

kmaterna / Strain\_2D

<> Code 5 Pull requests Actions Projects Security Insights

Strain rate modeling from GNSS velocity fields

MIT license

40 stars 16 forks 3 watching 1 Branch 3 Tags Activity


Public repository

master 1 Branch 3 Tags

Go to file t

Go to file + Add file

<> Code ...

	small change in plumbing outdir for moment calculation	5 days ago
Strain_Tools	small change in plumbing outdir for moment cal...	5 days ago
example	minor touch-ups to readme, example, and requir...	last year
test	removing trailing semicolons	5 days ago
.gitignore	revamping example map script for new netcdf fo...	3 years ago
_init_.py	adding documentation and plots of mean quanti...	3 years ago
citation.cff	adding executable to calculate moment (Savage ...	3 years ago
code.json	updating plots, readme, and fixing typos in exam...	2 years ago
contributing.md	minor touch-ups to readme, example, and requir...	last year
disclaimer.md	adding code.json and pre-release disclaimer	2 years ago
license.md	Revert "removing old license.md"	2 years ago
param_definitions.md	adding visr ability to handle up to 10 creeping fa...	2 years ago
readme.md	adding function to write moment-rate map into ...	last week
requirements.yml	minor touch-ups to readme, example, and requir...	last year
setup.py	adding executable to calculate moment (Savage ...	3 years ago
version.txt	updating plots, readme, and fixing typos in exam...	2 years ago

## 2D Strain Rate Calculators

python 3+ License MIT DOI 10.5281/zenodo.7328416

This library contains several methods to compute geodetic strain from a GPS velocity field. It is intended to be used as a means to compare various strain modeling techniques on the same input data (see [https://files.scec.org/s3fs-public/0930\\_Sandwell\\_UCERF\\_strain.pdf](https://files.scec.org/s3fs-public/0930_Sandwell_UCERF_strain.pdf) for an example showing the variability of strain modeling techniques). Several of the strain methods are borrowed from other authors, GitHub repositories, and papers. If you use the contributed strain techniques, credit should be given to the original authors accordingly.

### Requirements and Installation:

- Python3, numpy, scipy, matplotlib, pygmt

- GMT5 or higher: <https://www.generic-mapping-tools.org/>
- Third-party Matlab or Fortran codes may be required depending on the specific strain technique you select, as detailed further below
- Optional: Jupyter notebook is needed to run the tutorial

First, clone the Strain\_2D repository onto your computer and get into the top level directory.

To handle dependencies, it is easiest to create a conda environment for this software, although that step is not strictly required if you have

 [README](#)  [MIT license](#)  

```
conda env create -f requirements.yml
conda activate Strain_2D
```



Then, to install this library within the newly created conda environment, run in the top level directory of the repo:

```
pip install .
```



We have not yet tested the installation on Windows. Let us know if you have used the software on Windows!

## Usage:

The main executable for this library is `strain_rate_compute.py`. The behavior of the program is controlled using a config file that specifies inputs/outputs, general strain options, and any required parameters for various strain rate techniques. You can print a sample config file from the main executable `strain_rate_compute.py` under the flag `--print_config` (also try `--help` for more information).

An example run-string would be: `strain_rate_compute.py config.txt`

Input velocities must be in a text file. They must be a space-separated table with format as follows:

```
# lon(deg) lat(deg) VE(mm) VN(mm) VU(mm) SE(mm) SN(mm) SU(mm) name(optional)
```



In general, users are expected to clean their velocities before using Strain\_2D, according to their own needs in their own projects. Within Strain\_2D, the velocities will be filtered to include only those within the bounding box provided in the config parameter `range_data`.

## Outputs:

Output strain components and derived quantities (invariants, eigenvectors) are written as *pixel-node-registered* netCDF files and/or text files, and plotted in PyGMT.

To see what's inside the large netCDF file, you can call:

```
#!/bin/bash
infile="gpsgridded_strain.nc" # any output .nc file
ncdump -h $infile
```



To convert one layer of the netCDF file (such as dilatation) into a valid pixel-node-registered grdf file, you can extract with GDAL if you have it.

```
#!/bin/bash
outfile="dilatation.grd"
gmt_range="-121.01/-113.99/30.99/37.01" # technically half a pixel outside of each bound, for pixel-node-registration
gmt grdedit $infile=gd7HDF5:"$infile"://dilatation -R$gmt_range -T -G$outfile # send layer out pixel-node-registered grdf file,
gmt grdedit $outfile -Ev # top-bottom flip for latitude increasing vs column number increasing
```



## Contributing

If you're using this library and have suggestions, let me know! I'm happy to work together on the code and its applications. We are working on the contributing.md for guidelines.

See the section on API below if you'd like to contribute a method.

## Supported strain methods:

For more information on the parameters required by each method, see [Full Parameter Documentation](#).

1. **delaunay\_flat**: Delaunay Triangulation, the simplest method. The equations can be found in many papers, including Cai and Grafarend (2007), Journal of Geodynamics, 43, 214-238. No parameters are required to use this method.
2. **delaunay**: a generalization of the Delaunay Triangulation for a spherical earth. This Python implementation is based on Savage et al., JGR October 2001, p.22,005, courtesy of Bill Hammond's matlab implementation. No parameters are required to use this method.
3. **visr**: The "VISR" method is a fortran code for the interpolation scheme of Zheng-Kang Shen et al., Strain determination using spatially discrete geodetic data, Bull. Seismol. Soc. Am., 105(4), 2117-2127, doi: 10.1785/0120140247, 2015. <http://scec.ess.ucla.edu/~zshen/visr/visr.html>. You can download the source code, which must be compiled and linked on your own system, for example by : `gfortran -c voronoi_area_version.f90 / gfortran visr.f voronoi_area_version.o -o visr.exe`. Four additional config parameters are required to use this method.
4. **gpsgridded**: based on a thin-sheet elastic interpolation scheme from Sandwell, D. T., and P. Wessel (2016), Interpolation of 2-D vector data using constraints from elasticity, GRL. The implementation of the code is in GMT. Three additional config parameters are required to use this method.
5. **loc\_avg\_grad**: the weighted nearest neighbor algorithm of Mong-Han Huang and implemented in Handwerger, A. L., Huang, M. H., Fielding, E. J., Booth, A. M., & Bürgmann, R. (2019). A shift from drought to extreme rainfall drives a stable landslide to catastrophic failure. Scientific reports, 9(1), 1-12. Two additional config parameters are required to use this method.
6. **wavelets**: a wavelet-based matlab program from Tape, Muse, Simons, Dong, Webb, "Multiscale estimation of GPS velocity fields," Geophysical Journal International, 2009 (<https://github.com/carltape/surfacevel2strain>). Needs a matlab installation and some manual run steps.
7. **geostats**: a method based on traditional geostatistical anlysis, using kriging to do the interpolation. This method requires the user to set one model type and three parameters. The model is the type of correlation structure the user expects to exist in the underlying field. A "Nugget" model is a white-noise model, "Exponential" models a continuous but non-differentiable process, and "Gaussian" is both continuous and differentiable. All models require a nugget, which represents the level of point-wise variance in the observations, which for GNSS velocities is the same as data uncertainty. The Gaussian and Exponential models also require a "sill" and "range" to be specified. The sill characterizes the overall mean variance in the dataset, and should be specified as the total variance of the observations minus the nugget. Note that the nugget can be thought of as the variance of the data uncertainties, and the sill is the variance of the data itself. The range is the (isotropic) spatial correlation length scale. Currently, we only implement an isotropic version, although anisotropic and even spatially-varying methods can be used.

## Not included methods:

If you have another strain method that you'd be willing to contribute, I would love to work with you to include it! More methods results in a more robust estimate of strain rate variability. I have not included the following techniques for the reasons given:

1. **Spline**: based on Python's built-in spline interpolation of the velocity field. I have found that it doesn't always produce reasonable results.
2. **NDInterp**: based on Numpy's linear interpolation of the velocity field. It turns out to be basically the same as Delaunay.

## Units and Sign Conventions

This library uses the following units and sign conventions:

- station velocities: units of mm/yr in the internal format
- strain rates (exx, exy, eyy):
  - $exx = d_{udx}$
  - $exy = 0.5 * (d_{vdx} + d_{udy})$  . Tensor shear strain rate (*not engineering shear strain rate*)
  - units: nanostrain / yr ( $1e-9$  / yr)
- Rotation:
  - $W = 0.5 * (d_{vdx} - d_{udy})$
  - units: ( $1e-3$  radians)/yr, or radians / Ka
- Second invariant:
  - $Log\ of\ I2 = 0.5 * (exx * eyy - exy * exy)$
  - units: (nanostrain/yr)<sup>2</sup> (very unusual units; usually plotted on log scale for clarity)
- Dilatation: Positive means extension and negative means shortening.

- Dilatation =  $\epsilon_{xx} + \epsilon_{yy}$
- units: nanostrain / year
- Max Shear:
  - $\epsilon_{\max} = 0.5 * \sqrt{(\epsilon_{xx} - \epsilon_{yy})^2 + 4 * \epsilon_{xy}^2}$  . Tensor shear strain rate (*not engineering shear strain rate*)
  - units: nanostrain / year

## Internal Library API

If you're interested in contributing your own type of strain computation, I am happy to add new methods to the library. You would need to build a Python 'compute' function that matches the API of the other compute functions in the repository (see `strain/models/strain_2d.py` for template).

```
class your_strain_method(strain_2d.Strain_2d):
    def __init__(self, Params):
        # perform various types of parameter-setting and input-verification,
        # different for each technique
        return

    def compute(self, myVelfield):
        [Ve, Vn, rot_grd, exx_grd, exy_grd, eyy_grd, velfield, residual_velfield] = your_strain_method_compute(myVelfield...
etc);
        return [Ve, Vn, rot_grd, exx_grd, exy_grd, eyy_grd, velfield, residual_velfield];
```

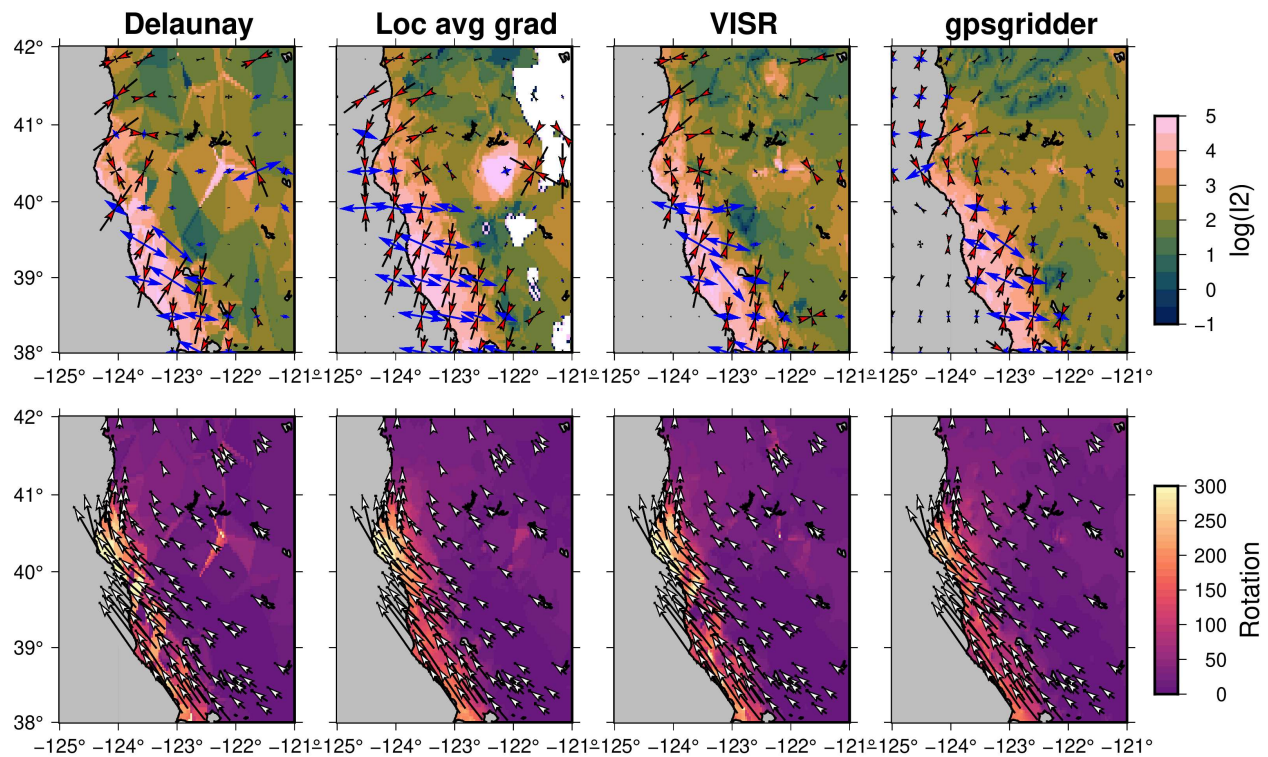
where:

- myVelfield is a 1D list of StationVel objects (very easy to use).
- lons is a 1D array of longitudes, in increasing order
- lats is a 1D array of latitudes, in increasing order
- Ve and Vn are 2D arrays of geodetic velocities, if the method computed interpolated velocities (not every method does this)
- rot\_grd, exx\_grd, exy\_grd, eyy\_grd are 2D arrays that correspond to lons/lats grids
  - exx/exy/eyy have units of nanostrain
  - rot has units of radians/Ka

## Example:

To reproduce the figure in the README:

1. In the example/ directory, run: `strain_rate_compute.py --print_config` to print an example config file (similar to the one provided here in the repo).
2. Run: `strain_rate_compute.py example_strain_config.txt` to compute delaunay strain rate using the parameters in the newly-created config file.
3. Change the method in the config file to try other methods, and re-run each time for different strain calculations with different parameters. Try with `gpsgridded`, `delaunay`, and `loc_avg_grad`. If you have the `visr` executable on your system, try with `visr` as well.
4. Run: `strain_rate_comparison.py example_strain_config.txt` to compute average strain rate maps from several results.
5. Navigate into `Display_output/`
6. Run: `./comparison_rows_example.sh -125/-121/38/42` to view a GMT plot with several strain rate calculations in Northern California together.



## Citing

### Releases 3





 **v1.1.1** Latest  
on Nov 17, 2022

[+ 2 releases](#)

### Packages

No packages published

### Contributors 4

-  kmaterna
-  jmaurer Jeremy Maurer
-  lucysandoe
-  yichiehlee Yi-Chieh Lee

### Languages

 Jupyter Notebook 92.8%  Python 7.0%  Shell 0.2%