Matrices and Least Square Fit

FOSSEE

1 Matrices

1.1 Basics

Matrix Creation

```
In []: C = array([[1,1,2], [2,4,1], [-1,3,7]])
```

It creates C matrix of shape 3x3
Shape is dimensions of given array.

```
In []: C.shape
Out[]: (3, 3)
In []: shape([[1,2],[4,5],[3,0]])
Out[]: (3, 2)
```

```
In []: B = ones_like(C)
```

B would be array of ones with the same shape and type as C.

```
In []: A = ones((3,2))
```

A would be new matrix of given shape(arguments), filled with ones.

```
In []: I = identity(3)
```

I would be identity matrix of shape 3x3

1.2 Accessing Elements

```
In []: C
Out[]:
array([[ 1,  1,  2],
       [ 2,  4,  1],
       [-1,  3,  7]])
In []: C[1,2]
Out[]: 1
```

Two indexes separated by `','` specifies [row, column]. So `C[1,2]` gets third element of second row(indices starts from 0).

```
In []: C[1]
Out[]: array([2, 4, 1])
```

Single index implies complete row.

1.3 Changing elements

```
In []: C[1,1] = -2
In []: C
Out[]:
array([[ 1,  1,  2],
       [ 2, -2,  1],
       [-1,  3,  7]])

In []: C[1] = [0,0,0]
In []: C
Out[]:
array([[ 1,  1,  2],
       [ 0,  0,  0],
       [-1,  3,  7]])
```

1.4 Slicing

Accessing rows with Matrices is straightforward. But If one wants to access particular Column, or want a sub-matrix, Slicing is the way to go.

```
In []: C[:,1]
Out[]: array([1, 0, 3])
```

First index(:) specifies row(':' implies all the rows) and second index(1) specifies column(second column).

```
In []: C[1,:]
Out[]: array([0, 0, 0])
```

Here we get second row(1), all columns(':') of C matrix.

```
In []: C[0:2,:]
Out[]:
array([[1, 1, 2],
       [0, 0, 0]])
```

Result is sub-matrix with first and second row(endpoint is excluded), and all columns from C.

```
In []: C[1:3,:]
Out[]:
array([[ 0,  0,  0],
       [-1,  3,  7]])

In []: C[:2,:]
Out[]:
array([[1, 1, 2],
       [0, 0, 0]])
```

`':2'` => start from first row, till and excluding third row.

```
In []: C[1:,:]
Out[]:
array([[ 0,  0,  0],
       [-1,  3,  7]])

In []: C[1:,:2]
Out[]:
array([[ 0,  0],
       [-1,  3]])
```

`'1:'` => Start from second row, till last row
`':2'` => Start from first column, till and excluding third column.

1.5 Striding

Often apart from sub-matrix, one needs to get some mechanism to jump a step. For example, how can we have all alternate rows of a Matrix. Following method will return Matrix with alternate rows.

```
In []: C[::2,:]
Out[]:
array([[ 1,  1,  2],
       [-1,  3,  7]])
```

`C[startR:stopR:stepR,startC:stopC:stepC]` => Syntax of mentioning starting index, ending index, and step to jump.
In above mentioned case, `'::2'` means, start from first row, till last row(both are blank), with step of 2, that is, skipping alternate row. After first row, C[startR], next row would be C[startR+stepR] and so on.

```
In []: C[:,::2]
Out[]:
array([[ 1,  2],
       [ 0,  0],
       [-1,  7]])
```

Same as above, just that here we get matrix with each alternate column and all rows.

```
In []: C[::2,::2]
Out[]:
array([[ 1,  2],
       [-1,  7]])
```

2 Matrix Operations

For a Matrix A and B of equal shapes.

```
In []: A.T # Transpose
In []: sum(A) # Sum of all elements
In []: A+B # Addition
In []: A*B # Element wise product
In []: dot(A,b) #Matrix multiplication
In []: inv(A) # Inverse
In []: det(A) # Determinant
```

Eigen Values and Eigen Vectors

```
In []: eig(A) #Eigen Values and Vectors
In []: eigvals(A) #Eigen Values
```

3 Least Square Fit Line

```
L = []
T = []
for line in open('pendulum.txt'):
    point = line.split()
    L.append(float(point[0]))
    T.append(float(point[1]))
Tsq = []
for time in T:
    Tsq.append(time*time)
plot(L, Tsq, '.')
```

This is exact curve we get from L Vs Tsq from data.This relation among L and Tsq is not of straight line. For getting Least Square Fit line, we have to solve the relations:
$L = m * Tsq + c$ (something similar to $y = m * x + c$)
For present scenario, we have L and corresponding Tsq values. For finding m and c at given points we use `lstlq` function provided by pylab. It returns the least-squares solution to an equation.
For finding Least Square Fit line for this particular data we have to do following steps:

```
In []: A = array([L, ones_like(L)])
```

A is 2x(Length of array L) array.

```
In []: A = A.T #now A.shape = (Length of array L)x2
In []: result = lstsq(A,TSq)
In []: coef = result[0]
In []: Tline = coef[0]*L + coef[1]
```

`coef[0]` is array with all m values, and `coef[1]` contains c.
To get the final plot.

```
In []: plot(L, Tline)
```