GPS-strain-calculator-InputCSV.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. Note that this version uses the formula employed by Corné Kreemer to compute the second invariant of the strain-rate tensor (e.g., Kreemer et al., 2014).

This version is coded in Mathematica 13.1, and was revised 22 August 2022 (error in M4 corrected) .

About the input data set

GPS site location and velocity data can be obtained from the Nevada Geodetic Laboratory at the University of Nevada-Reno (http://geodesy.unr.edu/NGLStationPages/gpsnetmap/GPSNetMap_-MAG.html). Resources include geodetic data from the MAGNET and NOTA networks, as well as other networks worldwide. As of late 2021, ***this is the current-best resource for obtaining high-quality GPS data worldwide.***

(Data from the Network of the Americas -- the old Plate Boundary Observatory GPS network and a Caribbean network operated by UNAVCO -- are available online through the good work of UNAVCO via
http://www.unavco.org/instrumentation/networks/status/pbo/gps)

This version of the GPS strain calculator was written to analyze velocity data derived from a comma-separated-value (CSV) file. The input dataset can be created in a text editor, saved as a text file, then substitute ".csv" for ".txt" to make it recognizable as a CSV file.

Each line (record) of the CSV input data file for this strain calculator has 10 items, corresponding to 10 columns in a flat file or matrix:

column 1:	record number in the fullDataSet
column 2:	4-letter text code for station
column 3:	longitude -- +ve is east, -ve is west
column 4:	latitude -- +ve is north, -ve is south
column 5:	reference frame for GPS site velocity data
column 6:	east velocity in mm/yr -- +ve is toward east, -ve is toward west
column 7:	north velocity in mm/yr -- +ve is toward north, -ve is toward south
column 8:	uncertainty of east velocity in mm/yr
column 9:	uncertainty of north velocity in mm/yr

column 10: data source

The first line (record) of the CSV file contains text column headings. For example,

RecNo,Station,Long,Lat,RefFrame,E-vel,N-vel,E-uncert,N-uncert,Source

Each line (record) that follows the header line is a data line containing a string of summary data for a given GNSS/GPS station. These data lines are numbered sequentially, beginning with 01. For example,

```
01,ABOO,37.809,8.992,AF,-0.5,0.55,0.27,0.27,http://geodesy.unr.edu/NGLStationPages/stations/ABOO.sta
02,ANKO,39.742,9.583,AF,2.11,2.41,1.77,1.35,http://geodesy.unr.edu/NGLStationPages/stations/ANKO.sta
03,ASAB,42.654,13.063,AF,15.59,11.98,0.24,0.23,http://geodesy.unr.edu/NGLStationPages/stations/AASAB.sta
04,BDMT,37.36,11.6,AF,-1.43,0.6,0.2,0.23,http://geodesy.unr.edu/NGLStationPages/stations/BDMT.sta
05,DA25,40.368,12.337,AF,-73.57,-14.34,2.79,0.8,http://geodesy.unr.edu/NGLStationPages/stations/DA25.sta
```

Dataset input

To modify this code so that it can be used for your dataset, do the following.

1. Enter the following as an input line (in the blue box, below):

```
rawInputDS = Import[];
```

2. Put your cursor between the square brackets in the `Import[]` statement and click to establish the insertion point we will need in the next step.

3. Go to the **Insert** menu, select **File Path**, navigate to the correct input data file and choose it, and the correct file path will be inserted at the cursor. In this example, the CSV data file is located on the desktop of Vince’s computer, and so *Mathematica* will insert the path “`/Users/vince/Desktop/TruckeeAreaGPS-velocities-20220821.csv`" between the brackets. The path will be different for every different file, and on every different computer.

```
rawInputDS=Import[“/Users/vince/Desktop/TruckeeAreaGPS-velocities-20220821.csv”];
```

You can make sure it is an input line by clicking on that line’s bracket on the far right edge of this window, going to the **`Format`** menu, selecting the **`Style`** submenu, and then choosing **`Input`**

In[1]:=
```
rawInputDS =
  Import["/Users/vince/Desktop/TruckeeAreaGPS-velocities-20220821.csv"];
```

This won' t work unless you follow directions and add the appropriate input line above this line. Your input line should look something like the example:
rawInputDS=Import["/Users/vince/Desktop/TruckeeAreaGPS-velocities-20220821.csv"];

Select three GPS sites for analysis

Dataset column headers

In[2]:=
```
{rawInputDS⟦1, 1⟧, rawInputDS⟦1, 2⟧, rawInputDS⟦1, 3⟧,
 rawInputDS⟦1, 4⟧, rawInputDS⟦1, 5⟧, rawInputDS⟦1, 6⟧,
 rawInputDS⟦1, 7⟧, rawInputDS⟦1, 8⟧, rawInputDS⟦1, 9⟧}
```

Out[2]=
```
{RecNo, Station, Long, Lat, RefFrame, E-vel, N-vel, E-uncert, N-uncert}
```

Dataset without headers

In[3]:=
```
fullDataSet = Table[{rawInputDS⟦i, 1⟧, rawInputDS⟦i, 2⟧, rawInputDS⟦i, 3⟧,
    rawInputDS⟦i, 4⟧, rawInputDS⟦i, 5⟧, rawInputDS⟦i, 6⟧, rawInputDS⟦i, 7⟧,
    rawInputDS⟦i, 8⟧, rawInputDS⟦i, 9⟧}, {i, 2, Length[rawInputDS]}];
MatrixForm[fullDataSet]
```

Out[3]//MatrixForm=
```
1  P090   -119.8   39.5728  NAM14  -6.93  5.44  0.03  0.02
2  P139  -119.722  39.9082  NAM14  -6.35  4.79  0.02  0.03
3  P144  -120.893  39.4667  NAM14  -9.26  7.39  0.05  0.04
4  P146  -120.537  39.3375  NAM14  -9.03  7.42  0.06  0.07
5  P147  -120.284  39.9374  NAM14  -7.41  5.95  0.03  0.06
6  P149  -120.105  39.6021  NAM14  -7.75   6.1  0.06  0.04
7  P150  -120.034  39.2924  NAM14  -8.49   6.8  0.06  0.03
8  P346  -120.867  39.7947  NAM14   -9.   7.22  0.08  0.06
```

Select the record numbers of the three GPS sites to be used in this analysis . For example, the definition of inData for an analysis run that involves velocities measured at GPS sites P146, P149, and P150 would be
inData={fullDataSet[[4]],fullDataSet[[6]],fullDataSet[[7]]};
because DEBK is record 19, SHIS is record 29, and DAKE is record 12 in the fullDataSet.

In[4]:=
```
inData = {fullDataSet⟦4⟧, fullDataSet⟦6⟧, fullDataSet⟦7⟧};
```

This won' t work unless you follow directions and add the appropriate record number in each of the three elements of the inData list. Your input line should look something like the exam-

ple:
```
inData={fullDataSet[[4]],fullDataSet[[6]],fullDataSet[[7]]};
```

```
In[5]:= MatrixForm[inData]

Out[5]//MatrixForm=
 4 P146 -120.537 39.3375 NAM14 -9.03 7.42 0.06 0.07
 6 P149 -120.105 39.6021 NAM14 -7.75  6.1  0.06 0.04
 7 P150 -120.034 39.2924 NAM14 -8.49  6.8  0.06 0.03

In[6]:= siteCode1 = inData⟦1, 2⟧; siteCode2 = inData⟦2, 2⟧; siteCode3 = inData⟦3, 2⟧;

In[7]:= refFrame = inData⟦1, 5⟧;
```

User - defined functions

The user-defined module **geog2UTMWGS84[]** converts from geographic coordinates (latitude, longitude) to UTM coordinates. The input arguments for the module **geog2UTMWGS84[]** are simply the decimal latitude and decimal longitude of a point. North latitude and east longitude are positive values; south latitude and west longitude are negative values. The output is a list with the following components: (1) a list with the UTM easting, UTM northing, and UTM zone of the point, and (2) a list with the pseudo-easting, pseudo-northing, and adjacent UTM zone to the west in which the pseudo coordinates are determined. This is useful when a set of location data contain UTM coordinates from two UTM zones, in which case all of the coordinates are expressed relative to the more westerly zone.

```
In[8]:= geog2UTMWSG84[latitude_, longitude_] :=
          Module[{latDeg, latRad, longDeg, longRad, rawZone, centMeridDeg,
             centMeridRad, centMeridPseudoDeg, centMeridPseudoRad, rawPseudoZone,
             a, flattening, b, kOught, e, ePrimeSquared, n, rho, nu, p, pseudoP,
             m1, m2, m3, m4, m, k1, k2, k3, k4, k5, answer, trueEasting,
             trueNorthing, trueZone, pseudoEasting, pseudoNorthing, pseudoZone},
            (* latRad is the latitude expressed in radians *)
            latRad = latitude * (π / 180);
            (* longRad is the longitude expressed in radians *)
            longRad = longitude * (π / 180);
            (* Computations to find the UTM zone and  *)rawZone = (longitude + 180) / 6;
            trueZone = If[((rawZone - IntegerPart[rawZone]) > 0),
               (IntegerPart[rawZone] + 1), IntegerPart[rawZone]];
            centMeridDeg = (-183 + (6 * trueZone));
            centMeridRad = centMeridDeg (π / 180);
            centMeridPseudoDeg = If[(centMeridDeg == -177), (177), (centMeridDeg - 6)];
            centMeridPseudoRad = centMeridPseudoDeg (π / 180);
            rawPseudoZone = (centMeridPseudoDeg + 180) / 6;
            pseudoZone = If[((rawPseudoZone - IntegerPart[rawPseudoZone]) > 0),
              (IntegerPart[rawPseudoZone] + 1), IntegerPart[rawPseudoZone]];
            (* Constants for WGS84 datum *)a = 6 378 137;
            flattening = 1 / 298.257223560;
            b = a - (a * flattening);
            kOught = 0.9996;
            (* Derived parameters *)e = Sqrt[1 - b^2 / a^2];
            ePrimeSquared = ((e * a) / b)^2;
            n = (a - b) / (a + b);
            rho = (a * (1 - (e^2))) / ((1 - (e^2 * Sin[latRad]^2))^1.5);
            nu = a / Sqrt[1 - (e^2 * Sin[latRad]^2)];
            p = longRad - centMeridRad;
            pseudoP = If[(centMeridPseudoDeg == 177),
              (Abs[longRad] - centMeridPseudoRad), (longRad - centMeridPseudoRad)];
            m1 = latRad * (1 - (e^2 / 4) - ((3 * e^4) / 64) - ((5 * e^6) / 256));
            m2 = Sin[2 * latRad] * (((3 * e^2) / 8) + ((3 * e^4) / 32) + ((45 * e^6) / 1024));
            m3 = Sin[4 * latRad] * (((15 * e^4) / 256) + ((45 * e^6) / 1024));
            m4 = Sin[6 * latRad] * ((35 * e^6) / 3072);
            m = a * (m1 - m2 + m3 - m4);
            k1 = m * kOught;
            k2 = kOught * nu * Sin[2 * latRad] / 4;
            k3 = (kOught * nu * Sin[latRad] * ((Cos[latRad])^3) / 24) *
              (5 - ((Tan[latRad])^2) + (9 * ePrimeSquared * ((Cos[latRad])^2)) +
                (4 * (ePrimeSquared^2) * ((Cos[latRad])^4)));
            k4 = kOught * nu * Cos[latRad];
            k5 = (kOught * nu * (Cos[latRad]^3) / 6) *
              (1 - (Tan[latRad]^2) + (ePrimeSquared * Cos[latRad]^2));
            (* Results *)trueNorthing = k1 + (k2 * p^2) + (k3 * p^4);
            trueEasting = 500 000 + (k4 * p) + (k5 * p^3);
            trueNorthing = k1 + (k2 * p^2) + (k3 * p^4);
            pseudoNorthing = k1 + (k2 * pseudoP^2) + (k3 * pseudoP^4);
            pseudoEasting = 500 000 + (k4 * pseudoP) + (k5 * pseudoP^3);
            (* Collect the results for output *)answer = {trueEasting,
              trueNorthing, trueZone, pseudoEasting, pseudoNorthing, pseudoZone};
            (* Output *)answer];
```

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[9]:= triGroupUTM[site1_, site2_, site3_] := Module[{d83, utmZones, e83, c83,
     c87, d87, e87, c88, d88, e88, finalEastings, finalNorthings, answer}
     (* Revised 20211208.  Input data are a list with UTM northing,
    easting, pseudonorthing, pseudoeasting,
    true zone number*), 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⟦1⟧, site1⟦3⟧];
    d87 = If[site2⟦5⟧ == c83, site2⟦1⟧, site2⟦3⟧];
    e87 = If[site3⟦5⟧ == c83, site3⟦1⟧, site3⟦3⟧];
    c88 = If[site1⟦5⟧ == c83, site1⟦2⟧, site1⟦4⟧];
    d88 = If[site2⟦5⟧ == c83, site2⟦2⟧, site2⟦4⟧];
    e88 = If[site3⟦5⟧ == c83, site3⟦2⟧, site3⟦4⟧];
    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[10]:= 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 (10^9 x 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 (10^9 x 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[11]:= 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

The array `temp1` is a list with the following components: (1) a list with the UTM easting, UTM northing, and UTM zone of the point, and (2) a list with the pseudo-easting, pseudo-northing, and adjacent UTM zone to the west in which the pseudo coordinates are determined

```
In[12]:= temp1 = geog2UTMWSG84[inData⟦1, 4⟧, inData⟦1, 3⟧];
```

The array `site1UTM` is a list with the following components: UTM easting, UTM northing, pseudo-easting, pseudo-northing, and UTM zone of the point.

```
In[13]:= site1UTM = {temp1⟦1⟧, temp1⟦2⟧, temp1⟦4⟧, temp1⟦5⟧, temp1⟦3⟧};

In[14]:= temp2 = geog2UTMWSG84[inData⟦2, 4⟧, inData⟦2, 3⟧];

In[15]:= site2UTM = {temp2⟦1⟧, temp2⟦2⟧, temp2⟦4⟧, temp2⟦5⟧, temp2⟦3⟧};

In[16]:= temp3 = geog2UTMWSG84[inData⟦3, 4⟧, inData⟦3, 3⟧];

In[17]:= site3UTM = {temp3⟦1⟧, temp3⟦2⟧, temp3⟦4⟧, temp3⟦5⟧, temp3⟦3⟧};

In[18]:= finalUTMcoord = triGroupUTM[site1UTM, site2UTM, site3UTM];

In[19]:= eastings = {finalUTMcoord⟦1, 1⟧, finalUTMcoord⟦1, 2⟧, finalUTMcoord⟦1, 3⟧};

In[20]:= northings = {finalUTMcoord⟦2, 1⟧, finalUTMcoord⟦2, 2⟧, finalUTMcoord⟦2, 3⟧};
```

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

```
In[21]:= meanX = 1/3 (eastings⟦1⟧ + eastings⟦2⟧ + eastings⟦3⟧);

In[22]:= meanY = 1/3 (northings⟦1⟧ + northings⟦2⟧ + northings⟦3⟧);

In[23]:= meanLat = (inData⟦1, 3⟧ + inData⟦2, 3⟧ + inData⟦3, 3⟧) / 3;

In[24]:= meanLong = (inData⟦1, 2⟧ + inData⟦2, 2⟧ + inData⟦3, 2⟧) / 3;
```

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

```
In[25]:=  m1 = ( eastings[[1]] - meanX  northings[[1]] - meanY )
               ( eastings[[2]] - meanX  northings[[2]] - meanY )  (* checked 20220821 *);
               ( eastings[[3]] - meanX  northings[[3]] - meanY )

In[26]:=  MatrixForm[%];
```

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[27]:=  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      )  (* checked 20220821 *);
               ( 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]] )

In[28]:=  MatrixForm[%];
```

Step 6: Invert matrix m2 to form matrix m3

```
In[29]:=  m3 = Inverse[m2] (* checked 20220821 *);

In[30]:=  MatrixForm[%];
```

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

Multiplying each input value by 0.001 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[31]:=  m4 = ( inData[[1, 6]] * 0.001 )
               ( inData[[1, 7]] * 0.001 )
               ( inData[[2, 6]] * 0.001 )  (* corrected 20220821 *);
               ( inData[[2, 7]] * 0.001 )
               ( inData[[3, 6]] * 0.001 )
               ( inData[[3, 7]] * 0.001 )

In[32]:=  MatrixForm[%];
```

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

```
In[33]:=  m5 = m3.m4 (* checked 20211221 *);
```

```
In[34]:= MatrixForm[%];
```

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

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

```
In[35]:= translationVector = {m5[[1, 1]], m5[[2, 1]]}(* checked 20211221 *);

In[36]:= out01 = N[m5[[1, 1]]];

In[37]:= out03 = N[m5[[2, 1]]];
```

The **translationSpeed** (below) is expressed in meters per year, relative to the reference frame in which the input velocities were expressed. This can be considered either a speed (m/yr) or an average annual displacement (m).

In[38]:= translationSpeed = $\sqrt{m5[[1,1]]^2 + m5[[2,1]]^2}$ (* checked 20211221 *);

```
In[39]:= out06 = N[translationSpeed];
```

In[40]:= unitTransVect = $\left\{\frac{\text{translationVector}[[1]]}{\text{Norm}[\text{translationVector}]}, \frac{\text{translationVector}[[2]]}{\text{Norm}[\text{translationVector}]}\right\}$
 (* checked 20211221 *);

```
In[41]:= unitNorthVector = {0, 1};
```

In[42]:= angle = $\left(\text{ArcCos}[\text{unitTransVect.unitNorthVector}]\left(\frac{180}{\pi}\right)\right)$ (* checked 20211221 *);

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

```
In[43]:= azTranslationVect = If[(unitTransVect[[1]] < 0), 360 - angle, angle];

In[44]:= out05 = N[azTranslationVect];
```

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

```
In[45]:= angularVelocityRadYr = m5[[3, 1]](* checked 20211221 *);

In[46]:= out11 = If[(angularVelocityRadYr < 0), "clockwise",
            If[(angularVelocityRadYr > 0), "anti-clockwise", "no rotation"]];
```

In[47]:= angVelNanoRadYr = N[angularVelocityRadYr / 10^{-9}] (* checked 20211221 *);

```
In[48]:= out07 = N[angVelNanoRadYr];

In[49]:= angularVelocityDegYr = angularVelocityRadYr * (180 / π) (* checked 20211221 *);

In[50]:= out09 = N[angularVelocityDegYr];
```

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

$$\varepsilon_{ij} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} \\ \varepsilon_{21} & \varepsilon_{22} \end{pmatrix}$$

```
In[51]:= m6 = ( m5[[4, 1]] m5[[5, 1]] ) (* "strain" units; checked 20211221 *);
              ( m5[[5, 1]] m5[[6, 1]] )
```

The output values below are expressed in nano - strain

```
In[52]:= out20 = N[m5[[4, 1]] / 10^-9] (* "nano-strain" units; checked 20211221 *);

In[53]:= out22 = N[m5[[5, 1]] / 10^-9] (* "nano-strain" units; checked 20211221 *);

In[54]:= out24 = N[m5[[6, 1]] / 10^-9] (* "nano-strain" units; checked 20211221 *);
```

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

```
In[55]:= eValues = Eigenvalues[m6];

In[56]:= e1 = If[eValues[[1]] > eValues[[2]], eValues[[1]], eValues[[2]]]
           (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[57]:= out12 = N[e1 / 10^-9] (* "nano-strain" units; checked 20211221 *);

In[58]:= e2 = If[eValues[[1]] > eValues[[2]], eValues[[2]], eValues[[1]]]
           (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[59]:= out15 = N[e2 / 10^-9] (* "nano-strain" units; checked 20211221 *);

In[60]:= eVectors = Eigenvectors[m6];

In[61]:= s1V = If[eValues[[1]] > eValues[[2]], eVectors[[1]], eVectors[[2]]];

In[62]:= s1UV = { s1V[[1]] / Norm[s1V] , s1V[[2]] / Norm[s1V] };

In[63]:= s1UVtest = Normalize[s1V];

In[64]:= s2V = If[eValues[[1]] > eValues[[2]], eVectors[[2]], eVectors[[1]]];
```

In[65]:= $\texttt{s2UV} = \left\{\frac{\texttt{s2V[[1]]}}{\texttt{Norm[s2V]}}, \frac{\texttt{s2V[[2]]}}{\texttt{Norm[s2V]}}\right\};$

```
In[66]:= s2UVtest = Normalize[s2V];
```

In[67]:= $\texttt{angleS1HAxis} = \left(\texttt{ArcCos[unitNorthVector.s1UV]}\left(\frac{180}{\pi}\right)\right);$

```
In[68]:= azimuthS1HAxis1 = If[(s1UV⟦1⟧ < 0), (360 - angleS1HAxis), angleS1HAxis];

In[69]:= azimuthS1HAxis2 =
           If[azimuthS1HAxis1 > 180, azimuthS1HAxis1 - 180, azimuthS1HAxis1 + 180];

In[70]:= out13 = N[azimuthS1HAxis1];

In[71]:= out14 = N[azimuthS1HAxis2];
```

In[72]:= $\texttt{angleS2HAxis} = \left(\texttt{ArcCos[unitNorthVector.s2UV]}\left(\frac{180}{\pi}\right)\right);$

```
In[73]:= azimuthS2HAxis1 = If[s2UV⟦1⟧ < 0, 360 - angleS2HAxis, angleS2HAxis];

In[74]:= azimuthS2HAxis2 =
           If[azimuthS2HAxis1 > 180, azimuthS2HAxis1 - 180, azimuthS2HAxis1 + 180];

In[75]:= out16 = N[azimuthS2HAxis1];

In[76]:= out17 = N[azimuthS2HAxis2];
```

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

In[77]:= $\texttt{maxShearStrain} = \texttt{N}\left[2\sqrt{\left(\left(\frac{1}{2}(\texttt{m6[[1, 1]]} - \texttt{m6[[2, 2]]})\right)^2 + (\texttt{m6[[1, 2]]})^2\right)}\right]$

```
  (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

In[78]:= $\texttt{out18} = \texttt{N}\left[\texttt{maxShearStrain} / 10^{-9}\right]$ `(* "nano-strain" units; checked 20211221 *);`

Step 14: Determine the areal strain between the GPS sites

```
In[79]:= circleArea = π;

In[80]:= s1 = 1 + e1;

In[81]:= s2 = 1 + e2;

In[82]:= ellipseArea = π × (s1) × (s2);
```

In[83]:= $\texttt{areaStrain2} = \frac{\texttt{ellipseArea - circleArea}}{\texttt{circleArea}};$

```
In[84]:= areaStrain1 = m6⟦1, 1⟧ + m6⟦2, 2⟧ (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[85]:= out19 = N[areaStrain1 / 10^-9] (* "nano-strain" units; checked 20211221 *);
```

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

The first invariant is the same as the area strain.

```
In[86]:= firstInvariant = (m6[[1, 1]] + m6[[2, 2]]) (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[87]:= out26 = N[firstInvariant / 10^-9] (* "nano-strain" units; checked 20211221 *);
```

The second invariant of the strain tensor is defined as $\sqrt{\varepsilon_1{}^2 + \varepsilon_2{}^2}$ where ε_1 and ε_2 are the largest and smallest (horizontal) principal strains, respectively (Kreemer et al., 2014).

The second invariant of the strain tensor is
$\sqrt{\varepsilon_1{}^2 + \varepsilon_2{}^2} = \sqrt{\varepsilon_{11}{}^2 + \varepsilon_{22}{}^2 + (2*(\varepsilon_{12}{}^2))}$.

```
In[88]:= secondInvariant = Sqrt[e1^2 + e2^2] (* "strain" units; checked 20211221 *);

In[89]:= secondInvariantTest = Sqrt[(m6[[1, 1]])^2 + (m6[[2, 2]])^2 + (2 * (m6[[1, 2]])^2)]
          (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[90]:= out27 = N[secondInvariant / 10^-9] (* "nano-strain" units; checked 20211221 *);

In[91]:= thirdInvariant = Det[m6]; (* "strain" units; checked 20211221 *)
```

The output value below is expressed in nano - strain

```
In[92]:= out28 = N[thirdInvariant / 10^-9] (* "nano-strain" units; checked 20211221 *);
```

Step 16: Compute the uncertainties

Matrix **m7** corresponds to the inverse of the data covariance matrix (cov $\mathbf{d}^{-1}$). Units are mm/yr.

In[93]:= $$\texttt{m7} = \begin{pmatrix} \frac{1}{\texttt{inData[[1,8]]}^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\texttt{inData[[1,9]]}^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\texttt{inData[[2,8]]}^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\texttt{inData[[2,9]]}^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\texttt{inData[[3,8]]}^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\texttt{inData[[3,9]]}^2} \end{pmatrix};$$

```
In[94]:= 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[95]:= m9 = PseudoInverse[m8.m7.m2];

In[96]:= uncertTransE = √m9⟦1, 1⟧ ;

In[97]:= out02 = N[ScientificForm[uncertTransE / 10^3]];

In[98]:= uncertTransN = √m9⟦2, 2⟧ ;

In[99]:= out04 = N[ScientificForm[uncertTransN / 10^3]];
```

uncertRotRad in radian per year

```
In[100]:=
uncertRotRad = √m9⟦3, 3⟧ ;
```

The output value below is expressed in nano - radian per year

```
In[101]:=
out08 = N[ScientificForm[uncertRotRad / 10^-9]];
```

uncertRotDeg in degrees per year

```
In[102]:=
uncertRotDeg = √m9⟦3, 3⟧ * (180 / π);

In[103]:=
out10 = N[ScientificForm[uncertRotDeg]];

In[104]:=
uncertExx = √m9⟦4, 4⟧ (* "strain" units; checked 20211221 *);

In[105]:=
uncertExy = √m9⟦5, 5⟧ (* "strain" units; checked 20211221 *);

In[106]:=
uncertEyy = √m9⟦6, 6⟧ (* "strain" units; checked 20211221 *);
```

The output value below is expressed in nano - strain

```
In[107]:=
out21 = N[uncertExx / 10^-9](* "nano-strain" units; checked 20211221 *);

In[108]:=
out23 = N[uncertExy / 10^-9](* "nano-strain" units; checked 20211221 *);

In[109]:=
out25 = N[uncertEyy / 10^-9](* "nano-strain" units; checked 20211221 *);
```

Matrix of output data

In[110]:=

```
outMatrix =
  {{"Reference frame for GPS site veolocities:", refFrame, " ", " "},
   {"Input Data", "----------", "-----", "----------"},
   {"GPS site code", siteCode1, " ", " "},
   {"location (longitude, latitude)", inData⟦1, 3⟧, inData⟦1, 4⟧, " "},
   {"E velocity (mm/yr)", inData⟦1, 6⟧, "+/-", inData⟦1, 8⟧},
   {"N velocity (mm/yr)", inData⟦1, 7⟧, "+/-", inData⟦1, 9⟧},
   {"GPS site code", siteCode2, " ", " "},
   {"location (longitude, latitude)", inData⟦2, 3⟧, inData⟦2, 4⟧, " "},
   {"E velocity (mm/yr)", inData⟦2, 6⟧, "+/-", inData⟦2, 8⟧},
   {"N velocity (mm/yr)", inData⟦2, 7⟧, "+/-", inData⟦2, 9⟧},
   {"GPS site code", siteCode3, " ", " "},
   {"location (longitude, latitude)", inData⟦3, 3⟧, inData⟦3, 4⟧, " "},
   {"E velocity (mm/yr)", inData⟦3, 6⟧, "+/-", inData⟦3, 8⟧},
   {"N velocity (mm/yr)", inData⟦3, 7⟧, "+/-", inData⟦3, 9⟧},
   {"Translation Data", "----------", "-----", "----------"},
   {"E component translation vector (m/yr)", out01, "+/-", out02},
   {"N component translation vector (m/yr)", out03, "+/-", out04},
   {"Translation vector azimuth (degrees)", out05, " ", " "},
   {"Translation vector speed (m/yr)", out06, " ", " "},
   {"Rotation Data", "----------", "-----", "----------"},
   {"Rotation angular velocity (degrees/yr)", out09, "+/-", out10},
   {"Rotation angular velocity (nano-radian/yr)", out07, "+/-", out08},
   {"Rotation direction", out11, " ", " "},
   {"Strain Ellipse Data", "----------", "-----", "----------"},
   {"Max horiz extension e1H (nano-strain/yr)", out12, " ", " "},
   {"Azimuth of max extension axis S1H", out13, "or", out14},
   {"Min horiz extension e2H (nano-strain/yr)", out15, " ", " "},
   {"Azimuth of min extension axis S2H", out16, "or", out17},
   {"Shear Strain & Area Strain", "----------", "-----", "----------"},
   {"Maximum shear strain (nano-strain/yr}", out18, " ", " "},
   {"Area strain (nano-strain/yr)", out19, " ", " "},
   {"Lagrangian Strain-Rate Tensor", "----------", "-----", "----------"},
   {"εxx (nano-strain/yr)", out20, "+/-", out21},
   {"εxy (nano-strain/yr)", out22, "+/-", out23},
   {"εyy (nano-strain/yr)", out24, "+/-", out25},
   {"1st invariant strain-rate tensor (nano-strain/yr)", out26, " ", " "},
   {"2nd invariant strain-rate tensor (nano-strain/yr)", out27, " ", " "},
   {"3rd invariant strain-rate tensor (nano-strain/yr)", out28, " ", " "}};
```

In[111]:=

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

Output

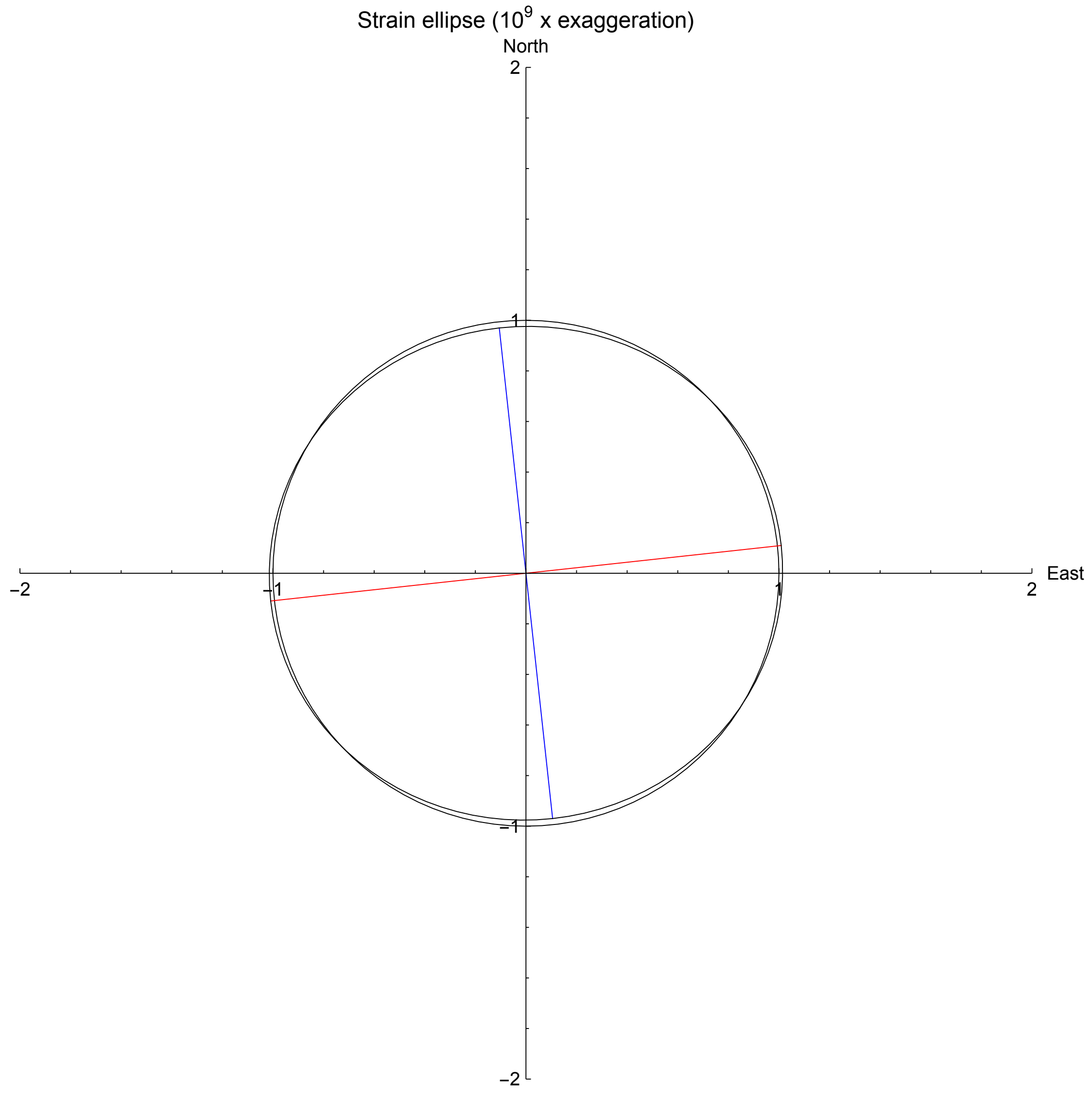
Plot of strain ellipse (exaggerated by 1 billion times)

Plot of the infinitesimal horizontal strain ellipse, exaggerated by 1 billion times. The S1h axis is the red line, and S2h is blue.

In[112]:=

```
summaryPlot
```

Out[112]=



Output Matrix

In[113]:=

```
MatrixForm[outMatrix]
```

Out[113]//MatrixForm=

```
      Reference frame for GPS site veolocities:          NAM14
                     Input Data                        ----------    -----   -
                   GPS site code                          P146
          location (longitude, latitude)                -120.537    39.3375
              E velocity (mm/yr)                          -9.03       +/-
              N velocity (mm/yr)                          7.42        +/-
                   GPS site code                          P149
          location (longitude, latitude)                -120.105    39.6021
              E velocity (mm/yr)                          -7.75       +/-
              N velocity (mm/yr)                           6.1        +/-
                   GPS site code                          P150
          location (longitude, latitude)                -120.034    39.2924
              E velocity (mm/yr)                          -8.49       +/-
              N velocity (mm/yr)                           6.8        +/-
                 Translation Data                      ----------    -----   -
    E component translation vector (m/yr)             -0.00842333     +/-    3.
    N component translation vector (m/yr)              0.00677333     +/-    2.
     Translation vector azimuth (degrees)               308.803
       Translation vector speed (m/yr)                 0.0108088
                   Rotation Data                       ----------    -----   -
     Rotation angular velocity (degrees/yr)          -1.17404×10^-6   +/-    8.
  Rotation angular velocity (nano-radian/yr)           -20.4908       +/-    1
                Rotation direction                     clockwise
                 Strain Ellipse Data                   ----------    -----   -
    Max horiz extension e1H (nano-strain/yr)            14.9555
      Azimuth of max extension axis S1H                 263.797        or
    Min horiz extension e2H (nano-strain/yr)           -24.3803
      Azimuth of min extension axis S2H                 353.797        or
             Shear Strain & Area Strain                ----------    -----   -
       Maximum shear strain (nano-strain/yr}            39.3357
           Area strain (nano-strain/yr)                -9.42482
            Lagrangian Strain-Rate Tensor              ----------    -----   -
                 εxx (nano-strain/yr)                   14.4962       +/-
                 εxy (nano-strain/yr)                   4.22567       +/-
                 εyy (nano-strain/yr)                   -23.921       +/-
 1st invariant strain-rate tensor (nano-strain/yr)     -9.42482
 2nd invariant strain-rate tensor (nano-strain/yr)      28.6018
 3rd invariant strain-rate tensor (nano-strain/yr)   -3.64618×10^-7
```

References

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.

Cronin, V.S., and Resor, P.G., 2021, Primer on infinitesimal strain analysis in 1, 2 and 3-D : accessible via https://croninprojects.org/Vince/Geodesy/GPS_Strain_Primer-20210613.pdf

Cronin, V.S., and Resor, P.G., 2021, Algorithm for triangle-strain analysis: accessible via https://croninprojects.org/Vince/Geodesy/TriangleStrainAlgorithm-20211220.docx

Kreemer, C., Blewitt, G., and Klein, E.C., 2014, A geodetic plate motion and Global Strain Rate Model: Geochemistry, Geophysics, Geosystems, v. 15, p. 3849-3889, doi:10.1002/2014GC005407.

Means, W.D., 1976, Stress and strain - Basic concepts of continuum mechanics for geologists : Englewood Cliffs, New Jersey, Prentice - Hall, Inc., 339 p.

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, P21, 995-22, 007.