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))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
c:\Users\joelv\Desktop\projects 2\math\system of linear equations.ipynb Cell 13 in <cell lin
e: 1>()
----> <a href='vscode-notebook-cell:/c%3A/Users/joelv/Desktop/projects%202/math/system%20of%2
0linear%20equations.ipynb#X15sZmlsZQ%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/system%20of%2
0linear%20equations.ipynb#X15sZmlsZQ%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/system%20of%2
0linear%20equations.ipynb#X15sZmlsZQ%3D%3D?line=3'>4</a> print(np.linalg.solve(A,B))

NameError: name 'np' is not defined
```

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  = [int(x) for x in input("Enter coeefficirnts of  first equation : ").split()]
m  = [int(x) for x in input("Enter coeefficirnts of  second equation : ").split()]
n  = [int(x) for x in input("Enter coeefficirnts of  third equation : ").split()]

p  = [int(x) for x in input("Enter the three constants in order: ").split()]




import numpy as np
A = np.array([l,m,n])
B = np.array(p)
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")
```

```
Enter coeefficirnts of  first equation : 1 2 -1
Enter coeefficirnts of  second equation : 2 1 4
Enter coeefficirnts of  third equation : 3 3 4
Enter the three constants in order: -1 2 1
[ 1.66666667e+00 -1.33333333e+00 -2.22044605e-16]
uding inverse method =  [ 1.66666667 -1.33333333  0.        ]
```

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 [ ]:
```python
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.]
```