Analysis and design in many areas of Science and Engineering are often done using simplified “linearized” mathematical models. If the system involved never operates outside of the range for which the linearization is valid, then all is well. All of the linear or reduced-order analysis in the world may be for naught, however, if the linearized range is exceeded by driving it beyond its small-signal range or outside of the range of validity for your linearized (or reduced-order) model.

Some of the most dramatic catastrophes involve the structural failure of buildings or bridges. Limiting the design to static analyses has usually been the cause. When dynamically stressed by earthquakes, turbulent winds, or traffic, structures begin to flex and oscillate. In a simplistic sense, if the forcing function of the wind or traffic matches the natural mechanical frequency of the structure, then a destructive “resonant” response may result. Owing to nonlinear behaviors, “resonance” is a misnomer. Analysis and prediction are difficult or impossible.

A classic example of this is given by Duffing’s Equation:

$$\ddot{x} + k\dot{x} + x^3 = B \cos t$$

1. Apply one of the numerical methods you’ve learned in this class to solve the above equation for x. Set it up so that k, B, and the integration timestep $\Delta t$ can be specified.
   a. Using an integration timestep of 100 ms, experiment with different values of k and B. Report on how many different responses you can observe.
   b. Afterward, ponder the significance of the equation. In terms of an actual physical system, what does each term represent? Consider mechanical, electrical, and other types of systems. Does the numerical integration method you’ve used influence the apparent behavior of this system? Explain.

Next, we’ll be considering fractals. One of the simplest ways I’ve found to create a fractal requires that you apply Newton’s Iterative Method to solve a nonlinear equation. Let’s consider the following simple equation:

2. For n = 2, 3, and 4, can you solve for all roots by inspection? Apply Newton’s method to solve the equation. Using various initializations for your iteration, verify that you can obtain all roots. (Exercise 4 builds on this and leads you through the actual creation of a fractal).
   a. Can you predict which root a given initialization converges to?
   b. Can you predict how many iterations it will take for the solution to converge?
   c. Can you predict whether the solution will converge for a given initialization?

Another way to create a fractal pattern is by using Markov Processes. A Markov Process is actually a set of iterative processes. The result of each iteration is referred to as a “state.” Which iteration is used to obtain the next state is a probabilistic function of which iteration was used last. An example will be given in class.
Fun with Programming: Additional Exercises. Suggest that you pick one of the following:

3. Learn more about Duffing’s Equation. Using the tool you developed in Exercise 1,
   a. Develop phase plane plots for Duffing’s Equation, showing examples of period 1, period 2, period 3, and chaos.
   b. For a chaotic response, plot the Poincaré section. (Hint: points must be sampled once each period of the forcing function. The period of the forcing function must be an integer multiple of the forcing function.)
   c. Comment on sensitivity to initial conditions, damping, and the magnitude of the forcing function. Use the results generated above to illustrate.

The following exercises require some knowledge of graphical programming:

4. Learn more about fractals. Using the tool you developed in Exercise 2,
   a. Equating each pixel position on the screen of your computer as a different initial condition on the complex number plane, iterate to a solution. Set the pixel’s color according to which root you converged to. Hint: using an initial value of (0,0) will result in a divide by zero in the Newton iteration. Work around it.
   b. Make a more detailed pattern by shading the pixel according to how many iterations it took to reach the root. Plot it as a bitmap. If using Matlab, consider using the “image” plotting function.
   c. Demonstrate self-similarity by “zooming in” (make successive runs with successively smaller “windows”).

   a. Implement it using the probabilities given. Display a monochrome image, or colorcode each pixel according the “state” that produced it.
   b. Skew the given probabilities. Comment on how the fractal changes.