Auger batch conversion and quantitative analyses
OVERVIEW/README

This set of python scripts/function can be used for:

- Batch import conversion of .sem, .spe and .map Auger files generated by PHI 700 Auger Nanoprobe (and related PHI instruments).
- Automatic combining via averaging of multiple Auger files on same samples (multiple spe files from same region generated using the Autotool loops with the PHI smartsoft control software)
- Batch smooth-differentiation of element peaks and subsequent quantitative analysis (similar functionality to PHI Multipak software but can be run in batch mode on all files in given directory)
- Background fitting, peak subtraction and integration of direct counts signal (alternate quantization method that doesn't rely on smoothing-differentiation)
- Plotting of individual spe files or creation of PDF reports containing all spectra in directory
- Source data along with background fits, peak fits, etc. are stored separately as CSV file and are thus accessible to any other plotting software
- Auger spectral imaging (using PHI multiplex mode and Autotool)

No GUI yet so for interactive plotting just open the csv files created in Originlab, Sigmaplot, Igor or whatever.… the key advantage of these scripts are:

-Batch quantitative analysis on hundreds of peaks on elements you chose (highly tweakable using AESquantparams)
-Ability to filter data with pandas to select the subsets of spectra you want,
-Batch plotting to PDF to quickly look at derivative plots or direct plots for hundreds of files (crucial to ensure that your batch quantification doesn't have a problem),
-Quick creation of sets of compositions done in consistent manner and recreation of these compositions with different k-factors/element subsets

These scripts are currently run in a python IDE (such as spyder) which allows run entire scripts or cell subsections in the editor plus giving access to Ipython console and variable explorer.
Currently if you want interactive plotting, you can just open the csv files in
Data processing is usually done sequentially (run import_main then quant_main then plot_main), However, quantization and subsequent plotting can be resumed later or redone with different parameters (just need to reload AugerParamLog and a few other log files that contain file names/paths and associated parameters).
The import, quant, plotting and utility functions are in separate files and these modules are commonly loaded via import from either import_main or quant_main. Quick descriptions of all the functions used are available in the Excel file. Within the main py files, there are lots of ways with pandas to select subsets of data (i.e. some keyword in comments column, Mg-rich composition) and then access only that subset of files for batch plotting, computing compositions, making scatter plots, etc.

Compositions can be derived based either on Savitsky-Golay smooth-differential method or direct integration… probably easiest to run both and look at how they compare.

LOG FILE SETUP

For the raw data file, most parameters are automatically stored in the text header of the sem, spe or map files. However the names of sample regions generally are not (one can use the Smartsoft labbook but an Excel file log is better). During your data collection session, it is best to create a logfile similar to the excel file example with "Auger_logbook" somewhere in the name to be found by import_main. If you didn't create a log you can skip step 1 and later manually enter sample info directly into an augerparamlog.csv file in sample or comments columns… just save them, open in Excel or some other editor.

Step 1: Create data log file in correct format

As mentioned you can skip this and do manually later, but it's probably best practice to correlated data filenames/filenumbers with the sample first. If you create an Auger_logbook.xls file, the python batch processing software will pull the first 5 fields from the data log file (manually created by user during session to keep track of sample name and associated files). Other fields sometimes in the log file are redundant (X, Y, Mag, etc. are pulled from file headers which are more accurate).

Easiest naming convention to avoid errors: Name your project something like "Si_wafer_11Jan16", set first file to 100 and let PHI autotool autonumber all the files after that (i.e. Si_wafer_11Jan16.100.sem, Si_wafer_11Jan16.101.spe,etc.). During data acquisition keep an excel logbook (see template) that tells exactly what was measured in 101.spe. Also directly before every .spe ,take a .sem file so you can make an image of which spatial areas were measured within the SEM image.

Auger logbook file containing:

1) Session/project name
2) filenumber – this is auto-incremented number assigned by PHI
3) lastnumber - if you are collecting multiple .spe files with loop tool, this has last file number of the subset of spe files that you want to combine via averaging… if you use different naming conventions
4) Sample name (only have single name even if collecting data from multiple spatial areas)
5) Comments
6) current (nA) *this can be inferred from GunVoltage but is not explicitly in header*

Brief description of combine/averaging: Our standard practice is to collect longer acquisitions as a series of sequential spe files (using Autotool to take before image, collect ~5 or more identical spe files and then collect an after image). Standard data processing is to just combine the 5 identical spe files via averaging. However the sequential sub-files are also available for analysis (in case of beam drift during acquisition or sample transformation/modification during beam exposure).

Step 2. IMPORT MAIN

Now you are ready to open import_main and then sequentially run the following cells (separated by #%%):

1. load modules (Auger, AESsmquant, AESintegquant, AESutils)
2. Set data directory (can use tinker but copy/paste cd in console is quicker) and use glob to find all the raw files you want
3. Check log file/ create directories etc. (optional)
4. Run main file import processing loop Augerbatchimport (can be somewhat slow with hundreds of files)… calls a ton of different subfunctions which you can peruse in the modules; note that the scripts will never alter your raw data file, although optionally they are moved to /sub
5. Create images with spatial area annotations on them (optional)… these are autosaved as e.g. 101_areas.jpg, but it assumes that .sem image was taken just before .spe file; there are also file-by-file functions to do this (i.e. annotateone)
6. Functions for import and data reorganization of Auger spectral images (done using multiplex mode and Autotool with user defined spatial array in somewhat hacked together method; see section below)
7. Run the combine via averaging loop (if applicable); if you are not using sequential file numbering + Excel log to tell which files to combine (fastest batch method), there are other ways to choose sub-spe files and combine-average them (autocombinespe, combinespelist)… combined files are typically autonamed i.e. 101103 is combine-averaging of spectra 101, 102 and 103.

Notes on the above sections in import_main

Load modules:

1) Pandas is required and is extensively used
2) To load functions (e.g. import Auger_batch_import_functions as Auger) the folder containing these scripts must be added to PYTHONPATH or you can use the sys.path.append statement with the correct subfolder

Set data directory:

All files in a single directory are processed, so set the data directory using os.chdir, tinker, cd in iPython console or whatever. Currently the glob statement does not search through sub-directories. After processing the raw source files (.spe, .sem, .map) are moved unmodified into the sub-directory named “sub” (unless movefiles set to False)

Check log file:

The function checklogfile will compare the actual existing files in the directory with the listings in the Excel log file. If there are discrepancies, they are output as print statements to the Ipython console. Possible discrepancies are: 1) data file mentioned in Excel log but not actually present in directory 2) data file present in directory but not mentioned in Excel log and 3) attempts to average/combine files that are not spectra (numbering error in Excel log wherein you’re trying to average an image file with a spectrum).

Run main file processing loop:

AugerParamLog = Auger.Augerbatchimport(filelist, Augerlogbook)

For every file in filelist, this looping function:

1) Converts each spe file to csv and computes S7D7 derivative (see PHI Multipak appendix for exact formula) and autosaves to csv
2) Converts sem files to jpg (can choose other formats with minor code change in img.convert)
3) Converts map files to separate jpg intensity images for each element
4) Creates AugerParamlog with many parameters associated with each file

If script is rerun and csv or jpg files already exist, they are not recreated but the AugerParamLog will be recreated. If for some reason you want to recreate them, just delete the .csv and rerun on whichever .spe file you want.

The AugerParamLog contains many parameters extracted from each file's text header: including filenumber, path, name, field of view, type (multiplex or survey), # spatial areas, # of acquisition cycles, timestep, stage X, Y, Z and X & Y image shifts and various calculated quantities, evbreaks (position of spectral breaks as index #s in multiples scans), (total acquisition time in seconds), total spatial areas scanned, details string describing scan type. Imageshift X and Y units are ???

Images (jpg by default but very easily changed to all Python Image are typically saved with a resolution such that 1 cm = 1 micron, which allows quick measure of feature sizes using the ruler in Photoshop (set Photoshop units to cm).

Combining spe files via averaging

AugerParamLog=Auger.combinespeloop(combinelist, AugerParamLog, movefiles=False)

One can combine a consecutive series of spe files (i.e. those created with AutoTool) using the combinespeloop function and with files to be combined indicated in this project's logbook. If your log file says to average-combine spe files 101-105 (i.e. filenumber=101, lastnumber=105 in xls file), a new csv named 101105 will be created which is an average of the counts/sec in 101 through 105. If these have multiple spatial areas (e.g. areas 1 and 2), then 101105 will contain a separate counts file for each (Counts1 and Counts2). Each of these columns is averaged separately (i.e. Counts1 in 101105 only averages data from area 1). When spe files are combined via averaging, only the combine averaged file is kept in the data directory (file 101105) with movefiles=True… the sub-files 101, 102, etc. are moved into the "sub" directory but are available there for analysis if necessary.

AugerParamLog=combinespelist(filelist, AugerParamLog, csvname='', movefiles=True)

One can also average-combine an arbitrary chosen list of spectra using the combinespelist command… just provide it with a list of the csv files you want (using glob for pattern matching, select files with tkinter or whatever)… if csvname is blank this will be autonamed based on

filenumbers combined or one can provide it with a name; Augerparamlog is return containing a new entry for this file

DONE WITH BATCH IMPORT CONVERSION
Now all of the above files (normally csv for spectra and jpg for sem and map files) are available for examination using any 3rd party software you choose.

Step 3: QUANT_MAIN
Now you can run quant _main which performs quantitative analysis via smooth-differentiation as well as via fitting and integration of direct counts. Augerparamlog normally contains every data file in your project folder (.sem, .spe, . map).. spelist is the subset of spectral files that you want to plot or process (filter this list with pandas to exclude problematic data or to select different subsets of spectral files)

1. Load modules and custom quant functions
2. Set data directory and choose subset of files (spelist) for quant process (if not still in memory from just imported files
3. Run smooth-differentiation quantitative analysis which returns Smdifpeakslog (lots of details regarding the elemental peaks in the S7D7 smooth-diff data)
4. Check smdifpeakslog accuracy with reportSD batch plotting (optional); probably best to do this before overwriting prior versions of smooth-differetial
5. Run direct integration quant method (background fitting and then direct integration of counts from subtracted data); returned in integquantlog; the direct integration does use the smdifpeakslog results (basically detects peak shift)
6. Plot direct counts and background fits (using reportcountsback) or use reportderivcnt/ reportderivcntall to plot deriv and direct spectra on same page.. can be pretty slow for big lists so maybe filter to smaller subset using spelist
7. Apply the k-factors to peak-to-peak amplitudes using calcadjcnts (for smdiff method) or to the integrated counts under peak (for integ method). This step converts amplitudes to scaled quantities that are used to then compute compositions (k-factors are in AESquantparams.csv and are determined from standards)
8. Compute Auger basis and compositions (atomic percent) using various sets of elements

Further con
<u>Import modules and custom quant functions:</u>
The functions for quant are stored in two separate files (named smdiffquant for all that use smooth-differentiation method and integquant for those using background fitting/direct integration)

<u>Set data directory/Choosing subsets of files/ Filtering bad data</u>

Having the AugerParamLog combined with pandas SQL-like database operations allows you to quickly produce subsets of file lists (named spelist by default) that contain the subset of files you want to process. For example if one know a certain spectral file is messed up for some reason (beam drift, weird oscillations in counts, count too weak, etc.) one can add the phrase "exclude" in the comments field of that spectrum in AugerParamsLog.csv. Then using masking (excludemask statement) you can quickly get rid of any questionable files or select files on any criteria/column in the AugerParamLog (i.e. by date, by project name, by # spatial areas or whatever) and work only with the solid data. Any files/areas not in the spelist are not plotted, process or added to composition plots. You don't necessarily have to scrub the files from Augerparamlog, integquantlog, backfitlog or smdifpeakslog.

Smooth-differentiation quantitative analysis

Normally I run the lines in these cells one by one (F9 in spyder).
First fill out/alter the Elements list. Each element/line chosen must have a corresponding entry in AESquantparams.csv. AESquantparams is very important to accurate quantitative/semi-quantitative analysis and each of its entries are separate described in the appendix below.

After you run smdifbatchquant, you need to manually save the results (i.e. smdifpeakslog.to_csv)… if you want to separately keep the results of a prior run for some reason, then manually rename that file before running smdifpeakslog.to_csv (automatically overwrites file). See description below for details of columns in smdifpeakslog

Before running the batch quantitative analysis using direct integration it is often useful to look over the plot reports from all files (esp. to exclude bad data) so do this next. Sometimes in the midst of choosing quant options (i.e. which elements, which lines of those elements), it is useful to make plot reports to quickly look through all of the available data. First select the spe files from the full AugerParamLog (also contains sem and map files) using pandas filtering options to create the spelist. Then run batch plotting to PDF for deriv data, integ data or both.

Plotting options:

reportSD – creates multipage PDF plots of chosen list of elements for all files in spelist; one must choose a set of plotelems (using Spyder F9 on desired row or any other means). Plot is saved in current working directory and is named according to PDFname. The function goes to smdifpeakslog, grabs the correct negative and positive peak positions and plots these as scatter points. If the two dots are exactly on the negative peak and corresponding positive peak then you can be assured that the amplitude for this peak is accurate (and quant results using smooth-derivative method are directly proportional to this amplitude).

```
plotelems=['S','C','Ca','O', 'Fe', 'Fe2','Fe1','Mg','Si'] # must match entries in AESquantparams.csv
reportSD(spelist, Smdifpeakslog, plotelems, AESquantparams, PDFname='SDreport.pdf')
```

reportcountsback – This plots the actual counts for all spectra all areas on separate PDF pages (and optionally the background fits if the backfit dataframe exists). Saved as PDFname in current data directory.

Direct integration quantitative analysis

Integbatchquant performs background fitting for all elements in Elements list, saves results to underlying csv data file, returns the parameters of the background fits (in backfitlog)… see appendix for a description of the results returned by the backfitlog and by the integquantlog. These are run separately for every spatial area in every spe file in spelist.

Background fitting is sample and element dependent. Fortunately it's easy to write separate functions that are callable to handle any number of different scenarios. If the defaults such as line or cubic don't work, you can use a different fittype in AESquantparam and create your own functions to handle individual peaks (i.e. make new functions analogous to fitcubic and makecubicbackground but use any other functions available from numpy or scipy). Then alter fitbackgrounds to call the new functions and handle the information they return.

Warning: Any background fits to a spectra are saved by default to its underlying csv data file and prior fits are overwritten. However the original counts column is never modified (and you also have the unmodified original *.spe file.

The default background fitting proceeds as follows:

Finds two regions (one above and one below) the element peak (using lower1, lower2, upper1, upper2 boundaries in AESquantparam); these boundaries are in index numbers and are all relative to the element's energy (negpeak value); any shift found during smdiff quant is then applied to shift these boundaries;

Perform linear fits over these regions (in refineboundfit function) and then possibly expand them (basically checks n adjacent channels with n specified as windowshift); if the additional points fall within some multiplier (default is 1.5) *standard deviations of the original fit, then the fit regions are slightly expanded;

Refit linear region over above-modified range, resulting in two linear fits (above and below peak)

If the two linear fits have similar slope & intercept (see comparelinfit function), then a single linear fit is made through the points above and below the background.

If the two linear fits have different slope, an interpolated fit is made in the region through the peak region (makeinterplinebackground).. spline style

If you don't like the way backgrounds are handled for any of your peaks, write your own function and insert it…. Use the fittype column in AESquantparams to tell fitting loop to choose whatever you define); In my standard case I use a special method for Ca due to large adjacent C peak and fittype=Ca calls these functions instead of standard ones

The background fits are saved to the csv file and now there's (optional) Gaussian fitting of the subtracted data with params stored in integquantlog).

The peak integration is then performed on the subtracted data (counts – background) with width specified in AESquantparams (using integpeaks function); all the relevant parameters are stored in integquantlog (see appendix for description)
Final step before computing compositions is to call AESinteg.calcadjcnts which applies sensitivity factor to the integcounts result and also errors (both in the k-factor and counting statistical errors) are computed and stored in integquantlog. See appendix or calcadjcnt function for details.

Comparing derivative and integral compositions:
By making a scattercompplot one can compare compositional calculations made in different ways (i.e. deriv. vs. integ. Composition or integ composition using Fe2 vs using Fe3 lines , etc.). This is often very useful to quickly find problematic spectra or batch quant errors. You can either plot the adjcnts directly (with basis=True; the adjcnts are already adjusted based on k-factors) or work with at. % (see composition section below). Especially useful is the dataframe with outliers from the linear regression that are returned (more outliers returned if you increase thresh setting). Then you can make a batch plot report of the underlying derivative and direct spectra from just these outliers… optional filtering by element beforehand.

```
compdata, outliers=AESplot.scattercompplot(Smdifcomp, integcomp, Elements,
joinlist=['Filenumber','Areanumber'], thresh=0.1, basis=True, errbars='y')
```

First two arguments: (smdifcomp and integcomp in this case) are two sets of compositional calculations to be compared via scatter plot
Joinlist: details the fields to use for the inner merge of this data… if set to filenumber, areanumber this will normally just compare the same file (i.e. 100.csv derivative composition vs. integral composition). Other joinlist options could be 'Sample', which would cross-compare different csv files on the same type of sample (larger scatter especially for heterogeneous samples).
Thresh – this is the threshold pvalue below which a point is considered an outlier; outlier points appear red in the scatter plots and are returned in the outliers dataframe
Basis: True- compares the corrected counts or amplitudes (directly proportional to counts for integ method or amplitude for derive method, False – here the atomic percentages are plotted instead of the corrected quantities
Errbars- 'x', 'y', or 'xy' – adds error bars to plots (generally only available for compositions derived from integral method, so setting depends on

Compdata – returns full inner merged compositional dataset (i.e. Fe is iron basis for the smdiff composition, Feb is iron basis for integ composition, %Fe and %Feb are iron compositions in at. % from comp datasets 1 and 2 respectively (smdifcomp and integcomp in this case).

Large outliers in the scatter plot indicate a clear problem with quantitative analyses. Since outlier info is returned (outlier dataframe) , one can then look at these spectra more carefully.

Outliers contains the residual, pval, and bonf ( ) for each point in each elemental scatter plot. If you want to examine a sulfur outlier, filter by element 'S', then sort by pval and plot using reportderivcnt. This will show the differentiated and raw counts from the outliers. Often problems will become apparent that lead to exclusion of this datapoint (again add exclude to AugerParamLog of this filenumber).

Computing elemental compositions:
The final step in calculating compositions is to again choose a subset of elements and then atomic percentage with errors is return (based on adjcnts and erradjcnts for all the elemental peaks chosen). The Mg at. % error is calculated by allowing Mg adjcnts to range from (adjcnts-erradjcnts to adjcnts+erradjcnts and at.% is calculated for both to give the range in Mg at. %) These are generally stored as xls files with the compositional sets in one tab and the underlying k-factors/peaks in another (using writecomps). These sets of compositions can then be plotted using matplotlib (including useful ternary plots) or can be opened in third-party software for plotting.

Fitting parameters in AESquantparams
You can make an altered duplicate version of AESquantparams (locally stored and loaded instead of global version). Sometimes this is desirable if you want to use different sets of standards for some samples (i.e. different k-factors, different allowable peak shifts, etc.)
Using 10 best spectra for each major element, ideal position of direct peak was determined relative to smdif peak… this difference stored in AESquantparams
See derivations in Auger_peak_positions.xlsx

After getting count integrations for all chosen elements (returned as separate lines in integquantlog), only can then calculate atomic compositions using any chosen subset of elements (just enter a new element list). First step is calcadjcounts which returns the counts corrected by kfactors and includes columns for errors (both error in each element's k-factor and counting statistical errors (2/sqrt(N)). Running calcadjcounts on existing integquant log does not modify counts, but uses current values of kfactors (either kfact from smooth-diff method or kfact2 from integral method) and associated errors to compute a basis for each element. These are added as columns to integquantlog.

Auger spectral imaging:
The PHI map procedure (and PHI *.map files) are always somewhat disappointing… after spending hours collecting data, the only data returned is a single image for each chosen element at your chosen resolution… no underlying spectra or integrated counts or anything. This is very qualitative data. Fortunately there is a workaround using multiplex and many spatial areas loaded via the Autotool. The annoying part is that a max of 20 areas is possible in a single

multiplex spe, so more areas than that have to be split into different spe files… however it's easy to reassemble this data with the enclosed scripts

Look at Auger_quantmap_main.py to see the process:

1) Use makesquarearray to create an n by n spatial array with an associated dataframe (QMpixarray is used to reassemble the collected data into an n by n elemental map) and an autotool file (loaded by PHI autotool and instructs the Smartsoft system to load 20 areas, run multiplex, load 20 more areas, run multiplex, etc.
2) Collect the spectral image/quantmap data; use tightly defined spectral regions in multiplex because serial acquisition time can make this process pretty slow (PHI map is only using 3 points); *might be best to reduce spectral resolution from 1eV/channel*
3) Reassemble the data from different spe files using combineQMdata… takes 5 files for 10x10 array and makes into a single file with 100 separate counts columns
4) Optional renumbering of spatial areas in SpatialArea log (to reflect fact that data has been combined to single csv)
5) From this single quantmap csv file, create amplitude maps (createampmaps) which compute S7D7 peak-to-peak amplitude for each peak and makes an nxn numpy array for each element passed (returned as a list of lists with element name string and corresponding numpy array)
6) Plotmaps function will plot each element map, basically gives you back what a PHI .map file has, but advantage is you have the underlying spectra (can get quantitative elemental ratios from any pixel in smdifpeakslog), also possible but not yet implemented is combination via averaging of adjacent pixels … i.e. draw an ROI on the amplitude map, choose all underlying data from the selected ROI and produce significantly better quantitative ratios from the combined data. Overlay image creates an SE image overlaid with an elemental map of your choice at variable opacity. Showsubareas can take compositional hotspots (e.g. Mg rich pixels from smdifpeakslog) and indicates their position in an image.

Other stuff

Background regions (Backregs list) is a list of energies (in eV) which gives a method to estimated the background noise level in the smooth-diff spectrum at various energies. I added these to get some measure of spectral signal/noise but this could be done a lot better.. not heavily used and will probably be changed. Anyway. ..

For shorter acquisions/ lower counts the noise amplitudes will be correspondingly higher and this gives a way to ballpark how significant your real peaks are. These amplitudes are stored in the lowbackamplitude and highbackamplitude columns in smdifpeakslog. Then you can compare the Amplitude of a given elements peak (amplitude column in smdifpeakslog) with noise from nearby regions (lowbackamplitude/highbackamplitude). These are also used later by various function to return actual compositions (threshold in calccomposition function is ratio of element amplitude to noise amplitude).

Alterations/Modification Notes:
If you change the pandas dataframe structure (i.e. columns or column names) in main, you must also make similar changes to the associated dataframes in the various function in the function definition py files.

See Auger_peak_positions for determinations of background fit regions, etc. from best C2010W crater spectra … possibly different parameters would be ideal for other samples (i.e. those with larger shift or different background
Larger fit ranges used for S and other low energy peaks due to occasional presence of low energy oscillation in counts (sometimes oscillation period of ~15eV or slightly more)
For different types of samples it's possible to use different kfactors, different energy ranges for fits, so a copy of the quantparams can be made, stored in data folder and loaded (instead of always using default global version)