Math 330 Fall 2020 **Final Project**

1. Consider the damped harmonic oscillator system discussed in class. Recall this equation shows up in a number of physical systems, including elastic pendulums and RLC circuits. In the simplest case, this results in one second order ODE (initial value) problem of the form:

$$x''[t] + 2\alpha x'[t] + \omega^2 x[t] = 0$$
$$x[0] = x_0 \quad x'[0] = v_0$$

with $\alpha, \omega \geq 0$. These constants depend on the specific problem parameters: for example, for the case of a simple, 1D pendulum, $\alpha = d/2m$ and $\omega = \sqrt{k/m}$, with m the mass of the pendulum, d the friction coefficient and k the stiffness constant of the spring.
(a) Using Mathematica's DSolve and operations such as Assuming, Simplify, Expand, etc, find the solution to this problem for the three possible cases:

- **underdamped** ($\alpha < \omega$)
- **critically damped** ($\alpha = \omega$)
- **overdamped** ($\alpha > \omega$).

Plot an example of the solution for each case (remember to use Evaluate). (b) Consider the following the following re-write of the equation in terms of a damping parameter $\zeta = \alpha/\omega$:

$$x''[t] + 2\zeta\omega x'[t] + \omega^2 x[t] = 0$$
$$x[0] = x_0 \quad x'[0] = v_0$$

Define a function $hSolve[\zeta, \omega, x_0, v_0, T]$ that plots the solution to this equation from $t = 0$ to $t = T$ given a choice of these 4 parameters. Use Manipulate to display these plots in an interactive way for $\zeta \in [0, 2]$, $\omega \in [1, 4]$, $x_0 \in [0, 2]$ and $v_0 \in [-1, 1]$. Choose the range of T such that you see at least a few oscillations (e.g. 4π). Adjust the range so it is chosen uniformly across examples.

Comment: what do you observe as you transition from the underdamped regime into the critically damped and overdamped?

(c) *Adding a forcing term*: recall we say a system is *driven* if the forcing term (right-hand-side) of the harmonic oscillator ODE is not zero.

Consider the forcing term $F(t) = \sin 2t$. Define a function $drivenhSolve[\zeta, \omega, x0, v0, T]$ that plots the solution to

$$x''[t] + 2\zeta\omega x'[t] + \omega^2 x[t] = \sin 2t$$
$$x[0] = x_0 \quad x'[0] = v_0$$

from $t = 0$ to $t = T$ given a choice of these 4 parameters. Once again, use Manipulate to display these plots in an interactive way, for the same ranges as in (b).

(d) Set $\zeta = 0$ (no damping). Comment on what happens as ω gets close to $\omega = 2$ from the left and from the right. What is special about the case $\omega = 2$?

(e) Now, set $\zeta = 0.1$ to introduce some damping. Vary ω around $\omega = 2$, and compare your results with what you observed in (d). How does damping affect this oscillator?

Choose one of these projects, and make sure to write the corresponding Matlab or Mathematica code, as well as a succinct report on your findings. A set of core instructions is included, which must be completed. Some additional content may be suggested, but is not required.

Option I: Elastic Pendulum in 2D Consider the motion of a pendulum consisting of a spring with constant K and resting length L meters, with a circular mass at the end that weights M kgs. This pendulum hangs from a point (which we label as $(0,0)$).

(A) Ignoring drag / friction, write down the equations of motion for the mass of the pendulum with position $X(t) = (x(t), y(t))$, assuming initial position $X(0) = (x_0, y_0)$ and initial velocity $V(0) = (vx_0, vy_0)$. Recall that there are two forces to consider in Newton's 2nd law:

- $F_{grav}\left(t, \begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 0 \\ -Mg \end{bmatrix}$ with $g = 9.81 m/s^2$.
- $F_{spr}\left(t, \begin{bmatrix} x \\ y \end{bmatrix}\right) = K\left(1 - \frac{L}{\sqrt{x^2+y^2}}\right)\begin{bmatrix} x \\ y \end{bmatrix}$

Given these equations of motion, re-write this system of 2 second order ODEs as 4 first order ODEs (remember: this is achieved by adding velocities $vx(t)$ and $vy(t)$ as auxiliary variables).

(B) Write a Matlab function *elastic_pendulum.m* taking (at least) input parameters M, L, K, initial data vectors $X0$ and $V0$, Nt number of points, final time T and a string called *method* (which determines whether ode45 or ode23 is used). This function should define the function $odefun(t, p)$ for

$$\frac{dp}{dt} = \text{odefun}(p, t) \quad p(t) = \begin{bmatrix} x(t) \\ y(t) \\ vx(t) \\ vy(t) \end{bmatrix} \quad p(0) = \begin{bmatrix} x_0 \\ y_0 \\ vx_0 \\ vy_0 \end{bmatrix}$$

solve it with either ode45 or ode23 (using a switch based on method), and evaluate it on a set of Nt equispaced points from 0 to T. The function should produce 3 outputs:

- xsol $(2 \times Nt + 1)$ array of x and y positions for the pendulum.
- vsol $(2 \times Nt + 1)$ array of vx and vy velocities for the pendulum.
- t $(1 \times Nt + 1)$ vector of times from 0 to T.

(C) Write a Matlab function that calls *elastic_pendulum.m* for a given choice of parameters and uses plot to display the motion of the pendulum in 2D space from $t = 0$ to $t = T$. You can use a big marker for the point $x(t), y(t)$, and should also display the line segment connecting it to the origin $(0,0)$. Remember to add *pause* to display each snapshot with a time delay (e.g. pause(0.1)).

(D) For $M = 1, L = 1$, run your code for increasing values of K, say, $K = [1, 10, 100, 1000]$. Comment on how the motion of the spring pendulum changes as K increases. Also, which method runs better for these cases, ode45 or ode23?

(E) **(Optional)** Add the capability to your Matlab function to (i) save the solution data on a matfile and (ii) save a video of the pendulum motion as an .avi file. Remember to add parameter values to the filename for the video.

(F) **(Optional)** Consider adding the following two features: (i) A friction force in the direction of the spring (proportional to velocity v) (ii) A nonlinear spring force.

Option II: Intro to Numeric PDE

In this project, you can learn the basics of solving PDE numerically. We will be solving the Poisson equation on the square $[0,1]^2$, which typically represents the long-term solution of a diffusion process (e.g. heat equation) or in electrostatics / magnetostatics (Maxwell).

That is, let $T(x,y)$ be the equilibrium temperature at position (x,y) on a square plate, given initial temperature profiles at the 4 sides of the boundary and heat source $Q(x,y)$. Then, it obeys the following PDE boundary value problem:

$$\begin{aligned} T_{xx}(x,y) + T_{yy}(x,y) &= Q(x,y) && \text{for all } (x,y) \in [0,1]^2 \\ T(x,0) = D(x) \quad T(x,1) &= U(x) && \text{for all } x \in [0,1] \\ T(0,y) = L(y) \quad T(1,y) &= R(y) && \text{for all } y \in [0,1] \end{aligned}$$

where $D(x), U(x), L(y), R(y)$ set initial temperature profiles for down, up, left and right sides, respectively. We ask the user to make sure that these conditions match at the vertices,

$$\begin{aligned} D(0) = L(0) &= T(0,0) \\ D(1) = R(0) &= T(1,0) \\ U(0) = L(1) &= T(0,1) \\ U(1) = R(1) &= T(1,1) \end{aligned}$$

Given an input n, we want to find approximate values for $T(x_i, y_j)$ at a uniform grid of $(n+1) \times (n+1)$ points on the square. The final result of this project will be a Matlab function called *Laplace_square.m* that outputs these values as an $n+1 \times n+1$ array.

(A) This exercise forces us to relate indices (i,j) on a square grid with a linear index k going from 1 to $(n+1)^2$. Write a Matlab function (e.g. call it *indexgrid.m*) that, given n, generates the following two outputs:

- Use *ndgrid* to generate a grid of indices (i,j) that each vary from 1 to $n+1$. Then, use reshape or write $i = i(:), j = j(:)$ to transform them into vectors of size $(n+1)^2 + 1$.
- Output the column vector $k = (1 : (n+1)^2)'$, representing an index going through all the points on the grid.

Display the output of this code for $n = 5$. Check that the following formulas hold:

1. $k(i,j) = (n+1)(j-1) + i$
2. $i(k) = mod(k-1, n+1) + 1$
3. $j(k) = (k - i(k))/(n+1) + 1$

Briefly explain, if you can, how one could have derived these formulas.

Discretizing the PDE: We will now use difference formulas to second derivatives T_{xx} and T_{yy} in the Poisson equation. Let the spacing $\Delta x = \Delta y = 1/n$, and $x_i = i\Delta x$, $y_j = j\Delta y$. We have:

$$T_{xx}(x_i, y_j) \simeq \frac{T(x_i + \Delta x, y_j) - 2T(x_i, y_j) + T(x_i - \Delta x, y_j)}{(\Delta x)^2} = n^2(T(x_{i+1}, y_j) - 2T(x_i, y_j) + T(x_{i-1}, y_j))$$
$$T_{yy}(x_i, y_j) \simeq \frac{T(x_i, y_j + \Delta y) - 2T(x_i, y_j) + T(x_i, y_j - \Delta y)}{(\Delta y)^2} = n^2(T(x_i, y_{j+1}) - 2T(x_i, y_j) + T(x_i, y_{j-1}))$$

Notice that for $i = 1, i = n+1, j = 1, j = n+1$ we know the values of T (these correspond to unknowns on the boundary), but we can use these approximations to get a set of linear equations for the unknown values *inside* of the square. Plugging these into the Poisson equation, we have the following system of $(n-1)^2$ linear equations on the unknowns $T_{i,j} = T(x_i, y_j)$, with $i, j \in 2, 3, \ldots, n$:

$$T(x_i, y_{j-1}) + T(x_{i-1}, y_j) - 4T(x_i, y_j) + T(x_{i+1}, y_j) + T(x_i, y_{j+1}) = (1/n^2)Q(x_i, y_j) \quad \text{for all } (i, j)$$
$$T(x_i, y_1) = D(x_i) \quad T(x_i, y_{n+1}) = U(x_i) \qquad \text{for all } i$$
$$T(x_1, y_j) = L(y_j) \quad T(x_{n+1}, y_j) = R(y_j) \qquad \text{for all } j$$

For numerical purposes, we have divided the right-hand-side by n^2. Notice that for a given unknown $T_{i,j}$, only itself and its four neighbors in the cardinal directions appear, that is, $T_{i-1,j}, T_{i+1,j}, T_{i,j-1}, T_{i,j+1}$. That means, if we create a matrix to encode this system as

$$\mathcal{L}T = B$$

, matrix $\mathcal{L}$ is an $(n-1)^2 \times (n-1)^2$ matrix with *at most* 5 non-zero entries per row.

(B) Write a Matlab function *Laplace square.m* with the following inputs: n,D, U, L, R (functions of one parameter) and Q (function of two parameters). Using *either* one for loop with parameter k (going from 1 to $(n-1+^2$ or two for loops with parameters i, j, build the following:

1. $(n-1)^2 \times (n-1)^2$ Matrix $\mathcal{L}$: If you go row by row, note that $L(k,k) = -4$, $L(k,k+1) = 1$, $L(k,k-1) = 1$, $L(k,k+n-1) = 1$, $L(k,k-n+1) = 1$. Explain why that is based on the equations above.
   You can also note this matrix is *pentadiagonal*, meaning it only contains non-zero entries in 5 diagonal bands. For additional challenge, find an equivalent way to generate this matrix using diag or spdiags (this latter one generates a matrix in the *sparse* format).

2. $(n-1)^2 \times 1$ right-hand-side B: Note that $B(k)$ is not just $(1/n^2)Q(x_{i(k)}, y_{j(k)})$. For points neighboring the boundary, you need to subtract boundary data that appear on the 2nd derivative formula. For example, if $i = 2$, then the term $T_{x_1,j} = L(y_j)$ needs to be subtracted from $Q(x_1, y_j)$ on the right-hand-side vector.

(C) Once you have checked that $\mathcal{L}$ and B are correct, solve the linear system using Gaussian elimination (backslash). Reshape T as an $(n-1) \times (n-1)$ array using the function reshape. Finally, generate a second array Tsq of size $(n+1) \times (n+1)$ that includes both the boundary data, and T. Suggestion: Set Tsq to a matrix of all zeroes. Set $Tsq(2:n, 2:n) = T$. Then, use a for loop to add the boundary data.
Make your Matlab function output Tsq.

(D) Write a Matlab script that solves the following two Poisson BVP:

$$T_{xx}(x,y) + T_{yy}(x,y) = 0 \quad \text{for all } (x,y) \in [0,1]^2$$
$$T(x,0) = 100x \quad T(x,1) = 100x^2 \qquad \text{for all } x \in [0,1]$$
$$T(0,y) = 0 \quad T(1,y) = 100 \qquad \text{for all } y \in [0,1]$$

$$T_{xx}(x,y) + T_{yy}(x,y) = 6400\sin(4\pi x)\cos(4\pi y) \quad \text{for all } (x,y) \in [0,1]^2$$
$$T(x,0) = 100x \quad T(x,1) = 100x^2 \qquad \text{for all } x \in [0,1]$$
$$T(0,y) = 0 \quad T(1,y) = 100 \qquad \text{for all } y \in [0,1]$$

and plots the resulting surface using surf (generate the x and y grids using ndgrid). Display the results for $n = 10, 20, 40$. Comment on both similarities and differences between the homogeneous and non-homogeneous solutions.

(E) **(Optional)** The theory tells you that, as $n \to \infty$, this solution converges to the true solution, and the errors between the approximate solution and the true solution to this equation are proportional to $1/n^2$. Can you demonstrate this using a log-log plot and solutions for increasing values of n? Note you do not have to know the true solution to do this; there is a method involving the differences between successive approximations.

Option III: RLC circuits

Consider the standard RLC circuit where a resistor (with resistance R), a capacitor (with capacitance C and an inductor L (with inductance L) are connected in series with a voltage source with voltage $V(t)$. Using Kirchhoff's law, we have

$$V(t) = V_R(t) + V_L(t) + V_C(t)$$

where V_R, V_L, V_C are the voltages associated with the resistor, inductor and capacitor, respectively. Using the relationships between these three and the current $I(t)$, we have:

$$I''[t] + \frac{R}{L}I'[t] + \frac{1}{LC}I[t] = 0$$
$$I'[0] = D_0 \quad I[0] = I_0$$

(A) Relate this back to the general harmonic oscillator on problem 1. Derive formulas for parameters α, ω and ζ in terms of R, L, C. In this context, α is known as the neper frequency and ω as the angular resonance frequency. Based on the discussion of problem 1, briefly explain what these parameters tell us about the current in our RLC circuit.

(B) Write a Matlab function *RLC_Circuit.m* that takes input parameters R, L, C, voltage function $V(t)$, final time initial data $I0$ and $D0$, Nt number of points, final time T and a string called *method* (which determines whether ode45 or ode23 is used). This function should define the function $odefun(t, p)$ for

$$\frac{dp}{dt} = \text{odefun}(p, t) \quad p(t) = \begin{bmatrix} I(t) \\ I'(t) \end{bmatrix} \quad p(0) = \begin{bmatrix} I_0 \\ D_0 \end{bmatrix}$$

solve it with either ode45 or ode23 (using a switch based on method), and evaluate it on a set of Nt equispaced points from 0 to T. The function should produce 3 outputs:

- Isol $(1 \times Nt + 1)$ array of current I values for the circuit.
- Dsol $(1 \times Nt + 1)$ array of rate of change of current for the circuit.
- t $(1 \times Nt + 1)$ vector of times from 0 to T.

(C) Write a Matlab function that calls *RLC_Circuit.m* for a given choice of parameters and uses plot to display the current $I(t)$ from $t = 0$ to $t = T$. Use subplot to show two sets of 3 results for different damping regimes (given by *zeta*), one with zero source voltage $V(t) = 0$, one with voltage $V(t) = \cos(\omega t)$. Comment on what you observe in both cases.

(D) For $R = 1, L = 1$, run your code for decreasing values of C, say, $C = [1, 0.1, 0.01, 0.001]$. Comment on how the current function changes. Also, which method runs better for these cases, ode45 or ode23?

(E) **Variable capacitance:** Write a separate function called *RLC_Circuit_VarC.m* where C is a function handle instead of a constant. Use a version of your script from question D to run this code for $R = 1, L = 1, C = 1 - 0.5\sin(t)$. for both zero and $V(t) = \cos(\omega t)$ voltage sources. Comment on how a numerical ODE solution is useful for this extended case.

(F) **(Optional)** Consider what modifications would be involved in (i) adding variable resistance and inductance functions, (ii) allowing for circuits in parallel and more general topologies.