Gem Data Pre-Processing Workflow

Jake Anderson

September 9, 2020

1 Installing the gemlog 0.3.2 software

You must have anaconda or miniconda installed to proceed. If you don't have either and don't know which to use, install miniconda from here: https://docs.conda.io/en/latest/miniconda.html

Then, follow the gemlog installation procedure at https://github.com/ajakef/gemlog/blob/master/README. Note that the resulting conda environment includes relevant packages like obspy and pandas.

2 Getting Started

First, download the zip file containing the inputs for this demonstration: https://github.com/ajakef/gemlog/blob/master/dem
You can use either wget, or follow the link and click the "Download" button. Move the file to some convenient folder, unzip it, and cd into the project folder.

Notice the structure of this project folder. It includes a "raw" folder where the gem data files go. You don't *have* to use the folder name "raw"–or even have a separate folder for the raw data files–but all the default settings assume you're doing it this way.

```
--Project_Folder/
  |__station_info.txt (table of SEED codes vs Gem serial number)
  |__raw/ (raw data from all loggers, FROM THIS PROJECT ONLY)
```

This example includes six two-hour files from six data loggers.

```
$ ls raw
FILE0010.096  FILE0010.103  FILE0010.109  FILE0011.088  FILE0011.106  FILE0012.077
```

Gem data files have the Gem's serial number as the extension, meaning that we have data from Gems 077, 088, 096, 103, 106, and 109. Empty or nearly-empty data files (like those from quick tests between field campaigns) can cause problems in the data conversion. It's a good idea to start each field campaign with clean disks, and to inspect your raw files and remove bad files before converting them.

The file "station_info.txt" is used to (optionally) assign SEED codes[1] (network, station, and location) to your output data. By default, your converted data will use the Gem serial number as the station name, and leave the network and location codes blank. If you want to assign different names, you need to include a comma-separated text file containing a table of serial numbers, network codes, station codes, and location codes. Because all Gem recordings are infrasound sampled at 100 Hz, they are automatically assigned the channel code HDF[2].

3 Converting raw gem data

Set your conda environment to whatever you set up during the installation using a terminal command like

```
conda activate gem
```

[1] Info on SEED and SEED codes at https://ds.iris.edu/ds/nodes/dmc/data/formats/seed/
[2] SEED channel naming convention: https://ds.iris.edu/ds/nodes/dmc/data/formats/seed-channel-naming/

and run the data conversion using the terminal command

```
gem2ms
```

This may take a while to run if you have a lot of data! gem2ms will add the following folders to your directory structure:

```
--Project_Folder/
  |__station_info.txt (unchanged, and unused in this step)
  |__raw/ (unchanged)
  |__mseed/ (NEW: contains hour-long miniSEED waveform files)
  |__metadata/ (NEW: contains csv files with state-of-health data)
  |__gps/ (NEW: contains gps logs)
  |__gem2ms_logfile.txt (NEW: contains a record of the conversion process and any messages)
```

By now, all the useful information has been extracted from the raw files; there is no reason to work with them further except for possible debugging. For more info on the gem2ms options and capabilities (e.g., the ability to write SAC files and the text format TSPAIR), run

```
gem2ms -h
```

3.1 Multiple conversion attempts

If the conversion is run multiple times, gem2ms will overwrite pre-existing miniSEED files. However, gps and metadata files are tagged with a conversion number at the end of their file name, so they are not overwritten. For a given Gem serial number, the file with the highest conversion number was created most recently. gem2ms_logfile.txt is appended to on each conversion attempt, so it is also not overwritten.

4 Organizing the data

If your dataset is from an array or network, you probably want to create a station map and assign network, station, and location codes to your miniSEED files. You can do this using python functions from gemlog. Before starting python, you need a csv file assigning Gem serial numbers to network, station, and location codes. In this example, 'station_info.txt' describes a network (NM) containing two stations (LADR and MANZ), each containing three locations; each of the six Gem serial numbers is assigned to a network, station, and location. For more information on SEED codes, see the IRIS page https://ds.iris.edu/ds/nodes/dmc/data/formats/seed/.

```
$ cat station_info.txt
077,NM,LADR,00
088,NM,LADR,01
103,NM,LADR,02
096,NM,MANZ,00
109,NM,MANZ,01
106,NM,MANZ,02
```

Begin by activating your conda environment as before:

```
conda activate gem
```

and open python within your project directory. Run the following code:

```
import gemlog
coords = gemlog.SummarizeAllGPS('gps', output_file = 'project_coords.csv',
       station_info = 'station_info.txt')
gemlog.rename_files('mseed/*', station_info = 'station_info.txt', output_dir = 'renamed_mseed')
```

The first command creates a new csv file containing the following columns:

- SN: Serial number
- lat: Latitude (degrees)
- lon: Longitude (degrees)
- lat_SE: Standard Error of latitude (degrees)
- lon_SE: Standard Error of longitude (degrees)
- t1: Time of first GPS sample (UTC)
- t2: Time of last GPS sample (UTC)
- num_samples: Number of GPS fixes logged
- network: Network code
- station: Station code
- location: Location code

It saves the same information to variable 'coords' as a pandas DataFrame. If the stationFile input (which assigns network, station, and location codes to each serial number) is not provided, the last three columns will be omitted.

The second command reads each miniSEED file from the 'mseed/' folder, assigns the corresponding network, station, and location codes, and writes it to the new folder 'renamed_mseed/'.

You can now create a basic station map using the following code. Note that the GPS coordinate averages may not be accurate enough for array processing if only a short period of data is used (less than a day or two).

```
import matplotlib.pyplot as plt
plt.plot(coords.lon, coords.lat, 'k.') # plot the lon and lat as black dots
```

Finally, you can create an obspy "Inventory" object and write it as a stationXML file using this code:

```
import obspy
from obspy.clients.nrl import NRL

## download the instrument response from the IRIS Nominal Response Library
nrl = NRL()
response = nrl.get_response(sensor_keys = ['Gem', 'Gem Infrasound Sensor v1.0'], datalogger_keys = ['Ge

## create an inventory of all sensors used in this project
inv = gemlog.make_gem_inventory('station_info.txt', coords, response)
inv.write('NM_inventory.xml', format='STATIONXML')
```

4.1 Inspect the Data

The following workflow can be used to read the renamed mseed data and deconvolve the instrument response. Because wind noise is severe at lower frequencies, it is generally necessary to apply a high-pass filter to obtain good data; 1 Hz is a good corner frequency to start with. Obspy's plot functions do not handle long plotting periods well; a program like PASSCAL's PQL is probably better for perusing the data.

```
import obspy

## read the data
data = obspy.read('renamed_mseed/*')
print(data)
```

```
## combine traces so that each station has one trace
data.merge()
print(data)

## deconvolve the instrument responses using the inventory already created
inv = obspy.read_inventory('NM_inventory.xml')
data.remove_response(inv)

## filter data above 1 Hz (lower frequencies are often wind noise)
data.filter("highpass", freq=1.0)

## trim the data around a known event
t1 = obspy.UTCDateTime('2020-05-10T12:14:00')
t2 = obspy.UTCDateTime('2020-05-10T12:15:00')
data.trim(t1, t2)

## plot the results
data.plot()
```