Due: Friday, April 12

Please be sure to read this assignment in detail before beginning.

1 Problem Introduction

In this homework assignment, you will be implementing a number of different numerical methods for solving systems of ODEs. In particular, we will be concerned with the mass spring systems given by

$$\vec{x}_t = A\vec{x} + \vec{b}(t), \quad \vec{x}(0) = \vec{x}_0,$$

where

$$A = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{pmatrix}, \quad \vec{b}(t) = \begin{bmatrix} 0 \\ f(t)/m \end{bmatrix}, \quad \vec{x}_0 = \begin{bmatrix} x_0 \\ v_0 \end{bmatrix},$$

and

$$\omega_0 = \sqrt{\frac{k}{m}}, \quad \zeta = \frac{c}{2m\omega_0}.$$

The system parameters are:

- m: the mass of the object
- k: the spring constant
- c: the damping coefficient
- $f(t)$: the outside forcing
- x_0: the initial displacement
- v_0: the initial velocity
- ω_0: the natural frequency of the system
- ζ: the damping ratio

For this homework, we will only be concerned with periodic forcing that takes the form

$$f(t) = \cos(\omega t).$$

2 Numerical Methods

In this homework, you will be responsible for creating function that implement the four methods we discussed in lecture to solve these mass spring systems:

- `FE`: Forward Euler
- `BE`: Backward Euler
- `CN`: Crank-Nicolson
- `RK4`: fourth order Runge-Kutta

Each of these functions will take 7 input values:

- `w0`, `z`, `m`: the natural frequency, damping ratio, and mass of the system
- `w`: the forcing frequency
- `x0`: the initial condition, given as a numpy.matrix object, column vector
- `T`: the final time to solve until
- `N`: the number of timesteps to use while solving

Each function should output two lists:

- `x`: a list of the displacement values at each timestep
- `t`: a list of the times corresponding to the values of `x`

For each of these functions, you should:

- Create the numpy.matrix `A` and compute the timestep `dt = T/N`.
- Preallocate the two length `N+1` lists `x` and `t` (you can even fill the list `t` with its values now, and store the initial displacement as `x[0]`).
- Perform any other initial setup (i.e., performing any matrix inversions or LU factorizations as necessary for the given method).
- Loop over each timestep:
  - Compute the numpy.matrix vector representing $\vec{b}(t)$ using the values of `w`, `m`, and the current time in the simulation.
  - Perform the update step for the given method.
  - Store the value of the updated displacement in your list `x`.

3 Testing the Methods

Once you have completed creating the code, you will need to test your methods. You will do this by specifically considering the following set of parameters:

$$\omega_0 = 1, \quad \zeta = 1, \quad m = 1, \quad \omega = 1.$$

You will note that this corresponds to an critically damped oscillator being forced at its natural frequency. We will take the initial conditions

$$\vec{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

so that there is no initial displacement or velocity. Instead, all motion will be determined by the forcing and the response. In this case, we expect the oscillator to ultimately match the forcing oscillations, as any other response to the forcing will be exponentially damped away. Indeed, the exact solution is given by

$$x(t) = \frac{1}{2}(\sin t - te^{-t}).$$

In order to test the four methods, we will consider solving until the final time of `T=10`. For each method, you will consider a range of `N` values and compute the error of the solution at time `T=10` (i.e., you should simulate until time `T=10`, and then compare that final value to the above function $x(t)$ evaluated at time $t = 10$, in absolute value).

You should choose a range of `N` values so that the size of the timestep, $\Delta t = T/N$, ranges from 0.1 to 10^{-5}, i.e., you should choose `N` to be 100, 1000, 10^4, 10^5, and 10^6, and you can include intermediate values if you wish.

Once you have computed the error for this range of `N` values for each method, you should generate four plots of error versus `N` on a log-log scale, and save these plots. You can do this by directly taking the log of the error and `N` values using `numpy.log10`, or by setting the scale of the plotting to be a log scale with the commands:

```
ax.set_yscale('log')
ax.set_xscale('log')
```

Make sure that your plots display the correct slopes before continuing.

4 Beats and Resonance

Now that we have verified that the methods are performing correctly, we can begin to explore some of the behaviors of the system. For this part of the homework, we will consider the model with parameters and initial condition

$$\omega_0 = 1, \quad \zeta = 0, \quad m = 1, \quad \vec{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We can see that this corresponds to an undamped oscillator with a natural frequency of 1.

For this part of the homework, we will solve until the final time of `T=100`. **You should select your favorite of the methods to use for this part and the next part of the homework**. Once you have selected your method, you will need to determine a suitable value of `N` to use. You can use the error plots from the first part of the assignment to inform your choice (noting that we have now increased the final time `T` by tenfold), or you can simply choose a value and examine your results to ensure that your selection is giving reasonable results. A good starting place is typically the value that gives $\Delta t \sim 10^{-3}$.

To ensure that your results are reasonable, recall that for an undamped system, the amplitude of the resulting oscillations should not decrease. If your timestep is too large, you will notice that the amplitude of each 'beat' (to be discussed shortly) will decrease slightly in time. This behavior is the result of what are known as *dissipative errors*, where energy is lost from the system due to a cumulation of numerical errors.

Once you have determined the value of `N` that you will use, simulate the mass spring system with the following forcing frequencies and plot the results:

$$\omega = 0.8, \quad \omega = 0.9, \quad \omega = 1.$$

In the first two cases, you will observe a phenomenon referred to as 'beats,' where the amplitude of the resulting oscillations will itself oscillate with the difference of the forcing frequency and the natural frequency. This occurs due to periods of either constructive or destructive interference between the two sinusoidal curves that make up the particular and homogeneous solutions.

Indeed, consider the case where $\omega = 0.8 = 4/5$. The exact solution is

$$x(t) = \frac{25}{9}\left(\cos\left(\frac{4t}{5}\right) - \cos t\right).$$

So in this case, we would expect to see beats with a frequency of $1/5$.

In the case where $\omega = 1$, we are forcing the undamped system at its natural frequency. So we would expect to see resonance, where the amplitude of the oscillations increases linearly in time.

Save your three plots of the solution $x(t)$ versus time for each of these forcing frequencies.

5 Frequency Response Function

Using the same method and values for `T` and `N` as you did in the previous section, you will now be responsible for creating what is known as a frequency response function or a Bode plot. Consider the (very) underdamped system:

$$\omega_0 = 1, \quad \zeta = \frac{1}{10}, \quad m = 1, \quad \vec{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Due to the presence of the nonzero damping, we do not expect to see exact resonance as we did in the previous section. However, the system can still respond with very large amplitudes when forced at a frequency near its natural frequency. Thus it is a common practice in engineering to test a product's response to a variety of forcing frequencies and to ensure that the resulting oscillations never exceed the materials' specifications.

For this part of the homework, you will consider a range of forcing frequencies `w` from 0.1 to 10, incrementing by 0.1. For each frequency, you should simulate the system until time `T=100`. You should then compute the maximum displacement achieved, in absolute value, over the course of the simulation and store that value (i.e., compute `max(abs(x))` and store it). You should then plot this maximum displacement versus frequency on a log-log scale. You should notice that this function peaks near the natural frequency of $\omega_0 = 1$ (which is at location 0 on the log scale), and the maximum displacement should drop off at nearly a straight line for larger forcing frequencies.