Solving System of Equations

A linear equation in variables $x_1, x_2, \ldots, x_n$ is an equation of the form $a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$, where $a_1, a_2, \ldots, a_n$ and b are constants. The constant a_i is called the coefficient of x_i.

A system of linear equations is a finite collection of linear equations in same variables.
The following is a system of m linear equations 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 matrix notation a linear system is $AX = B$, where
$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ & \ldots & & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$ is the coefficient matrix,
$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$.

Python provides the function numpy.linalg.solve() to solve the system 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 [15]:

```
import numpy as np
```

In [16]:

```
A=np.array([[1,2,-1],[2,1,4],[3,3,4]])
B=np.array([1,2,1])
np.linalg.solve(A,B)
```

Out[16]:

```
array([ 7., -4., -2.])
```

Here we need to observe that the matrix B is defined as a 1×3 array, whereas the same function will not work if B is defined as a 1×3 matrix.

In [17]:

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-3d6c91cbf3f4> in <module>()
      1 A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
      2 B=np.matrix([1,2,1])
----> 3 np.linalg.solve(A,B)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    392     signature = 'DD->D' if isComplexType(t) else 'dd->d'
    393     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 394     r = gufunc(a, b, signature=signature, extobj=extobj)
    395 
    396     return wrap(r.astype(result_t, copy=False))

ValueError: solve: Input operand 1 has a mismatch in its core dimension 0, w
ith gufunc signature (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 the linear system of equation.

In [32]:

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

Out[32]:

```
matrix([[ 7.],
        [-4.],
        [-2.]])
```

In [33]:

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

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-33-aec705622774> in <module>()
      1 A=np.matrix([[1,-1,1,-1],[1,-1,1,1],[4,-4,4,0],[-2,2,-2,1]])
      2 B=np.matrix([[2],[0],[4],[-3]])
----> 3 np.linalg.solve(A,B)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    392     signature = 'DD->D' if isComplexType(t) else 'dd->d'
    393     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 394     r = gufunc(a, b, signature=signature, extobj=extobj)
    395 
    396     return wrap(r.astype(result_t, copy=False))

C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _raise_
linalgerror_singular(err, flag)
     87 
     88 def _raise_linalgerror_singular(err, flag):
---> 89     raise LinAlgError("Singular matrix")
     90 
     91 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

In [6]:

```
A=np.matrix([[2,3,1],[1,-1,-2]])
B=np.matrix([[0],[0]])
np.linalg.solve(A,B)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-6-bd23729d3cf3> in <module>()
      1 A=np.matrix([[2,3,1],[1,-1,-2]])
      2 B=np.matrix([[0],[0]])
----> 3 np.linalg.solve(A,B)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    379     a, _ = _makearray(a)
    380     _assertRankAtLeast2(a)
--> 381     _assertNdSquareness(a)
    382     b, wrap = _makearray(b)
    383     t, result_t = _commonType(a, b)

C:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _assert
NdSquareness(*arrays)
    213         m, n = a.shape[-2:]
    214         if m != n:
--> 215             raise LinAlgError('Last 2 dimensions of the array must b
e square')
    216 
    217 def _assertFinite(*arrays):

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

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

System of Homogenous Linear Equations

The linear system of equations of the form $AX = 0$ is called system of homogenous linear equations.
The n-tuple $(0, 0, \ldots, 0)$ is a trivial solution of the system.
The homogeneous system of m equations $AX = 0$ in n unknowns has a non trivial solution if and only if the rank of the matrix A is less than n. Further if $\rho(A) = r < n$, then the system possesses $(n - r)$ linearly independent solutions.

Q. Check whether the following system of homogenous linear equation has non-trivial solution.
$x_1 + 2x_2 - x_3 = 0$
$2x_1 + x_2 + 4x_3 = 0$
$3x_1 + 3x_2 + 4x_3 = 0$

In [30]:

```
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[0],[0],[0]])
r=np.linalg.matrix_rank(A)
n=A.shape[1]
if (r==n):
    print("System has trivial solution")
else:
    print("System has", n-r, "non-trivial solution(s)")
```

```
System has trivial solution
```

In [31]:

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

Out[31]:

```
matrix([[ 0.],
        [ 0.],
        [-0.]])
```

Q. Check whether the following system of homogenous linear equation has non-trivial solution.

$$x_1 + 2x_2 - x_3 = 0$$
$$2x_1 + x_2 + 4x_3 = 0$$
$$x_1 - x_2 + 5x_3 = 0$$

In [29]:

```
A=np.matrix([[1,2,-1],[2,1,4],[1,-1,5]])
B=np.matrix([[0],[0],[0]])
r=np.linalg.matrix_rank(A)
n=A.shape[1]
if (r==n):
    print("System has trivial solution")
else:
    print("System has", n-r, "non-trivial solution(s)")
```

```
System has 1 non-trivial solution(s)
```

System of Non-homogenous Linear Equations

The linear system of equations of the form $AX = B$ is called system of non-homogenous linear equations if not all elements in B are zeros.
The non homogeneous system of m equations $AX = B$ in n unknowns is consistent (has a solution) if and only if the $\rho(A) = \rho([A|B])$.
If $\rho(A) = \rho([A|B])$, and

1. $\rho(A) = n$, then system has unique solution

2. $\rho(A) < n$, then system has infintely many solutions.

If $\rho(A) \neq \rho([A|B])$, then the system is inconsistent.

Q. Examine the consistency of the following system of equations and solve if consistent

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

In [28]:

```
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B), axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[1]
if (rA==rAB):
    if (rA==n):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
        print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
The system has unique solution
[[ 7.]
 [-4.]
 [-2.]]
```

Q. Examine the consistency of the following system of equations and solve if consistent

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

In [27]:

```
A=np.matrix([[1,2,-1],[2,1,5],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B), axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[1]
if (rA==rAB):
    if (rA==n):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
        print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
The system of equations is inconsistent
```

Q. Examine the consistency of the following system of equations and solve if consistent

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

In [26]:

```
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]])
AB=np.concatenate((A,B), axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[1]
if (rA==rAB):
    if (rA==n):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
        print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
The system has infinitely many solutions
```

Gauss Jordan Method

Q. Write a program to find the solution of a sysltem of linear equations using Gauss Jordan method.

In [25]:

```
m = int(input("Enter the number of equations:"))
n = int(input("Enter the number of variables:"))
print("Enter the entries of the coefficient matrix in a single line (separated by space): "
elements = list(map(int, input().split()))
A=np.matrix(elements).reshape(m, n)
print("Enter the entries of the matrix B in a single line (separated by space): ")
Belements = list(map(int, input().split()))
B=np.matrix(Belements).reshape(m, 1)
AB=np.concatenate((A,B), axis=1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A.shape[0]
if (rA==rAB):
    if (rA==n):
        print("The system has unique solution")
        for i in range (n):
            if (AB[i,i]==0):
                k=i+1
                while (AB[k,i]==0):
                    k=k+1
                AB[[i,k]]=AB[[k,i]]
            AB[i]=AB[i]/AB[i,i]
            for j in range (n):
                if (j!=i):
                    AB[j]= AB[j]-AB[j,i]*AB[i]
        Solution=AB[:n,n]
        print("Solution:")
        print(Solution)
    else:
        print("The system has infinitely many solutions")
else:
    print("The system of equations is inconsistent")
```

```
Enter the number of equations:2
Enter the number of variables:2
Enter the entries of the coefficient matrix in a single line (separated by s
pace):
1 2 3 4
Enter the entries of the matrix B in a single line (separated by space):
1 2
The system has unique solution
Solution:
[[1]
 [0]]
```

Cramer's Rule

Q. Write a prgram to solve a system of linear equations using Cramer's rule.

In [22]:

```
import copy as cp
```

In [23]:

```
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
Ax = cp.deepcopy(A)
Ay = cp.deepcopy(A)
Az = cp.deepcopy(A)
Ax[:, 0]=B
Ay[:, 1]=B
Az[:, 2]=B
x=np.linalg.det(Ax)/np.linalg.det(A)
y=np.linalg.det(Ay)/np.linalg.det(A)
z=np.linalg.det(Az)/np.linalg.det(A)
print(" x={:.2f} \n y={:.2f} \n z={:.2f}".format(x,y,z))
```

```
 x=7.00
 y=-4.00
 z=-2.00
```

Gauss Elimination Method

Q. Write a program to find the solution of a sysltem of linear equations using Gauss elimination method.

In [21]:

```
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B), axis=1)
m=A.shape[0]
n=A.shape[1]
for i in range (m):
    if (AB[i,i]==0):
        k=i+1
        while (AB[k,i]==0):
            k=k+1
        AB[[i,k]]=AB[[k,i]]
    AB[i]=AB[i]/AB[i,i]
    for j in range (i+1,m):
        AB[j]= AB[j]-AB[j,i]*AB[i]
z = AB[2,3]/AB[2,2]
y = (AB[1,3]-z*AB[1,2])/AB[1,1]
x = (AB[0,3]-z*AB[0,2]-y*AB[0,1])/AB[0,0]
print("Solution:")
print(" x={:.2f} \n y={:.2f} \n z={:.2f}".format(x,y,z))
```

```
Solution:
 x=7.00
 y=-4.00
 z=-2.00
```

In [ ]: