MATLAB

MATLAB is a computational software tool used by a variety of professions in order to model and solve various systems. Examples of these professions are, but not limited to, structural engineers, chemical engineers, controls engineers, neuroscientists, fluids engineers, and computational engineers. MATLAB is one of the best tools for solving different types of problems in the order of fractions of a second. Examples in your own undergraduate curriculum where MATLAB is used include:

- MIE 124
- MIE 201
- MIE 273
- MIE 302
- MIE 310
- MIE 340
- MIE 344
- MIE 354
- MIE 402
- MIE 415

To download MATLAB, please go to the MathWorks website: https://bit.ly/2JzfCdN.

To get introduced to MATLAB, here are a couple free online open courses you can use:

- https://bit.ly/32Y87Ca [1]
  - This course, which is developed and run by MITOpenCourseWare, provides an introduction to programming as a whole, the basics, using the command window in MATLAB, vectors, matrices, different types of loops, and also debugging in MATLAB.
- https://bit.ly/2VhTgjk or https://bit.ly/2JN4B6B
  - This course, developed by MathWorks, the developer of MATLAB, provides an opportunity for onboarding in MATLAB

In MIE, there are a few built-in MATLAB functions that we use more so than others. These functions, and a link to their help guides, are:

- figure() (https://bit.ly/31OH4Jr)
  - This command creates a new stand-alone plot in MATLAB that can then include any and all data you provide underneath.

---

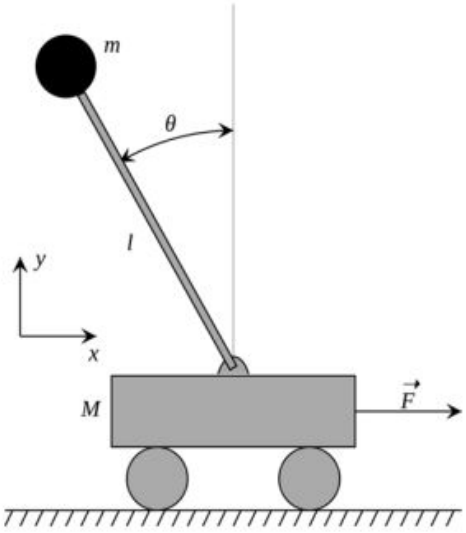
[1] Yossi Farjoun. *18.S997 Introduction To MATLAB Programming.* Fall 2011. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

- Plotting (https://bit.ly/30N4iOA)
  - This link provides a detailed list of everything you can do to make and improve your plots.
- textscan() (https://bit.ly/35jrV50)
  - Reading data or information from a text-file and converting it to a MATLAB array / matrix.
- readtable() (https://bit.ly/2OkhE2l)
  - This command reads and imports various different types of data files, including Excel files (.xls) and Comma Separated Values files (.csv).
- ode45() (https://bit.ly/2oXbz1i)
  - This command is useful for solving relatively simple differential equations.
- bode() (https://bit.ly/2VeSxzm)
  - This function automatically generates a bode plot (or provides the data for the magnitude and phase) of a frequency response analysis.
- MATLAB Answers (https://bit.ly/2oPh95D)
  - In this link, you can find access to problems that other people are running into and their solutions and you can ask your own questions as well. Similar to Stack Overflow.

Examples:

1)
The below example gives you an excellent introduction to completing iterative loops, working with vectors on large scales, and plotting various functions against each other.

HW 8
Inverted Pendulum on a Cart



The *Inverted Pendulum on a Cart* is a classic problem in dynamics and control theory. The inverted pendulum as seen in the diagram above is a pendulum with its center of mass is above the pivot point, with two degrees of freedom, $\boldsymbol{x}$ and $\boldsymbol{\theta}$. This is an unstable system, and so in order for the pendulum to be stable it needs help from the cart that it's mounted on. Imagine that this is like balancing an upturned broomstick on the end of your finger.

The goal of this problem is to command $\boldsymbol{F}$ at every point in time to keep the pendulum vertical (i.e. $\boldsymbol{\theta = 0}$). The parameters are $\boldsymbol{M}$ *(Mass of Cart),* $\boldsymbol{m}$ *(Mass of point at end of rod),* $\boldsymbol{l}$ *(Length of massless rod),* $\boldsymbol{F}$ *(Force on cart),* $\boldsymbol{\theta_0}$ *(Initial theta),* and $\boldsymbol{x}$ *(Position of cart).*

To describe the motion of the *Inverted Pendulum on a Cart* you will need the *Equations of Motion.* Using Lagrange's Equations, we get the two equations below. Note that a "dot" represents a derivative with respect to time, and so two dots is the second derivative with respect to time.

$$(\boldsymbol{M+m})\ddot{x} - \boldsymbol{ml\ddot{\theta}cos\theta + ml\dot{\theta}^2 sin\theta = F}$$

$$\boldsymbol{l\ddot{\theta} - gsin\theta = \ddot{x}cos\theta}$$

To solve this problem, we will need to use a Proportional Integral Controller (PI), where

$$F = k_p \times \theta + k_i \times \int \theta dt$$

You will choose the two gains, k_p and k_i, to keep the pendulum vertical. As you'll see below, the integral of theta will only be taken over a small window of time, not the entire time.

Function

Your inputs (all scalars) will be: ***M, m, L,*** k_p, k_i, θ_0 (initial value), ***noise, dt,*** F_{max}
And your outputs will be: ***x, θ, F, t*** (all vectors), ***work*** (a scalar), T_settle (a scalar)

First Define Parameters:

- $g = 9.81\ m/s^2$
- ***t*** (time vector), starts at 0, with time step ***dt***, until t_{final} , which is 100 seconds.
- initialize the ***x***, $\dot{x}$, $\ddot{x}$, θ, $\dot{\theta}$, and $\ddot{\theta}$ and ***F*** vectors as zeros, all the same size as ***t*** (time). Set the first value of the θ vector to θ_0
- Note θ has to be in radians

Using a for loop which will loop over time. In each time step:

- Calculate ***F*** using $F = k_p \times \theta + k_i \times \int \theta dt$
    - θ is current value of θ, $\int \theta\, dt$ is the integral of theta. For this problem, you will integrate theta over the most recent 0.2 seconds. Initially, if the total elapsed time is less than 0.2 seconds, just use the total time until 0.2 seconds is reached. You can use trapz or write your own numerical integrator.
    - Add random noise to ***F***: $F = F \times (1 + noise \times randn(1))$
    - If ***abs(F)*** is bigger than F_{max}, set it equal to $+/-F_{max}$ depending on the sign on ***F***.
- Calculate the current value of $\ddot{x}$ based on $\ddot{\theta}$, $\dot{\theta}$, θ, and ***F***
- Calculate the current value of $\ddot{\theta}$ based on current values of θ and $\ddot{x}$
  Calculate the next value of $\dot{x}$ and $\dot{\theta}$ using forward Euler. So, if this is the jth time, calculate the values at j+1. So $\dot{x}(j+1) = \dot{x}(j) + \ddot{x}(j) \cdot dt$ etc.
- Calculate the next value (j+1) of ***x*** and θ using forward Euler.
  So $x(j+1) = x(j) + \dot{x}(j) \cdot dt$ etc.

- If $\theta < -\pi/2$ or $\theta > \pi/2$, then break the loop (this is when the pendulum is below the horizontal), and set all values of x and theta after this to nan.
- Calculate $\boldsymbol{work} = \int \boldsymbol{abs}(\boldsymbol{F} \times \dot{\boldsymbol{x}})$ (Can use trapz())
- Calculate T_settle in the cases when the simulation finished i.e. when the pendulum does not pass below the horizontal, otherwise set it to nan. T_settle is the time after which the value of theta never exceeds 5% of the initial value of theta.

Script

1. Define initial parameters, something like $M=20\ kg$, $m=1\ kg$, $L=1$

2. Define dt as 0.01s

3. Run the function for $noise=0$, $F_{max} = 10^{10}$ (some huge number), and $\theta_0 = \pi/10$. Adjust k_p and k_i until you get "good" performance. How did you determine what good is? Hint, one of the gains will be positive, the other is negative!
    a. Justify why you chose the k_p and k_i that you did.
    b. Plot θ *vs time*, x *vs time*, and F *vs time* on one plot using 3 subplots and see what it looks like. Comment in the script.

4. Now run the code for increasing values of θ_0.
    a. Comment on the performance. When are you unable to stabilize the pendulum?
    b. Plot θ *vs time*, x *vs time*, and F *vs time* again for the largest value of θ_0 that you can make work and comment in your script.

5. Now introduce increasing value of noise for $\theta_0 = \pi/6$.
    a. How does noise affect performance? When does the system become unstable?
    b. Plot θ *vs time*, x *vs time*, and F *vs time* again and comment in your script.

6. Finally, decrease F_{max} for $\theta_0 = \pi/6$ and no noise until the system becomes unstable.
    a. How does the performance change?
    b. Plot θ *vs time*, x *vs time*, and F *vs time* again and comment in your script.

2) The following problem will give you a better understanding of operating loops in MATLAB and also using MATLAB as a computational tool for deconstructing further complicated problems.

This problem will deal with prime factorization. Any integer larger than 1 can be decomposed into a product of prime numbers. For example: 6 = 2*3 12 = 2*2*3 30 = 2*3*5 105 = 3*5*7. Prime factorization is actually a fundamental part of modern cryptography and code breaking. One technique for encryption is to use a very large number (128 bit, i.e. 2^128 possible combinations) to encrypt a message. This large number is the public key that everyone knows. This number is also the product of two large prime numbers, which are the private key. Once the message has been encrypted, only people who know the two prime numbers can decode the message. Since finding the prime factors of very large numbers requires substantial computation, this is a very secure way to encrypt information. For more details, see: http://en.wikipedia.org/wiki/RSA_(algorithm).

Your task is to write a function that determines the prime factors of an integer. Your function should have the following characteristics:

- The input is a single integer "N".
- The output is a vector "prime" that contains each of the prime factors of N.
- So your syntax could look something like:

```
function prime = mal_hw4(N)
```

- You may not use the "isprime", "primes", or "factor" functions.
- Useful functions may include "mod", "rem", and while or for loops.

Speed is key. This assignment will be a race against my code. Here is how the race will work. We will run your code 500 times, and track the time it takes to determine the prime factors of a large number that we choose. You can test your code using the following script (which we will run to test your code):

```
N = %(**INSERT NUMBER
HERE**)%;

% pick any number here N_sim = 500;

t = zeros(N_sim,1);

for j=1:N_sim
```

```
        tic;

        prime = mal_hw4(N);% put your function name here t(j,1) = toc;
end
T_avg = mean(t(2:end));
```

Just save this code as a script (m-file) with any name you'd like, replace "mal_hw4" with the name of your function, and enter any number you want for N. Once you have a function that works, run this script and check your T_avg. To give you a sense of how my code performs, here are a few examples:

N = 982566, prime = [2 3 3 13 13 17 19], T_avg = 8.0828*10^-6 s.

N = 102357894, prime = [2 3 41 416089], T_avg = 2.855*10^-5 s.

N = 544897453362, prime = [2 3 90816242227], T_avg = 0.0152 s.

N = 74489552332212, prime = [2 2 3 652607 9511793], T_avg =0.0397 s.

Notice we have discovered some pretty big prime numbers (90816242227!) by just entering in a random value of N. Your performance will depend on the computer you run on, but also on how efficient your code is.