Matrices

matrix is a 2 dimensional array where numbers, symbols or expressions are arranges into rows and columns. using the concept of lists, one can easily define a matrix in python.

we define 2 matrices A and B

In [ ]:
```
A = [[1,2,3],[4,5,6],[7,8,9]]
B = [[9,8,7],[6,5,4],[3,2,1]]

print(A)
print(B)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[9, 8, 7], [6, 5, 4], [3, 2, 1]]
```

In [ ]:
```
# from sympy import pprint

print(A + B)
print(2*B)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [9, 8, 7], [6, 5, 4], [3, 2, 1]]
[[9, 8, 7], [6, 5, 4], [3, 2, 1], [9, 8, 7], [6, 5, 4], [3, 2, 1]]
```

In [ ]:
```
A*B #multiplication of lists containing lists isnot possible
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 5 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb#X10sZmlsZQ%3D%3D?line=0'>1</a> print(A*B)

TypeError: can't multiply sequence by non-int of type 'list'
```

In [ ]:
```
A**2 #** or pow() doesnot work for list of lists
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 6 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb#X11sZmlsZQ%3D%3D?line=0'>1</a> A**2

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

In [ ]:
```
A[0].append(10)
```

In [ ]:
```
A
```

Out[ ]:
```
[[1, 2, 3, 10], [4, 5, 6], [7, 8, 9]]
```

the matrix A being defined as a list of lists, python supports adding of single element to rows or columns, which falsifies the very definition of a matrix

thorugh all the above command we can see that even though it is possible to define a matrix as a list of lists, matrix manipulation is not straight forward as desired

Numpy package

numpy is a library consisting of multidimensional array objects and package of functions for processing those arrays. Numpy stands for Numerical Python. The following operations can be performed in numpy

- Mathematical and logical operations on arrays
- fourier tranforms and routines for shape manipulation
- operations related to linear algebra

In [ ]:
```
import numpy as np
```

numpy provides matrix() and array() functions

In [ ]:
```
A1 = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(A1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [ ]:
```
M1 = np.matrix([[1,2,3],[4,5,6],[7,8,9]])

print(M1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

the difference between array() and matrix() ?

- array() provides n-dimensional array while matrix() is strictly 2-d
- the * and ** operator performs elementwise in array while in matrix they are used for multiplication and finding powers

common errrors

In [ ]:
```
N = np.matrix([1,2,3],[4,5,6],[7,8,9])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 19 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#X25sZmlsZQ%3D%3D?line=0'>1</a> N = np.matrix([1,2,3],[4,5,6],[7,8,9])

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\matrixlib
\defmatrix.py:145, in matrix.__new__(subtype, data, dtype, copy)
    142     data = _convert_from_string(data)
    144 # now convert data to an array
--> 145 arr = N.array(data, dtype=dtype, copy=copy)
    146 ndim = arr.ndim
    147 shape = arr.shape

TypeError: Field elements must be 2- or 3-tuples, got '4'
```

Matrix manipulation

In [ ]:
```
# shape function for order

M1.shape
```

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

In [ ]:
```
print(M1.shape[0]) #no of rows
print(M1.shape[1]) #no of columns
```

```
3
3
```

In [ ]:
```
# size function for number of elements
M1.size
```

Out[ ]:
```
9
```

In [ ]:
```
# function zeroes creates an array of zeroes
np.zeros((4,4))
```

Out[ ]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [ ]:
```
np.ones((4,4))
```

Out[ ]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

zeros provide an array of zeros and not a matrix

the zeroes and ones function gives float value by default

np.zeros((4,4),int)

In [ ]:
```
np.zeros((4,4),int)
```

Out[ ]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [ ]:
```
np.ones((4,4),dtype = int)
```

Out[ ]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Matrix operations

In [ ]:
```
d = np.matrix([[1,2,3],[4,5,6],[7,8,9]])

e = np.matrix([[1,2,3],[4,5,6],[7,8,9]])

f = np.matrix([[1,2,3],[4,5,6],[7,8,9]])
```

In [ ]:
```
d + e
```

Out[ ]:
```
matrix([[ 2,  4,  6],
        [ 8, 10, 12],
        [14, 16, 18]])
```

In [ ]:
```
np.add(d,e)
```

Out[ ]:
```
matrix([[ 2,  4,  6],
        [ 8, 10, 12],
        [14, 16, 18]])
```

In [ ]:
```
d - e
```

Out[ ]:
```
matrix([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

In [ ]:
```
np.multiply(d,e)
```

Out[ ]:
```
matrix([[ 1,  4,  9],
        [16, 25, 36],
        [49, 64, 81]])
```

In [ ]:
```
d.dot(f) # matrix multiplication
```

Out[ ]:
```
matrix([[ 30,  36,  42],
        [ 66,  81,  96],
        [102, 126, 150]])
```

In [ ]:
```
e/d
```

Out[ ]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

In [ ]:
```
np.divide(e,d)
```

Out[ ]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

Transpose of a matrix

In [ ]:
```
d.transpose()
```

Out[ ]:
```
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

In [ ]:
```
np.transpose(d)
```

Out[ ]:
```
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

In [ ]:
```
d.T
```

Out[ ]:
```
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
```

Upper and lower triangular matrix

In [ ]:
```
ut = np.triu(d)
ut
```

Out[ ]:
```
array([[1, 2, 3],
       [0, 5, 6],
       [0, 0, 9]])
```

In [ ]:
```
lt = np.tril(d)
lt
```

Out[ ]:
```
array([[1, 0, 0],
       [4, 5, 0],
       [7, 8, 9]])
```

numpy.linalg

linear algebra module

- rank, determinant, trace, inverse etc of an array
- eigen values of matrices
- matrix and vector products
- solve linear equations

determinant

In [ ]:
```
det = np.linalg.det(d)
det
```

Out[ ]:
```
0.0
```

inverse of matrix

In [ ]:
```
x = np.matrix([[1,2],[4,9]])
x**-1
```

Out[ ]:
```
matrix([[ 9., -2.],
        [-4.,  1.]])
```

In [ ]:
```
np.linalg.inv(x)
```

Out[ ]:
```
matrix([[ 9., -2.],
        [-4.,  1.]])
```

eigen values

In [ ]:
```
print(d)

np.linalg.eigvals(d)
```

Out[ ]:
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
array([ 1.61168440e+01, -1.11684397e+00, -9.75918483e-16])
```

In [ ]:
```
h = np.matrix([[1,0,0],[0,9,0],[0,0,8]])

np.linalg.eigvals(h)
```

Out[ ]:
```
array([1., 9., 8.])
```

In [ ]:
```

```

Identity matrix

verify the following properties by suitable examples

- symmetric transpose of identity matrix is itself
- identity matrix is nonsimgular
- inverse of identity matrix is itself
- eigenvalues are all 1's
- multiplying any matrix by the unit matrix , gives the marix itself
- We always get an identity after multiplying two inverse matrices

In [ ]:
```
import numpy as np
```

In [ ]:
```
i = np.matrix([[1,0,0],[0,1,0],[0,0,1]])

print("identity matrix =\n ",i)

trans = np.transpose(i)
```

```
print("transpose of  matrix =\n ",i)

de = np.linalg.det(i)

print("determinant :\n ", de)

inv = np.linalg.inv(i)

print("inverse = \n", inv)

eig = np.linalg.eigvals(i)

print("eigen values : ",eig)

a = np.matrix([[1,0,0],[0,5,0],[0,0,9]])

print("a = \n",a)

print("a*i = \n",a*i)

a_i = np.linalg.inv(a)
```

```
identity matrix =
  [[1 0 0]
 [0 1 0]
 [0 0 1]]
transpose of  matrix =
  [[1 0 0]
 [0 1 0]
 [0 0 1]]
determinant :
  1.0
inverse =
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
eigen values :  [1. 1. 1.]
a =
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
a*i =
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
```

Diagonal Matrix

verify the properties :

- addition, multiplication of diagonal matrix gives a diagonal matrix again:
- symmetric transpose of matrix is itself
- determinant of diag(ai) = a1*a2...
- identity matrix is nonsingular if diagonal entries are all on zero
- diag(ai..)^-1 = diaag(a1^-1,....an^-1)
- eigen values are diagonal entries

In [ ]:
```
d1 = np.matrix([[1,0,0],[0,2,0],[0,0,3]])
d2 = np.matrix([[4,0,0],[0,5,0],[0,0,6]])
print("d1 = \n",d1)
print("d2 = \n",d2)

d = d1 + d2
```

```
c = d1 * d2

print("d1 + d2 = \n",d)
print("d1 * d2 = \n", c)

tra = np.transpose(d1)
print("transpose = \n",d1)

#find inverse and eigen values of the same
```

```
d1 =
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
d2 =
 [[4 0 0]
 [0 5 0]
 [0 0 6]]
d1 + d2 =
 [[5 0 0]
 [0 7 0]
 [0 0 9]]
d1 * d2 =
 [[ 4  0  0]
 [ 0 10  0]
 [ 0  0 18]]
transpose =
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Upper triangular matrix

- adding returns utm
- multiplying results in utm
- transpose will be ltm
- inverse remains utm
- determinant od diag(a1...an) = a1*a2*....an
- eigen values are diagonal entries

In [ ]:
```
u1 = np.matrix([[1,4,5],[0,2,6],[0,0,3]])
u2 = np.matrix([[4,1,2],[0,5,3],[0,0,6]])

print("u1 = \n",u1)
print("u2 = \n",u2)

u = u1 + u2
c = u1 * u2

print("u1 + u2 = \n",u)
print("u1 * u2 = \n", c)

tra = np.transpose(u1)
print("transpose = \n",u1)

de = np.linalg.det(u1)

print("determinant =\n",de)
```

```
u1 =
 [[1 4 5]
 [0 2 6]
 [0 0 3]]
u2 =
 [[4 1 2]
 [0 5 3]
 [0 0 6]]
u1 + u2 =
 [[5 5 7]
 [0 7 9]
 [0 0 9]]
u1 * u2 =
 [[ 4 21 44]
 [ 0 10 42]
 [ 0  0 18]]
transpose =
 [[1 4 5]
 [0 2 6]
 [0 0 3]]
determinant =
 6.0
```

Singular Matrix

- det is 0
- a non-invertible matrix is reffered to as singlular matrix, i.e, when the determinant of a matrix is zero, we cannot find its inverse, there will be multiplicative inverse for the matrix
- Product of singular and non singular matri gives singular matrix

In [ ]:
```python
import numpy as np

s = np.matrix([[1,4,7],[1,4,7],[2,3,0]])


print("singular =\n",s)

trans = np.transpose(s)

print("transpose of  matrix =\n",trans)

de = np.linalg.det(s)

print("determinant :\n", de)

# inv = np.linalg.inv(s)

# print("inverse = \n", inv)

eig = np.linalg.eigvals(s)

print("eigen values : ",eig)

a = np.matrix([[1,0,0],[0,5,0],[0,0,9]])
print("a = \n",a)

print("a*s =\n",a*s)
d = np.linalg.det(a*s)
print("determinant of a*s = ",d)
```

```
singular =
 [[1 4 7]
 [1 4 7]
 [2 3 0]]
transpose of  matrix =
 [[1 1 2]
 [4 4 3]
 [7 7 0]]
determinant :
 0.0
eigen values :  [ 8.92261629e+00 -4.81385765e-17 -3.92261629e+00]
a =
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
a*s =
 [[ 1  4  7]
 [ 5 20 35]
 [18 27  0]]
determinant of a*s =  0.0
```

Rank of a Matrix

In [ ]:
```
from numpy.linalg import matrix_rank

i = np.eye(4)

print(i)

r = matrix_rank(i)

print("rank = ",r)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
rank =  4
```

In [ ]:
```
i = np.eye(4)
i[3,3] = 0

print(i)
r = matrix_rank(i)

print("rank = ",r)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
rank =  3
```

In [ ]:
```
p = np.matrix([[1,1,-1],[2,-3,4],[3,-2,3]])
q = np.matrix([[-1,-2,-1],[6,12,6],[5,10,5]])

print(p+q)
r = matrix_rank(p+q)

print("rank = ",r)
```

```
[[ 0 -1 -2]
 [ 8  9 10]
 [ 8  8  8]]
rank =  2
```

In [ ]:
```
from numpy.linalg import matrix_rank
```

```
i = np.ones((4,4))

print(i)
r = matrix_rank(i)

print("rank = ",r)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
rank =  1
```

In [ ]:
```
b = np.matrix([[2,-1,3],[1,0,1],[1,1,4]])
print(b)
r = matrix_rank(b)

print("rank = ",r)
print()
bt = np.transpose(b)

print(bt)
r = matrix_rank(bt)

print("rank = ",r)
```

```
[[ 2 -1  3]
 [ 1  0  1]
 [ 1  1  4]]
rank =  3

[[ 2  1  1]
 [-1  0  1]
 [ 3  1  4]]
rank =  3
```

In [ ]:
```
#show rank of p is no of non zero eigen values
p = np.matrix([[1,1,-1],[2,-3,4],[3,-2,3]])

print(p)
r = matrix_rank(p)

print("rank = ",r)


eig = np.linalg.eigvals(p)

print("eigen values : ",eig)
```

```
[[ 1  1 -1]
 [ 2 -3  4]
 [ 3 -2  3]]
rank =  2
eigen values :  [ 1.00000000e+00+0.00000000e+00j -2.24492778e-16+4.40332636e-08j
 -2.24492778e-16-4.40332636e-08j]
```

system of linear equations

a linear equation in variables x1,x2,..xn is an equation of the form a1x1, a2x2,...an,xn and b are constants.

the constants ai is called the coefficient of xi

A system of linear equations is a finite collection of linear equations in the same variables.

linear wquations in n variables $x_1, x_2 \ldots . x_n$

$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$

$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$

$\ldots$

$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m$

A solution of a linear system is an assignment of values to the variables $x_1, x_2 \ldots, x_n$ such that each of the equations is satisfied. The set of all solutions of a linear system is called the solution set of the system

in a matrix notation a linear system is $AX = B$, where

$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{11} & a_{12} & \ldots & a_{1n} \\ & \ldots & & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$ is the coefficient matrix ,

$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $B = \begin{bmatrix} xb1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$.

python has numpy.linalg.solve() to solve these sytem of linear equations

syntax

numpy.linalg.solve(A,B)

Q. Find all solutions for the linear system

$x_1 + 2x_2 - x_3 = 1$
$2x_1 + x_2 + 4x_3 = 2$
$3x_1 + 3x_2 + 4x_3 = 1$

In [ ]:
```
import numpy as np
A = np.array([[1,2,-1],[2,1,4],[3,3,4]])
B = np.array([1,2,1])

print(np.linalg.solve(A,B))
```

```
[ 7. -4. -2.]
```

In [ ]:
```
A = np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B = np.matrix([1,2,1])

print(np.linalg.solve(A,B))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 79 in <cell line: 4>()
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y145sZmlsZQ%3D%3D?line=0'>1</a> A = np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y145sZmlsZQ%3D%3D?line=1'>2</a> B = np.matrix([1,2,1])
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y145sZmlsZQ%3D%3D?line=3'>4</a> print(np.linalg.solve(A,B))

File <__array_function__ internals>:180, in solve(*args, **kwargs)

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\linalg\li
nalg.py:400, in solve(a, b)
    398 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    399 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 400 r = gufunc(a, b, signature=signature, extobj=extobj)
    402 return wrap(r.astype(result_t, copy=False))

ValueError: solve: Input operand 1 has a mismatch in its core dimension 0, with gufunc signat
ure (m,m),(m,n)->(m,n) (size 1 is different from 3)
```

this is because B is a $m \times 1$ matrix in the matrix equivalent of linear system of equations

In [ ]:
```
A = np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B = np.matrix([[1],[2],[1]])

print(np.linalg.solve(A,B))
```

```
[[ 7.]
 [-4.]
 [-2.]]
```

The function np.linalg.solve() works only if A is a non-singular matrix

In [ ]:
```
A = np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0],[-2,2,-2,1]])
B = np.matrix([[2],[0],[4],[-3]])

print(np.linalg.solve(A,B))
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 83 in <cell line: 4>()
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y152sZmlsZQ%3D%3D?line=0'>1</a> A = np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0],[-2,2,-2,
1]])
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y152sZmlsZQ%3D%3D?line=1'>2</a> B = np.matrix([[2],[0],[4],[-3]])
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y152sZmlsZQ%3D%3D?line=3'>4</a> print(np.linalg.solve(A,B))

File <__array_function__ internals>:180, in solve(*args, **kwargs)

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\linalg\li
nalg.py:400, in solve(a, b)
    398 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    399 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 400 r = gufunc(a, b, signature=signature, extobj=extobj)
    402 return wrap(r.astype(result_t, copy=False))

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\linalg\li
nalg.py:89, in _raise_linalgerror_singular(err, flag)
     88 def _raise_linalgerror_singular(err, flag):
---> 89     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

this is because determinant of A is 0

linalg.solve() only works oif the matrix is a square matrix

In [ ]:
```
A = np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0]])
B = np.matrix([[2],[0],[4]])

print(np.linalg.solve(A,B))
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\8-06-22.ipynb Cell 86 in <cell line: 4>()
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y153sZmlsZQ%3D%3D?line=0'>1</a> A = np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0]])
      <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y153sZmlsZQ%3D%3D?line=1'>2</a> B = np.matrix([[2],[0],[4]])
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/8-06-22.ipynb
#Y153sZmlsZQ%3D%3D?line=3'>4</a> print(np.linalg.solve(A,B))

File <__array_function__ internals>:180, in solve(*args, **kwargs)

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\linalg\li
nalg.py:387, in solve(a, b)
    385 a, _ = _makearray(a)
    386 _assert_stacked_2d(a)
--> 387 _assert_stacked_square(a)
    388 b, wrap = _makearray(b)
    389 t, result_t = _commonType(a, b)

File c:\Users\joelv\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\linalg\li
nalg.py:204, in _assert_stacked_square(*arrays)
    202 m, n = a.shape[-2:]
    203 if m != n:
--> 204     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

In [ ]:
```
```

Exercise problems

questions

1. write a program to solve any system of linear equations(user input), check whether the coefficient matrix is singular or not. If singular , print no solution. If non singular then print the inverse exst and solve the system of equation

In [ ]:
```
l = []
m = []
n = []
p = []



import numpy as np
A = np.array([[1,3,-5],[4,2,1]])
B = np.array([1,2])
if  (A.shape[0] == A.shape[1]):
    de = np.linalg.det(A)
    if de ==0:
        print("determinant :\n", de)
        print("therefore no solution")
    else:
        print(np.linalg.solve(A,B))
```

```
        # using inverse method X = A^-1 * B

        x =np.linalg.inv(A).dot(B)
        print("uding inverse method = " ,x)
else:
    print("no solution- not square matrix")
```

```
no solution- not square matrix
```

1. solve the following system of linear equations
   - $x_1 + 2x_2 - x_3 = -1$
   - $2x_1 + x_2 + 4x_3 = 2$
   - $3x_1 + 3x_2 + 4x_3 = 1$

In [ ]:
```
import numpy as np
A = np.array([[1,2,-1],[2,1,4],[3,3,4]])
B = np.array([-1,2,1])
de = np.linalg.det(A)
if de ==0:

    print("determinant :\n", de)
    print("therefore no solution")
else:
    print(np.linalg.solve(A,B))
```

```
[ 1.66666667e+00 -1.33333333e+00 -2.22044605e-16]
```

In [ ]:
```
# using inverse method X = A^-1 * B

x =np.linalg.inv(A).dot(B)
x
```

Out[ ]:
```
array([ 1.66666667, -1.33333333,  0.        ])
```

1. solve
   - $x_1 - x_2 + x_3 - x_4 = 2$
   - $x_1 - x_2 + x_3 + x_4 = 0$
   - $4x_1 - 4x_2 + 4x_3 = 4$
   - $-2x_1 + 2x_2 - 2x_3 + x_4 = -3$

In [ ]:
```
A = np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0],[-2,2,-2,1]])
B = np.matrix([[2],[0],[4],[-3]])
de = np.linalg.det(A)

if de ==0:

    print("determinant :\n", de)
    print("therefore no solution")
else:
    print(np.linalg.solve(A,B))
```

```
determinant :
 0.0
therefore no solution
```

1. solve
   - $x_1 - 2x_2 - x_3 = 1$
   - $2x_1 + x_2 - 5x_3 = 2$
   - $3x_1 + 3x_2 + 4x_3 = 1$

In [ ]:
```
import numpy as np
A = np.array([[1,-2,-1],[2,1,-5],[3,3,4]])
B = np.array([1,2,1])
de = np.linalg.det(A)
if de ==0:

    print("determinant :\n", de)
    print("therefore no solution")
else:
    print(np.linalg.solve(A,B))

# using inverse method X = A^-1 * B

x =np.linalg.inv(A).dot(B)
print("uding inverse method = " ,x)
```

```
[ 0.64516129 -0.09677419 -0.16129032]
uding inverse method =  [ 0.64516129 -0.09677419 -0.16129032]
```

1. solve
   - $x_1 + 2x_2 + x_3 = 4$
   - $4x_1 + 5x_2 + 6x_3 = 7$
   - $7x_1 + 8x_2 + 9x_3 = 10$

In [ ]:
```
import numpy as np
A = np.array([[1,2,-1],[4,5,6],[7,8,9]])
B = np.array([4,7,10])
de = np.linalg.det(A)
if de ==0:

    print("determinant :\n", de)
    print("therefore no solution")
else:
    print(np.linalg.solve(A,B))

# using inverse method X = A^-1 * B

x =np.linalg.inv(A).dot(B)
print("uding inverse method = " ,x)
```

```
[-2.00000000e+00  3.00000000e+00 -1.74463618e-16]
uding inverse method =  [-2.00000000e+00  3.00000000e+00 -3.88578059e-16]
```

1. solve
   - $x_1 + 3x_2 - 5x_3 = 1$
   - $4x_1 + 2x_2 + x_3 = 2$

In [ ]:
```
import numpy as np
A = np.array([[1,3,-5],[4,2,1]])
B = np.array([1,2])
if  (A.shape[0] == A.shape[1]):
    de = np.linalg.det(A)
    if de ==0:
        print("determinant :\n", de)
        print("therefore no solution")
    else:
        print(np.linalg.solve(A,B))

        # using inverse method X = A^-1 * B

        x =np.linalg.inv(A).dot(B)
        print("uding inverse method = " ,x)
else:
    print("no solution- not square matrix")
```

```
no solution- not square matrix
```

1. suppose a fruit seller sold 20 mangoes + 10 oranges in one day for 350 dollars. next day 17 mangoes + 22 oranges for 500 dollars. find the prices of one mango and one orange

In [ ]:
```
import numpy as np
A = np.array([[20,10],[17,22]])
B = np.array([350,500])
de = np.linalg.det(A)
if de ==0:

    print("determinant :\n", de)
    print("therefore no solution")
else:
    print(np.linalg.solve(A,B))

# using inverse method X = A^-1 * B

x =np.linalg.inv(A).dot(B)
print("uding inverse method = " ,x)
```

```
[10. 15.]
uding inverse method =  [10. 15.]
```

Solving non Linear equations

The solution to non linear equations is through matrix operations while for nonlinear they require solvers such as scipy optimize and fsolve to numerically find the solution Q1. **Solve the equation** $f(x) = x^2 - 5$

Solving a non linear eq involves first putting it inthe form of f(x)=0

We know the function f and we want to find the valur of x that gives f(x)=0

To solve this

- define the function f(x)
- set an initial guess for x.
- we need to import a library:from scipy.optimize import fsolve
- call the func fsolve: x=fsolve(f,x0)
- f is the name of the function ur solving and x0 is ur initial guess

Funtion in single variables

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

from scipy.optimize import fsolve
```

In [ ]:
```
from re import X


def f(x):
    return x**2 - 5

x = fsolve(f,2)
```

```
print(x,f(x))
```

```
[2.23606798] [-1.77635684e-15]
```

function in muliple variables

Q2. **solve the function** $x^{2} + y^{2} = 16

In [ ]:
```
def myf(z):
    x = z[0]
    y = z[1]

    F = np.empty((2))
    F[0] = x**2 + y**2 - 16
    return F

zguess = np.array([1,1])

z = fsolve(myf,zguess)

print(z,myf(z))
```

```
[1. 1.] [-14.   1.]
```

Exercises

solve single equations with one variable

1. $x^2 + x + 1 = 3$
2. $y + 2 * Cos(y) = 0$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

from scipy.optimize import fsolve
from re import X


def f(x):
    return x**2 - x - 2

x = fsolve(f,2)

print(x,f(x))
```

```
[2.] [0.]
```

In [ ]:
```
import math
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

from scipy.optimize import fsolve
from re import X


def f(y):
    return y - 2*math.cos(y)
```

```
y = fsolve(f,2)

print(y,f(y))
```

```
[1.02986653] [0.]
```

Sequence

what is a sequence?

A sequence is the generic term for an ordered set

In [ ]:
```
from sympy import SeqFormula, Symbol

n = Symbol('n')
s = SeqFormula(n**2, (n,0,5))
s.formula
```

Out[ ]: n^2

In [ ]:
```
#for value at a perticular
s.coeff(3)
```

Out[ ]: 9

supports slicing

In [ ]:
```
s[:]
```

Out[ ]:
```
[0, 1, 4, 9, 16, 25]
```

In [ ]:
```
s[1:4]
```

Out[ ]:
```
[1, 4, 9]
```

Sequence elements are displayed using list() command

In [ ]:
```
list(s)
```

Out[ ]:
```
[0, 1, 4, 9, 16, 25]
```

lets discuss on few types of sequence in python

- strings : a group of characters

to declare an empty string we may use the function str()

In [ ]:
```
name = str()

name
```

Out[ ]:
```
''
```

In [ ]:
```
name = str('ann')
name
```

Out[ ]:
```
'ann'
```

- list: is an ordered group of items. to declare it, we use the square brackets.

In [ ]:
```
groceries = ['milk','bread','eggs']
groceries
```

Out[ ]:
```
['milk', 'bread', 'eggs']
```

In [ ]:
```
# a list can hold all kind of items
mylist = [1,'2',3.0,False]
mylist
```

Out[ ]:
```
[1, '2', 3.0, False]
```

In [ ]:
```
#also, a list is mutable. This means we can change a value.
mylist[0] = 0
```

- tuples:a tuple is immutable group of items

In [ ]:
```
n = (1,2)
n
```

Out[ ]:
```
(1, 2)
```

- range

In [ ]:
```
# range(start, stop,step)

for i in range(0, 10, 2):
    print(i)
```

```
0
2
4
6
8
```

In [ ]:
```
# program to generate sequence 3,5,7,9....19

for i in range(1, 20, 2):
    print(i)
```

```
1
3
5
7
9
11
13
15
17
19
```

Sequence operations

concatenation

In [ ]:
```
[1, 2, 3] + [1, 2, 3]
```

Out[ ]:
```
[1, 2, 3, 1, 2, 3]
```

In [ ]:
```
(1,2,3) + (2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\joelv\Desktop\projects-2\math\8-06-22(math final).ipynb Cell 139 in <cell line: 1>()
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects-2/math/8-06-22%28math%20final%29.ipynb#Y262sZmlsZQ%3D%3D?line=0'>1</a> (1,2,3) + (2)

TypeError: can only concatenate tuple (not "int") to tuple
```

In [ ]:
```
(9,0,7)+(9,)
```

Out[ ]:
```
(9, 0, 7, 9)
```

repeat

In [ ]:
```
(1,2,3) *3
```

Out[ ]:
```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

In [ ]:
```
[1,2,3]*3
```

Out[ ]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [ ]:
```
"b"*3
```

Out[ ]:
```
'bbb'
```

membership

In [ ]:
```
'da' in 'daisy'
```

Out[ ]:
```
True
```

In [ ]:
```
'z' in [1,2,3,'z']
```

Out[ ]:
```
True
```

In [ ]:
```
1 in range(1,3)
```

Out[ ]:
```
True
```

sequence functions

len

In [ ]:
```
d = 'bla'

len(d)
```

Out[ ]:
```
3
```

In [ ]:
```
d = [1,2,234,2,234,23,33]
len(d)
```

Out[ ]:
```
7
```

max or min

In [ ]:
```
print(max([12,3,45,1]))
```

```
45
```

index

In [ ]:
```
"joel".index('o')
```

Out[ ]:
```
1
```

count

In [ ]:
```
"deeeeeez plolp".count("e")
```

Out[ ]:
```
6
```

In [ ]:
```

```

In [ ]:
```
from sympy import *

x, y = symbols('x,y')
f = Eq(x-y,-1)
f
```

Out[ ]:
$$x - y = -1$$

In [ ]:
```
g = Eq(y,x**2 +1)
g
```

Out[ ]:
$$y = x^2 + 1$$

In [ ]:
```
solve([f,g],[x,y])
```

Out[ ]:
```
[(0, 1), (1, 2)]
```

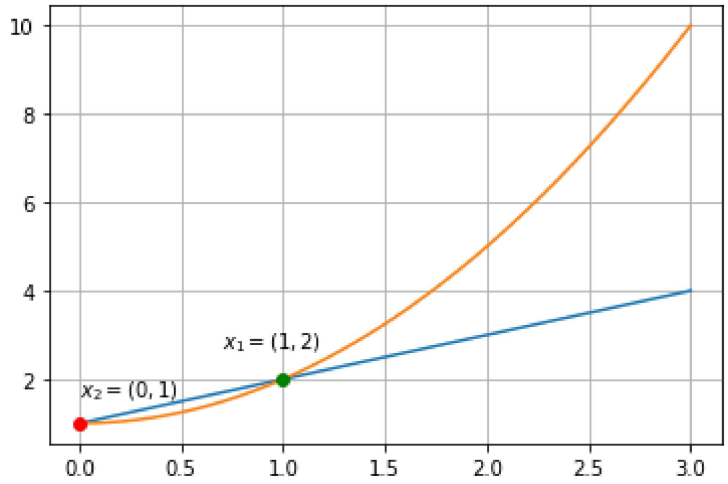
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,3,1000)
plt.plot(x,x+1,label = "$x - y = -1$")
plt.plot(x,(x**2)+1,label = "$y = x^{2} + 1$")
plt.grid()


plt.plot(1,2,'go')

plt.annotate('$x_{1} = (1,2)$',(0.7,2.7))
plt.plot(4.03896783e-28,1,'ro')
plt.annotate('$x_{2} = (0,1)$',(0,1.6))

plt.show()
```



In [ ]:

1) $y = 2x + 2$; $y = x - 1$ 2) $y = -2x - 7$; $y = x + 9$ 3) $y = -1$; $(x - 2)^2 + (y + 3)^2 = 4$ 4) $x = -6$; $(x + 3)^2 + (y - 1)^2 = 9$