Fakenet User Manual

Duncan Carr Agnew
Institute of Geophysics and Planetary Physics
Scripps Institution of Oceanography
University of California
La Jolla CA 92093-0225 USA
`dagnew@ucsd.edu`

Package Version 1.2.1
Manual Version 1.1.1, December 30, 2013

1 Introduction

Fakenet is a package (programs and data files) to create simulated data for earth-deformation networks, which can be used to test methods for detecting transient motions (Agnew, 2013). Each program in this package reads in a set of commands, which cause each program to read from input files, and write to one or more output files (Figure 1).

Fakenet computes timeseries that contain:

1. Randomly perturbed velocities.
2. Gaps, as specified by an input file.
3. Randomly-assigned noise levels for each time series, with noise types including white, flicker, and random-walk.
4. Common-mode noise that decorrelates with increasing distance between stations. This can include sinusoidal signals at specified periods, with randomly chosen amplitude and phase.
5. Signals from slip on faults, including propagating slip, with arbitrary time constants and a variety of time variations.
6. Annual signals with random amplitude and phase modulation, to simulate seasonal signals.

The current version of the program only provides simulated displacement series.

The final output is a set of files, each of which contains the time series for a particular location. As a convenience, the program combines these files into a single compressed tarfile; this, and the files it contains, all are assigned a unique name. The program also generates a *log file*, which documents the actual settings of the inputs, so that the input files and be recreated and the program rerun: this serves as a kind of "audit trail" of what was done.

2 Installation

The package is distributed as a gzipped tarfile called `fakenet-1.2-tar.gz`. Uncompressing and untarring this (which may be done by running the command `tar xvzf fakenet-1.2-tar.gz`) will create a directory `fakenet-1.2` containing the Fakenet source and four subdirectories: `src`, containing the source code; `doc`, containing the program manual; `examples`, containing script routines to create some example files; `data`, containing files of station and fault information, and `bin`, where the executable will be put.

The code is written in Fortran-77 and can be compiled by running `make fakenet`. Options for the Fortran compiler are set in the associated `makefile`. The executables will be placed in the local `bin` subdirectory.

2.1 Data Files Included with the Distribution

For ease of use (in Southern California, at any rate), the distribution includes, in the directory `data`, various files that can be drawn on in developing possible sources of deformation.

A set of fault files is derived from the SCEC Community Fault Model, or more precisely from version 3.0.5 of the rectilinear approximations to these faults with variable corner points, which have been adjusted to make the top and bottom edges horizontal.[1] Many of these faults have top edges that are (correctly) taken to be well above sea level; when computing planes for use in dislocation modeling, these faults are moved down vertically so that their top edge is at the surface of a halfspace. The file `planes.cfm` contains these fault planes, reformatted in the style of the fault files used here, but with names and segment numbers appended to each line. The file `index.cfm` gives a listing of the fault names in an approximately geographic ordering.

As well as the faults, the distribution includes a file (`basins.source`) that contains several arrays of shallow "volume" sources that are inside the outlines of the sedimentary basins of the San Fernando, San Gabriel, San Bernardino, and Coachella Valleys. Each one of the sources in the arrays is actually three orthogonal dislocations, each of which would undergo opening: this provides a good approximation to the usual "Mogi" model of a purely dilatational source. The array can be used to simulate the motions cause by changes in the water table over a particular basin.

The distribution also includes the following:[2]

[1]Source file at `http://structure.harvard.edu/cfm/download/cfm-r-3.0.5cvd.gz`

[2]All derived from the REASoN (`http://reason.scign.org/`) time series distributions and metadata,

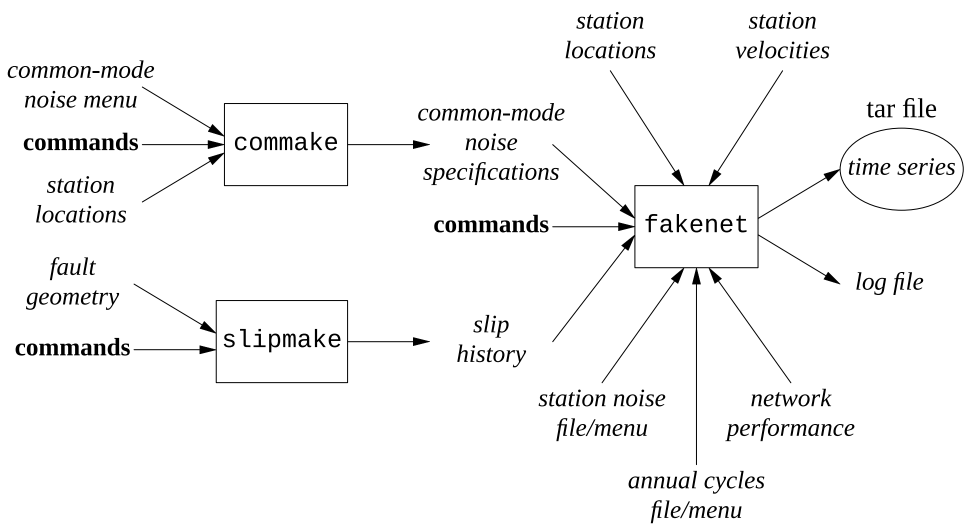

Figure 1: Flowchart for the Fakenet package; individual programs are in boxes. Input and output files are in *italics*.

- A file of coordinates for stations in California.
- A file of station performance, to mid-2012.
- A file of station velocities.

Finally, the distribution includes two files that provide "menus" that can be drawn from at random to provide different distributions of the noise at different stations. The file `noise.menu`, gives the estimated levels of white, flicker, and random-walk noise at 236 stations of the SCIGN network, as estimated by Langbein (2008), and provided by John Langbein (pers. comm., 2009). The file `annamp.menu` provides the amplitudes of annual variations estimated for the stations whose velocities and coordinates are provided.

2.2 Versions

The versions of the different programs that comprise Version 1.2.1 of the Fakenet package are 1.3 (for `commake`), 1.17 (for `slipmake`), 1.26 (for `fakenet`), and 1.5 (for `fakesubs`).

3 Operation

Figure 1 is a block diagram showing how a simulated time series for a geodetic network is created. The program that does this (`fakenet`) reads from several *input files*; these describe the station locations and velocities, the slip on faults, the parameters for common-mode noise, and so on. The program also reads instructions given in a **command file**. Some of the input files for `fakenet` are complicated enough other programs are used to generate them: `slipmake` generates the file giving fault slip, `commake` the file for common-mode noise. Keeping these programs separate from `fakenet`, though not strictly necessary, made development and debugging much easier. Like `fakenet`, these programs read both command and input files. For `commake`, the inputs are files of station locations and noise parameters; for {texttttslipmake, a file of fault geometry.

The command files can be typed in when the programs are run, but it is better to create then separately and pipe then to the programs. All the command files have a common style: a list of commands, one per line, terminated by an `execute` command, which causes the program to perform the tasks specified. The commands before the `execute` command may be in any order. If the same command is given more than once with different settings, the bottommost line in the list will be used. If the `execute` command is omitted, the end-of-file will trigger the execution.

This manual begins with a few simple examples (themselves included in the distribution), to illustrate how the programs can be used. Section 3.4 outlines some more complex examples that have been included to show additional features. Section 4 provides a reference for all the commands used in the command files, and Section 5 describes all the files used as input or output, while Section 6 describes the log files produced by `fakenet` to provide a record of what was done. Section 7 describes the mathematics of the computations.

3.1 Basic Commands for `slipmake`

This program reads an input file that specifies fault geometry; the command file describes how slip evolves on the fault. A minimal fault geometry file would look like:

```
35.0 -118.9 35.0 -119.0 -2 4 30
```

which describes a fault plane striking EW and dipping 30° to the North, with a top at 2 km depth and a down-dip length of 4 km (see Section 5.1 for more information). If this file was called `file.fault`, a basic command file for `slipmake` would be

```
faults file.fault
startslip 2009 100
rake 90
sample 1
ratecurve loglo
parameters 0.0001 1
output file.slip
moment -6
execute
```

which reads in the file, and writes out a file called `file.slip` that has slip equivalent to an earthquake of M_w equal to 6, rake 90°, and a time constant of 10^{-4} day, starting at 2009:100:00;00:00. The slip is output at 1 day intervals. The output slip file `file.slip` contains the lines:

```
g 35.0000 -118.9000 35.0000 -119.0000 t t 2.0 4.0 30.0
s 0.91727 90 2009 101 0 0 0
```

which describe the fault geometry (line starting with g) and the slip history (line starting with `s`): in this case, that one day after the start, the total slip is 0.917 meters. The file also contains other lines for plotting and diagnostics; see Section 5.1.

3.2 Basic Commands for `fakenet`

Now that we have a slip model, we can produce a time series. To do this, we will need to have a station file, which we name `file.sta`, and which gives station names and locations:

```
AAAA 35.05 -118.95 0
BBBB 35.00 -118.95 0
```

If the time series are to have slopes, we also need a velocity file, which we name `file.vel`, and which gives station velocities:

```
AAAA 10.  10.  0.
BBBB 12.  8.  0.
```

And if the time series are to be noisy, we also need a station noise file, which we name `file.noise`, and which gives the noise models for each station:

```
w AAAA 1.  1.  2.
r AAAA 1.  1.  3.
f BBBB 1.  1.  1.5
```

This gives station `AAAA` a mix of white and random-walk noise, and `BBBB` flicker noise, in all cases largest in the vertical.

If we ignore, for the moment, these last two files, the command file for `fakenet` then looks like:

```
location file.sta
faults file.slip
 velocity file.vel
 stanoise file.noise
begin 2009 100
end 2009 102
sample 1
execute
```

where we have effectively omitted the `velocity` and `stanoise` commands, by putting a space before them. This means that the output files from `fakenet` contain only the displacements from the slip; an example of the flexibility of the system, since in this mode it can be used to get coseismic displacements.

Running `fakenet` with this command file produced a file with the name `2009.245.192554.tar.gz`. To make the file name unique, it is simply the date, encoded as year.day.hourminse–so this name will not be the result you get on running this example.

If we uncompress this file and extract its contents, we get two files named `AAAA.2009.245.192554.csv` and `BBBB.2009.245.192554.csv`. These look like

```
Station AAAA, 35.0500, -118.9500, 0.00
2009-04-10, 0.000, 0.000, 0.000, 54931.0000
2009-04-11, -0.031, -88.901, -46.006, 54932.0000
2009-04-12, -0.031, -88.901, -46.006, 54933.0000
```

and

```
Station BBBB, 35.0000, -118.9500, 0.00
2009-04-10, 0.000, 0.000, 0.000, 54931.0000
2009-04-11, -0.057, -113.934, 294.873, 54932.0000
2009-04-12, -0.057, -113.934, 294.873, 54933.0000
```

In each of these, the first line gives the station information; subsequent lines give the time series, in the form of date, East, North, and Up displacements (in mm), and Modified Julian Date. In this case, we just have a "coseismic" step.

If we were to change the command file to ignore the fault slip, and read the noise and velocity files, so that it looks like:

```
location file.sta
 faults file.slip
velocity file.vel
stanoise file.noise
begin 2009 100
end 2009 102
sample 1
execute
```

we would get time-series files

```
Station AAAA, 35.0500, -118.9500, 0.00
2009-04-10, -0.391, 2.490, 1.057, 54931.0000
2009-04-11, -0.795, 1.231, 0.059, 54932.0000
2009-04-12, -0.437, 0.141, 0.817, 54933.0000
```

and

```
Station BBBB, 35.0000, -118.9500, 0.00
2009-04-10, 0.000, 0.000, 0.000, 54931.0000
2009-04-11, 0.036, -0.175, 0.468, 54932.0000
2009-04-12, -0.333, 0.122, 0.199, 54933.0000
```

which have a random variation with time.

3.3 Basic Commands for `commake`

To demonstrate the preprocessor for common-mode noise, we need a file `file.sta` with the same format as before, and also a file `file.comm1`, which looks like

```
w 1 0 2
r 2 2 4
f 2 1 4
```

in which each line contains noise parameters for the three directions: in this case, one line specifying white noise (line that starts with `w`); one line specifying random-walk noise (line that starts with `r`); and one line specifying flicker noise (line that starts with `f`). The simplest command file reads in these inputs, and writes out an output file `file.comm2`:

```
common file.comm1
output file.comm2
location file.sta
execute
```

with the output file looking like

```
l 35.02500 -118.95000
w 1.0000 0.0000 2.0000
f 2.0000 1.0000 4.0000
r 2.0000 2.0000 4.0000
```

which tells `fakenet` to: (1) generate, for each component, a timeseries of noise with the description given; (2) "locate" these at the coordinates given (in this case, at the centroid of our two-station network); and (3) apply them to all the stations. In this particular case the location of the series is irrelevant; Section 7.2 explains how this is used if there are multiple locations.

We could now use a command file

```
location file.sta
common file.comm2
begin 2009 100
end 2009 102
sample 1
execute
```

for `fakenet`; since this only specifies the common-mode signal as a source of displacements the time series are all the same, so we only show one of them:

```
Station AAAA, 35.0500, -118.9500, 0.00
2009-04-10, 0.000, 0.000, 0.000, 54931.0000
2009-04-11, 0.050, -0.356, 1.798, 54932.0000
2009-04-12, 0.407, -0.354, -3.563, 54933.0000
```

3.4 Examples

The `Examples` directory includes the examples given above, and also some more complicated ones, described here, that illustrate other features of the programs.

3.5 Additional Examples for `slipmake`

We start by seeing how `slipmake` can be used to subdivide a fault plane into subsegments. We add `length` and `depth` commands to the file we used in Section 3.1 to get:

```
faults file.fault
startslip 2009 100
rake 90
sample 1
ratecurve loglo
length 4
depth 3 9 3
parameters 0.0001 1
output file.slip
moment -6
execute
```

which produces an output file containing the lines

```
g 35.0155 -118.9000 35.0155 -118.9500 t t 3.0 6.0 30.0
s 0.30520 90 2009 101 0 0 0
g 35.0622 -118.8999 35.0622 -118.9500 t t 6.0 6.0 30.0
s 0.30520 90 2009 101 0 0 0
g 35.0155 -118.9500 35.0155 -119.0000 t t 3.0 6.0 30.0
s 0.30520 90 2009 101 0 0 0
g 35.0622 -118.9500 35.0622 -119.0001 t t 6.0 6.0 30.0
s 0.30520 90 2009 101 0 0 0
```

along with diagnostic lines. The `length` command specifies an along-strike length of 4 km; the actual length used is chosen to be as close to this as possible while also dividing the fault plane, along strike, into intervals of equal size. In the form shown, the `depth` command projects the fault plane up or down to get the requested top and bottom depths (reckoned positive down), and then divides this into an equal number of depth intervals, chosen to be close to the interval asked for through the third argument. In this case, the dip of the fault plane is such that 2 km down-dip is 1 km in depth, so we get exactly depth the interval requested: one pair of subsegments runs from depth of 3 km to 6 km depth (expressed in the first `g` line as a 3 km depth for the top of the fault, and a 6 km down-dip width); and the other pair runs from 6 km to 9 km (expressed in the second `g` line as a 6 km depth for the top of the fault, and the same down-dip width).

This, we end up with four `g` lines describing subsegments of the fault plane, and four `s` lines describing the slip on each subsegment: in this case the slips are all the same. Since the computed displacements would be unchanged, this subdivision, in this case, would have no effect.

But with other options this might not be so, especially if we wish to model propagating slip. To illustrate propagation, we use two widely separated faults on an EW line, the western one having twice the area of the eastern one, so that `file.fault` is:

```
35.0 -118.9 35.0 -119.0 -2 4 30
35.0 -119.9 35.0 -120.1 -2 4 30
```

We set up propagation in the command file:

```
faults file.fault
startslip 2009 100
rake 90
sample 1
ratecurve loglo
parameters .001 1
output file.slip
propagate 1 90
equalize m
moment -6
execute
```

This has three differences from the file used in Section 3.1: we have made the width of the fault-slip function 10 times larger, equalized the moment for the two faults, and introduced a propagation speed of 1 km/day at an azimuth of 90° measured East of North: that is, due East. Then `slipmake` produces an output file that contains the lines:

```
g 35.0000 -118.9000 35.0000 -119.0000 t t 2.0 4.0 30.0
d 35.0000 -118.9500 95.85182 47.9
s 0.45560 90 2009 196 0 0 0
s 0.00268 90 2009 197 0 0 0
s 0.00019 90 2009 198 0 0 0
s 0.00010 90 2009 200 0 0 0
s 0.00010 90 2009 240 0 0 0
g 35.0000 -119.9000 35.0000 -120.1000 t t 2.0 4.0 30.0
d 35.0000 -120.0000 0.00000 -47.9
s 0.22911 90 2009 101 0 0 0
s 0.00011 90 2009 102 0 0 0
s 0.00010 90 2009 116 0 0 0
```

Here we show diagnostic lines that start with `d`: these give the coordinates of the centroid of each subsegment (here taken to be the midpoint of the top edge), the delay time in days (which is always adjusted so the earliest delay is zero), and the distance, in km and along the direction of propagation, from the centroid of all the faults. If there is no propagation, only the centroid is given.

Because of the propagation, the eastern fault starts to slip 95.8 days after the western one, as shown by the dates of the first of the `s` lines for each fault. Because of the equalization of the moments, the larger western fault has half the slip of the eastern fault And, the larger time constant has spread the slip out over a longer time. The minimum temporal spacing is set by the `sample` command to be one day. However, slips are only given when the cumulative slip since the last entry exceeds 1 mm: this maintains accuracy while minimizing the number of dislocation solutions needs to be computed by `fakenet`.

3.6 Additional Examples for `commake`

To demonstrate additional features of `commake`, we need a longer input noise file (`file.comm1`) than the one we used in Section 3.3. This file will illustrate an important aspect of both `commake` and `fakenet`, namely their ability to select possible values at random, making it easy to run multiple simulations with noise characteristics that are the same overall, but differ from run to run.

```
w 1 0 2
w 1 2 4
r 2 2 4
r 1 1 2
f 3 2 3
f 2 1 4
s 2.  1.  1.  0.  90.  180.  365.25 2006 1 0 0 0
s 1.  0.  1.  0.  90.  180.  365.25 2006 1 0 0 0
s 3.  2.  1.  0.  0.  180.  183.  2006 1 0 0 0
s 1.  2.  3.  0.  0.  180.  183.  2006 1 0 0 0
```

As explained more fully in Section 5.3, the first six lines are possible noise levels in the three components, and the last four lines describe possible sinusoidal signals. If we run `commake` with the command file

```
common file.comm1
output file.comm2
distance 4 200
location file.sta
nsines 1
execute
```

the output (`file.comm2` is

```
l 36.82751 -118.95000
w 1.0000 2.0000 2.0000
f 3.0000 2.0000 3.0000
r 2.0000 1.0000 2.0000
s 1.0000 2.0000 3.0000 0.0 0.0 -180.0 183.0000 2006 1 0 0 0
l 35.00521 -116.75834
w 1.0000 2.0000 4.0000
f 2.0000 2.0000 4.0000
r 2.0000 1.0000 2.0000
s 1.0000 0.0000 1.0000 0.0 0.0 -180.0 365.2500 2006 1 0 0 0
l 33.22196 -118.95000
w 1.0000 0.0000 2.0000
f 2.0000 1.0000 4.0000
r 1.0000 1.0000 2.0000
s 3.0000 2.0000 1.0000 0.0 0.0 -180.0 183.0000 2006 1 0 0 0
l 35.00521 -121.14167
w 1.0000 2.0000 4.0000
f 2.0000 1.0000 3.0000
r 2.0000 2.0000 2.0000
s 1.0000 2.0000 3.0000 0.0 0.0 -180.0 183.0000 2006 1 0 0 0
```

This describes four sources of common-mode noise, each 200 km from the centroid of our two-station network, the positions being given by the `l` lines. The subsequent lines describe noise levels and a sinusoidal signal for the source; we can say "a sinusoidal signal" because the `nsines` command has limited the number of sinusoids for each source to one. Note that the noise level for each noise type and component is chosen from all levels provided for that noise type and component, so that the output can contain different combinations noise levels than any of the ones provided in the input.

3.7 Additional Examples for `fakenet`

The `samnoise` command in `fakenet` randomly selects noise levels from a menu, in the style we have already seen in `commake`. For example, using a file of possible noise levels `file.noises`:

```
w 1 0 2
w 1 2 4
r 2 2 4
r 1 1 2
f 3 2 3
f 2 1 4
```

and a command file

```
location file.sta
samnoise file.noises
begin 2009 100
end 2009 102
sample 1
execute
```

would be equivalent to using the command `stanoise` to read a file that looked like

```
w AAAA 1.0000 2.0000 2.0000
f AAAA 3.0000 2.0000 3.0000
r AAAA 2.0000 1.0000 2.0000
w BBBB 1.0000 0.0000 4.0000
f BBBB 2.0000 1.0000 3.0000
r BBBB 1.0000 2.0000 2.0000
```

Finally, `fakenet` can read in a file that describes when different stations were running; that is, the performance of the network. Such a file might look like

```
s AAAA 2004 001 m AAAA 2004 100 99 e AAAA 2004 366
```

which shows that the first and last days of data from station `AAAA` are the beginning and end of 2004, and that days 101 through 199 are missing. So, for this station, `fakenet` would write out a timeseries with these characteristics.

4 Command Reference

This section describes the available commands, in alphabetical order. The command names are given in `this font`, and arguments in *this one*. The first four letters of the command names are slightly separated from the rest, since these abbreviated forms are all that need to be input. Not all commands are used in all programs; Table 1 shows which commands are required (•) or optional (○) for each program, with a blank space meaning that the command is not used.

`annu al` *filename* In `fakenet`, this gives the name of the file of amplitudes and modulations of sinusoids (for each station) that is used to produce seasonal variations.

`area` *south north west east*: In both `commake`, and `fakenet`, this selects the area to be covered; when a file of station coordinates is read in, stations outside this area will be ignored. Coordinates are in decimal degrees, East positive.

`begi n` *yr day [h m s]*: In `fakenet`, this sets the start time of the output series; note that this is given in year and day of the year. For simulation of daily GPS series, only the year and day need to be entered; to allow simulation of finer intervals, the hour, minute, and second can be added.

`cent roid` *top* | *mid*: In `slipmake`, this sets how the centroid of each subsegment is found; the

Available Commands in Fakenet

Command	`slipmake`	`commake`	`fakenet`	Command	`slipmake`	`commake`	`fakenet`
`annu al`			○	`nsin es`		○	
`area`		○	○	`outp ut`	•	•	
`begi n`			•	`para meters`	•		
`cent roid`	○			`perf ormance`			○
`clea r`	○	○	○	`prop agate`	○		
`comm on`		•	○	`rake`	•		
`dept h`	○			`rann ual`			○
`dist ance`		•		`rate curve`	•		
`end`	•		•	`revi ew`	○	○	○
`equa lize`	•			`samn oise`			○
`exec ute`	•	•	•	`samp le`	•		•
`faul ts`	•		○	`seed`		○	○
`leng th`	○			`stan oise`			○
`loca tion`		•	•	`star tslip`	•		
`logo ut`			○	`velo city`			○
`modu late`			○	`verr or`			○
`mome nt`	•			`widt h`	○		
`msee d`			○				

Table 1:

locations of these centroids, relative to the centroid of all of them, determines how slip is delayed or advanced by the `propagate` command. If the argument is `top`, the centroid is the center of the top edge of each subsegment; if it is `mid`, the centroid is the center of each subsegment plane. (The latter is preferred; the former is available for backwards compatibility).

`clea r` *command*: In all three programs, this deletes the most recent occurrence of a command.

`comm on` *filename*: In `commake`, this gives the name of the file of common-mode noise parameters used to create common-mode noise values for each station (which is output by the program). In `fakenet`, the same name would be used to read the file output by `commake`, Section 5.3 describes the file format.

`dept h` *top [bottom [interval]]*: In `slipmake`, this sets the depths of the top and bottom of a slipping region, in km, positive down, and also the depth interval used to divide the fault plane into subsections. For example, `depth 5 10 1` would create 5 subsections whose tops were at depths of 5, 6, 7, 8, and 9 km. If this command is omitted, the top and bottom will be those from the fault file, and the depth interval will be the difference in their depths, so that there will be no subdivision of the plane. If only two arguments are given, the depth interval will (again) be their difference, with no subdivision; if only one argument is given, it will be taken to be the depth of the top; the depth of the bottom will be that present in the fault file. If this command is used, the `width` command may not be used.

`dist ance` *number distance*: In `commake`, this sets (1) the number of generators of common-mode noise, and (2) the distance, in kilometers, that they are from the centroid of the network.

`end` *yr day [h m s]*: In `fakenet`, this sets the end time of the output series; note that this is given in year and day of the year. For simulation of daily GPS series, only the year and day need to be entered; to allow simulation of finer intervals, the hour, minute, and second can be added.

`equa lize` *m*|*s*: In `slipmake`, this sets how the moment will be distributed over the subsections of a fault plane. If `m` is used, the moments will be equal for all subsections; if `s` is used, the slip will be the same on all subsections.

`exec ute`: In all three programs, this causes calculations to start, based on the commands read up to that point.

`faul ts` *filename*: In `slipmake`, this gives the name of the file of fault slips at specific times which is output by the program; in `fakenet`, the same name would be used to read the file. Section 5.1 describes the file format.

`leng th` *length*: In `slipmake`, this sets the approximate length interval, in km, used to divide the fault plane into subsections.

`loca tion` *filename*: In `commake`, and `fakenet`, this is the name of the file containing station names and locations. Section 5.2 describes the file format.

`logo ut` *[filename]* (formerly `logf ile` *[filename]*): In `fakenet`, this sets the name for the "log file", which contains the settings that were used for the program run (and could be used to rerun it).

This file is actually a shell script that contains, and can create, all the files needed to rerun the program, with both deterministic and randomly-chosen parameters specified. If no name is given (the preferred mode), the file name will be `yyyy.ddd.hhmmss.log`, where `yyyy` is the current year, `ddd` is the current day of the year, and `hhmmss` are the hour, minute, and second of the current time; this will match the name of the tarfile containing the time series. If a name is given, the name used will be the first 15 characters of that name. See Section 6 for a full discussion.

`msee d` *seed*: In `fakenet`, this command sets the (integer) value for the random number generator used in perturbing the velocities and sampling from the lists of possible noise values and annual amplitudes and modulations. If this command is not given, the default value is 12345.

`modu late` *amp phase* In `fakenet`, this sets the maximum amplitude and phase modulation levels to be used for annual sinusoids; the actual levels will be assigned randomly between zero and the maximum. The amplitude modulation is the amount of variation away from a constant, so 0.1 would imply roughly 10% variation. The phase variation is in radians.

`mome nt` *totalmom*: In `slipmake`, this sets the total moment release in N-m. If the value entered is negative, it will be taken to be a moment magnitude. If the `ratecurve` function is *linear* then *totalmom* will be taken to be a slip rate in mm/yr.

`nsin es` *number*: In `commake`, this sets the maximum number of sinusoids that can be applied to a generator of common-mode noise.

`outp ut` *filename*: In `commake`, and `slipmake`, the filename for the output file. For `commake`, this is the file of common-mode parameters. For `slipmake`, this is the file of fault subsections with slips and times. For `fakenet`, this command is not used; the files of simulated time series are always named `ssss.yyyy.ddd.hhmmss.csv`, where `ssss` is the station code, `yyyy` is the current year, `ddd` is the current day of the year, and `hhmmss` are the hour, minute, and second of the current time.

`para meters` *width [shape1[shape2]]*: In `slipmake`, this sets the parameters of the curve of moment release. The first parameter, which is in days, sets the relative width of the curve. The second and third parameters set the shape of the curve. See `ratecurve`, below, for a fuller description.

`perf ormance` *filename*: In `fakenet`, this sets the name for the file that describes network performance. Section 5.2 describes the file format.

`prop agate` *speed azimuth*: In `slipmake`, this sets the propagation speed (in km/day) and direction (in degrees, clockwise from North) of a pulse that propagates through the network, to create a spatial dependence of fault slip. The start time given by `startslip` will be applied to the first time to occur on the all the subsections, and the time delay between subsections will be given by the distance from the centroid to the center of each subsection, divided by the velocity, and scaled by the dot product of a unit vector from the centroid to the subsection, and a unit vector along the specified azimuth. So, for example, if the subsections are in an East-West line, and the direction of propagation is 0°, the slip will begin at the same time on all subsections because the dot product is zero. As shown in Section 3.5, a direction of 90° means that the slip is earliest on the west end and latest on the east end.

`rann ual` *filename* In `fakenet`, this gives the name of the file of amplitudes and periods of sinusoids that is sampled from to create seasonal variations. The `modu late` command is used to set the levels of amplitude and phase modulation applied to these sinusoids to make them more realistic.

`rake` *angle*: In `slipmake`, this sets the direction of slip, degrees. A value of 180° is right-lateral slip, 0° is left-lateral, 90° is reverse (thrust), and −90° is normal.

`rate curve` *name*: In `slipmake`, this sets the type of moment release curve. The functions currently supported are:

loglogistic	$\frac{\tau^a}{1+\tau^a}$	
burr	$1+(\tau^a)^b$	$(b<0)$
weibull	$1-\exp(-\tau^a)$	
linear	τ	
lntime	$\ln(1+\tau)$	

where τ is related to actual time by $\tau = t/w - t_s$, with w being the *width* parameter, and t_s being the time given by the `startslip` command; a and b are the first and second shape parameters from the `parameters` statement. All the functions are zero for $t < t_s$; Figure 2 shows curve values for different values of a and b.

`revi ew`: In all three programs, this prints out the commands entered and not cleared to that point.

`samn oise` *filename*: In `fakenet`, this gives the name of the file that describes noise parameters to be assigned to stations at random.

`samp le` *days*: In `slipmake` this sets the minimum sample interval (in days) for describing the fault slip. In `fakenet`, this sets the sample interval (in days) for the station timeseries. If this command is not given, the interval (in both programs) defaults to 1.0.

`seed` *seed*: In `commake` this command sets the (integer) seed value for the random number generator. In `fakenet`, this command sets the value for the random number generator used in producing the noise series. If this command is not given, the default value is 12345.

`stan oise` *filename*: In `fakenet`, this gives the name of a file that specifies noise parameters for each station.

`star tslip` *yr day [h m s]*: In `slipmake`, this sets the start time of slip on a fault. The year and day of the year are required; to allow simulation of finer intervals, the hour, minute, and second can be added.

`velo city` *filename*: In `fakenet`, this gives the name of a file that specifies velocities for each station.

`verr or` *vn ve vu*: In `fakenet`, this gives the name of a file that specifies the amount of random variation to be assigned to the velocities that are read in.

`widt h` *w* In `slipmake`, this sets the widths, along the down-dip plan, of the subdivisions of the fault. If this command is used, the `depth` command may not be used.

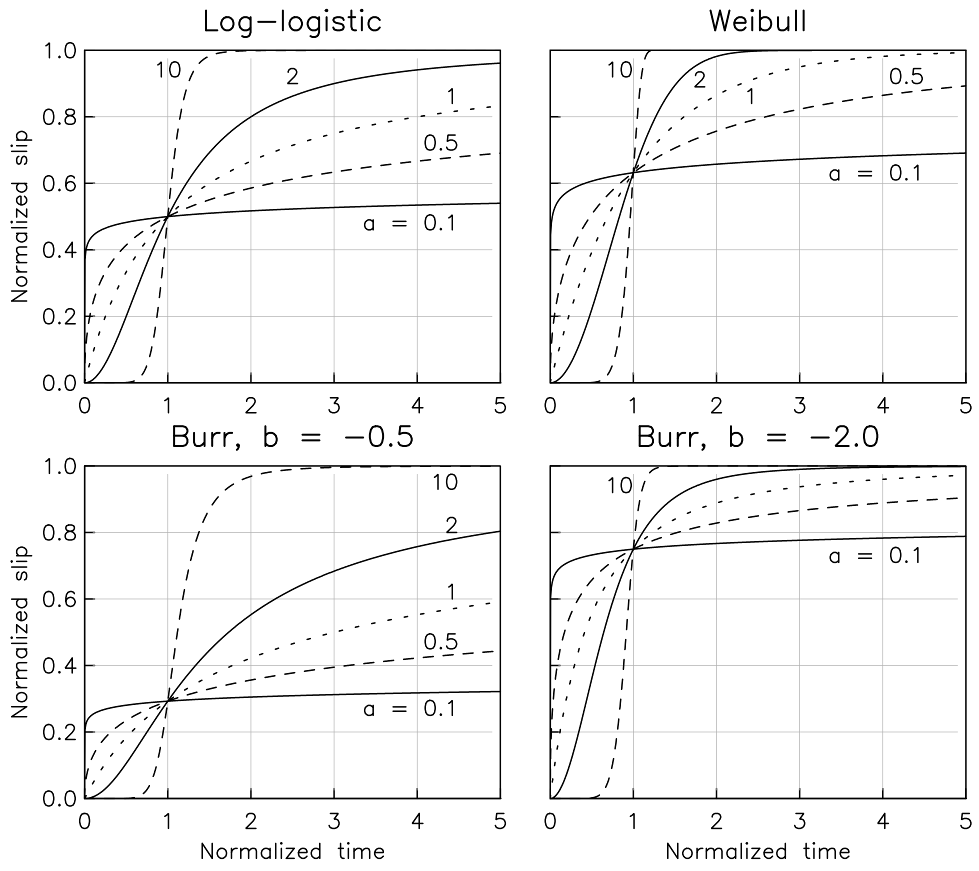

Figure 2: Cumulative slip, for different choices of functions and parameters.

5 File Formats

5.1 Fault Files

There are two types of fault files: the fault-plane files that are read by `slipmake`, and the slip files that are created by it, and read in by `fakenet`.

The fault-plane file contains a series of lines, each of which specifies the location, size, and orientation of a rectangular fault plane. An example would be

```
 -118.5881 33.9358 -118.6096 33.9680 16 38.2 -5.2
```

in which the first four entries give the longitude and latitude of the two upper corners of the plane (in this case, part of the Compton thrust). The next entry is the dip, in degrees, which is measured down from a horizontal line that is perpendicular to the top edge, and running to the right, if looking from the first corner to the second. In the example given, the top edge runs West and slightly north, so the perpendicular line points to the NE, and the fault plane dips 16° down from the horizontal. The following entry gives the distance (in km) from the top to the bottom edge of the fault plane (measured in the plane), and the final one the elevation of the top edge, in this case -5.2 km to show that it is at a depth of 5.2 km below sea level.

The slip file, which is output by `slipmake`, and read in by `fakenet`, contains five types of lines, for example:

```
 g 33.8701 -118.4253 33.8836 -118.4584 t t 6.0 3.6 16.0
 d 33.8769 -118.4419 2.22416
 c 33.8701 -118.4253 6.0
 c 33.8983 -118.4090 7.0
 c 33.9117 -118.4421 7.0
 c 33.8836 -118.4584 6.0
 a 12.265217
 s 0.00122 90 2001 224 12 0 0
 s 0.00156 90 2001 225 12 0 0
```

The first line, which describes the location and orientation of the slip plane, has a format similar to the lines in the fault file. The first two entries give the geographical coordinates of the top edge of the plane; this is followed by `t t` to indicate that the coordinates, and the depth, refer to the top edge. The next entry is the depth of the top edge (km, positive down), and the one following that is the down-dip length of the fault (in km). The final entry is the dip of the plane, in degrees; this is measured down from a horizontal line that is perpendicular to the top edge, and running to the right, if looking from the first corner to the second.

The lines beginning with `d` and `c` are informational and not used by `fakenet`. The line beginning with `d` gives the coordinates of the centroid, the delay (in days) applied when computing the slip, and the distances, along the direction of propagation, from the centroid of all the fault segments. The last two entries are zero if there is no propagation. The lines beginning with `c` give the locations (coordinates and depth, down positive) of the four corners of the subsegment; this is useful for making plots.

The lines beginning with `s` give the amount of slip (in meters) on the plane, the rake (in degrees), and the time at which the slip happened. The minimum time interval between these lines is set by the `sample` command in `slipmake`; slip is given only when the cumulative amount reaches 1 mm or more. A rake of 180° is right-lateral slip, 0° is left-lateral, 90° is reverse (thrust), and −90° is normal.

5.2 Station Files

Four types of files contain station-related information.

The one required file is the *location file* which contains station codes and coordinates (latitude, longitude, and height). A typical line in this file would be `pin1 33.61216 -116.45816 1256.17`

The *velocity file* gives the station velocities, which are used to put a trend into the time series; these can be randomly perturbed with the `verror` command. A typical line in this file would be `pin1 -16.8 18.4 -0.9`, the three values being the velocities (in mm/yr) in East, North, and Up.

The *performance file* describes the times of operation for all the stations. There are three types of entries for a station. The first one, for example `s pin1 1992 135`, gives the time of the first day of data. This should be followed by entries such as `m pin1 1992 180 2`, which indicate that there are two days missing after the date given (e.g., in this example, days 181 and 182 of 1992). Finally, there needs to be an entry of the form `e pin1 2009 130` to give the time of the last day of data.

Finally, the *station noise file* describes the noise parameters for each station. Usually these parameters would be selected randomly for each station based on noise parameters given in the noise file described in Section 5.3. However, an alternative is to specify the noise values for each station separately. The lines in this file look like

```
 w pin1 0.55 0.47 2.24
```

where the first letter is one of `w` (white noise), `f` (flicker noise), or `r` (random walk). The second entry is the station code, and the remaining three are the amplitudes for the noise in East, North, and Up; see Section 5.3.1 for the normalizations.

5.3 Noise Files

There are two types of noise files, one for specifying the noise models to be used for the common-mode noise, and one for the models to be used for the station noise. Both types can be thought of as giving a “menu” of possible values, which is selected from at random to produce the actual values used at each station (or each source of common-mode noise).

The format of the files that specify noise levels for `commake` is

```
w 1 0 2
w 1 2 4
r 2 2 4
r 1 1 2
f 3 2 3
f 2 1 4
s 2.  1.  1.  0.  90.  180.  365.25 2006 1 0 0 0
s 1.  0.  1.  0.  90.  180.  365.25 2006 1 0 0 0
```

If the first letter is one of `w` (white noise), `f` (flicker noise), or `r` (random walk), the three numbers are the amplitudes for the noise in East, North, and Up; see Section 5.3.1 for the normalizations. There may be as many as 100 specifications of each type. If the first letter is `s`, this specifies a sinusoid. The next three entries in such a line are the amplitudes (in mm) for the displacement in East, North, and Up, the next three are the phases, and the next one the period in days. The last 5 entries are the time that the phase is relative to, in year, day of year, hour, minute, and second.

The other type of file specifies general noise levels to be sampled from in `fakenet` (as opposed to levels for specific stations) and is read using the `samnoise` command. The lines in this file look like

```
w 0.55 0.47 2.24
```

where the first letter is one of `w` (white noise), `f` (flicker noise), or `r` (random walk). The next three entries are the amplitudes for the noise in East, North, and Up; see Section 5.3.1 for the normalizations. There may be up to 1000 lines of each type.

5.3.1 Normalizations for Noise Models

The amplitude used for white noise is the standard deviation of the series.

The amplitude used for a random-walk is the expected deviation in 365.25 days; that it, it corresponds to values given as $mm/yr^{0.5}$.

Normalizations for flicker noise are not completely standardized. In Fakenet, the normalization has been chosen to be consistent with that used in Langbein (2008), by comparing estimated power spectra for suitably-normalized time series (J. Langbein, pers. comm., 2009).

5.4 Annual-cycle Files

Like the files of possible noises (5.3), this file provides a "menu" of possible amplitudes and periods for cyclical variations. This will be sampled randomly to assign actual values to each station. The lines in this file look like

```
1.23 0.65 3.45
```

where the three entries on each line are the amplitudes for the noise in East, North, and Up. There may be up to 1000 lines of each type. There is also a file format that gives the amplitude, amplitude modulation, and phase modulation (in that order) for each of three coordinates, at each station; this is used mostly for providing information for re-runs in the audit file. The lines in this file look like:

```
pin1 1.23 0.08 0.05 0.65 0.02 0.09 3.45 0.10 0.01
```

6 Log Files for Reruns

In order to provide an "audit trail" for what was done, including the command `logfile name` in `fakenet` causes the program to write out a file called `name`. Note that this file is *not* included in the tar package of velocities. This file is actually a shell script; executing it will create files named `tst.coords`, `tst.vels`, `tst.perf`, `tst.noise`, `tst.slip`, and `tst.commod`, which contain all the information needed to run `fakenet` using the commands in the file `tst.cmmds`, also written out by the script. This command file looks like (e.g.)

```
seed 12345
begin 2009 100 0 0 0
end 2009 102 0 0 0
samp 1.000000
area -90.0000 90.0000 -180.0000 180.0000
location tst.coords
performance tst.perf
velocity tst.vels
faults tst.slip
stanoise tst.noise
common tst.commod
execute
```

Note that the noise file, being read in using `stanoise`, has the noise levels defined explicitly for each station, not drawn at random, and that velocities are not subject to random variation.

7 Algorithms

Any program that attempts to simulate random processes needs to be based on a "good" pseudo-random number generator (PRNG). Both `commake` and `fakenet` use the "Mersenne twister" of Matsumoto and Nishimura (1998), in a Fortran version by Tsuyoshi Tada. L'Ecuyer and Simard (2007), in their review of PRNG's, show that this is one that passes all the tests so far devised. This, like other PRNG's, produces random numbers uniformly distributed between 0 and 1. For some purposes random numbers with a Gaussian distribution are needed; these are computed using the polar transform (Thomas *et al.*, 2007).

7.1 Noise Models

The noise model for white noise is simply uncorrelated Gaussian random variates; random-walk noise is gotten by summing such variates (note that this does not given an exactly f^{-2} power spectral density throughout the entire frequency range.

The noise model for flicker noise comes from an algorithm developed by R. S. Voss and described in Gardner (1978). Take M PRNG's (in this case generators of independent Gaussian variates), and let the output of the m-th one, sampled k times, be r_{mk}. Then form the sum

$$x_n = \sum_{m=1}^{M} r_{mj} \tag{1}$$

where j is the integer part of $n/2^m$. This amounts to sampling from the first PNRG at every value of n, the second at every second value, the third at every fourth value, and so on. Summing these produces a series whose autocorrelation falls of as the logarithm of the lag, which gives the appropriate f^{-1} power spectral density.

7.2 Common-Mode Noise

The basic idea for generating noise that is approximately common-mode for nearby stations is to produce N time series, each of which is associated with a "common-mode location": we can think of a set of noise generators located at various points in the network. For simplicity, these locations are chosen to be on a circle with radius R and centered at the centroid of the network, and spaced evenly around the circle; R and N are set by the `distance` command. If we set $R = 0$, then the noise generator is at the centroid.

For a given station, let d_n be the distance to the n-th noise generator, in km. We then define the weight for that generator to be $w_n = 1/d_n$ for $d_n > 1$, and $w_n = 1$ for $d_n < 1$. If the n-th noise series is denoted as $[x]_n$, the common-mode noise series becomes

$$\frac{\sum_{n=1}^{N} w_n [x]_n}{\sum_{n=1}^{N} w_n} \tag{2}$$

Figure 3 shows how this can work. The left panel shows a map view of the actual GPS network, and also the centroid, and the locations of the noise generators for $R = 300$km, and $N = 4$. The right panel shows the noise series at 11 points along the line A-B, running from the North to the South generator. The gradual change with distance is clear: this signal will be more closely correlated for nearby than distant stations.

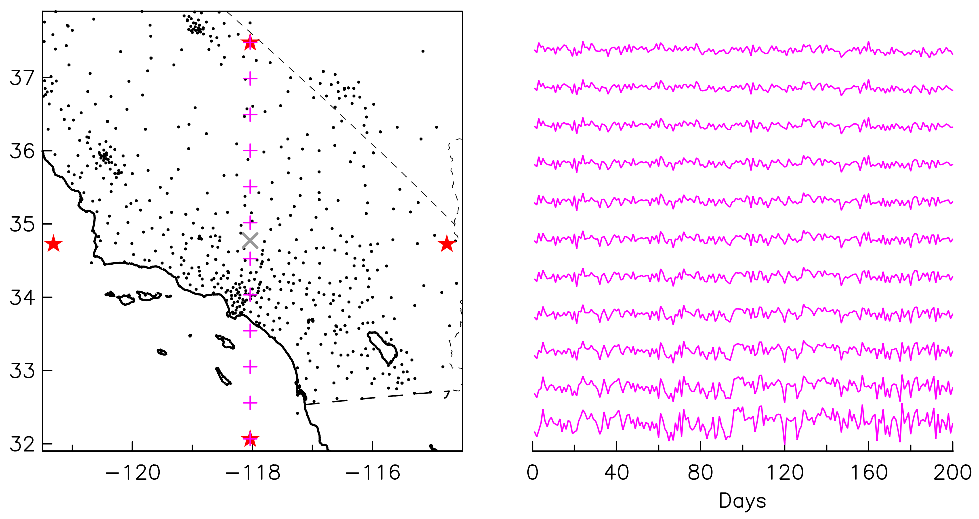

Figure 3: Left panel shows stations (dots), centroid (cross), and locations of noise generators (stars). Right panel shows time series that are computed for points along the line N-S (pluses).

8 Miscellaneous

To report bugs or ask questions, contact Duncan Agnew at the address given above.

The code for reading commands and using them to guide the execution was developed by Bob Parker.

I thank Adriano Gualandi for unearthing, and solving, several problems that appeared in moving to a newer Fortran compiler.

References

Agnew, D. C. (2013), Realistic simulations of geodetic network data: The Fakenet package„ *Seismol. Res. Lett.*, **84**, 426–432, doi:10.1785/gssrl.84.3.426.

Gardner, M. (1978), Mathematical games: white and brown music, fractal curves and 1/f fluctuations, *Sci. Am.*, **239(4)**, 16–32.

Langbein, J. (2008), Noise in GPS displacement measurements from Southern California and Southern Nevada, *J. Geophys. Res.*, **113**, B05,405, doi:10.1029/2007JB005247.

L'Ecuyer, P., and R. Simard (2007), TestU01: A C library for empirical testing of random number generators, *ACM Trans. Math. Softw.*, **33**, 22, doi:10.1145/1268776.1268777.

Matsumoto, M., and T. Nishimura (1998), Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Trans. Modeling Comput. Simul.*, **8**, 3–30.

Thomas, D. B., W. Luk, P. H. W. Leong, and J. D. Villaseñor (2007), Gaussian random number generators, *ACM Computing Surv.*, **39**, 11.