1 Refactoring

Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations, each of which “too small to be worth doing”.

However the cumulative effect of each of these transformations is quite significant. By doing them in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time.

– Martin Fowler
Refactoring: Improving the Design of Existing Code

Refactoring one’s code is the equivalent of editing your work in the writing process. As we begin to build on our programs from week-to-week, refactoring will become more and more important and valuable.

1.1 The $N$-coupled oscillator

Recall the equations that describe the $N$-coupled oscillator:

$$y_{pn} = A_{pn} \cos(\omega_n t),$$

where

$$A_{pn} = C_n \sin \left( \frac{pm\pi}{N + 1} \right) \quad \text{and} \quad \omega_n = 2\omega_0 \sin \left( \frac{n\pi}{2(N + 1)} \right).$$

Return to your program from last week that visually displays normal modes.

**Task 1:** Based upon our class discussion, refactor by:

- making more meaningful variable names
- grouping parts of your code and re-ordering the logical sequence
- removing any ‘magic numbers’
- making any stylistic changes to create more readable code.

**Task 2:** Refactor your code by defining an amplitude function that depends on $N$, the number of particles; $n$ the mode of interest; and $p$, the label for the $p^{th}$ particle. (Of course, use the variable names that you have carefully chosen.)

You will call this function from inside your plotting loop.

After doing so, check that your program still works as expected.
2 Fourier Analysis

The linear combination of several normal modes of oscillation is at the heart of Fourier analysis. So, having a way of generating a single normal mode will be very useful. We will now modify our code so that we may easily call on any particular normal mode.

Task 3: Modify your program by introducing a function that contains all the information needed to generate a single normal mode. That is, define a function that depends on $N$, the number of particles; $n$, the mode of interest; $p$, the label for the $p^{th}$ particle; and $t$, the time. (This function can then represent a ‘snapshot in time’ of the entire $N$-coupled oscillator.)

2.1 Arrays vs Functions

Generally, lists and arrays are easier to work with than functions. In fact, the computer algebra package Mathematica uses lists as its underlying structure.

Task 4: Modify your program by making a function involving $N$, $n$, and $t$ that returns the normal mode $n$ for the entire system as an array.

Task 5: Modify your plotting loop so that it no longer uses the xaxis variable. Now, arrange for your program to add three different normal modes with differing contributions. Display your linear combination interactively, and experiment with varying combinations and modes.

3 Final Report (due Monday, February 29)

Adapt your program to have the following settings:

(i) $N = 21$ oscillators;

(ii) addition of three normal modes with pre-set contributions;

(iii) displays propagation of total oscillation for 15 seconds.

Select a combination of modes strategically so as to illustrate a particular feature of Fourier analysis. Discuss your illustration using no more than one paragraph.

Submit your final program and paragraph, this time as email attachments.