ASTRO 530 **Spring 2026**

Homework Set #1 **due *Friday* January 16**

All normal homework rules posted on Canvas apply. I have provided a "drop box" on Canvas for your source code (or a link to it at some repository), which is required for every assignment. Every file that you write to generate figures should be included (you do not have to resubmit files/code/libraries from previous assignments unless you change them).

Remember to acknowledge or cite all of your resources, including each other and any generative AI you use (see the syllabus).

For all assignments, your target audience is yourself just before you started this assignment. As an exercise in LaTeX, in this first assignment you should include the following (possibly contrived) elements, in context: a plot with legible axes with units, a table with at least 2 rows and columns, an acknowledgements section, and a references section.

I will assume you are using Python in this class, and I have a complete set of solutions to the homeworks in Python. You may use another language, but unless it is IDL I won't be able to help you with it (and you may find it difficult to work with your classmates).

1. Computing and Plotting: Planck Function.

Throughout the semester, we will be making numerical computations of things like emergent intensity and flux from various simple atmospheres. The computational effort becomes more complex as we go along, but can be made easier if we take advantage of work early in the semester, i.e., by writing our code in modular fashion.

Each such function or procedure should be available in a library file or directory for future use. For instance, for this problem you might create a function called `Planck` in Python, and making it part of a module (perhaps called `astro530`) from which you can `import` it. My rule is any code that appears verbatim 3 times in your code should be separated out as a function.

Do *not* just keep one big notebook or py file and keep adding to it each week. In other words do *not* produce giant scripts from which you copy-paste useful elements homework after homework. Maintain a library and call functions in it from your script. Each homework should exist in a fresh script, perhaps called hw3.ipynb or some such, that only contains the code needed for that week's homework. You shouldn't need to paste any code into it, you should call code from a library that you've been building up all semester.

Another element of modularity is not repeating things like constants definitions. You need to have a consistent way to easily access physical constants in the units you expect.

You might write a function for this, but most modern scientific computing languages have them built in. Python users: use the `astropy.units` and `astropy.constants` modules for this. Do *not* define any constants "by hand" (mistyping or using inconsistent values for constants are some of the hardest bugs to track down!)

In general, it is fine to use existing library functions (and often better, in fact, because they will be robust and fast) but *you must be careful that you understand how they work and what their limitations are.* This usually means writing your own version and comparing outputs.

For example, we will very frequently wish to evaluate the Planck function $B_\nu(T)$, so you need one you can trust. Python has a function you can use in its `astropy.modeling.models` library with full units support. But we need to be careful: the Planck function is defined for both flux and intensity, and these differ by a factor of π—we need to know which one we're working with and what inputs it reliably accepts.

For this exercise, we will use wavenumbers, which are like wavelengths except that, being astronomers, we have to define them backwards as *inverse* wavelengths so that they're really frequencies. UC Berkeley infrared astronomer James Graham refers to them as "God's units," and I know better than to argue with James Graham about radiation. They really are convenient once you get used to them. Symbolically, wavenumbers are defined as $\tilde{\nu} = 1/\lambda$. For this exercise, measure λ in microns.

Python users: note that `astropy` knows all about wavenumbers! Check out the `equivalencies` feature and `u.spectral()`.

(a) Write your own Planck function you trust that accepts vector input and in units you understand. Use this to confirm and understand the input and output of a library Planck function. Explain in your assignment which Planck function `astropy` gives you by default, and write your own version of the "other one."

b) Make three plots: one of B_ν vs. $\tilde{\nu}$ on the range 0 to 12 $\mu\mathrm{m}^{-1}$, one of $\log B_\nu$ vs. $\tilde{\nu}$ on the same range, and one of $\log B_\nu$ vs. $\log \tilde{\nu}$ (on the range $-1.0 < \log \tilde{\nu} < 1.2$).

Note: When I write "plot y vs. $\log x$ " I mean that your x-axis should plot x with logarithmically spaced tick marks, not that you should calculate $\log x$ and plot that quantity with evenly spaced tick marks.

(b) On all 3 plots, show curves for T=10,000 K, 7,000 K, and 3,000 K. "log" is base 10. Choose appropriately illustrative y ranges, and include a legend.

In the next assignment, you will be evaluating the areas under these curves with numerical integrals.

Note: I get very picky about LaTeX details. For this assignment, familiarize yourself with how to render axis labels using LaTeX formatting in your plotting language. Python `matplotlib` users: use the `r'LaTeX stuff here'` formatting formalism.

Also, `matplotlib`'s defaults for scaling and offsetting axes are *terrible.* You should *never* publish a plot with a number at the top left or bottom right of a plot that tells the users how the axis is scaled. *Always* scale the axis yourself and provide the scaling (including units properly rendered in LaTeX) in the axis label.

Also, your plots need to be completely legible to me (and I am becoming more and more farsighted). This means

- You should never use screencaps to convert your plots into files: instead, use `matplotlib`'s `savefig` method to output vector graphics (I use the `svg` file format) so that LaTeX can reproduce them at full resolution.
- Making all numbers and characters large enough for me to easily read, including in the legends.
- Do not use light colors like yellow in your plots. They might look OK on your monitor but they won't generate enough contrast against white paper on your printer.
- Reviewing your printer's output to ensure it's all easily legible. You do not need to print in color, but if you use black-and-white make sure you use different line weights and styles to ensure different curves are easily distinguished.