Profiling

This document outlines the profiling of the code and attempts to improve both CPU and memory usage.

Tools

I am really not sure what tools to use for the best here. One thing is certain, this needs to be done at the command line to avoid bloating the code: in particular `jupyter notebook` is horrendously memory guzzling. The inbuilt profiling tools are controlled by runtime boolean parameters `b_profile_cpu` and `b_profile_mem`.

CPU time

See https://realpython.com/python-timer/#the-python-timer-code for hints.
Of the three methods described below, I have found cProfile to be the most useful: it does not require editing of the code and it produces a modest-sized profile file.

cProfile

This is a command line tool that can tell you in which functions the code spends most of its time. It is easy to use as it requires no modification to the code itself:

```
python -m cProfile -o output_dir/L-Galaxies.prof L-Galaxies.py
```

Then analyse using

```
python -m pstats output_dir/L-Galaxies.prof
```

which pops you into an interactive interface. The recommended basic commands are

- `strip` – cleans the output
- `sort (tot|cum)time` – sorts by time spent in functions excluding (tot) or including (cum) subfunctions.
- `stats #` – shows the top # entries.

Once you know which function/method is taking most of the time, then you can use a line profiler to determine which individual lines of code are at fault. This requires an `@profile` directive to be placed in front of the function to be profiled.

```
pip install line_profiler
kernprof -lv -o output/Mill/L-Galaxies.py.lprof L-Galaxies.py
python -m line_profiler output/Mill/L-Galaxies.py.lprof
```

C_time class

A simple class that holds a dictionary of numpy records:

- key – name of the record

- value – n_start, n_stop, cpu_time_start, cpu_time_total

with methods:

- __repr__ – prints out the dictionary.
- dump(filename) – saves the dictionary as a pickle file `filename`.
- start(name) – adds an entry to the dictionary with key `name`, or reopens an existing one.
- stop(name) – accumulates the time spent in cpu_time_total.

This routine should be relatively lightweight. It is used to track the time taken to process each graph.

codetiming.Timer

https://pypi.org/project/codetiming/

This can be used as a decorator to profile individual python functions. It is useful but the output seems incredibly bloated. For example, on processing just 1000 halos it produces an output file that is 300MB in size.