gps-strain-calculator-mathematica.nb

Code to determine the 2-D (horizontal) instantaneous or infinitesimal strain from velocity or displacement data from 3 adjacent-and-non-colinear GNSS stations

Coded in *Mathematica* 8 by Vince Cronin with help from Phil Resor, circa 2012. This version is coded in *Mathematica* 11, and was revised 13 October 2018.
The most current version of this code is accessible via
http://croninprojects.org/Vince/Codes/gps_strain_calculator_mathematica.nb

Input

Data for this analysis can be obtained from many sources, but you must be certain that the velocities for all three sites are expressed with respect to the same reference frame (e.g., NAM08 or IGS08).
The Plate Boundary Observatory GPS data are available online through the good work of UNAVCO via
http://www.unavco.org/instrumentation/networks/status/pbo/gps
Each row of the input data contains the following data types : GPS station number, longitude (west is negative), latitude (south is negative), E-W velocity (mm/yr), N-S velocity (mm/yr), E-W velocity uncertainty (mm/yr), N-S velocity uncertainty (mm/yr).
The following is an example of a valid input matrix.

```
inData = {{P574, -117.63389, 34.28677, -19.23, 19.61, 0.16, 0.19},
          {P577, -117.31891, 34.30461, -14.51, 16.15,  0.9,  0.09},
          {EWPP, -117.52559, 34.10420, -21.83, 22.56, 0.06, 0.07}};
```

Replace the data in the inData matrix below with your values.

In[1]:=
```
inData = {{P574, -117.63389, 34.28677, -19.23, 19.61, 0.16, 0.19},
          {EWPP, -117.52559, 34.10420, -21.83, 22.56, 0.06, 0.07},
          {P577, -117.31891, 34.30461, -14.51, 16.15,  0.9,  0.09}};
```

UTM Central Meridian

Input the central meridian that is spatially appropriate for the three PBO sites used in this analysis. If the three points involve two adjacent UTM zones, input the central meridian of the

zone to the west.

Longitude Range	Central Meridian
−138 to − 132	−135
−132 to − 126	−129
−126 to − 120	−123
−120 to − 114	−117
−114 to − 108	−111
−108 to − 102	−105
−102 to − 96	−99

Replace the value for longCentMeridian below with your value.

In[2]:=
```
longCentMeridian = -117;
```

The results are in the output section at the end of this notebook

User - Defined Functions

The purpose of the following function is to convert input data from geographic coordinates (latitude and longitude) to UTM using the UTM84 datum. Online converters are available via https://www.ngs.noaa.gov/NCAT/, http://www.rcn.montana.edu/Resources/Converter.aspx, https://awsm-tools.com/geo/utm-to-geographic, and elsewhere. The output data are

In[3]:=
```
geog2UTM[inputLat_, inputLong_] :=
  Module[{c42, c43, c44, c45, c46, c47, c48, c49, c50, c51, c53, c54, c56,
    c57, c58, c59, c60, c61, c62, c63, c65, c66, c67, c68, c69, c71, c72,
    c73, c74, c75, c78, c79, c80, c81, outData}, c42 = inputLat (π / 180);
   c43 = inputLong (π / 180);
   c44 = (inputLong + 180) / 6;
   c45 =
    If[(c44 - IntegerPart[c44]) > 0, (IntegerPart[c44] + 1), IntegerPart[c44]];
   (* c45 is the central meridian used to compute the location *)
   c46 = -183 + (6 * c45);
   c47 = c46 (π / 180);
   c48 = If[(c46 == -177), 177, c46 - 6];
   c49 = c48 (π / 180);
   c50 = (c48 + 180) / 6;
   c51 = If[(c50 - IntegerPart[c50]) > 0, IntegerPart[c50] + 1, IntegerPart[c50]];
   c53 = 6 378 137;
   c54 = 6 356 752.3142;
   c56 = 0.9996;
   c57 = √(1 - c54^2 / c53^2);
   c58 = ((c57 * c53) / c54)^2;
   c59 = (c53 - c54) / (c53 + c54);
```

```
c60 = (c53 * (1 - c57^2)) / ((1 - (c57^2 * Sin[c42]^2))^1.5);
c61 = c53 / Sqrt[1 - (c57^2 * Sin[c42]^2)];
c62 = c43 - c47;
c63 = If[(c48 == 177), (Abs[c43] - c49), (c43 - c49)];
c65 = c42 * (1 - (c57^2 / 4) - ((3 * c57^4) / 64) - ((5 * c57^6) / 256));
c66 = Sin[2 * c42] * (((3 * c57^2) / 8) + ((3 * c57^4) / 32) + ((45 * c57^6) / 1024));
c67 = Sin[4 * c42] * (((15 * c57^4) / 256) + ((45 * c57^6) / 1024));
c68 = Sin[6 * c42] * ((35 * c57^6) / 3072);
c69 = c53 * (c65 - c66 + c67 - c68);
c71 = c69 * c56;
c72 = c56 * c61 * Sin[2 * c42] / 4;
c73 = (c56 * c61 * Sin[c42] * (Cos[c42]^3) / 24) *
   (5 - (Tan[c42]^2) + (9 * c58 * (Cos[c42]^2)) + (4 * c58^2 * Cos[c42]^4));
c74 = c56 * c61 * Cos[c42];
c75 = (c56 * c61 * (Cos[c42]^3) / 6) * (1 - Tan[c42]^2 + (c58 * Cos[c42]^2));
c78 = c71 + (c72 * c62^2) + (c73 * c62^4);
(* c78 is the directly computed northing*)
c79 = 500 000 + (c74 * c62) + (c75 * c62^3);
(* c79 is the directly computed easting*)
c80 = c71 + (c72 * c63^2) + (c73 * c63^4);
(* c80 is the pseudo-
 northing based on the central meridian to the west *)
c81 = 500 000 + (c74 * c63) + (c75 * c63^3);
(* c81 is the pseudo-
 easting based on the central meridian to the west *)
outData = {c78, c79, c80, c81, c45};
outData];
```

The purpose of the following function is to jointly consider the UTM coordinates of three sites that might be in two adjacent UTM zones and re - express their coordinates in terms of the west - most zone of the set.

In[4]:=
```
triGroupUTM[site1_, site2_, site3_] :=
  Module[{d83, utmZones, e83, c83, c87, d87, e87, c88, d88, e88, finalEastings,
    finalNorthings, answer}, d83 = (site1[[5]] + site2[[5]] + site3[[5]]) / 3;
   utmZones = {site1[[5]], site2[[5]], site3[[5]]};
   e83 = StandardDeviation[utmZones];
   c83 = If[e83 > 5, 60, Floor[d83]];
   c87 = If[site1[[5]] == c83, site1[[2]], site1[[4]]];
   d87 = If[site2[[5]] == c83, site2[[2]], site2[[4]]];
   e87 = If[site3[[5]] == c83, site3[[2]], site3[[4]]];
   c88 = If[site1[[5]] == c83, site1[[1]], site1[[3]]];
   d88 = If[site2[[5]] == c83, site2[[1]], site2[[3]]];
   e88 = If[site3[[5]] == c83, site3[[1]], site3[[3]]];
   finalEastings = {c87, d87, e87};
   finalNorthings = {c88, d88, e88};
   answer = {finalEastings, finalNorthings}; answer
  ];
```

The purpose of the following function is to create a plot of the infinitesimal horizontal strain ellipse, exaggerated by a factor of one million.

In[5]:=
```
StrainEllipse[maxHextension_, minHextension_, azimuthS1H_, azimuthS2H_] :=
  Module[{a, circleData, circle, a2, b, line1, line2, redLine, blueLine,
    ellipseData, zMatrix, rotatedEllipseData, ellipsePlot}, a = 1;
   circleData = Table[{a Cos[i], a Sin[i]}, {i, 0, 2 π, (π / 50)}];
   circle = Graphics[Line[circleData], PlotRange → 2,
     Axes → True, LabelStyle → {FontFamily → "Arial", 10, GrayLevel[0]},
     AxesLabel → {HoldForm["East"], HoldForm["North"]}, PlotLabel →
      HoldForm["Strain ellipse (10^6 exaggeration)"], ImageSize → Large];
   a2 = 1 + (maxHextension * 0.001);
   b = 1 + (minHextension * 0.001);
   line1 = Line[{{a2 * Sin[azimuthS1H Degree], a2 * Cos[azimuthS1H Degree]},
      {-a2 * Sin[azimuthS1H Degree], -a2 * Cos[azimuthS1H Degree]}}];
   line2 = Line[{{b * Sin[azimuthS2H Degree], b * Cos[azimuthS2H Degree]},
      {-b * Sin[azimuthS2H Degree], -b * Cos[azimuthS2H Degree]}}];
   redLine = Graphics[{Red, line1}];
   blueLine = Graphics[{Blue, line2}];
   ellipseData = Table[{b Cos[i], a2 Sin[i]}, {i, 0, 2 π, (π / 50)}];
   zMatrix = ( Cos[-azimuthS1H Degree]  -Sin[-azimuthS1H Degree]
               Sin[-azimuthS1H Degree]   Cos[-azimuthS1H Degree] );
   rotatedEllipseData =
    Flatten[Table[{zMatrix.ellipseData[[i]]}, {i, 1, 101}], 1];
   ellipsePlot = Graphics[Line[rotatedEllipseData], PlotRange → 2,
     Axes → True, LabelStyle → {FontFamily → "Arial", 10, GrayLevel[0]},
     AxesLabel → {HoldForm["East"], HoldForm["North"]}, PlotLabel →
      HoldForm["Strain ellipse (10^6 exaggeration)"], ImageSize → Large];
   Show[circle, redLine, blueLine, ellipsePlot]];
```

The purpose of the following function is to determine the azimuth and speed (magnitude) of a total velocity vector, given the east and north velocity components. The units output are degrees of azimuth and total speed in the same units as the input velocity values (e.g., mm/yr).

In[6]:=
```
azimuthNSpeed[eastVelocity_, northVelocity_] :=
  Module[{unitNorthVector, totalVelVector, totalSpeed, unitTotalVelVector,
    angle, azTotalVelVector, answer}, unitNorthVector = {0, 1};
   totalVelVector = {eastVelocity, northVelocity};
   totalSpeed = Norm[totalVelVector];
   unitTotalVelVector = {eastVelocity / totalSpeed, northVelocity / totalSpeed};
   angle = ArcCos[unitTotalVelVector.unitNorthVector] (180 / π);
   azTotalVelVector = If[(unitTotalVelVector[[1]] < 0), 360 - angle, angle];
   answer = {azTotalVelVector, totalSpeed};
   answer];
```

Calculations

The steps referenced below are described in the document ***Algorithm for computing infinitesimal strain rate between three non-colinear GPS stations, given their N-S and E-W velocities, with a worked example*** (Vince Cronin and Phil Resor, 2012; revised 2018). The most current version of that document is available online via http://croninprojects.org/Vince/Geodesy/Current-Strain-Calculators/TriangleStrainAlgorithm.pdf

Step 1: Convert latitude & longitude to UTM

In[7]:=
```
site1 = inData[[1, 1]];
site2 = inData[[2, 1]];
site3 = inData[[3, 1]];
```

In[8]:=
```
site1UTM = geog2UTM[inData[[1, 3]], inData[[1, 2]]];
```

In[9]:=
```
site2UTM = geog2UTM[inData[[2, 3]], inData[[2, 2]]];
```

In[10]:=
```
site3UTM = geog2UTM[inData[[3, 3]], inData[[3, 2]]];
```

In[11]:=
```
finalUTMcoord = triGroupUTM[site1UTM, site2UTM, site3UTM];
```

In[12]:=
```
eastings =
  {finalUTMcoord[[1, 1]], finalUTMcoord[[1, 2]], finalUTMcoord[[1, 3]]};
```

In[13]:=
```
northings =
  {finalUTMcoord[[2, 1]], finalUTMcoord[[2, 2]], finalUTMcoord[[2, 3]]};
```

Step 2: Find the center of the triangle defined by the three sites

In[14]:=
$$\text{meanX} = \frac{1}{3}\left(\text{eastings}[[1]] + \text{eastings}[[2]] + \text{eastings}[[3]]\right);$$

In[15]:=
$$\text{meanY} = \frac{1}{3}\left(\text{northings}[[1]] + \text{northings}[[2]] + \text{northings}[[3]]\right);$$

In[16]:=
$$\text{meanLat} = \left(\text{inData}[[1, 3]] + \text{inData}[[2, 3]] + \text{inData}[[3, 3]]\right) / 3;$$

In[17]:=
$$\text{meanLong} = \left(\text{inData}[[1, 2]] + \text{inData}[[2, 2]] + \text{inData}[[3, 2]]\right) / 3;$$

Steps 3 & 4: Define the initial locations relative to the center of the triangle

In[18]:=
$$\text{m1} = \begin{pmatrix} \text{eastings}[[1]] - \text{meanX} & \text{northings}[[1]] - \text{meanY} \\ \text{eastings}[[2]] - \text{meanX} & \text{northings}[[2]] - \text{meanY} \\ \text{eastings}[[3]] - \text{meanX} & \text{northings}[[3]] - \text{meanY} \end{pmatrix};$$

The first row of matrix **m1** (above) contains the east - west and north - south coordinates of site 1 in a coordinate system whose origin is the center of the triangle. Units are meters, and the coordinate grid is aligned with the local UTM grid.

Step 5

In[19]:=
```
m2 = {{1, 0, -m1[[1, 2]], m1[[1, 1]], m1[[1, 2]], 0},
      {0, 1, m1[[1, 1]], 0, m1[[1, 1]], m1[[1, 2]]},
      {1, 0, -m1[[2, 2]], m1[[2, 1]], m1[[2, 2]], 0},
      {0, 1, m1[[2, 1]], 0, m1[[2, 1]], m1[[2, 2]]},
      {1, 0, -m1[[3, 2]], m1[[3, 1]], m1[[3, 2]], 0},
      {0, 1, m1[[3, 1]], 0, m1[[3, 1]], m1[[3, 2]]}};
```

Step 6: Invert matrix m2 to form matrix m3

In[20]:=
```
m3 = Inverse[m2];
```

Step 7: Create a column matrix called m4 that contains the velocity data for the three GPS sites (m/yr)

The input data provides the magnitude of east-west and north-south components of the site velocity vector in a geographic coordinate system. When these site data are converted to UTM coordinates, the divergence between the grid north and true north can be significant, so the site velocity vectors must be transformed into the UTM coordinate system using the grid convergence angle at each site.

Angular difference between grid north and true north for each site

Answers are given in degrees. Positive values indicate that true north is a positive (anti-clockwise) rotation from grid north. Negative values indicate that true north is a negative (clockwise) rotation from grid north.

In[21]:=
```
site1GridConvergence = ArcTan[Tan[(inData[[1, 2]] - longCentMeridian) Degree]
      Sin[inData[[1, 3]] Degree]] (180 / π);
```

In[22]:=
```
site2GridConvergence = ArcTan[Tan[(inData[[2, 2]] - longCentMeridian) Degree]
      Sin[inData[[2, 3]] Degree]] (180 / π);
```

In[23]:=
```
site3GridConvergence = ArcTan[Tan[(inData[[3, 2]] - longCentMeridian) Degree]
      Sin[inData[[3, 3]] Degree]] (180 / π);
```

The grid convergence at the centroid of the triangle, represented by `centroidGridConvergence`, is given in degrees. Positive values indicate that true north is a positive (anti-clockwise) rotation from grid north. Negative values indicate that true north is a negative (clockwise) rotation from grid north.

In[24]:=
```
centroidGridConvergence = ArcTan[
    Tan[(meanLong - longCentMeridian) Degree] Sin[meanLat Degree]] (180 / π);
```

In[25]:=
```
transformSite1Vector =
  ( Cos[site1GridConvergence Degree]   Sin[site1GridConvergence Degree] ) .
  ( -Sin[site1GridConvergence Degree]  Cos[site1GridConvergence Degree] )
   ( inData[[1, 4]] )
   ( inData[[1, 5]] );
```

In[26]:=
```
transformSite2Vector =
  ( Cos[site2GridConvergence Degree]   Sin[site2GridConvergence Degree] ) .
  ( -Sin[site2GridConvergence Degree]  Cos[site2GridConvergence Degree] )
   ( inData[[2, 4]] )
   ( inData[[2, 5]] );
```

In[27]:=
```
transformSite3Vector =
  ( Cos[site3GridConvergence Degree]   Sin[site3GridConvergence Degree] ) .
  ( -Sin[site3GridConvergence Degree]  Cos[site3GridConvergence Degree] )
   ( inData[[3, 4]] )
   ( inData[[3, 5]] );
```

The division of each input value by 1000 achieves a conversion from mm/yr to m/yr (or from mm to m, if the problem is framed in terms of an average annual displacement)

In[28]:=
```
m4 = ( transformSite1Vector[[1, 1]] / 1000
       transformSite1Vector[[2, 1]] / 1000
       transformSite2Vector[[1, 1]] / 1000
       transformSite2Vector[[2, 1]] / 1000
       transformSite3Vector[[1, 1]] / 1000
       transformSite3Vector[[2, 1]] / 1000 );
```

Step 8: Multiply matrix m4 by matrix m3 to yield a 6x1 column matrix called m5

In[29]:=
```
m5 = m3.m4;
```

Step 9: Find the infinitesimal translation vector for the triangle defined by the three GPS stations

In[30]:=
```
translationVector = {m5[[1, 1]], m5[[2, 1]]};
```

The coordinates of the **translationVector** (above) are {east-west velocity, north-south velocity} expressed in meters per year, relative to the stable North American reference frame (NAM08). This can be considered either a velocity vector (m/yr) or an average annual displacement vector (m).

In[31]:=
```
translationSpeed = Sqrt[m5[[1, 1]]^2 + m5[[2, 1]]^2];
```

The translation speed represented by **translationSpeed** (above) is expressed in meters per year, relative to the stable North American reference frame (NAM08). This can be considered either a speed (m/yr) or an average annual displacement (m). Translation speed represented by **translationSpeedMM** (below) is expressed in millimeters/year.

In[32]:=
```
translationSpeedMM = 1000 * translationSpeed;
```

In[33]:=
```
unitTransVect = {translationVector[[1]] / Norm[translationVector], translationVector[[2]] / Norm[translationVector]};
```

In[34]:=
```
unitNorthVector = {0, 1};
```

In[35]:=
```
angle = (ArcCos[unitTransVect.unitNorthVector] (180/π)) + centroidGridConvergence;
```

In[36]:=
```
azTranslationVect = If[(unitTransVect[[1]] < 0), 360 - angle, angle];
```

In[37]:=
```
N[azTranslationVect];
```

The azimuth of the translation vector represented by **azTranslationVect** (above) is given in degrees, and is the horizontal angle measured clockwise from north to the direction the translation vector points.

Interlude: Calculations related to creating the graphics associated with a typical GNSS-crustal-strain presentation, assuming you already have a map graphic that shows the site locations

Note : The site number (1, 2, or 3) is defined by the order in which the site data is listed in the input table (first row of input data is site 1)

Velocities in the North American reference frame (mm/yr)

The first number in each output pair is the azimuth in degrees; the second number is the speed in millimeters per year (or average annual displacement in meters) relative to the NAM08 reference frame.

In[38]:=
```
site1AzimuthNSpeed = azimuthNSpeed[inData[[1, 4]], inData[[1, 5]]];
```

In[39]:=
```
site2AzimuthNSpeed = azimuthNSpeed[inData[[2, 4]], inData[[2, 5]]];
```

In[40]:=
```
site3AzimuthNSpeed = azimuthNSpeed[inData[[3, 4]], inData[[3, 5]]];
```

The output of each of the preceding three lines of code are the site ID, azimuth, speed (mm/yr), east velocity, and north velocity relative to the NAM08 reference frame.

Velocities relative to the centroid of the GNSS station triangle (mm/yr), with the translation vector removed

In[41]:=
```
site1NormAzNSpeed =
  azimuthNSpeed[(inData[[1, 4]] - (1000 * translationVector[[1]])),
   (inData[[1, 5]] - (1000 * translationVector[[2]]))];
```

In[42]:=
```
site2NormAzNSpeed =
  azimuthNSpeed[(inData[[2, 4]] - (1000 * translationVector[[1]])),
   (inData[[2, 5]] - (1000 * translationVector[[2]]))];
```

In[43]:=
```
site3NormAzNSpeed =
  azimuthNSpeed[(inData[[3, 4]] - (1000 * translationVector[[1]])),
   (inData[[3, 5]] - (1000 * translationVector[[2]]))];
```

The output of each of the preceding three lines of code are the site ID, azimuth, and speed (mm/yr) relative to the center of the triangle defined by the three PBO sites.

Distance between sites before displacement (km)

Distance between the first site and the second site (km)

In[44]:=
```
sideLength12km =
  √((m1[[1, 1]] - m1[[2, 1]])^2 + (m1[[1, 2]] - m1[[2, 2]])^2) / 1000;
```

Distance between the second site and the third site (km)

In[45]:=
```
sideLength23km =
  √((m1[[2, 1]] - m1[[3, 1]])^2 + (m1[[2, 2]] - m1[[3, 2]])^2) / 1000;
```

Distance between the first site and the third site (km)

In[46]:=
```
sideLength13km =
  √((m1[[1, 1]] - m1[[3, 1]])^2 + (m1[[1, 2]] - m1[[3, 2]])^2) / 1000;
```

Distance between sites after displacement (km)

Distance between the first site and the second site (km)

In[47]:=
```
sideLengthEnd12km =
  √((m10[[1, 1]] - m10[[2, 1]])^2 + (m10[[1, 2]] - m10[[2, 2]])^2) / 1000;
```

... Part: Part specification m10[[1, 1]] is longer than depth of object.

... Part: Part specification m10[[2, 1]] is longer than depth of object.

... Part: Part specification m10[[1, 2]] is longer than depth of object.

... General: Further output of Part::partd will be suppressed during this calculation.

Distance between the second site and the third site (km)

In[48]:=
```
sideLengthEnd23km =
  √((m10[[2, 1]] - m10[[3, 1]])^2 + (m10[[2, 2]] - m10[[3, 2]])^2) / 1000;
```

... Part: Part specification m10[[2, 1]] is longer than depth of object.

... Part: Part specification m10[[3, 1]] is longer than depth of object.

... Part: Part specification m10[[2, 2]] is longer than depth of object.

... General: Further output of Part::partd will be suppressed during this calculation.

Distance between the first site and the third site (km)

In[49]:=
```
sideLengthEnd13km =
  √((m10[[1, 1]] - m10[[3, 1]])^2 + (m10[[1, 2]] - m10[[3, 2]])^2) / 1000;
```

... Part: Part specification m10[[1, 1]] is longer than depth of object.

... Part: Part specification m10[[3, 1]] is longer than depth of object.

... Part: Part specification m10[[1, 2]] is longer than depth of object.

... General: Further output of Part::partd will be suppressed during this calculation.

End Interlude

Step 10: Find the value of the rotational velocity, Ω, in matrix m5: the value in $m5_{31}$

In[50]:=
```
angularVelocityRad = m5[[3, 1]];
```

In[51]:=
```
angVelNanoRadYr = angularVelocityRad / 10^-9;
```

Step 11: Define the 2-D Lagrangian strain tensor (ε_{ij}) and call it matrix m6

In[52]:=
```
m6 = ( m5[[4, 1]]  m5[[5, 1]]
       m5[[5, 1]]  m5[[6, 1]] );
```

Step 12: Determine the orientations of the principal infinitesimal strain axes by finding the eigenvectors of the symmetrical Lagrangian infinitesimal strain tensor

In[53]:=
```
eValues = Eigenvalues[m6];
```

In[54]:=
```
e1 = If[eValues[[1]] > eValues[[2]], eValues[[1]], eValues[[2]]];
```

In[55]:=
```
e2 = If[eValues[[1]] > eValues[[2]], eValues[[2]], eValues[[1]]];
```

In[56]:=
```
e1 - e2;
```

In[57]:=
```
eVectors = Eigenvectors[m6];
```

In[58]:=
```
s1V = If[eValues[[1]] > eValues[[2]], eVectors[[1]], eVectors[[2]]];
```

In[59]:=
```
s1UV = {s1V[[1]] / Norm[s1V], s1V[[2]] / Norm[s1V]};
```

In[60]:=
```
s2V = If[eValues[[1]] > eValues[[2]], eVectors[[2]], eVectors[[1]]];
```

In[61]:=
```
s2UV = {s2V[[1]] / Norm[s2V], s2V[[2]] / Norm[s2V]};
```

In[62]:=
```
angleS1HAxis = (ArcCos[unitNorthVector.s1UV] (180 / π)) + centroidGridConvergence;
```

In[63]:= azimuthS1HAxis1 = If[(s1UV[[1]] < 0), (360 - angleS1HAxis), angleS1HAxis];

In[64]:= azimuthS1HAxis2 =
  If[azimuthS1HAxis1 > 180, azimuthS1HAxis1 - 180, azimuthS1HAxis1 + 180];

In[65]:= angleS2HAxis = $\left(\text{ArcCos}[\text{unitNorthVector.s2UV}]\left(\frac{180}{\pi}\right)\right)$ + centroidGridConvergence;

In[66]:= azimuthS2HAxis1 = If[s2UV[[1]] < 0, 360 - angleS2HAxis, angleS2HAxis];

In[67]:= azimuthS2HAxis2 =
  If[azimuthS2HAxis1 > 180, azimuthS2HAxis1 - 180, azimuthS2HAxis1 + 180];

Step 13: Determine the magnitude of the maximum infinitesimal shear strain (γ_{max}) at 45° to the maximum principal strain axis

In[68]:= maxShearStrain = $2\sqrt{\left(\left(\frac{1}{2}\left(\text{m6}[[1,1]] - \text{m6}[[2,2]]\right)\right)^2 + \left(\text{m6}[[1,2]]\right)^2\right)}$;

Step 14: Determine the areal strain between the GPS sites

In[69]:= circleArea = π;

In[70]:= s1 = 1 + e1;

In[71]:= s2 = 1 + e2;

In[72]:= ellipseArea = $\pi \times (\text{s1}) \times (\text{s2})$;

In[73]:= areaStrain = $\frac{\text{ellipseArea} - \text{circleArea}}{\text{circleArea}}$;

In[74]:= m6[[1, 1]] + m6[[2, 2]];

Step 15: Identify/compute the second and third invariants of the strain tensor

In[75]:= firstInvariant = $(\text{m6}[[1,1]] + \text{m6}[[2,2]])$;

In[76]:= secondInvariant = $((\text{m6}[[1,1]])\ (\text{m6}[[2,2]])) - (\text{m6}[[1,2]])^2$;

In[77]:= thirdInvariant = Det[m6];

Step 16: Compute the uncertainties

In[78]:=
```
m7 =
```

$$\begin{pmatrix} \frac{1}{\text{inData}[[1,6]]^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\text{inData}[[1,7]]^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\text{inData}[[2,6]]^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\text{inData}[[2,7]]^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\text{inData}[[3,6]]^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\text{inData}[[3,7]]^2} \end{pmatrix};$$

In[79]:=
```
m8 = Transpose[m2];
```

We use a Moore - Penrose matrix inverse (PseudoInverse) to compute M9, because the matrix resulting from m8.m7.m2 can be a poorly conditioned matrix.

In[80]:=
```
m9 = PseudoInverse[m8.m7.m2];
```

In[81]:=
```
uncertTransE = √m9[[1, 1]] ;
```

In[82]:=
```
uncertTransN = √m9[[2, 2]] ;
```

In[83]:=
```
uncertRot = √m9[[3, 3]] ;
```

In[84]:=
```
uncertExx = √m9[[4, 4]] ;
```

In[85]:=
```
uncertExy = √m9[[5, 5]] ;
```

In[86]:=
```
uncertEyy = √m9[[6, 6]] ;
```

In[87]:=
```
summaryPlot = StrainEllipse[e1 * 10^9, e2 * 10^9, azimuthS1HAxis1, azimuthS2HAxis1];
```

Output

Distances between PBO sites before displacement

Distance between the first site and the second site (km)

In[88]:=
```
{{site1, site2}, sideLength12km}
```

Out[88]=
```
{{P574, EWPP}, 22.5701}
```

Distance between the second site and the third site (km)

In[89]:=
```
{{site2, site3}, sideLength23km}
```

Out[89]=
```
{{EWPP, P577}, 29.2641}
```