ENGG1003 - Lab Week 10

Brenton Schulz

Task 1: Scalar Arithmetic and Scripts

Write a MATLAB script which calculates the energy stored in a spring stretched (or compressed) by a distance x, with spring constant k. The equation is:

$$E = \frac{1}{2}kx^2$$

Ensure that the variables are initialised separately so that the script can easily be re-used.

NB: The same script can easily be adapted to calculate kinetic energy ($\frac{1}{2}mv^2$), capacitor energy ($\frac{1}{2}CV^2$), inductor energy ($\frac{1}{2}LI^2$), and possibly other values I can't think of right now.

Task 2: Projectile Motion - Vector Arithmetic and Plots

Write a MATLAB script which plots the path of a particle undergoing projectile motion given its initial velocity. The velocity is specified as a speed, v_0, and angle from the horizon, θ. Call your script `projectile.m`.

As the particle moves, the horizontal, x, and vertical, y, displacements as a function of time, t, can be calculated as:

$$x = v_0 t \cos(\theta)$$
$$y = v_0 t \sin(\theta) - \frac{1}{2}gt^2$$

where g is acceleration due to gravity. If we choose positive x to be "upwards" then g, in SI units, is $-9.8\ m/s^2$.

Your code should declare a time vector which is long enough to plot the particle's path until it returns to $y = 0$. This is achieved by declaring t from 0 to:

$$t = \frac{2v_0 \sin(\theta)}{g}$$

To keep the output plot reasonably smooth, declare t with a few hundred to a thousand points. You may use the `linspace()` function or `start:interval:end` syntax to declare t. The exact value of the final element in t is not crucial.

Task 3: Image Scaling - 3D Array Indexing

Write a MATLAB script which reads an image file with `imread()`, uses matrix indexing to scale the image down by a factor of 1/10th, then displays the result with `image()`.

You may use any image of your choosing. If in doubt, the image used in lectures is available on Blackboard.

Task 4: Image Blur

Write a MATLAB script which blurs an image by making each pixel's colour the mean of the pixels around it. For this task, the mean of several pixels should be calculated separately for each of the red, green, and blue channels.

The "surrounding" pixels may be any one of the following definitions:

- The mean of a pixel and those directly above, below, left, and right of it
- The mean of the 8 pixels directly adjacent
- The mean of the 24 pixels in a box of "radius" 2 surrounding the central pixel
- The *weighted* mean of the 24 pixels defined above where the central pixel has a weighting of 1/2, the "radius 1" box has a weighting of 1/4 and the "radius 2" box a weighting of 1/8th. For this calculation the weighted sum of all pixels should be divided by $1/2+8\times1/4+16\times1/8 = 4.5$ to avoid changing the image brightness.

You are encouraged to attempt multiple definitions and compare the results.

Write your first draft of the script using as many `for` loop structures as you feel comfortable with. Once the code works, have a think about how parts of it could be vectorised, perhaps with complicated array indexing.