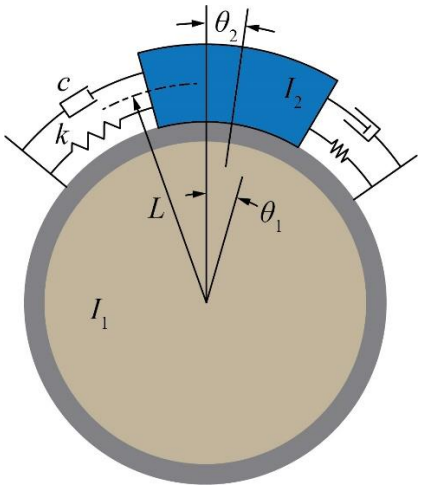
ENGR 105 – Introduction to Scientific Computing

Assignment 11

Due by 11:59 pm on Weds. 4/6/2022 on Gradescope

Start planning/working on your final project!

Problem 1 (50 pts): You are designing a tuned mass damper system for persons that suffer from essential tremor and Parkinson's disease. The system is worn on the wrist much like a watch with the intention of damping rotations of the wrist. The system consists of a weight connected to springs and dampers that follows a track around the user's wrist. You wish to investigate the effect of the damping coefficient, *c*, on the performance of the system. A diagram of the system follows.



The coupled, second order linear ordinary differential equations of motion for the system are as follows.

$$\begin{bmatrix} I_1 & 0 \\ 0 & I_2 \end{bmatrix}\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}+\begin{bmatrix} cL^2 & -cL^2 \\ -cL^2 & cL^2 \end{bmatrix}\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}+\begin{bmatrix} kL^2 & -kL^2 \\ -kL^2 & kL^2 \end{bmatrix}\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}=\begin{bmatrix} T\cos(\omega t) \\ 0 \end{bmatrix}$$

I_1 is the moment of inertia of the wrist (units: kg·m^2), I_2 is the moment of inertia of the tuned mass damper, *c* is the coefficient of damping (units: N·s·m^{-1}), *L* is the distance from the center of the users wrist to the tuned mass damper (units: m), *k* is the stiffness of the system (units: N·m^{-1}), θ_1 is the current angle of the wrist (units: rad), θ_2 is the current angle of the mass, *T* is the torque of the wrist oscillations (units: N·m), ω is the angular frequency of the oscillations (units: rad·s^{-1}), *t* is time (units: s), and the overdots represent derivatives with respect to time. Note that angular frequency (units: rad/s) is related to frequency (units: Hz), *f*, via the following relationship.

$$\omega = 2\pi f$$

Your objective is to perform several frequency sweeps of the system response for a given set of physical parameters and various values of *c*. Central to this effort is the function

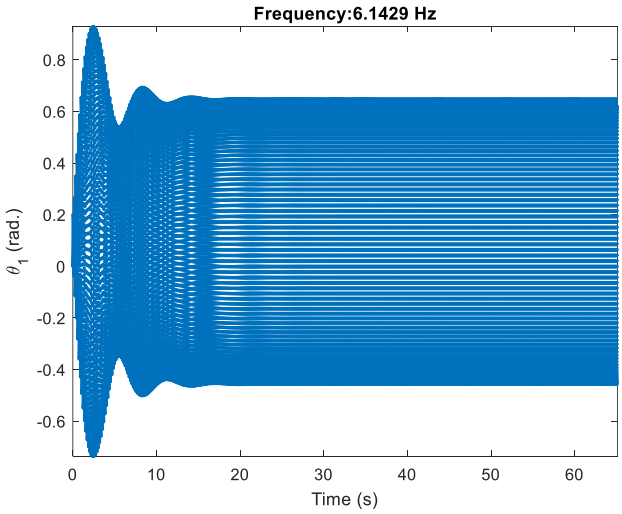
`frequencySweep`, which will take in the system and solution parameters and report the frequencies investigated and the resulting amplitude of the wrist at each frequency. This function should have the following function declaration.

```
function [f,amp] = frequencySweep(I1,I2,c,k,L,T,fL,fU,N,Np,Ns,vis)
```

Inputs `I1`, `I2`, `c`, `k`, `L`, and `T` are scalars that represent the physical parameters shown in the system diagram. Inputs `fL` and `fU` are the lower and upper frequency bounds, respectively, of the frequency sweep. Input `N` is the number of equally spaced steps simulated between the frequency sweep bounds, `fL` and `fU` inclusive, and is assumed to be positive and integer. Input `Np` is a scalar that represents the number of periods in each solution (at each frequency investigated) and is assumed to be positive and integer. Input `Ns` is a scalar that represents the minimum number of solution time steps in each solution period and is assumed to be positive and integer. Input `vis` is a scalar Boolean (takes the value of `true` or `false`). When `vis=true`, `frequencySweep` should plot θ_1 as a function of time for each simulation. These plots will be useful when evaluating if each simulation has reached steady state.

Output `f` is a 1 x `N` vector that contains each of the frequencies investigated in units of Hz, starting with `fL` and ending with `fU`. Output `amp` is a 1 x `N` vector that contains the peak-to-peak amplitude of the wrist for each frequency investigated, which should be computed as the difference between the maximum and minimum of θ_1 for the last period of each simulation. The last period of the simulation is used when computing amplitude in order to ensure that the simulation has reached steady-state.

The following plot shows a representative solution of the system at a given frequency and for `Np = 400` periods of integration time. This plot was generated since `vis` was set to `true` during the integration. Note that θ_1 varies considerably in the first 20 seconds of the integration (transient behavior) and settles afterwards (steady-state behavior). The amplitude of the wrist for this simulation was calculated from the last .16279 s (1 / 6.1429 Hz) of the simulation (the last period of the simulation).



Because the default solution resolution of `ode45()` is insufficient for this model, it will be necessary to enforce a maximum time step for the differential equation solver. You are tasked

with this by enforcing a minimum number of time steps (`Ns`) in each solution period. This can be achieved by setting the `MaxStep` property of the ODE solver using the following strategy.

```
options = odeset('MaxStep',maxStep);
```

`options` should then be provided as fourth argument to `ode45()` when it is invoked. `maxStep` is related to `Ns` according to the following.

$$\text{maxStep} = \frac{1}{f \cdot \text{Ns}}$$

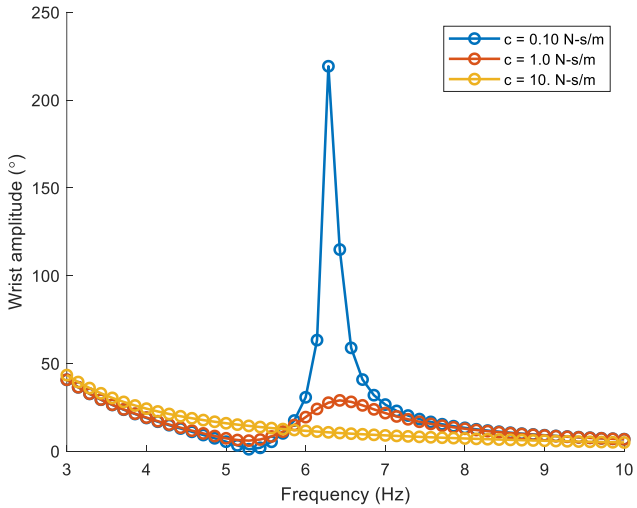
Your `frequencySweep` function should call on the state-space representation of the equations of motion (EOMs) stored in a function with the following function declaration.

```
function dtheta = massDamperEOMs(t,x,T,f,I1,I2,L,k,c)
```

Inputs `t`, `T`, `f`, `I1`, `I2`, `L`, `k`, and `c` maintain the same definitions prescribed for `frequencySweep()` and are all scalar. Input `x` is a generalized representation of θ and will be a 1 x 4 vector.

Each simulation at a given frequency should start with the initial conditions $\left[\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\right] = [0,0,0,0]$. These initial conditions should be prescribed within function `frequencySweep`, not function `massDamperEOMs`.

Finally, create a script that calls on `frequencySweep()` to investigate wrist amplitude as a function of frequency for `c = .1`, `c = 1`, and `c = 10` and the following simulation parameters: `I1 = 5e-4`, `I2 = 2e-4`, `k = 250`, `L = .03`, `T = .1`, `fL = 3`, `fU = 10`, `Np = 200`, `Ns = 40`, and `N = 50`. This script should produce a plot that looks very similar to the following.



Upload all associated m files and a jpg copy of your amplitude plot. Identify the names of all files associated with this problem in your README.txt.

Problem 2 (50 pts): Monty the mouse has been placed in a maze. Due to his nearsightedness, he has trouble efficiently navigating the maze. He starts moving in a random direction, picks a random direction to move when he bumps into a wall, and randomly changes direction every couple of steps (as mice tend to do!).

Create a simulation of Monty traversing a maze until he reaches an exit. This simulation should have the following function declaration.

```
function noSteps = MontyTheMouse(mazeImage,row,col,s2c,realTimeVis)
```

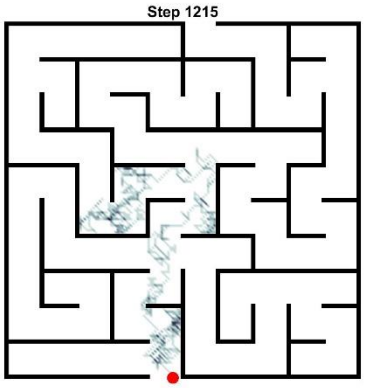
Output `noSteps` represents the total number of steps it took for Monty to reach an exit. A step is considered any move to a neighboring index on his movement matrix. Neighboring indices include any combination of -1, 0, and +1 row and/or column moves the current index noninclusive – that is, a step to the north, northeast, east, southeast, south, southwest, west, or northwest.

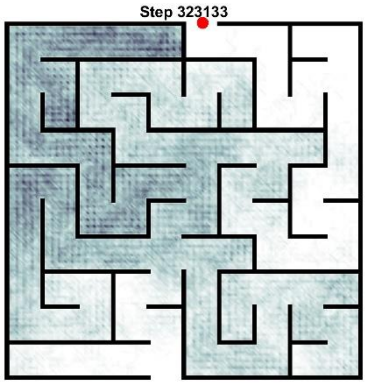
Input `mazeImage` is the name of the .png maze image that Monty needs to solve. Inputs `row` and `col` are nonzero, positive, and integer scalars that represent the starting row and column location of Monty, respectively.

Input `s2c` is a nonzero, positive, and integer scalar that represents the number of steps Monty will take before randomly changing direction. The counter that determines if Monty should randomly change direction should reset when he bumps into a wall or has taken `s2c` steps since the last random change in direction.

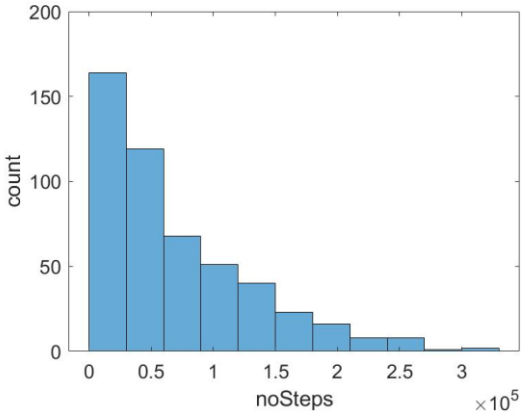
Input `realTimeVis` controls real-time visualization of Monty and his history of positions. It anticipates a character vector input of `'slow'`, character vector input of `'fast'`, or an empty set. If `realTimeVis` reads `'slow'`, Monty's position should be updated for each step using `drawnow`. If `realTimeVis` reads `'fast'`, Monty's position should be updated for each step at the rate of `drawnow limitrate`. If `realTimeVis` is an empty set, only the final visualization of Monty's path should be shown.

The real-time and final visualizations should include the maze walls in black, the current position of Monte as a red, filled circle, and the frequency with which he visited particular indices. The example images below show Monty's fastest and slowest path to finding an exit for a starting location of `row = 86` and `col = 58` and for 500 simulations. The historical path of Monty utilizes an inverted `bone` color scale to represent the number of times Monty has visited a particular index. White pixels have not been visited while those that have been visited frequently appear dark gray to black.





Create a script that calls on `MontyTheMouse` 500 times with a starting location of `row = 86` and `col = 58`, `s2c = 3`, saves each final simulation figure in .jpg format using a reasonable naming convention and at a resolution of 300 dpi, and displays a histogram of the results like that shown below.



You have been provided with `10 by 10 orthogonal maze.png` on which to run the simulation to produce the histogram. As well, you are provided with `20 by 20 orthogonal maze.png` and `40 by 40 orthogonal maze.png` to test the generality and performance of your algorithm.

Upload all associated m files and jpg copies of the slowest solution, the fastest solution, and the histogram. Identify the names of all files associated with this problem in your README.txt.