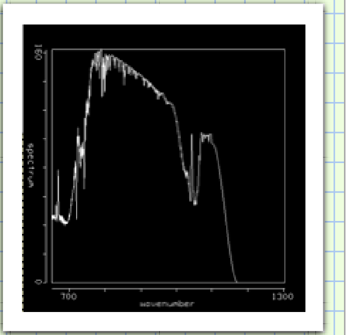
Progress report on test bench and tools - March 2004

- Test bench hardware consists of the Tines cluster (mostly Flenser, since the distributed computing aspect is not yet being heavily explored) and, unfortunately, barrage - which has MatLab, needed to run the reference version of the algorithms.
- The procedure for testing is as follows:
    - *given* a reference version of an algorithm, in Matlab, it is first necessary to prepare a version for testing purposes:
        - This was done by decomposing the Matlab script into fragments, which would then be remixed into test set generators via a python script which would also feed in needed arguments. Thus, the original matlab reference script:
        - *resample2.m*
        - Would be processed by a python remixer:
        - *resampleRunner.py*
        - And would produce, with appropriate inputs, the following matlab fragment, to be run instead of the original reference script:
        - *maxradLW_rolloff600-1100.real8.1025_genref0.9*
            - `% out file: maxrad_test/out_mlabcris_maxradLW_rolloff600-1100_vr0-9.real8.1025`

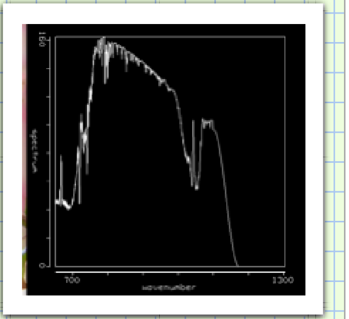
```
format long g
warning off
clear

in.v = fbfread( 'wavenumber_scales/wnLW.real8.1025', 1);

in.r = fbfread( 'maxrad_test/maxradLW_rolloff600-1100.real8.1025', 1 );

R = 10;
N = 2048;

vlaser_l = mean(diff(in.v))*N*R;

vlaser_b = vlaser_l / 0.9;
out.v = in.v / vlaser_l * vlaser_b;

in.dv = mean(diff(in.v));
out.dv = mean(diff(out.v));
jnk1 = repmat(in.v,1,length(in.v));
jnk2 = repmat(out.v,1,length(out.v));
arg3 = (jnk1' - jnk2) / out.dv;
F1 = (in.dv / out.dv) * sinc(arg3) ./ sinc(arg3/N);
out.r = F1 * in.r;

ff0 = fopen('maxrad_test/out_mlabcris_maxradLW_rolloff600-1100_vr0-9.real8.1025', 'write');
temp = fwrite(ff0, out.r, 'double');
fclose(ff0);
```

        - Finally, this last file was used to generate the matlab-reference spectrum:

- viewmultispectra ../wavenumbers/giftsWnLtrans0_9.REAL8.1025 dry_scene/ *out_mlabcris_maxradLW_roll600-1100_0_9.real8.1025*

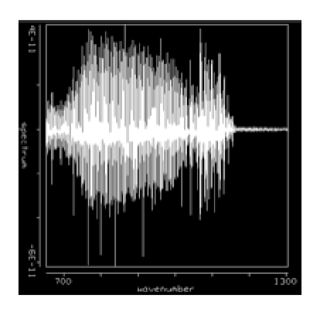


- The trick of applying rolloff:
  - Although the reference matlab code applied a rolloff to input spectra inside the resampler, it was implied that such a rolloff would occur at some earlier stage, and should thus not be included as part of the C++ code.
  - The first attempt to match the algorithms, namely to run an equivalent rolloff in python before calling the C++ candidate algorithm, revealed a subtle bug in numerical python which caused ringing errors between the two versions of the output.
  - At this point, I decided to apply the (matlab) rolloff to all input spectra before they were processed by either algorithm, to assure uniformity of input.
  - The lesson learned was that the exact nature of the rolloff profoundly affects subsequent results, especially if the rolloff induces a discontinuity in the derivative of the spectrum (as is the case in the algorithm presented to me), thus causing rining far into the "stable" parts of the spectrum.
- automated testing suites and naming conventions:
  - While it was desirable to come up with a good, consistent naming convention for all spectrum, wavenumber, and other product files resulting from testing with various combinations of input parameters, this goal was not completely reached, in the interest of timeliness. As is, the naming reflects all input parameters which affected the creation of a particular product (input spectrum, rolloff, transformation applied, algorithm used), but in a somewhat inconsistent fashion across different product types (but consistent within a product type, such as output spectrum).
  - This degree of consistent naming convention allowed the writing of summarizer scripts - but they needed to be fairly tied to the nature of the underlying tests, so they couldn't be generalized to other kinds of testing.
  - The naming problem was exacerbated in the resampling stage by the fact that wavenumber scales would change upon applying the transformation, and that there was a need to identify which wavenumber scale applied to which spectrum.
- Generating the C++ candidate implementation results
  - Was fairly easy, as there was no need to adjust the interface of the implementation, which would produce output spectra given input spectra and other parameters.
  - The python wrapper code which ran this version:
  - *resampleWrapper.py*:
  - would then produce output spectra named similarly to the matlab outputs:
  - viewmultispectra ../wavenumbers/giftsWnLtrans0_9.REAL8.1025 dry_scene/ *out_ccmemo_maxradLW_roll600-1100_0_9.real8.1025*



- Comparing the two output spectra was left to a Jython/VisAD application, *viewdiffspectra.py* which would also generate a numerical summary (max diff, RMS error) in addition to plotting the results.
  - A final script, *run_comparetests.py* would loop over all the test cases, comparing the matlab/c++ pairs using *viewdiffspectra* and concatenating the report summary in two files, difftestresults_LW and difftestresults_SMW.

- The corresponding comparison for the example spectra shown above:
- Wavenumber scale: wavenumber_scales/wnLWtrans0_9.real8.1025
  Spectrum 1: maxrad_test/out_mlabcris_maxradLW_rolloff600-1100_vr0-9.real8.1025
  Spectrum 2: maxrad_test/out_ccnote_maxradLW_rolloff600-1100_vr0-9.real8.1025
  Max difference: 4.686739885073621E-11
  RMS: 1.7518622255226314E-11
  plot saved in file: maxrad_test/diffplot_maxradLW_vr0-9.jpg



- Discussion of differences between the two implementations:
  - There are some minor, if very consistent differences between the two versions of the algorithm - they seem to depend more on the vlaser ratio than on the input spectrum.