These are just a few comments that might be of interest to someone reproducing or modifying this work. Most pertain to the last code “chunk” containing R code for a script that could make the filtering changes called for in this document.

Variance spectra, coherence and phase spectra, and cospectra are all constructed using Ranadu functions. Ranadu::VSpec() produced all the variance spectra. Ranadu::CohPhase() produced the coherence/phase plots, sometimes using the data.frame returned from that function in order to plot multiple results on the same plot, as in Figs. 5 and 6. CohPhase was also modified to return the cospectrum when a new argument, “returnCospectrum”, is set TRUE. The modifications have been incorporated into my github repository (https://github.com/WilliamCooper/Ranadu.git). The returned cospectrum from CohPhase is not smoothed, unlike other quantities produced by this routine, so I used a simple smoother: Ranadu::SmoothInterp() uses Savitzgy-Golay smoothing, and I used 49 points to give something like a 20-point effective window.

I have not explored single-pass filtering (which here could be explored by changing the “filtfilt()” function to “filter”). This should be checked sometime because it would be useful to incorporate this into nimbus if possible.

The flux calculation leading to Fig. 7 was developed for this project and is at best a “beta” result in which I am not yet fully confident. The “Flux” in the title of that plot, however, is a straightforward result from <wprime*Tprime> and is reliable.

The last code chunk applies the filtering advocated in this memo to a high-rate netCDF file. There is little value in applying this to LRT files because the 1-Hz frequency is not sufficient to apply the filters used here. (It might be worth exploring to apply this at high-rate and then average to 1-Hz?) The following are comments regarding the code in this last chunk:

1. The intent is that this could be extracted to a file FilterDynamicHeating.R that could be run as an RScript to modify an existing high-rate netCDF file.

2. The code segment copies the netCDF file to a new one with “T” suffix and then modifies that new file. If the new file already exists, it overwrites it!

3. It would be straightforward to modify this script to accept “Project” and “Flight” arguments either as run-time arguments or via interaction with a user. Janine has done this before to scripts I wrote, and she will know how to do this in ways that might meet her batch-processing needs. For now, I just edit the script and insert the Project and Flight variables.

4. For general use, this should probably look for the available temperature variables and then modify all that are available. That would avoid failure if run on a file that does not contain one of the current variables. It could also be extended to ATH2.

5. As noted in the document, there is some uncertainty regarding what recovery factor was used originally. I’ll update the document if this can be resolved.

6. The script is slow! The time is mostly consumed inserting attributes; inserting the data is relatively fast. It appears that the processor (incorporated into the R package “ncdf4” that I

use) has to spend a long time moving parts of the file around when it adds new variables and attributes. On my home system, it takes on the order of 3-4 minutes to process a high-rate file.

7. These calculations use a humidity-dependent function for the specific heat of air at constant pressure, as represented by Ranadu::SpecificHeat(, 1). It's not clear to me what nimbus does.

8. Although I don't think this should be used on a 1-Hz file, there is some protection against failure if it is. To avoid asking for filters outside what can be represented at 1 Hz, there is a branch that uses 2-s and 2.2-s filter cutoff periods instead for a 1-Hz file. This should have no effect at 2-s (used for ATF1) and only a small effect at 2.2-s (used for ATH1).

9. The reason for the zoo::na.approx calls is that the filter function that follows will fail if there are missing values. Therefore these calls are attempts to replace missing values with interpolated values to avoid that problem. As a last resort, if missing values remain, they are set to zero. "rule=2" handles sequences of missing values at the beginning or end of the time series by duplicating the last non-missing value.