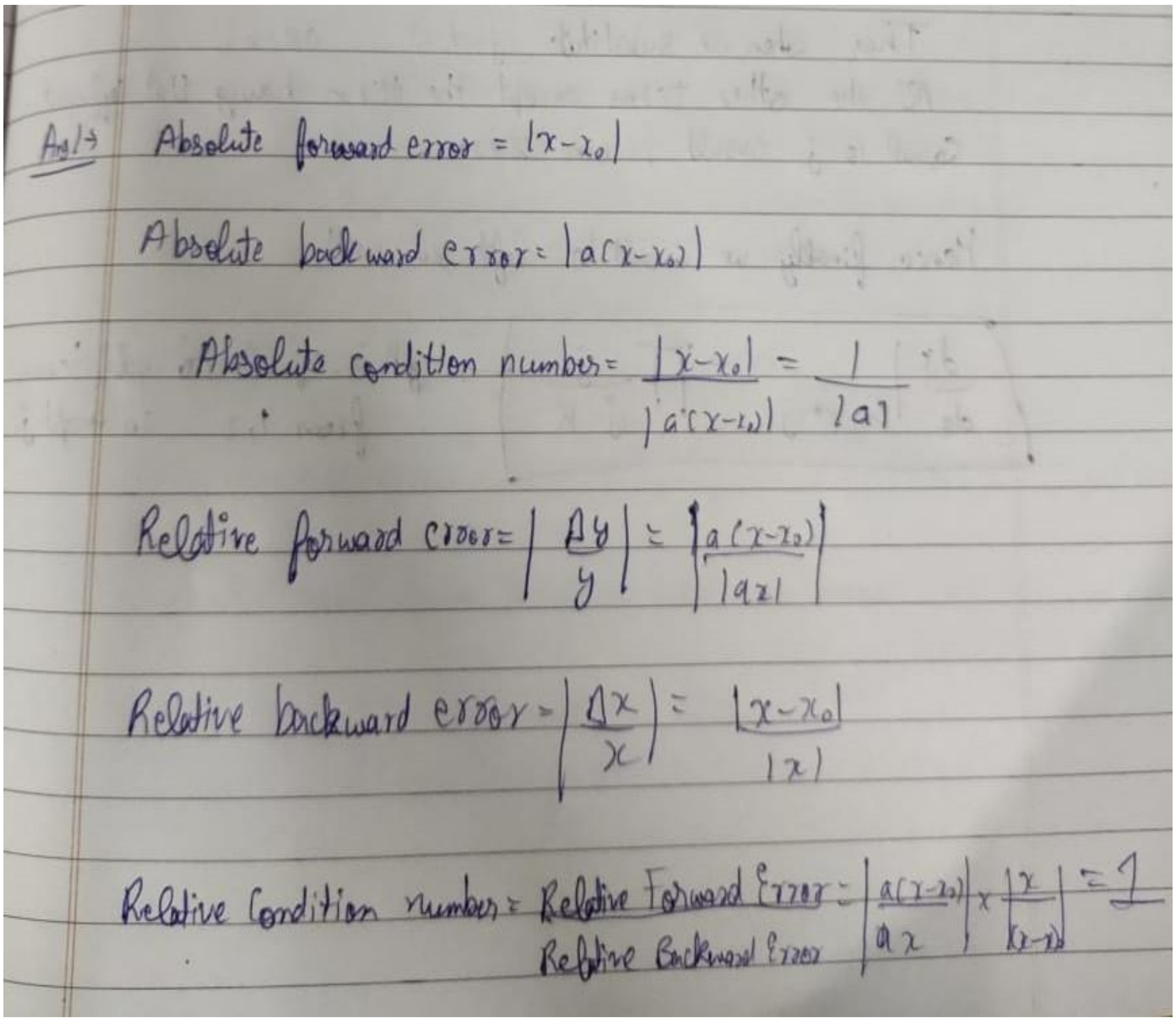
VAIBHAV RAJPAL

2020146

Ans 1:



Ans1→ Absolute forward error = $|x - x_0|$

Absolute backward error = $|a(x - x_0)|$

Absolute condition number = $\dfrac{|x - x_0|}{|a(x - x_0)|} = \dfrac{1}{|a|}$

Relative forward error = $\left|\dfrac{\Delta y}{y}\right| = \left|\dfrac{a(x - x_0)}{|ax|}\right|$

Relative backward error = $\left|\dfrac{\Delta x}{x}\right| = \dfrac{|x - x_0|}{|x|}$

Relative Condition number = $\dfrac{\text{Relative Forward Error}}{\text{Relative Backward Error}} = \left|\dfrac{a(x - x_0)}{ax}\right| \times \left|\dfrac{x}{(x - x_0)}\right| = \underline{1}$

Ans 2:

Ans 2>

(a) $(x-1)^{\alpha}$

Absolute condition number = $|f'(x)|$

$$f(x) = (x-1)^{\alpha}$$
$$f'(x) = \alpha(x-1)^{\alpha-1}$$

$\Rightarrow |\alpha(x-1)^{\alpha-1}|$

This function have larger absolute condition number for larger x which is $x = \pm\infty$, the absolute condition number would be large.

Relative Condition number = $\left|\frac{x \cdot f'(x)}{f(x)}\right|$

$$\Rightarrow \left|\frac{x \cdot \alpha \cdot (x-1)^{\alpha-1}}{(x-1)^{\alpha}}\right|$$

$$\Rightarrow \left|\frac{\alpha x}{x-1}\right|$$

$$\Rightarrow \left|\frac{\alpha(x-1+1)}{x-1}\right|$$

$$\Rightarrow \left|\frac{\alpha(x-1)}{x-1} + \frac{\alpha}{x-1}\right|$$

$$\Rightarrow \left|\alpha + \frac{\alpha}{x-1}\right|$$

As α is a constant the relative condition number would give a large value for $x = 1$.

(b) $\ln(x)$

Absolute condition number $= |f'(x)|$

$f(x) = \ln x$

$f'(x) = \frac{1}{x}$

Absolute condition number $= \left|\frac{1}{x}\right|$

It will be large for $x = 0$

Relative condition number $= \left|\frac{x \cdot f'(x)}{f(x)}\right|$

$\Rightarrow \left|\frac{x \times 1}{x \times \ln x}\right| = \left|\frac{1}{\ln x}\right|$

It will be large for $x = 1$, it cannot be -1 because negative value are not included in domain of $\ln x$

(c) $x^{-1} e^{x}$

Absolute condition number $= |f'(x)|$

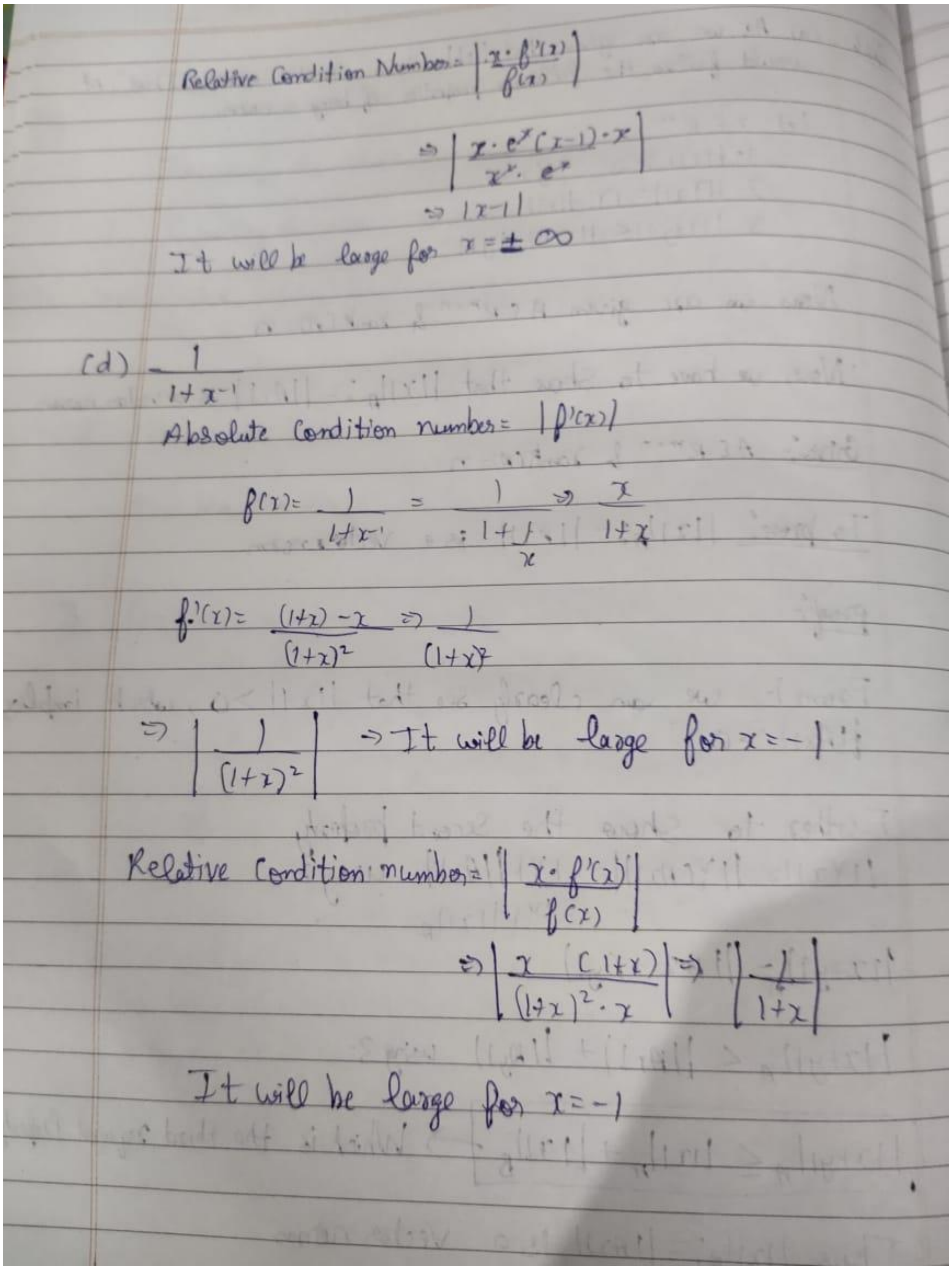
$f(x) = x^{-1} e^{x}$

$f'(x) = -x^{-2} e^{x} + x^{-1} e^{x}$

$\Rightarrow \frac{e^{x}}{x} - \frac{e^{x}}{x^2} \Rightarrow \frac{x e^{x} - e^{x}}{x^2} \Rightarrow \frac{e^{x}(x-1)}{x^2}$

$\Rightarrow \left|\frac{e^{x}(x-1)}{x^2}\right|$

It will be large for $x = 0, +\infty$



Relative Condition Number = $\left|\frac{x \cdot f'(x)}{f(x)}\right|$

$$\Rightarrow \left|\frac{x \cdot e^{x}(x-1) \cdot x}{x^{2} \cdot e^{x}}\right|$$

$$\Rightarrow |x-1|$$

It will be large for $x = \pm \infty$

(d) $\frac{1}{1+x^{-1}}$

Absolute Condition number = $|f'(x)|$

$$f(x) = \frac{1}{1+x^{-1}} = \frac{1}{1+\frac{1}{x}} \Rightarrow \frac{x}{1+x}$$

$$f'(x) = \frac{(1+x)-x}{(1+x)^2} \Rightarrow \frac{1}{(1+x)^2}$$

$\Rightarrow \left|\frac{1}{(1+x)^2}\right|$ $\Rightarrow$ It will be large for $x = -1$

Relative Condition number = $\left|\frac{x \cdot f'(x)}{f(x)}\right|$

$$\Rightarrow \left|\frac{x \quad (1+x)}{(1+x)^2 \cdot x}\right| \Rightarrow \left|\frac{1}{1+x}\right|$$

It will be large for $x = -1$

Ans 3:

Ans 3 → (a) $f(x(\varepsilon)) + \varepsilon\, p(x(\varepsilon)) = 0$

Differentiating the above fxn we get

~~$f'(x(\varepsilon)) + \varepsilon p'(x(\varepsilon)) + p(x(\varepsilon))$~~

$$f'(x(\varepsilon))\cdot x'(\varepsilon) + \varepsilon\, p'(x(\varepsilon))\cdot x'(\varepsilon) + p(x(\varepsilon)) = 0$$

$$f'(x(\varepsilon))\frac{dx}{d\varepsilon} + \varepsilon\, p'(x(\varepsilon))\frac{dx}{d\varepsilon} + p(x(\varepsilon)) = 0$$

$\underline{\underline{\varepsilon = 0}}$

$$\frac{dx}{d\varepsilon}\cdot f'(x(0)) + 0\cdot p'(x(\varepsilon))\frac{dx}{d\varepsilon} + p(x(0)) = 0$$

As $x(0) = x^*$

$$\frac{dx}{d\varepsilon}\cdot f'(x^*) + p(x^*) = 0$$

$$\boxed{\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0} = -\frac{p(x^*)}{f'(x^*)}}$$

(b) Here we are given $p(x) = x^{19}$ & $f(x) = (x-1)(x-2)(x-3)\ldots$

Using the result from previous part we know

$$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0} = -\frac{p(x^*)}{f'(x^*)}$$

$p(x^*) = (x^*)^{19}$

$p'(x^*) = (x^*-2)\ldots(x^*-20) + (x^*-1)\ldots(x^*-20) + \ldots$ [Which basically means it is product of all terms leaving one of them]

$$\left.\frac{dx}{d\epsilon}\right|_{\epsilon=0} = \frac{-(x^*)^{19}}{(x^*-2)\ldots(x^*-20)+(x^*-1)\ldots(x^*-20)+\ldots+(x^*-1)\ldots(x^*-20)}$$

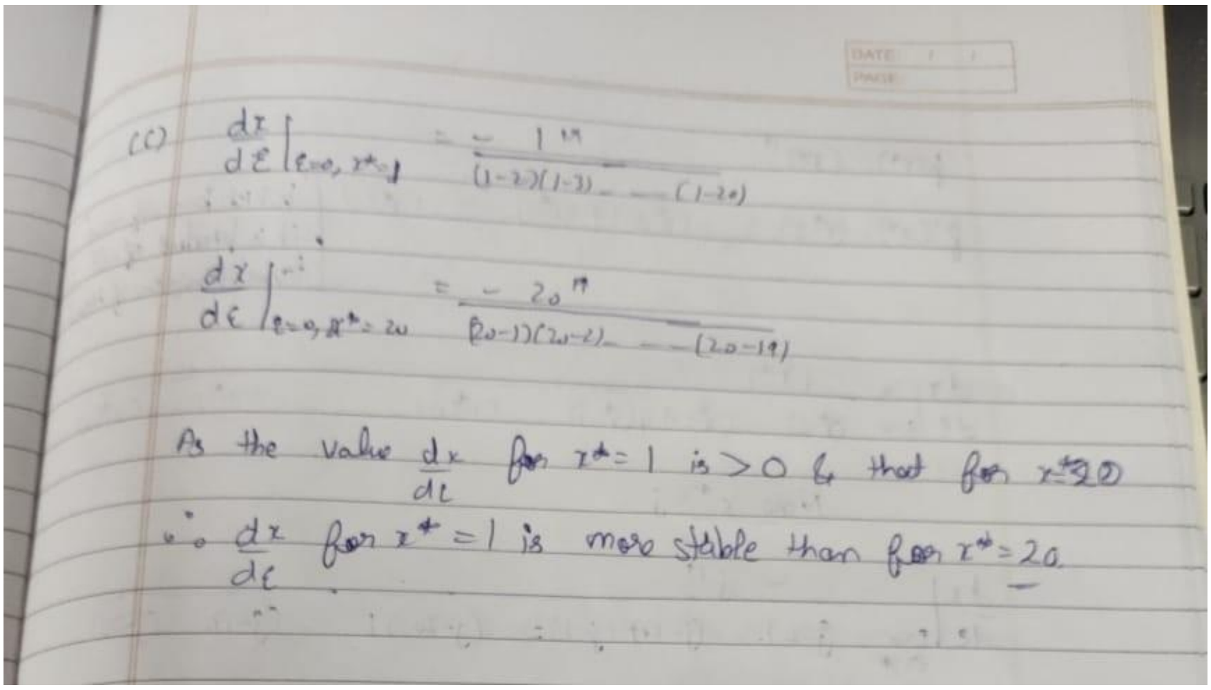
<u>Now $x^* = j$</u>

$$\left.\frac{dx}{d\epsilon}\right|_{\substack{\epsilon=0\\ x^*=j}} = \frac{-j^{19}}{(j-2)\ldots(j-20)+(j-1)\ldots(j-20)+\ldots(j-1)\ldots(j-20)}$$

Thus, when we substitute $j = 1, 2, \ldots$ or 20
All the other terms except the term having that coefficient equal to j would become zero.

Hence, finally we would be left out with

$$\boxed{\left.\frac{dx}{d\epsilon}\right|_{\epsilon=0,\, x^*=j} = -\prod \frac{j}{j-K}} \rightarrow$$ jK is equal to all terms from $1, 2, \ldots 20$ except j.



(c) $\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0,\, x^*=1} = \frac{-1^{19}}{(1-2)(1-3) \text{---} (1-20)}$

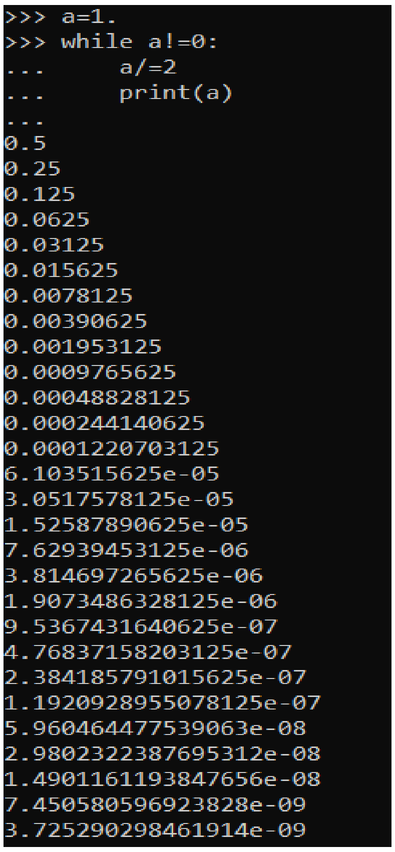
$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0,\, x^*=20} = \frac{-20^{19}}{(20-1)(20-2) \text{---} (20-19)}$

As the value $\frac{dx}{d\varepsilon}$ for $x^*=1$ is > 0 & that for $x^*=20$

$\therefore$ $\frac{dx}{d\varepsilon}$ for $x^*=1$ is more stable than for $x^*=20$.

Ans 4:

a. Output from the code is as follows:

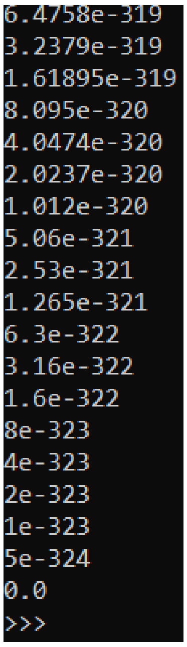


```
>>> a=1.
>>> while a!=0:
...     a/=2
...     print(a)
...
0.5
0.25
0.125
0.0625
0.03125
0.015625
0.0078125
0.00390625
0.001953125
0.0009765625
0.00048828125
0.000244140625
0.0001220703125
6.103515625e-05
3.0517578125e-05
1.52587890625e-05
7.62939453125e-06
3.814697265625e-06
1.9073486328125e-06
9.5367431640625e-07
4.76837158203125e-07
2.384185791015625e-07
1.1920928955078125e-07
5.960464477539063e-08
2.9802322387695312e-08
1.4901161193847656e-08
7.450580596923828e-09
3.725290298461914e-09
```

.

.

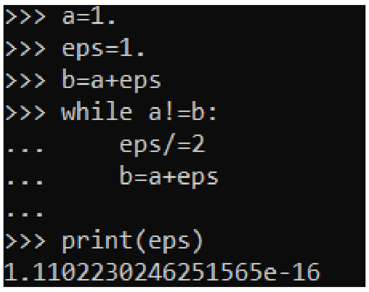
.



```
6.4758e-319
3.2379e-319
1.61895e-319
8.095e-320
4.0474e-320
2.0237e-320
1.012e-320
5.06e-321
2.53e-321
1.265e-321
6.3e-322
3.16e-322
1.6e-322
8e-323
4e-323
2e-323
1e-323
5e-324
0.0
>>>
```

Here starting from 1 the number get's half in every iteration and before it tends to zero it reaches the underflow value of floating point system

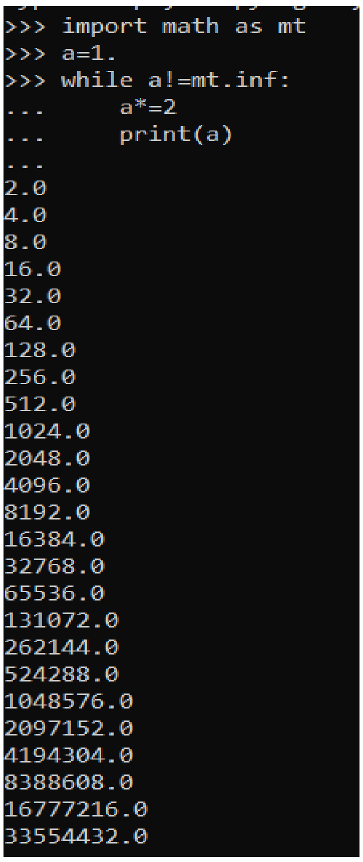
b. The output is as follows:



```
>>> a=1.
>>> eps=1.
>>> b=a+eps
>>> while a!=b:
...     eps/=2
...     b=a+eps
...
>>> print(eps)
1.1102230246251565e-16
```

Here also when we started 'eps' with 1 we are reducing it to its half every time and 'a' would become equal to 'b' and this would depend upon the machine precision as it is the upper bound on the error caused by rounding off in floating point numbers and here also round off would occur because of the precision up to 16 decimal places

c. The output is as follows:



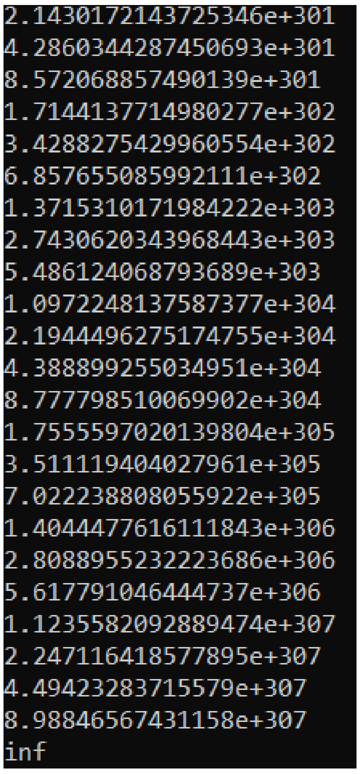
```
>>> import math as mt
>>> a=1.
>>> while a!=mt.inf:
...     a*=2
...     print(a)
...
2.0
4.0
8.0
16.0
32.0
64.0
128.0
256.0
512.0
1024.0
2048.0
4096.0
8192.0
16384.0
32768.0
65536.0
131072.0
262144.0
524288.0
1048576.0
2097152.0
4194304.0
8388608.0
16777216.0
33554432.0
```

.

.

.

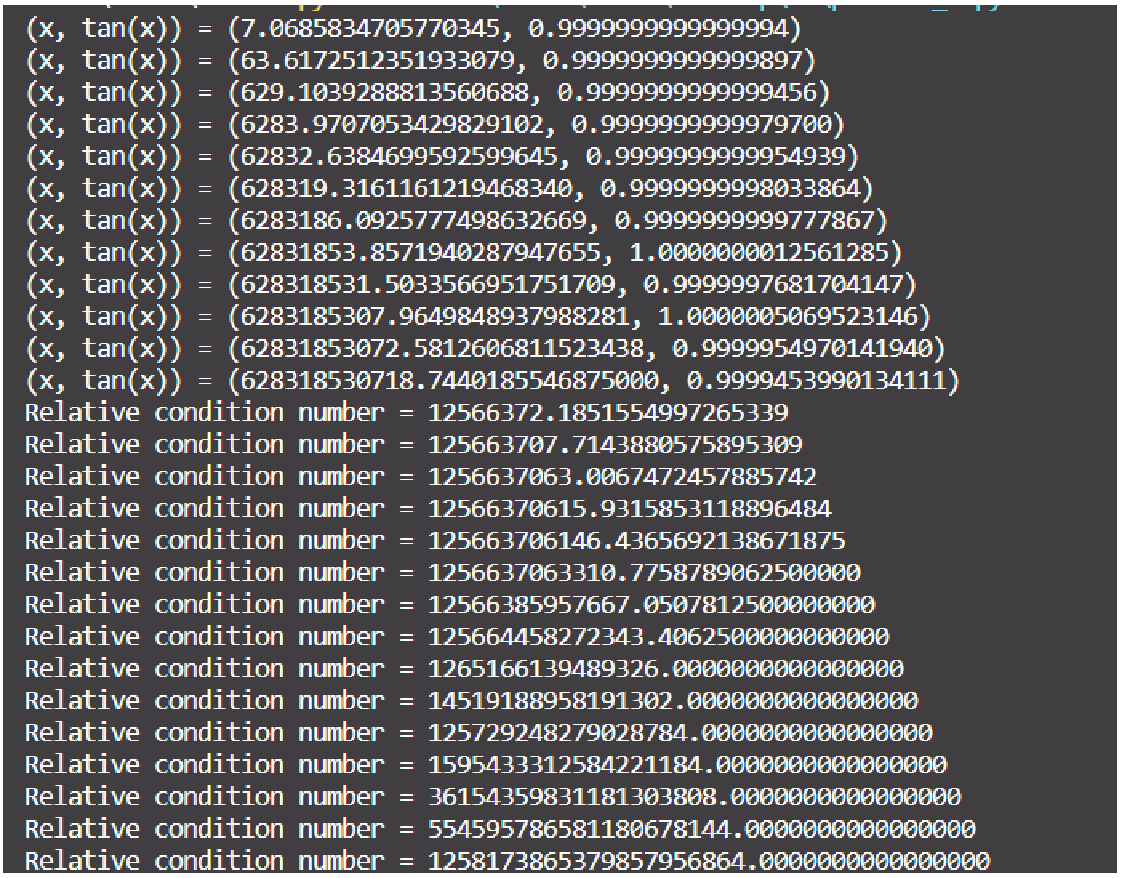
.



```
2.1430172143725346e+301
4.2860344287450693e+301
8.572068857490139e+301
1.7144137714980277e+302
3.4288275429960554e+302
6.857655085992111e+302
1.3715310171984222e+303
2.7430620343968443e+303
5.486124068793689e+303
1.0972248137587377e+304
2.1944496275174755e+304
4.388899255034951e+304
8.777798510069902e+304
1.7555597020139804e+305
3.511119404027961e+305
7.022238808055922e+305
1.4044477616111843e+306
2.8088955232223686e+306
5.617791046444737e+306
1.1235582092889474e+307
2.247116418577895e+307
4.49423283715579e+307
8.98846567431158e+307
inf
```

Every time a is being multiplied by 2 and because of which it is increasing exponentially and as a result of which overflow occurs and beyond which it cannot print anything and thus terminate towards infinity.

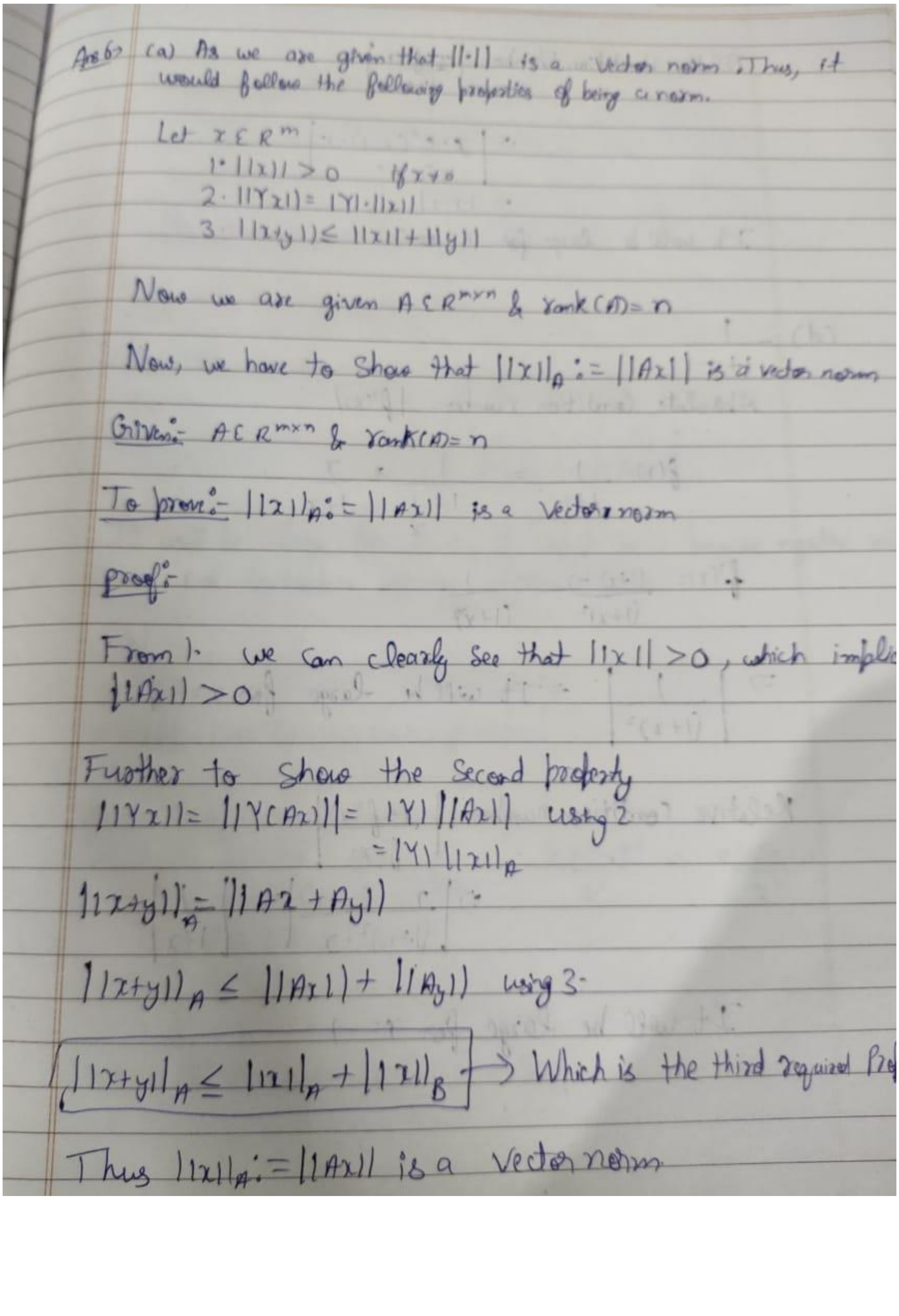
Ans 5 : Output is as follows:



```
 (x, tan(x)) = (7.0685834705770345, 0.9999999999999994)
 (x, tan(x)) = (63.6172512351933079, 0.9999999999999897)
 (x, tan(x)) = (629.1039288813560688, 0.9999999999999456)
 (x, tan(x)) = (6283.9707053429829102, 0.9999999999979700)
 (x, tan(x)) = (62832.6384699592599645, 0.9999999999954939)
 (x, tan(x)) = (628319.3161161219468340, 0.9999999998033864)
 (x, tan(x)) = (6283186.0925777498632669, 0.9999999999777867)
 (x, tan(x)) = (62831853.8571940287947655, 1.0000000012561285)
 (x, tan(x)) = (628318531.5033566951751709, 0.9999997681704147)
 (x, tan(x)) = (6283185307.9649848937988281, 1.0000005069523146)
 (x, tan(x)) = (62831853072.5812606811523438, 0.9999954970141940)
 (x, tan(x)) = (628318530718.7440185546875000, 0.9999453990134111)
 Relative condition number = 12566372.1851554997265339
 Relative condition number = 125663707.7143880575895309
 Relative condition number = 1256637063.0067472457885742
 Relative condition number = 12566370615.9315853118896484
 Relative condition number = 125663706146.4365692138671875
 Relative condition number = 1256637063310.7758789062500000
 Relative condition number = 12566385957667.0507812500000000
 Relative condition number = 125664458272343.4062500000000000
 Relative condition number = 1265166139489326.0000000000000000
 Relative condition number = 14519188958191302.0000000000000000
 Relative condition number = 125729248279028784.0000000000000000
 Relative condition number = 1595433312584221184.0000000000000000
 Relative condition number = 36154359831181303808.0000000000000000
 Relative condition number = 554595786581180678144.0000000000000000
 Relative condition number = 1258173865379857956864.0000000000000000
```

As we can see here that x is pie/4 + a multiple of 2*pie which simplifies to:
tan(pie/4 + 2*pie*n)=tan(pie/4)
Which in actuality have a value 1 and should be that only for all the values of x but because there is a precision of 16 places for floating point number the result is not accurately 1 it is either slightly less than or more than 1 because of this only and there would be a difference of 10^-16 between the actual and computed value.

Ans 6:



Ans 6> (a) As we are given that $||\cdot||$ is a vector norm. Thus, it would follow the following properties of being a norm.

Let $x \in R^m$

1. $||x|| > 0$ if $x \neq 0$
2. $||\gamma x|| = |\gamma| \cdot ||x||$
3. $||x+y|| \leq ||x|| + ||y||$

Now we are given $A \in R^{m\times n}$ & $\text{rank}(A) = n$

Now, we have to show that $||x||_A := ||Ax||$ is a vector norm

Given:- $A \in R^{m\times n}$ & $\text{rank}(A) = n$

To prove:- $||x||_A := ||Ax||$ is a vector norm

proof:-

From 1. we can clearly see that $||x|| > 0$, which implic

$||Ax|| > 0$

Further to show the second property

$||\gamma x|| = ||\gamma(Ax)|| = |\gamma| \, ||Ax||$ using 2

$= |\gamma| \, ||x||_A$

$||x+y||_A = ||Ax + Ay||$

$||x+y||_A \leq ||Ax|| + ||Ay||$ using 3.

$\boxed{||x+y||_A \leq ||x||_A + ||x||_B} \rightarrow$ Which is the third required Pro

Thus $||x||_A := ||Ax||$ is a vector norm

(b) $\|A\|_F := \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2}$

To Verify the three properties of norm.

1. Expanding the above expression

$\|A\| = \sqrt{|a_{00}|^2 + |a_{01}|^2 + \ldots + |a_{mn}|^2} > 0$ [As the square root of sum of square terms is always +ve]

2. $\|\gamma x\| = |\gamma|\,\|x\|$

$\Rightarrow \sqrt{|\alpha a_{00}|^2 + |\alpha a_{01}|^2 + \ldots + |\alpha a_{mn}|^2} = |\alpha|\sqrt{|a_{00}|^2 + |a_{01}|^2 + \ldots + |a_{mn}|^2}$

$= |\alpha|\,\|A\|_2$

3. $\|x+y\|_2 \leq \|x\|_2 + \|y\|_2$

$$\Rightarrow \left(\sum_{i=1}^{m}\sum_{j=1}^{n}\left(|x_{ij}|^2 + |y_{ij}|^2\right)\right)^{1/2} \leq \left(\sum_{i=1}^{m}\sum_{j=1}^{n}|x_{ij}|^2\right) + \left(\sum_{i=1}^{m}\sum_{j=1}^{n}|y_{ij}|^2\right)$$

$\leq \|x\|_2 + \|y\|_2$

(c) $u \in R^m$ & $v \in R^n$

$\boxed{E = uv^T}$ $\quad ||E||_F = ||E||_2 = ||u||_2 ||v||_2$

Given:- $u \in R^m$ & $v \in R^n$ & $E = uv^T$

To Prove:- $||E||_F = ||E||_2 = ||u||_2 ||v||_2$

Proof:-

Using the Sub-multiplicative property of the norms, we know that

$$||AB|| \leq ||A|| \cdot ||B||$$

$\Rightarrow ||uv^T|| \leq ||u||_2 \cdot ||v^T||_2$

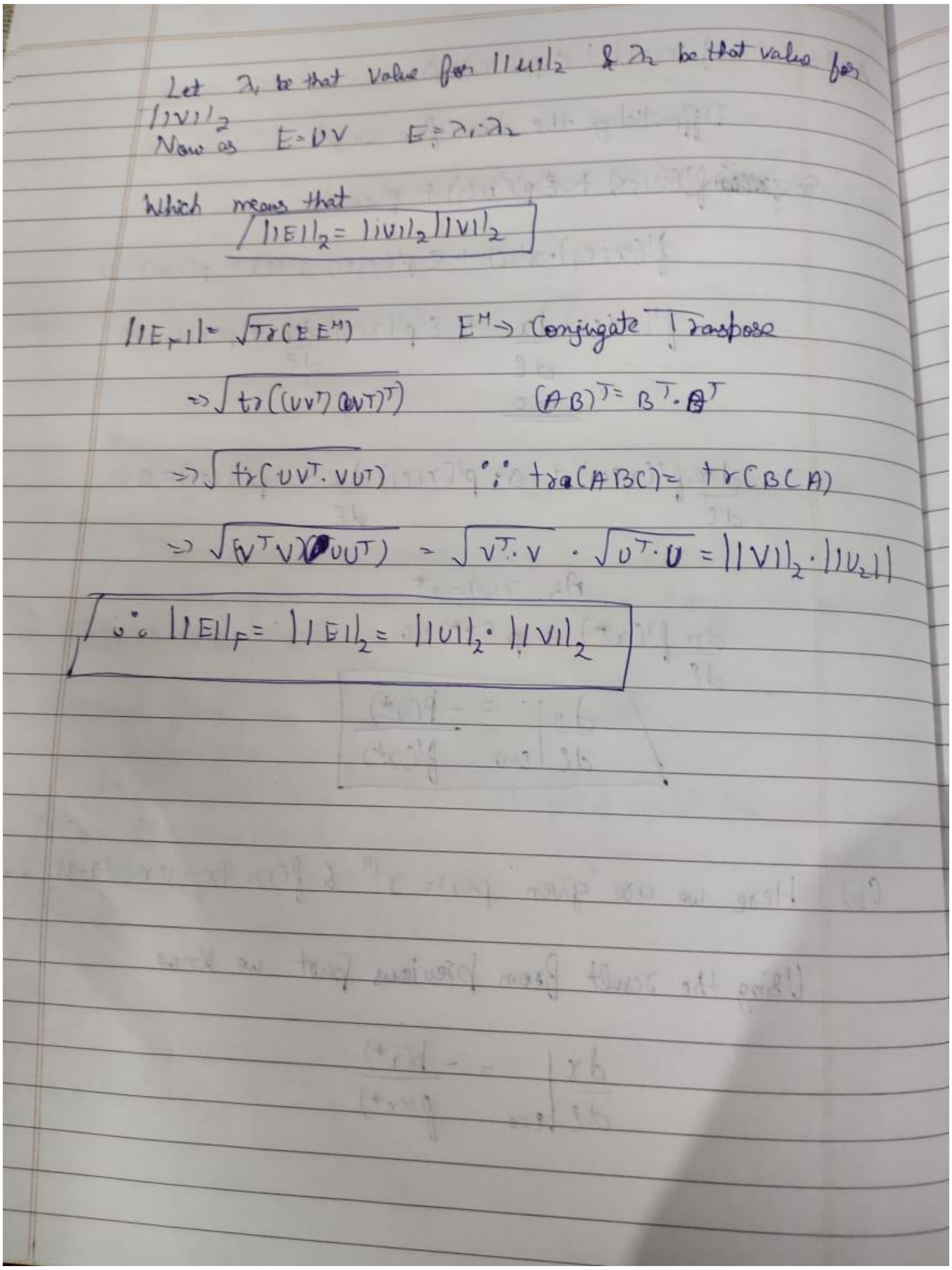
$\Rightarrow ||E|| \leq ||u||_2 \cdot ||v^T||_2$

$\because$ We know that $||v|| = ||v^T||$

$\Rightarrow ||E|| \leq ||u|| \cdot ||v||$

Now considering the case for second Norm. We know that we can find second Norm as

$\Rightarrow \sqrt{\lambda_{largest}(AA^T)} \rightarrow$ Which is largest eigen value of a matrix which a product of itself & it's traspose



Let λ_1 be that value for $||u||_2$ & λ_2 be that value for $||v||_2$

Now as $E = UV$ $\quad E = \lambda_1 \cdot \lambda_2$

Which means that

$$\boxed{||E||_2 = ||u||_2 ||v||_2}$$

$||E_F|| = \sqrt{Tr(E E^H)}$ $\quad E^H \rightarrow$ Conjugate Transpose

$\Rightarrow \sqrt{tr((UV^T)(UV^T)^T)}$ $\quad (AB)^T = B^T \cdot A^T$

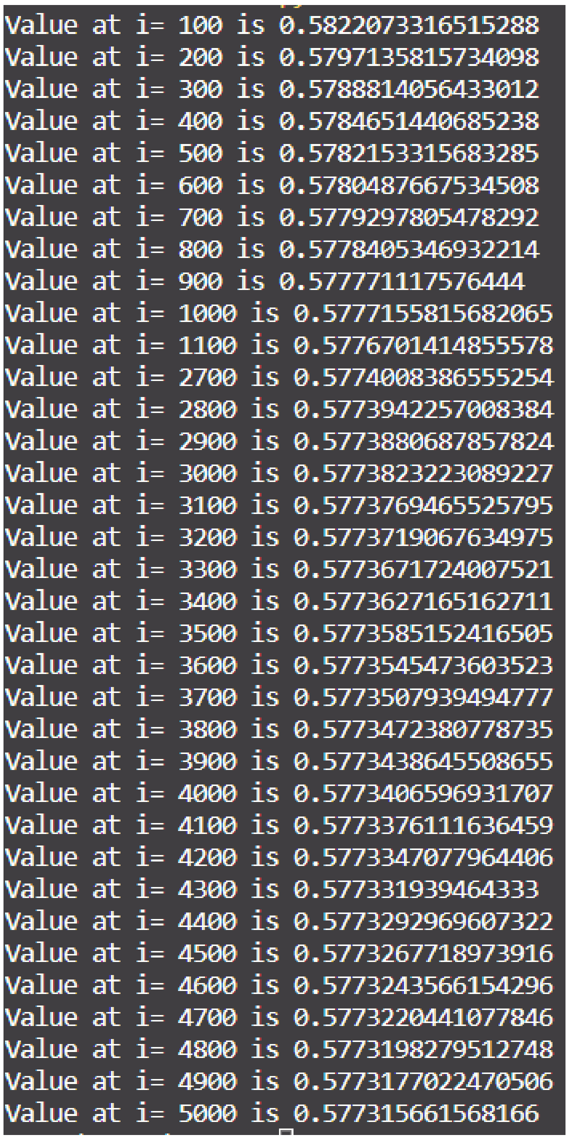
$\Rightarrow \sqrt{tr(UV^T \cdot VU^T)}$ $\quad \because tra(ABC) = tr(BCA)$

$\Rightarrow \sqrt{(V^T V)(UU^T)} = \sqrt{V^T \cdot V} \cdot \sqrt{U^T \cdot U} = ||V||_2 \cdot ||U_2||$

$$\boxed{\therefore ||E||_F = ||E||_2 = ||U||_2 \cdot ||V||_2}$$

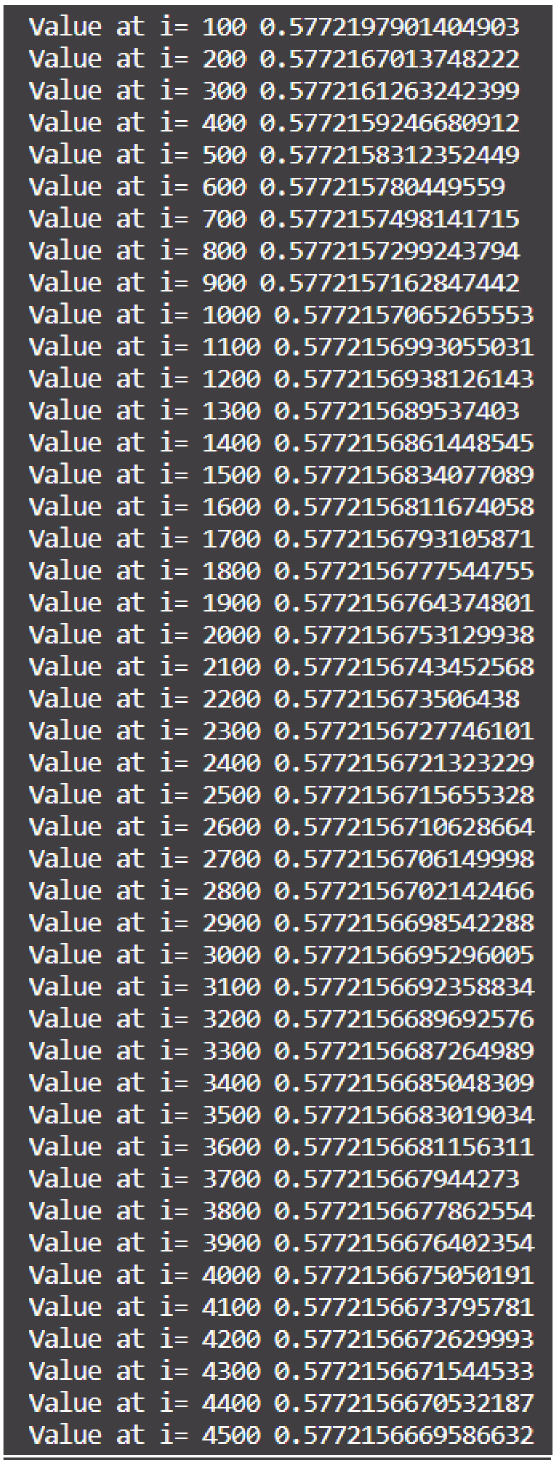
Ans 7:

Output for first code:

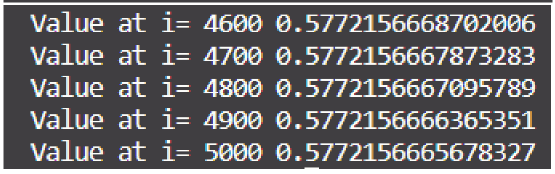


```
Value at i= 100 is 0.5822073316515288
Value at i= 200 is 0.5797135815734098
Value at i= 300 is 0.5788814056433012
Value at i= 400 is 0.5784651440685238
Value at i= 500 is 0.5782153315683285
Value at i= 600 is 0.5780487667534508
Value at i= 700 is 0.5779297805478292
Value at i= 800 is 0.5778405346932214
Value at i= 900 is 0.577771117576444
Value at i= 1000 is 0.5777155815682065
Value at i= 1100 is 0.5776701414855578
Value at i= 2700 is 0.5774008386555254
Value at i= 2800 is 0.5773942257008384
Value at i= 2900 is 0.5773880687857824
Value at i= 3000 is 0.5773823223089227
Value at i= 3100 is 0.5773769465525795
Value at i= 3200 is 0.5773719067634975
Value at i= 3300 is 0.5773671724007521
Value at i= 3400 is 0.5773627165162711
Value at i= 3500 is 0.5773585152416505
Value at i= 3600 is 0.5773545473603523
Value at i= 3700 is 0.5773507939494777
Value at i= 3800 is 0.5773472380778735
Value at i= 3900 is 0.5773438645508655
Value at i= 4000 is 0.5773406596931707
Value at i= 4100 is 0.5773376111636459
Value at i= 4200 is 0.5773347077964406
Value at i= 4300 is 0.577331939464333
Value at i= 4400 is 0.5773292969607322
Value at i= 4500 is 0.5773267718973916
Value at i= 4600 is 0.5773243566154296
Value at i= 4700 is 0.5773220441077846
Value at i= 4800 is 0.5773198279512748
Value at i= 4900 is 0.5773177022470506
Value at i= 5000 is 0.577315661568166
```

Output for second code:



```
Value at i= 100 0.5772197901404903
Value at i= 200 0.5772167013748222
Value at i= 300 0.5772161263242399
Value at i= 400 0.5772159246680912
Value at i= 500 0.5772158312352449
Value at i= 600 0.577215780449559
Value at i= 700 0.5772157498141715
Value at i= 800 0.5772157299243794
Value at i= 900 0.5772157162847442
Value at i= 1000 0.5772157065265553
Value at i= 1100 0.5772156993055031
Value at i= 1200 0.5772156938126143
Value at i= 1300 0.577215689537403
Value at i= 1400 0.5772156861448545
Value at i= 1500 0.5772156834077089
Value at i= 1600 0.5772156811674058
Value at i= 1700 0.5772156793105871
Value at i= 1800 0.5772156777544755
Value at i= 1900 0.5772156764374801
Value at i= 2000 0.5772156753129938
Value at i= 2100 0.5772156743452568
Value at i= 2200 0.577215673506438
Value at i= 2300 0.5772156727746101
Value at i= 2400 0.5772156721323229
Value at i= 2500 0.5772156715655328
Value at i= 2600 0.5772156710628664
Value at i= 2700 0.5772156706149998
Value at i= 2800 0.5772156702142466
Value at i= 2900 0.5772156698542288
Value at i= 3000 0.5772156695296005
Value at i= 3100 0.5772156692358834
Value at i= 3200 0.5772156689692576
Value at i= 3300 0.5772156687264989
Value at i= 3400 0.5772156685048309
Value at i= 3500 0.5772156683019034
Value at i= 3600 0.5772156681156311
Value at i= 3700 0.577215667944273
Value at i= 3800 0.5772156677862554
Value at i= 3900 0.5772156676402354
Value at i= 4000 0.5772156675050191
Value at i= 4100 0.5772156673795781
Value at i= 4200 0.5772156672629993
Value at i= 4300 0.5772156671544533
Value at i= 4400 0.5772156670532187
Value at i= 4500 0.5772156669586632
```



```
Value at i= 4600 0.5772156668702006
Value at i= 4700 0.5772156667873283
Value at i= 4800 0.5772156667095789
Value at i= 4900 0.5772156666365351
Value at i= 5000 0.5772156665678327
```

Thus from the values of these two codes we can clearly see that second code converges more rapidly than the first code.

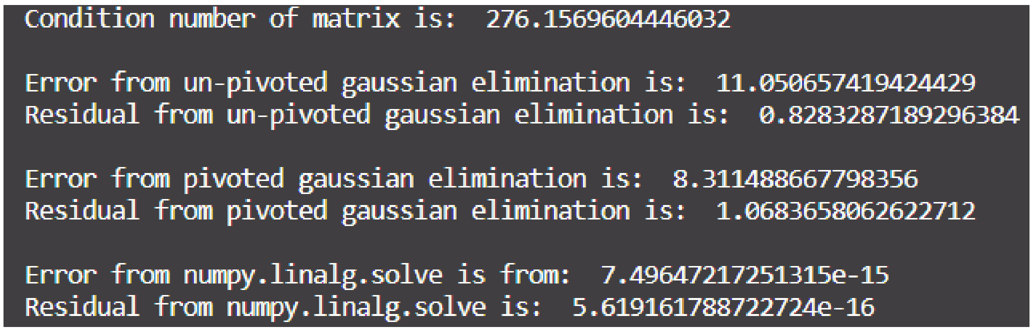
Ans 8:

To run this problem there are a total of 3 files for it named as problem_8.py, problem_8a.py and problem_8b.py where problem_8a.py and problem_8b.py contains the implementation of gaussian elimination with no pivoting and gaussian elimination with partial pivoting. Now problem_8.py in the central file which imports the other two files and you simply have to run this file only once and you see the output for various matrix's with different dimensions as given in the pdf. All the output would be there at a single run.

The format of output is as follows:
For every type of matrix there would be 4 different outputs corresponding to n=10,20,30 and 40 and for each one of them there would be the value printed with a statement indicating which value is it.

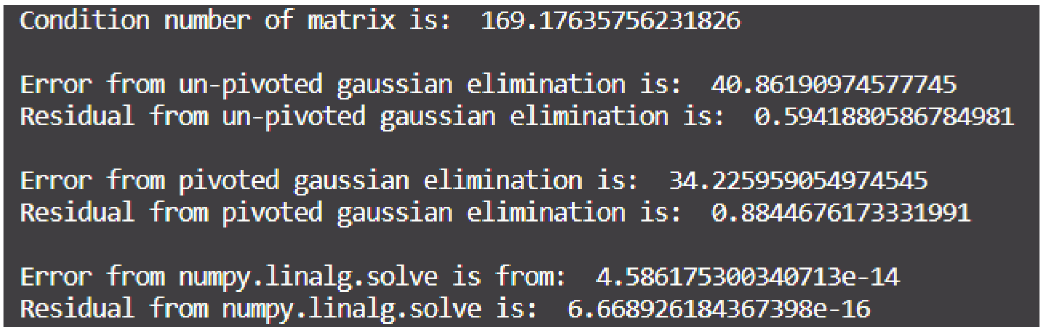
Output for matrix 1 → A random matrix of size n × n with entries uniformly sampled from [0, 1) :

N=10:



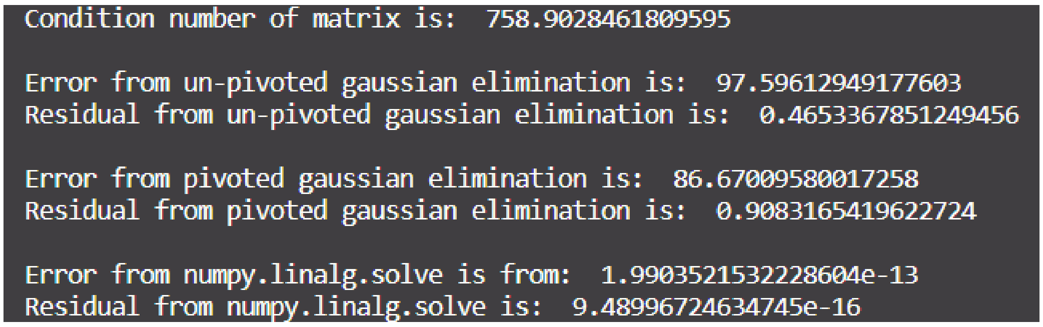
```
Condition number of matrix is:  276.1569604446032

Error from un-pivoted gaussian elimination is:  11.050657419424429
Residual from un-pivoted gaussian elimination is:  0.8283287189296384

Error from pivoted gaussian elimination is:  8.311488667798356
Residual from pivoted gaussian elimination is:  1.0683658062622712

Error from numpy.linalg.solve is from:  7.49647217251315e-15
Residual from numpy.linalg.solve is:  5.619161788722724e-16
```

N=20:

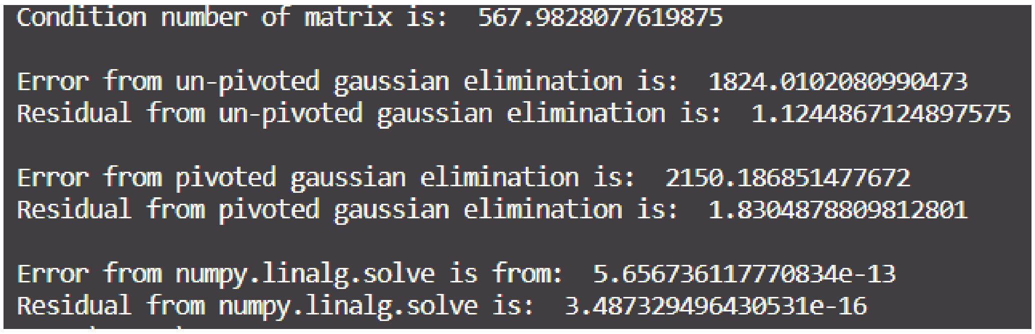


```
Condition number of matrix is:  169.17635756231826

Error from un-pivoted gaussian elimination is:  40.86190974577745
Residual from un-pivoted gaussian elimination is:  0.5941880586784981

Error from pivoted gaussian elimination is:  34.225959054974545
Residual from pivoted gaussian elimination is:  0.8844676173331991

Error from numpy.linalg.solve is from:  4.586175300340713e-14
Residual from numpy.linalg.solve is:  6.668926184367398e-16
```

N=30:


```
Condition number of matrix is:  758.9028461809595

Error from un-pivoted gaussian elimination is:  97.59612949177603
Residual from un-pivoted gaussian elimination is:  0.4653367851249456

Error from pivoted gaussian elimination is:  86.67009580017258
Residual from pivoted gaussian elimination is:  0.9083165419622724

Error from numpy.linalg.solve is from:  1.9903521532228604e-13
Residual from numpy.linalg.solve is:  9.48996724634745e-16
```

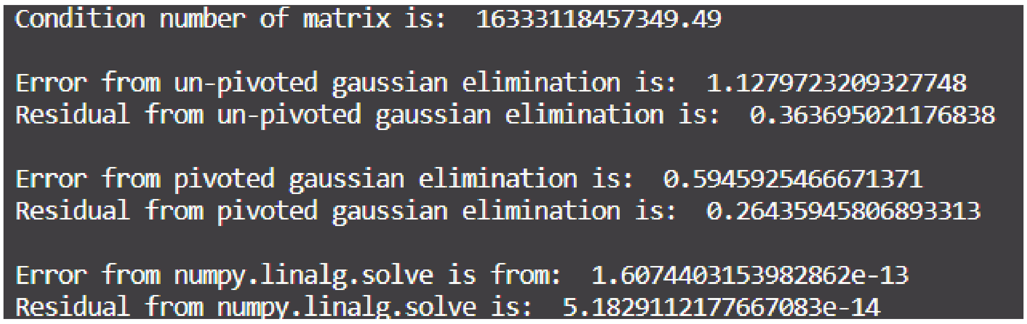
N=40:


```
Condition number of matrix is:  567.9828077619875

Error from un-pivoted gaussian elimination is:  1824.0102080990473
Residual from un-pivoted gaussian elimination is:  1.1244867124897575

Error from pivoted gaussian elimination is:  2150.186851477672
Residual from pivoted gaussian elimination is:  1.8304878809812801

Error from numpy.linalg.solve is from:  5.656736117770834e-13
Residual from numpy.linalg.solve is:  3.487329496430531e-16
```

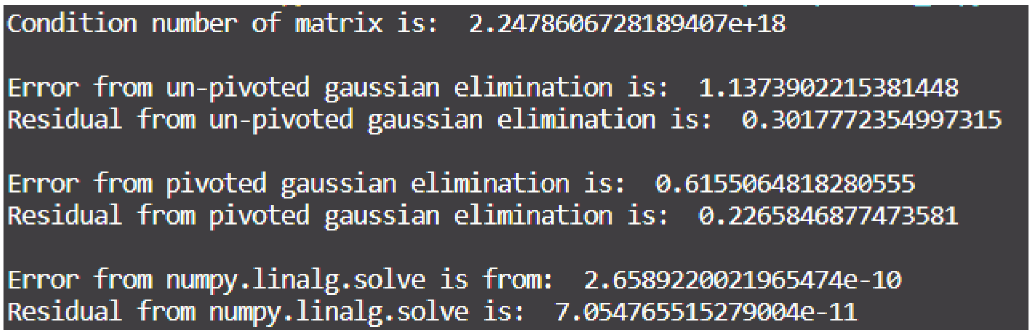
The condition number of a matrix closer to 1, more lesser is the chance of the matrix being singular and vice-a-versa. Thus, which we can observe from the values from above. Using pivoting helps in numerical stability and which also can be observed in each of them. Also this matrix have random values every time commenting on error would be difficult.

Output for matrix 2$\rightarrow$ The Hilbert matrix

N=10:

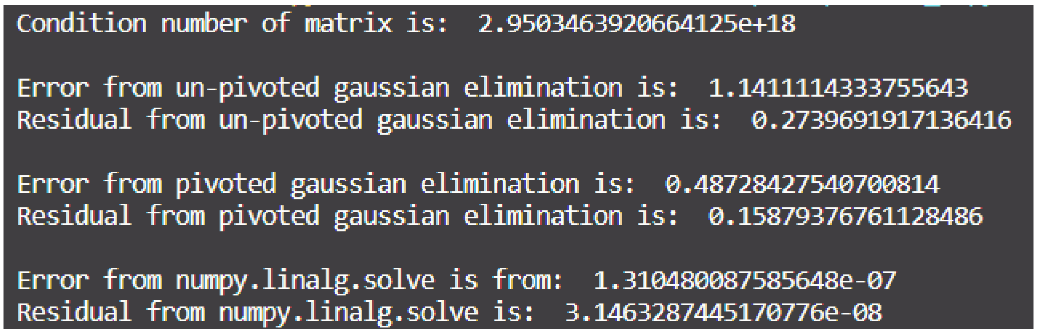

```
Condition number of matrix is:  16333118457349.49

Error from un-pivoted gaussian elimination is:  1.1279723209327748
Residual from un-pivoted gaussian elimination is:  0.363695021176838

Error from pivoted gaussian elimination is:  0.5945925466671371
Residual from pivoted gaussian elimination is:  0.26435945806893313

Error from numpy.linalg.solve is from:  1.6074403153982862e-13
Residual from numpy.linalg.solve is:  5.1829112177667083e-14
```

N=20:



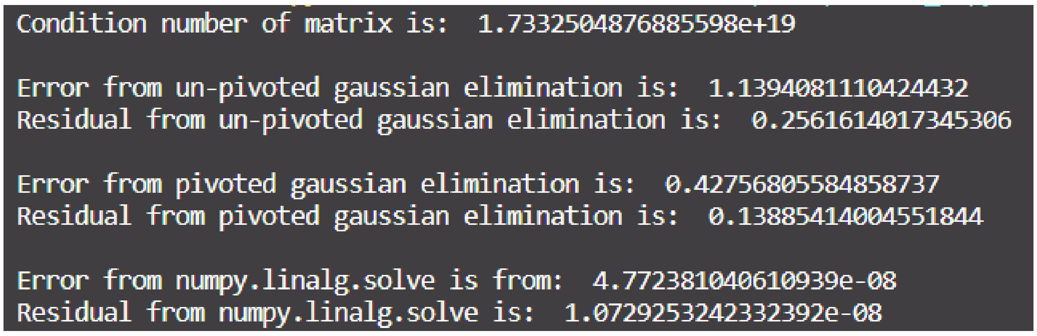
```
Condition number of matrix is:  2.2478606728189407e+18

Error from un-pivoted gaussian elimination is:  1.1373902215381448
Residual from un-pivoted gaussian elimination is:  0.3017772354997315

Error from pivoted gaussian elimination is:  0.6155064818280555
Residual from pivoted gaussian elimination is:  0.2265846877473581

Error from numpy.linalg.solve is from:  2.6589220021965474e-10
Residual from numpy.linalg.solve is:  7.054765515279004e-11
```

N=30:



```
Condition number of matrix is:  2.9503463920664125e+18

Error from un-pivoted gaussian elimination is:  1.1411114333755643
Residual from un-pivoted gaussian elimination is:  0.2739691917136416

Error from pivoted gaussian elimination is:  0.48728427540700814
Residual from pivoted gaussian elimination is:  0.15879376761128486

Error from numpy.linalg.solve is from:  1.310480087585648e-07
Residual from numpy.linalg.solve is:  3.1463287445170776e-08
```

N=40:

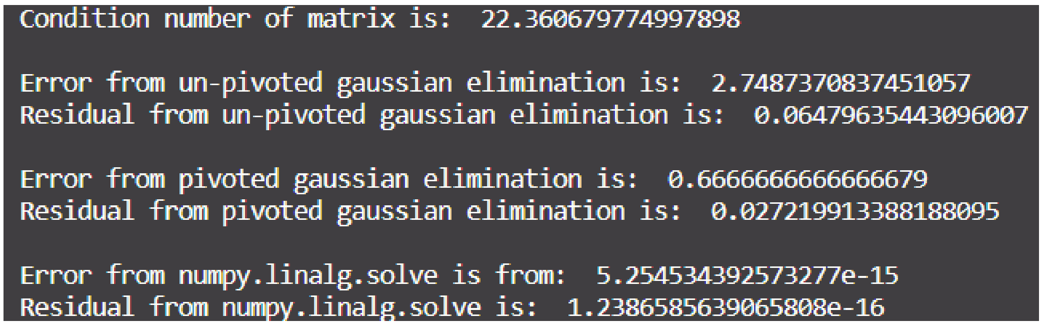


```
Condition number of matrix is:  1.7332504876885598e+19

Error from un-pivoted gaussian elimination is:  1.1394081110424432
Residual from un-pivoted gaussian elimination is:  0.2561614017345306

Error from pivoted gaussian elimination is:  0.42756805584858737
Residual from pivoted gaussian elimination is:  0.13885414004551844

Error from numpy.linalg.solve is from:  4.772381040610939e-08
Residual from numpy.linalg.solve is:  1.0729253242332392e-08
```

Here also we can observe how close or far a matrix is from being singular and using pivoting how the numerical stability is increasing. Here we can also observe that the error and residual for equation Ax=b is also quite small.

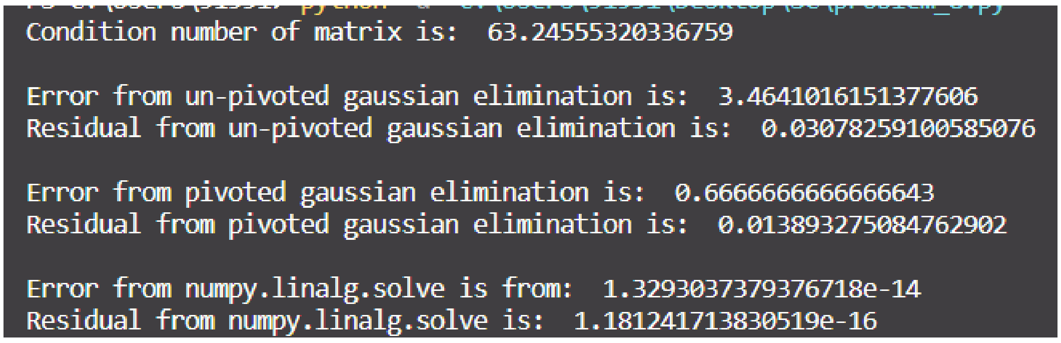
Output for matrix 3→ Having element as 1 for diagonal and upper triangular matrix and -1 for lower triangular matrix.

N=10:



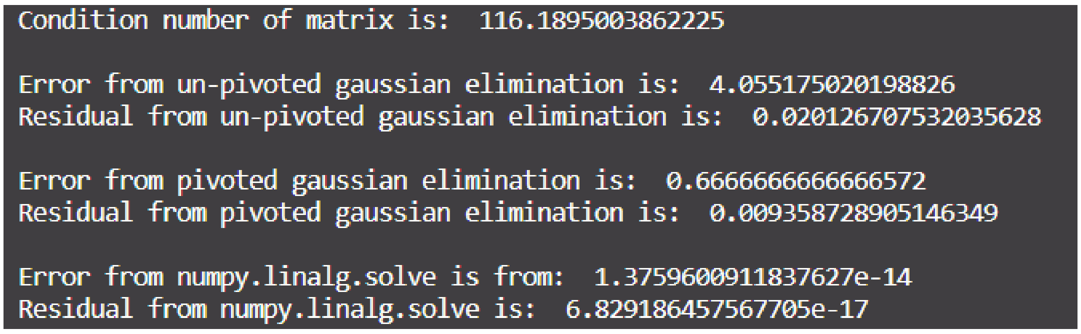
```
Condition number of matrix is:  22.360679774997898

Error from un-pivoted gaussian elimination is:  2.7487370837451057
Residual from un-pivoted gaussian elimination is:  0.06479635443096007

Error from pivoted gaussian elimination is:  0.6666666666666679
Residual from pivoted gaussian elimination is:  0.027219913388188095

Error from numpy.linalg.solve is from:  5.254534392573277e-15
Residual from numpy.linalg.solve is:  1.2386585639065808e-16
```

N=20:



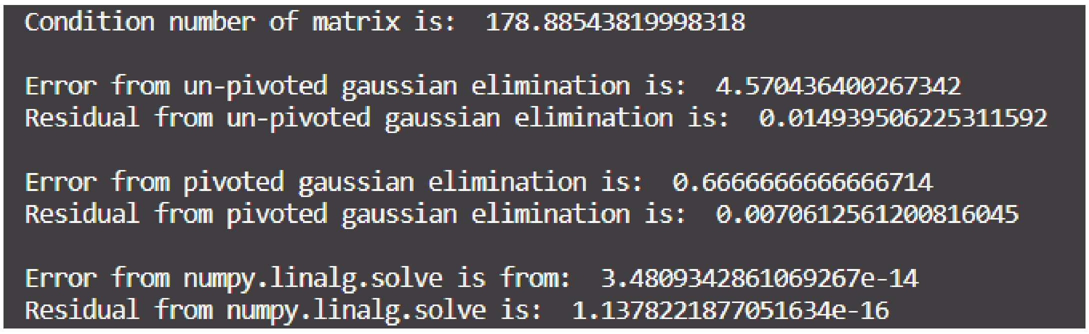
```
Condition number of matrix is:  63.24555320336759

Error from un-pivoted gaussian elimination is:  3.4641016151377606
Residual from un-pivoted gaussian elimination is:  0.03078259100585076

Error from pivoted gaussian elimination is:  0.6666666666666643
Residual from pivoted gaussian elimination is:  0.013893275084762902

Error from numpy.linalg.solve is from:  1.3293037379376718e-14
Residual from numpy.linalg.solve is:  1.181241713830519e-16
```

N=30:



```
Condition number of matrix is:  116.1895003862225

Error from un-pivoted gaussian elimination is:  4.055175020198826
Residual from un-pivoted gaussian elimination is:  0.020126707532035628

Error from pivoted gaussian elimination is:  0.6666666666666572
Residual from pivoted gaussian elimination is:  0.009358728905146349

Error from numpy.linalg.solve is from:  1.3759600911837627e-14
Residual from numpy.linalg.solve is:  6.829186457567705e-17
```

N=40:



```
Condition number of matrix is:  178.88543819998318

Error from un-pivoted gaussian elimination is:  4.570436400267342
Residual from un-pivoted gaussian elimination is:  0.014939506225311592

Error from pivoted gaussian elimination is:  0.6666666666666714
Residual from pivoted gaussian elimination is:  0.0070612561200816045

Error from numpy.linalg.solve is from:  3.4809342861069267e-14
Residual from numpy.linalg.solve is:  1.1378221877051634e-16
```

Here also using the condition number we can see that how close or far is matrix is from being singular . Numerical stability is increasing on pivoting and error is also quite small.