`enoise` Data Simulator User Manual

Date: 13-March-2014
Author: Zheng Meyer-Zhao
Simulator design: C. Y. Kuo

Introduction

`enoise` (Enhanced-`anoise`) is developed based on the `anoise` simulator made by Geoff Crew. It enhances `anoise` by simulating non-zero baseline data. The goal here is to simulate ALMA and Mark5b-like data in VDIF format and correlate them with each other.

The `enoise` software consists of the following components:

1. random, which generates random numbers used as common signals, delay signals, and station noise.
2. input, which contains example input files that will be used by the simulator. Users should define the input files for the simulation.
3. genv2dvex.sh, a script that uses user defined input information to generate .v2d file, vex file, and .calc file to be used by DiFX.
4. usemodel.py and difxmodel.py scripts, which are provided by Adam Deller to calculate the delay and delay rate information.
5. `enoise`, which uses the produced information from other components to generate non-zero baseline simulation data.

This document provides the general information of each component in the software package and its corresponding usage. A complete example is shown in the Appendix.

Prerequisite

DiFX-2.3.0, GNU Scientific Library (gsl, gsl-devel) and FFTW3 (fftw, fftw-devel).

`enoise` Components

random

The design concept of the current release (v1.0) of `enoise` is to reuse the generated data, as it is computationally expensive to generate high quality random numbers. Using this design, users only need to generate random numbers once and reuse them for each experiment. The disadvantage of this design is that users need to have enough disk space available (i.e. 500GB ~ 1TB, depends on the amount of data to generate). A `Makefile` is available in the directory, to compile the code, simply use the `make` command. The file names are pre-defined, i.e. common.dat for common signals shared by all antennas, delay.dat for random delay numbers, noise0.dat and noise1.dat … for station noise, the number of noise file depends on the number of stations in the simulation. An example script gennoise.sh is also included in the directory to show how to generate different random number files.

```
Usage: genran [options]
where the options are:

   -s <float>    seed to use
   -h <int>      quality of random number 1 - 5,
                 where 5 is the best.
                 1 corresponds to gsl_rng_rand,
                 2 corresponds to gsl_rng_taus2,
                 3 corresponds to gsl_rng_mt19937,
                 4 corresponds to gsl_rng_ranlux, and
                 5 corresponds to gsl_rng_ranlux389
   -n <string>   file name and duration, e.g. common.dat:4.4
```

input

This directory provides example files of user input. `station.cat` and `target.cat` contain information regarding antenna station and observation target respectively. Users can add more stations or targets to these files. `anoise.inp` is a template of user input information with respect to the experiment, i.e. this file allows a user to select a target source, antennas to use, and observing start time and end time. `anoise.inp.withalma` shows how to include alma in the experiment, however, in order to use this input file, one needs the DiFX zoomband enhancement.

genv2dvex.sh

This script uses information given in the input directory to generate `.v2d` and `vex` file. It also calls commands `vex2difx` and `calcif2` to generate files required by correlation. All output files are stored in a directory generated by the script. NOTE that, in order include ALMA in the simulation and generate .v2d and vex file automatically, one also needs the zoomband enhancement of DiFX which is not yet available in the DiFX repository. Therefore, an example directory with ALMA related simulation input files is also provided to demonstrate how enoise can be used to generate ALMA data and correlate it with normal VLBI stations.

```
Usage: genv2dvex.sh tdir obs_info
```

usemodel.py and difxmodel.py

These two scripts calculates the delay and delay rate information of each antenna using information provided in the .calc file. It creates a directory drate to store the information it generates.

```
Usage: usemodel.py calc_file sec+1
```

```
where sec is the observation time in seconds
```

enoise

The enoise program generates the actual experiment data. Users can specify the random data directory and the `drate` directory (delay and delay rate information generated by `usemodel.py`), station information etc.. NOTE that, the station names given to enoise have to be in alphabetic order, this is because of the delay and delay rate files corresponding to each station are generated in this way.

```
Usage: enoise [options] stn1 [stn2 ...]
where the options are:
  -v            verbose, may be repeated for more
  -b <int>      samples used for bit statistics (1024^2)
  -d <float>    duration of observation (2.0)
  -n <string>   make adjustments to common noise;
                use "help" for details
  -r <float>    report processing interval (usecs)
  -t <float>    2-bit threshold in sigma units (1.00)
The -r flag produces a progress report for the "duration".
The -t option is not implemented.
Use "help" as a station name for station configuration options.
A sample invocation giving a (very) short sample with 2 channels
of ALMA data (station AL) and 4 channels of typical VLBI data
(station SP) is:
enoisez -v -d 0.00064 -n corr:0.05 \
-n tone:5,0.01 -n tone:40,0.01 -n tone:75,0.01 -n tone:110,0.01 \
-n pathd:/path/to/ndata -n pathr:/path/to/drate \
AL:AL.vdif:alma62.5x2,1.5:24@6415080 \
SP:SP.vdif:vlbi32.0x4:24@6415080
If -d is 0.0, the first packet is generated, but no
data is created; which allows you to check what you will eventually
get without waiting a long time for it....
```

To see the parameters available of option `-n`, one can use command:

```
$ enoise -n help
  corr:<float>        amplitude of common signal relative
```

```
                     to the station receiver noise (0.01)
  tone:<freq,amp>      adds a tone at the <float> freq,amp
  tone:<freq,amp,phs>  specifies phase as well
  fftn:<int>           force underlying fft to have this many
                       samples/us (0 -- means work it out)
  slices:<int>         number of 1-us sample groups per fft (1)
  pathd:<string>       path to simulation data, maximum 50 characters
  pathr:<string>       path to delay and rate info,
                       maximum 50 characters
  skyfreq:<float>      sky frequency in MHz
  limit:<float>        threshold to calculate delay error
  spsmul:<int>         multiplier of sps (samples per microsecond)
  stnoise:<float>      ratio of station noise [0..1]
```

For station specification and possible frequency arrangement, one can use command:

```
$ enoise help
The station specification is a string of the form:
  ID:file[:type[choff][:time]]
where the type specifies the VDIF packet format for 2-letter
station ID, written to file with one of these types:
 vlbi512      one single-channel of 512 MHz
  alma500      one single-channel of 500 MHz
  trad8.0xN    N (256) channels of 8.0 MHz
  vlbi32.0xN   N (64) channels of 32.0 MHz
  vlbi64.0xN   N (32) channels of 64.0 MHz
  alma62.5xN   N (32) channels of 62.5 MHz
  sma32.0xN    N (64) channels of 32.0 MHz
  carma32.0xN  N (64) channels of 32.0 MHz
  smto32.0xN   N (64) channels of 32.0 MHz
  iram3032.0xN N (64) channels of 32.0 MHz
where in the multi-channel cases the number of channels
needed for ~ 2 GHz sampling is indicated in parentheses.
The number of channels is required to be a power of 2 and
```

```
at most 256; and in all cases, 2-bit sampling is assumed.
 A channel specification may follow.  It can be either or both of
    +C or -C    to shift all channels in FFT space
    ,G          to insert a gap of G spectral points
where the units of C and G are MHz (most likely)
In the -C case, the width of the zeroth channel will be reduced
so that only one side band is used.  If both are given, the shift
must come first.  Both are floating point quantities, but rounding
in FFT space may not give you exactly what you want.
Finally, the (starting) time specification is epoch@seconds.
```

Appendix

An example is shown below to go through each component described in the previous section:

```
$ export $ENOISE_HOME=/path/to/enoise
# generate random numbers
# this step only needs to be done once
# e.g. 4.4 seconds data will be generated in common.dat with 8000
numbers for each microsecond
$ cd enoise_working_directory
$ mkdir ndata
$ cd ndata
$ genran -s 58325 -h 5 -n common.dat:2.0 -n delay.dat:0.5 -n
noise0.dat:2.0 -n noise1.dat:2.0 -n noise2.dat:2.0
# change directory back to the working directory
$ cd ..

# copy the input directory to the working directory, e.g. the same
directory where ndata is.
$ cp -r $ENOISE_HOME/input .
# modify input/anoise.inp
SOURCE=M87
STATION="CARMA, SMA, SMTO"
BAND="32.0x4, 32.0x4, 32.0x4"
GAP="0.0, 0.0, 0.0"
```

```
SHIFT="0.0, 0.0, 0.0"
START_TIME=2012y075d05h58m00s
END_TIME=2012y075d05h58m04s
FREQ_OBS=230000.0

# run genv2dvex.sh and usemodel.py
# genv2dvex.sh will generate a directory g3-7000
$ $ENOISE_HOME/genv2dvex.sh g3 input/anoise.inp
# we would like to do a 4 seconds simulation here, therefore the
parameter for usemodel.py is 4+1=5
$ $ENOISE_HOME/usemodel.py g3-7000/g3_7000.calc 5

# go to the experiment directory to generate simulated data
$ cd g3-7000
$ $ENOISE_HOME/enoise/enoise -v -d 4.0 -n corr:0.05 \
-n pathd:/path/to/ndata -n pathr:/path/to/drate \
CM:CM.vdif:vlbi32.0x4:24@6415080 \
SP:SP.vdif:vlbi32.0x4:24@6415080 \
ST:ST.vdif:vlbi32.0x4:24@6415080
```

Now we have everything we need to start correlation with DiFX.