# Ref step code

Basic description of the code:

* graphframetest.py – the main program
* noname.py – the GUI as generated with wxFormBuilder (note version 3.5, usually incompatible with other versions)
* analysis.py – the analysis object
* graph\_data.py – a graph that presents measurement results
* gpib\_data.py – the main thread of IEEE488 actions, reads through table and controls the instruments according to the table. Instrument objects return strings that get sent to the try\_command function.
* gpib\_inst.py – an instrument class for all instruments, returns strings that need to be evaluated/executed by the gpib\_data thread.
* stuff.py – methods and classes for threads launched by the GUI
* pywxgrideditmixin.py – an enhancement for wxGrid that facilitates cut/paste by region
* tables.py looks after putting Excel data into wxGrids
* GridMaker.py is a class for printers to grid, it checks the grid size prior to printing and updates it if necessary. Allows for printing of a whole column or row.

Initially, a user is expected to upload a dictionary file to the program. This file will be used for communicating with the instruments. Here a data table can be uploaded too, containing the information for the IEE thread to step through. This file is not necessary; it can be generated by the program using the “generate table” button. This operation is handled within the main thread of the program, and relies on the ranges of the instruments in order to generate an appropriate ref-step table. The table can be used to calibrate either the ranges of the DVM or of Source X. These ranges can be untrue to the ranges of the instrument, as long as each specified range is enveloped by a real instrument range (or else overload and out of range errors will occur).

The run button will start the worker thread reading through the table with an optional safety check routine.

## The worker thread

The current worker thread does safety checks (are instruments wired correctly and are they responding) then goes into reading the table for instructions. Each instruction is sent to the instrument class, that returns a string to be evaluated by the ‘try\_command’ function. The try command function will either ‘eval’ or ‘exec’ the string depending on if it needs to return or not. It first however checks if the thread wants to abort, if it does it will not send commands to the instruments anymore, so that the thread basically skips through the current commands until it terminates. In the try\_command function, if the thread wants to abort it first does a check to see if the instruments have been made safe, if they have not, the MakeSafe function is called. This function bypasses the try\_command, and sends the safety routines to all the instruments directly. If it fails to send the instructions, the program deems the situation unsafe. The make safe button turns red and it prints which instruments the communication failed with.

Also in the thread, there are several points in which it needs to wait, those are made into loops that continuously check if the thread wants to abort until the wait time is up. This is so that the thread can be aborted without waiting for it to finish its pause.

The thread saves a log file of all commands as a text document, and an excel file containing a copy of the control table with each individual data point from each measurement printed to the right.

## Data analysis

“Analyse data” is executed from the main thread. It creates an analysis.Analyser object that requires the name of the raw data file to be analysed.

The analysis object reads the table and saves the entire thing as an array. It then attempts the analysis for both the DVM and Source X.

* Reading from the start, it identifies the first mid point and finish points of each sequence, by looking for three repeated settings on source X.
* The midpoint and start point are used to infer the end point, so the ascent and decent must be symmetric.
* For start and end points, all readings between are turned into GTC objects using the number of readings, standard deviation and mean.
* Such a list of GTC object is sent to be split up into data applicable to the meter ratios, and data for source X (of the first four data points, each instrument has one which they do not use in the ratio formula).
* Computes linearity and gain ratios for the ascending and descending sections for both the meter and the source X, using the same data, in separate functions.
* These ratios are printed to the results sheet in the same workbook
* The row below that which was previously the last row becomes the new start row for the next set of measures and the process repeats.

It computes the gain ratio and linearity ratios using the the GTC. It is possible to move the entire computation to a separate thread, instead of an object within the main thread. However the computations are quite fast so no noticeable freeze time is induced.

## Generate table

A table can be generated based on the ranges of the instruments, this is done using GridMaker.py. The class has two functions, one for basing the columns on the meter’s ranges and one for the source. They are almost identical, but require slightly different treatments if ranges are exceeded by other instruments. It initially computes the ref step, as a tenth of the difference between the top of a particular range and the bottom of the range. In most cases the bottom of the range is zero. Since the ref-step is common on both meter and source X ranges, if one of these two instruments does not reach zero on some range then the other instrument will either not reach zero, or not reach the top of its range.

If the table is generated according to the ranges of say the meter, the analysis of the data of source X will still be meaningful (gain ratios etc) it’s just that they won’t cover the entire ranges of the instrument.

## Things to possibly add or change:

* Prior to generating a table, check the instruments’ ranges and re-order if necessary. Now users are required to provide a list of ranges in ascending order.
* Temperature and relative humidity can be recorded, they could be given as a key word argument to the instrument class and if no query exists for them, the thread skips the command.
* Instrument ranges to be just the true ranges of the instruments. Include a third table that will have calibration info. This is: min voltage, max voltage, number of readings, delay before reading, delay between readings, number of repetitions. Then final ratios can be printed to this table.
* Let users repeat ascending/descending blocks, this is handled if the above is done.
* Where do ratios go? This is handled if point 3 is done. Right now they simply go into a new sheet in the raw file

## Bugs (that I know of)

When pressing load table, and exiting without selecting a table, python wants a file to be selected and gives an error.

Plot stops after many data points for some reason.

After a reset, the plot does not erase previous data.