MANUAL FOR EST_NOISE7.2X

JOHN LANGBEIN
US GEOLOGICAL SURVEY
MENLO PARK, CA
LANGBEIN@USGS.GOV

1. Introduction

The program *est_noise7.2x* does time-series analysis to extract trends (velocities), offsets, and other parameters that characterize GNSS observations and simultaneously estimate the parameters the characterize the background noise of the data. Basically, it does least-squares regression, but importantly, it quantifies the temporal correlations of the data to provide unbiased estimates of the parameters and their statistical errors that model the time dependences of the observations. Standard least-squares regression typically assumes that each observation is independent, and drawn from a Gaussian distribution. However, for many time-series, the data are temporally correlated. *est_noise7.2x* simultaneously "measures" the temporal correlations and estimates the usual parameters that one often seeks from a time series. The optimization is measured using the maximum likelihood estimator. Although this code was written primarily for GNSS observations of positions, it can be applied to other time series, too.

est_noise7.2x is an update of my older program, *est_noise6.50*. There are several improvements:

(1) It provides a mechanism to analyze large data sets rapidly. For data with no gaps in the time series, the speed-up can be up to a factor of 50. Typically, for many GPS time series, which have a few gaps, the speed-up is about a factor of 8.
(2) It can take a variety of time formats including modified julian day and GMT style formats
(3) The data and ancillary matrices are now double precision which minimizes apparent singularities in the inversion of data covariance matrices having large temporal correlations.

Date: January 16, 2020.

(4) Since the input or configuration to this program is a series of answers to questions, the program provides a *journal* file of the responses. One can edit and then reuse the *journal* file.
(5) All subroutines are public domain. The legacy *est_noise6.50* used some subroutines from *Numerical Recipes, Press et al.*. Those included a Fourier Transform, and the *Nelder and Mead* (1965), Downhill simplex optimization codes. I have written my own version of Nelder and Mead and used the public domain FFT from netlib.org
(6) The same set of inputs for *est_noise6.50* can be used with *est_noise7.2x* which yields the same results; that is *est_noise7.2x* is backwards compatible with *est_noise6.50*.

The development and discussion of *est_noise* can be found in the references.

1.1. **Time dependent models.** The following types of time dependence are included with *est_noise*; rate, changes in rate, offsets, sinusoids, exponential, $1 - e^{-t/\tau}$, and Omori decay, $log(1 + t/\tau)$. In addition, more functions can be added. For instance, some observations of interest, such a borehole strainmeter data, one might include the dependence of another set of observations such as atmospheric pressure. The pressure data can be used as another time-dependent function and the program finds the admittance (or gain) of the pressure upon the strain data. Specifying these functions is done by answering questions that the program asks. You will need to know the times of the offsets, the time of the start of the exponential trend, and the period of time spanning the rate change. More information will be provided later in the manual.

1.2. **Noise models for temporal correlations.** The temporal correlations can be modeled in terms of a power spectra, $P(f)$, where f is frequency. *est_noise* has a library of a few, simple noise models which can replicate a variety of temporal correlations. These include, white noise, where $P(f) = \sigma^2/f_{ny}$ where f_{ny} is the Nyquist frequency and σ is the amplitude of white noise. This is represents, uncorrelated observations, and that the data are Normally distributed with a standard deviation of σ.

However, most data sets have temporal correlations that can be represented by a power-law power-spectrum, $P(f) = P_o/f^\beta$. If $\beta = 1$, then this is termed flicker noise, and if $\beta = 2$, then this is termed random-walk noise.

In addition, *Langbein* (2004) provided two more types of noise; generalized Gauss-Markov and bandpass filtered noise. Generalized Gauss Markov noise can be represented by

$$P(f) = \frac{P_o}{[f^2 + (\alpha/2\pi)^2]^{\beta/2}} \tag{1}$$

where the power spectra is frequency independent at the lowest frequencies, but becomes a power law when $f > \alpha/2\pi$. (Note: this corrects equation 15 in *Langbein* (2004)).

For band pass filtered noise, it is represented by:

$$P(f) = {\sigma_{bp}}^2 [\frac{(f/f_h)^2}{(1+(f/f_l)^2)(1+(f/f_h)^2)}]^{2p} \tag{2}$$

where f_l and f_h are the high and low frequency limits of the filter and p is the number of poles.

All of these noise sources can be added together to provide a more complex and realistic representation of the temporal correlations. As *Langbein* (2017) demonstrates, there are two different ways to add these noise sources and that can impact the computational requirements. It customary to assume that each noise source is independent and add them in quadrature, which has been done with the legacy program *est_noise6.50* and is offered as one option in *est_noise7.2x*. Alternatively, each of these noise models is represented by a time-domain equivalent, or filter functions. These filters can be added together to form the covariance matrix. Computationally, this offers some advantages for which the other mode of *est_noise7.2x* uses.

1.3. **Related programs.** The tar-ball contains several other programs that augment *est_noise* which can help test the code and/or help evaluate the results.

bust_5: This removes outliers from time series based upon a running median

compare_wander7: Computes a spectrum of *wander* or drift based upon *Agnew* [1992]. This is useful to assess the quality of the estimated noise model obtained from *est_noise*.

psd_calc7: Computes the equivalent power spectral density using the noise model consist with *est_noise*. This involves re-scaling the coefficients.

gen_noise7: This program generates a time series of simulated, colored noise using the noise models discussed above.

adjust_1: removes sinusoids, offsets, rate, changes in rate, and exponential/Omori decay when given their specified amplitudes. This is used in companion with *bust_5*.

In the `example0` directory, I provide 3 scripts that combine the use of all of these programs to remove outliers and to perform the time series analysis to extract trends and the background noise of the data.

Finally, *Bos et al.* (2012) provides an alternative to *est_noise* and that code can be found at

`http://segal.ubi.pt/hector/`

est_noise primary advantage is that it has no approximations. For power-law noise

with indices greater than 1, *hector* makes an approximation to force the covariance matrix to be Toeplitz and uses a fast Toeplitz solver to invert the covariance. *est_noise*, when using the simple addition of filter functions provides similar computational speed to *hector* without the approximation. In reality, for *hector*, approximation is relatively minor.

2. Compiling the program

The program is written in Fortran, primarily the legacy Fortran 77. Unlike the legacy *est_noise6.50*, I'm not able to provide an executable file. Instead, you will need to compile the program and its subroutines. I've successfully compiled the program on Mac OS and Linux OS, using both gfortran and the Intel Fortran compilers. I have not tried compiling and running the program on Windows OS. The best performance comes with the Intel compiler and its MKL libraries. By using gfortran and the associated openblas library, the computation speed is reduced by approximately 30%. For Linux and gfortran, you may need to download the openblas library.

I have provided a *compile* script; to invoke,

```
./compile.sh
```

For which:

```
Script used to compile est_noise
Usage:  compile.sh mac/linux i/g
where argument one designates the OS, mac or linux
and argument two designates the compiler
i intel ifort
g gfortran
```

For compiling on a Mac, you will need Apple's Xcode installed; I think you get this at Apple's online "App" store. With the Intel option, *i*, this requires Intel Fortran and their Math Kernal Library. Otherwise, you'll need *gfortran* which can be obtained at `http://hpc.sourceforge.net/`

For compiling on Linux either Intel Fortran or *gfortran* will work. If the Intel compiler with MKL is not available, then gfortran is easily obtained (if not already installed). Depending on the Linux distribution, a package installer, such as `yum`, will install gfortran.

In addition, if your are using gfortran on linux, the openblas library needs to be installed either in `/usr/lib` or `/usr/lib64` and the listing will look like:

```
ls -lt /usr/lib/*openblas*
lrwxrwxrwx 1 root root 22 Jan 28 2015 /usr/lib/libopenblas.so.0 -> libopenblas-r0.2.11.so
-rwxr-xr-x 1 root root 15134496 Aug 24 2014 /usr/lib/libopenblas-r0.2.11.so
```

If not present, you can simply download the openblas libraries using a package installer such as `yum`, that is `sudo yum install openblas`, which is used with the

CentOS dialect of Linux. IN ADDITION, you will need to *comment-out* one of two lines in compile.sh script near line 40.

Note for ubuntu OS: For this OS, I downloaded openblas from
`http://www.openblas.net/` (look for download tab near top of page) and follow the install instructions
`https://github.com/xianyi/OpenBLAS/wiki/Installation-Guide`
using `make`. You will need to run `make` a few times with different arguments but those are provided by the output of the previous `make` command. The `apt-get install` may work, too.

A word of apology: Unfortunately, compiling *est_noise* is some-what a "craps shoot". Depending on the versions of compilers and the versions of your OS, you may need to change the various compiler options to be able to not only successfully compile *est_noise*, but to get it to run, too. In the `compile.sh` script, you will see a number of "commented-out" compile statements; these might provide you some hints about what you might need to change to get the program(s) to work.

3. Initial test

An example is included with the distribution. Critically, it contains the seed file, `seed.dat`, the data `canp.e`, and the input driver file, `est_canp.in`. These are found in the `example` directory. To run,
`../bin/est_noise7.22 < est_canp.in`
On a single core, 64 bit Linux computer running 2.2GHz, the program take 48 seconds when compiled with the Intel Fortran and 68 seconds with gfortran.

Although the program can run over multiple cores, I find that it is most efficient to use a single core. That is, if you are analyzing several time series, one should assign a single process of *est_noise* to run on a single core processing a single time series. That way, if you have 8 core available, one can be simultaneously analyzing 8 different time series. However, one needs to create a directory structure for each time series. The Linux utility, `taskset`, assigns a process to specific cores. For example,
`taskset -c 0 est_noise7.22 < est_canp.in`
assigns the process to the first core. Unfortunately, this utility is not available on Macs.

The input file, `est_canp.in`, contains some annotations about the meaning of each line. You can compare the standard out (aka screen dump) with `est_canp.out`. The most critical information is found at the tail end of the output which includes the estimate of rates, offset, log-trends, sinusoids, and the noise model. More information is contained in the following sections.

4. Input files

At the minimum, *est_noise* requires two input files, one `seed.dat`, which is a single integer number to act as a seed to a random number generator, and a file of data. When the program is run, it will start asking questions on the standard output and responses are required to be typed into the standard input. Your answers are recorded in a journal file which you can re-use after changing its name. This section is broken into two parts, the first part discusses formats, in particular for the data and specifying time, and the second part documents the questions and answers required by *est_noise*

4.1. Data formats.

est_noise handles five different formats for time. Within the program, the basic unit is day. The time formats are specified by:

- `otr` Time is specified as year, and day of year ranging from 1 to 365 (or 366 for leap year). Finer resolution is specified as fraction of a day. For instance, 1515 and 10 seconds on 1, December 2015 is `2015 335.635532407`. These are read and stored as double precision values.
- `otd` Time is specified as year, month, and day of month. Month is a number from 1 to 12. For 1515 and 10 seconds on 1, December 2015, it is specified as `20151201.635532407`
- `otx` Time is specified as year, month, and day of month. Month is a number from 1 to 12. For 1515 and 10 seconds on 1, December 2015, it is specified as `2015 12 1.635532407`
- `gmt` This uses the time format consistent with the GMT plotting package, *www.soest.hawaii.edu/gmt/*. Therefore, 1515 and 10 seconds on 1, December 2015, it is specified as `2015-12-01T15:15:10.0`
- `mjd` Time is specified as modified Julian day; Therefore, 1515 and 10 seconds on 1, December 2015, it is specified as `57357.635532407`. Day zero is 17, November, 1858.

The time series of data requires a time stamp, using one of 5 formats, the observation, followed by its "error". Actually, that last column, although required by the `read` statement, is not used. Some thought is needed with regards to the data-value. Although the program is written for double precision, the Fortran format statements provide limited resolution. For instance, if the actual scale of the variations in the observations are a few millimeters, I suggest that the observations be scaled to millimeters and not meters.

4.2. *est_noise* questions and responses.

The question/answers are generally grouped into two parts, one that specifies the components of the time-dependent functions and the second part that specifies the components of the noise model. The time dependent model is the summation of a rate, changes in rate, offsets, sinusoids,

exponentials, $1 - e^{-t/\tau}$, logarithmic trends, $log(1 + t/\tau)$, and auxiliary functions or data, $a(t)$.

Format of input: Provide the format of dates used to specify the limits of the data and the time dependent model; use one of the five abbreviations listed in *Data formats*.

Number of time series: This will usually be `1`, but in special cases, such as legacy electronic distance meter data where near simultaneous measurements were made to neighboring monuments, this value could be greater than `1`. This option adds additional columns in the design matrix having either 0 or 1 depending upon whether the measurement was made to monument 1 or 2...

Time interval: Specify the time interval to analyze the data. Initial date followed by ending date. The dates must be consistent with the date format descriptor specified in the first answer. It is possible that your data set spans a longer time interval than you wish to analyze. The input provides a means to limit the interval for analysis. In addition, the **start time has a special meaning** as it provides the reference time. For instance, when fitting a secular trend to data, $d_i = B + R(t_i - t_0)$, the nominal value B represents the estimate of d at the **start time**, t_0.

Secular rate, y/n: Estimate the secular rate, *yes or no*. That is, estimate R in $d_i = B + R(t_i - t_0)$. If `n`, then $R = 0$.

Number of rate changes: Normally `0`, but if a rate change is specified, then the number of rate changes is specified. Following this entry, if not equal to `0`, are `n` lines that specify each interval of the rate change, both the start and end date of the change using the same time format entered above. Each rate change interval must be within the specified interval of the data given in **Time interval**. The k^{th} rate change, r_k over the interval t_{k1} to t_{k2} is:

$$\begin{aligned} d_i &= B + R(t_i - t_0) && \text{for } t_i < t_{k1} \\ &= B + R(t_i - t_0) + r_k(t_i - t_{k1}) && \text{for } t_{k1} \le t_i \le t_{k2} \\ &= B + R(t_i - t_0) + r_k(t_{k2} - t_{k1}) && \text{for } t_i > t_{k2} \end{aligned} \tag{3}$$

None of the time intervals for the rate changes should fall outside the limits specified in *Time interval* discussed above.

Number of sinusoids: Input the number of sinusoids. If the number is more than `0`, then list the period of each sinusoid in **days**. The k^{th} sinusoid is the sum of the sine and cosine;

$$d_i = B + R(t_i - t_0) + C_k \cos(2\pi(t_i - t_0)/p_k) + S_k \sin(2\pi(t_i - t_0)/p_k) \tag{4}$$

where p_k is the period of the k^{th} sinusoid with amplitudes C_k and S_k. Likewise, t_0 is the **start time**

Number of offsets: The number of offsets is specified. If not `0`, then the time of each offset is listed. The offset is modeled as a *Heaviside* or unit step with the first value of the step corresponding to the time specified of the offset.

Number of exponential and/or log functions: Many time series have exponential or logarithmic trends. This option allows one to estimate the amplitude, and if desired, the time constant for each of these trends. If the number is not `0`, then for each of the k_{th} trends require:

Time of start of trend: Input the initial date of the trend, t_T, using format entered previously

Time constant and either fix/float: This is two entries, the value of the time constant in **years** and whether you want *est_noise* to estimate the time constant, `float` or hold the time constant `fix`'ed. If `float` then the entered time constant is a guess and the program uses the Nelder and Mead (1965) algorithm to find the optimal time constant assuming the current guess at the values of the data covariance.

Type of trend, exponential or logarithmic: Enter `e` for *exponential*, $Ae^{-(t_i-t_T)/\tau}$, or `m` for *logarithmic* or *Omori's law*, $Alog(1+(t_i-t_T)/\tau)$

Specify the date format of the data: The date format of the data may be different than the format of the dates the specify the time dependence. See above for acceptable formats.

Name of Data file: Provide the name of the data file. See above for acceptable formats.

Specify then number of auxiliary data files: Usually, this will be `0`. However, for some data sets, such as borehole strainmeter data, this will be required as atmospheric pressure data is used to adjust the raw strain data. One can enter other functions which might impact the observations. The time dependent function is set-up as:

$$d_i = B + R(t_i - t_0) + ...A_k(a_{k,i-j}) + ... \tag{5}$$

where A_k is the *gain* or *sensitivity* of the k^{th} function (or data-channel) made up of $a_{k,i-j}$ *i observations* with a lag (in time) of j. If the number of auxiliary data is not `0`, then the following entries are required;

Format of data: Use one of the abbreviations listed under data formats.

Name of Data file: Provide the name of the data file; Again, each record of data must have time, the data value, and value for error, which is not used.

Lead/Lag time: Usually `0` but can take on other values. It must be integer multiples of the data sample interval given units of days.

The program will match the $a_{k,i-j}$ with the corresponding data, d_i. If one of the pair is missing, then that line in the design matrix relating to the observation/data is deleted.

Sampling interval: The nominal sampling interval, in days of the data.

Type of error model: Acceptable values are `n`, `a`, `f`, or `c`. See discussion in the introduction, but usually this will either be `n`, or `a`. With `n`, the components of the noise model are added in quadrature consistent with the legacy *est_noise6.50*. With the other three, filter functions of the noise model are simply added. For data sets with few gaps or missing data, the program runs most efficiently taking advantage of the algorithm discussed by *Langbein* (2017) option `f` should be selected. For gappy data, Cholesky decomposition is employed and option `c` should be selected. By using option `a`, the program automatically selects between the *fast* or *Cholesky* algorithms; if there are more than 25% missing data, the program defaults to Cholesky decomposition to invert the data covariance matrix.

Substitute synthetic noise for data: This will be typically `n` or *no*. If `y`, the program will ask a set of questions that specify the parameters of a noise model. Synthetic data are created with the same sampling interval as the data, then at time when data are *missing*, the synthetic data are deleted. **Note that this option has not been tested with the current version of *est_noise7.2x* but had been used extensively in the past. Use with caution!**

Decimation type: Typically, this will be `0`, but one can put anything between `0` and `3`. With `0`, all measurements are used. Otherwise, the data will be decimated using the following, ad-hoc schedule:

- **1:** Keep two observations, delete the third, keep two, delete the third; uses 2/3 of the available data.
- **2:** Keep two observations, delete the next two, keep two, delete two; uses 1/2 of the available data.
- **3:** Keep two observations, delete the next three, keep two, delete three; uses 2/5 of the available data.

If *Type of error model* is anything but `n`, the *decimation type* defaults to `0`, or no decimation.

White noise amplitude: Set-up the parameters to model the white noise component. Two entries are required with the first being a *guess* at the value of the white noise and the second being `float` or `fix`. If `float`, the program will find the optimal value of white noise that maximizes the log-likelihood. If `fix`, the program holds the value of white noise *fixed*. As an option, one can let *est_noise* make the guess at the initial white noise value. To do this, enter `-99999.0 float`.

Amplitude of first power law noise model: Set-up the amplitude for the first power law/Gauss Markov noise model. Two entries are required with the first being a *guess* at the value of the amplitude of power law and the second being `float` or `fix`. If `float`, the program will find the optimal value of the amplitude of power-law noise that maximizes the log-likelihood. If `fix`, the program holds the amplitude of power-law noise *fixed*. Note that the unit of the amplitude is $unit/yr^{\beta/4}$. As an option, one can let *est_noise* make the guess at the initial amplitude of power-law noise. To do this, enter `-99999.0 float`. The program will complete its guess after the next parameter is entered as the power-law index is required.

Index of first power law noise model: Continuation of the set-up of the first power-law/Gauss Markov noise model. Two entries are required with the first being a *guess* at the of power law index and the second being `float` or `fix`. If `float`, the program will find the optimal value of the index of power-law noise that maximizes the log-likelihood. If `fix`, the program holds the index of power-law noise *fixed*. If the index, $\beta = 1$, then this is considered as flicker noise and if $\beta = 2$, then this is considered as random-walk noise. See equation 1.

Gauss Markov frequency: Continuation of the set-up of the first power-law/Gauss Markov noise model. Two entries are required with the first being a *guess* at the $\alpha/2\pi$ frequency and the second being `float` or `fix`. If `float`, the program will find the optimal value G-M frequency that maximizes the log-likelihood. If `fix`, the program holds the G-M frequency *fixed*. See equation 1. In most cases, your entry will be `0 fix`. Note that the unit of α is radians/year. NOTE: The input is the GM period, $1/\alpha$, in days.

Bandpass filter limits: Pass-band for band-pass filtered noise model in units of cycles/year. Two numbers are required. For typical seasonal noise, use `0.5 2.0`. See equation 2.

Number of poles for bandpass filtered noise: A integer value between `1` and `4` needs to be entered. See equation 2.

Amplitude of bandpass filtered noise: Two entries are required with the first being a *guess* at the value of the amplitude of bandpass filtered noise and the second being `float` or `fix`. If `float`, the program will find the optimal value of the amplitude that maximizes the log-likelihood. If `fix`, the program holds the amplitude *fixed*. If bandpass filtered noise is not considered, enter `0 fix`.

Index of second power law noise model: Set-up of the second power-law noise model. Two entries are required with the first being a *guess* at the of power law index and the second being `float` or `fix`. If `float`, the program will find the optimal value of the index of power-law noise that maximizes

the log-likelihood. If `fix`, the program holds the index of power-law noise *fixed*. If the index, $\beta = 1$, then this is considered as flicker noise and if $\beta = 2$, then this is considered as random-walk noise.

Amplitude of second power law noise model: Continuation of the set-up the second power law noise model. Two entries are required with the first being a *guess* at the value of the amplitude of power law and the second being `float` or `fix`. If `float`, the program will find the optimal value of the amplitude of power-law noise that maximizes the log-likelihood. If `fix`, the program holds the amplitude of power-law noise *fixed*. Note that the unit of the amplitude is $unit/yr^{\beta/4}$. Here, for the second power law entry, *est_noise* does **not** provide an option for the program to make its guess at the power law amplitude.

Add white noise to data: Usually, this will be `0`. In some rare cases where the data are characterized by extreme power-law noise with no detectable white noise, one may need to add white noise to the observations to keep the inversion of the data covariance becoming numerically singular.

Although this is a long list of inputs, the program supplies descriptive queries. The answers supplied to the queries are preserved into a *journal file* call `estin.jrn`. Note; one may need to delete extra lines following the line specifying the name of the data file(s).

As suggested above, *est_noise* can provide a guess for both the white noise and the power-law amplitude. This is useful if one has no idea of the scale of the error in the data. To do this, the program computes two values of "root mean squared"; one that represents the high frequency component and a second one for the complete data set. The high frequency RMS is a measure of the white noise component while the RMS for the entire data set is a measure of the combination of white noise plus colored noise components. The white noise estimate, σ_w, is simply RMS $[d(i+1) - d(i)]/\sqrt{2}$. To compute the colored noise, one first computes RMS of the data residuals after fitting a time-dependent model to the data using least-squares with a diagonal covariance matrix and this is designated as σ_t. The colored noise portion is the difference between the total and the white noise; $\sigma_c = \sqrt{({\sigma_t}^2 - {\sigma_w}^2)}$. From a power spectrum of colored noise, one integrates over frequency to obtain ${\sigma_t}^2 = (1-n)P_c f^{1-n}$ evaluated at the nyquist frequency and the low-frequency limit. Then, using equation 11 of *Langbein* (2004), the power-law amplitude for the specified index of n is estimated: ${\sigma_{pl}}^2 = (2 * f_{ny})^{(1-n/2)} P_c (2\pi)^n/2$, where f_{ny} is the Nyquist frequency. Finally, for the case where the covariance matrix is *additive* rather than *quadrature*, the white noise estimate needs to be reduced by $\sqrt{{\sigma_w}^2 - \sigma_w \sigma_{pl} \delta t^{n/4}}$ where δt is the sampling interval.

5. Output

The program provides several outputs, with the standard out being the most informative. Often, especially when running the program in batch mode, one should direct the standard output to a file. An example is shown in the `./example` directory where *est_noise* was run as

```
../bin/est_noise7.22 < est_canp.in > est_canp.out
```

In addition to the standard output are `resid.out` and `covar.out` which augments the information presented by the standard output. Specifically, each line in `resid.out` is the time, the difference between the observed and predicted observation, the predicted value from the model, and the observation. `covar.out` provides the correlation coefficients between the parameters of the time-dependent model and, secondly, their cross correlations. For *trouble-shooting*, a file, `tauexp.out` is provided showing the estimates of time-constants if the program is required to estimate these terms for either an exponential or logarithmic trend.

The standard output is divided roughly into four sections: Some statistics of the sampling of the data; The estimates of the parameters of the time-dependent model *assuming that the data are temporally independent, Gaussian distributed*; A sequence of trials of noise parameters as the program iterates to an optimal value of log-likelihood; And finally a section listing the estimates of the parameters of the time-dependent model and the accompanying model for noise.

The various statistics compiled from the raw data include the number of points, the statistics of the data sampling, the number of missing data, and an estimate of the white noise component made by taking the differences between adjacent observations.

The program then assumes that the data covariance is a diagonal matrix and carries out the least-squares analysis to estimate the size of each parameter and its standard error. The standard error is rescaled based upon the standard deviation of the fit of the model to the data, assuming a diagonal covariance matrix. (Presumably, these would be the values you would get by plugging the data and the design matrix into spreadsheet).

The bulk of the standard output shows the values of each trial noise model and its corresponding likelihood. If the time-constant of either an exponential or log-function is being estimated, it is shown, too.

Once the program maximizes the likelihood, the estimates of the parameters of time-dependent model and their standard errors are shown (they are recomputed at each iteration of noise modeling). The program attempts to estimate the uncertainty of the parameters of the noise model by examining the curvature of the log-likelihood function during the optimization. I don't believe these results; they look too small.

Along with the value of the logarithm of the likelihood, MLE, both AIC and BIC, Akaike/Bayesian Information Criteron which both factor in the total degrees of freedom from both the time-dependent and the noise modeling.

6. Known problems

The source program `time.f` is written by me a long time ago prior to the internet. I keep updating the code to include the current, and future dates, but I think the current version is limited to time around 2040.

As noted above, the estimates of the standard errors for the parameters of the noise model are probably too small.

The program has been extensively tested and run using the `otr` format. Recently, I did some testing with the `gmt` format. I wouldn't be surprised to find bugs with the other time-formats.

When using the *f* or *fast* option to create and invert the data covariance matrix and using bandpass filtered noise as one component of the data-covariance, *est_noise7.2x* might "crash" or provide non-sensible results. This is usually identified by seeing that there are a lot of NaN for some or all of the trials of covariance parameters in the standard output. On the flip side, *est_noise7.2x* works fine using the *c* or *cholesky* option, but for data sets with few gaps, the calculation is slower. (The legacy option/quadrature *n* works fine, too). I believe that the problem of deconvolving a filter function having a bandpass component is that the filter function is oscillatory with some points that are near zero. The inverse filter function is one that, when convolved with the original filter, yields the *delta* function. The algebra required to solve this problem has division, and given that the original filter functions has terms that are close to zero, their is a potential of obtaining large, oscillatory terms in the inverse function. Additional work, using DFT rather than deconvolution, did not provide satisfactory results. My advise when bandpass filter noised is desired, run *est_noise7.2x* as usual, then, if the *f* option is used and the results of the analysis either doesn't make sense and/or there are NaN and Infinities in screen dump, then re-run with the *c* option.

7. Companion Programs

As mentioned briefly in section 1.3, the *est_noise* package includes five other programs to either generate temporally correlated noise, *gen_noise*, remove outliers, *bust* in conjunction with *adjust*, and evaluate the results, *compare_wander* and *psd_calc*. All of these are fortran programs that the user provides answers from a list of questions from each program. Since *gen_noise* has a number of questions, a *journal* file is created. With the exception of *compare_wander*, the documentation provided above for *est_noise* should provide sufficient information to run each of these programs.

The input to *compare_wander* requires two output files from *est_noise*, `resid.out` and `max.dat`. The `resid.out` lists the misfits to the time-dependent model prescribed for *est_noise*, while `max.dat` provides the estimated parameters of the noise model. The output of *compare_wander* is found in `wander.out` which provides a measure of the drift; this is characterized by computing the RMS of $(x(t+\tau)-x(t))$ as a function of various intervals, τ, for time series (*Agnew* 1992). The quality of the noise model provided by *est_noise* can be assessed through simulating data using the noise model from *est_noise* and computing the drift from each set of simulated data. I recommend that at least 200 simulations. The drift from each simulation is stored, then, for each τ interval, the RMS drift is sorted and then tabulated to establish the median drift along with the 68% and 95% confidence levels. These statistics, along with the drift of the original data should be plotted for a visual assessment of the quality of the modeled noise and time dependence.

8. Sample scripts

In the directory `example0`, I provide three scripts to analyze two different time series. Each script should be sufficiently documented such that they could be reused for other applications. However, to see the results, they do need the GMT plotting library; *www.soest.hawaii.edu/gmt/*.

The three scripts are:

cleanEst.sh: This should be used first as it 1) makes a preliminary estimate of the parameters of the time dependence of the data using a *canned* noise model that is roughly appropriate for continuous GNSS data; 2) detrends those data, 3) removes the outliers, and 4) reinserts the trends. The residuals to the fit are plotted so that one can add additional time-dependent models as required or remove outliers not caught by *bust_5*. In addition, the input to successive calls to *est_noise* is set-up, which are required in the next two scripts.

EstNoise.sh: Uses the cleaned data, `data.cl` and the file `est0.in` from *cleanEst.sh* along with a user selected noise model and runs *est_noise* to find the optimal parameters of the noise model. The residuals to this fit are plotted. In addition, the output of *est_noise* is feed into *compare_wander* and the RMS drift of the data, along with the statistics from a set from simulations are plotted. Finally, the equivalent power spectral density is plotted.

EstNoiseAll.sh: Instead of evaluating a single type of noise model, this script evaluates a sequence of noise models as a batch process such that the best noise model can be identified after evaluating the changes in Log(likelihood) and the corresponding drift. Initially, both the random-walk plus white noise and the flicker plus white noise models are computed. The script picks the

one with the *maximum likelihood* and identifies it as the so called *null* model. Next, the power law model is evaluated and its likelihood is compared with that from the *null* model. Similarly, the flicker plus random-walk noise is evaluated. Of these four, it is likely that one of these is a clear "winner". However, the script goes on to evaluate both First order Gauss-Markov and Generalized G-M noise. A final set of evaluations uses bandpass filtered noise superimposed on either flicker or random-walk noise, and flicker plus random walk noise (the default) or power-law noise with the `-P` option. Using the `-S` option, one can limit the number of noise models tested from the original eight. For all of these, a drift curve is evaluated.

These scripts currently set for the `otr` format and mostly set-up for the `gmt` format.

Prior to running each of these scripts, you will need to edit each script to identify where the executable programs reside. This line should be located near the top of the file containing the script with an environment variable `progs`. The actual work by these scripts is carried out in `/tmp/SCRATCH`.

By typing the command, one should get documentation with regards to the inputs to these files. The `example0` directory contains the scripts, two data sets, files that are required for the inputs, and the output plot files.

8.1. **Example 1.** The data are collected from a site with observations spanning both the San Simeon and Parkfield Earthquakes (2003 and 2004 respectively). The time dependent model includes sinusoids, a secular rate, offsets due to both earthquakes, `canp.off`, two Omori law, decay functions and a rate change, `canp.trd`, and a few observations deleted just after the Parkfield earthquake, `canp.edit`. The initial "cleaning" is obtained by:

```
cleanEst.sh -d canp.e -f otr -O canp.off -T canp.trd -E canp.edit
```

Because the time constant for the Omori function for the Parkfield earthquake is being estimated, *est_noise* runs slower than normal. The results can be found in `canp.e.ps` and `est_canp.e.out`. To speed-up the next set of evaluations, I have modified the `.trd` file by **fixing** the Omori time constant for the Parkfield Earthquake to 0.0061746 years which is shown in `canp1.trd`. Consequently, I re-run *cleanEst.sh*

```
cleanEst.sh -d canp.e -f otr -O canp.off -T canp1.trd -E canp.edit
```

To evaluate a specific noise model, I run *EstNoise.sh* as:

```
EstNoise.sh -d canp.e -M PL
```

The outputs of this analysis are shown in `canp.e_PL.ps` and `est_canp.e_PL.out`. The plot shows the residuals to a fit to a time dependent model, the drift of the data compared to simulated data having the same model of noise, and the equivalent

power spectra of the noise model. The drift suggests that observed data has slightly more noise over the 300 to 1000 day periods that the noise provided by a power-law noise model. Consequently, I could try a combination of flicker and random walk.

```
EstNoise.sh -d canp.e -M FLRW
```

Indeed, examination of the drift, `canp.e_FLRW.ps` suggests an improvement in the fit. In addition, the increase in Log likelihood, from -7667.7 to -7660.9 from the PL to the FLRW model suggests that the flicker and random-walk model of noise is more appropriate for this data set. Likewise, other noise models can be evaluated.

However, a more comprehensive search for the optimal noise model is carried out with *EstNoiseAll.sh*. For this example, it is invoked with:

```
EstNoiseAll.sh -d canp.e
```

Examination of the results of the drift spectra along with the values of Log(likelihood) (MLE), AIC, and BIC shown in `canp.e_all.ps` suggests that the FLRW noise model is best since it has the largest MLE and the smallest AIC and BIC.

8.2. **Example 2.** This is an example of simulated noise having first order Gauss Markov noise. In addition, there is an offset, and the data use the *gmt* format. Initially, the data are "cleaned";

```
cleanEst.sh -d noise2.gmt -f gmt -E noise.edit -O noise.off
```

The results are presented in `noise2.gmt.ps` and `est_noise2.gmt.out`.

Noise modeling is accomplished by:

```
EstNoise.sh -d noise2.gmt -M RW
```

Although other noise models can be evaluated, PL, FOGM, and GM, none seem to be much better than the random walk model.

Likewise, a search of optimal noise models is provided by invoking:

```
EstNoiseAll.sh -d noise2.gmt
```

Examination of the results, `noise2.gmt_all.ps` suggests that RW is probably the best noise model since its closest competitor, FOGM, has marginally larger MLE (1.39) and lower AIC (-0.8) relative to the null model of RW. On the other hand, BIC for FOGM is worse than for RW. The most conservative choice would be to chose the model having random-walk represent the data at its longest periods which provides larger standard error in rate relative to the FOGM model. NOTE: See may comments about AIC and BIC, as well as a discussion of FOGM noise in Threshold_dMLE_v3.pdf.

9. Updates

The following incremental changes have been made; December 28, 2015

7.21: This is the first, fully working version.

7.22: The sizes of some of the common block variables have been reduced along with elimination of some code, and its dimensioned arrays. This is also a working version, but the size is still too big for static linking on Linux

More recent updates are included in a file called `BugFix.txt`.

10. References

Agnew, D. (1992), The time domain behavior of power law noises, *Geophys. Res. Lett.*, doi:10.1029/91GL02832.

Bos, M. S., R. M. S. Fernandes, S. P. D. Williams, and L. Bastos (2012), Fast error analysis of continuous GPS observations, *J. Geod.*, doi:10.1007/s00190-007-0165-x.

Langbein, J., and H. Johnson (1997), Correlated error in geodetic time series: Implications for time-dependent deformation, *J. Geophy. Res.*, *102*, 591–604.

Langbein, J. (2004), Noise in two-color electronic distance meter measurements revisited, *J. Geophy. Res.* doi:10.1029/ 2003JB002819.

Langbein, J. [2008], Noise in GPS displacement measurements from Southern California and Southern Nevada, *J. Geophy. Res.* doi10.1029/2007JB005247.

Langbein, J. [2009], Computer algorithm for analyzing and processing borehole strainmeter data, *Comput. Geosci.*, 36(5), 61119, doi:10.1016/j.cageo.2009.08.011

Langbein, J. [2012], Estimating rate uncertainty with maximum likelihood: differences between power-law and flickerandom-walk models, *J. Geod.* doi:10.1007/s00190-012-0556-5

Langbein, J. [2017], Improved method for maximum likelihood analysis of time series with temporally correlated errors, *J. Geod.91* , 985?994 (2017) doi:10.1007/s00190-017-1002-5.

Nelder, J. A., and R. Mead (1965), A simplex method for function minimization, *Computer Journal 7:* 308?313. doi:10.1093/comjnl/7.4.308

Press,, W.H, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, (1992) *Numerical Recipes in Fortran 77*, second edition, Cambridge University Press.

Williams, S. D. P., Y. Bock, P. Fang, P. Jamason, R. M. Nikolaidis, L. Prawirodirdjo, M. Miller, and D. J. Johnson (2004), Error analysis of continuous GPS position time series, *J. Geophy. Res.* doi:10.1029/2003JB002741.