In [ ]:
```
import numpy as np
from scipy.linalg import *
```

Homework #6

Linear Algebra for Dynamics

Equations of motion in Newtonian ($F = ma$) or Lagrangian ($\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) - \frac{\partial L}{\partial x_i} = F_i$) are solved with Linear Algebra equations. Specifically, the equations are a combination of differential and algebraic equations. Integrating the differential equations numerically is done with discrete steps, which creates another linear algebra problem.

There are two main linear algebra problems that we are concerned with here

1. $\mathbf{Ax} = \mathbf{b}$
2. $\mathbf{Ax} = \lambda\mathbf{Bx}$

Take the following matrix, saved as array `A`:

In [ ]:
```
A=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
A
```

Out[ ]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

A is an $m \times n$ matrix where m=4 rows and n=3 columns

In [ ]:
```
np.shape(A)
```

Out[ ]:
```
(4, 3)
```

The tranpose of A is 3×4

In [ ]:
```
A.T # <array>.T is the numpy method to transpose an array, for our purposes array =
```

Out[ ]:
```
array([[ 1,  4,  7, 10],
       [ 2,  5,  8, 11],
       [ 3,  6,  9, 12]])
```

Problem 1

Try making an array that is 2 rows $\times$ 3 columns. Then take its transpose so it is 3 rows $\times$ 2 columns

In [ ]:
```
B = np.array([[1, 2, 3], [4, 5, 6]])
print(B)

print(B.T)
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
```

Matrices and vectors are sets of linear equations

Representation of linear equations:

1. $5x_1 + 3x_2 = 1$
2. $x_1 + 2x_2 + 3x_3 = 2$
3. $x_1 + x_2 + x_3 = 3$

in matrix form:

$$\begin{bmatrix} 5 & 3 & 0 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$Ax = b$

Vectors

column vector x (length of 3):

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

row vector y (length of 3):

$[y_1 y_2 y_3]$

vector of length N:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

The i^{th} element of x is x_i

Matrices

elements in the matrix are denoted $B_{row\ column}$

In general, matrix, B, can be any size, $M \times N$ (M-rows and N-columns):

$$B = \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1N} \\ B_{21} & B_{22} & \dots & B_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{M1} & B_{M2} & \dots & B_{MN} \end{bmatrix}$$

Multiplication

A column vector is a $1 \times N$ matrix and a row vector is a $M \times 1$ matrix

Multiplying matrices is not commutative

$AB \neq BA$

Inner dimensions must agree, output is outer dimensions.

A is $M1 \times N1$ and B is $M2 \times N2$

C=AB

Therefore N1=M2 and C is $M1 \times N2$

If $C' = BA$, then N2=M1 and C is $M2 \times N1$

e.g. $A = \begin{bmatrix} 5 & 3 & 0 \\ 1 & 2 & 3 \end{bmatrix}$

$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$

C=AB

$[2 \times 4] = [2 \times 3][3 \times 4]$

The rule for multiplying matrices, A and B, is

$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$

In the previous example,

$C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} = 5*1 + 3*5 + 0*9 = 20$

Multiplication is associative:

$$(AB)C = A(BC)$$

and distributive:

$$A(B+C) = AB + AC$$

Note: You can multiply matrices in Python with `@`

In [ ]:
```
A=np.array([[5,3,0],[1,2,3]])
B=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
C=A@B # multiply AB
C
```

Out[ ]:
```
array([[20, 28, 36, 44],
       [38, 44, 50, 56]])
```

In [ ]:
```
# B@A
```

Problem 2

Given:

$$A = \begin{bmatrix} 5 & 3 \\ 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$$

$$C = \begin{bmatrix} 11 & 5 \\ 5 & 4 \end{bmatrix}$$

Calculate $D1 = A(B+C)$ and $D2 = AB + AC$. Are they equal?

In [ ]:
```
A=np.array([[5,3],[1,2]])
B=np.array([[1,2],[5,6]])
C=np.array([[11,5],[5,4]])
D1=A@(B+C) # replace with your algebra
D2=A@B + A@C # replace with your algebra

D1 == D2
```

Out[ ]:
```
array([[ True,  True],
       [ True,  True]])
```

Differentiation

In many applications in mechanics, scalar and vector functions that depend on one or more variables are encountered. An example of a scalar function that depends on the system velocities and possibly on the system coordinates is the kinetic energy. Examples of vector functions are the coordinates, velocities, and accelerations that depend on time. Let us first consider a scalar function f that depends on several variables q1, q2, ... , and qn and the parameter t, such that

The total derivative df/dt becomes

Where $\frac{\partial f}{\partial \mathbf{q}}$ can be rewritten as

In [ ]:
```
U=np.array([[1,2,3],[0,4,5],[0,0,6]])
L=np.array([[1,0,0],[2,3,0],[4,5,6]])
D=np.diag([1,2,3])
print('upper triangular matrix, U:')
print(U)
print('lower triangular matrix, L:')
print(L)
print('diagonal matrix, D:')
print(D)
```

```
upper triangular matrix, U:
[[1 2 3]
 [0 4 5]
 [0 0 6]]
lower triangular matrix, L:
[[1 0 0]
 [2 3 0]
 [4 5 6]]
diagonal matrix, D:
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Problem 3

Make a function that returns the partial derivative of $f(q_1, q_2, q_3, t)$ with respect to $\mathbf{q}$

$\frac{\partial f}{\partial \mathbf{q}}$

In [ ]:
```
def dfdq(q1,q2,q3,t):
    # your work
```

```
    df = np.array([q3, -6*q2, q1])
    return df
```

Use your function `dfdq` to create a function that returns the total derivative of $f(q_1, q_2, q_3, t)$ with respect to t

if $q_1(t) = 2t$, $q_2(t) = 5t$, and $q_3(t) = t^2$

In [ ]:
```
def dfdt(q1,q2,q3,t):
    df=dfdq(q1,q2,q3,t)
    dfdt = df*np.array([2, 5, 2*t]) + 10*t
    return dfdt
t = 1
q1 = 2*t
q2 = 5*t
q3 = t**2

dfdt(q1, q2, q3, t)
```

Out[ ]:
```
array([  12, -140,   14])
```

For a number of functions, $f_1 \ldots f_n$

The total derivative is a similar form

or

Problem 4

Create a function that returns $\frac{\partial \mathbf{f}}{\partial \mathbf{q}}$ where $\mathbf{f}$ is defined as

In [ ]:
```
def dfdq2(q,t):
    '''Here q=[q1,q2,q3] and t = time'''
    # your work here
    f1 = np.array([2*q[0], 6*q[1], 0, -15*q[3]**2])
    f2 = np.array([0, 2*q[1], -2*q[2], 0])
    f3 = np.array([q[3], q[2], q[1], q[0]])

    df = np.array([f1, f2, f3])

    return df
```

Use your `dfdq2` to calculate $\frac{\partial \mathbf{f}}{\partial \mathbf{q}}$ when q=[1,3,5], and t=3

In [ ]:
```
q=np.array([1,3,5, 1])
t=3
dfdq2(q,t)
```

Out[ ]:
```
array([[  2,  18,   0, -15],
       [  0,   6, -10,   0],
       [  1,   5,   3,   1]])
```

Inverse of matrices

The inverse of a square matrix, A^{-1} is defined such that

$$A^{-1}A = I = AA^{-1}$$

Not all square matrices have an inverse, they can be *singular* or *non-invertible*

The inverse has the following properties:

1. $(A^{-1})^{-1} = A$
2. $(AB)^{-1} = B^{-1}A^{-1}$
3. $(A^{-1})^T = (A^T)^{-1}$

In [ ]:
```
A=np.random.rand(3,3)
A
```

Out[ ]:
```
array([[0.68593578, 0.88935458, 0.49059219],
       [0.06491161, 0.67814298, 0.02589029],
       [0.97760485, 0.22243549, 0.7983029 ]])
```

In [ ]:
```
Ainv=inv(A)
inv(Ainv)
```

Out[ ]:
```
array([[0.68593578, 0.88935458, 0.49059219],
       [0.06491161, 0.67814298, 0.02589029],
       [0.97760485, 0.22243549, 0.7983029 ]])
```

In [ ]:
```
B=np.random.rand(3,3)
```

In [ ]:
```
print(inv(np.dot(A,B)))
print('==')
print(np.dot(inv(B),inv(A)))

inv(A.T)

inv(A).T
```

```
[[ -910.85576952  1030.72681536   549.61954022]
 [-1497.46823818  1701.00850236   901.62930481]
 [ 1957.04544874 -2219.25871035 -1178.75441687]]
==
[[ -910.85576952  1030.72681536   549.61954022]
 [-1497.46823818  1701.00850236   901.62930481]
 [ 1957.04544874 -2219.25871035 -1178.75441687]]
```

Out[ ]:
```
array([[ 20.87534666,  -1.03318244, -25.27614976],
       [-23.41827567,   2.6495101 ,  27.93986436],
       [-12.06932368,   0.54900796,  15.87982517]])
```

Orthogonal Matrices

Vectors are *orthogonal* if x^T y=0, and a vector is *normalized* if $||x||_2 = 1$. A square matrix is *orthogonal* if all its column vectors are orthogonal to each other and normalized. The column vectors are then called *orthonormal* and the following results

$U^TU = I = UU^T$

and

$||Ux||_2 = ||x||_2$

Determinant

The **determinant** of a matrix has 3 properties

1. The determinant of the identity matrix is one, $|I| = 1$

2. If you multiply a single row by scalar t then the determinant is $t|A|$:

$$t|A| = \begin{bmatrix} tA_{11} & tA_{12} & \dots & tA_{1N} \\ A_{21} & A_{22} & \dots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \dots & A_{MN} \end{bmatrix}$$

Determinant (con'd)

3. If you switch 2 rows, the determinant changes sign:

$$-|A| = \begin{bmatrix} A_{21} & A_{22} & \dots & A_{2N} \\ A_{11} & A_{12} & \dots & A_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \dots & A_{MN} \end{bmatrix}$$

Determinant (con'd)

4. inverse of the determinant is the determinant of the inverse:

$$|A^{-1}| = \frac{1}{|A|} = |A|^{-1}$$

Calculating the Determinant

For a 2×2 matrix,

$$|A| = \left|\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\right| = A_{11}A_{22} - A_{21}A_{12}$$

For a 3×3 matrix,

$$|A| = \left|\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}\right| =$$

$$A_{11}A_{22}A_{33} + A_{12}A_{23}A_{31} + A_{13}A_{21}A_{32} - A_{31}A_{22}A_{13} - A_{32}A_{23}A_{11} - A_{33}A_{21}A_{12}$$

For larger matrices, the determinant is more involved

Special Case for determinants

The determinant of a diagonal matrix $|D| = D_{11}D_{22}D_{33}...D_{NN}$.

Similarly, if a matrix is upper triangular (so all values of $A_{ij} = 0$ when $j < i$), the determinant is

$$|A| = \left|\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1N} \\ 0 & A_{22} & \dots & A_{2N} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & A_{NN} \end{bmatrix}\right| = A_{11}A_{22}A_{33}...A_{NN}$$

eqn

Problem 5

Find the sum $A + B$, the determinants, $|A|$ and $|B|$, trace(A), and trace(B)

In [ ]:
```
print(A:= np.array([[-3, 8, -20.5],
                    [5, 11, 13],
                    [7, 20, 0]]))
print(B:= np.array([[0, 3.2, 0],
                    [-17.5, 5.7, 0],
```

```
                    [12, 6.8, -10]]))

print(sum:= A+B)

print(np.linalg.det(A))
print(np.linalg.det(B), 1)
print(np.trace(A))
print(np.trace(B))
```

```
[[ -3.    8.  -20.5]
 [  5.   11.   13. ]
 [  7.   20.    0. ]]
[[  0.    3.2   0. ]
 [-17.5   5.7   0. ]
 [ 12.    6.8 -10. ]]
[[ -3.   11.2 -20.5]
 [-12.5  16.7  13. ]
 [ 19.   26.8 -10. ]]
1036.4999999999995
-560.0000000000003 1
8.0
-4.3
```

In [ ]: