PCAIM User's Manual

Andrew Kositsky
California Institute of Technology

12 May, 2010

Version 1.0 COMPILATION DATE: May 12, 2010, TIME: 13:52

PCAIM User's Manual

January 03, 2010

Published online at http://www.tectonics.caltech.edu/resources/pcaim/download.html on January 04, 2010.

Revised by Hugo Perfettini, February 10, 2010.

This manual is dedicated to
my mother, father and brother
for helping me become who I am today

Contents

List of Figures

List of Tables

Acknowledgements

The materials herein are based on methodology designed by Andrew Kositsky and Jean-Philippe Avouac, and on research applications by Andrew Kositsky, Hugo Perfettini, Nina Lin, and Marion Thomas. The PCAIM code which this manual documents was designed and written by Andrew Kositsky and Hugo Perfettini, and includes scripts originally written by Y. Okada (`disloc.f`, based on [Oka85] and [Oka92]) with contributions from Nina Lin (InSAR time-series inversion), Yaru Hsu (original single-epoch GPS inversion), L. Shure (`fnnls.m`), Martin King (the original of `conjugate_gradient.m`), Nathan Srebro (algorithm of `decomp_srebro_EM` and `decomp_srebro_EM_projection`), Peter Cervelli (`local2llh`), and an unknown author (possibly Peter Cervelli) (`polyconic.m` and `llh2localxy.m`).

This manual would not have been possible without the support of my colleagues and friends. The manual was improved thanks to proof reading by my friends Kim Hancock, Morgan Appleberry, Linda Granger, and Cassandra Jerde. The code has benefitted from helpful testing by Nina Lin, Marion Thomas, Yaru Hsu, Thomas Ader.

I thank Tapio Schneider for his advise applied mathematics methods, and in particular for his idea of multi-component joint inversion via block diagonal Green's functions allowing the easy incorporation of sparse datasets.

I thank Yangyang Liu for helping me compile the .m-files in Section 9 and for answering so many of my LaTeX questions.

I thank Eric Stansifer for the (now) obvious definitions of α, β, and γ in Appendix C.

I thank Hugo Perfettini for his partnership in developing and testing this public version of the code.

Most of all, I thank Jean-Philippe for his years of guidance, care, and mentorship.

We acknowledge support by the Gordon and Betty Moore Foundation (through the Tectonics Observatory), the National Science Foundation (through grant EAR-0838495), the Institut de Recherche pour le Developpement, and the Caltech SURF program.

Chapter 1

Introduction

This is a public-beta version of the Principal Component Analysis-based Inversion Method (PCAIM) software package. If you have any suggestions or comments, please e-mail `pcaim@gps.caltech.edu`.

1.1 Purpose

The primary purpose of this code is to allow the inversion of time series of surface displacement, strain or tilt for the time evolution of a source of deformation at depth (namely slip on a pre-determined faults system, opening of dykes, or magmatic inflation). We assume that the user is familiar with the PCAIM of [KA10] and with the theory relating subsurface deformation and surface displacements assuming an elastic medium (e.g., [Oka85, Oka92]; [Mog58]; [Coh99]). The code has been written in MATLAB so as to make portability an almost non-existent issue. The code has been tested on Mac OS 10.4-6, Windows and Linux operating systems, and it has also been tested on MATLAB versions 2008a, 2008b, 2009a. While we did not hope to program in every conceivable type of useful input data, we have provided a standardized method through which a user can modify the code to include customized data types and Green's functions (relating surface displacements, or strain, with subsurface sources of deformation). In this public edition 1.0 of PCAIM, we provide functionality for arbitrary sets of:

- continuous 3-component GPS data
- continuous 2-component GPS data
- campaign 3-component GPS data (with enough data samples)

- campaign 2-component GPS data (with enough data samples)

with the optional addition of

- a single InSAR image

The next version of the code will allow in addition inversion of SBAS-processed InSAR, electronic distance meter (EDM), and creep meter time-series.

Because we have data loading functions, fault models, inversion algorithms, and plotting functions all built-in to the PCAIM software, the user has all the ingredients for inversion of single InSAR images, static coseismic inversions, and interseismic coupling maps providing a consistent framework to analyze a variety of data.

In addition to these direct functions of the code, we have designed and included a number of tools for creating fault geometries (Section 2.6), calculating Green functions (Section 2.6.3), and computing a discrete approximation of the Laplacian on an irregular sampling grid (Section 2.6.2). The user can employ these separately from the inversion routine to design a source geometry for any purpose (including producing a source geometry for a later inversion routine), or the user can define a suite of source geometries and automatically find the optimal geometry form this suite by iterating over the inversion routine.

Another advantage of the PCAIM program is the customizability of the script. By being coded as simply as possible with an online database of user-provided additions, the code is meant to be easy to understand and extend.

1.2 Installing the Software

To install the PCAIM software, the user needs to:

1. Register for the software at `http://www.tectonics.caltech.edu/resources/pcaim/`.

2. Download the zip archive from `http://www.tectonics.caltech.edu/resources/pcaim/`.

3. Expand the archive.

4. Put the resulting folder (henceforth to be called the "main PCAIM folder") in the location of the user's choice. There is a folder called `Code` within the main PCAIM folder (henceforth to be called the "code folder").

5. Find the file PCAIM_driver in the main PCAIM folder and change the string assigned to `code_dir` to the full path of the code folder on the user's computer.

6. The user may need to compile the Fortran code (see Section 1.2.1) to compute Green's functions with the software package.

7. The software should now be useable.

There are two ways to make the code accessible to MATLAB.

1. Each time the user opens MATLAB and desires to use the code, manually open PCAIM_driver.m and click on the "run" button (Green Arrow at the top of the editor window. MATLAB will ask if the user wants to change the current directory, or add the directory of the .m-file to the PATH variable, click "change directory." After about a second the code will add all of the proper sub directories for the code and make all the PCAIM scripts accessible to the user.

2. Add the main PCAIM folder and all of its sub directories to the default MATLAB path.

1.2.1 Green's functions

The Green's functions to convert fault slip on a rectangular fault or point source at depth to surface displacement were written in FORTRAN by Yoshimitsu Okada [Oka92], and a convenient wrapper has been written by by Hugo Perfettini. While we include several compiled versions of the FORTRAN code, we have also included the source code. Instructions for compilation written by Hugo Perfettini are below:

Requirements:

- ar: basic unix command

- gfortran: free Fortran compiler (GNU product). gfortran can be download for windows, Mac OS X (tiger , leopard, snow leopard), and linux at:
  http://gcc.gnu.org/wiki/GFortranBinaries, or
  http://hpc.sourceforge.net/.

Instructions:

1. Install the Fortran compiler

2. From the main PCAIM directory, go the GREENFUNC directory:
   `cd Code/Fault\ Related/GREENFUNC`

3. Make sure the compiler script 'compile' is executable on linux or unix, typing:
   `chmod +x ./compile`

4. Build the subroutines listed in 'Sublist' typing, and the programs listed in 'Proglist' typing:
   `./compile`

5. Check the results:

   a) Go in the 'bin' folder: `cd bin`

   b) Execute the point source program, typing:
      `./displacement_green_fcn_point_source`

   c) Execute the rectangle program, typing:
      `./displacement_green_fcn_rectangle`

   d) Check that the results are ok by comparing with the included TEST files.

      - For rectangular dislocation, type:
        `diff GREEN_FCN.rect GREEN_FCN.rect.TEST`
      - For point source dislocation, type:
        `diff GREEN_FCN.trg GREEN_FCN.trg.TEST`

      If everything is ok, the user should get the prompt with no messages. This means that the files `GREEN_FCN.rect` and `GREEN_FCN.rect.TEST` are identical (which they should be). In case they are not, check the output file `GREEN_FCN.rect` and see if the differences with `GREEN_FCN.rect.TEST` are not marginal (i.e., due to rounding on the last digit).

From here we give an overview of the theory behind PCAIM. While we strongly suggest the user review the theory behind PCAIM, if the user wishes to preview the results of the code via a tutorial, the user may skip to Chapter 4.

Chapter 2

Theory

In this chapter we review the assumptions and methods for translating a set of surface displacement or strain data (e.g. InSAR images, GPS time-series, strain meter time-series, etc.) into a source model (e.g. point, triangular or rectangular fault patches, 'Mogi' inflation sources, etc.) at depth. For convenience and clarity we use the terminology associated to the case where the source of deformation is slip on a fault.

2.1 Overview

We give here an overview of the methodology implemented in the PCAIM code from [KA10]. The reader is referred to [KA10] for more details.

Let us consider a set of geodetic positions measured at a number of sites and at a number of dates, called epochs. The measurements made at different sites might correspond to different epochs. We call a set of data measured at the same location and orientation (e.g. the North component of a GPS measurement station) a time-series. We place time-series in a $m \times n$ matrix, X_0, where each row corresponds to a single time-series, and each column corresponds to all data measured at a given epoch. For entries where we have no measurement we fill in a default value and mark these entries as missing data.

We suppose that displacements are due to an unknown, time-dependent slip distribution on a discretized fault with known geometry α. The slip vector on each subfault is decomposed into a strike and a dip component. We assume that the medium surrounding the fault is elastic, and we represent fault slip by a matrix $\mathcal{L}$ where each row refers to both components of slip (strike-slip and dip-slip) on a given subpatch and each column refers to an epoch. Let G_α denotes the Green's functions relating surface displacements with fault slip at depth (decomposed into a strike-slip component and a dip-slip component), given a fault geometry α, and C is a matrix with each row equal to a constant, representing the position of the corresponding site for a zero slip. Then surface displacements (with the exception of missing data) then obey:

$$X_0 = G_\alpha \mathcal{L} + C. \tag{2.1}$$

The Greens function's G_α can be computed from the semi-analytical solutions of [Oka92] for a dislocation embedded in an elastic homogeneous half-space using the scrips provided with this code. The Green's function could alternatively be computed based on the triangular fault patch source model of [Mea07] or a multi-layer elastic half-space models (e.g., [XY89]).

Determination of the time-dependent slip model corresponding to the measurements requires inversion of that linear system. The Principal Component Analysis-based Inversion Method relies on the following principles:

1. The datasets can be decomposed as the sum of components, each component being associated with a pattern of surface displacement and a time function. (Linearity)

2. Only a small number of components is generally necessary to explain most of the data. (Low-Rank)

3. The pattern of surface displacements associated with each component can be inverted for some principal slip distribution. (Invertibility)

4. The fault slip distribution corresponding to the original dataset can be derived by linear combination of the principal slip distributions. (Linearity)

In practice PCAIM flows as follows:

1. Center X_0 along its rows and call the centered matrix X.

2. Decompose and approximate X as the matrix product of at least two matrices (e.g. $X \approx UV^t$), with the left-most matrix of low rank.

3. Invert the left matrix (columns of U) for slip distributions L as if they were ordinary displacement vectors at the surface via some Green's function matrix G, i.e. solve the matrix equation $G \cdot L = U$.

4. Sum the slip distributions multiplied by their associated time functions (V^t = all matrices in the matrix product except the left-most) (LV^t). Then as $G \cdot L \approx U$, $G \cdot LV^t \approx UV^t \approx X$.

$$\begin{tikzcd}
\text{Surface Displacement} \arrow[r, "\text{Linear Decomposition}"] \arrow[d, shift right, "\text{Inversion}"'] & \sum_{i=1}^{r} \text{Linear Component}(i) \arrow[d, shift right, "\text{Inversion}"'] \\
\text{Slip History} \arrow[u, shift right, "\text{Elastic Dislocation Forward Modeling}"'] & \sum_{i=1}^{r} \text{Slip Distribution}(i) \arrow[l, "\text{Linear Recombination}"'] \arrow[u, shift right, "\text{Elastic Dislocation Forward Modeling}"']
\end{tikzcd}$$

Figure 2.1: General schematic for PCAIM.

It is useful to think of PCAIM based on the diagram in Figure 2.1. The left-hand track represents directly translating displacement data into a slip model by inverting the difference in surface displacement between consecutive epochs for incremental fault slip. PCAIM instead divides the displacement data into the sum of linear components. Each of the components can be inverted individually into a corresponding slip distribution. It should be noticed that each individual component corresponds to a linear combination of the contributions from various sources and not to a particular, identifiable physical source. In general, each component has no obvious physical meaning when considered alone, although the various components might be recombined to extract the contribution of particular sources [KY06, KY09].

The major functions of the actual program as-written are diagramed in Figure 2.2.

2.2 Basic Assumptions

In order to apply PCAIM to a dataset, one needs to accept certain general assumptions:

Invertibility: The observations between any two epochs can be plausibly modeled as resuting from a distribution of slip on a pre-defined fault $\mathcal{F}$.

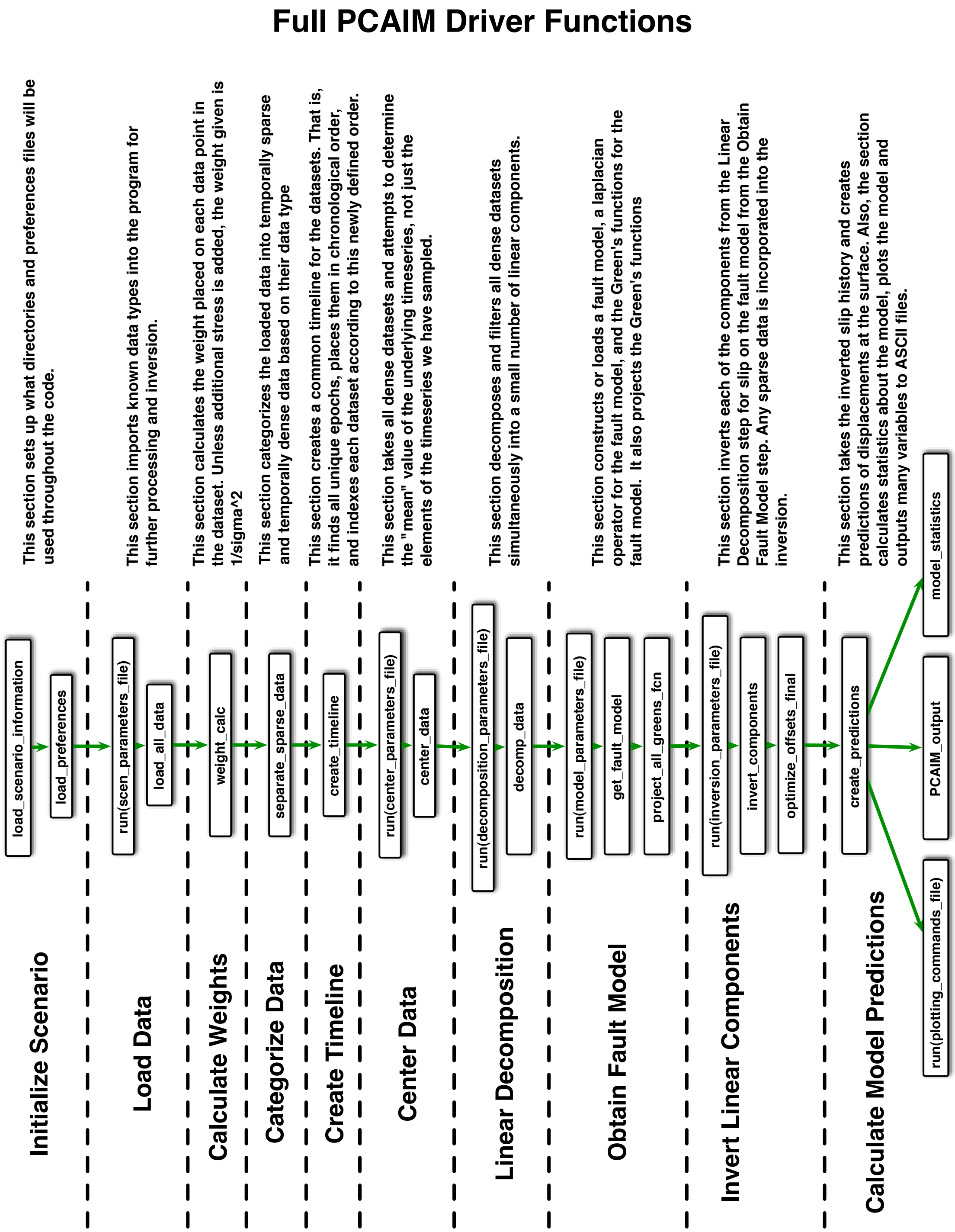

Figure 2.2: Diagram of the full PCAIM program.

Linearity: We can calculate the effects to the observation points from finite dislocations on a pre-defined fault using Green's functions that are linear both in time and between sources.

Low-Rank: We assume that most of the data can be described with a small number of linear components. That is, the data matrix X has low-rank.

2.2.1 Invertibility

For any vector of measurements on the surface $\vec{d}$, we assume that there exists a fault slip history $\vec{l}$ in $\mathcal{F}$ such that if $\boldsymbol{G}$ is a set of Green's functions converting slip at depth on fault $\mathcal{F}$ to surface displacements,

$$\vec{d} = \boldsymbol{G}(\vec{l}). \tag{2.2}$$

2.2.2 Linearity

For any set of Green's functions $\mathcal{G} = \{G_1, G_2, \cdots, G_N\}$ relating slip on patches on $\mathcal{F}$ to an observation point p and time variations of slip at depth on these patches $\mathcal{T} = \{f_1(t), f_2(t), \cdots, f_N(t)\}$ defined for some $t = t_1, t_2$, the change in displacement at p between t_1 and t_2 is,

$$\begin{aligned} d_p(t_1 \to t_2) &= d_p(t_2) - d_p(t_1) && (2.3)\\ &= \mathcal{G}\left(\mathcal{T}(t_1 \to t_2)\right) && (2.4)\\ &= \sum_{l=1}^{N} \left(G_l f_l(t_1 \to t_2)\right) && (2.5)\\ &= \sum_{l=1}^{N} \left(G_l f_l(t_2) - G_l f_l(t_1)\right). && (2.6) \end{aligned}$$

In equation 2.3 we use the linearity of measurements at the observation locations between times t_1 and t_2. In equation 2.4 we use invertibility of displacements between any two epochs. In equation 2.5 we use the linearity in space assumption, specifically the effect of two dislocations on point p is the sum of the effect of each dislocation on observation point p. In equation 2.6 we use the linearity in time assumption, specifically the effect on the observation point p for each dislocation is the difference of the cumulative effect on the observation point p of the difference of slip at depth at times t_2 and t_1.

The Linearity assumption is intentionally written to be very general.

A benefit of the linearity assumption regarding the measurements of surface displacements is that we can decompose the time-series into a number of linear components. For example, we can thus apply singular value decomposition (SVD) (or any other linear decomposition composed of linear combinations of the columns) to a complete matrix of the time-series and invert the components.

2.2.3 Low-Rank

The low-rank assumption is what allows us to truncate the decomposition and approximate the original matrix. We only want to model real surface displacements as slip at depth, not any noise that may be present in the dataset. However, every dataset has noise and we need a way to filter out the noise. Principal Component Analysis (PCA) via truncation of a singular value decomposition is a common solution to this problem. Traditional PCA effectively assumes that every datum has the same error, or that the data matrix has a rank-1 error matrix (so we can perform weighted SVD). This is far from true in the case of most GPS time series, especially where there are missing data in the time-series. We take use the same approach of traditional PCA of modeling the data as a sum of a small number of linear components, but we employ a different decomposition as described in Section 2.4 that allows arbitrary weights places on each datum.

The user must justify this assumption by (1.) assessing the amount of data χ^2 explained by each component, and (2.) demonstrating the residuals from the chosen number of components can be considered as noise.

2.3 Centering

PCAIM method relies on the fact that each component U can be inverted for fault slip at depth [KA10]. There must be a solution of displacement at depth, L, such that when we multiply on the left by the Green's functions

$$U \approx GL. \tag{2.7}$$

Then we note that we can replace U in $X \approx USV^t$ by equation 2.7 to get

$$GLSV \approx X. \tag{2.8}$$

However, except in the case of spatially continuous data (such as InSAR), there is no simple way to compute the relative values of the rows of X as they all have arbitrary offsets. In other words, the displacement time-series $[-2, -1, 1, 2]$ represents the same deformation over four epochs as the time series $[1000, 1001, 1003, 1004]$. This means that we would want our representation in USV^t to be the same for both. These two time-series viewed as vectors are close to orthogonal. The only difference between them is a constant offset of 1002 – which means nothing geodetically. Nonetheless, mathematically the difference between $A = \begin{bmatrix} -2 & -1 & 1 & 2 \\ -2 & -1 & 1 & 2 \end{bmatrix}$ being a rank-1 matrix and $B = \begin{bmatrix} -2 & -1 & 1 & 2 \\ 1000 & 1001 & 1003 & 1004 \end{bmatrix}$ being a rank-2 matrix is very significant. A one-component decomposition of A completely explains the data whereas the one-component decomposition of B does not. In other words, in the case where the time series from the different sites have missing data at different epochs the naive centering of X could introduce non-physical offsets between the various rows of X.

In order to avoid this issue in cases where the data is *missing completely at random* (MCAR), we can just remove the error-weighted mean from the each time-series individually [LR02]. This forces the time functions in V to be zero mean or very close to zero mean; otherwise the weighted norm of V and any row of our data matrix X would be different and we could improve the fit by adjusting X up or down by a constant. We say "very close to zero" because V will likely only have a very particular weighted mean (not necessarily the arithmetic mean) that is zero. The MCAR assumption, however, almost never holds for real data sets. For example, cGPS3 stations often are missing large chunks of data from being installed after the surrounding stations, equipment failure, theft, or inability to collect data. This means there may be a systematic bias in the estimate of the mean in comparison to an entire time-series. See Figure 2.3 for an example of how using the weighted mean on datasets with data that are not MCAR gives a wrong answer for the mean. A wrong mean, as we have seen, can decrease the amount of data explained for a given number of components. As our principal goal is to explain as much data as possible with the smallest number of components, we need a better way to estimate the mean.

One solution to this problem is to model the time series and impute the missing data values [LR02]. However, even this solution suffers two drawbacks. First it's not entirely clear how to model the time series without assuming a functional form for the time series, something we intentionally are avoiding as such an assumption would further bias the final solution. Second, it is not altogether clear what weight to give these new data points. Instead we invoke the assumption of low-rank in order to design a model where all time functions (i.e. columns of V) to have zero mean and the mean of each time-series is allowed to vary.

Because we expect the signal at all closely placed temporally dense stations to be approximately from the same underlying time functions (this is implicit in the low-rank

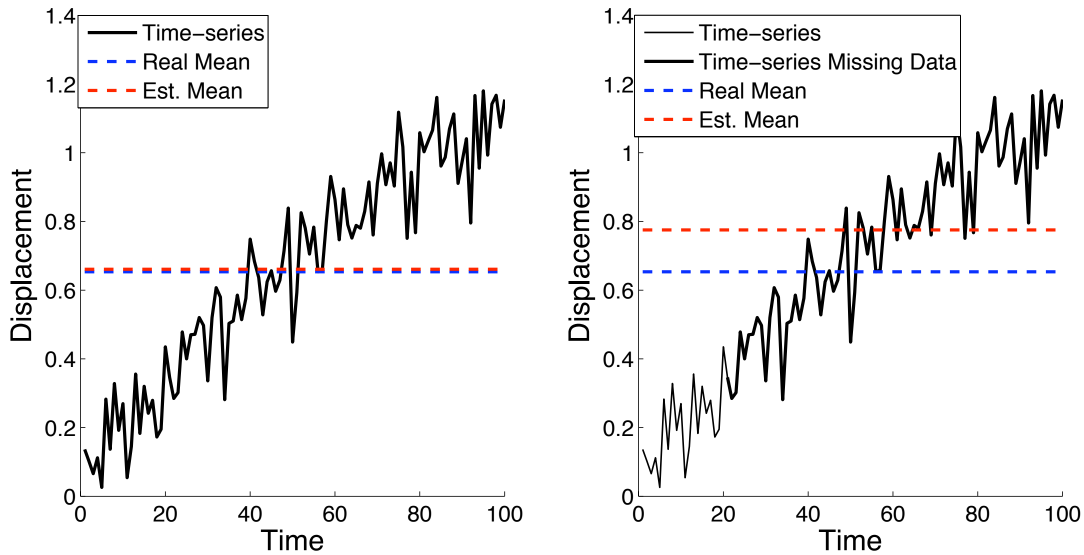

Figure 2.3: *The Importance of Proper Centering:* We model a logarithmic decay, for instance from post-seismic relaxation, for 100 equally-spaced epochs with a time constant of 50 epochs. The true mean of this time-series is 0.6534, of which we have a good estimate (0.66068) even with error of $\sigma = 0.1$ (Left). However, if we are missing the first 20 epochs, the estimate of the mean is 0.77544 (Right). If we knew the functional form of the time-series, it would be easy to get a good estimate of the mean.

assumption), simply with different offsets and geometric factors, our strategy is to find the mean at the same time as we do a linear decomposition. That is, we want to find a decomposition

$$USV^t + M \approx X \tag{2.9}$$

with USV^t as low rank as possible explaining as much chi^2 as possible. A further complication we need to avoid is a trade-off between USV^t and M. For any k, if $(USV^t + M)$ is a good model of X, then $US(V^t + c) + (M + \sum USc) = (USV^t + M)$ is also a good model for any constant matrix c (indeed, this holds for any matrix with constant rows). While this does not actually change U (and consequently does not change L) in the case given, it represents additional degrees of freedom we wish to remove from the model. By forcing V to have zero mean, these degrees of freedom are removed and the means for the datasets are determined for X by the best values of M. How we find this decomposition of X into $USV^t + M$ is described in Section 2.4 as it relies on the same basic approach as the general decomposition step of the PCAIM algorithm.

2.4 Decomposition

If we have a data set that is not missing any data and has identical error bars on each datum from any given time-series (or identical error bars on each datum from any given epoch), then we can decompose the data into a small number of linear components using truncated Singular Value Decomposition (tSVD) or weighted truncated SVD (wtSVD). tSVD minimizes the variance between the original matrix and a rank-k matrix. For nearly-complete time-series, such as that recorded by the SuGAR network from the post-seismic relaxation from the Nias 2005 earthquake, we can impute the missing data by assumption of some functional form for the each time-series[KA10]. However, the time-series may not always be so complete, we may not want to assume a particular functional form for the time-series, or the data may have greatly varying error bars. To decompose the data into a number of linear components in this case, we need a different decomposition than SVD.

The 'reduced Chi-squares', χ^2, is a most commonly used quantity to characterize the fit between observations and predictions. Mathematically, it is defined to be

$$\chi^2 = \sum_{i,j} \left(\frac{X_{\text{model}}(i,j) - X_{\text{dat}}(i,j)}{\sigma(i,j)} \right)^2 . \tag{2.10}$$

As this is (or at least can be) what we are trying to minimize when we say we want to "fit the data with a model," it makes sense for our decomposition to attempt to minimize this quantity. The expression for a general linear model X_{model} is $UV = X_{\text{model}}$, where U and V are matrices of compatible dimension and UV is the standard matrix product of U and V. We will often refer to U as the spatial functions or spatial basis functions, and to V as the temporal functions or temporal basis functions. For matrix entries $X(i,j)$ where we do not have any data, we assign $\sigma(i,j) = \infty$ and choose $X(i,j)$ to be any finite value. Thus the contribution from the (i,j) entry will be $\left(\frac{X_{\text{model}}(i,j)-X_{\text{dat}}(i,j)}{\sigma(i,j)}\right)^2 = \left(\frac{X_{\text{model}}(i,j)-X_{\text{dat}}(i,j)}{\infty}\right)^2 = 0$. This formulation of the decomposition problem allows us to consider decompositions for incomplete datasets and datasets with highly variable error bars.

To the best of the author's knowledge, this decomposition, referred to here as the Srebro-Jaakkola decomposition, was first introduced in [SJ03] and has been successfully used in a number of applications in other fields. One important and counter-intuitive point is that several properties of the Srebro-Jaakkola decomposition are quite different from traditional SVD. A few of those points the author deems most important are listed below.

1. *The best rank-k approximation of X_{dat} is not the rank-$(k-1)$ approximation of X_{dat} plus an additional component.*

   In traditional SVD, we compute the first component, subtract out the first component from the matrix, compute the second component, subtract out the second component from the matrix and so on until we have a zero matrix left over. By the nature of singular vectors, the best approximation of X_{dat} with k linear components is the first k components as found iteratively above. However, this is not true with the Srebro-Jaakkola decomposition. For example, take the following data and error matrices:

$$X_{\text{dat}} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 1 & 3 & 4 & 5 \\ 1 & 2 & 3 & 5 \end{pmatrix}, \quad X_{\text{err}} = \begin{pmatrix} 1 & 10 & 1 & 1 \\ 10 & 10 & 10 & 10 \\ 1 & 1 & 1 & 1 \\ 10 & 1 & 1 & 1 \end{pmatrix} \tag{2.11}$$

   For clarity, we write the decomposition as USV^t like in SVD, where U, V are orthogonal matrices and S is diagonal. All calculations were done to double precision and are truncated here for clarity.

   The best one-component model of X_{dat} is,

$$U_1 = \begin{pmatrix} -0.407041 \\ -0.576466 \\ -0.558243 \\ -0.436313 \end{pmatrix}, S_1 = (12.735031), V_1 = \begin{pmatrix} -0.232065 \\ -0.400816 \\ -0.547421 \\ -0.697010 \end{pmatrix} \tag{2.12}$$

   and has $\chi^2 = 3.949337$.
   The best two-component model is,

$$U_2 = \begin{pmatrix} -0.414111 & 0.193785 \\ -0.556585 & -0.776221 \\ -0.54068 & 0.136531 \\ -0.475805 & 0.584198 \end{pmatrix}, S_2 = \begin{pmatrix} 13.155585 & 0 \\ 0 & 0.799207 \end{pmatrix},$$

$$V_2 = \begin{pmatrix} -0.209242 & -0.729478 \\ -0.379241 & -0.37701 \\ -0.532292 & -0.183232 \\ -0.727364 & 0.540511 \end{pmatrix} \tag{2.13}$$

and has a $\chi^2 = 0.566818$.
However, if we compute the second component as the first component of $(X_{\text{dat}} - U_1 V_1)$, we have,

$$U_2' = \begin{pmatrix} -0.407041 & -0.407176 \\ -0.576466 & 0.213847 \\ -0.558243 & -0.249282 \\ -0.436313 & -0.852254 \end{pmatrix}, S_2' = \begin{pmatrix} 12.735031 & 0 \\ 0 & 1.085772 \end{pmatrix}, \tag{2.14}$$

$$V_2' = \begin{pmatrix} -0.232065 & 0.387613 \\ \\ -0.400816 & 0.188841 \\ \\ -0.547421 & -0.063751 \\ \\ -0.697010 & -0.900017 \end{pmatrix} \tag{2.15}$$

with a $\chi^2 = 1.064647$.
Feel free to play with this example in the file `PCAIM_manual_examples.m`.

While it's difficult to compare these closely by eye, a few aspects strike us immediately. First, the first column of V and the first column of U are not the same for the two decompositions. Second, the first weight (similar to singular values) is not the same. Applying some numerical test, we indeed see that U_1 and V_1 do not even lie in the space spanned by the columns of U_2, V_2!

$$\max(S_2(:)) - \max(S_1(:)) = 0.420554 \tag{2.16}$$
$$\text{norm}(U_1' \cdot U_2) = 0.999546 \tag{2.17}$$
$$\text{norm}(V_1' \cdot V_2) = 0.999899 \tag{2.18}$$

2. *The weight of a given component is not proportional to its fraction of χ^2 explained.*

In traditional SVD or wSVD, the singular values are proportional to the amount of chi^2 explained by the model, but this does not hold for the Srebro-Jaakkola decomposition. This is more straightforward and more easily seen than the previous property. Consider the data and error matrices:

$$X_{\text{dat}} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 10^2 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{pmatrix}, \quad X_{\text{err}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 10^{2.5} & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{2.19}$$

It is clear that a very good first component is $U = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}, V = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$.

In fact the best one-component model is very close to this, $U = \begin{pmatrix} 1.0000 \\ 2.0000 \\ 3.0001 \\ 4.0000 \end{pmatrix}, V = \begin{pmatrix} 1.0006 \\ 1.9920 \\ 3.0019 \\ 4.0025 \end{pmatrix}$. Normalizing the components, we get $S = 30.0003$.

Since X_{dat} is a rank-2 matrix, it's clear that a best two-component model is $U = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 3 & 10^7 \\ 4 & 0 \end{pmatrix}, V = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 3 & 1 \\ 4 & 0 \end{pmatrix}$.

Normalized, this is $S = \begin{pmatrix} 30 & 0 \\ 0 & 10^7 \end{pmatrix}$. By any reasonable metric, the $||S_2||_2 = 10^7$ causes the two-component model to have a very large amount of variance explained compared to the one-component model with $S_1 \approx 10^1$. However, it is equally clear that the second component is unnecessary to explain the data because of the large error of $X_{\text{dat}}(3, 2)$ which is principally responsible for the large norm of S_2. Thus despite the very large "weight" in S accorded to the second component (similar to singular values from SVD), much more of the χ^2 from the data is explained by the addition of the component with a smaller "weight".

These two considerations change the way the various ranks of decompositions must be interpreted. In order to compare the fit of two different models we cannot simple compare the "singular values"[1] (S) or weights from the decomposition, and we cannot build an

[1]The author admits these are not strictly singular values, but the use of vocabulary is for those familiar with SVD.

optimal model iteratively the way we could using SVD.

For our purposes, we have implemented three different methods for finding two different incarnations of this decomposition. As a verification of the EM and CG algorithms, the user is free to check that the EM algorithm and MATLAB's SVD routine obtain the same result (up to numerical error) if the tolerance on the EM algorithm is small enough and the error matrix is a constant value.

2.4.1 Srebro-Jaakkola Decomposition – Expectation Maximization

Using code given to the author by Nathan Srebro as the base [Sre], we have implemented an expectation maximization (EM) routine for computing the optimal rank-k decomposition. See [SJ03] for precise definition of the EM algorithm.

2.4.2 Srebro-Jaakkola Decomposition – Conjugate Gradient

Using code posted for free use online by Martin King [Kin05] for the general conjugate gradient algorithm, we implemented the local search for the optimal rank-k decomposition. In the author's experience, almost all[2] starting locations end up at the same minimum which the author takes to be the global minimum. We use the objective function equal to the weighted χ^2 and the derivatives of the weighted χ^2 with respect to each entry of U and each entry of V. In order to reduce computation time, we have replaced the general line search algorithm of [Kin05] with an exact minimum solving along the direction of search for this specific objective function

2.4.3 Srebro-Jaakkola Decomposition – CG Meanless-V for Centering

Using code posted for free use online by Martin King [Kin05] for the general conjugate gradient algorithm as a base, we implemented the local search for the optimal rank-k decomposition. In the author's experience, almost all[3] starting locations end up at the same minimum which the author assume to be the global minimum. We use the objective function equal to the weighted χ^2 for the model $UV^t + M$ and the derivatives of the weighted χ^2 with respect to each entry of U, the first $n-1$ entries of V, and the mean for each time-series M. In order to reduce computation time, we have replaced the general

[2]Except for where U and/or V start out being identically zero.

[3]Except for where U and/or V start out being identically zero.

line search algorithm of [Kin05] with an exact minimum solving along the direction of search for this specific objective function. Analytical expression are in Appendix C.

2.4.4 Derivatives for CG algorithm

For each of the objective functions we wish to use with the CG algorithm, we need to find the analytic derivative. We do that in Appendix B so the user can verify the calculation if desired.

2.5 Temporally Dense vs. Sparse Data

A major advantage to the decomposition in Section 2.4 is that we can accommodate missing data points. This allows us to do combined analysis datasets with different fundamental frequencies, such as cGPS networks and campaign GPS measurements. Problems arise, however, when there are too few data compared to the number of components used.

In many ways, modeling only makes sense in terms of prediction. If we are not attempting to predict anything, we might as well fit every set of N data points with an order $N-1$ polynomial, which will always fit perfectly.

Similarly, if we have only N epochs worth of data for some dataset, the combination of a mean (1 parameter per time series) and $N-1$ components ($N-1$ parameters per time series) can fully explain almost any time series[4]. This implies that any error will be fully included in the inversion step. More importantly, the predictions from the model depend very heavily on the individual value of the error on each datum. We avoid this difficulty by having two designations for data:

sparse data, meaning (number of epochs) $\lessapprox$ (number of components), and

dense data, meaning (number of epochs) $\gg$ (the number of components).

Any data that is designed dense is centered and used in the decomposition. Any data that is designated sparse is not centered or used in the decomposition. Instead, it is added to the inversion as a set of additional linear equations (see Section 2.7.2 for details) and we say that the sparse dataset allows us to apply *sparse constraints* to the inversion.

[4]The mean + $N-1$ components explaining any time series does depend slightly on the time functions in V, but this holds for all but specially constructed V. Proof is left as an exercise to the user.

2.6 Fault Models

The decomposition and inversion methods do not depend on the type of the data or of the source of deformation as long as there exists an appropriate set of Green's function relating each observation to each source element. However, for the purposes of this manual, we only describe the construction of a finite fault model, of a discrete Laplacian for the fault model, and the Green's functions for that fault model.

For this version of the code, we have implemented two different types of source models, rectangular dislocations and point dislocations in a homogenous elastic half-space [Oka92].

2.6.1 Construction

Constructing a fault model is an attempt to approximate the true crustal structure at or beneath the earth's surface, and we provide two simple formalisms with the code to help the user build such a model. The first will be referred to as a *rectangular source model* [Oka92], which is composed of finite dislocations in an elastic half-space on rectangular patch elements with two edges of each patch parallel to the $z = 0$ surface. The second will be referred to as a *point source model* [Oka92], which is composed of finite dislocations in an elastic half-space on point elements with a given seismic mechanism. The source model as a whole is a finite collection of sources from whatever formalism we choose.

Each source element is composed of several parameters which describe its dimensional extent and influence. These definitions are given by 7 parameters for the rectangular source element case (local x, local y, local z, strike angle, dip angle, length, width) and 6 parameters for the point source element case (local x, local y, local z, strike angle, dip angle, area). In both cases (local x, local y, local z) are measured from the local coordinate frame's origin to the center of the source element. See Figure 2.4 for a graphical description of these parameters. The format for these input files is in Section 8.2.1.

We include scripts that allow the user to specify a list of points (format in Section 8.2.3) which are assumed to be on a two-dimensional surface from which an approximately regular triangular mesh is resampled. The user may apply smoothing to this surface if the points are approximate (e.g. from a relocated earthquake catalog.)

2.6.1.1 Resampling the Fault

Assume we already have a fault surface (or a cloud of points that approximate a fault surface) we want to decompose the fault into individual patches. Instead of sampling regularly in geographic coordinates or a standard local reference frame (i.e. using a co-

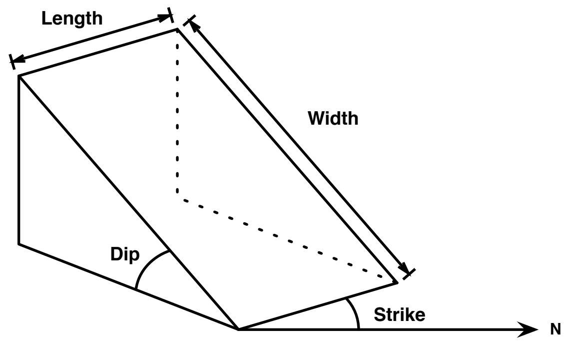

Figure 2.4: *Fault Element Description:* a graphical description of the source elements for the rectangular source element case (strike angle, dip angle, length, width) and the point source element case (strike angle, dip angle, area). (x,y,z) for any element is measured from the local coordinate origin to the center point of the element.

ordinate transform to change (E,N,U) triples into (x, y, z) triples, where at the origin the x direction is E and the y direction is N), we transform into a coordinate frame where x and y lay in the best-fitting plane to the original fault model. This allows us to intelligently resample any fault surface that deviates as much as $\approx 45°$ away from being planar; spacing will then differ by less than a factor of $\sqrt{2}$. Resampling the fault depths z in (x, y) either the geographical reference frame or a standard local reference does not do the job. For example, consider a model of a strike-slip fault. There is at most one line of points in the fault surface, and it is not well-defined at what depth these should lie. Even strike-slip faults that are not perfectly vertical have problems in that small changes in dip can drastically change the resampling density in geographical or standard local coordinate frames (i.e. (x, y, z)). Instead of sampling in one of these reference frames, we transform into a reference frame defined by the user (or estimated automatically). The first two axes of this reference frame are assumed to be an approximately best-fitting plane to the fault surface and the third axis is normal to the first two axes.

2.6.1.2 Visual representation

In order to visualize a source model, we often plot the rectangular source elements as rectangles and the point source elements as triangular patches in a half-space. However, this can be cumbersome when the number of patches becomes large. Instead we represent both source elements as colored circles in a three-space where the color of the circle depends on the amount of displacement on that element. If the direction of slip is important (as it almost always is), vectors will be plotted representing the direction of slip. We often plot contextual information for the slip model, such as the position of surface observations and coast information. An example of this type of plotting is given in Figure 2.5, and the user should note that within MATLAB the user can view the fault from any angle by using the rotation tool in the MATLAB plot window.

2.6.2 Discrete Laplacian

The two-dimensional Laplacian[5]

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{2.20}$$

is often used to regularize ill-posed problems and smooth rough functions.

In order to use it in the context of a discrete grid such as our source model, we must find a discrete version of this equation. The user is likely familiar with the traditional computational template,

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \quad , \tag{2.21}$$

used to approximate the Laplacian to 1st order (That is, the order of the error decays as $(\Delta x)^2$) at the central square on a regular square grid [Ise04]. That is to say, we approximate

$$\nabla^2(x,y) = -4f(x,y) + f(x+\delta x, y) + f(x-\delta x, y) + f(x, y-\delta y) + f(x, y+\delta y)$$

where $\delta x = \delta y$ is the spacing of our regular square grid. This (or a minor variant thereof) has been used successfully to regularize the solution to many geophysical inversions. However by allowing a geometry defined by randomly distributed point sources, we need an approximation of the Laplacian that works for irregular grids. We obtain a satisfactory solution to the problem of an irregularly sampled planar grid from [Hui91], which is summarized in Appendix D. This gives us the approximation,

$$\Delta f_0 \approx \sum_{i=1}^{N} w_i^{(2)} (f_i - f_0), \tag{2.22}$$

[5] For Mogi (inflation) sources, the three-dimensional Laplacian is needed.

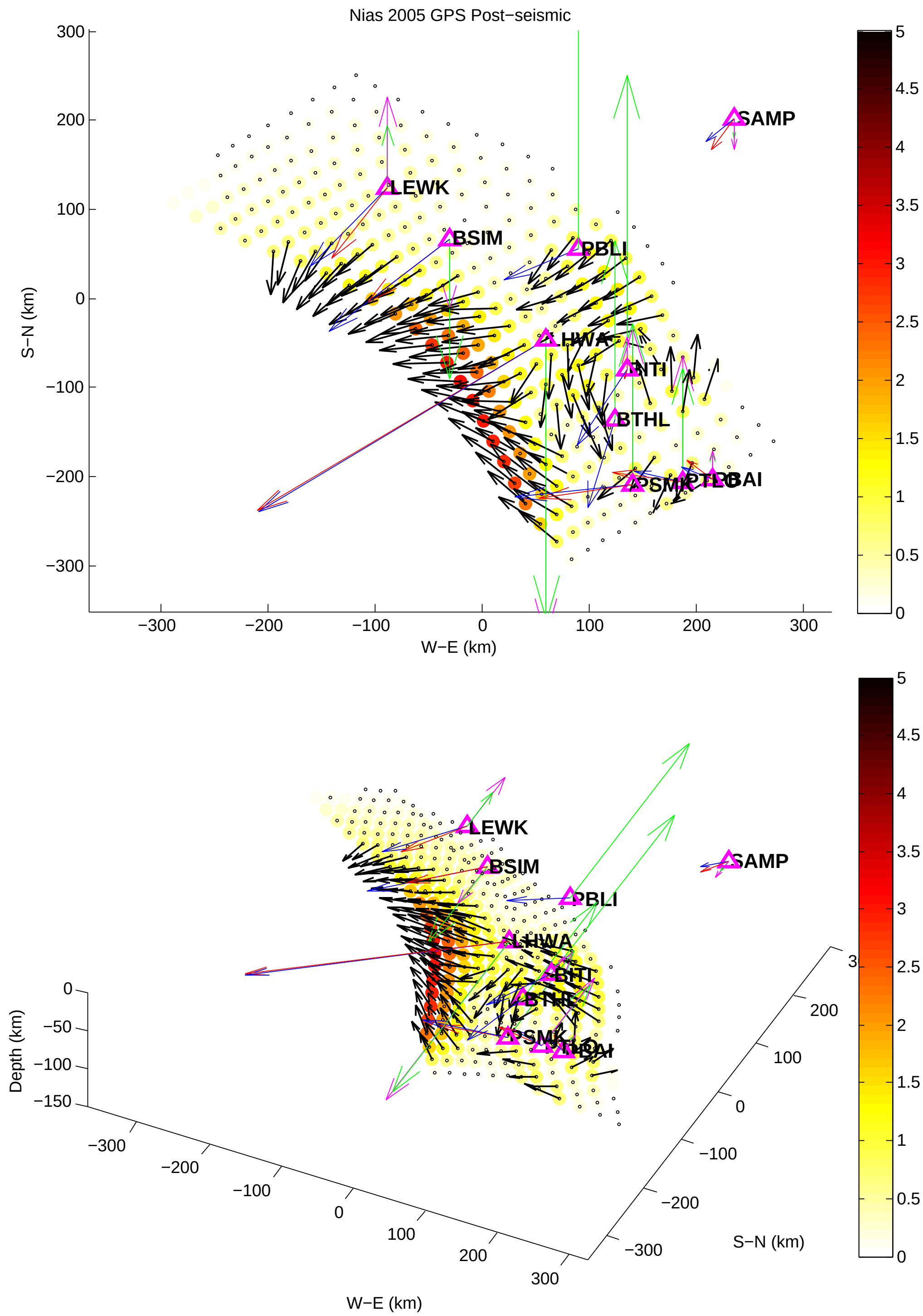

Figure 2.5: *Sample slip plot:* example slip and displacement plots in MATLAB representing superficial afterslip on the Sunda Megathrust from the Nias, 2005 Earthquake [HSA+06, KA10]. Note that these are *not* intended to be publication-quality plots.

where f_0 is the function value at the "central" point, f_i is the function values at the ith neighboring point, N is the number of neighboring points defined by the user, and w_i are the weights derived in Appendix D. We put Δ into matrix form so that for any slip distribution l, Δl is the discrete Laplacian of l. Within the code we use this discrete Laplacian both as regularization of the inversion step and as a method of smoothing a swarm of points estimating a fault surface.

When applied either in order to smooth a fault surface or regularize an inversion, we must decide on the weight the Laplacian will have. We define a parameter `lap_weight` toward this end, which is a double and is linearly multiplied by the discrete Laplacian operator in the code before the Laplacian operator is used.

2.6.3 Green's Functions

The Green's functions are calculated using the original Fortran from [Oka92] via a wrapper by Hugo Perfettini. See Section 1.2.1 for details.

2.7 Inversion

Once we have the spatial functions for dense datasets ($U = [U_1, U_2, \cdots U_r]$, r is the number of components used in the decomposition step) and the Green's functions (G), we are ready for the inversion step.[6] We let l be some unknown slip distribution at depth, then the general formula for the surface displacement field d resulting from l is

$$G(l) = d \tag{2.23}$$

The inverse problem is then solving for l given d and a known form of the Green's functions. We have assumed that G is linear, so Equation 2.23 becomes,

$$G \cdot l = d. \tag{2.24}$$

Because we actually have r surface displacement fields $(U_1, \cdots, U_r)$ we wish to invert and the Green's functions are the same for each displacement field, Equation 2.24 becomes the set of equations,

$$\begin{aligned} G_i \cdot l_i = U_i, \quad i = 1, \ldots, r \\ G \cdot l_i = U_i, \quad i = 1, \ldots, r \end{aligned} \tag{2.25}$$

[6]Note that the user may wish to rescale the spatial functions and Green's functions to allow different input datasets to have different weights during the inversion. This is done by multiplying the Green's function and spatial functions for the ith dataset by some constant p_i.

By independence of rows and columns of a matrix, solving Equation 2.25 is equivalent (both in computation time and solution) to solving,

$$\begin{pmatrix} G & 0 & 0 & \cdots & 0 \\ 0 & G & 0 & \cdots & 0 \\ 0 & 0 & G & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & G \end{pmatrix} \cdot \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_r \end{pmatrix} = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_r \end{pmatrix} \tag{2.26}$$

This is the basic form with which we approach the inversion step of the algorithm. The block-diagonal matrix on the left hand side (LHS) of Equation 2.26 is called the *design matrix* for our problem, the vector on the LHS of Equation 2.26 is called the *solution* or *solution to the inversion problem*, and the vector on the RHS of Equation 2.26 is the *data* or *data for the inversion problem*.

2.7.1 Regularization

The inversion of geophysical data for source models is often an ill-posed problem. This means we have many solutions to the inversion that are mathematically plausible, so the solution to the problem of fitting the observed surface displacements is not unique. To reformulate the ill-posed problem we make additional assumptions to *regularize* the problem.

While there are a number of regularization methods, the default regularization method implemented in the code assumes that the smoothest slip distribution fitting the data is the most plausible solution. In practice we impose a least-squares penalty on a non-zero value of a discrete Laplacian operator[7] (Δ) on the slip in the dip-slip and strike-slip dislocations for each slip distribution. To impose this penalty, we augment the design matrix with a block-diagonal form of the Laplacian (one block per component being inverted) and augment the data vector with zeros (one zero per patch per component being inverted) in Equation 2.26 to obtain,

[7] Section 2.6.2 has a brief description of this operator and Appendix D has a detailed account of its derivation for a non-uniformly sampled plane, which is a good local description for fault models without splays.

$$\begin{pmatrix} G & 0 & 0 & \cdots & 0 \\ 0 & G & 0 & \cdots & 0 \\ 0 & 0 & G & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & G \\ \Delta & 0 & 0 & \cdots & 0 \\ 0 & \Delta & 0 & \cdots & 0 \\ 0 & 0 & \Delta & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \Delta r \end{pmatrix} \cdot \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_r \end{pmatrix} = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_r \\ \vec{0} \\ \vec{0} \\ \vec{0} \\ \vdots \\ \vec{0} \end{pmatrix} \tag{2.27}$$

If there is no sparse data in the datasets, then this is the final form of the inverse problem and we can use either a standard least-squares algorithm, non-negative least-squares algorithm, L_p-norm solve for any p, etc., to solve the inverse problem.

2.7.2 Addition of Sparse Constraints

So far sparse data have not been considered. Since inversion is the step that determines the final slip model, we must add these data here before performing the inversion. We will further augment the design matrix and data vector with constraints representing the displacement at the surface from the sparse datasets.

We start with the joint inversion formulation from Equation 2.27. We note that from the decomposition assumptions, the slip at between epochs t_1 and t_2 is

$$L[V(t_2,:) - V(t_1,:)]^t,$$

so displacement d_A on the surface at some set of points point A from the slip between epochs t_1 and t_2 is,

$$G_A \cdot L[V(t_2,:) - V(t_1,:)]^t = d_A, \tag{2.28}$$

where G_A are the Green's functions for point A only. By writing out the definition of matrix and vector multiplication, we can see that the ith row of the matrix Equation 2.28 is,

$$\sum_j G_A(i,j) \cdot \sum_k L(j,k)[V(t_2,k) - V(t_1,k)] = d_A(i) \quad (2.29)$$

$$\sum_j G_A(i,j) \cdot \sum_k [V(t_2,k) - V(t_1,k)]L(j,k) = \quad (2.30)$$

$$[V(t_2,1)G_A(i,:), V(t_2,2)G_A(i,:), \cdots, V(t_2,r)G_A(i,:)] \cdot \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ V_r \end{pmatrix} = \quad (2.31)$$

which, as it is true for all i, is equivalent to,

$$[V(t_2,1)G_A, V(t_2,2)G_A, \cdots, V(t_2,r)G_A] \cdot \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ V_r \end{pmatrix} = d_A. \quad (2.32)$$

This means we now have the predicted displacement between t_1 and t_2 at the surface for any set of observation points A in terms of an expression linear in slip parameters with no other unknowns. We can augment the design matrix with the LHS of Equation 2.32 and augment the data vector with d_A (RHS of Equation 2.32), and we have added a sparse dataset as an additional set of equations in our linear system. If we do similar calculations for all time periods at which we have sparse data, we obtain a design matrix that takes into account every sparse dataset measurement at the same time.

In practice, we are not quite done. Because the original d have all been normalized, it's possible that the relative weight of the sparse dataset is much larger than that of the original dataset. To correct for this we multiple each dataset by a scaling factor p_k which can be different for each dataset. This gives us the freedom to emphasize any given sparse dataset as little or as much as the user desires. Also it allows for renormalization of uncertainties if the user is unsure that the uncertainties assigned to the various types of data are equivalent or if uncertainties are not given *a priori*.

Chapter 3

Practice

In this chapter wedemonstrate the general style of the code so the user can better understand the conventions we (try to) follow and the MATLAB shortcuts we used.

3.1 MATLAB Review

In this subsection we review a few of the more commonly used MATLAB funcionalities in our code. Readers familiar with cell structures and matrix indexing/manipulation/reshaping, might move to Section 3.2 directly.

There are several matrix shortcuts in MATLAB that we use over and over again in the code: `reshape`, `(:)`, and `transpose` (`transpose(A)` can also be written as `A'`).

For example, suppose we have a matrix `A = [1,2,3,4;5,6,7,8;9,10,11,12;13,14,15,16]`. Then:

```
A(:) =
     1
     5
     9
    13
     2
     6
    10
    14
     3
     7
    11
    15
     4
     8
    12
    16

reshape(A,8,2) =
     1     3
     5     7
     9    11
    13    15
     2     4
     6     8
    10    12
    14    16


reshape(A',8,2) =
     1     9
     2    10
     3    11
     4    12
     5    13
     6    14
     7    15
     8    16

A_prime = A';
A_prime(:) =
1
2
3
4
5
6
7
8
9
10
11r
12
13
14
15
16
```

The MATLAB functions `max`, `min`, `mean` are all applied to the columns of an input matrix. Thus `max(A) = [13,14,15,16]`. However, we often want to find the maximum element (minimum element, mean) of an entire matrix. In order to do this, we apply the function to `A(:)`, which is a column vector (e.g. `max(A(:)) = [16]`).

Another common convenience we use repeatedly are cell arrays. These are effectively matrices whose elements can be arbitrary variable types, as opposed to normal matrices which can only hold single-value numerical variable types such as `int`, `single`, or `double`. To access indexes of a cell array we use curly-braces (`{}`) in place of parentheses. For instance, `B = {{0,'zero'},[1,2;3,4]; 'geologist', [5;6;7]}` is a two-by-two cell array with another cell array (`0,'zero'`) in entry `B{1,1}`, a matrix (`[1,2;3,4]`) in entry `B{1,2}`, a string (`'geologist'`) in entry `B{2,1}` and finally a column vector (`[5;6;7]`) in entry `B{2,2}`. Since `B{1,1}` is itself, a cell, we can further index this entry to reach the elements of the cell-within-a-cell, as in `B{1,1}{1}` to get the numerical value `0` and `B{1,1}{2}` to get the string `'zero'`.

Our typical usage of cell arrays is to hold matrices or strings of different size that contain data about different datasets or recording stations. For example the cell array `X_dat` contains k cells, where k is the number of datasets in the scenario. Each cell

Word	Abbrev.
number, number of	n
function	fcn
time-series	tseries
calculate, calculation	calc
decomposition	decomp
scenario	scen

Table 3.1: *Abbreviations in the Code:* these are the standard abbreviations we use throughout the code.

contains the data matrix (with each row of the data matrix corresponding to a single time-series from that dataset and each column corresponding to a single epoch from that dataset) from one of these datasets. Suppose we have loaded two datasets. The first is a continuous GPS network consisting of 10 stations recording daily over 300 days, and the second is a set of three continuous strain meters recording daily over 400 days. Then `X_dat{1}` would be a 30-by-300 matrix and `X_dat{2}` would be a 3-by-400 matrix. To access the second time-series from the second dataset, we would write `X_dat{2}(2,:)`.

3.2 Naming Conventions

We replace some words with abbreviations as listed in Table 3.1. For example, the number of datasets is the variable `n_datasets` and the number of time-series is `n_tseries`.

Often the calculation of a common quantity `var_name` will be done via a function `var_name_calc`. For example, the function used to compute the total number of epochs (`n_epochs`) is called `n_epochs_calc`.

We often use the shorthand m as the number of time-series, n as the number of epochs, and N as the number of components.

Chapter 4

Tutorial – Inversion of Nias 2005 Postseismic

We assume that the user is familiar with basic MATLAB syntax, MATLAB cell arrays, and MATLAB plotting functions. If the user need a refresher, please go through MATLAB's built-in tutorials or tutorials online.

4.1 Geological Background

The Sunda trench in South East Asia has been the location of at least five major earthquakes in the last 200 years (Figure 4.1), including the December 26, 2004 Andaman-Ache Earthquake which killed almost 250,000 people.

4.2 A First Run

The only part of the code the user neesd to modify in order to run it on the user's local system is the variable `working_dir` in the file `PCAIM_driver.m`, which should be the local directory into which the user puts the PCAIM code folder. At this point, the user can execute `PCAIM_driver.m` after changing the present working directory to the folder containing `PCAIM_driver.m` (we suggest pressing F5 to do this).

Several plots will pop up describing the fit to the data, the displacement and slip associated with various components, and a cumulative slip model with the GPS position vectors.

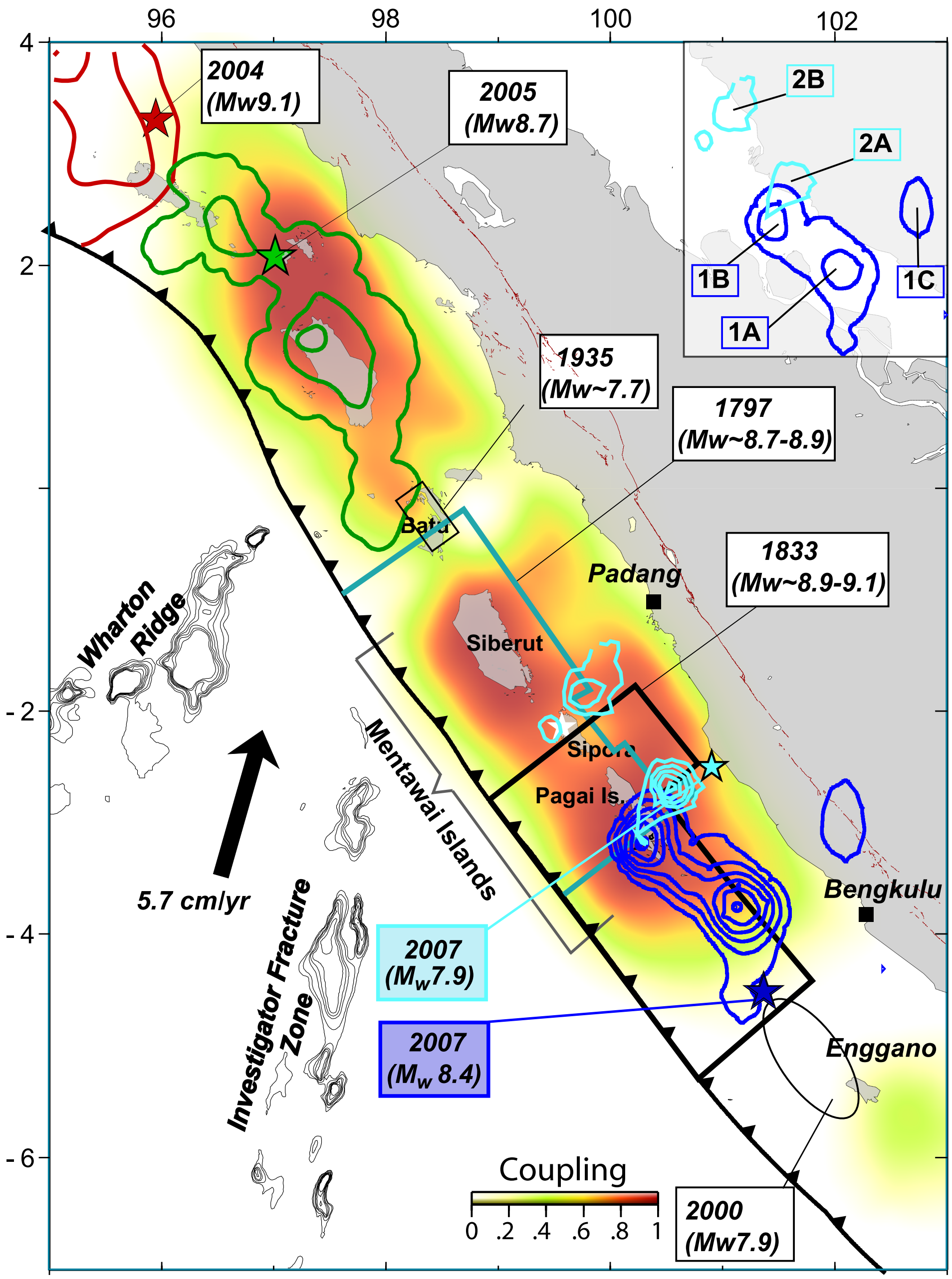

Figure 4.1: *The Sunda Trench:* The geological background for the Nias postseismic relaxation example. Adapted from [CAS+08].

4.3 Parameters to Vary

While the inversion from Section 4.2 happens to be close to the author's chosen model, there is a suite of possible models, all of which comes from the same data. Investigating this suite for reasonable end-member models as well as the user's chosen model is much of the art of inversion. In addition to the basic assumptions from section 2.2, we make assumptions about the centering of the data (section 2.3), the number of temporal and spatial functions allowed (section 2.4), aspects of the fault model (section 2.6), how to proceed with the inversion (section 2.7), and regularization of the inverse problem (section 2.7.1). In this section the user can look at how varying some of the more common parameters affects the final slip model and data fit.

4.3.1 Scenario Definition: `scen_parameters.m`

```
1)   first_epoch   = 1;
2)   last_epoch    = 450;
3)   time_unit     = 'day';
4)   sig_time      = 3;
5)   observation_unit    = 'cm';
6)   sparse_types = {'Sparse InSAR'};
7)   X_rescale = {1};
```

By default the scenario uses all the data between date "1" (Line 1) and date "450" (Line 2) in the time unit of "day" (Line 3). In this case, the dates "1" and "450" correspond to the number of days from the main shock of the Nias 2005 Earthquake. However, time could have as just as well been in decimal years or days from January 3rd, 1942. The only parameters that would have to be adjusted to achieve the same results are `first_epoch`, `last_epoch`, and `time_unit`. Epochs are defined to be identical if they agree to 3 digits (Line 4) after the decimal point of whatever the internal time unit is, and the model displacements will be given in centimeters (Line 5). Finally, any dataset with the type "Sparse InSAR" (Line 6) is declared to be temporally sparse and should not be used in construction of the time functions.

The optional parameter `X_rescale` allows rapid rescaling of the data errors, which is especially useful when there are multiple datasets and the user wishes to rescale the relative weight of the two datasets in the centering, decomposition and inversion.

Try the following changes separately and together:

1. Change the first epoch to 100 to only invert data after day 100 (inclusive).

2. Change the last epoch to 200 to only invert data before day 200 (inclusive).

3. Change the time unit to yr to convert all the input days into years. Note that this conversion is only a multiplicative factor (i.e. day d is converted to year $y = d/365.25$) and affects the meaning of `first_epoch` and `last_epoch`. For example, the values `first_epoch = 1`, `last_epoch = 450`, `time_unit = yr`, only imports data after year 1.0 and before year 450.0. Since the input data is in days after the main shock, any dates before 365.25 days after the main shock will not be imported or used.

4. Change the observation unit to m. This does not affect the program except that all measurements and predictions are 100 times smaller than if the observation unit were cm.

4.3.2 Centering: `center_parameters.m`

```
1)  n_comp_mean = 1;
2)  center_function = 'non-basic';% 'basic'
3)  iter_max = 10^5; % needs to be of the order 10^5-10^6
4)  tol = 10^(-7); % should be on the order of 10^(-7) or smaller
5)  mean_function = 'decomp_CG_means';
6)  func = 'func_mean_zero_sum_V_transform_corrected';
7)  dfunc = 'dfunc_mean_zero_sum_V_transform_corrected';
```

<FILE CONTINUES, UNNECESSARY FOR TUTORIAL>

Centering is an important part of data pre-processing for the inversion and can be somewhat delicate (Section 2.3). The number of components to be used is on Line 1, the centering function is on Line 2, the maximum number of iterations (for centering functions that use such) is on Line 3, the tolerance between iterations (for centering functions that use such) is on Line 4, the function for determining the means is on line 5 if the centering is not `basic`, and the objective function and derivative of the objective functions for the particular `mean_function` listed here are on Lines 6 and 7, respectively.

Try the following changes separately and together:

1. Change the centering function to 'basic'. This will use the weighted mean of the data instead of the mean of a `n_comp_mean`-component model, and the rest of the options are bypassed. Using 'basic' centering is much faster, but as we are missing data at several stations for a significant period of time it can give confusing results.

Compare the slip history of Patch 73 and the principal displacement fields with 'basic' and 'non-basic' centering.

2. Change the number of components for determining the data mean to 7. (For `non-basic` centering)

3. Change the maximum number of iterations to 10. Once again, this is much faster but the results can be confusing. Compare the slip history of patch 73 and the principal displacement fields between 10 and 100,000 iterations. (For `non-basic` centering)

4. Change the function tolerance to 0.01 and 10^-15. The former will suffer from being a poor model of the data, whereas the latter will take more time and give a slightly better fit to the data. (For `non-basic` centering)

4.3.3 Decomposition: `decomposition_parameters.m`

```
1) n_comp = 1;

%%%%%%%%%%%%%%
%%%% EXPECTATION MAXIMIZATION METHOD
%%%%%%%%%%%%%%

2) % decomp_fcn = 'decomp_srebro_EM_project';
3) %decomp_options = {'tol',10^(-7),'max_iter',5*10^4};

%%%%%%%%%%%%%%
%%%% SIMULTANEOUS MULTICOMPONENT CONJUGATE GRADIENT METHOD
%%%%%%%%%%%%%%

4) decomp_fcn = 'decomp_srebro_CG_simultaneous';

5) max_iter = 5*10^5;

6) tol = 10^(-15);

<FILE CONTINUES, UNNECESSARY FOR TUTORIAL>
```

These parameters affect the decomposition of the data after the centering has taken place. Line 1 is incredibly important. The model resulting from the inversions depends on the number of components used during the decomposition. The simplest model in PCAIM is a single component model, in which the slip distribution only varies in amplitude but not spatial pattern. As we increase `n_comp` to the rank of the data matrix, we fit every single small variation in data regardless of amplitude or correlation with other components. Lines 2 and 3 offer an alternative (EM) decomposition to the standard CG method we employ, so these Lines are multually exclusive with the rest of the `decomposition_parameters` file. If the user desires to invoke these options, the user should comment the rest of the file below Line 3 otherwise these selections will be overwritten (by Line 4 and one of the last lines of the file). Lines 4, 5 and 6 all specify the variable parts of the current implementation of the CG algorithm. The rest of the file calculations certain variables that are pre-calculated here for efficiency purposes and the user does not need to change them.

Try changing the following parameters:

1. Change `n_comp` to 2, 5, and 10. Notice both the increase in computation time and complexity of the resulting slip models.

2. Change `tol` to $10^{-1}, 10^{-5}, 10^{-10}$ and observe the difference in goodness-of-fit (i.e. the residual.)

3. Change `max_iter` to 10^1 while having a small `tol`. The program will likely reach the maximum number of iterations and the fit to the data will almost certainly not be satisfactory.

4. Comment all the lines after `SIMULTANEOUS MULTICOMPONENT CONJUGATE GRADIENT METHOD` and uncomment the two lines after `EXPECTATION MAXIMIZATION METHOD`. This will change the decomposition method to EM. Note that in general it is significantly slower. However, there are artificial cases which it does faster. We suggest changing back to the CG method after attempting this change because the CG algorithm is so much faster.

4.3.4 Fault Model: `model_parameters.m`

Try changing the following parameters:

1. Within `laplacian_options`, change the integer after `'n_neighbours'` to 3, 4, and 10.

2. Within `laplacian_options`, change the integer after `'free_surface_depth'` to 100, remove the string `'no_slip_points'` and remove the vector `[1:12,12:12:288,277:288]`. This will make it so all patches with a depth of less than 100 km are allowed to have arbitrary slip on them, i.e. there will be no implicit tapering of slip to zero at the edges.

3. Reset `laplacian_options` to what it started as, and change the vector `[1:12,12:12:288,277:288]` to `[1:12,12:12:288,277:288, 73:76]`. This changes which points on the fault plane have significant penalties on them having slip. In particular in in this case, we put a very large penalty on slip at patches 73 through 76, in the middle of the main superficial afterslip patch in addition to the non-surface edge patches already penalized.

4.3.5 Inversion: `inversion_parameters.m`

Try changing the following parameters:

1. Change the weight of the Laplacian, `lap_weight`. by power by 10 between 10^{-4} and 10^{4}. This is one of the most subjective parts of the code and requires that the user make judgments calls as to the correct regularization weight. We suggest the user try many, many different `lap_weight` values before deciding on one.

2. Set

```
invert_options = {...
    'FixedRake',rake,...
    };
```

This chapter has outlined common changes to the major parameters affecting the inversion results. We strongly encourage the user to play with these and the plotting functions until the user becomes comfortable with the results of the various parameter changes. We can almost guarantee that the first inversion attempted will not be the best inversion geophysically or statistically.

Chapter 5

Tutorial – Loading New cGPS2/cGPS3 Datasets

We go through the process of changing the dataset with which the PCAIM code is working step by step for the user. For clarity we will everywhere use cGPS3 as the data type, but everything here will work equally well for cGPS2 data except that any uses of `cGPS3` must be replaced, of course, by `cGPS2`.

5.1 Setup

1. Make a new folder somewhere on the user's hard driver for the files associated with the user's new dataset. We will call this the *scenario folder*.

2. Make a copy of `load_scenario_information_Nias.m` in the same directory and rename the copy to something about the user's scenario (e.g. if the user is working on Pisco, `load_scenario_information_Pisco.m` is a good name).

3. Replace `scen_name`'s assignment with something that makes sense for the user's scenario.

4. Replace `scen_dir` with the relative path from the `PCAIM Code` folder or full path of the user's scenario. If the user's scenario folder is in the same directory as `PCAIM_driver.m` and has the same name as `scen_name`, then the user does not have to change this assignment.

5. Copy the originals of seven other files listed below the assignment of `scen_dir` into the user's scenario folder and change their names as the user feels appropriate (e.g. change `Nias_data_input_file` to `Pisco_data_input_file` if the user is studying Pisco). The seven files are:

   a) `Nias_data_input_file`
   b) `scen_parameters`,
   c) `center_parameters`,
   d) `decomposition_parameters`,
   e) `model_parameters`,
   f) `inversion_parameters`, and
   g) `plotting_commands`.

Next we're going to edit three of these seven files and their internally referenced files. This is all we have to do in order to change scenarios!

5.1.1 data_file

1. Decide where to put the user's data folder and edit the `data_input_file` according to the specifications in Section 8.1.1. This should consist of one line and give the basic information stating we want to load a GPS dataset and where to find a file containing all the station information.

2. Create a cGPS3 information file in the location specified by the user's data list file according to the specifications in Section 8.1.2, or copy the file `gps_stations.dat` from the Nias folder and edit the lines to the values appropriate for the user's cGPS3 dataset. We suggest doing the latter and doing a find-and-replace for most of the paths.

3. Format the user's data files according to the specifications in Section 8.1.2.1 and place them in files with names agreeing with the cGPS3 information file.

5.1.2 scen_parameters_file

1. Change `first_epoch` and `last_epoch` to agree with the first and last epoch of the user's dataset the user wishes to consider. If the user is have problems loading the first or last data point, decrease `first_epoch`/increase `last_epoch` a small amount.

2. Change `time_unit` to the time unit (e.g. `yr` for year or `day` for days).

3. Change `sig_time`, which is how many decimal places to keep when rounding dates dates to determine if epochs are the same or different. For example, epochs 1999.154 and 1999.153 would be the same epoch for any integer `sig_time` ≤ 2, but they would be considered different epochs for any integer `sig_time` ≥ 3.

4. Change `observation_unit` to be whatever the user wants the internal computations to be done in. If the user's dataset unit is different, the dataset will be automatically converted.

5. Do not worry about `sparse_types` for this tutorial, but this string should agree with whatever the data type (e.g. cGPS3) is that the user wishes to consider temporally sparse.

5.1.3 `center_parameters_file`

The user need not change anything here to check if the import was successful.

5.1.4 `decomposition_parameters_file`

The user need not change anything here to check if the import was successful.

5.1.5 `model_parameters_file`

I assume the user already has a fault model (either rectangular patches or point sources) constructed, but we do not assume the user has the Green's function for the user's fault model.

1. Assign the *system-readable* path of the Green's function Fortran binaries to `GreensExternalFcnDir`. In other words, on Linux/Mac systems there must be a backslash before spaces, etc.

2. Assign the overall tectonic motion angle in the horizontal plane (in degrees counterclockwise from East) to `ang_tect`. If the user's fault has vertical (dip = 90°) patches, then the user should change `vect_tect=[cosd(ang_tect);sind(ang_tect);0]` to a unit vector in three-space so that a principal rake direction on these surfaces is defined.

3. Assign the path to the user's fault description file (following the conventions in either of Sections 8.2.1.1 or 8.2.1.2) in the cell after the string `'LoadFaultModel'`.

4. Assign the path to the user's origin description file (following the conventions in Section 8.2.2) in the cell after `'Origin'`.

5. If the user's fault is composed of rectangular patches, leave the string `'RectangleFault'`. If the user's fault is composed of triangular patches, delete the string `'RectangleFault'`.

5.1.6 `inversion_parameters_file`

The user need not change anything here to check if the import was successful. However, the user may want to adjust `lap_weight` to get a better initial model.

5.1.7 `plotting_commands_file`

The user need not change anything here to check if the import was successful, though it may help to download and use a coast file for visualization purposes (see Appendix A).

Now the user's code should run through the whole algorithm and produce slip models (albeit, probably rather low quality ones that may or may not fit the data well) if everything was entered correctly!

To hone the model to something geophysically reasonable that also fits the data, the user should adjust the various parameters listed in Chapter 4.

5.2 The Art of Inversion

This subject is too small for a single section, but we'll list some pointers and general debugging ideas for when the user becomes stuck something isn't working correctly. The author has come across many such helpful hints in the quality time he has spent with the PCAIM code.

5.2.1 Strategic Approach

- Save after centering. Centering can take a long time to do right on large datasets. If the user stops the inversion after centering and saves the current state of the program, the user can load and run the script from this point instead of re-centering the dataset.

- Start small. Try a one-component model with an unrestricted inversion at first. This will take less time to run and will be easier to evaluate since a single time function and spatial function describe the entire scenario. The user does not even have to watch a movie of the slip history to understand the slip evolution on the fault.
- Write scripts to compute the user's "standard plots" and "standard output variables", and place this script at the end of `PCAIM_driver.m`. This makes viewing and calculation these automatic and saves the user time.
- Open any of parameters files by using the `open` command to the right of the comment character % on the line where the parameter file is `run`.

5.2.2 When things don't work

Most of the time something will go wrong with the inversion scenario. Here are some general troubleshooting suggestions that may help.

- `clear all`, `close all`, restart the script from the last point at which it seemed to be working correctly.
- Test end cases, such as `gamma = 10^-10` or `gamma = 10^10`. If these don't work out as expected then there's a problem with a variable or the program.
- Check the signs of the user's data, fault model, etc.. Having fault patches above the surface or datasets implying movement in opposite directions will give poor results compared to what the user expects.
- Look at the difference between the model and data both as a time-series and as a spatial distribution. What looks good in one may not look as good in the other.
- Look at the predictions at epochs/locations where the user does not have data. If the predictions are very far off from what is expected at that location, something may be wrong with the centering, decomposition or inversion.
- Use the "click-to-get-a-red-dot" breakpoints and internal debugger of MATLAB. Note that `clear all` removes all breakpoints. See `dbcont`, `dbstep`, `dbquit` in MATLAB's help.

Chapter 6

Checklist – Adding a new type of data

Adding a new type of data for inversions is more involved than loading the user's own dataset of a different type. However, it has been successfully done on old versions of the code by at least three different users so far.

Scripts the user needs to edit:

`scen_parameters.m`: If the new data type is temporally sparse, then the user needs to add the data type to the cell of strings `sparse_types`.

`load_all_data.m`: add a case to the switch statement corresponding to the user's data type. Also the user must write a `load_<DATATYPE>_data` function to load the user's data. For consistency, we preferred to use the same style as cGPS2, cGPS3 and InSAR

Warning: If the user has a spatially continuous datasets (e.g. InSAR) that is also temporally dense, then the user may need to write a new centering algorithm. The current one assumes that there is one mean per time-series to be removed, which is not true for some spatially continuous dataets. Otherwise, there is nothing in the centering the user needs to change.

Green's Functions: The user will need to provide the user's own Green's functions or write a function to calculate the Green's functions from the current script setup. For example, strain meters have Green's functions that are projections of the direction of slip on the strike-slip, dip-slip components of slip for the patch which they measure. This Green's function is not currently implemented, so the code itself would have to be changed to check for data type "Strain Meter" and deal differently with those time-series. Most likely this could be done most easily by putting additional items into `get_fault_model_options` (inside the `model_parameters` file) that are parsed within `get_fault_model`.

Similarly, the user will need to write cases for the new data type in `project_all_greens_fcn`.

`create_predictions`: The user may need to change this script to properly calculate the predictions and model of the user's data.

Plotting: The user may need to add functionality to the plotting scripts to allow a proper display of the user's dataset or to answer the user's geophysical questions.

Chapter 7

Comprehensive Guide to Options

This chapter describes all of the valid options the user can set in the various parameter files.

7.1 load_scenario_information

There are 8 variables to be set in `load_scenario_information`. The examples are from the Nias 2005 Postseismic scenario.

1. `scen_name`. This is a colloquial name that describes the scenario. E.g.:

   ```
   scen_name                    = 'Nias 2005 Postseismic';
   ```

2. `scen_dir`. This is the directory in which all the information for the scenario is expected to be located. E.g.:

   ```
   scen_dir                     = [scen_name];
   ```

3. `data_file`. This is the file in which all datasets for the scenario are listed. E.g.:

   ```
   data_file                    = [scen_dir,'Nias_data_input_file'];
   ```

4. `scen_parameters_file`. This is the file in which all the scenario parameters are set. E.g.:

```
scen_parameters_file          = [scen_dir,'scen_parameters'];
```

5. `center_parameters_file`. This is the file in which all parameters are set for the centering step of the algorithm. E.g.:

```
center_parameters_file        = [scen_dir,'center_parameters'];
```

6. `decomposition_parameters_file`. This is the file in which all parameters are set for the decomposition step of the algorithm. E.g.:

```
decomposition_parameters_file = [scen_dir,'decomposition_parameters'];
```

7. `model_parameters_file`. This is the file in which all parameters are set for the fault model step of the algorithm. E.g.:

```
model_parameters_file         = [scen_dir,'model_parameters'];
```

8. `inversion_parameters_file`. This is the file in which all parameters are set for the inversion step of the algorithm. E.g.:

```
inversion_parameters_file     = [scen_dir,'inversion_parameters'];
```

7.2 data_file

The format and valid input for `data_file` is located in Section 8.1.1.

7.3 scen_parameters_file

There are 6-7 variables to be set within `scen_parameters_file`. The examples are from the Nias 2005 Postseismic scenario.

1. `first_epoch`. This is the first epoch during which data is to be allowed, inclusive. E.g. to only import data with epoch of 1 or afterward the user would write:

```
first_epoch   = 1;
```

2. `last_epoch`. This is the last epoch during which data is to be allowed, inclusive. E.g. to only import data with epoch of 450 or before the user would write:

```
last_epoch    = 450;
```

3. `time_unit`. This is the time unit to be used internally by MATLAB. Valid time units are listed in Table 8.2 E.g. to tell MATLAB to use days as the internal time unit, the user would write:

   ```
   time_unit     = 'day';
   ```

4. `sig_time`. This is the number of significant digits after the decimal place assumed MATLAB for comparing epochs. This should be an integer. E.g. to tell MATLAB to use three digits after the decimal point to compare epochs, the user would write:

   ```
   sig_time     = 3;
   ```

5. `observation_unit`. This is the distance unit to be used internally by MATLAB. Valid distance units are listed in Table 8.2 E.g. to tell MATLAB to use centimeters as the internal distance unit, the user would write:

   ```
   observation_unit     = 'cm';
   ```

6. `sparse_types`. This is a cell array containing strings corresponding to the sparse data sources. E.g. to tell MATLAB to use InSAR data as sparse data, the user would write:

   ```
   sparse_types = {'InSAR'};
   ```

7. `X_rescale`. This is an optional cell array that contains rescaling factors for each datasets. These rescaling factors can take one of two forms for each dataset. The first form is a real constant (e.g. 7.235) that rescales all the data in the corresponding dataset by that constant. This is primarily used to adjust the weight imposed during the inversion on one or both datasets. For example, if we wish to decrease the weight on an InSAR dataset by a factor of 10, we could change its entry in `X_rescale` from 1 to 0.1. The second usage of `X_rescale` is when, for some k, `size(X_rescale{k}) == size(X_dat{k})`. In this case, for all valid i, j, we rescale `X_weight{k}(i,j)` by `X_rescale{k}(i,j)`. A final possible value (which is also the default) for `X_rescale` is the empty cell .

   For example, we we had two datasets, the first is a short GPS time series consisting of 3 time-series over 10 epochs and the second is a large set of InSAR images, then all of the following would be valid assignments of `X_rescale`:

- `X_rescale = {}`. This gives both datasets their original weight.
- `X_rescale = {1,1}`. This gives both datasets their original weight.
- `X_rescale = {1,1}`. This gives both datasets their original weight.
- `X_rescale = {2,0.5}`. This gives the GPS time-series twice the normal weight and the InSAR images half the usual weight.
- `X_rescale = {[1,1,1,1;1,1,1,1;0,1,1,1],1}`. This gives both datasets their original weight except for the first epoch of the last time-series for the GPS dataset, which is rescaled to having zero weight.

7.4 center_parameters_file

Currently there are 2 methods of centering, `basic` and `advanced`. As these have different options, we deal with them separately. The user specifies which type of centering to use through the definition of the variable `center_function`, which can either take the value `basic` or `advanced`. In practice, we string compare against `'basic'`, so any string other than `basic` (e.g. `non-basic`, `advanced`, `moose`, etc.) for this variable will use the advanced method of centering.

The variable to set to determine this choice is `center_function`. E.g.

```
center_function = 'advanced';
```

7.4.1 basic centering

Basic centering finds and subtracts the weighted mean from all dense time-series.

No options are used, so no more variables need to be defined.

7.4.2 advanced centering

Advanced centering uses the CG algorithm to find a local minimum of the mean values for each dense time-series with a `n_comp_mean` component model. The user must also specify `mean_function`, the function to-be-used for determining the mean; and `mean_options`, the options for the mean function to use. As these options depend on the `mean_function`, we will define them in Section 7.4.2.1. E.g.

```
mean_function = 'decomp_CG_means';
n_comp_mean = 1;
```

7.4.2.1 decomp_CG_means

For the advanced centering function `decomp_CG_means`, which is currently the only advanced centering algorithm, the mean options must include 12 arguments:

1. The string `'func'` followed by the name of an .m-file that computes the objective function the user wishes to minimize. E.g.:

   ```
   func = 'func_mean_zero_sum_V_transform_corrected';
   ```

2. The string `'dfunc'` followed by the name of an .m-file that computes the gradient of objective function the user wishes to minimize.

   ```
   dfunc = 'dfunc_mean_zero_sum_V_transform_corrected';
   ```

3. The string `'iter_max'` followed by a positive integer gives the number of iterations the CG algorithm goes through before stopping unless convergence is reached earlier.

   ```
   iter_max = 10^5;
   ```

4. The string `'tol'` followed by a positive double gives the maximum function difference between two iterations of the CG algorithm such that convergence is assumed to be achieved.

   ```
   tol = 10^(-7);
   ```

5. The string `'func_options'` followed by a cell array (definitions of each entry to be found in Section 10) that contains the options for the function `func`.

   ```
   func_options = {...
       X_dat,...      1
       X_weight,...   2
       u_index,...    3
       v_index,...    4
       n_datasets,... 5
       means_index,...6
       v_end_entry,...7
       v_index_size,...8
   ```

```
    n_epochs,...     9
    n_comp_mean,...     10
    transformation_matrix_inv,... 11
    v_index_in_x... 12
    };
```

Note that the numbers after the three dots are comments denoting the cell number.

6. The string `'dfunc_options'` followed by a cell array (definitions of each entry to be found in Section 10) that contains the options for the function `dfunc`.

```
dfunc_options = {...
    X_dat,...       1
    X_weight,...    2
    u_index,...     3
    v_index,...     4
    n_datasets,...  5
    means_index,... 6
    v_end_entry,... 7
    v_index_size,...8
    X_row,...       9
    n_comp_mean,...10
    n_tseries,...  11
    n_epochs,...   12
    X_time_index,...13
    transformation_matrix,...   14
    transformation_matrix_inv,...15
    v_index_in_x,...16
    };
```

Note that the numbers after the three dots are comments denoting the cell number.

`mean_options` as a whole is thus given by:

```
mean_options = {...
    'func',func,...
    'dfunc',dfunc,...
    'iter_max',iter_max,...
    'tol',tol,...
```

```
'func_options',func_options,...
'dfunc_options',dfunc_options ...
};
```

7.5 decomposition_parameters_file

The 3 variables that need to be defined within this script are:

1. `n_comp`. This is the number of linear components to be used in the decomposition. It should be a positive integer.

   ```
   n_comp = 2;
   ```

2. `decomp_fcn`. This is the function that will be used for the decomposition. The two options are `decomp_srebro_CG_simultaneous` and `decomp_srebro_EM`, which are described in Sections 7.5.2 and 7.5.1 respectively. E.g.:

   ```
   decomp_fcn = 'decomp_srebro_CG_simultaneous';
   ```

3. `decomp_options`. This is a cell object whose entries depend on the `decomp_fcn`, so will be dealt with in Sections 7.5.2 and 7.5.1.

7.5.1 Conjugate Gradient – Simultaneous

We strongly recommend the use to use the Conjugate Gradient algorithm instead of the Expectation Maximization algorithm because it is much faster for every case we've tested. The variables in the options that need to be defined are:

1. The string `'func'` followed by the name of an .m-file that computes the objective function the user wishes to minimize. E.g.:

   ```
   func = 'func_multi_component';
   ```

2. The string `'dfunc'` followed by the name of an .m-file that computes the gradient of objective function the user wishes to minimize.

   ```
   dfunc = 'dfunc_multi_component';
   ```

3. The string `'iter_max'` followed by a positive integer gives the number of iterations the CG algorithm goes through before stopping unless convergence is reached earlier.

   ```
   iter_max = 10^5;
   ```

4. The string `'tol'` followed by a positive double gives the maximum function difference between two iterations of the CG algorithm such that convergence is assumed to be achieved.

   ```
   tol = 10^(-7);
   ```

5. The string `'func_options'` followed by a cell array (definitions of each entry to be found in Section 10) that contains the options for the function `func`.

   ```
   func_options = {...
       X_dat,...    1
       X_weight,... 2
       u_index,...  3
       v_index,...  4
       n_datasets...5
       };
   ```

   Note that the numbers after the three dots are comments denoting the cell number.

6. The string `'dfunc_options'` followed by a cell array (definitions of each entry to be found in Section 10) that contains the options for the function `dfunc`.

   ```
   dfunc_options = {...
       X_dat,...       1
       X_weight,...    2
       u_index,...     3
       v_index,...     4
       n_datasets,...  5
       X_row,...       6
       n_comp,...      7
       n_tseries,...   8
       n_epochs,...    9
   ```

```
X_time_index...10
};
```

Note that the numbers after the three dots are comments denoting the cell number.

decomp_options for the CG algorithm is given by:

```
decomp_options = {...
    'func',func,...
    'dfunc',dfunc,...
    'iter_max',iter_max,...
    'tol',tol,...
    'func_options',func_options,...
    'dfunc_options',dfunc_options ...
    };
```

1.

7.5.2 Expectation Maximization

We strongly recommend the use to use the Conjugate Gradient algorithm instead of the Expectation Maximization algorithm because it is much faster for every case we've tested.

The 4 inputs that need to be defined within decomp_options are:

1. 'tol',tol, that is, the string 'tol' followed by the tolerance for the linear decomposition function. E.g.:

   ```
   tol = 10^(-12);
   ```

2. 'max_iter',max_iter, that is, the string 'max_iter' followed by the maximum number of iterations for the EM algorithm. E.g.:

   ```
   max_iter = 5*10^4;
   ```

E.g.:

```
decomp_options = {'tol',tol,'max_iter',max_iter};
```

7.6 model_parameters_file

This script is the most maleable set of options in the code. Three classes of options are avaliable for this script:

1. Building a fault model
2. Determination of Laplacian/regularization terms
3. Green's functions

All of these options need to be put into a single cell array `get_fault_model_options` separated by commas.

7.6.1 Fault Model

The user must either provide a list of points from which to build a point-source fault model using the internal algorithms described in Section 2.6, or the user must give a file of fault elements (described in Section 8.2) and tell the program if the fault elements are rectangular.

- The user must either enter the string `'BuildFaultModel'` followed by the cell `fault_model_parameters_constrct` XOR `'LoadFaultModel'` followed by `load_fault_model_file`. The format for `load_fault_model_file` is given in Section 8.2.1 and is different for rectangular and point source fault elements. For building the fault model, `fault_model_parameters_constrct` has 10 entries, examples of which are below:

```
'Bhuj 2001 Postseismic/faultmodel/fault_points.par',... %input_file
        82 ... % strike angle = fault_model_parameters{2};
        'Bhuj 2001 Postseismic/faultmodel/bhuj_fault_model.trg'...%outputfile_pcaim
        'Bhuj 2001 Postseismic/faultmodel/dislo.trg'...%outputfile_okada
        10^1 ...%smooth_param  = fault_model_parameters{5};
        10 ...%nx            = fault_model_parameters{6};
        10 ...%ny            = fault_model_parameters{7};
        200 ...%ang_tect      = fault_model_parameters{8};
        'v4' ...%interp_method = fault_model_parameters{9};
        3 ... %N_nearest     = fault_model_parameters{10};
```

  `input_file` is the relative path to a list of points in the local coordinates in the format from Section 8.2.3, the `strike angle` is an approximately average strike angle for the fault surface the user wishes to build (for the purposes of resampling axes), the `outputfile_pcaim` is the output file for PCAIM to use internally, `outputfile_okada` is the output file for the Fortran Okada scripts to use, `smooth_param` is the weight of the smoothing parameter, `nx` and `ny` are the approximate number of patches to be made along the strike-slip and dip-slip directions respectively, `ang_tect` is the approximate overall tectonic angle, `interp_method` is the interpolation method used by the resampling algorithm, and `N_nearest` is the number of nearest neighbors to sample for construction of the discrete Laplacian for smoothing the fault surface.

- The user must enter the string `'Origin'` followed by `origin_file`, whose format is given in Section 8.2.2.
- If the fault is composed of rectangular elements, the user must enter the string `'RectangleFault'`
- The user must also include the string `'tect_vect'` followed by a three vector that describes the overall tectonic motion for the area. This is used in the rake calculations.

7.6.2 Laplacian/Regularization

Thse user must either provide a set of parameters defining how to compute the Laplacian (format given in the follow paragraphs) or a file in which a Laplacian (or other regularlization matrix) is located. The format for this latter file is in Section 8.2.4.

In order to load in a Laplacian, we need to have `'BuildLaplacian',laplacian_options` in our `get_fault_model_options`, or we need to have `'LoadLaplacian',laplacian_file` there. `laplacian_file` is the path to a file containing the Laplacian to-be-used. `laplacian_options` is more complicated and the elements of this cell array are listed below.

- `'n_neighbours',n_neighbors`. This option tells the Laplacian how many neighboring points (an integer) to use in its approximation of the Laplacian. We recommend somewhere between 4 and 10. Try various numbers and see how the results change.
- `'no_slip_points', no_slip_points` . This option specifies a number of fault elements (integers in a vector) that have heavy penalties applied to any slip on them. This is useful for forcing slip to go to zero at the fault boundaries. This option overrides `'free_surface_depth'`.

- `'free_surface_depth', free_surface_depth`. This option gives a depth such that:

  1. For any patch whose center is above `free_surface_depth`, there are no penalties on slip and the standard Laplacian is used.
  2. For any patch who center is below `free_surface_depth`, slip is heavily penalized.

  This option is overridden by `'no_slip_points'`.

- `'projected'`. This string should be included if the user wants the algorithm to guess the edge patches (on which slip should be reduced to zero) on the fault using a slight modification of the MATLAB convex hull algorithm.

- `'scaling_edge_factor', scaling_edge_factor` is the multiplicative factor (double) of a standard distance between patches that is used to guess whether a given patch is on the edge or not.

- `'strike_angle', strike`. This option gives the strike angle for each patch.

- `'dip_angle', dip`. This option gives the dip angle for each patch.

7.6.3 Green's Functions

The user can either build or load the Green's functions for the inversion routine. If the Green's functions are to be loaded, the user should follow the formatting description in Section 8.2.6. If the Green's functions are to be constructed, the user must provide the *system-readable* path of the Green's function Fortran binaries, and include in `get_fault_model_options` the three options `'BuildGreensFunction', all_position, GreensExternalFcnDir` in that exact order. Note that `all_position` has been defined in earlier calculations.

7.7 inversion_parameters_file

There are only two required variables in `inversion_parameters_file`, `lap_weight` and `invert_options`.

`lap_weight` is the linear weight applied to the Laplacian in order to change the strength of the regularization term in the inversion (Sections 2.6.2, 2.7.1).

As with all the other options sets, `invert_options` can contain a number of options to describe how to perform the inversion, or it can be left blank to use the default options.

- `'Positivity'`. This option forces positive slip at depth with one component.
- `'Fixed Rake', rake`. This option forces slip on every patch to lie in the one-dimensional subspace defined by `rake` (the rake angle) for that patch.
- `'PseudoInverse'`. This option makes the inversion algorithm use the pseudo inversion (`A_inv = pinv(A); s = A_inv * d;`) instead of MATLAB's build-in backslash operator as the default(`s = A\d ;`).
- `'SparseConstraint', SparseConstraint`. This option is used to include an InSAR image (or other sparse constraint matrix constructed in `project_all_greens_fcn`) in the inversion. See Section 2.7.2 for details.
- `'SparseWeight', SparseWeight`. This option allows the user to change the weight the sparse constraint has compared to every other datasource.
- `'Sparse_d', Sparse_d`. This option allows the input of the sparse data vector that complements the `SparseConstraint` during the inversion. See Section 2.7.2 for details.
- `'NoSmoothing'`. This option removes the regularization via the Laplacian. This option may not work unless `'PseudoInverse'` is also used.

7.8 plotting_commands_file

This file is not as well-documented or structured because it is assumed the user will heavily customize the plotting functions to the user's own purposes and aesthetic desires.

The variables that may usefully be defined for many of the plotting functions are:

1. `coast_file_name`, which is the path to a MATLAB-readable coast file (e.g. `'Nias 2005 Postseismic/Nias_Coast.dat'`).
2. `AZ,EL`, which are the azimuth and elevation of the 3D viewing angle.

Chapter 8

File Conventions

8.1 Data Input

8.1.1 List of Data Inputs

For each dataset the user must list the dataset information file in which to find more information about the dataset. Format for each type of data in the following subsections. Each data source will be in some directory <data_root> that is set by the user.

Default Location:<data_root>/<scenario_name>_data_input_file

Format: No header. Must be 5 columns. Vertical bar "|" separated columns. Leading and trailing white space on the entries will be removed.

(1)	(2)	(3)	(4)	(5)
Name (1) |	Type (1) |	Path (1) |	Time Unit (1) |	Distance Unit (1)
Name (2) |	Type (2) |	Path (2) |	Time Unit (2) |	Distance Unit (2)
⋮	⋮	⋮		
Name (M) |	Type (M) |	Path (M) |	Time Unit (M) |	Distance Unit (M)

In the preceding table, M is the number of data sets to be loaded, Name is the colloquial name of the dataset, Type is one of the acceptable data types (Table 8.1), Path is the absolute path or relative path to the information file for that dataset,Time Unit is one of the value time units (Table 8.2), and Distance Unit is one of the valid distance units (Table 8.2).

Example:

```
Nias 2005 GPS Postseismic | cGPS3 | Nias 2005 Postseismic/Nias_cGPS3_stations.dat | day | cm
```

Data Types
cGPS3
cGPS2
InSAR

Table 8.1: Acceptable data types for List of Data Inputs from Section 8.1.1.

.

Time Unit	Time Code
decimal years	yr
decimal months	month
decimal days	day
decimal hours	hour
decimal minutes	min
decimal seconds	sec

Length Unit	Length Code
decimal meters	m
decimal centimeters	cm
decimal milimeters	mm

Table 8.2: Acceptable units for List of Data Inputs from Section 8.1.1.

8.1.2 Dataset information File – cGPS3

For each cGPS3 station, the user must give the path to the specific data file. Format for cGPS3 data files is in section 8.1.2.1.

Default Location:<data_root>/<scenario_name>_cGPS3_stations.dat

Format: No header. Must be 4 columns. Vertical bar "|" separated columns. Leading and trailing white space on the entries will be removed.

(1)	(2)	(3)	(4)
Name (1) |	Path (1) |	Longitude (1) |	Latitude (1)
Name (2) |	Path (2) |	Longitude (2) |	Latitude (2)
⋮	⋮	⋮	⋮
Name (M) |	Path (M) |	Longitude (M) |	Latitude (M)

In the preceding table, M is the number of data sets to be loaded, Name is the colloquial name of the station, Directory is the absolute path or relative path to the data file for that station, Longitude is the longitude of the station in decimal degrees, and Latitude is the latitude of the station in decimal degrees.

Example:[1]

[1] Longitudes/Latitudes have been truncated for readability.

```
BITI | Nias 2005 Postseismic/Nias_cGPS3_timeseries/BITI  | 97.81 | 1.08
BSIM | Nias 2005 Postseismic/Nias_cGPS3_timeseries/BSIM  | 96.33 | 2.41
BTHL | Nias 2005 Postseismic/Nias_cGPS3_timeseries/BTHL  | 97.71 | 0.57
LEWK | Nias 2005 Postseismic/Nias_cGPS3_timeseries/LEWK  | 95.80 | 2.92
LHWA | Nias 2005 Postseismic/Nias_cGPS3_timeseries/LHWA  | 97.13 | 1.38
PBAI | Nias 2005 Postseismic/Nias_cGPS3_timeseries/PBAI  | 98.53 | -0.03
PBLI | Nias 2005 Postseismic/Nias_cGPS3_timeseries/PBLI  | 97.41 | 2.31
PSMK | Nias 2005 Postseismic/Nias_cGPS3_timeseries/PSMK  | 97.86 | -0.09
PTLO | Nias 2005 Postseismic/Nias_cGPS3_timeseries/PTLO  | 98.28 | -0.05
SAMP | Nias 2005 Postseismic/Nias_cGPS3_timeseries/SAMP  | 98.71 | 3.62
```

8.1.2.1 Data File – cGPS3

Default Location: `<data_root>/<scenario_name>_cGPS3_timeseries/<station_name>`, where `<station_name>` is the name of a given station. For example, the station MKMK in the SuGAR network in a Nias scenario should be located at `<data_root>/Nias_cGPS3_timeseries/MKMK`. If the user has multiple cGPS3 datasets for this scenario, append a short string of characters to identify each (e.g. add SuGAR if they are SuGAR network stations, IRD if the stations belong to the IRD, etc.)

Format: No header. Must be 7 columns. White space separated columns.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
Epoch(1)	$d_E(1)$	$d_N(1)$	$d_U(1)$	$\sigma_E(1)$	$\sigma_N(1)$	$\sigma_U(1)$
Epoch(2)	$d_E(2)$	$d_N(2)$	$d_U(2)$	$\sigma_E(2)$	$\sigma_N(2)$	$\sigma_U(2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Epoch(M)	$d_E(M)$	$d_N(M)$	$d_U(M)$	$\sigma_E(M)$	$\sigma_N(M)$	$\sigma_U(M)$

In the preceding table, M is the number of cGPS3 stations from this dataset to be loaded, Epoch is epoch of the measurement, d_E is the displacement in the East direction, d_N is the displacement in the North direction, d_U is the displacement in the Up direction, σ_E is 1-σ uncertainty on d_E, σ_N is 1-σ uncertainty on d_N, and σ_U is 1-σ uncertainty on d_U.

Example:[2]

```
1.5     -5      5      10      0.7     0.3     2.2
2.48    -5.1    4.1    10.2    1.2     0.7     3.8
3.51    -5.5    5      11.8    1       0.7     3
4.49    -5.5    4.3    10.3    0.6     0.3     1.6
9.5     -6      2.7    11.1    1.8     1.6     6.1
```

[2]From `Nias 2005 Postseismic/Nias_cGPS3_timeseries/LHWA`

```
...
...
331.72    -16        -9.2     15.5     0.6     0.2     1.2
332.71    -15.5      -9.6     14.2     0.6     0.3     1.4
333.73    -15.7     -11.1     15.4     0.5     0.1     1
```

8.1.3 Dataset information File – cGPS2

For each cGPS2 station, the user must give the path to the specific data file. Format for cGPS2 data files is in section 8.1.3.1.

Default Location:`<data_root>/<scenario_name>_cGPS2_stations.dat`

Format: No header. Must be 4 columns. Vertical bar "|" separated columns. Leading and trailing white space on the entries will be removed.

(1)	(2)	(2)	(4)
Name (1) |	Path (1) |	Longitude (1) |	Latitude (1)
Name (2) |	Path (2) |	Longitude (2) |	Latitude (2)
⋮	⋮	⋮	⋮
Name (M) |	Path (M) |	Longitude (M) |	Latitude (M)

In the preceding table, M is the number of data sets to be loaded, Name is the colloquial name of the station, Directory is the absolute path or relative path to the data file for that station, Longitude is the longitude of the station in decimal degrees, and Latitude is the latitude of the station in decimal degrees.

Example:

```
BHUJ | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/BHUJ | 69.65 | 23.25
BIRN | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/BIRN | 69.71 | 23.66
DHAM | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/DHAM | 70.14 | 23.33
GAND | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/GAND | 70.10 | 23.07
HAJP | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/HAJP | 69.21 | 23.69
LODA | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/LODA | 69.89 | 23.39
MAND | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/MAND | 69.35 | 22.83
NAKA | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/NAKA | 69.29 | 23.36
NALI | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/NALI | 68.84 | 23.26
NARA | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/NARA | 68.54 | 23.68
RAJK | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/RAJK | 70.74 | 22.29
RAPR | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/RAPR | 70.64 | 23.57
RATN | Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/RATN | 70.36 | 23.86
```

8.1.3.1 Data File – cGPS2

Default Location: `<data_root>/<scenario_name>_cGPS2_timeseries/<station_name>`, where `<station_name>` is the name of a given station. For example, the station RATN from [CBR+09] should be located at `<data_root>/Bhuj_cGPS2_timeseries/RATN`. If the user has multiple cGPS2 datasets for this scenario, append a short string of characters to identify each (e.g. add SuGAR if they are SuGAR network stations, IRD if the stations belong to the IRD, etc.)

Format: No header. Must be 5 columns. White space separated columns.

(1)	(2)	(2)	(4)	(5)
Epoch(1)	$d_E(1)$	$d_N(1)$	$\sigma_E(1)$	$\sigma_N(1)$
Epoch(2)	$d_E(2)$	$d_N(2)$	$\sigma_E(2)$	$\sigma_N(2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Epoch(M)	$d_E(M)$	$d_N(M)$	$\sigma_E(M)$	$\sigma_N(M)$

In the preceding table, M is the number of cGPS2 stations from this dataset to be loaded, Epoch is epoch of the measurement, d_E is the displacement in the East direction, d_N is the displacement in the North direction, σ_E is 1-σ uncertainty on d_E, and σ_N is 1-σ uncertainty on d_N.

Example:[3]

```
 87.66      -1.79      -1.91       1.2       2.4
259.33      -5.02      -6.17       1.8       4.1
379.86      -6.25      -7.79       1.8       4.6
606.32      -7.84      -9.88       1.1       3.1
2165.9     -12.29     -15.74       2.1       3.9
```

8.1.4 Dataset information File – InSAR

For each InSAR image, the user must give the path to the specific data file. Format for InSAR data files is in section 8.1.4.1. Format for InSAR LOS files is in section 8.1.4.2.

Default Location:`<data_root>/<scenario_name>_InSAR.dat`

Format: No header. Must be 4 columns. Vertical bar "|" separated columns. Leading and trailing white space on the entries will be removed.

[3]From `Bhuj 2001 Postseismic/Bhuj_cGPS2_timeseries/RATN`

(1)	(2)	(3)	(4)
Name (1) |	Data Path (1) |	LOS Path (1) |	Epochs (1)
Name (2) |	Data Path (2) |	LOS Path (2) |	Epochs (2)
⋮	⋮	⋮	⋮
Name (M) |	Data Path (M) |	LOS Path (M) |	Epochs (M)

In the preceding table, M is the number of data sets to be loaded, Name is the colloquial name of the image, Data Path is the absolute path or relative path to the data for that image, LOS Path is the absolute path or relative path to the LOS file for that image, and Epochs is the first scene acquisition epoch followed by a space then the second scene acquisition epoch.

Example:[4]

```
ALOS | Pisco 2007/Pisco_InSAR/ALOS | Pisco 2007/Pisco_InSAR/losALOS | 2007.677 2008.41
```

8.1.4.1 Data File – InSAR Data

Default Location: `<data_root>/<scenario_name>_InSAR/<image_name>.data`, where `<image_name>` is the name of a given station. If there are multiple InSAR datasets for this scenario (for example, the InSAR images are separated based on source), append a short string of characters to identify each folder.

Format: No header. Must be 3 or 4 columns. If 3 columns the standard error is assumed to be the same on all measurements. White space separated columns.

(1)	(2)	(3)	(4)
Longitude (1)	Latitude(1)	d(1)	$\sigma(1)$
Longitude(2)	Latitude(2)	d(2)	$\sigma(2)$
⋮	⋮	⋮	⋮
Longitude(M)	Latitude(M)	d(M)	$\sigma(M)$

In the preceding table, M is the number of InSAR stations from this dataset to be loaded, Epoch is epoch of the measurement, Longitude is the longitude of the pixel, Latitude is the latitude of the pixel, d is the displacement along the LOS direction, and σ is 1-σ uncertainty on d.

Example:

```
100     1     -3.4   1
100.1   1     -3.4   1
100.2   1     -3.4   1
100     1.1   -3.4   1
```

[4] Longitudes/Latitudes have been truncated for readability.

```
100.1   1.1   -3.5   1
100.2   1.1   -3.7   1
100     1.2   -3.5   1
100.1   1.2   -3.7   1
100.2   1.2   -3.9   1
```

8.1.4.2 Data File – InSAR LOS

Default Location: `<data_root>/<scenario_name>_InSAR/<image_name>.los`, where `<image_name>` is the name of a given station.

Format: No header. Must be 3 columns. White space separated columns.

(1)	(2)	(3)
E(1)	N(1)	U(1)
E(2)	N(2)	U(2)
⋮	⋮	⋮
E(M)	N(M)	U(M)

In the preceding table, M is the number of InSAR stations from this dataset to be loaded, E is the Eastward projection of the LOS vector, N is the Northward projection of the LOS vector, and U is the Vertical (up defined to be positive) projection of the LOS vector. Note that the Euclidean length of the vector [E, N, U] should always be 1.

Example:

```
0.89047      0.3729     0.26079
 0.2681     0.70068     0.66118
0.28288     0.93861     0.19744
0.73123    0.013062     0.68201
```

8.2 Fault Models

8.2.1 Patches

8.2.1.1 Rectangular Patches

Default Location: `<data_root>/<scenario_name>_fault/fault.rect`.

Format: No header. Must be 7 columns. White space separated columns.

(1)	(2)	(3)				
E(1)	N(1)	U(1)	Strike (1),	Dip (1)	Length (1)	Width (1)
E(2)	N(2)	U(2)	Strike (2),	Dip (2)	Length (2)	Width (2)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
E(M)	N(M)	U(M)	Strike (M),	Dip (M)	Length (M)	Width (M)

In the preceding table, M is the number of InSAR stations from this dataset to be loaded, E(i) is the east position of the center of the ith patch in the local coordinate frame, N(i) is the north position of the center of the ith patch in the local coordinate frame, U(i) is the depth of the ith patch in the local coordinate frame (beneath the surface is positive), Strike(i) is the strike angle of the ith patch, Dip(i) is the dip angle of the ith patch, Length(i) is the length of the ith patch along strike, Width(i) is the width of the ith patch along dip.

Example:[5]

```
 83.3603 -292.5633    9.7608  327.5937    9.9992   24.9685   18.2791
 98.1062 -282.2383   12.9347  327.5927    9.9991   24.9681   18.2793
112.8523 -271.9132   16.1086  327.5917    9.9990   24.9677   18.2795
```

8.2.1.2 Point Source/Triangular Patches

Default Location: `<data_root>/<scenario_name>_fault/fault.rect`.

Format: No header. Must be 15 columns. White space separated columns.

(1)	(2)	(3)	(4)	(5)	(6)	(7-15)
E(1)	N(1)	U(1)	Strike (1),	Dip (1)	Area (1)	Vertices (1)
E(2)	N(2)	U(2)	Strike (2),	Dip (2)	Area (2)	Vertices (2)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
E(M)	N(M)	U(M)	Strike (M),	Dip (M)	Area (M)	Vertices (M)

In the preceding table, M is the number of InSAR stations from this dataset to be loaded, E(i) is the east position of the center of the ith patch in the local coordinate frame, N(i) is the north position of the center of the ith patch in the local coordinate frame, U(i) is the depth of the ith patch in the local coordinate frame (beneath the surface is positive), Strike(i) is the strike angle of the ith patch, Dip(i) is the dip angle of the ith patch, Area(i) is the area of the ith patch, and Vertices (i) are the vertices of the local triangular elements in the order (x, y, z) in order of increasing depth. If two points have the same depth, then the one with less distance along the strike is listed first (e.g. the Southern most point if a fault has a strike angle of 0).

[5]From `Nias_fault_description_LATLON_ORG1.8N96.6E.dat`

Example (first 6 columns):[6]

```
5.6104    1.8081    5.1734  257.1143   53.8047    3.3068
6.5971    2.3625    4.7394  256.9257   53.6227    3.2926
4.1781   11.9993   -8.3346  255.7129   52.2788    3.1918
```

Example (last 9 columns):

```
7.7556    1.6939    5.9793    4.4519    2.4767    3.9297    4.6238    1.2537    5.6112
7.5837    2.9168    4.3094    4.4519    2.4767    3.9297    7.7556    1.6939    5.9793
6.2087   12.7003   -8.5651    3.0769   12.2602   -9.0129    3.2488   11.0372   -7.4259
```

8.2.2 Origin

This file contains the pre-defined origin for the fault model.

Default Location:<data_root>/<scenario_name>_cGPS2_stations.dat

Format: No header. Must be 2 columns and 1 row. White space separated columns. Leading and trailing white space on the entries will be removed.

(1)	(2)
Longitude	Latitude

In the preceding table, Longitude is the longitude of the local coordinate system origin in decimal degrees, and Latitude is the latitude of the local coordinate system origin in decimal degrees.

Example:

```
96.6   1.8
```

8.2.3 Points for Fault Building

Default Location: <data_root>/<scenario_name>_fault/initial_points.par.

Format: No header. Must be 3 columns. White space separated columns.

(1)	(2)	(3)
E(1)	N(1)	Z(1)
E(2)	N(2)	Z(2)
⋮	⋮	⋮
E(M)	N(M)	U(M)

In the preceding table, M is the number of InSAR stations from this dataset to be loaded, E is the East coordinate of a point on or near the fault surface, N is the North

[6] From running `make_fault_model` on the Nias `Bhuj 2001 Postseismic/faultmodel/fault_points.par`

coordinate of a point on or near the fault surface, and Z is the Depth (down defined to be positive) of a point on or near the surface.

Example:[7]

```
-3.1218791e+01   6.9216024e+00   1.0000000e-02
 2.8131214e+01   1.5781335e+01   1.0000000e-02
-2.8190051e+01  -1.5227328e+01   3.0810000e+01
 3.1249082e+01  -7.4752676e+00   3.0810000e+01
```

8.2.4 Laplacian

Default Location: `<data_root>/<scenario_name>_fault/lap`.

Format: No header. Must have exactly as many columns and rows as patches in the fault. Must be an ascii matrix. If the matrix is designated Lap, then it must obey $Lap(i,:) \cdot l$ =estimate of the Laplacian at patch i for slip distribution l.

Example: Suppose we have a square fault divided into nine patches arranged on an integer lattice and patches are labeled

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \quad ,$$

and slip is assumed to be zero outside of the nine patches. Then a valid Laplacian file for a second-order estimate of the discrete Laplacian would be:

```
-4  1  0  1  0  0  0  0  0
 1 -4  1  0  1  0  0  0  0
 0  1 -4  0  0  1  0  0  0
 1  0  0 -4  1  0  1  0  0
 0  1  0  1 -4  1  0  1  0
 0  0  1  0  1 -4  0  0  1
 0  0  0  1  0  0 -4  1  0
 0  0  0  0  1  0  1 -4  1
 0  0  0  0  0  1  0  1 -4
```

8.2.5 Green's Functions – Already Projected

Default Location: `<data_root>/<scenario_name>_fault/greensfcn`.

[7] From `fault_points.par` for Bhuj.

Format: No header. Must have exactly as many columns as patches in the fault, and exactly as many rows as time-series. Must be an ascii matrix. If the matrix is designated G, then it must obey $G(i,:)\cdot l$ =displacement of time series i for slip distribution l. l must be of the form $[ss_1; ss_2; \cdots ss_N; ds_1; ds_1; \cdots ds_N]$, where ss_j is the strike-slip component of slip on the jth patch and ds_j is the dip-slip component of slip on the jth patch.

Example: If we have four points on the surface and one dislocation at depth (e.g. the example used for GREENFUNC), a possible set of Green's functions for a cGPS2 dataset is

```
          0    0.0003741
 -5.796e-05            0
          0    0.0001968
 -4.159e-05            0
          0     0.000577
 -0.0001362            0
  -0.001089    0.0001124
  0.0009522    1.391e-05
```

Note that already projected Green's functions will need to bypass the projection step of `project_all_greens_fcn` function call after `get_fault_model`.

8.2.6 Green's Functions – Not Projected

Default Location: `<data_root>/<scenario_name>_fault/greensfcn`.

Format: No header. Must have exactly as many columns as patches in the fault, and must have three times as many rows as distinct observation points on the surface. Must be an ascii matrix. If the matrix is designated G, then it must obey $G(3*(i-1)+1,:)\cdot l$ =E displacement of observation location i for slip distribution l, $G(3*(i-1)+2,:)\cdot l$ =N displacement of observation location i for slip distribution l, and $G(3*(i-1)+3,:)\cdot l$ =U displacement of observation location i for slip distribution l. l must be of the form $[ss_1; ss_2; \cdots ss_N; ds_1; ds_1; \cdots ds_N]$, where ss_j is the strike-slip component of slip on the jth patch and ds_j is the dip-slip component of slip on the jth patch.

Example: If we have four points on the surface and one dislocation at depth (e.g. the example used for GREENFUNC), a possible set of Green's functions is

```
          0    0.0003741
 -5.796e-05            0
          0    -0.001187
          0    0.0001968
```

```
  -4.159e-05            0
           0   -0.0009161
           0     0.000577
  -0.0001362            0
           0    -0.001275
   -0.001089    0.0001124
   0.0009522    1.391e-05
  -3.799e-05    0.0004727
```

Chapter 9

.m-Files

In this chapter we give detailed descriptions of the main `.m` files used in the PCAIM package, their inputs, outputs and a general sense of what they do. This chapter is mostly complete but the descriptions of the functions is somewhat lacking compared to an ideal manual. This will be improved in future editions of the manual. Also note that some of the more recent functions have not been included.

This chapter is broken up into five sections:

1. Data Loading/Conventions
2. Data Processing
3. Decompositions
4. Inversions
5. Plotting

9.1 Data/Conventions Loading

These scripts load data or set conventions for the rest of the package.

m-File Summary for PCAIM_driver.m

File Name: PCAIM_driver.m File Type: script
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: `PCAIM_driver` is a self-contained driver file that runs all the scripts necessary to:

1. load multiple datasets with different sampling epochs
2. center the (temporally) dense data, decompose the dense data into linear components
3. create or load a source model/Greens, functions/Laplacian for the source model
4. invert both dense and sparse for dislocations in the source model

See also `set_paths`, `load_scenario_information_Nias`, `load_preferences`, `load_all_data`, `weight_calc`, `create_timeline`, `separate_sparse_data`, `center_data`, `decomp_data`, `get_fault_model`, `invert_components`, `optimize_offsets_final`, `create_predictions`, `plot_model`, `model_statistics`.

Variables

• `scenario_name` is a string denoting the directory in which the models are to be saved and data is to be found.

• `time_unit` is a string denoting what time unit (e.g. ‘year’, ‘day’) will be used as fundamental to the analysis. All time data will be converted into this unit.

• `sig_time` is an integer denoting the number of significant digits after the decimal point to be used to determine if two epochs are the same or not during epoch comparison. This is necessary as number of significant figures for the input times are generally not identical across multiple sources.

• `length_unit` is a string denoting what length unit (e.g. ‘mm’, ‘cm’) will be used as fundamental to the analysis. All length data and errors will be converted into this unit.

• `X_dat` is a cell-structure where each cell contains a matrix of the imported data from a different data source (cGPS3, cGPS2, InSAR, etc.). Each row is one "station" (e.g. for cGPS3) or "location" (e.g. each pixel for InSAR data), and each column is the epoch for each station in that cell.

• `X_err` is a cell-structure where each cell contains a matrix of the imported 1-σ error estimates for each data point from a different data source (cGPS3, cGPS2, InSAR, etc.).

• `X_weight` is a cell-structure where each cell contains a matrix of the imposed multiplicative modifications to the weight of each data point from a different data source (cGPS3, cGPS2, InSAR, etc.). These imposed modifications allow the user to manually reweight portions of the data which his/her geophysical intuition suggests are being either over or under fit. Note this manual reweighting will nearly always worsen the resulting χ^2 of the decomposition. You can think of this as a "fudge factor" for the decomposition and inversion.

• `X_time` is a cell-structure where each cell contains a vector of the imported epochs from a different data source (cGPS3, cGPS2, InSAR, etc.). The j^{th} entry of the vector in the k^{th} cell corresponds to the j^{th} column of the k^{th} cell of `X_err` and `X_dat`.

• `stn_name` is a cell-structure where each cell contains a cell structure of strings of the names of the stations in `X_dat` for data types that have station names. In particular, the i^{th} cell of the k^{th} cell in `stn_name` corresponds to the i^{th} row of the k^{th} cell of `X_err` and `X_dat`.

• `data_type` is a cell-structure where the k^{th} cell contains a string denoting the type of data (e.g. 'cGPS3') in the k^{th} cell of `X_dat`.

• `input_list_file` is a string containing the absolute path of the file containing a list of information on all the data input sources for this scenario.

• `cGPS3_stations` is a cell structure of strings listing the allowed stations. Case sensitive.

• `first_epoch` is a scalar denoting the first allowed epoch in the timeseries.

• `last_epoch` is a scalar denoting the last allowed epoch in the timeseries.

• `timeline` is a vector where the j^{th} entry is the j^{th} unique epoch in chronologically order from any of the data sources.

• `X_time_index` is a cell structure where the k^{th} cell is an index to `X_time` from `timeline`. In other words, the k^{th} cell is vector of the same size as the k^{th} cell of `X_time` such that `timeline(X_time_index{k}{j})=X_time{k}{j}`.

• `n_comp` is a positive integer specifying the number of components for the decomposition of the data matrix into linear components.

• `U` is a $m \times N$ matrix representing the spatial function of the linear decomposition $X \approx USV^t$. The j^{th} column is the spatial function of the j^{th} component.

• `S` is a $N \times N$ diagonal matrix of the weights of the components of the linear decomposition $X \approx USV^t$. The entry (j, j) is the weight of the j^{th} component.

• `V` is $n \times N$ matrix representing the temporal function of the linear decomposition $X \approx USV^t$. The j^{th} column is the temporal function of the j^{th} component.

• `tol` is the convergence tolerance for the linear decomposition function.

• `iter_max` is the maximum number of iterations of the linear decomposition algorithm.

m-File Summary for load_all_data.m

File Name:	load_all_data.m	File Type: function
Author:	Hugo Perfettini	
Maintainer:	Hugo Perfettini	
Contact E-mail:	pcaim@gps.caltech.edu	
Version:	1.0.0.0	
File Description:	Loads all types of data into the PCAIM script. See also PCAIM_driver.	

Input

`load_all_data(data_file,first_epoch,last_epoch,time_unit,sig_time,observation_unit,X_time,position,X_dat,X_err,data_info,data_type,options)`

- `data_file:` full path of file containing dataset information locations.
- `first_epoch:` the earliest epoch to allow data.
- `last_epoch:` the latest epoch to allow data.
- `time_unit:` the time unit to be used internally during calculations.
- `sig_time:` number of significant digits after the decimal point when rounding epochs.
- `observation_unit:` output observation units (m,cm,mm).
- `x_time{i}:` cell containing the time vector of set #i.
- `position{i}:` cell containing the longitude and latitude vectors of dataset i (e.g., the long and lat of observation points, `longitude=positioni(:,1);latitude=position{i}(:,2))`.
- `X_dat{i}:` cell containing the displacement vector of dataset i.
- `X_dat{i}(k,l):` observation for set `i`, time series `k`, at `epoch X_time{i}(l)`.
- `X_err{i}:` same as `X_dat`, but contains the 1-sigma standard errors.
- `data_info{i}:` cell containing informations about dataset i. For cGPS: `data_info{i}{1}{j}`: name of station j within dataset i; `data_info{i}{2}{j}`: path of gps file of station j within dataset i. For InSAR: `data_info{i}{1}`: name of set #i; `data_info{i}{2}`: path to displacement file (towards satellite) of dataset i; `data_info{i}{3}`: path to los file of dataset i.
- `data_type{i}:` type of data considered (e.g., cGPS3, InSAR,...).
- `options:` custom options to use during data input, if any.

Output

`[X_time,position,X_dat,X_err,data_info,data_type] = load_all_data`

- Same definitions as input variables where applicable, except they now contain the loaded datasets from `data_file`.

m-File Summary for load_cGPS3_data.m

File Name: load_cGPS3_data.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: This function loads in 3-component continuous GPS time series from correctly formatted data files and deposits them in the correct data structures. It will only load stations whose names are in the cell structure `cGPS3_station_set`, and it will only load data between epochs `first_valid_epoch` and `last_valid_epoch`. The data is all converted to the scenario unit conventions. See also load_all_data, load_InSAR_data.

Input

```
load_cGPS3_data(input_list,first_epoch,last_epoch,time_unit,sig_time,observation_unit,X_time,position,X_dat,X_err,stn_info,data_type,options)
```

- `input_list`: correctly formatted string from the data_file of the previous script. Format is: Dataset Name | Data Type | Path/To/Dataset/File | Time Unit | Length Unit.
- `first_epoch`: the earliest epoch to allow data.
- `last_epoch`: the latest epoch to allow data.
- `time_unit`: the time unit to be used internally during calculations.
- `sig_time`: number of significant digits after the decimal point when rounding epochs.
- `observation_unit`: output observation units (m,cm,mm).
- `X_time{i}`: cell containing the time vector of set #i.
- `position{i}`: cell containing the longitude and latitude vectors of dataset i (e.g., the long and lat of GPS stations, longitude=position{i}(:,1);latitude=position{i}(:,2)).
- `X_dat{i}`: cell containing the displacement vector of dataset i.
- `X_dat{i}(k,l)`: observation for set i, time series k, at epoch X_time{i}(l).
- `X_err{i}`: same as X_dat, but contains the 1-sigma standard errors.
- `data_info{i}`: cell containing informations about dataset i. For cGPS: data_info{i}{1}{j}: name of station j within dataset i data_info{i}{2}{j}: path of gps file of station j within dataset i.
- `data_type{i}`: type of data considered (e.g., cGPS3, SAR,...).
- `options`: not used by cGPS3.

Output	`[X_time,position,X_dat,X_err,stn_info,data_type] = load_cGPS3_data`

- Same definitions as input variables where applicable, except they now contain the loaded datasets from input_list.

m-File Summary for load_InSAR_data.m

File Name:	load_InSAR_data.m	File Type: function
Author:	Hugo Perfettini	
Maintainer:	Hugo Perfettini	
Contact E-mail:	pcaim@gps.caltech.edu	
Version:	1.0.0.0	
File Description:	Loads InSAR images into the PCAIM program. See also load_all_data, load_cGPS3_data.	

Input

`load_InSAR_data(input_list,first_epoch,last_epoch,time_unit,sig_time,observation_unit,X_time,position,X_dat,X_err,data_info,data_type,options)`

- `input_list`: correctly formatted string from the data_file of the previous script. Format is: Dataset Name | Data Type | Path/To/Dataset/File | Time Unit | Length Unit.
- `first_epoch`: the earliest epoch to allow data.
- `last_epoch`: the latest epoch to allow data.
- `time_unit`: the time unit to be used internally during calculations.
- `sig_time`: number of significant digits after the decimal point when rounding epochs.
- `observation_unit`: output observation units (m,cm,mm).
- `X_time{i}`: cell containing the time vector of set #i.
- `position{i}`: cell containing the longitude and latitude vectors of dataset i (e.g., the long and lat of GPS stations, longitude=position{i}(:,1);latitude=position{i}(:,2)).
- `X_dat{i}`: cell containing the displacement vector of dataset i.
- `X_dat{i}(k,l)`: observation for set i, time series k, at epoch X_time{i}(l).
- `X_err{i}`: same as X_dat, but contains the 1-sigma standard errors.
- `data_info{i}`: cell containing informations about dataset i. For cGPS: data_info{i}{1}{j}: name of station j within dataset i data_info{i}{2}{j}: path of gps file of station j within dataset i.
- `data_type{i}`: type of data considered (e.g., cGPS3, SAR,...).
- `options`: not used by cGPS3.

Output

`[X_time,position,X_dat,X_err,data_info,data_type]=load_InSAR_data`

- Same definitions as input variables where applicable, except they now contain the loaded datasets from input_list.

m-File Summary for load_preferences.m

File Name:	load_preferences.m — File Type: script
Author:	Andrew Kositsky
Maintainer:	Hugo Perfettini
Contact E-mail:	pcaim@gps.caltech.edu
Version:	1.0.0.0
File Description:	load_preferences is the correct place to set any preferences necessary previous to the loading data steps. For now, this is just initialization of variables. See also PCAIM_driver, set_defaults.

m-File Summary for `read_date.m`

File Name: `read_date.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: `read_date(date,separator)` parses `date` via the separator character `separator`. Example: `sample_date='2007/08/20 | 13:54:56.34'; read_date(sample_date,'|')`. See also `load_cGPS3_data`, `load_InSAR_data`.

Input `read_date(date,separator)`

- `date:` a date in the format: `YYYY/MM/DD` <`seperater`> `HH:MI:SS` where `YYYY` is the year, `MM` is the month, `DD` is the day, `HH` is the hour, `MI` is the minute, `SS` is the decimal seconds (arbitrary number of digits after the first two if decimal is needed.)
- `separator:` a character that separates the year-month-day from the hour-minute-second

---

Output `date_output=read_date`

- `date_output:` same as the input date but in decimal years.

m-File Summary for separate_sparse_data.m

File Name: separate_sparse_data.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Divides the data into "dense" and "sparse" data.

Input

`separate_sparse_data(X_time_index,position,X_dat,X_err,data_info,data_type,X_weight,sparse_list)`

- The input arguments to SEPARATE_SPARSE_DATA are the same as the output arguments of LOAD_ALL_DATA, with the exception of SPARSE_LIST, which is a cell structure of strings containing the names of datatypes considered sparse to the user.

Output

`[X_time_index_dense,position_dense,X_dat_dense,X_err_dense,data_info_dense,data_type_dense,X_weight_dense,X_time_index_sparse,position_sparse,X_dat_sparse,X_err_sparse,data_info_sparse,data_type_sparse,X_weight_sparse,all_position]=separate_sparse_data`

- The output is similarly the same as the inputs, except that variables that have _DENSE after them are only contain dense data and those that have _SPARSE after them only contain sparse data. The only exception is ALL_POSITION, which is the list of all observatino point positions on the surface listed first for dense datasets, then for sparse datasets.

m-File Summary for set_defaults.m

File Name: set_defaults.m File Type: script
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Assigns the default values of various parameters and initializes variables that need to be initialized. See also PCAIM_DRIVER, LOAD_PREFERENCES.

Variables

- n_comp
- n_comp_mean
- mean_function
- basic
- center_function
- meanless_V
- X_dat
- X_err
- X_rescale
- X_time'
- X_loc
- data_type
- stn_name
- long
- lat
- options
- position

m-File Summary for `weight_calc.m`

File Name: `weight_calc.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: Weigh the various data entries for optimization `X_WEIGHT = WEIGHT_CALC(X_err)` recasts each entry of each cell of `X_err` as `X_WEIGHT{k,l}(i,j) = 1 / X_ERR{k,l}(i,j)^2`. For instance, this means calculation of χ^2 of the data is then `(X_MODEL - X_DAT).^2 * X_WEIGHT` during the decomposition (for minimization of χ^2). `X_WEIGHT = WEIGHT_CALC(X_ERR, X_RESCALE)` recasts each entry of each cell of `X_err` as `X_WEIGHT{k,l}(i,j) = 1 / X_ERR{k,l}(i,j)^2 * X_RESCALE{k,l}(i,j)`. This allows consistent reweighting of the data for decomposition purposes while not modifying the original error bars. `X_WEIGHT = WEIGHT_CALC(X_ERR, X_RESCALE,P)` recasts each entry of each cell of `X_ERR` as `X_WEIGHT{k,l}(i,j) = 1 / abs(X_ERR{k,l}(i,j))^P * X_RESCALE{k,l}(i,j)`. This allows consistent reweighting of the data for decomposition purposes and use of the pth power on the error weight. Example: `X_err = {[1:5;2:0.5:4]}`; `X_rescale = {ones(2,5)}`; `X_rescale{1}(1,1) = 1/5`; `X_weight = weight_calc(X_err,X_rescale)`. See also `PCAIM_DRIVER`.

Input

`weight_calc(X_err,X_rescale,p)`

- `X_err`
- `X_rescale`
- `p`

Output

`X_weight = weight_calc`

- `X_weight`

9.2 Decompositions

These scripts perform linear decompositions of the data.

m-File Summary for `calc_abg_multi_component.m`

File Name: `calc_abg_multi_component.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `X_dat = func_options{1}; X_weight = func_options{2}; u_index = func_options{3}; v_index = func_options{4}; n_datasets = func_options{5};` Example: `PCAIM_driver` See also `dfunc_mean_zero_sum_V_transform`, `conjugate_gradient`, `decomp_srebro_CG_simultaneous`, `decomp_data`, `PCAIM_driver`.

Input

`calc_abg_multi_component(x,r,func_options)`

- `x`
- `r`
- `func_options`

Output

`[alpha, beta, gamma] = calc_abg_multi_component`

- `alpha`
- `beta`
- `gamma`

m-File Summary for calc_abg_zero_sum_V_transform.m

File Name: calc_abg_zero_sum_V_transform.m

File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: The time functions buried in the input guess X are in an orthogonal basis of a subspace M of all time functions such that $w \in M$ if and only if $sum(w) = 0$. In order to compute the objective function in this case, we transform back into a basis with basis vectors $[1, 0, ..., -1]$, $[0, 1, 0, ..., -1]$, ..., $[0, 0, ..., 1, -1]$, from which it is easy to calculate the displacement at each time for each component. Example: `PCAIM_driver`. See also `dfunc_mean_zero_sum_V_transform`, `conjugate_gradient`, `decomp_srebro_CG_simultaneous`, `decomp_data`, `PCAIM_driver`.

Input

`calc_abg_zero_sum_V_transform(x,r,func_options)`

- `x`
- `r`
- `func_options`

Output

`[alpha, beta, gamma] = calc_abg_zero_sum_V_transform`

- `alpha`
- `beta`
- `gamma`

m-File Summary for decomp_data.m

File Name: decomp_data.m File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: DECOMP_DATA General decomposition function call. [U,S,V,CHI2_MODIFIED,ELAPSED_TIME,ITER_NUM] = DECOMP_DATA(X_DAT, X_TIME_INDEX, X_WEIGHT, N_COMP, DECOMP_FUNCTION, DECOMP_OPTIONS) calls DECOMP_FUNCTION in EVAL with the same inputs and outputs (except DECOMP_FUNCTION is ommitted from the inputs) as this function. This allows the user to change the decomposition function without needing to edit PCAIM_driver. Example: PCAIM_driver. Also see DECOMP_SREBRO_CG_SIMULTANEOUS, decomp_srebro_EM, PCAIM_driver.

Input

decomp_data(X_dat, X_time_index, X_weight, n_comp, decomp_function, decomp_options)

- X_dat
- X_time_index
- X_weight
- n_comp
- decomp_function
- decomp_options

Output

[U,S,V,chi2_modified,elapsed_time,iter_num] = decomp_data

- U
- S
- V
- chi2_modified
- elapsed_time
- iter_num

m-File Summary for decomp_srebro_CG_simultaneous.m

File Name: decomp_srebro_CG_simultaneous.m

File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: DECOMP_SREBRO_CG_SIMULTANEOUS Simultanous multicomponent low-rank matrix approximation using CG on weighted F-norm [U,S,V,CHI2_MODIFIED,ELAPSED_TIME,ITER_NUM] = DECOMP_SREBRO_CG_SIMULTANEOUS(X_DAT, X_TIME_INDEX, X_WEIGHT, N_COMP, DECOMP_OPTIONS) uses the conjugate gradient algorithm on N_COMP components simultaneously to find the local minimum of the objective function, given in DECOMP_OPTIONS, with weights X_WEIGHT. Example: PCAIM_driver. See also decomp_data, decomp_srebro_EM, PCAIM_driver.

Input

`decomp_srebro_CG_simultaneous(X_dat, X_time_index, X_weight, n_comp, decomp_options)`

- X_dat
- X_time_index
- X_weight
- n_comp
- decomp_options

Output

`[U,S,V,chi2_modified,elapsed_time,iter_num] = decomp_srebro_CG_simultaneous`

- U
- S
- V
- chi2_modified
- elapsed_time
- iter_num

m-File Summary for `decomp_srebro_EM.m`

File Name: `decomp_srebro_EM.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `DECOMP_SREBRO_EM` Simultanous multicomponent low-rank matrix approximation using EM on weighted F-norm `[U,S,V,CHI2_MODIFIED,ELAPSED_TIME,ITER_NUM] = DECOMP_SREBRO_EM(X_DAT, X_TIME_INDEX, X_WEIGHT, N_COMP, DECOMP_OPTIONS)` uses a expectation maximization routine on `N_COMP` components simultaneously to find the local minimum of the objective function, given in `DECOMP_OPTIONS`, with weights `X_WEIGHT`. Example: `PCAIM_driver`. See also `decomp_data`, `DECOMP_SREBRO_CG_SIMULTANEOUS`, `PCAIM_driver`.

Input

`decomp_srebro_EM(X_dat, X_time_index, X_weight, n_comp, decomp_options)`

- `X_dat`
- `X_time_index`
- `X_weight`
- `n_comp`
- `decomp_options`

Output

`[U,S,V,chi2_modified,elapsed_time,iter_num] = decomp_srebro_EM`

- `U`
- `S`
- `V`
- `chi2_modified`
- `elapsed_time`
- `iter_num`

m-File Summary for gradient_descent_chi2.m

File Name: gradient_descent_chi2.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description:

Input

```
gradient_descent_chi2(X_dat,x,func,dfunc,iter_max,ftol,
func_options,dfunc_options)
```

- X_dat
- x
- func
- dfunc
- iter_max
- ftol
- func_options
- dfunc_options

Output

```
[x,F,iter] = gradient_descent_chi2
```

- x
- F
- iter

9.2.1 Centering

m-File Summary for `center_data.m`

File Name: `center_data.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: CENTER_DATA centers the input data matrix X_DAT. [X_DAT,MEAN_OFFSETS] = CENTER_DATA(X_DAT, X_TIME_INDEX, X_WEIGHT, CENTER_FUNCTION, MEAN_FUNCTION, N_COMP_MEAN, MEAN_OPTIONS) has two primary run methods at this point, CENTER_FUNCTION = 'basic', where the MEAN_OFFSETS are estimated via a weighted mean of the data, and CENTER_FUNCTION = 'advanced' where the MEAN_OFFSETS are estimated via a N_COMP_MEAN component linear decomposition plus mean offsets model. MEAN_FUNCTION is the name of the function to-be-called for determining the optimal values of the means, and MEAN_OPTIONS are the options necessary for MEAN_FUNCTION. X_DAT, X_TIME_INDEX, X_WEIGHT as inputs are all the same as in the script PCAIM_DRIVER. X_DAT as an output is the input X_DAT with the MEAN_OFFSETS estimate removed. Example: PCAIM_driver. See also PCAIM_DRIVER.

Input

`center_data(X_dat, X_time_index, X_weight,center_function, mean_function, n_comp_mean, mean_options)`

- `X_dat`
- `X_time_index`
- `X_weight`
- `center_function`
- `mean_function`
- `n_comp_mean`
- `mean_options`

Output

`[X_dat,mean_offsets] = center_data`

- `X_dat`
- `mean_offsets`

m-File Summary for decomp_CG_means.m

File Name: decomp_CG_means.m   File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Decompose X_dat into linear components and mean offsets [MEAN_OFFSET_FINE,ELAPSED_TIME,ITER_NUM] = DECOMP_CG_MEANS(X_DAT, X_TIME_INDEX, X_WEIGHT, U,S,V, MEAN_OPTIONS) decomposes the data cell X_DAT into a number of linear components N_COMP_MEAN specified in MEAN_OPTIONS and one mean estimate per time series. X_DAT, X_TIME_INDEX, X_WEIGHT are all the same as in PCAIM_DRIVER; U,S,V form an initial guess at the decomposition (U*S,V are the assumed components); and MEAN_OPTIONS gives the function FUNC and derivative DFUNC of the function to be used by the conjugate gradient algorithm, the maximum number of iterations within the conjugate gradient algorithm (iter_max), convergence tolerance (TOL), and options for FUNC and DFUNC. Example: PCAIM_driver. See also CENTER_DATA, DECOMP_MEANS, PCAIM_DRIVER.

Input

```
ecomp_CG_means(X_dat, X_time_index, X_weight, U,S,
V, mean_options)
```

- X_dat
- X_time_index
- X_weight
- U
- S
- V
- mean_options

Output

```
[mean_offset_fine,elapsed_time,iter_num] = decomp_CG_
means
```

- mean_offset_fine
- elapsed_time
- iter_num

m-File Summary for decomp_means.m

File Name: decomp_means.m | File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: DECOMP_MEANS decomposes X_dat to achieve an estimate of the mean offsets DECOMP_MEANS is an eval statement call with X_DAT, X_TIME_INDEX, X_WEIGHT as defined in PCAIM_DRIVER, some MEAN_FUNCTION from which we will get MEAN_OFFSET_FINE with which to correct the mean of X_DAT, the ELAPSED_TIME of the mean-searching procedure, and the number of iterations ITER_NUM of the algorithm used), an initial guess at a linear decomposition U,S,V, and options for MEAN_FUNCTION in MEAN_OPTIONS. DECOMP_MEANS is programmed generally so that the user can change the MEAN_FUNCTION to be used easily. All that needs to happen is the input and output arguments are the same as the default MEAN_FUNCTION, DECOMP_CG_MEANS. Example: PCAIM_driver. See also DECOMP_CG_MEANS, CENTER_DATA, PCAIM_DRIVER.

Input

```
decomp_means(X_dat, X_time_index,X_weight, U,S,V, mean_function, mean_options)
```

- X_dat
- X_time_index
- X_weight
- U
- S
- V
- mean_function
- mean_options

Output

```
[mean_offset_fine,elapsed_time,iter_num] = decomp_means
```

- mean_offset_fine
- elapsed_time
- iter_num

9.2.2 Conjugate Gradient

m-File Summary for `conjugate_gradient.m`

File Name: `conjugate_gradient.m` File Type: function
Author: Martin King; heavily modified by Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description:

1. Conjugate Gradient Method with Flecther-Reeves (or Polak-Ribiere) to find a vector `x` that gives a MINIMUM of a function (a scalar).
2. Ideas taken from J.R. Shewchuk and Numerical Recipes.
3. You must modify your own function to minimise in a Matlab function file called `func.m` (scalar output) and the first derivative of that function in a matlab function file called `dfunc.m`, which has a vector output in `(del/del(x1) del/del(x2) ... etc)'`.
4. If you want to MAXIMISE a function, multiply -1 to the output of `func.m` and dfunc.m (be careful here, `dfunc.m` may use func.m; to be safe, give `dfunc.m` the original output of func.m and then multiply -1 to dfunc.m at the end). If you are brave, reverse the search direction r and d for maximisation.
5. If the method is not converging or is giving you a solution that doesn't make sense, change the initial guess.
6. As an example, a simple function is given in func.m and its gradient vector in `dfunc.m`. Change the initial guess to `x = [1 ; 1]` for example, the solution it gives is incorrect. The reason is obvious if you plot the function (it is the saddle points).
7. I have used these scripts to optimise a fairly complicated function. They seem to work well. If you notice any bug or have any comment, please email me king at ictp at it.

Input `conjugate_gradient(X_dat,x,func,dfunc,iter_max,ftol, func_options,dfunc_options)`

- X_dat
- x
- func
- dfunc
- iter_max
- ftol
- func_options
- dfunc_options

Output

`[x,F,iter] = conjugate_gradient`

- x
- F
- iter

m-File Summary for dfunc_mean_zero_sum_V_transform_corrected.m

File Name: dfunc_mean_zero_sum_V_transform_corrected.m
File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: FUNC_MULTI_COMPONENT is a multicomponent Srebro objective function for CG. [F_OUT] = DFUNC_MEAN_ZERO_SUM_V_TRANSFORM_CORRECTED(X,FUNC_OPTIONS) calculates the derivative of the modified reduced chi-square value for the input guess X, using DFUNC_OPTIONS to parse this vector input: X_dat = dfunc_options1; X_weight = dfunc_options2; u_index = dfunc_options3; v_index = dfunc_options4; n_datasets = dfunc_options5; X_row = dfunc_options6; v_end_entry = dfunc_options7; v_index_size = dfunc_options8; X_row = dfunc_options9; n_comp_mean = dfunc_options10; n_tseries = dfunc_options11; n_epochs = dfunc_options12; X_time_index = dfunc_options13; transformation_matrix = dfunc_options14; transformation_matrix_inv= dfunc_options15; v_index_in_x = dfunc_options16. The time functions buried in the input guess X are in an orthogonal basis of a subspace M of all time functions such that $w \in M$ if and only if sum(w) = 0. In order to compute the objective function in this case, we transform back into a basis with basis vectors [1, 0 , ... -1], [0,1,0,...,-1], ... [0,0,...,1,-1], from which it is easy to calculate the displacement at each time for each component. Example: PCAIM_driver See also dfunc_mean_zero_sum_V_transform, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

dfunc_mean_zero_sum_V_transform_corrected(x,dfunc_options)

- x
- dfunc_options

Output

[fprime_out] = dfunc_mean_zero_sum_V_transform_corrected

- `fprime_out`

m-File Summary for dfunc_mean_zero_sum_V_transform.m

File Name: dfunc_mean_zero_sum_V_transform.m
File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: FUNC_MULTI_COMPONENT multicomponent Srebro objective function for CG. [F_OUT] = DFUNC_MEAN_ZERO_SUM_V_TRANSFORM(X,FUNC_OPTIONS) calculates the derivative of the modified reduced chi-square value for the input guess X, using DFUNC_OPTIONS to parse this vector input: X_dat = dfunc_options1; X_weight = dfunc_options2; u_index = dfunc_options3; v_index = dfunc_options4; n_datasets = dfunc_options5; X_row = dfunc_options6; v_end_entry = dfunc_options7; v_index_size = dfunc_options8; X_row = dfunc_options9; n_comp_mean = dfunc_options10; n_tseries = dfunc_options11; n_epochs = dfunc_options12; X_time_index = dfunc_options13; transformation_matrix = dfunc_options14; transformation_matrix_inv= dfunc_options15; v_index_in_x = dfunc_options16. The time functions buried in the input guess X are in an orthogonal basis of a subspace M of all time functions such that $w \in M$ if and only if sum(w) = 0. In order to compute the objective function in this case, we transform back into a basis with basis vectors [1, 0 , ... -1], [0,1,0,...,-1], ... [0,0,...,1,-1], from which it is easy to calculate the displacement at each time for each component. Example: PCAIM_driver. See also dfunc_mean_zero_sum_V_transform, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

`dfunc_mean_zero_sum_V_transform(x,dfunc_options)`

- x
- dfunc_options

Output

`[fprime_out] = dfunc_mean_zero_sum_V_transform`

- fprime_out

m-File Summary for dfunc_multi_component.m

File Name: dfunc_multi_component.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: DFUNC_MULTI_COMPONENT multicomponent Srebro objective function derivative for CG. [F_PRIME_OUT] = DFUNC_MULTI_COMPONENT(X,DFUNC_OPTIONS) calculates the derivative of the modified reduced chi-square value for the input guess X, using DFUNC_OPTIONS to parse this vector input: X_dat = dfunc_options1; X_weight = dfunc_options2; u_index = dfunc_options3; v_index = dfunc_options4; n_datasets = dfunc_options5; X_row = dfunc_options6; n_comp_mean = dfunc_options7; n_tseries = dfunc_options8; n_epochs = dfunc_options9; X_time_index = dfunc_options10; Example: PCAIM_driver See also func_multi_component, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

dfunc_multi_component(x,dfunc_options)

- x
- dfunc_options

Output

[fprime_out] = dfunc_multi_component

- fprime_out

m-File Summary for func_mean_zero_sum_V_transform_corrected.m

File Name: func_mean_zero_sum_V_transform_corrected.m

File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: FUNC_MULTI_COMPONENT Multicomponent Srebro objective function for CG. [F_OUT] = FUNC_MEAN_ZERO_SUM_V_TRANSFORM_CORRECTED(X,FUNC_OPTIONS) calculates the modified reduced chi-square value for the input guess X, using FUNC_OPTIONS to parse this vector input: X_dat = func_options1; X_weight = func_options2; u_index = func_options3; v_index = func_options4; n_datasets = func_options5; mean_index = func_options6; v_end_entry = func_options7; v_index_size = func_options8; n_epochs =func_options9; n_comp =func_options10; transformation_matrix_inv = func_options11; v_index_in_x = func_options12. The time functions buried in the input guess X are in an orthogonal basis of a subspace M of all time functions such that $w \in M$ if and only if sum(w) = 0. In order to compute the objective function in this case, we transform back into a basis with basis vectors [1, 0 , ... -1], [0,1,0,...,-1], ... [0,0,...,1,-1], from which it is easy to calculate the displacement at each time for each component. Example: PCAIM_driver See also dfunc_mean_zero_sum_V_transform, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

func_mean_zero_sum_V_transform_corrected(x,func_options)

- x
- func_options

Output

[f_out] = func_mean_zero_sum_V_transform_corrected

- f_out

m-File Summary for func_mean_zero_sum_V_transform.m

File Name: func_mean_zero_sum_V_transform.m

File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: FUNC_MULTI_COMPONENT multicomponent Srebro objective function for CG. [F_OUT] = FUNC_MEAN_ZERO_SUM_V_TRANSFORM(X,FUNC_OPTIONS) calculates the modified reduced chi-square value for the input guess X, using FUNC_OPTIONS to parse this vector input: X_dat = func_options1; X_weight = func_options2; u_index = func_options3; v_index = func_options4; n_datasets = func_options5; mean_index = func_options6; v_end_entry = func_options7; v_index_size = func_options8; n_epochs =func_options9; n_comp =func_options10; transformation_matrix_inv = func_options11; v_index_in_x = func_options12. The time functions buried in the input guess X are in an orthogonal basis of a subspace M of all time functions such that $w \in M$ if and only if sum(w) = 0. In order to compute the objective function in this case, we transform back into a basis with basis vectors [1, 0 , ... -1], [0,1,0,...,-1], ... [0,0,...,1,-1], from which it is easy to calculate the displacement at each time for each component. Example: PCAIM_driver. See also dfunc_mean_zero_sum_V_transform, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

`func_mean_zero_sum_V_transform(x,func_options)`

- x
- func_options

Output

`[f_out] = func_mean_zero_sum_V_transform`

- f_out

m-File Summary for func_multi_component.m

File Name: func_multi_component.m File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: FUNC_MULTI_COMPONENT multicomponent Srebro objective function for CG. [F_OUT] = FUNC_MULTI_COMPONENT(X,FUNC_OPTIONS) calculates the modified reduced chi-square value for the input guess X, using FUNC_OPTIONS to parse this vector input: X_dat = func_options1; X_weight = func_options2; u_index = func_options3; v_index = func_options4; n_datasets = func_options5; Example: PCAIM_driver. See also dfunc_multi_component, conjugate_gradient, decomp_srebro_CG_simultaneous, decomp_data, PCAIM_driver.

Input

`func_multi_component(x,func_options)`

- x
- func_options

Output

`[f_out] = func_multi_component`

- f_out

9.3 Fault Related

These scripts define, load, and/or manipulate aspects of the code concerning the fault model.

m-File Summary for compute_laplacian_driver.m

File Name: compute_laplacian_driver.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: COMPUTE_LAPLACIAN_DRIVER Driver script to generate Laplacian [LAP,IEDGE]=COMPUTE_LAPLACIAN_DRIVER(FAULT_MODEL,OPTIONS) takes in a fault model, computes an approximation of the Laplacian for the fault model, and attempts to find the edges of the fault and assigns these to IEDGE. Example: PCAIM_driver. See also compute_laplacian, get_fault_model, PCAIM_driver.

Input

`compute_laplacian_driver(fault_model,options)`

- fault_model
- options

Output

`[Lap,iedge] = compute_laplacian_driver`

- Lap
- iedge

m-File Summary for compute_laplacian.m

File Name: compute_laplacian.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: COMPUTE_LAPLACIAN Generate Laplacian with respect to a set of points. LAP = COMPUTE_LAPLACIAN(X,Y,Z,N) takes in the coordinates of a set of points that are randomly scattered on an unknown surface and generates a discrete approximation of the Laplacian using the nearest N points. X, Y, and Z are column vectors of the same length, and N must be a positive integer. Example: x = repmat([0:5],6,1); x = x(:); y = repmat([0:5]',1,6); y = y(:); z = repmat([0:0.1:0.5],6,1); z = z(:); N = 4; Lap = compute_laplacian(x,y,z,N). See also compute_laplacian, get_fault_model, PCAIM_driver.

Input

`compute_laplacian(x,y,z,N)`

- x
- y
- z
- N

Output

`Lap = compute_laplacian`

- Lap

m-File Summary for compute_point_source.m

File Name: compute_point_source.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: COMPUTE_RECTANGULAR_SOURCE Compute Okada Greens function for rectangular patches. G = COMPUTE_RECTANGULAR_SOURCE(X,Y,Z,STRIKE,DIP,AREA,VERTICES, POSITION, GREENSEXTERNALFCNDIR) computes the Green's functions for patches with 6 parameters X, Y, Z, STRIKE, DIP, AREA and positions on the surface given in POSITION. GREENSEXTERNALFCNDIR tells the function where to look for the compiled FORTRAN code that does the computation. Example: PCAIM_driver. See also get_fault_model,compute_rectangular_source, PCAIM_driver.

Input

`compute_point_source(x,y,z,strike,dip,area,vertices, position,GreensExternalFcnDir)`

- x
- y
- z
- strike
- dip
- area
- vertices
- position
- GreensExternalFcnDir

Output

`G = compute_point_source`

- G

m-File Summary for compute_rectangular_source.m

File Name: compute_rectangular_source.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: COMPUTE_RECTANGULAR_SOURCE Compute Okada Greens function for rectangular patches. G = COMPUTE_RECTANGULAR_SOURCE(X,Y,Z,STRIKE,DIP,LENGTH,WIDTH, POSITION, GREENSEXTERNALFCNDIR) computes the Green's functions for patches with 7 parameters X, Y, Z, STRIKE, DIP, LENGTH, WIDTH and positions on the surface given in POSITION. GREENSEXTERNALFCNDIR tells the function where to look for the compiled FORTRAN code that does the computation. Example: PCAIM_driver. See also get_fault_model, compute_point_source, PCAIM_driver.

Input

`compute_rectangular_source(x,y,z,strike,dip,length, width,position,GreensExternalFcnDir)`

- x
- y
- z
- strike
- dip
- length
- width
- position
- GreensExternalFcnDir

Output

`G = compute_rectangular_source`

- G

m-File Summary for find_rectangle_param.m

File Name: find_rectangle_param.m File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: FIND_RECTANGLE_PARAM Find useful values for rectangular patches. [RAKE,AREA,VERTICES,STRIKE_VECT,UPDIP_VECT,NORMAL_VECT]= FIND_RECTANGLE_PARAM(X,Y,Z,STRIKE,DIP,LENGTH,WIDTH,VECT_TECT) Takes in the seven defining parameters for an Okada formulation rectangular patch and a vector defining the overall tectonic motion vector, and it outputs the rake on each patch in a vector RAKE, the area of each patch in a vector AREA, the vertices of the rectangle in a matrix VERTICES, the strike vectors in a matrix STRIKE_VECT, the up-dip vectors in a matrix UPDIP_VECT, and the normal vectors to each patch in a matrix NORMAL_VECT. Example: PCAIM_driver. Also see find_triangle_param, PCAIM_driver.

Input

find_rectangle_param(x,y,z,strike,dip,length,width, vect_tect)

- find_rectangle_param(x
- y
- z
- strike
- dip
- length
- width
- vect_tect

Output

[rake,area,vertices,strike_vect,updip_vect,normal_vect] = find_rectangle_param

- rake
- area
- vertices
- strike_vect
- updip_vect
- normal_vect

m-File Summary for `find_triangle_param.m`

File Name: `find_triangle_param.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: `FIND_TRIANGLE_PARAM` Find useful values for triangular-point sources. `[XC,YX,ZC,STRIKE,DIP,RAKE,AREA,VERTICES,STRIKE_VECT, UPDIP_VECT, NORMAL_VECT]= FIND_TRIANGLE_PARAM(T1,T2,T3,VECT_TECT)` Takes in the three corners of the patches and the overall tectonic motion vector, and it outputs the x,y,z coordinates of the center of the triangular patch (`XC`, `YC`, `ZC`), the strike vector for each patch (`STRIKE`), the up-dip vector (`DIP`), the rake on each patch in a vector (`RAKE`), the area of each patch in a vector (`AREA`), the vertices of the rectangle in a matrix (`VERTICES`), the strike vectors in a matrix (`STRIKE_VECT`), the up-dip vectors in a matrix (`UPDIP_VECT`), and the normal vectors to each patch in a matrix (`NORMAL_VECT`). Example: `PCAIM_driver`. Also see `find_triangle_param`, `PCAIM_driver`.

Input

`find_triangle_param(t1,t2,t3,vect_tect)`

- `t1`
- `t2`
- `t3`
- `vect_tect`

Output

`[xc,yc,zc,strike,dip,rake,area,vertices,strike_vect, updip_vect,normal_vect] = find_triangle_param`

- `xc`
- `yc`
- `zc`
- `strike`
- `dip`
- `rake`
- `area`
- `vertices`

- `strike_vect`
- `updip_vect`
- `normal_vect`

m-File Summary for get_fault_model.m

File Name: get_fault_model.m File Type: function
Author: Andrew Kositsky and Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: GET_FAULT_MODEL Load or build a fault model, Greens fcn, and Laplacian. [G,LAP,FAULT_MODEL,ORIGIN,RECTANGULAR_FAULT_FLAG,IEDGE] = GET_FAULT_MODEL(GET_FAULT_MODEL_OPTIONS,LAPLACIAN_OPTIONS) generates/loads the fault model, Green's functions, and Laplacian for the inversion scenario. Example: PCAIM_driver. See also load_fault_model_rect, load_fault_model_point, load_greens_function, compute_rectangular_source, compute_point_source, load_laplacian, compute_laplacian_driver, PCAIM_driver.

Input

get_fault_model(get_fault_model_options,laplacian_options)

- get_fault_model_options
- laplacian_options

Output

[G,Lap,fault_model,origin,rectangular_fault_flag, iedge] = get_fault_model

- G
- Lap
- fault_model
- origin
- rectangular_fault_flag
- iedge

9.3.1 Green's Function

m-File Summary for load_fault_model_point.m

File Name: load_fault_model_point.m File Type: function
Author: Andrew Kositsky and Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: LOAD_FAULT_MODEL_POINT Load 6 parameters plus vertices of point patches. [X,Y,Z,STRIKE,DIP,AREA,VERTICES] =LOAD_FAULT_MODEL_RECT(FaultModelFile) loads the X, Y, Z coordinates of each rectangular fault element from the file FAULTMODELFILE with respect to some pre-defined origin. FAULTMODELFILE is purely numeric (no header) with a format defined by the "indexes" below. By default, these are set to: east_index = 1; East offset of center from origin (km) north_index = 2; North offset of center from origin (km) depth_index = 3; depth of lower edge of fault (km) strike_index = 4; strike, clockwise from N (degrees) dip_index = 5; dip angle, from the horizontal (degrees) area_index = 6; fault length along the strike direction(km) first_vertex_index = 7; fault width in dip direction (km). It is reasonable to modify this file if your standard input data comes in a different order. It is assumed that all columns after first_vertex_index are other vertices. Example: FaultModelFile = ['Pisco 2007 Postseismic/faultmodel/'pisco.trg']; [x,y,z,strike,dip,length,width] = load_fault_model_rect(FaultModelFile). See also load_fault_model_rect, get_fault_model, PCAIM_driver.

Input

`load_fault_model_point(FaultModelFile)`

- FaultModelFile

Output

`[x,y,z,strike,dip,area,vertices] = load_fault_model_point`

- x
- y
- z
- strike

- `dip`
- `area`
- `vertices`

m-File Summary for load_fault_model_rect.m

File Name: load_fault_model_rect.m File Type: function
Author: Andrew Kositsky and Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: LOAD_FAULT_MODEL_RECT Load all 7 parameters of the rectangular patches. [X,Y,Z,STRIKE,DIP,LENGTH,WIDTH] =LOAD_FAULT_MODEL_RECT(FaultModelFile) loads the X, Y, Z coordinates of each rectangular fault element from the file FAULTMODELFILE with respect to some pre-defined origin, the STRIKE and DIP angles of the fault element, and the LENGTH and WIDTH of the fault element. FAULTMODELFILE is purely numeric (no header) with a format defined by the "indexes" below. By default, these are set to: east_index = 1; East offset of center from origin (km) north_index = 2; North offset of center from origin (km) depth_index = 3; depth of lower edge of fault (km) strike_index = 4; strike, clockwise from N (degrees) dip_index = 5; dip angle, from the horizontal (degrees) length_index = 6; fault length along the strike direction (km) width_index = 7; fault width in dip direction (km) It is reasonable to modify this file if your standard input data comes in a different order. It is assumed that all columns after first_vertex_index are other vertices. Example: FaultModelFile = ['Nias 2005 Postseismic/fault/'Nias_fault_description_LATLON_ORG1.8N9 6.6E.dat']; [x,y,z,strike,dip,length,width] = load_fault_model_rect(FaultModelFile). See also load_fault_model_point, get_fault_model, PCAIM_driver.

Input

`load_fault_model_rect(FaultModelFile)`

- FaultModelFile

Output

`[x,y,z,strike,dip,length,width] =load_fault_model_rect`

- x
- y
- z

- `strike`
- `dip`
- `length`
- `width`

m-File Summary for `load_greens_function.m`

File Name: `load_greens_function.m` File Type: function

Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `LOAD_GREENS_FUNCTION` Loads in an existing Greens function from a file. `G = LOAD_GREENS_FUNCTION(GREENSFUNCTIONFILE)` loads in a pre-defined Greens function matrix from a file, `GREENSFUNCTION FILE`, and places it in the Greens function matrix `G`. The `GREENSFUNCTIONFILE` is assumed to be in the format: `disloc_x_val = data1`; `disloc_y_val = data2`; `observation_x_val = data3`; `observation_y_val = data4`; `G_E_str_slip_val = data5`; `G_N_str_slip_val = data6`; `G_U_str_slip_val = data7`; `G_E_dip_slip_val = data8`; `G_N_dip_slip_val = data9`; `G_U_dip_slip_val = data10`; where: `DISLOC_X_VAL` is the location of the dislocation in x from the origin, `DISLOC_Y_VAL` is the location of the dislocation in y from the origin, `OBSERVATION_X_VAL` is the location of the observation points in x from the origin, `OBSERVATION_Y_VAL` is the location of the observation points in y from the origin, `G_E_STR_SLIP_VAL` is the effect of unit slip in the strike-slip direction on displacement in the east direction at the observation point, `G_N_STR_SLIP_VAL` is the effect of unit slip in the strike-slip direction on displacement in the north direction at the observation point, `G_U_STR_SLIP_VAL` is the effect of unit slip in the strike-slip direction on displacement in the up direction at the observation point, `G_E_DIP_SLIP_VAL` is the effect of unit slip in the dip-slip direction on displacement in the east direction at the observation point, `G_N_DIP_SLIP_VAL` is the effect of unit slip in the dip-slip direction on displacement in the north direction at the observation point, `G_U_DIP_SLIP_VAL` is the effect of unit slip in the dip-slip direction on displacement in the up direction at the observation point. The dislocation-station pairs are assumed to be in the order: `Disloc 1-Obs 1 Disloc 1-Obs 2 Disloc 1-Obs 3 ... Disloc 1-Obs n Disloc 2-Obs 1 Disloc 2-Obs 2 Disloc 2-Obs 3 ... Disloc j-Obs i Disloc j-Obs i+1 ... Disloc N-Obs n`, where `N` is the total number of dislocations (`Disloc|`) and n the total number of observation points on the surface (`Obs`). Example: `PCAIM_driver`. See also `PCAIM_driver`.

Input `load_greens_function(GreensFunctionFile)`

- `GreensFunctionFile`

Output `G = load_greens_function`

- `G`

m-File Summary for project_all_greens_fcn.m

File Name: project_all_greens_fcn.m File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: PROJECT_ALL_GREENS_FCN Project all Green's functions based on datatype. Projects the original Green Function matrix G, taking into account the type of data given in DATA_TYPE for both dense and sparse datasets. Futhermore, the sparse datasets are converted into sparse constraints for the inversion step. Example: PCAIM_driver. Also see project_greenfunctions, sparse_constraint_InSAR_calc, PCAIM_driver.

Input

`project_all_greens_fcn(G,all_position,X_dat,X_dat_sparse,data_type,data_type_sparse,data_info,data_info_sparse,S,V,X_time_index_sparse,n_comp)`

- G
- all_position
- X_dat
- X_dat_sparse
- data_type
- data_type_sparse
- data_info
- data_info_sparse
- S
- V
- X_time_index_sparse
- n_comp

Output

`[G_projected_dense,G_projected_sparse,sparse_constraint] = project_all_greens_fcn`

- G_projected_dense
- G_projected_sparse
- sparse_constraint

m-File Summary for project_greenfunctions.m

File Name: project_greenfunctions.m   File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: PROJECT_GREENFUNCTION Project ENU Green fcns onto the correct direction. G_PROJECTED = PROJECT_GREENFUNCTION(G,DATA_TYPE,LOS_FILE,G_INDEX_VECTOR) Projects the original Green Function matrix G, taking into account the type of data given in DATA_TYPE. The index vector G_INDEX_VECTOR{i} contained the indices corresponding to the dataset i. Example: nstat1 = 6; nstat2 = 13; npixel = 733; los_vector =rand(npixel,3); save('los.txt','los_vector','-ascii'); Gsar = rand(3*npixel,size(G,2)); Gcgps2=rand(3*nstat2,size(G,2)); Gcgps3 = rand(3*nstat1,size(G,2)); G = [Gcgps3; Gsar; Gcgps2]; data_type{1} = 'cGPS3'; los_file{1}=''; G_index_vector{1} = [1:3*nstat1]; data_type{2} = 'SAR'; los_file{2}='los.txt'; G_index_vector{2} = max(G_index_vector{1}) + 1:max(G_index_vector{1}) + 1 + 3*npixel; data_type{3} = 'cGPS2'; los_file{3} = ''; G_index_vector{3} = max(G_index_vector{2}) + 1:max(G_index_vector{2}) + 3*nstat2; Note: The G_index_vector can be also obtained directly using: G_index_vector = build_G_index_vector(G,position); G_projected = project_greenfunctions(Gnew,data_type,los_file,G_index_vector); G_projected{1}: Green function matrix relative to set 1 (cgps3); G_projected{2}: Green function matrix relative to set 2 (SAR); G_projected{3}: Green function matrix relative to set 3 (cgps2); See also PCAIM_driver.

Input

project_greenfunctions(G,data_type,los_file,G_index_vector)

- G

- data_type
- los_file
- G_index_vector

Output

`G_projected=project_greenfunctions`

- G_projected

m-File Summary for `build_smooth_surface.m`

File Name: `build_smooth_surface.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: build a smooth surface passing through a group of points of coordinates given by the vectors `x`, `y` and `z`. `N_nearest` is the number of neighbour used to compute the laplacian (e.g., `N_nearest = 5`) , the degree of smoothing being given by the parameter `lambda` (e.g., `lambda = 1000`). `interp_method` is a string describing the interpolation method used by `griddata` (e.g., `interp_method = 'cubic'`). See `griddata` documentation for more details (type `help griddata`).

Input

`build_smooth_surface(x,y,z,nx,ny,lambda,N_nearest,interp_method)`

- `x`
- `y`
- `z`
- `nx`
- `ny`
- `lambda`
- `N_nearest`
- `interp_method`

Output

`[xi,yi,zi]=build_smooth_surface`

- `xi`
- `yi`
- `zi`

m-File Summary for `find_string_in_cell.m`

File Name: `find_string_in_cell.m` File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: find for the string `xfind` within the cell `xcell`. `iflag = 1` if the string was encountered once, and `iflag = 0` otherwise. The corresponding index with the cell is given as `index`.

Input

`find_string_in_cell(xcell,xfind)`

- `xcell`
- `xfind`

Output

`[iflag,index]=find_string_in_cell`

- `iflag`
- `index`

m-File Summary for `find_triangle_param.m`

File Name: `find_triangle_param.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: Convention: find fault parameters knowing the coordinates of the three points `t1`, `t2` and `t3` defining the triangle. The tectonic vector `vect_tect` defines the direction towards which the fault moves and is used to estimate the rake of the triangle.

Input

`find_triangle_param(t1,t2,t3,vect_tect)`

- `t1`
- `t2`
- `t3`
- `vect_tect`

Output

`[xc,yc,zc,strike,dip,rake,area,vertices,strike_vect, updip_vect,normal_vect]=find_triangle_param`

- `(xc,yc,zc)`: coordinates of the center of the triangles.
- `strike,dip, rake, and area`: strike ,dip, rake, and area vector of the triangles.
- `vertices`: vertices of the triangle given by `[t1,t2,t3]`.
- `strike_vect`: along strike base vector for each triangle.
- `updip_vect`: updip base vector for each triangle.
- `normal_vect`: normal to the fault base vector for each triangle.

m-File Summary for `make_fault_model.m`

File Name: `make_fault_model.m` File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description:

Input

```
make_fault_model(input_file,outputfile_pcaim,
outputfile_okada,index2file,vect_tect,smooth_param,
angle,nx,ny,options_make_fault_model)
```

- `input_file` file containing the initial set of points of the smooth surface. When provided, `index2file(1)`, `index2file(2)` and `index2file(3)` are the column numbers of `input_file` corresponding to the east, north and updip coordinates. If not provided, `index2file` has a default value of `[1 2 3]` (i.e., `long`, `lat`, and `updip` is assumed). `input_file` can also be given directly as a 3d vector.
- `outputfile_pcaim` complete name (including path) of the output file corresponding to the fault model using the PCAIM code triangular format.
- `outputfile_okada` complete name (including path) of the output file corresponding to the fault model using Okada's format. This file is used for computation of Green functions. If `outputfile_pcaim` and `outputfile_okada` are leave empty, no output is written.
- `index2file`
- `vect_tect` 3d tectonic vector given in the geographical reference frame. Used to find the rake of the triangular patches.
- `smooth_param` value of the smoothing parameter (e.g., 1e2). The higher, the smoother the surface. When `smooth_param` $\rightarrow +\infty$, the smoothing surface reduces to a plane.
- `angle` rotation angle before meshing.
- `nx` number of triangles along the direction given by angle.
- `ny` number of triangles along the direction perpendicular to the direction given by angle.
- `options_make_fault_model` Possible options. ex: `options_make_fault_model = {'InterpMethod','v4','Nneighbour',10,'NaN'}`; 'InterpMethod': Possible interpolation methods are 'v4', 'cubic', 'linear', 'nearest'. Those options come from matlab routine `griddata` (type 'help griddata' for more details). 'NaN': Force to leave points with NaN coordinates, if existing. By default, make_fault_model removes those points. 'Nneighbour': Number of neighbours used by the function `compute_laplacian` (default value is 5). Option 'Nneighbour' should be followed by the number of neighbours.

Output

```
[xc,yc,zc,strike,dip,rake,area,vertices,strike_vect,
updip_vect,normal_vect]=make_fault_model
```

- xc
- yc
- zc
- strike
- dip
- rake
- area
- vertices
- strike_vect
- updip_vect
- normal_vect

m-File Summary for refine_fault_model_rectangle.m

File Name: refine_fault_model_rectangle.m
File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Build a finer rectangular fault model. The size of the mesh is dl_new (along strike direction) and dw_new (down-dip direction), with dl_new>dl and dw_new>dw, dl and dw being the initial rectangular mesh. The inputs are fault_param of size (n_rect,7), the 7 fault parameters being (x,y,z,strike,dip,dl,dw) for each of the n_rect rectangles. The output fault_param_new is the new fault model. See test_refine_fault_model_rectangle for an example. IMPORTANT: All the rectangles need to have the same size. If not, use this routine for every set of a given size.

Input

`refine_fault_model_rectangle(fault_param,dl_new,dw_new)`

- fault_param: Initial fault parameters.
- dl_new: New along strike length.
- dw_new: New down-dip length.

Output

`fault_param_new=refine_fault_model_rectangle`

- fault_param_new: New fault parameters.

m-File Summary for rotate2d_center_matrix.m

File Name: rotate2d_center_matrix.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: `[x1,y1] = rotate2d_center(x0,y0,xc,yc,angle)` Rotate a 2d point `(x0,y0)` of an angle `theta` (counterclockwise), the coordinates of rotation center being `(xc,yc)`. The rotated point coordinates are the ouput `(x1,y1)`.

Input

`rotate2d_center_matrix(x0,y0,xc,yc,theta)`

- x0
- y0
- xc
- yc
- theta

Output

`[x1,y1]=rotate2d_center_matrix`

- x1
- y1

m-File Summary for test_make_fault_model.m

File Name: test_make_fault_model.m File Type: script
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description:

Variables

9.4 General

These are general use functions that are used in many other functions.

m-File Summary for `add_trailing_slash.m`

File Name: `add_trailing_slash.m` File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: `ADD_TRAILING_SLASH` Adds a trailing file separation character to string. `DIRECTORY = ADD_TRAILING_SLASH(DIRECTORY)` checks whether the end character of the input string `DIRECTORY` is a file separation character ('/' or '\'), and if not it adds the machine-specific file separator. Example: `directory = '/home'`; directory = add_trailing_slash(directory); directory = 'C: \WINDOWS'; directory = add_trailing_slash(directory); See also `PCAIM_DRIVER`.

Input `add_trailing_slash(directory)`

- `directory`

Output `directory = add_trailing_slash`

- `directory`

m-File Summary for build_X_matrix.m

File Name: build_X_matrix.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: BUILD_X_MATRIX Build the full matrix of one of the X_* cell structures. This function builds the matrix X_matrix from the input cell structure X. X can be X_DAT, X_ERR, or X_WEIGHT. The missing entries are assumed to be zero, which means for X_ERR the user must post-process by switching zeros to Infs. Example: X_dat = sin([1,2,3,7,8,9;-1,-2,-3,-7,-8,-9]*pi/10),cos([4,5,6,7]*pi/10)*6; X_time_index = [1,2,3,7,8,9],[4,5,6,7]; X_data_matrix = build_X_matrix(X_dat,X_time_index). See also PCAIM_driver.

Input

`build_X_matrix(X,X_time_index)`

- X
- X_time_index

Output

`X_matrix = build_X_matrix`

- X_matrix

m-File Summary for `create_G_index_proj.m`

File Name: `create_G_index_proj.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `CREATE_G_INDEX_PROJ` Create row index of surface-projected Greens fcn. `[X_G_INDEX,X_G_INDEX_SPARSE] = CREATE_G_INDEX_PROJ(X_DAT,X_DAT_SPARSE)` Find row indexes to the surface-projected Green's functions based on the number of datasets in `X_DAT`, `X_DAT_SPARSE` and the number of time series per dataset. Functionality to similar to that of `CREATE_G_INDEX`, execpt that `CREATE_G_INDEX` worked on Green's functions that have `E`, `N`, `U` components of surface displacement regardless of observation type. Example: `PCAIM_driver`. See also `CREATE_G_INDEX`, `PCAIM_driver`.

Input

`create_G_index_proj(X_dat,X_dat_sparse)`

- `X_dat`
- `X_dat_sparse`

Output

`[X_G_index,X_G_index_sparse] = create_G_index_proj`

- `X_G_index`
- `X_G_index_sparse`

m-File Summary for create_G_index.m

File Name: create_G_index.m   File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: CREATE_G_INDEX Find an index to the rows of unprojected Green's functions. G_INDEX_VECTOR=CREATE_G_INDEX(G,POSITION) takes a POSITION cell, containing the lon,lat of each observation points and an unprojected (i.e. 3 directions per observation point) Greens function matrix "G" (used to check consitency between the dimension of G and POSITION but not really needed otherwise), send back a cell G_INDEX_VECTOR such that G_INDEX_VECTOR{i} is an index pointing towards the indices of G relative to the observation points of dataset i. Example: PCAIM_driver. See also get_fault_model, PCAIM_DRIVER.

Input

`create_G_index(G,position)`

- G
- position

Output

`G_index_vector = create_G_index`

- G_index_vector

m-File Summary for create_timeline.m

File Name: create_timeline.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: CREATE_TIMELINE Create a universal timeline out of cell of data epochs. [X_TIME_INDEX,TIMELINE]=CREATE_TIMELINE(X_TIME) takes the sets of epochs listed in the cells of X_TIME and combines them into a universal timeline containing all the dates precisely once and in chronological order. This universal timeline is store in TIMELINE, and X_TIME_INDEX indexes each data set in this new timeline. That is, for the ith dataset, TIMELINE(X_TIME_INDEX{i}) == X_TIME{i} is always true. Example: X_time = {unique(round(rand(1,10)*10)),unique(round(rand(1,10)*10))}; [X_time_index,timeline]=create_timeline(X_time). See also PCAIM_DRIVER.

Input

`create_timeline(X_time)`

- X_time

Output

`[X_time_index,timeline] = create_timeline`

- X_time_index
- timeline

m-File Summary for dist_fcn.m

File Name: `dist_fcn.m` File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: DIST_FCN Finds the Euclidean distance between two input parameters. Example: `x1 = [0,0,0,0]; x2 = [1,1,1,1]; dist = dist_fcn(x1,x2)`. See also `PCAIM_driver`.

Input

`dist_fcn(x1,x2)`

- `x1`
- `x2`

Output

`dist = dist_fcn`

- `dist`

m-File Summary for extract_from_cell.m

File Name: extract_from_cell.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: EXTRACT_FROM_CELL Extracts and concatenates matrices from a cell object. [CONCATENATED_CELL,CELL_INDEX] = EXTRACT_FROM_CELL(CELL_OBJ) extracts matrices from CELL_OBJ and attempts first to concatenate them vertically (i.e., [CELL_OBJ{1}; CELL_OBJ{2}; ...]). If the number of columns is not the same in each cell of CELL_OBJ, EXTRACT_FROM_CELL tries to concatenate them horizontally (i.e., [CELL_OBJ{1}, CELL_OBJ{2}, ...]). If this fails, EXTRACT_FROM_CELL will throw an error. Example: cell_obj = cell(4,1); for i = 1:4 cell_obji = rand(randi(5,1,1),3); end [concatenated_cell,cell_index] = extract_from_cell(cell_obj). See also PCAIM_DRIVER.

Input

`extract_from_cell(cell_obj)`

- cell_obj

Output

`[concatenated_cell,cell_index] = extract_from_cell`

- concatenated_cell
- cell_index

m-File Summary for find_disp_ratio.m

File Name: find_disp_ratio.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: FIND_DISP_RATIO Find the ratio of input and output time units. DISP_RATIO = FIND_DISP_RATIO(INPUT_DISP_UNIT,OUTPUT_DISP_UNIT) finds the correct multiplicative factor to convert between the two displacement units. Example: input_disp_unit = 'm'; output_disp_unit= 'mm'; disp_ratio=find_disp_ratio(input_disp_unit,output_disp_unit). See also find_time_ratio, PCAIM_driver.

Input

`find_disp_ratio(input_disp_unit,output_disp_unit)`

- input_disp_unit
- output_disp_unit

Output

`disp_ratio = find_disp_ratio`

- disp_ratio

m-File Summary for `find_index_ndim.m`

File Name: `find_index_ndim.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: `FIND_INDEX_NDIM` Finds the index_in th set of indexes for dimension `ndim`. `IND_RANGE=FIND_INDEX_NDIM(INDEX_IN,NDIM)` calculates the `index_in` th set of indexes, where each index corresponds to ndim entries. For example: `1:2 = FIND_INDEX_NDIM(1,2)`; `7:8 = FIND_INDEX_NDIM(4,2)`; `10:12 = FIND_INDEX_NDIM(4,3)`. Examples: `find_index_ndim(1,2)`, `find_index_ndim(4,2)`, `find_index_ndim(4,3)`. See also `PCAIM_driver`.

Input

`find_index_ndim(index_in,ndim)`

- `index_in`
- `ndim`

Output

`ind_range = find_index_ndim`

- `ind_range`

m-File Summary for `find_time_ratio.m`

File Name: `find_time_ratio.m` File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `FIND_TIME_RATIO` Find the ratio of input and output time units. `TIME_RATIO=FIND_TIME_RATIO(INPUT_TIME_UNIT,OUTPUT_TIME_UNIT)` finds the correct multiplicative factor to convert between the two time units. Example: `input_time_unit = 'day'; output_time_unit= 'year'; time_ratio=find_time_ratio(input_time_unit,output_time_unit).` See also `find_disp_ratio`, `PCAIM_driver`.

Input

`find_time_ratio(input_time_unit,output_time_unit)`

- `input_time_unit`
- `output_time_unit`

Output

`time_ratio = find_time_ratio`

- `time_ratio`

m-File Summary for get_date_string.m

File Name: get_date_string.m File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: `GET_DATE_STRING` Transform current time into a string. The precision of the time is given in the input `format_string` as: `'yr'`, `'month'`, `'day'`, `'hour'`, `'min'`, and `'sec'`. Example: `get_date_string('min')`, gives `'2010_1_12_16_32'`, when used the 12th of january 2010, at 4:32 pm.

Input

`get_date_string(format_string)`

- `format_string`: `'yr'`, `'month'`, `'day'`, `'hour'`, or `'min'`.

Output

`current_time_name = get_date_string`

- `current_time_name`: String of the form `'year_month_day_hour_minute'`.

m-File Summary for `llh2localxy.m`

File Name: `llh2localxy.m`
File Type: function
Author: Unknown (Peter Cervelli?)
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description:

Input

`llh2localxy(llh,ll_org)`

- `llh`
- `ll_org`

Output

`[xy] = llh2localxy`

- `[xy]`

m-File Summary for `local2llh.m`

File Name:	`local2llh.m`	File Type: function
Author:	Peter Cervelli	
Maintainer:	Hugo Perfettini	
Contact E-mail:	`pcaim@gps.caltech.edu`	
Version:	1.0.0.0	
File Description:	`llh = local2llh(xy,origin)` Converts from local coorindates to longitude and latitude given the `[lon, lat]` of an origin. `origin` should be in decimal degrees. Note that heights are ignored and that `xy` is in km. Output is `[lon, lat, height]` in decimal degrees. This is an iterative solution for the inverse of a polyconic projection.	

Input

`local2llh(xy,origin)`

- `xy`
- `origin`

Output

`llh = local2llh`

- `llh`

m-File Summary for n_entries_calc.m

File Name: n_entries_calc.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: `N_ENTRIES_CALC` Calculates number of elements in each cell of input var. `N_ENTRIES = N_ENTRIES_CALC(X_TIME_INDEX)` returns a `numel(X_TIME_INDEX)` vector filled with the number of elements in each cell of the input cell structure, `X_TIME_INDEX`. Example: `X_time_index = {[1,2,3],[1,4,5,6],[4,5,7]}; n_entries = n_entries_calc(X_time_index).` See also `n_epochs_calc`, `n_entries_edge_calc`, `n_tseries_calc`, `PCAIM_driver`.

Input

`n_entries_calc(X_time_index)`

- `X_time_index`

Output

`n_entries = n_entries_calc`

- `n_entries`

m-File Summary for `n_entries_edge_calc.m`

File Name: `n_entries_edge_calc.m` File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: `N_ENTRIES_EDGE_CALC` Calculates the edges of an integer entries vector. `N_ENTRIES` is a vector of integers corresponding to the number of rows or columns to be filled in a large matrix. `N_ENTRIES_START` and `N_ENTRIES_FINISH` are the first and last indexes of each of the subcomponents of the matrix to be filled. Example: `n_entries = [1;5;3]; [n_entries_start,n_entries_finish] = n_entries_edge_calc(n_entries)`. See also `N_ENTRIES_CALC`, `N_EPOCHS_CALC`, `N_TSERIES_CALC`.

Input `n_entries_edge_calc(n_entries)`

- `n_entries`: a vector of integers corresponding to the number of rows or columns to be filled in a large matrix.

Output `[n_entries_start,n_entries_finish] = n_entries_edge_calc`

- `n_entries_start`: the first indexes of each of the subcomponents of the matrix to be filled.
- `n_entries_finish`: the last indexes of each of the subcomponents of the matrix to be filled.

m-File Summary for n_epochs_calc.m

File Name: n_epochs_calc.m File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: N_EPOCHS_CALC Calculate the total number of epochs in all datasets. [N_EPOCHS, UNIQUE_EPOCHS] = N_EPOCHS_CALC(X_TIME_INDEX) computes the number of unique epochs in cells of X_TIME_INDEX, a cell structure containing vectors of integers corresponding to data acquisition epochs. Example: X_time_index = {[1,2,3];[2,3,4,5,6,7];[9,10]}; [n_epochs, unique_epochs] = n_epochs_calc(X_time_index). See also n_entries_calc, n_entries_edge_calc, n_tseries_calc, PCAIM_driver.

Input

n_epochs_calc(X_time_index)

- X_time_index

Output

[n_epochs, unique_epochs] = n_epochs_calc

- n_epochs
- unique_epochs

m-File Summary for n_tseries_calc.m

File Name: n_tseries_calc.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: N_TSERIES_CALC Compute the number of time series per dataset. [N_TSERIES,N_TSERIES_VEC] = N_TSERIES_CALC(X_DAT) computes the number of time series in input datasets contain in X_DAT. N_TSERIES is the total number of time series between all datasets and N_TSERIES_VEC is a vector with the number of time series in each dataset as its elements. Example: X_dat = [1,2,3,4;4,5,6,7],[1,1,1,17,6,3]; [n_tseries,n_tseries_vec] = n_tseries_calc(X_dat). See also n_epochs_calc, n_entries_calc, n_entries_edge_calc, PCAIM_driver.

Input

`n_tseries_calc(X_dat)`

- X_dat

Output

`[n_tseries,n_tseries_vec] = n_tseries_calc`

- n_tseries: the total number of time series between all datasets.
- n_tseries_vec: a vector with the number of time series in each dataset as its elements.

m-File Summary for polyconic.m

File Name: polyconic.m
File Type: function
Author: Unknown (Peter Cervelli?)
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Polyconic Projection.

Input

`polyconic(Lat, Diff_long, Lat_Orig)`

- `Lat`: Latitude (decimal seconds).
- `Diff_long`: Differential Longitude (decimal seconds) relative to Central Meridian.
- `Lat_Orig`: Latitude of Origin (decimal seconds).

Output

`[xy] = polyconic`

- `x`: Distance from Central Meridian.
- `u`: Distance from Origin to Latitude.

m-File Summary for remove_char.m

File Name: remove_char.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: REMOVE_CHAR Remove leading and trailing strings from string. REMOVE_CHAR(DIRTY_STRING,STRING_TO_REMOVE) removes leading and trailing copies of STRING_TO_REMOVE from DIRTY_STRING and returns the cleaned string as CLEAN_STRING. Example: dirty_string = ' Arthur could not find his towel. '; string_to_remove = ' '; clean_string=remove_char(dirty_string,string_to_remove). See also PCAIM_driver.

Input

remove_char(dirty_string,string_to_remove)

- dirty_string
- string_to_remove

Output

clean_string=remove_char

- clean_string

m-File Summary for rotate2d_center.m

File Name: rotate2d_center.m | File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: ROTATE2D_CENTER Rotate point in a Cartesian plane about a given origin. [X1,Y1]=ROTATE2D_CENTER(X0,Y0,XC,YC,ANGLE) rotates the points with x-coordinates in X0 and y-coordinates in Y0 by an amount ANGLE given in degrees about central point (XC,YC). The output points are in the same order as the input points with x-coordinates in X1, and y-coordinates in Y1. Example: x0 = [2,1]; y0 = [0,1]; xc = 1; yc = 0; theta = 90; [x1,y1]=rotate2d_center(x0,y0,xc,yc,theta). See also polyconic, llh2localxy, local2llh.

Input

rotate2d_center(x0,y0,xc,yc,theta)

- x0
- y0
- xc
- yc
- theta

Output

[x1,y1]=rotate2d_center

- x1
- y1

m-File Summary for `save_results.m`

File Name: `save_results.m` File Type: script
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: Save the work space and variables given by the user. See script `test_save_results` for an example. You can load the files using `load(file_name,format)`, where `format` is for instance `'-mat'` (matlab binary format), or `'-ascii'` (`ascii` format).

Input `save_results`

options are:

- `'Name',name:` Option to give the name `name` of the .mat output file where the environnement is saved. Default is `'date_time'`. This option needs to be immediately followed by the name of the file given as a string.
- `'Directory',directory_name:` Option to give the directory `directory_name` of the `.mat` output file. Default is `'temp_pcaim'`. This option needs to be immediately followed by the name of the directory given as a string.
- `'Variables2Save',Variables:` Option to save some specific variables given in the cell of strings `Variables`. This option needs to be followed immediately with a cell of strings, each of them containg the variables the user want to save.
- `'Format',format:` Format of the saved variable files given in `format`. See matlab `save` command for possible format (e.g., `'Format','-ascii'`).

Output `NONE`

m-File Summary for set_default_value.m

File Name: set_default_value.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: SET_DEFAULT_VALUE Announces the default value of string will be set. VAR = SET_DEFAULT_VALUE(STRING,VALUE) displays to the user than STRING is going to be set. If VALUE is a numeric object, its value is also printed. Example: string= 'n_comp'; value = 1; var = set_default_value(string,value). Also see decomp_srebro_EM, decomp_srebro_CG_simultaneous.

Input

`set_default_value(string,value)`

- string
- value

Output

`var = set_default_value`

- var

m-File Summary for w_mean.m

File Name: w_mean.m

File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version:

File Description: W_MEAN Compute the weighted mean of a data matrix given a weight matrix. MEANS = W_MEAN(DATA,WEIGHT) computes the weighted mean of each row of the matrix DATA according to the individual weights from the matrix WEIGHT. Note that size(DATA) == size(WEIGHT). MEANS = W_MEAN(DATA,WEIGHT,ERROR_FLAG) computes the weighted mean of each row of the matrix DATA according to the individual weights from the matrix 1./ABS(WEIGHT)^2 if ERROR_FLAG == 1. If ERROR_FLAG = 1, then this syntax is the same as ommitting ERROR_FLAG. Example: data = [0,0,0,1,1,1,2,2,2;... 0,1,2,3,4,5,6,7,8]; weight = [4,4,4,4,4,4,4,4,4;... 400,4,4,4,4,4,4,4,4]; error = 1./sqrt(weight); means_weight = w_mean(data,weight) means_error = w_mean(data,error,1).

Input

`w_mean(data,weight,error_flag)`

- data
- weight
- error_flag

Output

`means = w_mean`

- means

9.5 Inversions

These scripts perform or support inversion on the decomposed data.

m-File Summary for `fnnls.m`

File Name: `fnnls.m` File Type: function
Author: L. Shure
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version:
File Description: `FNNLS` Non-negative least-squares. Adapted from `NNLS` of Mathworks, Inc. `x = fnnls(XtX,Xty)` returns the vector `X` that solves `x = pinv(XtX)*Xty` in a least squares sense, subject to `x >= 0`. Differently stated it solves the problem `min ||y - Xx||` if `XtX = X'*X` and `Xty = X'*y`. A default tolerance of `TOL = MAX(SIZE(XtX)) * NORM(XtX,1) * EPS` is used for deciding when elements of x are less than zero. This can be overridden with `x = fnnls(XtX,Xty,TOL)`. `[x,w] = fnnls(XtX,Xty)` also returns dual vector `w` where `w(i) < 0` where `x(i) = 0` and `w(i) = 0` where `x(i) > 0`. See also `NNLS` and FNNLSb. L. Shure 5-8-87. Revised, 12-15-88,8-31-89 LS. (Partly) Copyright (c) 1984-94 by The MathWorks, Inc. Modified by R. Bro 5-7-96 according to Bro R., de Jong S., Journal of Chemometrics, 1997, xx. Corresponds to the `FNNLSa` algorithm in the paper http://newton.foodsci.kvl.dk/rasmus.html. Modified by S. Gunn 20-9-97. Reference: Lawson and Hanson, "Solving Least Squares Problems", Prentice-Hall, 1974.

Input

`fnnls(XtX,Xty,tol)`

- `XtX`
- `Xty`
- `tol`

Output

`[x,w] = fnnls`

- `x`
- `w`

m-File Summary for inversion_type.m

File Name: inversion_type.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: INVERSION_TYPE Perform the actual inversion operation given options. S = inversion_type(D,A,N_PATCHES,N_COMP,INVERSION_OPT) gives the best solution to the inverse problem A*S = D given the number of components N_COMP and the options in INVERSION_OPT. Example: PCAIM_driver. See also INVERT_COMPONENTS, PCAIM_DRIVER.

Input

inversion_type(d,A,n_patches,n_comp,inversion_opt)

- d
- A
- n_patches
- n_comp
- inversion_opt

Output

s=inversion_type

- s

m-File Summary for invert_components.m

File Name: invert_components.m File Type: function
Author: Andrew Kositsky and Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: INVERT_COMPONENTS Set up and execute the inversion for slip at depth. L = INVERT_COMPONENTS(U,G_PROJECTED_DENSE,GAMMA,LAP,N_COMP,OPTIONS) finds a possibly constrained, regularized least-squares solution to inverting U for dislocation at depth with Green's functions G_projected_dense. Sparse constraints can be added in options. GAMMA is the smoothing of the Laplacian LAP, and N_COMP is the number of components in the model. Example: PCAIM_driver. See also PCAIM_driver.

Input

`invert_components(U,G_projected_dense,gamma,Lap,n_comp, options)`

- U
- G_projected_dense
- gamma
- Lap
- n_comp
- options

Output

`L=invert_components`

- L

m-File Summary for optimize_offsets_final.m

File Name: optimize_offsets_final.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: OPTIMIZE_OFFSETS_FINAL Take a dislocation model and optimize offsets. [X_DAT,FINAL_OFFSETS] = OPTIMIZE_OFFSETS_FINAL(X_DAT,X_ERR,G,L,S,V,X_TIME_INDEX) finds the constant offsets for each timeseries required to minimize final model chi-squared. The amount of these offsets is FINAL_OFFSETS, and X_DAT is returned with these offsets subtracted. Example: PCAIM_driver. See also invert_components, create_G_index_all, PCAIM_driver.

Input

optimize_offsets_final(X_dat,X_err,G_projected_dense,L,S,V,X_time_index)

- X_dat
- X_err
- G_projected_dense
- L
- S
- V
- X_time_index

Output

[X_dat,final_offsets] = optimize_offsets_final

- X_dat
- final_offsets

m-File Summary for sparse_constraint_InSAR_calc.m

File Name: sparse_constraint_InSAR_calc.m File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: SPARSE_CONSTRANT_INSAR_CALC Find the sparse constraint matrix for the inversion step. SPARSE_CONSTRAINT = SPARSE_CONSTRANT_INSAR_CALC(G_InSAG_INSAR, S, V, INSAR_TIME_INDEX,N_COMP) currently takes the Greens function G_INSAR for a single InSAR image and combines it with S and V at the times from INSAR_TIME_INDEX to give a linear equation for the surface displacement of the InSAR image given our N_COMP model. Example: PCAIM_driver. See also PCAIM_driver.

Input

`sparse_constraint_InSAR_calc(G_InSAR, S, V, InSAR_time_index,n_comp)`

- G_InSAR
- S
- V
- InSAR_time_index
- n_comp

Output

`sparse_constraint = sparse_constraint_InSAR_calc`

- sparse_constraint

9.6 Plotting and Statistics

These scripts perform plotting commands or compute statistics relating to the model.

m-File Summary for `build_slip_vectors.m`

File Name:	`build_slip_vectors.m` File Type: function
Author:	Hugo Perfettini
Maintainer:	Hugo Perfettini
Contact E-mail:	`pcaim@gps.caltech.edu`
Version:	1.0.0.0
File Description:	Take the `slip` vector on the local fault axes, and transform it as a unit vector in the geographical `(east,north,up)` reference frame.

Input `build_slip_vectors(U_str,U_updip,fault_model)`

- `U_str`
- `U_updip`
- `fault_model`

Output `slip_vector=build_slip_vectors`

- `slip_vector`

m-File Summary for `change_xlim_ylim.m`

File Name: `change_xlim_ylim.m` File Type: function

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: `CHANGE_XLIM_YLIM` Increases the `xlim/ylim` of current plot to include `x`, `y`. `CHANGE_XLIM_YLIM(X,Y)` sets `xlim` to `(min([X,xlim]),max([X,xlim]))` and `ylim` to `(min([Y,ylim]),max([Y,ylim]))`.

Input

`change_xlim_ylim(x,y)`

- `x`
- `y`

Output

`NONE`

m-File Summary for chi2_calc_all.m

File Name: chi2_calc_all.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description:

Input

`chi2_calc_all(X_dat,X_err,X_model,X_dat_sparse,X_err_sparse,X_model_sparse)`

- X_dat
- X_err
- X_model
- X_dat_sparse
- X_err_sparse
- X_model_sparse

Output

`[chi2,chi2_dense,chi2_sparse] = chi2_calc_all`

- chi2
- chi2_dense
- chi2_sparse

m-File Summary for chi2_calc.m

File Name: chi2_calc.m
File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description:

Input

`chi2_calc(X_data_matrix,X_error_matrix,X_model_matrix)`

- `X_data_matrix`
- `X_error_matrix`
- `X_model_matrix`

Output

`[chi2] = chi2_calc`

- `chi2`

m-File Summary for create_predictions.m

File Name: create_predictions.m File Type: function
Author: Andrew Kositsky
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: CREATE_PREDICTIONS Convert an inversion into a dislocation model [X_PRED,X_PRED_SPARSE,X_MODEL,X_MODEL_SPARSE,DISLOC_CUM] = CREATE_PREDICTIONS(G,L,S,V,X_TIME_INDEX,X_TIME_INDEX_SPARSE, X_G_INDEX,X_G_INDEX_SPARSE) converts the output from INVERT_COMPONENTS and translates it into

1. predictions (X_PRED, X_PRED_SPARSE) for each data set and observation location for each epoch regardless of whether data was sampled there at that time (i.e. it predicts values even where we don't have data),

2. modeled values (X_MODEL, X_MODEL_SPARSE) for each data set and observation location at exactly the same epochs as the original dataset (i.e. it models measurement values only where we really have entries in X_DAT(_SPARSE) so X_MODEL(_SPARSE) the same size as X_DAT(_SPARSE)), and

3. cumulative dislocation (DISLOC_CUM) for the model, with the first of dislocation assigned to have the value zero.

Example: PCAIM_driver. See also INVERT_COMPONENTS, PCAIM_DRIVER.

Input

create_predictions(X_dat,X_dat_sparse,G_projected_dense,G_projected_sparse,L,S,V,X_time_index,X_time_index_sparse)

- X_dat
- X_dat_sparse
- G_projected_dense
- G_projected_sparse
- L
- S
- V

- X_time_index
- X_time_index_sparse

Output

[X_pred,X_pred_sparse,X_model,X_model_sparse,slip_cum] = create_predictions

- X_pred
- X_pred_sparse
- X_model
- X_model_sparse
- slip_cum

m-File Summary for model_statistics.m

File Name:	model_statistics.m	File Type: script
Author:	Andrew Kositsky	
Maintainer:	Hugo Perfettini	
Contact E-mail:	pcaim@gps.caltech.edu	
Version:	1.0.0.0	
File Description:		

Variables

m-File Summary for `plot_coast.m`

File Name: `plot_coast.m` File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: plot coast file from `http://rimmer.ngdc.noaa.gov/mgg/coast/getcoast.html` (before download, make sure to choose "Matlab" in Coast Format options).

Input

`plot_coast(fault_model,origin,coast_file_name,options)`

- `fault_model`
- `origin`
- `coast_file_name`
- `options`

Output

`NONE`

m-File Summary for `plot_edge.m`

File Name: `plot_edge.m` File Type: script
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description:

Variables

m-File Summary for plot_field.m

File Name: plot_field.m — File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: plot the field given in `field`, using symbols and the fault model given in `fault_model`. The plot is centered at the origin given by `origin`, and unit is `km`. options are listed below. See also the script `test_plot_field` for an example.

Input

`plot_field(fault_model,origin,field,options)`

options:

- `'Perspective',view_angles:` view angles is a 2D vector `[azimuth,elevation]`. Default values are `[0,90]` (equivalent to `view(2)`). Type `help view` to get more details.
- `'ColorMap',mycolormap:` will use the colormap `mycolormap`.
- `'ColorScale',Type:` set the color scale. If `Type='Auto'`, auto scale is assumed. In case `Type=[min_field,max_field]`, `field` will be plotted in the bound range `[min_field,max_field]`.
- `'FieldVector',field_vector,vector_scale,vector_color,vector_width:` `field_vector` is a 3*size(field,1) matrix (a 3D vector for each element of field). `vector_scale` is a scaling factor (change the length of the vectors), default value being 1. `vector_color` gives the color of the vector in matlab size. `vector_width` sets the width of the vectors. Note that all those options have to be given if the option `'FieldVector'` is active.
- `'ColorBar':` display the color bar.
- `'ColorBarLabel',label:` display the string `label` along the color bar axis (ex: `label='slip (cm)'`)
- `'MarkerArea',marker_area:` Set marker area. This is needed when calling function scatter3. Default value is 200.
- `'AutoScale',auto_scale_factor:` auto scale the figure using the bounds `auto_scale_factor*[min(x),max(x),min(y),max(y)]`, `(x,y)` being the coordinates of the points.
- `'PatchSymbol',symbol_type:` plot the points using symbols given in `symbol_type` (color and type). Ex: `'PatchSymbol','ko'` to plot black circles.
- `'SymbolSize',symbol_size:` set the size of the symbols to `symbol_size`.

Output

`NONE`

m-File Summary for `plot_field_patches_rectangular.m`

File Name: `plot_field_patches_rectangular.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description: plot the field given in `field`, using rectangular patches and the fault model given in `fault_model`. The plot is centered at the origin given by `origin`, and unit is km. options are listed below. See also the script `test_plot_patches_rectangular` for an example.

Input

`plot_field_patches_rectangular(fault_model,origin,field,options)`

options are identical to `plot_field`. The only additional option is:

- `'Shading',shading_type`: Set the shading used by patch (`shading_type='Faceted','Flat'`, or `'Interp'`). Type `help patch` for more details.

Output

`NONE`

m-File Summary for plot_field_patches_point_source.m

File Name: plot_field_patches_point_source.m
File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: plot the field given in `field`, using triangular patches and the fault model given in `fault_model`. The plot is centered at the origin given by `origin`, and unit is km. options are listed below. See also the script `test_plot_patches_point_source` for an example.

Input

`plot_field_patches_point_source(fault_model,origin,field,options)`

options are identical to `plot_field_patches_rectangular`.

Output

NONE

m-File Summary for plot_gps_stations.m

File Name: plot_gps_stations.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: plot gps stations at (long_dense, lat_dense). ex: options_plot_gps = {'MarkerStyle', '^m', 'MarkerSize', 10, 'Name', name, [5,5]}; vectors. 'MarkerStyle': Style (color and symbol type) used to plot the GPS stations. 'MarkerSize': Size of the marker. If the option 'Name' is given, the name of the stations given in the string vector name will be plotted, and will be offset (in km) from the position of the stations (long_dense, lat_dense) by the amount dx and dy (dx: east offset, dy:north offset).

Input `plot_gps_stations(long_dense,lat_dense,origin,options_plot_gps)`

- `long_dense`:
- `lat_dense`:
- `origin`: the center of the plot.
- `options_plot_gps`: options for plot_gps_stations.

Output `NONE`

m-File Summary for `plot_gps_vectors.m`

File Name: `plot_gps_vectors.m` File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: `pcaim@gps.caltech.edu`
Version: 1.0.0.0
File Description: plot displacement vectors at `(long_dense,lat_dense)` for the model `X_model` and the data `X_dat`. ex: `options_plot_gps_vectors = {'VectorScale',2,2,'ColorModel','b','g','ColorData','r','m'};` `'VectorScale'`: scale for the horizontal and vertical (in this very order) vectors. `'ColorModel'`: colors of the modeled horizontal and vertical (in this order) displacements vectors. `'ColorData'`: colors of the horizontal and vertical (in this order) data displacements vectors.

Input

`plot_gps_vectors(long_dense,lat_dense,X_model,X_dat,data_type_string,origin,options_plot_gps_vectors)`

- `long_dense`
- `lat_dense`
- `X_model`
- `X_dat`
- `data_type_string`
- `origin`: the center of the plot.
- `options_plot_gps_vectors`: options for plot_gps_vectors.

Output

`NONE`

m-File Summary for `plot_L.m`

File Name: `plot_L.m` File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: `pcaim@gps.caltech.edu`

Version: 1.0.0.0

File Description:

Input

`plot_L(L,ncomp,fault_model,origin,options_plot_L,coast_file_name,options_plot_coast)`

- `L`
- `ncomp`
- `fault_model`
- `origin`
- `options_plot_L`
- `coast_file_name`
- `options_plot_coast`

Output

`NONE`

m-File Summary for plot_labeled_points.m

File Name: plot_labeled_points.m File Type: script
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description:

Variables

m-File Summary for plot_model.m

File Name: plot_model.m

File Type: script

Author: Andrew Kositsky

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description:

Variables

m-File Summary for plot_SAR.m

File Name: plot_SAR.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: plot SAR displacement (towards satellite) at (long_sparse,lat_sparse) for the model SAR_model and the data SAR_data. ex: options_sparse = {'Marker','o','LineWidth',2,'SizeData',200,'PlotType', 'Absolute','MarkerArea',400}; 'Marker': Type of Marker for the modeled points, the data being displayed as points within the modeled points. 'LineWidth': width ot the modeled symbol edge. 'SizeData': Size of the symbol used to plot the model. 'PlotType': 'Absolute' plots both the model and the data, while 'Residual' plots the difference between data and model (in this precise order). 'MarkerArea': Area of the marker used to plot data (the larger, the bigger).

Input

`plot_SAR(long_sparse,lat_sparse,SAR_model,SAR_data, origin,options_sparse)`

- long_sparse
- lat_sparse
- SAR_model
- SAR_data
- origin: the center of the plot.
- options_sparse: options for plot_SAR.

Output

NONE

m-File Summary for plot_time_series.m

File Name: plot_time_series.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: plot the time series (data + model) with error bars. `Xtime` is the time vector, `Xmodel` the prediction of the model, `Xdata`, the original data with their errors given in `Xerror`. Displacements are given in the (east,north,up) reference frame. The model will be plotted as a continuous lines, while the data will be plotted using error bars. options_plot_time_series: ex: `options_plot_time_series = {'Grid', 'Name', name, 'DispUnit',observation_unit, 'TimeUnit', time_unit, 'Bounds', [tmin,tmax], 'ModelPlotStyle', 'b-', 'LineWidth', 2, 'DataPlotStyle', 'ms'};` `'Grid'`: displays a grid (default = no grid). `'ModelPlotStyle'`: this option allows the user to change the line style (symbol, color, line style) of the model. It needs to be followed by a string such as `'ob-'`. Default value is `'b.-'` (blue dots connected with a continous line). `'LineWidth'`: line width of the model plot. The line width needs to be immediately given after `'LineWidth'`. Default value is 1. `'DataPlotStyle'`: this option allows the user to change the plotting style (symbol and color) of the data, plotted as error bars. It needs to be followed by a string such as `'mo'`. Default value is `'ro'` (red circles). `'TimeUnit'`: to plot the time vector with its proper unit, given by the field immediately following `'TimeUnit'`. Any string will be plotted as it is, meaning that you could use `'year'`, `'yr'`, `'day'`, `'s'`, `'second'`, `'century'`, `'Bounds'`, `[tmin,tmax]`: to specify the limit of the time axis, an autoscale being done on the vertical axis (displacements) using the data (and not the model). If this option is empty, autoscale is performed.

Input

`plot_time_series(Xtime,Xmodel,Xdata,Xerror,options_plot_time_series)`

- `Xtime`
- `Xmodel`
- `Xdata`

- Xerror
- options_plot_time_series

Output NONE

m-File Summary for plot_V.m

File Name: plot_V.m File Type: function
Author: Hugo Perfettini
Maintainer: Hugo Perfettini
Contact E-mail: pcaim@gps.caltech.edu
Version: 1.0.0.0
File Description: Plot the first ncomp eigenvectors V of the PCAIM decomposition. options_plot_V: ex: options_plot_V = {'Grid','Time',X_time{1},'TimeUnit',time_unit,'LineWidth', 2,'LineStyle','o-r'}; 'Grid': displays a grid (default=no grid). 'Time': to plot a given time vector (default is the index of the V's). After time, the user needs to give the time vector, which size is supposed to be compatible with the size of V (e.g., size(V,1) = numel(time_vector). 'LineStyle': to change LineStyle, i.e., symbols, lines, colors. This option needs to be immediately followed by a string giving the line style (ex: 'ko-'). Same syntax as matlab plot. Default values are 'bo-' (blue circles connected by a continuous line). 'LineWidth': to change the width of the lines. The next entry needs to be the line width (ex: 2.5). Default value is 1. 'TimeUnit': to plot the time vector with its proper unit, given by the field immediately following 'TimeUnit'. Any string will be plotted as it is, meaning that you could use 'year', 'yr', 'day', 's', 'second', 'century',...

Input

plot_V(V,ncomp,options_plot_V)

- V
- ncomp
- options_plot_V

Output

NONE

m-File Summary for slip_potency_calc.m

File Name: slip_potency_calc.m   File Type: function

Author: Hugo Perfettini

Maintainer: Hugo Perfettini

Contact E-mail: pcaim@gps.caltech.edu

Version: 1.0.0.0

File Description: compute the slip potency as a function of time `slip_pot(t) = Sum_{j=1,npatch}(|slip(j,t)| x area(j))`.

Input

`slip_potency_calc(slip,area)`

- slip
- area

Output

`slip_pot=slip_potency_calc`

- slip_pot

9.7 Testing Scripts

There are a number of testing scripts for easily probing the interaction of scripts and the input variables. These are not be documented here.

Chapter 10

Variables

- `A` The design matrix for inversion.
- `all_position` cell array containing all positions of the observation points on the surface.
- `alpha` 2nd order coefficient in the step-size for the exact conjugate gradient method for χ^2 calculations.
- `angle` rotation angle before meshing.
- `area` area of a patch in km^2.
- `basic` centering method that uses the weighted mean.
- `beta` 1st order coefficient in the step-size for the exact conjugate gradient method for χ^2 calculations.
- `cell_index`
- `G_index_vector`
- `cell_obj`
- `center_function`
- `cGPS3_stations` is a cell structure of strings listing the allowed stations. Case sensitive.
- `chi2_dense` vector containing contribution to the χ^2 from each dense dataset in a separate element.
- `chi2_modified` calculation of chi^2 using `X_weight` instead of `X_error`.
- `chi2_sparse` vector containing contribution to the χ^2 from each sparse dataset in a separate element.
- `chi2` χ^2 of a dataset or of entire scenario.
- `clean_string` a returned string with the desire characters removed.

• `coast_file_name`

• `concatenated_cell` the entries of a cell structure concatenated to form a giant matrix or vector.

• `d` data vector for solving the linear inversion problem.

• `data_file:` full path of file containing dataset information locations.

• `data_info_sparse` information about the sparse datasets (see `data_info{i}`).

• `data_info{i}:` cell containing informations about dataset i. For cGPS: data_info{i}{1}{j}: name of station j within dataset i data_info{i}{2}{j}: path of gps file of station j within dataset i.

• `data_type_sparse` data type of about the sparse datasets (see `data_type{i}`).

• `data_type_string` data type of a dataset as a string rather than a cell.

• `data_type{i}:` type of data considered (e.g., cGPS3, SAR,...) in cell format.

• `data` data matrix of which we desire to find the means.

• `date_output:` same as the input_date but in decimal years.

• `date:` a date in the format: `YYYY/MM/DD` <`seperater`> `HH:MI:SS` where `YYYY` is the year, `MM` is the month, `DD` is the day, `HH` is the hour, `MI` is the minute, `SS` is the decimal seconds (arbitrary number of digits after the first two if decimal is needed.)

• `decomp_function` the function to be used for decomposition

• `decomp_options` the options to be used for decomposition

• `dfunc_options` the options to be used for the derivative of the objective function during the conjugate gradient algorithm.

• `dfunc` name of the derivative of the objective function to be used

• `Diff_long:` Differential Longitude (decimal seconds) relative to Central Meridian.

• `dip` dip angle of a fault element.

• `directory` a directory path.

• `dirty_string` string with undesired characters still present.

• `disp_ratio` ratio of the input (current) to output (desired) units for displacement.

• `elapsed_time` total time elapsed since the beginning of a decomposition step.

• `error_flag` a Boolean flag to designate if there is an `X_err` matrix as an input.

• `F` current function value during the conjugate gradient method scripts.

• `fault_model` a large matrix that completely describes the current fault model. Defined in `get_fault_model.m`

• `field` a vector field for plotting.

• `final_offsets` the final constant offsets for each time-series to minimize the unexplained χ^2.

• `first_epoch` is a scalar denoting the first allowed epoch in the timeseries.

• `fprime_out` the derivative of the objective function either in the original basis or orthogonal basis (if applicable).

- `ftol` the function tolerance for the conjugate gradient algorithm.
- `func_options` options for the objective function during the conjugate gradient algorithm.
- `func` objective function used during the conjugate gradient algorithm.
- `G_index_vector` cell array that indexes `G` based on the size of the original input datasets in `X_dat`.
- `G_InSAR` the Green's functions for an InSAR image.
- `G_projected_sparse` the projected Green's functions for a sparse constraint.
- `G` all of the unprojected Green's functions.
- `gamma` 0th order coefficient in the step-size for the exact conjugate gradient method for χ^2 calculations.
- `GreensExternalFcnDir` path to the binary of the Green's function to be used.
- `iedge` indexes of the "no slip edge patches" on a fault surface.
- `iflag` Boolean flag to indicate whether or not a string has been found.
- `index_in` number of n-tuples in we wish to go for calculating indexes.
- `index` general indexing variable.
- `index2file` conversion vector from an input format matrix to the internal format matrix (switches columns).
- `input_disp_unit` displacement unit of the input data source.
- `input_file` file containing the initial set of points of the smooth surface. When provided, `index2file(1)`, `index2file(2)` and `index2file(3)` are the column numbers of `input_file` corresponding to the east, north and updip coordinates. If not provided, `index2file` has a default value of `[1 2 3]` (i.e., `long`, `lat`, and `updip` is assumed). `input_file` can also be given directly as a 3d vector.
- `input_list_file` is a string containing the absolute path of the file containing a list of information on all the data input sources for this scenario.
- `input_list:` correctly formatted string from the data_file of the previous script. Format is: Dataset Name | Data Type | Path/To/Dataset/File | Time Unit | Length Unit.
- `InSAR_time_index` time indexes for InSAR image only.
- `interp_method` interpolation method for fault formation.
- `inversion_opt` options for the inversion step.
- `iter_max` is the maximum number of iterations of the linear decomposition algorithm.
- `iter` current iteration number.
- `L` principal slip distributions.
- `lambda` weighting of the smoothing parameter for fault surface construction.
- `Lap` matrix form of the discrete Laplacian.

- `last_epoch` is a scalar denoting the last allowed epoch in the timeseries.
- `lat_dense` latitude of the dense time-series
- `Lat_Orig`: Latitude of Origin (decimal seconds).
- `lat_sparse` latitude of the sparse time-series
- `Lat`: Latitude (decimal seconds).
- `lat` latitude in decimal seconds
- `length_unit` is a string denoting what length unit (e.g. 'mm', 'cm') will be used as fundamental to the analysis. All length data and errors will be converted into this unit.
- `length` length of a rectangular fault patch.
- `long_dense` longitude of dense time-series.
- `long_sparse` longitude of sparse time-series.
- `long` longitude.
- `los_file` path to the line-of-sight file for an InSAR image (or other directional data)
- `mean_function` function to find the mean estimates.
- `mean_offset_fine` fine estimate of the mean offset necessary to center the datasets.
- `mean_offsets` gross estimate of the mean offset necessary to center the datasets using the weighted mean.
- `mean_options` options to be used during the calculation of the mean estimates.
- `means` the mean values of matrix rows from `w_mean`
- `n_comp_mean` number of components to be used for estimating the means.
- `n_comp` is a positive integer specifying the number of components for the decomposition of the data matrix into linear components.
- `n_entries_finish`: the last indexes of each of the subcomponents of the matrix to be filled.
- `n_entries_start`: the first indexes of each of the subcomponents of the matrix to be filled.
- `n_entries` number of time-series in each dataset as a vector.
- `n_epochs` total number of epochs in all datasets.
- `N_nearest` number of nearest patches to use in Laplacian computations.
- `n_patches` total number of patches in the fault model.
- `n_tseries_vec`: a vector with the number of time series in each dataset as its elements.
- `n_tseries`: the total number of time series between all datasets.
- `N` number of nearest patches to use in Laplacian computations.
- `ncomp` number of components for linear decomposition.
- `ndim` total dimension of some matrix.
- `normal_vect`: normal to the fault base vector for each triangle.

• `nx` number of triangles along the direction given by angle.

• `ny` number of triangles along the direction perpendicular to the direction given by angle.

• `observation_unit:` output observation units (m,cm,mm).

• `options_make_fault_model` options for fault model construction.

• `options_plot_coast` options for plotting the coast on final figures.

• `options_plot_gps_vectors:` options for plot_gps_vectors.

• `options_plot_gps:` options for plot_gps_stations.

• `options_plot_L` options for plotting the principal slip distributions

• `options_plot_time_series` options for plotting time series

• `options_plot_V` options for plotting the principal time functions

• `options_sparse:` options for plot_SAR.

• `options` general options for some routine.

• `origin:` the center of the plot or origin of local coordinate frame.

• `output_disp_unit` string containing the target displacement unit

• `outputfile_okada` complete name (including path) of the output file corresponding to the fault model using Okada's format. This file is used for computation of Green functions. If `outputfile_pcaim` and `outputfile_okada` are leave empty, no output is written.

• `outputfile_pcaim` complete name (including path) of the output file corresponding to the fault model using the PCAIM code triangular format.

• `p` power for weighted mean calculation.

• `position{i}:` cell containing the longitude and latitude vectors of dataset i (e.g., the long and lat of GPS stations, longitude=position{i}(:,1);latitude=position{i}(:,2)).

• `r` direction of search during conjugate gradient algorithm.

• `rake` vector containing rake angle for each patch.

• `rectangular_fault_flag` flag to indicate whether or not the fault model is composed of rectangular elements.

• `S` weights for each component, equivalent to singular values for SVD.

• `s` slip at depth as a solution to the linear inversion problem $s = A\backslash d$.

• `SAR_data` data variable for InSAR data loading procedure and plotting.

• `SAR_model` model variable for InSAR plotting.

• `scenario_name` is a string denoting the directory in which the models are to be saved and data is to be found.

• `separator:` a character that separates the year-month-day from the hour-minute-second

• `sig_time:` number of significant digits after the decimal point when rounding epochs.

• `slip_pot` total slip potency, a scaler.

• `slip_vector` the vectors for plotting slip in geographical coordinates or local EN coordinates.

• `slip` magnitude of slip on each patch.

• `smooth_param` value of the smoothing parameter (e.g., 1e2). The higher, the smoother the surface. When `smooth_param` $\to +\infty$, the smoothing surface reduces to a plane.

• `sparse_constraint` matrix to augment the design matrix during the inversion step for including sparse data in the inversion.

• `stn_name` is a cell-structure where each cell contains a cell structure of strings of the names of the stations in `X_dat` for data types that have station names. In particular, the i^{th} cell of the k^{th} cell in `stn_name` corresponds to the i^{th} row of the k^{th} cell of `X_err` and `X_dat`.

• `strike_vect:` along strike base vector for each triangle.

• `strike` strike angle for each individual patch or mean strike angle the fault as a whole. In degrees.

• `string_to_remove` string that should be removed from input variables.

• `string` string to be found in some cell.

• `t1` vertex 1 of a triangular patch

• `t2` vertex 2 of a triangular patch

• `t3` vertex 3 of a triangular patch

• `theta` angle for rotation.

• `time_unit:` the time unit to be used internally during calculations.

• `timeline` is a vector where the j^{th} entry is the j^{th} unique epoch in chronologically order from any of the data sources.

• `tol` is the convergence tolerance for the linear decomposition function.

• `U_str` slip in the strike-slip direction.

• `U_updip` slip in the dip-slip direction.

• `u:` Distance from Origin to Latitude.

• `U` is a $m \times N$ matrix representing the spatial function of the linear decomposition $X \approx USV^t$. The j^{th} column is the spatial function of the j^{th} component.

• `unique_epochs` total number of unique epochs.

• `updip_vect:` updip base vector for each triangle.

• `V` is $n \times N$ matrix representing the temporal function of the linear decomposition $X \approx USV^t$. The j^{th} column is the temporal function of the j^{th} component.

• `value` input of default value to be set for some parameter.

• `vect_tect` 3d tectonic vector given in the geographical reference frame. Used to find the rake of the triangular patches.

• `vertices:` vertices of the triangle given by `[t1,t2,t3]`.

- `w` `w(i) < 0` where `x(i) = 0` and `w(i) = 0` where `x(i) > 0`, from fnnls
- `weight` weight matrix for use in `w_mean`
- `width` width of a rectangular fault element.
- `X_dat{i}(k,l)`: observation for set `i`, time series `k`, at `epoch X_time{i}(l)`.
- `X_dat` is a cell-structure where each cell contains a matrix of the imported data from a different data source (cGPS3, cGPS2, InSAR, etc.). Each row is one "station" (e.g. for cGPS3) or "location" (e.g. each pixel for InSAR data), and each column is the epoch for each station in that cell.
- `X_data_matrix` matrix version of`X_dat`.
- `X_err_sparse` same as `X_err`, but only contains sparse data
- `X_err{i}`: same as `X_dat`, but contains the 1-sigma standard errors for the corresponding elements.
- `X_error_matrix` matrix version of`X_err`.
- `X_G_index_sparse` integers indexing G for entries in `X_SOMETHING_sparse`
- `X_G_index` integers indexing G for entries in `X_SOMETHING`
- `X_matrix` general matrix version of `X_SOMETHING`.
- `X_model_matrix` matrix version of `X_model`.
- `X_model_sparse` model of the surface displacement field at all epochs if and only if was original data there, only for sparse data.
- `X_model` model of the surface displacement field at all epochs if and only if was original data there.
- `X_pred_sparse` predictions of the surface displacement field at all epochs regardless of whether there was original data there or not, only for sparse data.
- `X_pred` predictions of the surface displacement field at all epochs regardless of whether there was original data there or not.
- `X_rescale` cell array of doubles or matrixes for rescaling the errors on any datum in any datasets.
- `X_time_index_sparse` `X_time_index` for sparse datasets only.
- `X_time_index` is a cell structure where the k^{th} cell is an index to `X_time` from `timeline`. In other words, the k^{th} cell is vector of the same size as the k^{th} cell of `X_time` such that `timeline(X_time_index{k}{j})=X_time{k}{j}`.
- `X_time{i}`: cell containing the time vector of set #i.
- `X_time` is a cell-structure where each cell contains a vector of the imported epochs from a different data source (cGPS3, cGPS2, InSAR, etc.). The j^{th} entry of the vector in the k^{th} cell corresponds to the j^{th} column of the k^{th} cell of `X_err` and `X_dat`.
- `X_weight` is a cell-structure where each cell contains a matrix of the imposed multiplicative modifications to the weight of each data point from a different data source (cGPS3, cGPS2, InSAR, etc.). These imposed modifications allow the user to manually

reweight portions of the data which his/her geophysical intuition suggests are being either over or under fit. Note this manual reweighting will nearly always worsen the resulting χ^2 of the decomposition. You can think of this as a way to add a "fudge factor" to the decomposition and inversion.

- `x`: first variable representing x coordinate (typically East), or current guess in conjugate gradient algorithm.
- `X` a general `X_SOMETHING` matrix.
- `x0` second variable representing x coordinate (typically East).
- `x1` third variable representing x coordinate (typically East).
- `xc` central x (typically East) coordinate, either for rotation or the center of a patch.
- `xcell` target cell for finding a string.
- `Xdata` single matrix from `X_dat`.
- `Xerror` single matrix from `X_err`.
- `xfind` string to find inside a cell array.
- `xi` fourth variable representing x coordinate (typically East).
- `Xmodel` single matrix from `X_model`.
- `Xtime` single matrix from `X_time`.
- `XtX` "X transpose times X" for fnnls routine.
- `Xty` "X transpose times data vector" for fnnls routine.
- `y` first variable representing y coordinate (typically North).
- `y0` second variable representing y coordinate (typically North).
- `y1` third variable representing y coordinate (typically North).
- `yc` central y (typically North) coordinate, either for rotation or the center of a patch.
- `yi` fourth variable representing y coordinate (typically North).
- `z` first variable representing depth.
- `zc` central depth coordinate, either for rotation or the center of a patch.
- `zi` second variable representing depth.

Bibliography

[CAS+08] M. Chlieh, J. P. Avouac, K. Sieh, D. H. Natawidjaja, and John Galetzka. Heterogeneous coupling of the sumatran megathrust constrained by geodetic and paleogeodetic measurements. *JOURNAL OF GEOPHYSICAL RESEARCH*, 113:B05305, 2008.

[CBR+09] D.V. Chandrasekhar, R. Bürgmann, C.D. Reddy, P.S. Sunil, and D.A Schmidt. Weak mantle in NW India probed by geodetic measurements following the 2001 Bhuj earthquake. *Earth Planet. Sci. Lett.*, 2009.

[Coh99] S. C. Cohen. Numerical models of crustal deformation in seismic zones. *Adv. Geophys.*, 41:134–231, 1999.

[HSA+06] Y. J. Hsu, M. Simons, J. P. Avouac, J. Galetzka, K. Sieh, M. Chlieh, D. Natawidjaja, L. Prawirodirdjo, and Y. Bock. Frictional afterslip following the 2005 Nias-Simeulue earthquake, Sumatra. *Science*, 312(5782):1921–1926, 2006.

[Hui91] Geertjan Huiskamp. Difference formulas for the surface Laplacian on a triangulated surface. *Journal of Computational Physics*, 95(2):477 – 496, 1991.

[Ise04] Arieh Iserles. *Numerical Analysis of Differential Equations*. Cambridge University Press, 2004.

[KA10] Andrew P. Kositsky and Jean-Philippe Avouac. Inverting geodetic time-series with a principal component analysis-based inversion method (PCAIM). *Journal of Geophysical Research: Solid Earth*, 2010.

[Kin05] Martin King. Matlab m-files for multidimensional nonlinear conjugate gradient method, 2005. `http://users.ictp.it/~mpking/cg.html` (now defunct).

[KY06] M. Kawamura and K. Yamaoka. Spatiotemporal characteristics of the displacement field revealed with principal component analysis and the mode-rotation technique. *Tectonophysics*, 419:55–73, 2006.

[KY09] M. Kawamura and K. Yamaoka. Temporal relationship between the 2000 Miyake-Kozu seismovolcanic activity and the 2000 Tokai slow-slip event. *Tectonophysics*, pages 45–59, 2009.

[LR02] Roderick Little and Donald Rubin. *Statistical Analysis with Missing Data*. Wiley-Interscience, 2nd edition, 2002.

[Mea07] Brendan J. Meade. Algorithms for the calculation of exact displacements, strains, and stresses for triangular dislocation elements in a uniform elastic half space. *Comput. Geosci.*, 33(8):1064–1075, 2007. http://summit.fas.harvard.edu/~meade/meade/Publications_files/meadetriangulardislocations2007.pdf.

[Mog58] Kiyoo Mogi. Relations between the eruptions of various volcanoes and the deformations of the ground surfaces around them. *Bull. Earthq. Res. Inst.*, 36, 1958.

[Oka85] Y. Okada. Surface deformation to shear and tensile faults in a half space. *Bull. Seism. Soc. Am.*, 75:1135–1154, 1985.

[Oka92] Yoshimitsu Okada. Internal deformation due to shear and tensile faults in a half-space. *Bulletin Of The Seismological Society Of America*, 82(2):1018–1040, 1992.

[SJ03] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Twentieth International Conference on Machine Learning*, 2003. http://ttic.uchicago.edu/~nati/Publications/SrebroJaakkolaICML03.pdf.

[Sre] Nathan Srebro. Re: Weighted low-rank approximations request. Personal Correspondence to Andrew Kositsky. Sept. 10, 2009.

[XY89] X. Xie and Z. Yao. A generalized reflection-transmission coefficient matrix method to calculate static displacement field of a dislocation source in a stratified half-space. *Chin. J. Geophys.*, pages 191–205, 1989.

Appendix A

Downloading Coast Files

You can download the appropriate coastline files from `http://rimmer.ngdc.noaa.gov/mgg/coast/getcoast.html`. Instructions for using these within your plots with the code is below.

1. Download the appropriate coastline information.

   a) Input the upper latitude, westernmost longitude, easternmost longitude and lower latitude for your desired coast region.

   b) Choose any coastline database. We suggest the default ("World Vector Shoreline")

   c) Choose anything for "Compression method for extracted ASCII data"

   d) Choose "Matlab" for "Coast Format options"

   e) Choose "GMT Plot" for "Coast Preview options"

   f) Click on the "SUBMIT Extract the Coastline File" button.

2. Expand the resulting file if necessary, and save with a reasonable name so the user can remember what it is later (e.g. `Nias_coast.dat`) and put it inside the appropriate scenario folder.

3. Within the `plotting_commands_file` for the target scenario, replace the `coast_file_name` with the path of the new coast file.

Appendix B

Derivatives of χ^2

In order to implement the conjugate gradient method to find a local minimum of the χ^2 function, we need to know the derivative. Since taking derivatives numerically is usually computationally intensive compared to applying an analytical formula for relatively simple functions, we here analytically find the derivatives of χ^2 as a function of the spatial basis functions U, the temporal basis functions V, and when applicable, the timeseries means M.

We will pick each of the elements of the vectors independently, and we consider χ^2 to be a function of U and V. Since there are no boundary points in the domain of χ^2, χ^2 has a lower bound ($\chi^2(U,V) \not< 0$,), and for any $U, V \to \infty$ where the other is non-zero, $\chi^2 \to \infty$ it follows from calculus that χ^2 must have a non-zero number of global minima. Thus at a global minimum, $\frac{\partial \chi^2}{\partial u(l,k)} = \frac{\partial \chi^2}{\partial v(i,k)} = 0$ for all i, k. This does not guarantee there are no local minima, but in practice we have nearly always reached the same minimum given random starting conditions, enough iterations and a small enough tolerance.

$\chi^2(U,V)$

$$\chi^2 = \sum_{i,j} \left(\frac{\sum_{k=1}^{r} [U_{ik} V_{jk}] - X_{ij}}{\sigma_{ij}} \right)^2 \tag{B.1}$$

Derivative of χ^2 with respect to U

$$
\begin{aligned}
\frac{\partial \chi^2}{\partial U_{lm}} &= \frac{\partial \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik} V_{jk} - X_{ij}}{\sigma_{ij}} \right)^2}{\partial U_{lm}} && \text{(B.2)} \\
&= \sum_{i,j} \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik} V_{jk} - X_{ij}}{\sigma_{ij}} \right)^2}{\partial U_{lm}} && \text{(B.3)} \\
&= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik} V_{jk} - X_{ij}}{\sigma_{ij}} \right) \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik} V_{jk} - X_{ij}}{\sigma_{ij}} \right)}{\partial U_{lm}} && \text{(B.4)} \\
&= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik} V_{jk} - X_{ij}}{\sigma_{ij}} \right) \frac{\delta_{km} \partial \left(\sum_{k=1}^{r} U_{ik} V_{jk} \right)}{\sigma_{ij} \partial U_{lm}} && \text{(B.5)} \\
&= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{lk} V_{jk} - X_{ij}}{\sigma_{ij}} \right) \frac{\delta_{li} \partial \left(U_{im} V_{jm} \right)}{\sigma_{ij} \partial U_{lm}} && \text{(B.6)} \\
&= 2 \sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk} V_{jk} - X_{lj}}{\sigma_{lj}^2} \right) V_{jm} && \text{(B.7)}
\end{aligned}
$$

Derivative of χ^2 with respect to V

$$\frac{\partial \chi^2}{\partial V_{lm}} = \frac{\partial \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}}\right)^2}{\partial V_{lm}} \tag{B.8}$$

$$= \sum_{i,j} \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}}\right)^2}{\partial V_{lm}} \tag{B.9}$$

$$= 2\sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}}\right) \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}}\right)}{\partial V_{lm}} \tag{B.10}$$

$$= 2\sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}}\right) \frac{\delta_{km}\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk}\right)}{\sigma_{ij}\partial V_{lm}} \tag{B.11}$$

$$= 2\sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{ij}}{\sigma_{ij}}\right) \frac{\delta_{lj}\partial \left(U_{im}V_{jm}\right)}{\sigma_{ij}\partial V_{lm}} \tag{B.12}$$

$$= 2\sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{lj}}{\sigma_{lj}^2}\right) U_{im} \tag{B.13}$$

$\chi^2(U, V, M)$, V has zero mean

$$\chi_m^2 = \sum_{i,j} \left(\frac{\sum_{k=1}^{r}[U_{ik}V_{jk}] - X_{ij} + M_i}{\sigma_{ij}}\right)^2, \quad \text{for all } k \ \sum_{j=1}^{n-1}(V_{jk}) = -V_{nk} \tag{B.14}$$

Derivative of χ^2_m with respect to M

$$\frac{\partial \chi^2_m}{\partial M_l} = \frac{\partial \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)^2}{\partial M_l} \tag{B.15}$$

$$= \sum_{i,j} \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)^2}{\partial M_l} \tag{B.16}$$

$$= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right) \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)}{\partial M_l} \tag{B.17}$$

$$= 2 \sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{lj} + M_l}{\sigma_{lj}^2} \right) \frac{\partial M_l}{\partial M_l} \tag{B.18}$$

$$= 2 \sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{lj} + M_l}{\sigma_{lj}^2} \right) \tag{B.19}$$

Derivative of χ^2_m with respect to U

$$\frac{\partial \chi^2_m}{\partial U_{lm}} = \frac{\partial \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)^2}{\partial U_{lm}} \tag{B.20}$$

$$= \sum_{i,j} \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)^2}{\partial U_{lm}} \tag{B.21}$$

$$= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right) \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}} \right)}{\partial U_{lm}} \tag{B.22}$$

$$= 2 \sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}^2} \right) \frac{\delta_{li}\delta_{km}\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk} \right)}{\partial U_{lm}} \tag{B.23}$$

$$= 2 \sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{lj} + M_l}{\sigma_{ij}^2} \right) \frac{\partial \left(U_{lm}V_{jm} \right)}{\partial U_{lm}} \tag{B.24}$$

$$= 2 \sum_{j} \left(\frac{\sum_{k=1}^{r} U_{lk}V_{jk} - X_{lj} + M_l}{\sigma_{lj}^2} \right) V_{jm} \tag{B.25}$$

Derivative of χ^2_m with respect to V

$$
\begin{aligned}
\frac{\partial \chi^2_m}{\partial V_{lm}} &= \sum_{i,j} \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right)^2}{\partial V_{lm}} && \text{(B.26)} \\
&= 2\sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}} \right) \frac{\partial \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij}}{\sigma_{ij}} \right)}{\partial V_{lm}} && \text{(B.27)} \\
&= 2\sum_{i,j} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk} \right)}{\partial V_{lm}} && \text{(B.28)} \\
&= 2\sum_{i} \sum_{j=1}^{n} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk} \right)}{\partial V_{lm}} && \text{(B.29)} \\
&= 2\sum_{i} \left[\sum_{j=1}^{n-1} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk} \right)}{\partial V_{lm}} + \right. && \text{(B.30)} \\
&\qquad \left. \left(\frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik}V_{nk} \right)}{\partial V_{lm}} \right] \\
&= 2\sum_{i} \left[\sum_{j=1}^{n-1} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{jk} - X_{ij} + M_i}{\sigma_{ij}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik}V_{jk} \right)}{\partial V_{lm}} + \right. && \text{(B.31)} \\
&\qquad \left. \left(\frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} U_{ik} \left[- \sum_{j=1}^{n-1} V_{jk} \right] \right)}{\partial V_{lm}} \right] \\
&= 2\sum_{i} \left[\sum_{j=1}^{n-1} \left(\frac{\sum_{k=1}^{r} U_{ik}V_{lk} - X_{il} + M_i}{\sigma_{il}^2} \right) \frac{\partial \left(\sum_{k=1}^{r} \delta_{l,j}\delta_{m,k} U_{ik}V_{jk} \right)}{\partial V_{lm}} + \right. && \text{(B.32)} \\
&\qquad \left. \left(\frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) \delta_{l,j} \frac{\partial \left(\sum_{k=1}^{r} \delta_{l,j}\delta_{m,k} U_{ik} \left[- \sum_{j=1}^{n-1} V_{jk} \right] \right)}{\partial V_{lm}} \right] \\
&= 2\sum_{i} \left[\left(\frac{\sum_{k=1}^{r} U_{ik}V_{lk} - X_{il} + M_i}{\sigma_{il}^2} \right) U_{im} + \right. && \text{(B.33)} \\
&\qquad \left. \left(\frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) \frac{\partial \left(-U_{im}V_{lm} \right)}{\partial V_{lm}} \right] \\
&= 2\sum_{i} \left[\left(\frac{\sum_{k=1}^{r} U_{ik}V_{lk} - X_{il} + M_i}{\sigma_{il}^2} \right) U_{im} - \left(\frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) U_{im} \right] \\
&= 2\sum_{i} \left[\left(\frac{\sum_{k=1}^{r} U_{ik}V_{lk} - X_{il} + M_i}{\sigma_{il}^2} - \frac{\sum_{k=1}^{r} U_{ik}V_{nk} - X_{in} + M_i}{\sigma_{in}^2} \right) U_{im} \right] && \text{(B.34)}
\end{aligned}
$$

Appendix C

Analytical Minimum

This appendix analytically solves the problem of finding the minimum along a fixed direction of search from a fixed location for two specific objective functions, χ^2 and χ^2_m from Equations (B.1, B.14). This will tell us how large of a step d we should make in the direction $\vec{r}$ from our current position of $U^0 V^{t0}(+M^0)$, where $+M^0$ is only included in the χ^2_m case.. We will do this by first finding the minimum of χ^2_m and then showing that we have also calculated the minimum for χ^2 by setting some of the terms equal to zero.

Our problem is to find the minimum of the equation:

$$\chi^2_m = \sum_{i,j} \left(\frac{\sum_{k=1}^{r} [U_{ik} V_{jk}] - X_{ij} + M_i}{\sigma_{ij}} \right)^2 \tag{C.1}$$

$$= \sum_{i,j} \left(\frac{\sum_{k=1}^{r} [(U^0_{ik} + dU^r_{ik})(V^0_{jk} + dV^r_{jk})] - X_{ij} + (M^0_i + dM^r_i)}{\sigma_{ij}} \right)^2 , \tag{C.2}$$

where $U^r_{ik}, V^r_{jk}, M^r_i$ compose the search direction $\vec{r}$ and d is the step-size and sign. As the search direction and current location are fixed, the only free variable in Equation C.2 is d. Thus the correct step size can be found by finding all the minimum of the fourth-order polynomial in d represented by Equation C.2. As we know $\lim_{d \to \pm\infty} \chi^2_m = \infty$ (Paragraph 1 of Appendix B), by the extreme value theorem we know the minimum of $\chi^2_m(d)$ is at a point where $\frac{\partial \chi^2_m(d)}{\partial d} = 0$.

Define $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$ as:

$$\alpha_{ij} = \sum_k (U^0_{ik} V^r_{jk}) \tag{C.3}$$

$$\beta_{ij} = \sum_k (U^r_{ik} V^0_{jk} + V^r_{j,k} U^0_{ik} + M^r_i \tag{C.4}$$

$$\gamma_{ij} = \sum_k (U^0_{ik} V^0_{jk} + M_i - X_{ij}, \tag{C.5}$$

and substituting $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$ into Equation C.2, we have

$$\chi^2_m = \sum_i \sum_j \frac{(\alpha_{ij} d^2 + \beta_{ij} d + \gamma_{ij})^2}{\sigma^2_{ij}} \tag{C.6}$$

$$= \sum_i \sum_j \frac{\alpha^2_{ij} d^4 + 2\alpha_{ij}\beta_{ij} d^3 + (2\alpha_{ij}\gamma_{ij} + \beta^2_{ij}) d^2 + (2\beta_{ij}\gamma_{ij}) d + \gamma^2_{ij}}{\sigma^2_{ij}} \tag{C.7}$$

We now take the derivative $g(d) \equiv \frac{\partial \chi^2_m(d)}{\partial d}$ with χ^2_m in the form from Equation C.7, solve for the zeros of $g(d)$ using standard expressions, and pick the value of d from these zeros minimizing $\chi^2_m(d)$ at the real valued zeros of $g(d)$. Thus we have found the minimum of $\chi^2_m(d)$ along the search direction $\vec{r}$ from the starting position $U^0 V^{t0}$.

All that remains to be shown is that this approach works for $\chi^2(d)$ as well. It's clear through substitution that Equation C.7 is equal to equation B.1 if in the definitions of β and γ, the quantities M^r_i and M^0_i are set to zero. Thus the final expression we get for the derivative can be used for computing the minimum of both χ^2 and χ^2_m along $\vec{r}$ as long as we define β, γ differently for the two cases, and by the same logic as the χ^2_m case we have found the optimal step size d in one step.

□

Appendix D

Laplacian

We here outline the work from [Hui91] to derive an analytic expression for a good approximation of the Laplacian on an irregularly sampled plane.

Consider the problem of computing the Laplacian on an irregularly sampled plane locally as depicted in Figure D.1. Compared to the regular polygon case, there is a break in symmetry and there are irregular distances between the neighbors and central point. We correct the distances by using a linear interpolation scheme to estimate the values of points equidistant (distance r', to be specified later) from the central point p_0. In particular, we know $\vec{r'} = \lambda\vec{r}_i + \mu\vec{r}_{i+1}$, so we estimate $f(r') \approx f_0 + \lambda(f_i - f_0) + \mu(f_{i+1} - f_0)$, with $\lambda, \mu \geq 0$. (The additional inequality $\lambda + \mu \leq 1$ is required if $\vec{r'}$ is in the triangle, but we allow $\vec{r'}$ to cross the edge $(\vec{r}_{i+1} - \vec{r}_i)$. We can then express λ and μ in terms of r_i, r_{i+1}, r' and the angles α and ϕ_i. Specifically,

$$\lambda = \frac{r'}{r_i}\frac{\sin(\phi_i - \alpha)}{\sin(\phi_i)}, \qquad \mu = \frac{r'}{r_{i+1}}\frac{\sin(\alpha)}{\sin(\phi_i)} \tag{D.1}$$

and thus,

$$f(r', \alpha) - f_0 = \frac{r'}{r_i}\frac{\sin(\phi_i - \alpha)}{\sin(\phi_i)}(f_i - f_0) + \frac{r'}{r_{i+1}}\frac{\sin(\alpha)}{\sin(\phi_i)}(f_{i+1} - f_0). \tag{D.2}$$

Define $x = r'\cos(\theta), y = r'\sin(\theta)$, we can take the integral around p_0 at radius r' of the truncated Taylor expansion,

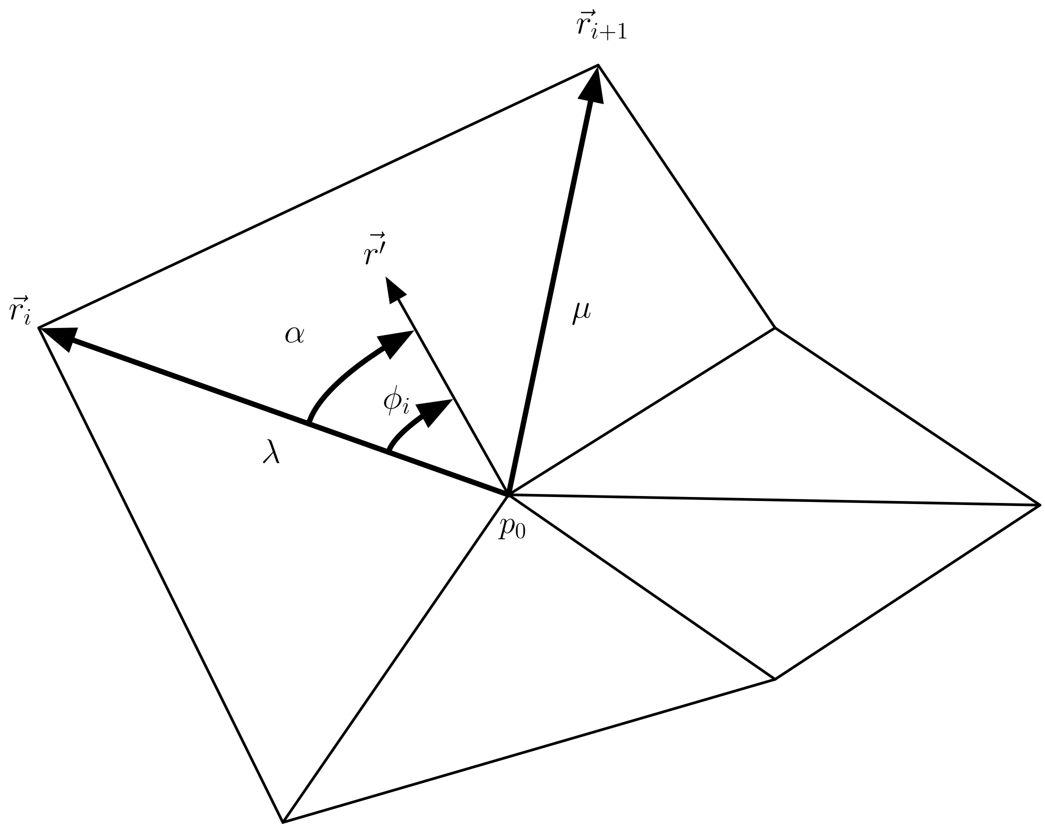

Figure D.1: General situation for an irregular triangular planar grid. Modeled after [Hui91, Figure 2]

$$\begin{aligned}
\int_0^{2\pi}(f(r',\theta)-f_0)d\theta \quad &\approx \quad r\int_0^{2\pi}\cos(\theta)d\theta\,\frac{\partial f}{\partial x}\bigg|_{p_0}+r\int_0^{2\pi}\sin(\theta)d\theta\,\frac{\partial f}{\partial y}\bigg|_{p_0} && \text{(D.3)}\\
&\quad +r^2\int_0^{2\pi}\cos(\theta)\sin(\theta)d\theta\,\frac{\partial^2 f}{\partial x\partial y}\bigg|_{p_0}\\
&\quad +\frac{1}{2}r^2\int_0^{2\pi}\cos^2(\theta)d\theta\,\frac{\partial^2 f}{\partial x^2}\bigg|_{p_0}+\frac{1}{2}r^2\int_0^{2\pi}\sin^2(\theta)d\theta\,\frac{\partial^2 f}{\partial y^2}\bigg|_{p_0}\\
&= \quad 0+0+0+\frac{\pi r^2}{2}\,\frac{\partial^2 f}{\partial x^2}\bigg|_{p_0}+\frac{\pi r^2}{2}\,\frac{\partial^2 f}{\partial y^2}\bigg|_{p_0} && \text{(D.4)}\\
&= \quad \frac{\pi r^2}{2}\Delta f. && \text{(D.5)}
\end{aligned}$$

Returning to Eqn. D.2, we also know that,

$$\begin{aligned}
&= \int_0^{2\pi}(f(r',\theta)-f_0)d\theta && \text{(D.6)}\\
&= \int_0^{2\pi}\left(\frac{r'}{r_i}\frac{\sin(\phi_i-\theta)}{\sin(\phi_i)}(f_i-f_0)+\frac{r'}{r_{i+1}}\frac{\sin(\theta)}{\sin(\phi_i)}(f_{i+1}-f_0)\right)d\theta && \text{(D.7)}\\
&= \sum_{i=1}^{N}\left[\frac{r'}{r_i}\cdot\frac{1-\cos(\phi_i)}{\sin(\phi_i)}(f_i-f_0)+\frac{r'}{r_{i+1}}\cdot\frac{1-\cos(\phi_i)}{\sin(\phi_i)}(f_{i+1}-f_0)\right] && \text{(D.8)}\\
&= \sum_{i=1}^{N}\frac{r'}{r_i}\left(\frac{1-\cos(\phi_i^-)}{\sin(\phi_i^-)}+\frac{1-\cos(\phi_i^+)}{\sin(\phi_i^+)}\right)(f_i-f_0), && \text{(D.9)}
\end{aligned}$$

where the last equality holds by reordering terms and defining ϕ_i^+ as the angle from $\vec{r}_i$ to $\vec{r}_{i+1}$, and ϕ_i^- as the angle from $\vec{r}_{i-1}$ to $\vec{r}_i$, where $r_0 \equiv r_N$ and $r_1 \equiv r_{N+1}$.

Equating these two expressions for $\int_0^{2\pi}(f(r',\theta)-f_0)d\theta$, we arrive at the approximation,

$$\Delta f_0 \approx \frac{4}{r'}\frac{1}{2\pi}\sum_{i=1}^{N}\left(\frac{1-\cos(\phi_i^-)}{\sin(\phi_i^-)}+\frac{1-\cos(\phi_i^+)}{\sin(\phi_i^+)}\right)\frac{f_i-f_0}{r_i}. \quad \text{(D.10)}$$

A good choice for r' seems to be such that in the case of equal angles, $r' = \bar{r}\frac{N}{2\pi}$[1],

[1]See original text for details.

where $\bar{r} \equiv$ mean distance of neighboring points to ${p_0}^2$. This implies that, defining

$$\Phi_{tot} = \sum_{i=1}^{N} \left(\frac{1-\cos(\phi_i^-)}{\sin(\phi_i^-)} + \frac{1-\cos(\phi_i^+)}{\sin(\phi_i^+)} \right),$$

we have the approximation

$$\Delta f_0 \approx \sum_{i=1}^{N} w_i^{(2)} (f_i - f_0), \tag{D.11}$$

where

$$w_i^{(2)} = \frac{4}{\bar{r}} \cdot \frac{1}{\Phi_{tot}} \cdot \frac{1}{r_i} \left(\frac{1-\cos(\phi_i^-)}{\sin(\phi_i^-)} + \frac{1-\cos(\phi_i^+)}{\sin(\phi_i^+)} \right). \tag{D.12}$$

[2]Though this choice ($\neq 0$) doesn't change the result because of linear interpolation assumption for f – for the non-linear interpolation case we need to pick the optimal distance and/or steps for interpolation carefully (e.g. Iserles)