Case Study - Planetary Motion

This case study solves for pair-wise interactions between planets in the Solar system. Because this is quite a heavy calculation we can speed it up a lot! The code changes quite a lot, so we've provided multiple files all doing the same calc.

For the C calls, you will need a C compiler (e.g. gcc) and to build the C code as a "shared object". A simple build script for gcc is provided.

- planets.py

  - A pure Python implementation

  Info:

  Pluto is included as a planet, and we use G=1 for simplicity. You will need matplotlib for the "demo" to work, and if you're working remotely using ssh you will need X forwarding (ssh -X)

  Try:

  Run the demo (expect ~30 seconds), read the code, and work out how it works

  Profile! Even though we "know" doing numerics in Python wont be fast, we should verify where the time is being spent before proceeeding!

- planets.c

  - A pure C implementation

  Info:

  Before we go on, this code is simple enough to replicate in C and that tells us immediately how fast we are aiming to get it - it'll also be useful later to have this code.

  Try:

  Compile with gcc (gcc planets.c -o planets -lm) The -lm means "link the maths library) and run. You can time this using `time planets`. Observe. Try compiling with optimization (-O3) and rerunning. Does it help much?

(Hard) Estimate the number of operations in the C code - how fast might it go? (There is no threading and unlikely any vectorisation).

- planets_jit.py

  - Pure Python using numba.jit to speed up the core

  Info:

  Based on the profile from the first section, we focus first on pair_force and integrate

  Try:

  Run the new demo (expect a significant speedup!). Compare to the old code - note how little we had to change! This is a super example of a cheap bonus

  Profile again! We've changed the runtime a lot, so we have no idea where the remaining time might be spent. Note if you profile the _first_ run, where the jitter runs, you will get a lot of unhelpful info

  Any guesses where "pair_force" has gone? What's the hotspot now?

- planets_hybrid.py

  - Outsource the "pair_force" to C instead

  Info:

  Suppose the 'jit' didn't work, or wasn't reliable - we can "outsource" the hotspot into some C code. You will need to compile a shared object for pairforce using the build_pairforce script before running the Python

  Try:

  Compile the .so and run the example. How much does this help?

  (Hard) Why is it so terrible?

- planets_lib.py

  - Outsource everything to C

  Info:

  We want the Python wrapper for the interface and the plotting, but the calculations could be much faster. But we can't just pass off parts of the load. A solution is to implement the entire solve in C, and just start it from Python and get the result out for examining and plotting. You can complile the C library shared object using the build_planets_lib script

  Try:

  Compile the .so and run the example. How fast are we now? How does this compare to the pure C?