Matrix Library

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #15

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Introduction:

Python is similar to Matlab, but...

Python and Matlab handle matricies differently

- Matlab is a matrix language designed for scientists and engineers
- Python is a language designed for the general public

In Python, arrays are treated like strings, not matrices.

Matlab

```
>> A = [1,2,3];

>> B = 2*A
B =
     2   4   6

>> C = A + 2
C =
     3   4   5
```

Python

```
>>> A = [1,2,3]

>>> B = 2*A

>>> print(B)
[[1, 2, 3],[1, 2, 3]]

>>> C = A+2
TypeError: unsipported types for
__add__
```

Problem: How do you use matricies in Python?

Matrices make some problems *much* easier to solve.

Python 3 solution:

- Use the library NumPi
- Not available for MicroPython

MicroPython solution:

- Create your own NumPi library
  - Matrix library
  - This lecture
- Requires some coding

```
>>> import matrix

>>> A = [[1,2,3],[4,5,6]]

>>> matrix.display(A)
     1.000     2.000     3.000
     4.000     5.000     6.000

>>> N = matrix.size(A)

>>> print(N)
[2, 3]

>>> AT = matrix.transpose(A)
>>> matrix.display(AT)
     1.000     4.000
     2.000     5.000
     3.000     6.000
```

In this lecture, we'll create matrix functions for the following operations:

- Display(A) display an nxm matrix with a formatted output
- Zeros(n,m) create an nxm matrix containing all zeros
- Ones(n,m) create an nxm matrix containing all ones
- Eye(n,n) create an nxn matrix with ones on the diagonal (identity matrix)
- Random(n,m) create an nxm matrix with random numbers in the range of (0,1)
- Transpose(A) return the transpose of A
- Add(A,B) return the sum of matrix A + B
- Multiply(A,B) return the result of AB
- Multk(A,k) return the scalar multiplication kA
- Inverse(A) return the matrix inverse of A

Defining a 2-dimensional matrix in Python

Start with defining a matrix.

- Example: A = 2x3 matrix
- Two sets of brackets needed
  - One for rows
  - One for columns

Each element can be addressed

- Counting starts at zero
- 1st term = row
- 2nd term = column

Row and column matricies are different

- Each needs two sets of brackets
- Rows and columns

```
>>> A = [[1,2,3],[4,5,6]]

>>> A
[[1, 2, 3], [4, 5, 6]]

>> A[0][2]
3

>>> # Row matrix
>>> A = [[1,2,3]]
>>> A[0][1]
2

>>> # column matrix
>>> A = [[1],[2],[3]]
>>> A[1][0]
2
```

Matrix Functions: Display & Size

Import the matrix library

- gives access to its routines
- needs to be saved on your Pico board

display(A):

- Display a matrix
- Fixed decimal format
  - makes the display prettier

size(A):

- Return the dimensions of A
- Returned as a 1x2 array
  - [Rows, Columns]

```
>>> import matrix

>>> A = [[1,2,3],[4,5,6]]

>>> matrix.display(A)
     1.000     2.000     3.000
     4.000     5.000     6.000

>>> N = matrix.size(A)

>>> print(N)
[2, 3]
```

Transpose(A)

- Return the transpose of matrix A

mult_k(A, k)

- Scalar multiplication
- Result is k * A
  - Each element is multiplied by k

zeros(n,m)

- Returns an nxm matrix
- All terms are zero

eye(n,m)

- Returns the identity matrix
- n should equal m

```
>>> AT = matrix.transpose(A)
>>> matrix.display(AT)
     1.000     4.000
     2.000     5.000
     3.000     6.000


>>> B = matrix.mult_k(A, 2.3)
>>> matrix.display(B)
     2.300     4.600     6.900
     9.200    11.500    13.800


>>> A = matrix.zeros(2,3)
>>> matrix.display(A)
     0.000     0.000     0.000
     0.000     0.000     0.000


>>> I = matrix.eye(3,3);
>>> matrix.display(I)
     1.000     0.000     0.000
     0.000     1.000     0.000
     0.000     0.000     1.000
```

rand(n,m)

- Returns an nxm matrix
- Each element is a random number
- Range = (0,1)

add(A, B)

- Returns A + B
- A and B must have the same dimensions

inv(A)

- Returns the inverse of A
  - Uses Gauss elimination
- A must be NxN

mult(A, B)

- Returns A*B
- Inner dimensions must match

```
>>> A = matrix.rand(3,3)
>>> matrix.display(A)
     0.810     0.540     0.170
     0.222     0.291     0.498
     0.696     0.200     0.510

>>> C = matrix.add(A,I)
>>> matrix.display(C)
     1.810     0.540     0.170
     0.222     1.291     0.498
     0.696     0.200     1.510

>>> AI = matrix.inv(A)
>>> matrix.display(AI)
     0.352    -1.771     1.582
     1.683     2.126    -2.636
    -1.140     1.542     0.835

>>> C = matrix.mult(A, AI)
>>> matrix.display(C)
     1.000    -0.000     0.000
    -0.000     1.000     0.000
     0.000    -0.000     1.000
```

add_k(A, k)

- Add a scalar to a matrix
- Same as Matlab command A + k

```
>>> A = matrix.zeros(3,3)
>>> B = add_k(A, 1)
>>> matrix.display(B)
     1.000     1.000     1.000
     1.000     1.000     1.000
     1.000     1.000     1.000
```

linespace(x0, dx, x1)

- Create a vector
- Starting at x0
- Ending at x1
- Step size dx

```
>>> t = linspace(-1,0.5,1)
>>> matrix.display([t])
    -1.000  -0.500  0.000  0.500 1.000
```

logspace(a, b, n)

- Create a 1xn vector
- Starting at 10**a
- Ending at 10**b

```
>>> w = logspace(-1,1,5)
>>> matrix.display(w)
0.100 0.316  1.000  3.162  10.000
```

power(A, k)

- Raise each element of A to the kth power
- Same as Matlab command A.^k

append(A, B)

- Append B to the right of A
- Create a matrix [A, B]

```
>>> A = linspace(0,1,4)
>>> A = [A]
>>> B = matrix.power(A, 2)
>>> matrix.display(A)
0.000  1.000  4.000  9.000 16.000


>>> t = linspace(0, 1, 4)
>>> t = matrix.transpose([t])
>>> t0 = matrix.power(t, 0)
>>> t2 = matrix.power(t, 2)
>>> B = matrix.append(t2,t)
>>> B = marix.append(B, t0)
>>> matrix.display(B)
  0.000  0.000  1.000
  1.000  1.000  1.000
  4.000  2.000  1.000
  9.000  3.000  1.000
 16.000  4.000  1.000
```

Convolution & Polynomials

Convolution is used to multiply polynomials

Example:

$(2 + 3x + 4x^2)(5 + 6x)$

Answer

$10 + 27x + 38x^2 + 24x^3$

```
>>> import matrix

>>> A = [2,3,4]
>>> B = [5,6]
>>> C = matrix.conv(A,B)
>>> C
[10, 27, 38, 24]
```

Convolution & pdf's

Convolution is also used to combine pdf's

Example: *Flaming Sphere*

- D&D Spell
- Find the odds of doing N damage
- N = the sum of two 6-sided dice
- N = 2d6

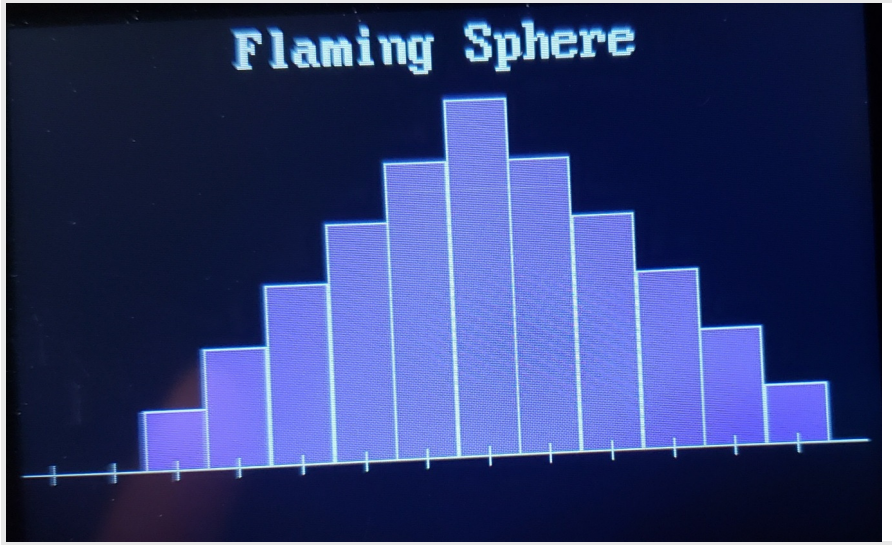
Note:

- N has the odds of doing {0,1, 2...}
- Same data in a bar chart is easier to see

```
import matrix
import LCD

p = 1/6
d6 = [0,p,p,p,p,p,p]
N  = matrix.conv(d6,d6)
matrix.display([N])
0.000  0.000  0.028  0.056  0.083..

Navy = LCD.RGB(0,0,5)
White = LCD.RGB(150,150,150)
LtBlue = LCD.RGB(50,50,150)
LCD.Init()
LCD.Clear(Navy)
LCD.Bar(N, White, LtBlue)
LCD.Title('Flaming Sphere')
```



Shatter:

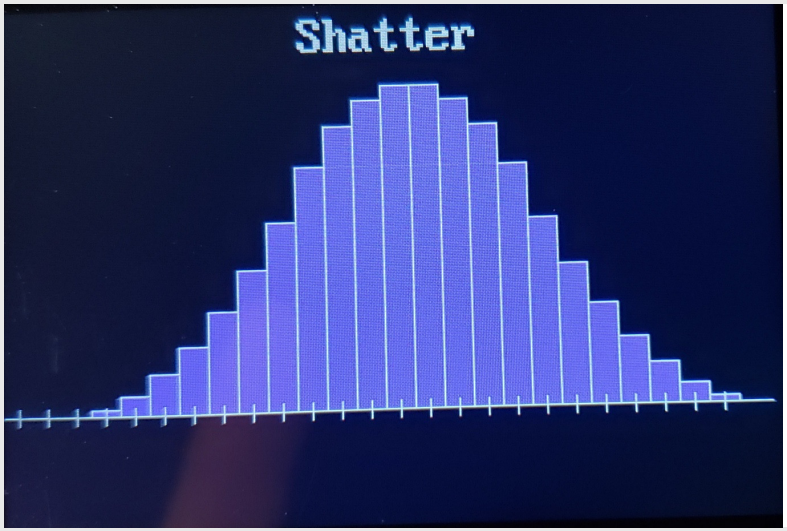
- D&D 2nd-level spell
- Damage = 3d8
  - Convolve the pdf for a d8
  - Three times

```
import matrix
import LCD
p = 1/8
d8 = [0,p,p,p,p,p,p,p,p]
d8x2 = matrix.conv(d8,d8)
d8x3 = matrix.conv(d8x2,d8)

Navy = LCD.RGB(0,0,5)
White = LCD.RGB(150,150,150)
Red = LCD.RGB(150,0,0)
LtBlue = LCD.RGB(50,50,150)

LCD.Init()
LCD.Clear(Navy)

matrix.BarChart(d8x3, White, LtBlue)
matrix.Title('Shatter',White, Navy)
```



Note:

- The pdf is a bell-shaped curve
- Central Limit Theorem

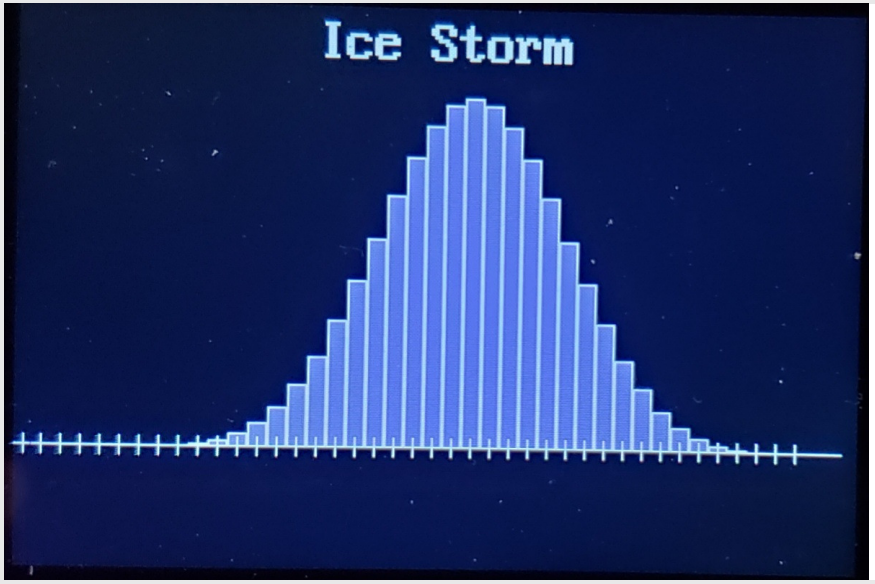
Ice Storm

- D&D 4th level spell
- Does 4d6 + 2d8
  - Convolve d6 four times
  - Convolve d8 twice
  - Convolve the result

```
import matrix
import LCD
p = 1/8
d8 = [0,p,p,p,p,p,p,p,p]
d8x2 = matrix.conv(d8,d8)
p = 1/6
d6 = [0,p,p,p,p,p,p]
d6x2 = matrix.conv(d6,d6)
d6x4 = matrix.conv(d6x2,d6x2)
IceStorm = matrix.conv(d8x2, d6x4)
Navy = LCD.RGB(0,0,5)
White = LCD.RGB(150,150,150)
LtBlue = LCD.RGB(50,50,150)
LCD.Init()
LCD.Clear(Navy)
matrix.BarChart(d8x3, White, LtBlue)
matrix.Title('Shatter',White, Navy)
```

Note:

- Bell-shaped curve shows up again



Matrix Library Application

- Least Squares Curve Fitting

Given a set of data (x, y), determine the least squares curve fit

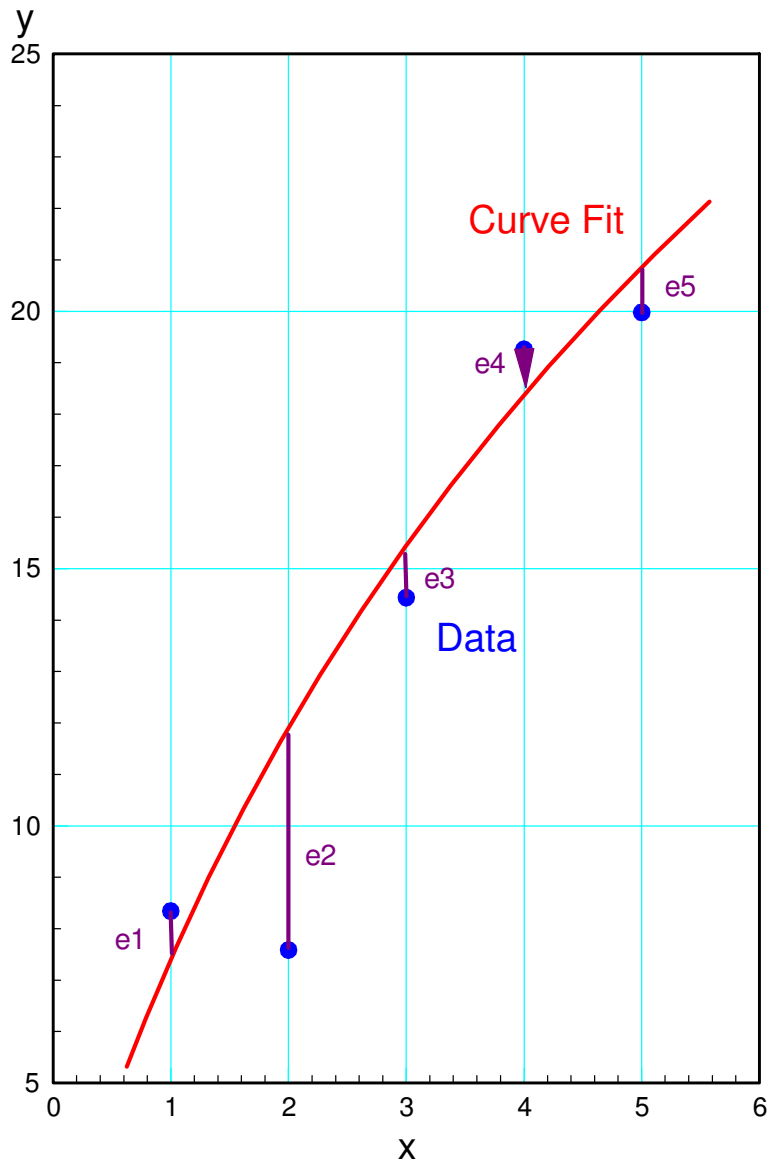
$$y \approx ax^2 + bx + c$$

This problem is easier to solve if you place it in matrix form

$$Y = \begin{bmatrix} x^2 & x & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = BA$$

The least-squares solution for A is

$$A = (B^T B)^{-1} B^T Y$$



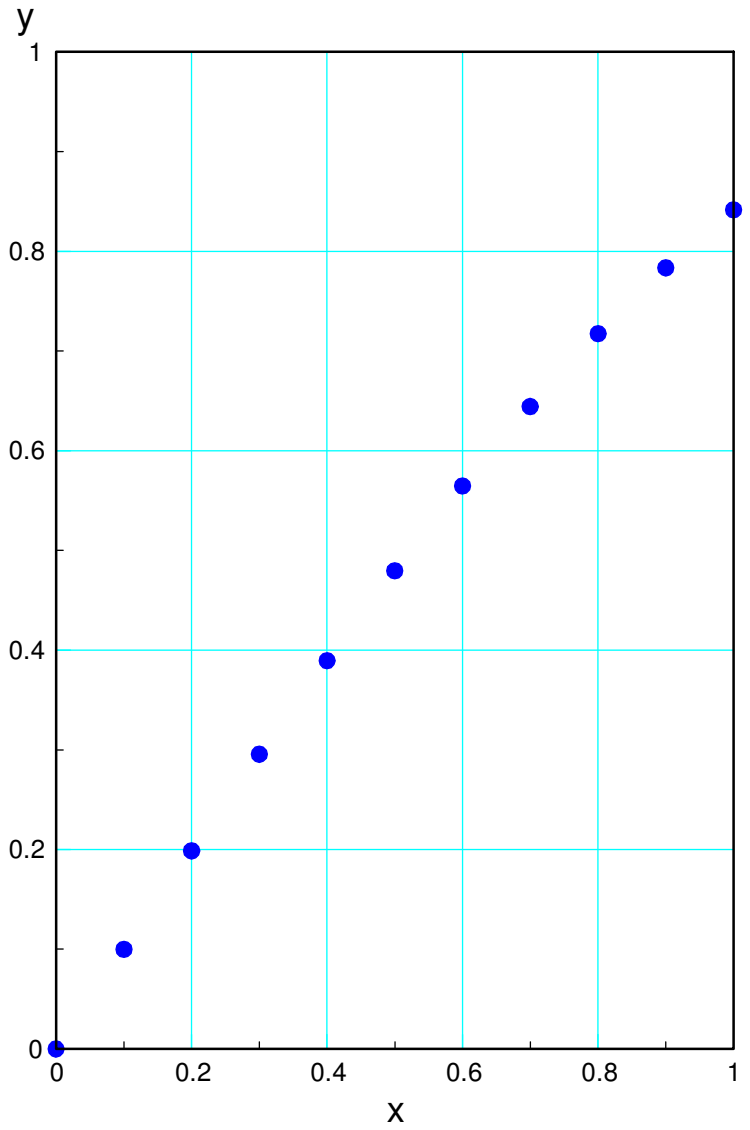
For example, determine a parabolic approximation for sin(x) over the range of (0,1)

$$\sin(x) \approx ax^2 + bx + c$$

First, define the basis function, B:

$$B = \begin{bmatrix} x^2 & x & 1 \end{bmatrix}$$

Then compute A using matrix operations.



Least Squares Example

Approximate y = sin(x)

- Range: (0 < x < 1.6)
- Using least squares
- $y = ax^2 + bx + c$

Basis funciton:

- $B = [x^2, x, 1]$

Least Squares Solution

- A = [a, b, c]
- $A = inv(B^TB)*B^TY$

$\sin(x) = -0.326\ x^2 + 1.175\ x - 0.017$

```
import matrix
import LCD
import math
import random

t = matrix.linspace(0,0.1,1.6
)
t = matrix.transpose([t])
n = len(t)

Y = matrix.zeros(n, 1)
for i in range(0,n):
    Y[i][0] = math.sin(t[i][0])

t2 = matrix.power(t,2)
t0 = matrix.power(t,0)
B = matrix.append(t2,t)
B = matrix.append(B,t0)

Bt = matrix.transpose(B)
BtB = matrix.mult(Bt,B)
BtBi = matrix.inv(BtB)
BtY = matrix.mult(Bt,Y)
A = matrix.mult(BtBi,BtY)
matrix.display(A)
```

Shell

```
-0.326
 1.175
-0.017
```

Again, the results are easier to see in a plot

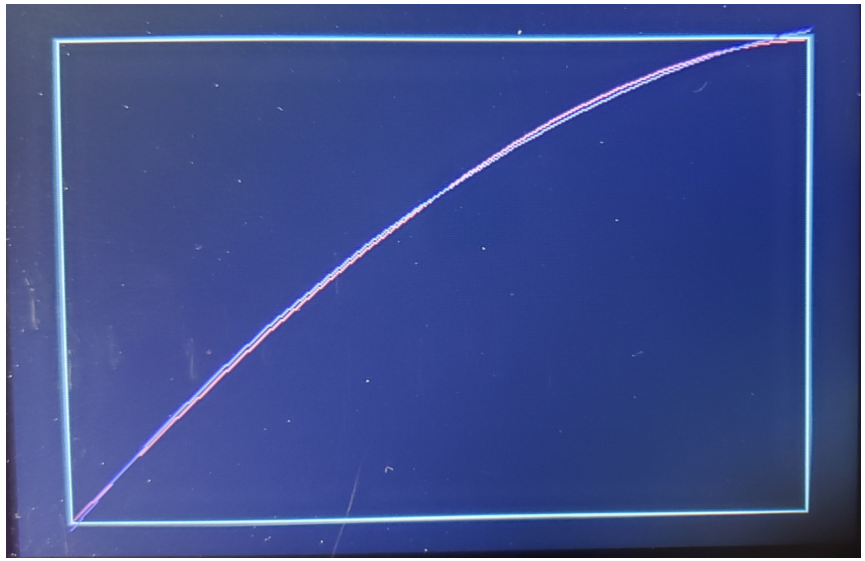
- Plot t vs. Y and t vs B*A
  - Curve fit: Yest = B*A

```
Navy = LCD.RGB(0,0,5)

LCD.Init()
LCD.Clear(Navy)

BA = matrix.mult(B, A)
Data = matrix.append(Y, BA)
matrix.Plot(t,Data)
```

Where least squares really shines is when you're collecting data and want to curve fit your actual data. More on this later when we cover data collection and recursive least squares.

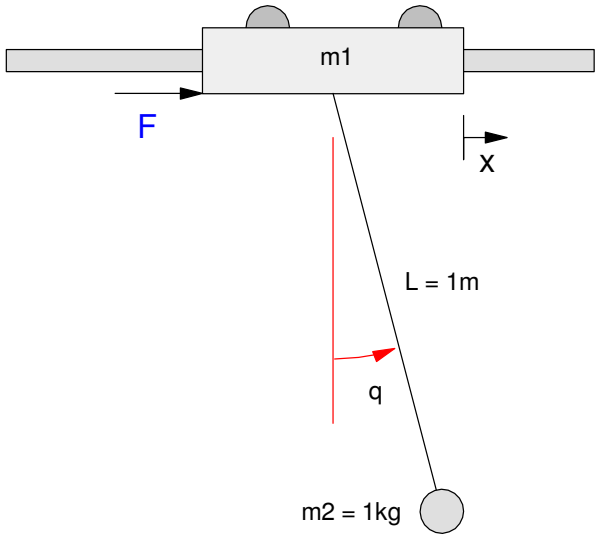


Use of Matricies: Dynamic Simulations

- Gantry System

Finally, simulate the dynamics of a gantry system

- A force is applied to a mass (m1)
- This pushes the mass left and right
- Attached to the mass is a 1m string, connected to a 1kg load (m2).



This models something you find in shop floors: an overhead gantry system is used to lift and move heavy objects, such as an engine block, across the shop floor.

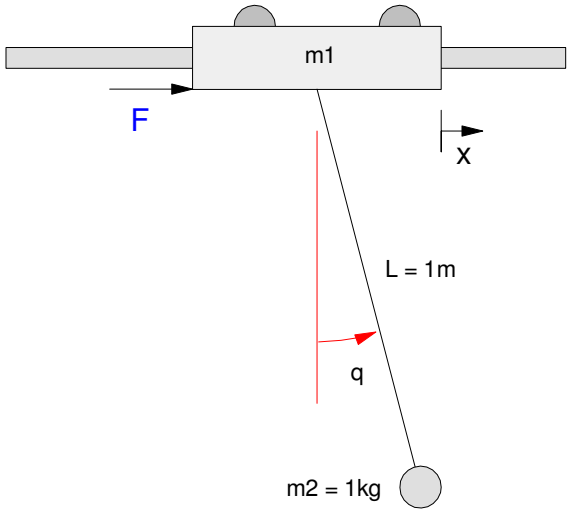
The goal of this system is to move the grantry while avoiding swinging motion of the load

Gantry Dynamics

To simulate, you need the dynamics.

- Lecture #7 for ECE 463 Modern Control

$$\begin{bmatrix} 3 & \cos\theta \\ \cos\theta & 1 \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta}^2 \sin\theta \\ -g\sin\theta \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} F$$



where

- x is the position of the cart
- θ is the angle of the beam
- F is the force on the base, and
- g is the acceleration due to gravity.

Translation

- If you know the force, the angles, and the position, you can compute the acceleration
- Integrate once and you get velocity
- Integrate again and you have position

Python Program (part 1)

$$\begin{bmatrix} 3 & \cos\theta \\ \cos\theta & 1 \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta}^2 \sin\theta \\ -g\sin\theta \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} F$$

$$M\ddot{X} = C(\theta, \dot{\theta}) + F$$

Define the system state

- X[0] = position (x)
- X[1] = angle (q)
- X[2] = velocity (dx/dt)
- X[3] = angular velocity (dq/dt)

Define the mass matrix

Define the coriolis forces

Compute the acceleration

- Uses a matrix inverse, multiply, and add

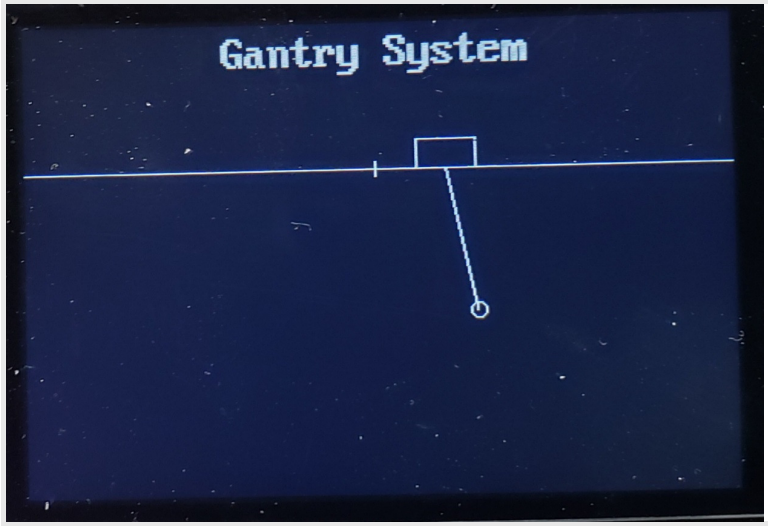
```
def GantryDynamics(X, U):
    x = X[0]
    q = X[1]
    dx = X[2]
    dq = X[3]
    g  = 9.8
    M = [[3, cos(q)],[cos(q),1]]
    C = [[dq*dq*sin(q)],[-g*sin(q)]]
    F = [[U],[0]]
    Mi = matrix.inv(M)
    MC = matrix.mult(Mi,C)
    MF = matrix.mult(Mi,F)
    ddX = matrix.add(MC,MF)
    dX = [dx,dq,ddX[0][0],ddX[1][0]]
    return(dX)
```

Python Program (part 2)

Once you know the position and angle, display the gantry system on the LCD

- Compute the (x,y) location of the cart (m1)
- Compute the (x,y) location of the load (m2)
- Draw the cart and load

```
def GantryDisplay(X, Color):
    x = X[0]
    q = X[1]
    x0 = 240 + x*100
    y0 = 100
    x1 = x0 + 100*sin(q)
    y1 = y0 + 100*cos(q)
    Navy = LCD.RGB(0,0,5)
    White = LCD.RGB(150,150,150)
    LCD.Line(0,y0,479,y0,White)
    LCD.Line(240,y0-5,240,y0+5,White)
    LCD.Box(x0-20,y0,x0+20,y0-20,Color)
    LCD.Line(x0,y0,x1,y1,Color)
    LCD.Circle(x1,y1,5,Color)
```



Python Program (part 3)

Numerical Integration

- Euler Integration

```
def Integrate(X, dX, dt):
    Y = [0,0,0,0]
    for i in range(0,4):
        Y[i] = X[i] + dX[i] * dt
    return(Y)
```

Integration = Area under the curve

Euler Integration:

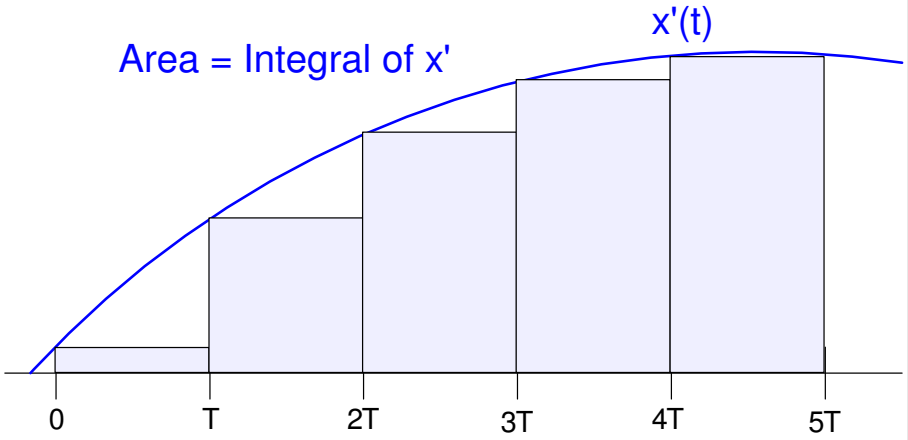
- Approximate with rectangles

Simple:

- Doesn't need old data

Not that accurate

- Other forms of integration exist



Python Program: Main Routine

Initialize everything

- Include subroutines for
  - Dynamics
  - Display
  - Integration
- Find the initial position of the joystick
  - Interprit as zero input
- Initialize the LCD display

```
# Gantry
import matrix
import LCD
import math
import time
import machine

def GantryDynamics:
# insert code here
def GantryDisplay:
# insert code here
def Integrate:
# insert code here

a2d0 = machine.ADC(0)
F0 = a2d0.read_u16()

White = LCD.RGB(150,150,150)
Navy = LCD.RGB(0,0,5)

LCD.Init()
LCD.Clear(Navy)
```

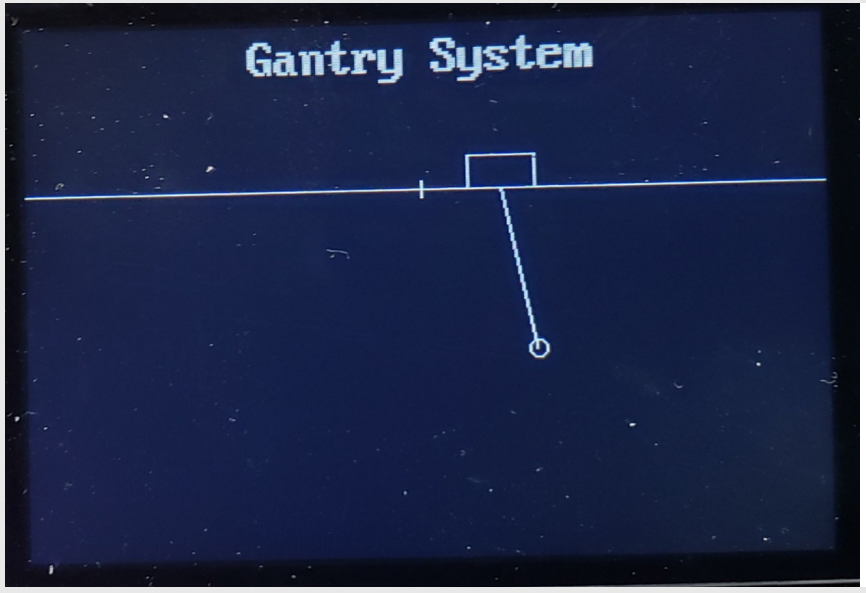
Python Program: Main Loop

- Set the initial position at x = -2m
- Set the sampling rate at 20ms
- Run for 10 seconds
  - Read the joystick for force input
  - Compute the acceleration
  - Clear the old display
  - Integrate, and
  - Display the new position

Similar techniques can be used to model other dynamic systems such as a heat equation, inverted pendulum, double pendulum, ball and beam systems, etc.

```
k = 50 / 32000

X = [-2,0,0,0]
dt = 0.02
t = 0
while(t < 10):
    F = (a2d0.read_u16()-F0)*k
    dX = GantryDynamics(X, F)
    GantryDisplay(X,Navy)
    X = Integrate(X, dX, dt)
    t = t + dt
    GantryDisplay(X,White)
    time.sleep(dt)
```



Summary:

Matrices are really useful:

- Some problems are much easier to solve with them
- Example: least squares curve fitting

Python is *not* a matrix language

- By writing your own matrix routines, it's workable
- Place these in a library and you can use them with future programs

The result isn't as user friendly as Matlab, but it works.

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/