NIF codes and tutorial readme
Noel Bartlow, University of Missouri, last updated 11/17/2015

This is a preliminary distribution of the Network Inversion Filter codes written for MATLAB. The code is somewhat rough but commented. These versions have been tested in MATLAB version 2014a. These codes have been primarily developed by Paul Segall, Shin'ichi Miyazaki, Yosuke Aoki, Zhen Liu, Andrew Bradley, and myself. I currently maintain and continue to develop them. For relevant citations, please see the end of this document.

The NIF is a powerful tool for inverting fault slip as a function of time. It is primarily used to study slow slip or post seismic transient slip. These codes are written assuming you are using GPS positions as data, but in principle the filter can be altered to invert tilt, strain, or other types of data as well.

For info about any individual function, in MATLAB you can type help *functionname* and that will print out the header comments which list inputs and outputs of the function.

Steps to run the tutorial:

1) open matlab, make the current directory NIF (it's a good idea to clear all also)
2) set your path using:
addpath('./objects')
addpath('./YoMesh')
addpath('./geodesy')
2) open gparms.m in this folder. Change the scratch directory path as needed. Set the save file name to something unique. Run gparms.
3) Run NIFmain(gp) on the matlab command line.
4) plot the output using:

```
load example/coastline.dat
slip_hist(10, 70, 8, 5, [-125 -122], [44.5 49.4], gp.svfn, coastline);
plotlist=['P374'; 'P376'; 'NWBG'; 'P409'; 'P417'; 'P430'; 'ELSR'; 'P436'];
GPS_fit_plot(plotlist, gp.svfn)
```

you can check out all the variables output by the filter by typing:
load(gp.svfn)

of interest are slip_b, slip_f, rate_b, and rate_f which are time-dependent slip and rate, backwards smoothed (b) or forward filtered only (f). The rows represent sub faults and the columns represent epochs.

ETS2011_data represents a slightly longer time series than ETS2011_data2. ETS2011_setup4 is a much finer sub fault mesh that takes much much longer to run than ETS2011setup_coarse.

The general steps to run the NIF as as follows:

step 0) Before running the NIF, you need to generate 3 input files: one containing data, one containing secular station velocities, and one containing Green's Functions. These can be created however you like, but I have included code to generate the Green's Functions for triangular meshes here. See the "Optional codes to create input files" section of this read me for more. Also see the "File descriptions" section for details on formatting the input files, and the examples folder for example input files.

1) make sure the necessary folders are included in your MATLAB path variable. Type on the MATLAB command line:

```
addpath('./objects')
addpath('./YoMesh')
addpath('./geodesy')
```

Or replace the paths in the quotes as needed. If you do not have the proper folders in your path, you will receive the errors like "Undefined function 'state' for input arguments of type 'double'."

2) open gparms.m. This script sets various filter variables and specifies your input files. It also specifies the beginning part of the name of the output file. This must be unique for every run of the filter, or you will encounter an error. This is to prevent accidentally writing over previous output files. The scratch directory will contain files used in the running of the filter, and will be overwritten on subsequent runs unless you change the scratch directory.

The data structure, secular velocity file, and setup file have specific expected structures. See the example folder for examples (ETS2011_short.mat is a data file, ETS2011_setup4.mat is a setup file, ETS2011_secular.mat is a secular velocity file). A description of the variables included in each file is given at the end of this document. Codes used to create this files are not being released at this time, however you may contact me at nbartlow@ucsd.edu to ask for my current versions.

Once you've specified your inputs, you can also use gparms.m to change your spatial and temporal smoothing parameters, and turn on various flags. proj_use_fwd and proj_use_bwd should be set to 1 to implement non-negativity; non-negativity is applied to slip-rate but not to slip. You can turn on the gp.mle flag to perform a grid search over multiple smoothing parameters to find the maximum likelihood solution. This performs a run of the forward filter for every grid point and take a long time.

Once you like your settings, run gparms.m. This will output a structure called gp.

3) Run the main code with gp as an input by typing: NIFmain(gp). This main code will call the forward filter code (forward_nnls.m) and the backward filter code (backward.m). It will print various things to the screen. The screen output of the forward filter looks like:

[average slip [cm], average slip-rate [cm/yr]] = 0.41942 37.1002
[chi-square value at each epoch] = 1.0187
loop wall clock time = 19.840841
[The epoch and the time] = 12 2011.452055

The screen output of the backward filter looks like:

Running Smoother
[k kk] = 11 11
qp start.....et = 0.000024 llx - x_hatll_2 = 2.393158e+00
[average slip [cm], average slip-rate [cm/yr]] = 0.28299 26.4397

When finished, a plot showing your final slip distribution and station locations will appear. The filter can take a while to run, and run time scales roughly with the number of fault patches squared.

4) load the filter results by typing load(gp.svfn)

3) Run plotting codes as desired. The main plotting codes I have included here are slip_hist.m and GPS_fit_plot.m.

Slip_hist takes as it's input the beginning and end epochs for a slip-rate snapshot plot, the number of epochs to skip between panels, the duration of each panel, longitude limits, latitude limits, the full name of the savefile (stored as gp.svfn), and a coastline file.

GPS_fit_plot takes a list of station names to be plotted and the save file.

friendly tip: All the parameters you set in gparms.m write out to a text file that has the same name as the save file, but ends with .cmt. This way you can always check things like smoothing parameters in an easy text file!

File descriptions:
The code that I use to make the secular velocity is an entirely separate Kalman filter which is not ready for release. However, secular velocities for many stations are published or publicly available, and there are also other existing methods for obtaining secular station velocities. For slow slip studies, these secular velocities should specify the velocity of the station in between slow slip events (inter-SSE velocity).

The structure for the data file needs to be built out of individual GPS time series, and the code to create it will vary based on time series format. Email Noel Bartlow to obtain a short (and messy) script that formats UNAVCO GPS position time series, or make your own.

Data file:

This file contains a single structure that must be called stationstruct. This structure's

length is equal to the number of data epochs which must be in order (first to last) but do not have to be evenly spaced. Each epoch contains the fields data, names, DOY, and year. For example, the data for the first epoch can be viewed by typing stationstruct(1).data.

data: an N by 6 array containing the data (assumed to be GPS displacement in units of mm) and uncertainties. N is the number of stations with data for the given epoch. First 3 columns contain the positions (E, N, U); columns 4-6 contain their uncertainties. Only stations with data during the given epoch will have data included in this array.

names: the 4 character names of the stations included in this epoch. Must be N by 4 char type.

DOY: integer day of year

year: decimal year

Setup file:

origin: A lat, long point near the fault used for map projections

ID_G: a list of all GPS stations used in the inversion. The order of this list is important and must correspond of the order of the rows or Kern_GPS, and really everywhere else.

Ncomps: number of total station components, generally number of stations * 3 (E, N, U)

Nbasis1: number of fault sub patches

Kern_GPS: these are the Green's functions, or more accurely Green's function factors. Each set of 3 rows represents a station, with E, N, and U components (index i) and each column represents a fault patch (index j). Each entry corresponds to how much station component i moves due to unit slip on patch j. Units are all meters. Kern_GPS should be Ncomps x Nbasis1 in size.

Lap: This is a desecrate laplacian matrix that can multiply a vector representing slip on each fault patch and approximate the second spatial derivative of that slip distribution. Used for spatial smoothing. Should be Nbasis1 x Nbasis1 in size.

lat_G, lon_G: latitude and longitude coordinates of the GPS stations, same order as ID_G.

nd_ll: nodes of a triangular mesh. Each row is a node, col 1 are longitudes, col 2 are latitudes, col 3 are depths (negative numbers)

el: indices to the nodes in nd_ll that define each triangle. each row is a triangle with 3 columns that correspond to row indices of nd_ll.

nd_ll and el follow the convention expected by matlab's trisurf plotting routine, and are only used for plotting. If you calculate your Green's functions on a rectangular or any other mesh, you can not include these variables. In that case, remove the call to trisurf in NIFmain. You will need to write your own plotting codes.

Secular velocity file:

This file contains secular velocities for each station and seasonal sine and cosine terms. These are corrected for in the forward filter code. Secular velocities are given in m/year and seasonal amplitudes in meters.

Ev: East velocities at each station, same order as ID_G.
Ea1: Amplitude of east annual sine components at each station, same order as ID_G.
Ea2: Amplitude of east annual cosine components, I think you know the order by now.
Es1: Amplitude of east semiannual sine components
Es2: Amplitude of east semiannual cosine components

similar variables exist starting with N or U for the other components.

Optional codes to create input files:
To create the setup file:
1) First run create_fault_mesh.m. See the comments of this code for the proper citation. This requires as an input a contour file, which is a .txt file with a longitude, latitude, and depth for each fault point, on separate lines. Points should be listed in order from shallow to deep, and along coherent lines in lat-long space. Depths are negative. This file may contain NaNs (they are just removed). A sample contour file is included as example/cascadia_contours2012.txt (These contours are for the Cascadia subduction zone, from McCrory et al., JGR, 2012 supplementary material). For example, to create a mesh for Cascadia with 15 km triangles, type:

create_fault_mesh('example/Cascadia_contours2012.txt', 20, 43.5, 49.5, -130, -110, -10, -70, 'example/myfault.mat')

Note that although the depth limits are -10 and -70, the resulting mesh goes from 20 to 65 km depth. The actual depth range will usually bit a bit narrower than specified. Do not specify a shallow depth shallower than 10 km with the example file as it may break the mesh.

2) Now run GF_setup.m to create the elastic Green's functions. This file assumes a homogenous elastic half space and a triangular mesh input as output by create_fault_mesh.m, following the method of Okada. See the comments of this code for the proper citation.

First, you need to add some directories to the matlab path:
addpath('./tridisloc3d')
addpath('./rot_subs')

Then, run GF_setup passing in your fault mesh, information about your station locations and names, and a savefile name. Following on the previous example, first load information about some GPS stations in the Cascadia region:

load example/GPS_info.mat

In this example file, ID_G contains the station names, lat_G, lon_G, and h_G contain lat, long, and height info. These do not need to be hyper-precise, and in fact for the purposes of this example h_G is all 0s. Now you can create Green's Functions by running:

GF_setup('example/myfault.mat', 52, 0.25, lat_G, lon_G, h_G, ID_G, 'example/mysetupfile.mat')

Here 52 refers to the direction of slip on each triangle, which is constant over the entire mesh in this example. Alternatively, that input variable can be a vector of the same length as the el variable in the fault mesh file, specifying a slip direction for each triangle individually. Type help GF_setup for more info.

Now you have a setup file that can be specified in gparms as gp.setup_mat.

Relevant citations:

Original NIF publication:

Segall, P., and M. Matthews (1997), Time dependent inversion of geodetic data, *J. Geophys. Res.*, **102**, 22,391–22,409.

More recent publications using code closer to this version (spatial smoothing method updated in 2006, other minor updates and non-negativity added in 2011):

Miyazaki, S., P. Segall, J. J. McGuire, T. Kato, and Y. Hatanaka (2006), Spatial and temporal evolution of stress and slip rate during the 2000 Tokai slow earthquake, *J. Geophys. Res.*, **111**, B03409, doi:10.1029/2004JB003426

Bartlow, N. M., S. Miyazaki, A. M. Bradley, and P. Segall (2011), Space-time correlation of slip and tremor during the 2009 Cascadia slow slip event, Geophys. Res. Lett., 38, L18309, doi:10.1029/2011GL048714.

For the triangular dislocation codes (tridisloc3d, neighbors.m, tri_laplacian, etc, called by GF_setup.m):

Liu, Z., P. Segall, 2005, Detecting and imaging aseismic transient deformation in southern California, *Eos Trans. AGU, 86*(52), Fall Meet. Suppl., Abstract G51B-0815.

Thomas, A. L. (1993), Poly3D: A three-dimensional, polygonal element, displacement discontinuity boundary element computer program with applications to fractures, faults, and cavities in the Earth's crust, M.S. thesis, Dep. of Geol. and Environ. Sci., Stanford Univ., Stanford, Calif.

For building a triangular mesh (create_fault_mesh.m):

Fukushima, Y., V. Cayol, and P. Durand (2005), Finding realistic dike models from interferometric synthetic aperture radar data: The February 2000 eruption at Piton de la Fournaise, J. Geophys. Res., 110, B03206, doi:10.1029/2004JB003268.

For the non-negativity constraint on slip-rate:

Simon, D., and D. L. Simon (2006), Kalman filtering with inequality constraints for turbofan engine health estimation, *IEE Proc. Control Theory Appl.*, **153**,371–378.