bassi: Bayesian Analysis for Slow Slip Inversions

Francisco Romero

email: francisco.romero@cimat.mx

April 2023

1 Introduction

This packages provides tools for a Bayesian approach to infer fault displacements in a Slow Slip Event through data readings of GPS stations displacements. The model and analysis is based in the work of Montesinos-López et al. (2020), this manual provides a very basic and simplified explanation of the model and techniques utilized, we refer to the cited papers for more detailed explanations.

2 Model and Priors

The model consists of a multiple linear regression with constraints on the coefficient, namely

$$\mathbf{U}|\mathbf{D} \sim N_{3N}(\mathbf{X}\mathbf{D}, \Sigma)$$

where $\mathbf{U} \in \mathbb{R}^3$ are the displacements (North, East, Vertical) registered at N GPS stations expressed in a single vector. It's supposed that $\Sigma = \mathbf{I}_N \otimes \gamma$ where $\gamma = diag([\sigma_x^2, \sigma_y^2, \sigma_z^2])$ and the entries represent the standard deviations in the North, East, and Vertical direction, while $\mathbf{I}_N$ is the identity matrix of size $N \times N$ and $\otimes$ it's the Kronecker product.

We considere a GP prior distribution for the slips $\mathbf{D} = (d_s^1, d_d^1, d_s^2, d_d^2, ..., d_s^M, d_d^M)^T$, that is, $\mathbf{D} \sim N\left(0, \frac{1}{\sigma_\beta^2}\mathbf{A}_0\right)$ with truncated support $d_s^i \in (a_s, b_s)$ y $d_d^i \in (a_d, b_d)$ where $\frac{1}{\sigma_\beta^2}\mathbf{A}_0$ it's the precision matrix, and

$$A_0 = \beta \mathbf{W}\mathbf{C}^{-1}\mathbf{W}\beta$$

where $\mathbf{W}$ is the Weight Matrix, $\mathbf{C}$ is the Matérn Covariance Matrix and $\beta = \mathbf{I} \otimes \beta$ with $\beta = diag([\beta_s, \beta_d])$. We refer to Montesinos-López et al. (2020) for further explaining.

3 Design

The package was designed considering that the bayesian analysis consists of three big parts, the Fault/Subfault configuration, the Event registered given the Fault configuration that encompasses the data observed, the GPS station data, the Tractions Matrix; and the Analysis that results of applying the model to the event for some given parameters. Each part has its own object designed. This means that each *fault* object could apply to many *event* objects if there are multiple SSE in the region, and we could make different *analysis* objects should we change parameters such as standard deviations in the North, East, or Vertical Direction, or the correlation lengths, along with MCMC configuration parameters.

For the package to work, folders and data must be organized according in a specific way according to the previous discussion. There must be a Fault folder containing data and information to read related only to the fault configuration. Inside the Fault folder must be the Event folder containing all the information related solely to the Slow Slip Event. The package generates inside the Event folder various Analysis folder according to the configuration. For example:

```
.\Fault\Event\Analysis
```

For a guide on how to use the package, we recommend to read Chapter 7 and using Chapter 4,5, and 6 as a more detailed guide on the object classes provided by the package.

4 The *fault* class

The *fault* class encompasses the information about the Fault, and its discretization in subfaults such as the shape and size of the subfaults, its coordinates in some CRS along with depth/height, and the Weight Matrix. Subfault shape is supposed to be rectangular and all subfaults have the same size. Weight Matrix should be provided and located in fault folder.
The parameters are

- *name*:name of the fault, must coincide with folder name containing subsequent Fault data, and Event folders.

- *rot_ang*: rotation angle of Fault with respect to the East Direction.

- *shape*: shape of fault (number of rows, number of columns).

- *sfsize*: size of each subfault in Km (row width, column width).

The methods for this class are:

- **Sf_size_fromfil**(*filename="subfault_size.txt"*):

  Reads subfault size in Km from file. Used if not initialized or needs to be redefined.

  Parameters:

  filename: ***str***
  Name of file containing subfault sizes located in Fault folder.

  Returns:

  None

- **WeightMat**(*W_file="W_m.txt"*):

  Returns Weight Matrix for Fault Config as Numpy array, file must be located in fault folder or specified subfolder of fault folder.

  Parameters:

  W_file: ***str***
  name or subdirectory of Weight Matrix File.

  Returns:

  W: numpy.array

- **rows**():

  Returns number of rows of Subfault Configuration.

  Parameters:

  None

  Returns:

None

- **cols**():

  Returns number of columns of Subfault Configuration.

  Parameters:

  None

  Returns:

  None

- **sf_x_coords**(*filename="sfcoorx.txt"*):

  Returns Subfaults East coordinates as Pandas Dataframe. CRS not assigned.

  Parameters:

  filename: ***str***
  name of file containing x coordinates of some CRS.

  Returns:

  pandas.Dataframe

- **sf_y_coords**(*filename="sfcoory.txt"*):

  Returns Subfaults North coordinates as Pandas Dataframe. CRS not assigned.

  Parameters:

  filename: ***str***
  name of file containing y coordinates of some CRS.

  Returns:

pandas.Dataframe

- **sf_z_coords**(*filename="sfcoorz.txt"*):

  Returns Subfaults Vertical coordinates as Pandas Dataframe. CRS not assigned.

  Parameters:

  filename: ***str***
  name of file containing z coordinates of some CRS.

  Returns:

  pandas.Dataframe

5 The *event* class

The *event* object contains the information about the SSE, such as data observed, number of GPS stations, coordinates and depth. Each *event* object has in itself an *fault* object. Observed Data may come in different coordinate systems. For this analysis we may suppose that either it coordinate system is (North, East, Vertical) or along the Fault (general) Direction and Vertical, both cases in meters. This is important once we define the standard deviations for the prior since they must be rotated according to the data observed. Also, the traction Matrix T its assumed to be computed using the observed data and its same coordinate system. The *event* class allows to choose if xy coordiantes are NE directions or along Fault direction, and also if vertical coordinates are depth or height.

The parameters are

- *Fault*:Fault object describing the Fault configuration of the Event.

- *name*: filename containing station data such as names and coordinates. file must be inside Event folder.

- *stats_file*: filename containing station data such as names and coordinates. file must be inside Event folder. Stat data is read as a pandas.DataFrame.

- *U_file*: filename of observation data in Event Folder. If not specified can be later redefined which is useful in case of synthetic data.

- *NE_direction*: True if observations xy coordinates are North and East Directions, and False if observations are along fault direction.

- *Z_positive*: True if Z coordinates (depth) are considered positive, false if considered negative.

- *T_file*: name of file containing Traction Matrix in Event Folder.

The methods included in this class are

- **U_fromfile**(*filedir, NE_direction=False, Z_positive=False*):

  Redefines observations U matrix from filedir. Must be Specified if observations are along NE_direction and if Z observations are positive.

  Parameters:

  filedir: ***str***
  directory of file from which to read observations data.

  NE_direction: ***bool***
  True if data xy coordinates are in North and East directions. If false, data xy coordinates are assumed to be along Fault direction.

  Z_positive: ***bool***
  True if data z coordinate (depth) is considered positive. If false, data z coordinates is considered negative. **Returns:**

  None

- **U_rot**():

  Rotates (and redefines) data matrix along fault direction.

  Parameters:

None

Returns:

None

- **create_synthetic**(la1, la2, sig2_bet, mode1, Sn, Se, Sv, beta1=1., beta2=0.2, two_mode = False, mode2=None, W_file="W_m.txt"):

  Simulate synthetic data with given parameters.

  Parameters:

  la1 : float
  Correlation length for the strike component.

  la2 : float
  Correlation length for the dip component.

  sig2_bet : float
  variance of the prior distribution; sig_beta$\hat{2}$.

  mode1: int
  index of subfault which would contain the highest slip magnitude

  Sn: float
  standard deviation of displacements in North Direction

  Se: float
  standard deviation of displacements in East Direction

  Sv: float
  standard deviation of displacements in Vertical Direction

  beta1:float
  along strike precision for prior precision Matrix

  beta2:float
  along dip precision for prior precision Matrix

two_mode : bolean, optional
If two_mode = True simulate the synthetic data with two modes. The default is False.

W_file:str
filename of Weight Matrix. Must be contained in Fault Folder.

Returns:

None

6 The *analysis* class

The *analysis* object applies the model and MCMC algorithm given some parameter defined in the object along with the *fault* and *event* objects to simulate and obtain a Markov Chain that has the posterior distribution of the slips.

The parameters are

- *Event*:Event object describing the Event data.
- *folder_name*: name of the folder to save analysis data file must be inside Event folder.
- *Sn*: Standard deviation in north direction.
- *Se*: Standard deviation in east direction.
- *Sv*: Standard deviation in vertical direction.
- *s_supp:*: tuple containing slip along strike direction support.
- *d_supp*:tuple containing slip along dip direction support.
- *d_supp*:tuple containing slip along dip direction support.
- *synt_data*:True if data to analyze is synthetic, false if data to analyze is real.

- *beta1*:Precision for strike component, default is 1.

- *beta1*:Precision for dip component, default is 0.2 .

As for the methods included in the objects, only the ones intended for the user will be listed, the other methods shouldn't be used by the user.

- **Post**(*la1, la2, synt = True, la1_s = None, la2_s = None, sig2_betar = None*):

  Function to compute the posterior mean and posterior covariance matrix of the multivariate truncated normal distribution.

  Parameters:

  la1: ***float***
  Correlation length for the strike component.

  la2: ***float***
  Correlation length for the dip component.

  synt: ***Boolean***
  If synt = True uses the synthetic data.
  If synt = False uses the real data.

  la1_s: ***float***
  Correlation length for the strike component for the synthetic data.

  la2_s: ***float***
  Correlation length for the dip component for the synthetic data.

  sig2_betar: ***float***
  Variance in definition of prior Covariance Matrix
  Returns:

  Posterior mean and posterior covariance matrix.

- **DIC**(*data*):

Computes DIC (Deviance Information Criterion) of posterior simulated data.

Parameters:

data: ***numpy.array***
Posterior simulated data

Returns:

Deviance information criterion.

- **Run_MCMC**(*la1, la2, itera = 1, n = 15000000, ss = 2000, BurnIn = 60000, saveoutput = False, saveconfig=False*):

  Simulate from the truncated normal calculating the DIC.

  Parameters:

  la1 : float
  Correlation length for the strike component.

  la2 : float
  Correlation length for the dip component.

  itera : int
  Number of the iteration.

  n: int
  number of simulations: length of the chain without Burn-in

  ss: int
  Sample size; number of simulations to save.

  saveoutput : bolean
  If saveoutput = True, save the simulations else only return the simulations simu and the log energy logEnergy. The default is True.

  saveconfig : bolean
  If saveconfig = True, save the MCMC config in a file else the MCMC config is not saved. The default is True.

Returns:

Save la1, la2, and the DIC in the DIC.txt file.

- **DIC**(*la1,la2*):

  Computes Coefficient of Variation.

  Parameters:

  la1: ***float***
  Correlation length for the strike component.

  la2: ***float***
  Correlation length for the dip component.

  Returns:

  Saves Coefficient of Variaton and returns it as a *numpy array*.

7 How to use bassi?

We recommend using the file *bassi_example.py* while following this guide for a better understanding.

First, bassi must be installed. Donwload bassi from https://github.com/FranciscoRomeroCIMAT/bassi and install the whl. file in the dist folder with the command

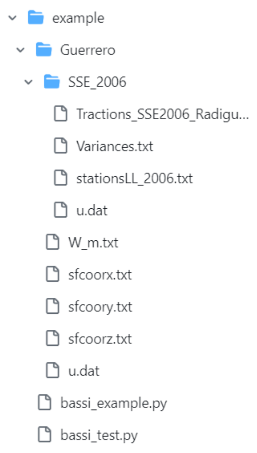
```
    pip install bassi-0.8-py3-none-any.whl
```

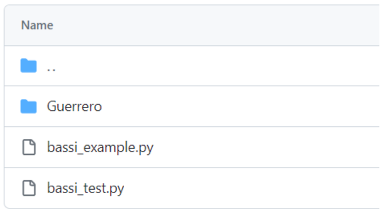
if already installed, run the command

```
    pip install --upgrade --no-deps --force-reinstall  bassi-0.8-
    py3-none-any.whl
```

Next, all data must be organized according to the previous discussion. Let's assume we have all files necessary for the use of bassi. For this example we will use *bassi_example.py* in the example folder. Data should be organized like in figure Create a new .py file located in the root directory containing the Fault data folder. In this example we create *bassi_example.py* in the directory containing *Guerrero* folder which should look something like this

The only file read from inside the Fault folder is the Weight Matrix. We can define the *fault* object with the data provided running the next code





```python
import bassi as ba
import os

path=os.getcwd()+"/"
GGap=ba.fault(name="Guerrero",
                   rot_ang=-21.1859,shape=(19,31),
                   sfsize=(20.563905505456397,20.002022023921285),
                   rootdir=path)
```

We have created a *fault* object named *GGap* with some specified parameters. *name* should be the same as the fault folder, in this case *"Guerrero"*. *rot_ang* is the rotation angle of the Fault with respect to the North-East Direction, *shape* is a tuple containing the number of (rows, columns) of the discretization of the fault. *sfsize* is a tuple containing (row width, column width) of each subfault, and we specified the root directory (that contains fault folder) with *rootdir*. The only file read from inside the fault folder is the Weight Matrix which by defaults assumed is called *"W_m.txt"*.

Next, we need to define the *event* object with the next code

```python
SSE_2006=ba.event(GGap, name="SSE_2006",
                           stats_file="stationsLL_2006.txt",
             U_file="u.dat", NE_direction=False, Z_positive=False,
    T_file="Tractions_SSE2006_Radiguet_x_subfaults_areas.in")
```

We have defined a *event* object called *SSE_2006*, note that the *GGap* object is a parameter. *name* must be set equal to the event folder that contains the data, *stats_file* refers to the file containing the stations data, *U_file* is the name of the file containing the data observed, *NE_direction* and *Z_positive* are used to indicate whether data observed is in North-East direction or along fault direction, and if the z coordinate indicates depth or height, and finally, *T_file* is the name of the file containing the traction matrix. In this step, the files that are read are *stats_file*, *U_file*, *T_file*.

Now, we must define an *analysis* class object that contains most of the parameters specification. First, we run the next command

```
Test_analysis=ba.analysis(SSE_2006, folder_name="Test_analysis",
                                Sn=0.0021, Se=0.0025, Sv=0.0051,
                                s_supp=(-0.1,0.1),
                                d_supp=(-0.0804,0.4),
                                beta1=1.,beta2=0.2 )
```

this will create an *analysis* object called *Test_analysis*. The parameters that are set along the object are the standard deviations $\sigma_x, \sigma_y, \sigma_z$ along North, East and vertical directions with Sn, Se and Sv, the precisions $beta_1$ and $beta_2$ along dip and strike components, and the support for the prior with *s_supp* and *d_supp*. Once we define the object, it will create a new folder inside the Event folder called $folder\_name$ which would contain all subsequent analysis made with the object.

Once all three objects are defined, we use only the *analysis* object to obtain simulations and results. We will first analyze the real data with the next command line

```
Test_analysis.Post(la1=35, la2=45, synt = False, la1_s = None,
la2_s = None, sig2_betar = None)

simu,logpost = Test_analysis.Run_MCMC(35, 45, itera=1,
                    n = 50000,
                    BurnIn = 6000,
                    ss = 5000,
                    saveoutput = True, saveconfig=True)
```

In this code, the method *Test_analysis.Post()* will create the prior covariance and precision matrix Σ_0 and A_0. These are not meant to be used by the user, but are necessary for the bayesian analysis, once the are created, there's no need to run *Test_analysis.Post* again. The correlation lenghts are specifiec with *la1* and *la2*. The method *Test_analysis.Run_MCMC()* generates the posterior sample for the correlation lengths specified via *la1* and *la2*, and *Test_analysis.Post()* needs to be run previously with the same correlation lenghts.
n refers to the number of simulations that the MCMC algorithm makes, *BurnIn* is the considered Burn-In while *ss* is the saves to sample. *saveoutput* indicates if output should be save while *saveconfig* indicates if MCMC configuration data should be saved. It's recommended that *saveoutput* and *saveconfig* are set to True. *Test_analysis.Run_MCMC()* returns a tuple *(simu, logpost)* and saves the output in an *output* folder. *simu* contains the posterior sample while *logpost*

contains the Log-Posterior of the posterior distribution.

Also, it should be noted that both the methods *Test_analysis.Post()* and *Test_analysis.Run_MCMC()* create files and folders inside the $folder\_name$ folder. Files created by *Test_analysis.Post()* are not meant to be handled by the user, and are required and used by *Test_analysis.Run_MCMC()*. *Test_analysis.Run_MCMC()* create new files inside a folder which should be named like *Sim_la1_35_la2_45* which are also not meant to be used by the user, inside this folder should be an *output* folder containing the files relevant to the user such as the simulations, the Deviance Criterion Information (DIC), MCMC config files and others.

References

Montesinos-López, J. C., A. Capella, J. A. Christen, and J. Tago (2020). Uncertainty quantification for fault slip inversion.