non-square matrices reflecting over-constrained cases with more GPS data than is minimally required to obtain a solution, Allmendinger and others (2012) recommend using the following formulation that employs transpose **M** matrices (*after* Press and others, 1986, and Menke, 1984)

$$\mathbf{x} = [\mathbf{M}^{\mathrm{T}}\,\mathbf{M}]^{-1}\ \mathbf{M}^{\mathrm{T}}\mathbf{y}$$

The built-in *Mathematica* function **`PseudoInverse`** can also be used to invert both square and rectangular matrices.

Once the components of the deformation gradient tensor are known, we complete the 2-D process as follows. Given the $\mathbf{M}^{-1}$ matrix we just computed, we have the components of the displacement gradient tensor e_{ij}

$$e_{ij} = \begin{pmatrix} e_{xx} & e_{xy} \\ e_{yx} & e_{yy} \end{pmatrix}$$

The 2-D strain tensor is given by

$$\varepsilon_{ij} = \begin{pmatrix} e_{xx} & \frac{(e_{xy} + e_{yx})}{2} \\ \frac{(e_{yx} + e_{xy})}{2} & e_{yy} \end{pmatrix}$$

and the 2-D rotation tensor is

$$\Omega_{ij} = \begin{pmatrix} 0 & \frac{(e_{xy} - e_{yx})}{2} \\ \frac{(e_{yx} - e_{xy})}{2} & 0 \end{pmatrix}$$

The angle of rotation is equal to the length of vector $|\Omega_{xy}| = |\Omega_{yx}|$.

Computation

Insert known data into governing equation to find unknowns

```
In[2]:= matrixM =
     1  0  inputData[[1, 5]]  inputData[[1, 6]]         0                  0
     0  1         0                  0         inputData[[1, 5]]  inputData[[1, 6
     1  0  inputData[[2, 5]]  inputData[[2, 6]]         0                  0
     0  1         0                  0         inputData[[2, 5]]  inputData[[2, 6
     1  0  inputData[[3, 5]]  inputData[[3, 6]]         0                  0
     0  1         0                  0         inputData[[3, 5]]  inputData[[3, 6

     ;
```

```
In[3]:= inverseM = Inverse[matrixM];
```

```
In[4]:= matrixY = {{inputData[[1, 8]]}, {inputData[[1, 7]]},
          {inputData[[2, 8]]}, {inputData[[2, 7]]},
          {inputData[[3, 8]]}, {inputData[[3, 7]]}};
```

In Mathematica, matrix multiplication is done using a dot symbol -- that is, a period -- or by using the built-in function **`Dot`**[*m,k*] where *m* and *k* are the names of the two matrices.

```
In[5]:= matrixX = Flatten[inverseM.matrixY];
```

The matrix **eij** is the 2-D displacement gradient tensor

```
In[6]:= eij =
          {{matrixX[[3]], matrixX[[4]]}, {matrixX[[5]], matrixX[[6]]}};
```

```
In[7]:= displGradTensor = MatrixForm[eij];
```

The matrix **epsij** is the 2-D Lagrangian strain tensor

```
In[8]:= epsij = {{eij[[1, 1]], ((eij[[1, 2]] + eij[[2, 1]]) / 2)},
            {((eij[[2, 1]] + eij[[1, 2]]) / 2), eij[[2, 2]]}};
```

```
In[9]:= lagrangeStrainTensor = MatrixForm[epsij];
```

Rotation

omega12 is the rotation angle in degrees; positive is a counterclockwise rotation.

```
In[10]:= omega12 = ((eij[[1, 2]] - eij[[2, 1]]) / 2) (180 / π);
```

Translation

transCoord is a list of the coordinates of the translation vector

```
In[11]:= transCoord = {matrixX[[1]], matrixX[[2]]};
```

The length of the translation vector (**transDistance**) is given in meters.

```
In[12]:= transDistance = √(transCoord[[1]]^2 + transCoord[[2]]^2);
```

unitTransVect is the unit vector that is coincident with the translation vector.

```
In[13]:= unitTransVect = {(transCoord[[1]] / transDistance),
             (transCoord[[2]] / transDistance)};
```

transAngle is the angle between the north-directed vector and the translation vector.

```
In[14]:= transAngle = ArcCos[northUnitVector.unitTransVect] (180 / π);
```

The azimuth of the translation vector (**transAzimuth**) is measured clockwise from north.

```
In[15]:= transAzimuth =
           If[(unitTransVect[[1]] < 0), (360 - transAngle), transAngle];
```

Greater and lesser horizontal extension axes

vectNorm2D and **unitVector2D** are two user-defined functions that determine the length of a 2-D vector and find the unit vector corresponding to an arbitrary 2-D vector, respectively.

```
In[16]:= vectNorm2D[x_] := √x.x ;
```

```
In[17]:= unitVector2D[x_] :=
           {x[[1]] / vectNorm2D[x], x[[2]] / vectNorm2D[x]};
```

northUnitVector is defined as a unit vector that points north, and from which azimuths can be computed.

```
In[18]:= northUnitVector = {0, 1};
```

eigenVects is the set of eigen vectors for the symmtrical Lagrangian strain tensor. They are unit vectors that coincide with the principal axes of the strain tensor.

```
In[19]:= eigenVects = Eigenvectors[epsij];

In[20]:= eigenVector1 = {eigenVects[[1, 1]], eigenVects[[1, 2]]};

In[21]:= unitEVect1 = unitVector2D[eigenVector1];

In[22]:= eigenVector2 = {eigenVects[[2, 1]], eigenVects[[2, 2]]};

In[23]:= unitEVect2 = unitVector2D[eigenVector2];
```

The eigenvalues are the magnitudes of the principal strains

```
In[24]:= axisLengths = Eigenvalues[epsij];
```

The variable "**a**" is the length of the minimum extension axis of the 2-D horizontal strain ellipse; "**b**" is the length of the maximum extension axis.

Variable **a** is the extension along the maximum extension axis, and variable **a1** is the length of the corresponding semi - major axis after deformation of a circle that had an initial radius of 1.

```
In[25]:= a = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
           axisLengths[[1]], axisLengths[[2]]];

In[26]:= a1 = 1 + a;
```

minExtAxis is the minimum extension axis, defined as being associated with the larger eigenvalue of the two **axisLengths**.

```
In[27]:= minExtAxis = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
           unitEVect1, unitEVect2];
```

angleA is the angle between the north-directed vector and the minimum extension axis.

```
In[28]:= angleA = ArcCos[northUnitVector.minExtAxis] (180 / π);
```

minExtAxisAz is a list providing both options for the azimuth of the minimum extension axis.

```
In[29]:= minExtAxisAz1 = If[(minExtAxis[[1]] < 0), (360 - angleA), angleA];

In[30]:= minExtAxisAz2 = If[(minExtAxisAz1 > 180),
           minExtAxisAz1 - 180, minExtAxisAz1 + 180];

In[31]:= minExtAxisAz = {minExtAxisAz1, minExtAxisAz2};
```

Variable **b** is the extension along the maximum extension axis, and variable **b1** is the length of the corresponding semi - major axis after deformation of a circle that had an initial radius of 1.

```
In[32]:= b = If[(Abs[axisLengths[[1]]] < Abs[axisLengths[[2]]]),
           axisLengths[[1]], axisLengths[[2]]];

In[33]:= b1 = 1 + b;
```

maxExtAxis is the maximum extension axis, defined as being associated with the smaller eigenvalue of the two **axisLengths**.

```
In[34]:= maxExtAxis = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
           unitEVect2, unitEVect1];
```

angleB is the angle between the north-directed vector and the maximum extension axis.

```
In[35]:= angleB = ArcCos[northUnitVector.maxExtAxis] (180 / π);
```

maxExtAxisAz is a list providing both options for the azimuth of the maximum extension axis.

```
In[36]:= maxExtAxisAz1 = If[(maxExtAxis[[1]] < 0), (360 - angleB), angleB];

In[37]:= maxExtAxisAz2 = If[(maxExtAxisAz1 > 180),
            maxExtAxisAz1 - 180, maxExtAxisAz1 + 180];

In[38]:= maxExtAxisAz = {maxExtAxisAz1, maxExtAxisAz2};
```

Area strain

The area of a circle that has a radius of ***r*** is equal to πr^2, and the area of an ellipse with semi-major and semi-minor axes ***a*** and ***b*** is equal to πab.

```
In[39]:= circleArea = π;

In[40]:= ellipseArea = π a1 b1;

In[41]:= areaStrain = (ellipseArea - circleArea) / circleArea;
```

Prepare to plot the results

The angle between the X axis (that is, the east-west axis) of the original GPS-station triangle and the longer principal axis is called "**theta0**".

```
In[42]:= theta0 = If[(minExtAxis[[1]] < 0),
             ((π / 180) (90 - (minExtAxisAz1 - 180))),
             ((π / 180) (90 - minExtAxisAz1))];
```

The angle **theta** combines theta0 with the rigid-body rotation omegaij so that the axes of the resulting plot image are north-south and east-west.

```
In[43]:= theta = theta0 + (omega12 * (π / 180));

In[44]:= rotatedEllipse =
           Table[{(a1 Cos[t] Cos[theta]) + (b1 Sin[t] * (-Sin[theta])),
             (a1 Cos[t] Sin[theta]) + (b1 Sin[t] Cos[theta])},
            {t, 0, 2 π, π / 36}];
```

The plot range should be at least a bit larger than the strain ellipse, and this is handled by a variable called **maxplot**.

```
In[45]:= maxplot = 1.1 a1;
```

The first plot file (plot1) plots a black ellipse with semimajor axis length of "a", semiminor axis length "b", and major axis inclined "theta" degrees relative to the east-directed vector.

```
In[46]:= plot1 = ListLinePlot[rotatedEllipse,
            AspectRatio → 1, AxesLabel → {"East", "North"},
            PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
            PlotStyle → {Black}];
```

The second plot file (plot2) plots a dashed green reference circle.

```
In[47]:= referenceCircle =
           Table[{(Cos[t] Cos[theta]) + (Sin[t] * (-Sin[theta])),
              (Cos[t] Sin[theta]) + (Sin[t] Cos[theta])},
             {t, 0, 2 π, π / 36}];
```

```
In[48]:= plot2 = ListLinePlot[%, AspectRatio → 1,
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Thick, Dashed, Green}];
```

The third plot file (plot3) creates a blue line along the major axis of the ellipse.

```
In[49]:= plot3 = ListLinePlot[{{a1 Cos[theta], a1 Sin[theta]},
              {-a1 Cos[theta], -a1 Sin[theta]}}, AspectRatio → 1,
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Blue}];
```

The fourth plot file (plot4) creates a red line along the minor axis of the ellipse.

```
In[50]:= plot4 = ListLinePlot[
             {{b1 Cos[theta + (π / 2)], b1 Sin[theta + (π / 2)]},
              {-b1 Cos[theta + (π / 2)], -b1 Sin[theta + (π / 2)]}},
             AspectRatio → 1, PlotRange → {{-maxplot, maxplot},
               {-maxplot, maxplot}}, PlotStyle → {Red}];
```

The fifth plot file (plot5) merges the previous four plot files into a single file for display.

```
In[51]:= plot5 = Show[plot2, plot1, plot3, plot4];
```

Results

Stations used

```
In[52]:= {inputData[[1, 1]], inputData[[2, 1]], inputData[[3, 1]]}

Out[52]= {146, 149, 150}
```

Cartesian components of translation vector

```
In[53]:= transCoord

Out[53]= {-0.207876, 0.107522}
```

Length (meters) **and Azimuth** (degrees) **of translation vector**

```
In[54]:= {transDistance, transAzimuth}

Out[54]= {0.234038, 297.35}
```

Displacement gradient tensor

```
In[55]:= displGradTensor
```

Out[55]//MatrixForm=

$$\begin{pmatrix} -1.47675 \times 10^{-8} & 4.77711 \times 10^{-8} \\ -2.86486 \times 10^{-8} & -1.83532 \times 10^{-8} \end{pmatrix}$$

Lagrangian strain tensor

In[56]:= **lagrangeStrainTensor**

Out[56]//MatrixForm=

$$\begin{pmatrix} -1.47675\times 10^{-8} & 9.56129\times 10^{-9} \\ 9.56129\times 10^{-9} & -1.83532\times 10^{-8} \end{pmatrix}$$

Magnitude and azimuth of maximum principal extension

In[57]:= **{b, maxExtAxisAz}**

Out[57]= $\{-6.83245\times 10^{-9}, \{230.31, 50.3102\}\}$

Magnitude and azimuth of minimum principal extension

In[58]:= **{a, minExtAxisAz}**

Out[58]= $\{-2.62883\times 10^{-8}, \{320.31, 140.31\}\}$

Area strain (dilation; negative strain indicates less area after deformation)

In[59]:= **areaStrain**

Out[59]= -3.31208×10^{-8}

Infinitesimal rotation (clockwise + ve)

In[60]:= **omega12**

Out[60]= 2.18926×10^{-6}

In[61]:= **plot5**

Out[61]=

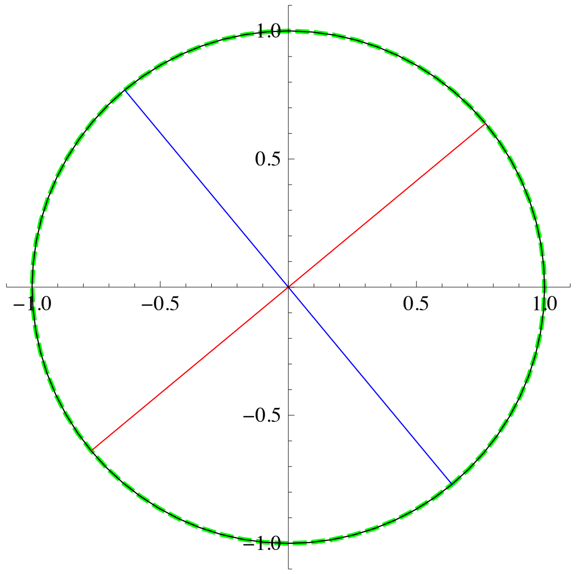

Figure 17-4. Plot of the strain ellipse (black ellipse) relative to the coordinate axes oriented north-south (vertical axis in plot) and east-west (horizontal axis). The axes are labeled in dimensionless strain. The red axis is the minimum extension axis. The green dashed circle shows a circle prior to deformation, so the black ellipse depicts that same circle after deformation.

17.4 Checking our results with SSPX

Introduction

The code presented above is based on Allmendinger, Cardozo and Fisher (2012) and Cardozo and Allmendinger (2009), which also explain the basis for the algorithms implemented in SSPX. Hence, we can check our code against the output of SSPX, given the same input data.

Input data

I created a text file called **`TahoeTest1.txt`** with the following content:

```
X 2 LL
146 -120.537284 39.337459 -0.01025 0.00715
149 -120.104975 39.60212988 -0.00933 0.00555
150 -120.03385482 39.292380598 -0.01107 0.00597
```

The header (first) line indicates that the input data do not include uncertainties (X), that we want a 2-D solution (2) and the locations are provided in longitude latitude form (LL). The three data records include the station identifier, longitude, latitude, E-W velocity (mm/yr) and N-S velocity (mm/yr) separated by spaces.

Running SSPX

The following is a bare-bones set of instructions for obtaining information that we can compare with our results. Start SSPX. Go to the File menu and select **`Load Stations from text`**. Using the appropriate browse window, navigate to the input data file (**`TahoeTest1.txt`**) and select it. Go to the **`Strain`** menu and select **`Best Fit For All`**. Go back to the File menu and select Save Strain as txt, and provide a name for the output file.

SSPX is a feature-rich program that provides graphics and text output, as well as an extensive help menu. You should work your way through the other options so that you can take full advantage of its capabilities.

Comparison of results

Our translation vector: {-0.207876, 0.107522}

SSPX translation vector: {-2.078779e-01 ± 2.088447e-03, 1.075201e-01 ± 2.088447e-03}

Our displacement gradient tensor: $\begin{pmatrix} -1.47675\times10^{-8} & 4.77711\times10^{-8} \\ -2.86486\times10^{-8} & -1.83532\times10^{-8} \end{pmatrix}$

SSPX displacement gradient tensor:
$$\begin{pmatrix} -1.476743\,e-08 \ \pm\ 3.952435\,e-10 & 4.777148\,e-08 \ \pm\ 4.917541\,e-10 \\ -2.864858\,e-08 \ \pm\ 3.952435\,e-10 & -1.835277\,e-08 \ \pm\ 4.917541\,e-10 \end{pmatrix}$$

Our Lagrangian strain tensor: $\begin{pmatrix} -1.47675\times10^{-8} & 9.56129\times10^{-9} \\ 9.56129\times10^{-9} & -1.83532\times10^{-8} \end{pmatrix}$

SSPX Lagrangian strain tensor: $\begin{pmatrix} -1.476743\,e-08 & 9.561450\,e-09 \\ 9.561450\,e-09 & -1.835277\,e-08 \end{pmatrix}$

Our magnitude and orientation of principal extensions:

emax: -6.83245×10^{-9} 230.31° or 50.3102°

emin: -2.62883×10^{-8} 320.31° or 140.31°

SSPX magnitude and orientation of principal extensions.

emax: -6.832045e-09 ± 8.713311e-10 50.31° ± 0.36°

emin: -2.628815e-08 ± 1.566647e-11 320.31° ± 0.36°

Our volume strain: -3.31208×10^{-8}

SSPX volume strain: -3.312019e-08 ± 8.869975e-10

Our infinitesimal rotation axis: 2.18926×10^{-6}

SSPX infinitesimal rotation axis: 2.189274e-06 ± 2.764824e-0.9

SSPX provides an assessment of uncertainty that we have not yet incorporated; however, we have reproduced the results of SSPX with our *Mathematica*-based code with very minor differences.

Let’s clear the previous variables so that we can be certain that we are computing everything anew in the code that follows.

```
In[62]:= Clear[inputData, matrixM, matrixY, matrixX, eij, epsij, omega12,
           transDistance, unitTransVect, transAngle, transAzimuth,
           northUnitVector, eigenVects, eigenVector1, unitEVect1,
           eigenVector2, unitEVect2, axisLengths, a, b, minExtAxis,
           angleA, minExtAxisAz1, minExtAxisAz2, minExtAxisAz,
           maxExtAxis, angleB, maxExtAxisAz1, maxExtAxisAz2, circleArea,
           ellipseArea, areaStrain, theta0, theta, rotatedEllipse,
           referenceCircle, maxplot, plot1, plot2, plot3, plot4, plot5];
```

17.5 Boiling out the fat: *n*-station version without uncertainties

The following is a bare-bones code in *Mathematica* to generate strain information from GPS velocity data for 3 or more stations, based on the preceding discussion.

```
In[63]:=
  inputData =
   146   −120.537284    39.337459    2347.844   712 247.260  4 357 118.219  0.00715  −0.01025  −0.001
   149   −120.104975    39.60212988  2634.6887  748 566.261  4 387 604.030  0.00555  −0.00933  −0.001
   150   −120.03385482  39.292380598 2619.0828  755 806.266  4 353 418.463  0.00597  −0.01107  −0.000
   090   −119.799853081 39.572803815 1503.6541  774 885.45   4 385 237.69   0.00456  −0.00825  −0.000

  ;
```

```
In[64]:= vectNorm2D[x_] := Sqrt[x.x];
```

$$\texttt{vectNorm2D[x\_] := }\sqrt{\texttt{x.x}}\texttt{;}$$

```
In[65]:= unitVector2D[x_] :=
           {x[[1]] / vectNorm2D[x], x[[2]] / vectNorm2D[x]};
```

```
In[66]:= matrixM = Flatten[
            Table[{{1, 0, inputData[[i, 5]], inputData[[i, 6]], 0, 0},
               {0, 1, 0, 0, inputData[[i, 5]], inputData[[i, 6]]}},
              {i, 1, Length[inputData]}], 1];
         matrixY = Flatten[Table[{{inputData[[i, 8]]},
               {inputData[[i, 7]]}}, {i, 1, Length[inputData]}], 1];
         matrixX = Flatten[PseudoInverse[matrixM].matrixY];
         eij =
           {{matrixX[[3]], matrixX[[4]]}, {matrixX[[5]], matrixX[[6]]}};
         epsij = {{eij[[1, 1]], ((eij[[1, 2]] + eij[[2, 1]]) / 2)},
            {((eij[[2, 1]] + eij[[1, 2]]) / 2), eij[[2, 2]]}};
         omega12 = ((eij[[1, 2]] - eij[[2, 1]]) / 2) (180 / π);
```

```
In[72]:= transDistance = √(matrixX[[1]]^2 + matrixX[[2]]^2);
         unitTransVect = {(matrixX[[1]] / transDistance),
              (matrixX[[2]] / transDistance)};
         transAngle = ArcCos[northUnitVector.unitTransVect] (180 / π);
         transAzimuth =
            If[(unitTransVect[[1]] < 0), (360 - transAngle), transAngle];
```

```
In[76]:= eigenVects = Eigenvectors[epsij];
         eigenVector1 = {eigenVects[[1, 1]], eigenVects[[1, 2]]};
         unitEVect1 = unitVector2D[eigenVector1];
         eigenVector2 = {eigenVects[[2, 1]], eigenVects[[2, 2]]};
         unitEVect2 = unitVector2D[eigenVector2];
         axisLengths = Eigenvalues[epsij];
```

```
In[82]:= northUnitVector = {0, 1};
```

```
In[83]:= a = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
            axisLengths[[1]], axisLengths[[2]]];
         minExtAxis = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
            unitEVect1, unitEVect2];
         angleA = ArcCos[northUnitVector.minExtAxis] (180 / π);
         minExtAxisAz1 = If[(minExtAxis[[1]] < 0), (360 - angleA), angleA];
         minExtAxisAz2 = If[(minExtAxisAz1 > 180),
            minExtAxisAz1 - 180, minExtAxisAz1 + 180];
         minExtAxisAz = {minExtAxisAz1, minExtAxisAz2};
         b = If[(Abs[axisLengths[[1]]] < Abs[axisLengths[[2]]]),
            axisLengths[[1]], axisLengths[[2]]];
         maxExtAxis = If[(Abs[axisLengths[[1]]] > Abs[axisLengths[[2]]]),
            unitEVect2, unitEVect1];
         angleB = ArcCos[northUnitVector.maxExtAxis] (180 / π);
         maxExtAxisAz1 = If[(maxExtAxis[[1]] < 0), (360 - angleB), angleB];
         maxExtAxisAz2 = If[(maxExtAxisAz1 > 180),
            maxExtAxisAz1 - 180, maxExtAxisAz1 + 180];
         maxExtAxisAz = {maxExtAxisAz1, maxExtAxisAz2};
```

```
In[95]:= circleArea = π;
         ellipseArea = π (1 + a) (1 + b);
         areaStrain = (ellipseArea - circleArea) / circleArea;

In[98]:= theta0 =
           If[(minExtAxis[[1]] < 0), ((π / 180) (90 - (minExtAxisAz1 - 180))),
            ((π / 180) (90 - minExtAxisAz1))];
         theta = theta0 + (omega12 * (π / 180));

In[100]:= rotatedEllipse = Table[
             {((1 + a) Cos[t] Cos[theta]) + ((1 + b) Sin[t] * (-Sin[theta])),
              ((1 + a) Cos[t] Sin[theta]) + ((1 + b) Sin[t] Cos[theta])},
             {t, 0, 2 π, π / 36}];
          referenceCircle = Table[
             {(Cos[t] Cos[theta]) + (Sin[t] * (-Sin[theta])),
              (Cos[t] Sin[theta]) + (Sin[t] Cos[theta])}, {t, 0, 2 π, π / 36}];

In[102]:= maxplot = 1.1 (1 + a);

In[103]:= plot1 = ListLinePlot[rotatedEllipse,
             AspectRatio → 1, AxesLabel → {"East", "North"},
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Black}];
          plot2 = ListLinePlot[referenceCircle, AspectRatio → 1,
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Thick, Dashed, Green}];
          plot3 = ListLinePlot[{{(1 + a) Cos[theta], (1 + a) Sin[theta]},
              {- (1 + a) Cos[theta], - (1 + a) Sin[theta]}}, AspectRatio → 1,
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Blue}];
          plot4 = ListLinePlot[{{(1 + b) Cos[theta + (π / 2)],
                (1 + b) Sin[theta + (π / 2)]}, {- (1 + b) Cos[theta + (π / 2)],
                - (1 + b) Sin[theta + (π / 2)]}}, AspectRatio → 1,
             PlotRange → {{-maxplot, maxplot}, {-maxplot, maxplot}},
             PlotStyle → {Red}];
          plot5 = Show[plot2, plot1, plot3, plot4];
```

Results

Stations used

```
In[108]:= Table[inputData[[i, 1]], {i, 1, Length[inputData]}]

Out[108]= {146, 149, 150, 90}
```

Cartesian components of translation vector

```
In[109]:= {matrixX[[1]], matrixX[[2]]}

Out[109]= {-0.264769, 0.117573}
```

Length (meters) **and Azimuth** (degrees) **of translation vector**

```
In[110]:= {transDistance, transAzimuth}

Out[110]= {0.289699, 293.944}
```

Displacement gradient tensor

```
In[111]:= MatrixForm[eij]

Out[111]//MatrixForm=
```

$$\begin{pmatrix} 2.23318\times10^{-9} & 5.7969\times10^{-8} \\ -3.16519\times10^{-8} & -2.01548\times10^{-8} \end{pmatrix}$$

Lagrangian strain tensor

```
In[112]:= MatrixForm[epsij]

Out[112]//MatrixForm=
```

$$\begin{pmatrix} 2.23318\times10^{-9} & 1.31586\times10^{-8} \\ 1.31586\times10^{-8} & -2.01548\times10^{-8} \end{pmatrix}$$

Magnitude and azimuth of maximum principal extension

```
In[113]:= {b, maxExtAxisAz}
```

Out[113]= $\{8.31499\times10^{-9}, \{245.194, 65.1939\}\}$

Magnitude and azimuth of minimum principal extension

```
In[114]:= {a, minExtAxisAz}
```

Out[114]= $\{-2.62366\times10^{-8}, \{335.194, 155.194\}\}$

Area strain (dilation; negative strain indicates less area after deformation)

```
In[115]:= areaStrain
```

Out[115]= -1.79216×10^{-8}

Infinitesimal rotation (clockwise + ve)

```
In[116]:= omega12
```

Out[116]= 2.56745×10^{-6}

```
In[117]:= plot5
```

Out[117]=

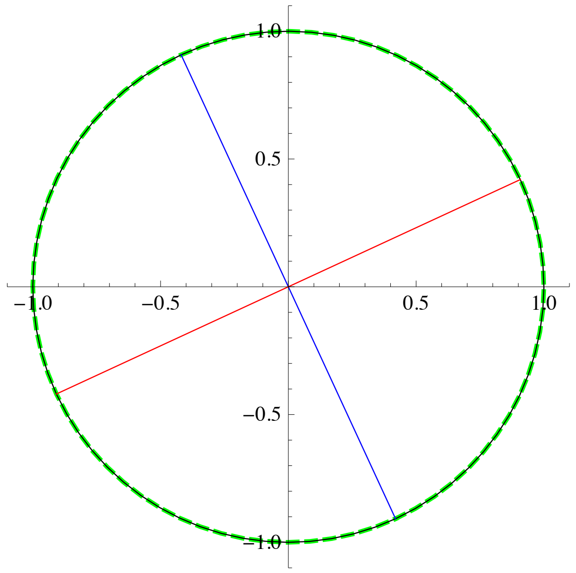

Figure 16-5. Plot of the strain ellipse (black ellipse) relative to the coordinate axes oriented north-south (vertical axis in plot) and east-west (horizontal axis). The axes are labeled in dimensionless strain. The red axis is the minimum extension axis. The green dashed circle shows a circle prior to deformation, so the black ellipse depicts that same circle after deformation.

17.6 References

Allmendinger, R.W., Loveless, J.P., Pritchard, M.E., and Meade, B., 2009, From decades to epochs -- Spanning the gap between geodesy and structural geology of active mountain belts: Journal of Structural Geology, v. 31, p. 1409-1422, doi: 10.1016/j.jsg.2009.08.008.

Allmendinger, R.W., Cardozo, N, and Fisher, D.M., 2012, **Structural Geology Algorithms, Vectors and Tensors** : Cambridge University Press, 289 p., ISBN 978 - 1 - 107 - 40138 - 9..

Cardozo, N., and Allmendinger, R.W., 2009, SSPX -- A program to compute strain from displacement/velocity data: Computers and Geosciences, v. 35, p. 1343-1357, doi: 10.1016/j.cageo.2008.05.008.

Ferguson, John, 1994, **Introduction to Linear Algebra in Geology** : London, Chapman & Hall, 203 p., ISBN 0 - 412 - 49350 - 0.

Fossen, Haakon, 2010, **Structural Geology** : New York, Cambridge University Press, 463 p., ISBN 978 - 0 - 521 - 51664 - 8.

Frank, F.C., Deduction of Earth strains from survey data: Bulletin of the Seismological Society of America, v. 56, no. 1, p. 35-42.

Jaeger, J.C., 1964, Elasticity, Fracture and Flow: Methuen, New York, 212 p.

Malvern, L.E., 1969, Introduction to the mechanics of a continuous medium: Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 713 p.,

Menke, W., 1984, **Geophysical Data Analysis -- Discrete Inverse Theory**: Orlando, Florida, Academic Press.

Oertel, G., 1996, **Stress and Deformation -- A Handbook on Tensors in Geology** : New York, Oxford University Press, 292 p., ISBN 0 - 19 - 509503 - 0.

Prescott, W.H., 1976, An extension of Frank's method for obtaining crustal shear strains from survey data: Bulletin of the Seismological Society of America, v. 66, no. 6, p. 1847-1853.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., 1986, **Numerical Recipes -- The Art of Scientific Computing**: Cambridge, Cambridge University Press.

Pollard, D.D., and Fletcher, R.C., 2005, **Fundamentals of Structural Geology** : New York, Cambridge University Press, 500 p., ISBN 0 - 521 - 83927 - 0; see especially chapters 2 and 5.

Savage, J.C., Gan, W., and Svarc, J.L., 2001, Strain accumulation and rotation in the Eastern California Shear Zone: Journal of Geophysical Research, v. 106, B10, P. 21,995-22,007.

Wolfram MathWorld has a web resource on eigenvectors and eigen decomposition available at
http : // mathworld.wolfram.com/Eigenvector.html.

Other *Mathematica* resources are available for download at

Emmanuel Amiot, "Eigenvectors by Hand" from The Wolfram Demonstrations Project http://demonstrations.wolfram.com/EigenvectorsByHand/

Yaroslav Bulatov, "Linear Transformation with Given Eigenvectors" from The Wolfram Demonstrations Project http://demonstrations.wolfram.com/LinearTransformationWithGivenEigenvectors/

——————————————————————————————

Postscript

J.C. Savage offered me the following advice in an email message:

"For your purposes the formulation in spherical coordinates in my Eastern California shear zone paper is too complicated. Nor do I believe it is worthwhile to include the vertical motions. Here is a simplified version for the horizontal strains which is as accurate as justified by local (within 200 km) data :
Calculate homogeneous strain approximation to observed velocities
1. Convert the latitude and longitude of your stations to grid coordinates (northing and easting).Try Google search under "convert latitude longitude to utm" for automatic conversions.
2. Translate to center of mass coordinates.x' = x - xbar and y' = y - ybar where xbar and ybar are the averages of x and y, respectively.
3. I presume you have the east vE and north vN velocities of the stations from PBO. Then solve the following equations (Jaeger, J.C., Elasticity, Fracture, and Flow, Methuen, London, 1964, p.39) by least squares for the strain (exx, exy, and eyy) and rotation (w) rates :
vE = exx x' + exy y' - w y'
vN = exy x' + eyy y' + w x'
Notice I use tensor strain rates not engineering strain as used by Jaeger. There will be a pair of those equations for each station, and you will need at least three stations to solve for the 4 unknowns. More stations will increase the redundancy and give better estimates of standard deviations.
4. Calculate principal strain rates from exx, exy, and eyy. Setting up the least squares solution to the 6 or more equations will depend to some extent upon the least squares program that you have.".

——————————————————————————————

Bill Hammond offered the following in an email message:

"Here's what I normally do. I use a little matlab script I wrote a while back that takes the lat, lon, north velocity, east velocity (and uncertainties), and it spits out the stain and rotation rates. This is based on the appendix of Savage et al., 2001, which takes the spherical geometry of the Earth surface into account. You probably wanted something simpler, maybe even in 2D, but I realized that by the time it was all written out in 2D, you might as well do it on a sphere. Perhaps what this subroutine lacks in simplicity is made up for by its ease of use (if you have matlab)."

The MatLab script that Bill provided is available via
http://bearspace.baylor.edu/Vince_Cronin/www/GradStruct/GradStructHome12.html