1. **Matrix condition numbers, [2+1+2pt]** Let us explore matrix norms and condition numbers.

(a) For the following matrix given by

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

calculate $\|A\|_1$, $\|A\|_2$, $\|A\|_\infty$ as well as the condition numbers for each norm by hand. Is A well or poorly conditioned?[1]

(a)

$A = \begin{bmatrix} 1 & -2 \\ 3 & -2 \end{bmatrix}$ $\quad A^{-1} = \frac{1}{4}\begin{bmatrix} -2 & 2 \\ -3 & 1 \end{bmatrix}$

1-norm (largest column sum)

$\|A\|_1 = 4$ $\quad \|A^{-1}\|_1 = \frac{5}{4}$ $\quad \kappa_1(A) = 5$

∞-norm (largest row sum)

$\|A\|_\infty = 5$ $\quad \|A^{-1}\|_\infty = 1$ $\quad \kappa_\infty(A) = 5$

2-norm max eigen of A^TA

$A^TA = \begin{bmatrix} 10 & -8 \\ -8 & 8 \end{bmatrix}$ $\quad (A^{-1})^T(A^{-1}) = \frac{1}{16}\underbrace{\begin{bmatrix} 13 & -7 \\ -7 & 5 \end{bmatrix}}_{C}$

$\chi(A^TA) = (x-10)(x-8) - 64$

$= x^2 - 18x + 16$

$= \frac{18 \pm \sqrt{260}}{2}$

$\lambda_{max} = 9 \pm \sqrt{65}$

$\Rightarrow \|A\|_2 = \sqrt{9+\sqrt{65}}$

$\chi(C) = (x-13)(x-5) - 49$

$= x^2 - 18x + 16$

$x = 9 \pm \sqrt{65}$

$\Rightarrow \|A^{-1}\|_2 = \sqrt{\frac{9+\sqrt{65}}{16}}$

$\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2$

$= \sqrt{\frac{146 + 18\sqrt{65}}{16}}$

$= \sqrt{\frac{73 + 9\sqrt{65}}{8}}$

For all norm, A is well-conditioned.

(b) Recall the formulas from Theorems 2.7 and 2.8 in the text book. If you assume that taking the absolute value and determining the maximum does not contribute to the overall computational cost, how many *flops* (floating point operations) are needed to calculate $\|A\|_1$ and $\|A\|_\infty$ for $A \in \mathbb{R}^{n\times n}$? By what factor will the calculation time increase when you double the matrix size?

(b)

FLOPs: - n^2 for Calculating the Column/row Sum (n rows/cols, n addition for calculating)

- n for finding the maximum.

as $n = o(n^2)$ we ignore it

Flops. n^2

If the size of the matrix is doubled,

the flop will be increased by factor 4.

(c) Now implement a simple code that calculates $\|A\|_1$ and $\|A\|_\infty$ for a matrix of any size $n \geq 1$. Try to do this without using loops[2]! Using system sizes of $n_1 = 100$, $n_{k+1} = 2n_k, k = 1, \ldots, 7$, determine how long your code takes[3] to calculate $\|A\|_1$ and $\|A\|_\infty$ for a matrix $A \in \mathbb{R}^{n_k \times n_k}$ with random entries and report the results. Can you confirm the estimate from (b)? Please also hand in your code.

see code file.

2. **Induced matrix norms, [2+2+1+1pt]** Let $A, B \in \mathbb{R}^{n\times n}$ and let the matrix norm $\|\cdot\|$ be induced by/subordinate of a vector norm $\|\cdot\|$.

2 of 2

(a) Show that $\|AB\| \leq \|A\|\|B\|$.

(a) First of all,

As $\|A\| = \max \frac{\|Ax\|}{\|x\|}$. $\|Ax\| \leq \|A\|\|x\| \quad \forall x \in \mathbb{R}^n$

Consider $\|ABx\|$. as $Bx \in \mathbb{R}^n$

we have $\|ABx\| \leq \|A\|\|Bx\|$

$\Rightarrow \|ABx\| \leq \|A\|\|B\|\|x\|$

$\Rightarrow \frac{\|ABx\|}{\|x\|} \leq \|A\|\|B\|$

As this holds for arbitary x

$\|AB\| = \max \frac{\|ABx\|}{\|x\|} \leq \|A\|\|B\|$

(b) For the identity matrix $I \in \mathbb{R}^{n\times n}$, show that $\|I\| = 1$.

(b) $\forall v. \quad \frac{\|Iv\|}{\|v\|} = \frac{\|v\|}{\|v\|} = 1$

$\Rightarrow \|I\| = \max_v \frac{\|Iv\|}{\|v\|} = 1$

(c) For A invertible, show that $\kappa(A) \geq 1$, where $\kappa(A)$ is the condition number of that matrix A corresponding to the norm $\|\cdot\|$. Use the above two properties with $B := A^{-1}$ for your argument.

(c)

$$k(A) = \|A\| \cdot \|A^{-1}\| \geq \|I\| \qquad \|I\| = 1$$

$$\Rightarrow k(A) \geq 1$$

(d) Argue that the Frobenius matrix norm $\|A\|_F := \left(\sum_{i,j=1}^{n} a_{ij}^2\right)^{1/2}$ cannot be induced by a suitable vector norm. *Hint*: Use one of the points above.

(d) $\|I\|_F = \sqrt{n}$

For $n > 1$, $\|I\|_F \neq 1$.

However, as is shown in (b)

If is a induced norm, $\|I\|_p = 1$

$\Rightarrow \|I\|_F$ is not a induced norm.

3. **Sharpness of condition number estimates [4pt]** Let $A \in \mathbb{R}^{n\times n}$ be invertible. Let $b \in \mathbb{R}^n \setminus \{0\}$, and $Ax = b$, $Ax' = b'$ and denote the perturbations by $\Delta b = b' - b$ and $\Delta x = x' - x$. Show that the inequality obtained in Theorem 2.11 is *sharp*. That is, find vectors $b, \Delta b$ for which

$$\frac{\|\Delta x\|_2}{\|x\|_2} = \kappa_2(A)\frac{\|\Delta b\|_2}{\|b\|_2}.$$

Proof. We proceed the proof by the order of hints.

1° Given (λ, v), $\|Av\|_2^2 = \lambda \|v\|_2^2$.

By def. $Av = \lambda v$ and $v^T \cdot A^T = \lambda \cdot v^T$

$$\Rightarrow \|Av\|_2^2 = v^T A^T A v = \lambda\ v^T v = \lambda \|v\|_2^2$$

2° Let λ_{max}, v_{max} be the maximal eigenvalue and corresponding eigenvector of A^TA.

First of all. $\lambda_{max} = \|A\|_2^2$

Therefore. set $x = v_{max}$.

$$\|Ax\|_2^2 = \lambda_{max}\|x\|_2^2 = \|A\|_2^2 \|x\|_2^2$$

and we get $\|Ax\|_2 = \|A\|_2 \|x\|_2$ As desired.

3° Similarly.

$\Rightarrow \|A^{-1}\|_2 = \sqrt{\lambda'_{max}}$ where λ'_{max} is the maximum eigenvalue of $(A^{-1})^T(A^{-1}) = (A^T)^{-1}\cdot(A^{-1}) = (A\cdot A^T)^{-1}$

Here we claim the eigenvalue of $(A\cdot A^T)^{-1}$ is the inverse of A^TA

If λ an eigenvalue of A^TA.

$A^TA\,v = \lambda v$. As A is invertible. Let $A^T\phi = v$.

$A^TA\,A^T\phi = \lambda A^T\phi$

$(A^T)^{-1}\,A\cdot A^T\phi = \lambda\,(A^T)^{-1}\cdot A^T\phi$

$\Rightarrow A\cdot A^T\phi = \lambda\phi$ which implies λ is also an eigenvalue for $A^T\cdot A$.

Then it suffices to take the inverse.

From the diagonalized version of the matrix. it is easy to see the eigenvalue of the inverse is the inverse of eigenvalue.

Here. $\|A^{-1}\|_2 = \sqrt{\max \lambda(AA^T)} = \frac{1}{\sqrt{\min \lambda(AA^T)}} = \frac{1}{\sqrt{\lambda_{min}}}$

Let $\Delta x = v_{min}$, which is the eigenvector corresponds to λ_{min}

$\|\Delta b\|_2^2 = \|A\Delta x\|_2^2 = \lambda_{min}\,\|\Delta x\|_2^2$

$\Rightarrow \|\Delta x\|_2^2 = \frac{1}{\lambda_{min}}\|\Delta b\|^2 = \|A^{-1}\|_2^2\,\|\Delta b\|^2$

$\Rightarrow \|\Delta x\|_2 = \|A^{-1}\|_2\,\|\Delta b\|_2^2$

4° Combining 1° 2° and 3°.

$$\begin{cases} \|\Delta x\| = \|A^{-1}\Delta b\| = \|A^{-1}\|\,\|\Delta b\| \\ \|b\| = \|Ax\| = \|A\|\cdot\|x\| \end{cases}$$

Combine. $\Rightarrow \frac{\|\Delta x\|}{\|x\|} = K(A)\cdot\frac{\|\Delta b\|}{\|b\|}$

4. **Least squares [3pt]**. We believe that a real number Y is approximately determined by X with the model function

$$Y = a\exp(X) + bX^2 + cX + d\,.$$

We are given the following table of data for the values of X and Y:[4]

X	0.0	0.5	1.0	1.5	2.0	2.0	2.5
Y	0.0	0.20	0.27	0.30	0.32	0.35	0.27

Using the above data points, write down 7 equations in the four unknowns a, b, c, d. The least squares solution to this system is the best fit function. Write down the normal equations for this system, and solve them with MATLAB/Python/Julia. Plot the data points (X, Y) as points[5] and the best fit function.

Write the Equation in Matrix Form.

$$\begin{bmatrix} \exp(0) & 0^2 & 0 & 1 \\ \exp(0.5) & 0.5^2 & 0.5 & 1 \\ \exp(1.0) & 1.0^2 & 1.0 & 1 \\ \exp(1.5) & 1.5^2 & 1.5 & 1 \\ \exp(2.0) & 2.0^2 & 2.0 & 1 \\ \exp(2.0) & 2.0^2 & 2.0 & 1 \\ \exp(2.5) & 2.5^2 & 2.5 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0.2 \\ 0.27 \\ 0.30 \\ 0.32 \\ 0.35 \\ 0.27 \end{bmatrix}$$

X θ Y

Normal Equation

$$(X^T X)\,\theta = X^T Y$$

5. **Floating point arithmetic, [2+2pts]** Consider the Harmonic sum

$$H(N) := \sum_{i=1}^{N} \frac{1}{i}, \tag{1}$$

which satisfies

$$H(N) = \Psi(N+1) - \Psi(1) \tag{2}$$

where Ψ is the digamma special function.[6]

(a) While software uses double precision by default, you can force it to use single precision using the `single` function. To compute $H(3)$, you could write (in MATLAB)

```
result = single(0.);
result = result + single(1.)/single(1.);
result = result + single(1.)/single(2.);
result = result + single(1.)/single(3.);
```

Implement a function that computes the sum $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}$ for arbitrary N (in single precision). Now run this function for $N \in \{2^1, 2^2, \ldots, 2^{30}\}$. Plot, using a double-logarithmic plot, the relative error

$$\frac{|H(N) - H_{\text{ex}}(N)|}{|H_{\text{ex}}(N)|}$$

as a function of N, where $H_{\text{ex}}(N)$ is given by (2)

See attached Code

(b) For large N, we add a small number ($\frac{1}{N}$) to a large number ($H(N-1)$), which can lead to numerical error. Rearrange the sum to avoid this issue (i.e., start summing from the smallest element) and repeat the computation from the previous task. Plot the error again and compare to the error when using forward summation.

See Attached Code

HW_3_code

March 6, 2022

[1]:
```
import numpy as np
import time
import matplotlib.pyplot as plt
from scipy.special import digamma
from tqdm import tqdm
```

0.1 Question 1

[2]:
```
def norm_1(A):
    """
    Input: 2D numpy array
    Output: float
    """
    if A.shape[0] != A.shape[1]:
        raise ValueError("Input should be square.")

    # Take the absolute value of each entry
    # Get columns sum
    # n^2 complexity
    col_sum = np.sum(abs(A), axis=0)


    # return the max of the col sum
    # n complexity
    return max(col_sum)

def norm_infinity(A):
    """
    Input: 2D numpy array
    Output: float
    """
    if A.shape[0] != A.shape[1]:
        raise ValueError("Input should be square.")

    # Take the absolute value of each entry
    # Get columns sum
    # n^2 complexity
```

```
    row_sum = np.sum(abs(A), axis=1)


    # return the max of the col sum
    # n complexity
    return max(row_sum)
```

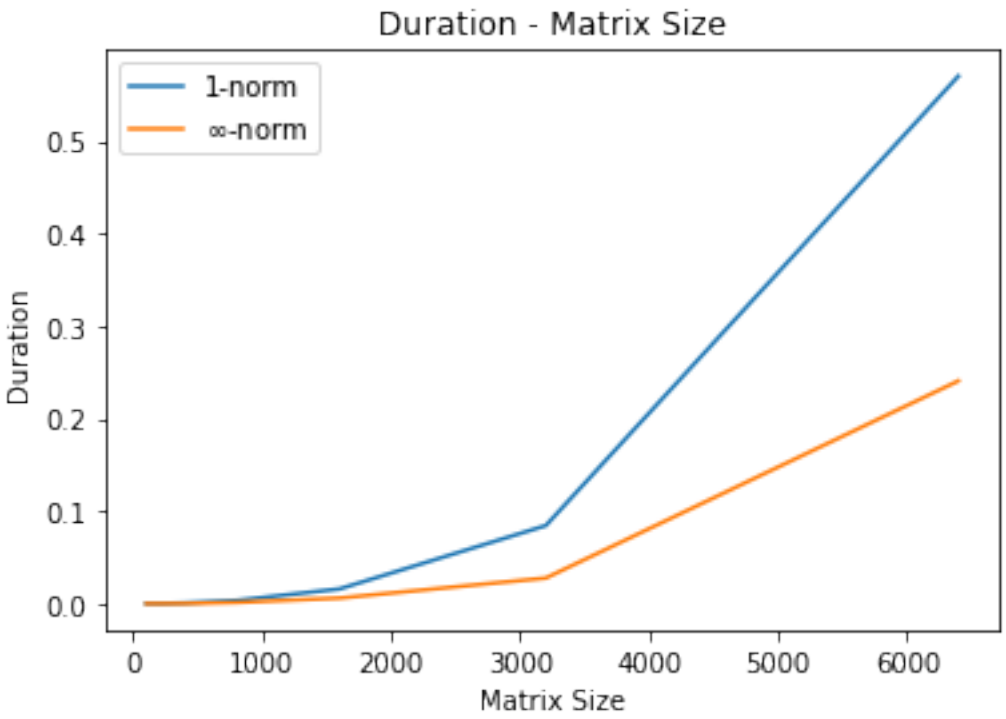
[3]:
```
sizes = [int(100*(2**i)) for i in range(7)]
duration_1 = {}
duration_inf = {}
for size in sizes:
    A = np.random.random((size, size))
    start = time.time()
    norm_1(A)
    end = time.time()
    duration_1[size] = end - start

    start = time.time()
    norm_infinity(A)
    end = time.time()
    duration_inf[size] = end - start
```

[4]:
```
print("\nFor 1 norm\n")
for i in range(7):
    print(f"The duration for {100*(2**i):4d} x {100*(2**i):4d} matrix is␣
 ↪{duration_1[100*(2**i)]:.6f}")
print("\nFor infinity-norm\n")
for i in range(7):
    print(f"The duration for {100*(2**i):4d} x {100*(2**i):4d} matrix is␣
 ↪{duration_inf[100*(2**i)]:.6f}")
```

```
For 1 norm

The duration for  100 x  100 matrix is 0.000358
The duration for  200 x  200 matrix is 0.000275
The duration for  400 x  400 matrix is 0.000971
The duration for  800 x  800 matrix is 0.003606
The duration for 1600 x 1600 matrix is 0.016091
The duration for 3200 x 3200 matrix is 0.084503
The duration for 6400 x 6400 matrix is 0.570026

For infinity-norm

The duration for  100 x  100 matrix is 0.000173
The duration for  200 x  200 matrix is 0.000176
The duration for  400 x  400 matrix is 0.000289
The duration for  800 x  800 matrix is 0.001526
```

```
The duration for 1600 x 1600 matrix is 0.006191
The duration for 3200 x 3200 matrix is 0.027819
The duration for 6400 x 6400 matrix is 0.240773
```

[5]:
```
# Visualization
plt.plot(sizes, list(duration_1.values()), label = "1-norm")
plt.plot(sizes, list(duration_inf.values()), label = r"$\infty$-norm")
plt.xlabel("Matrix Size")
plt.ylabel("Duration")
plt.title("Duration - Matrix Size")
plt.legend()
plt.show()
```



From the result, we could say the flop will be increased roughly by factor 4 when the matrix size is doubeled, which is consistent with what we found analytically.

0.2 Question 4

[6]:
```
X = np.array([0.0, 0.5, 1.0, 1.5, 2.0, 2.0, 2.5])
Y = np.array([0.0, 0.20, 0.27, 0.30, 0.32, 0.35, 0.27])
X_hat = np.ones((4, len(X)))
X_hat[0] = np.exp(X)
X_hat[1] = X**2
X_hat[2] = X
```

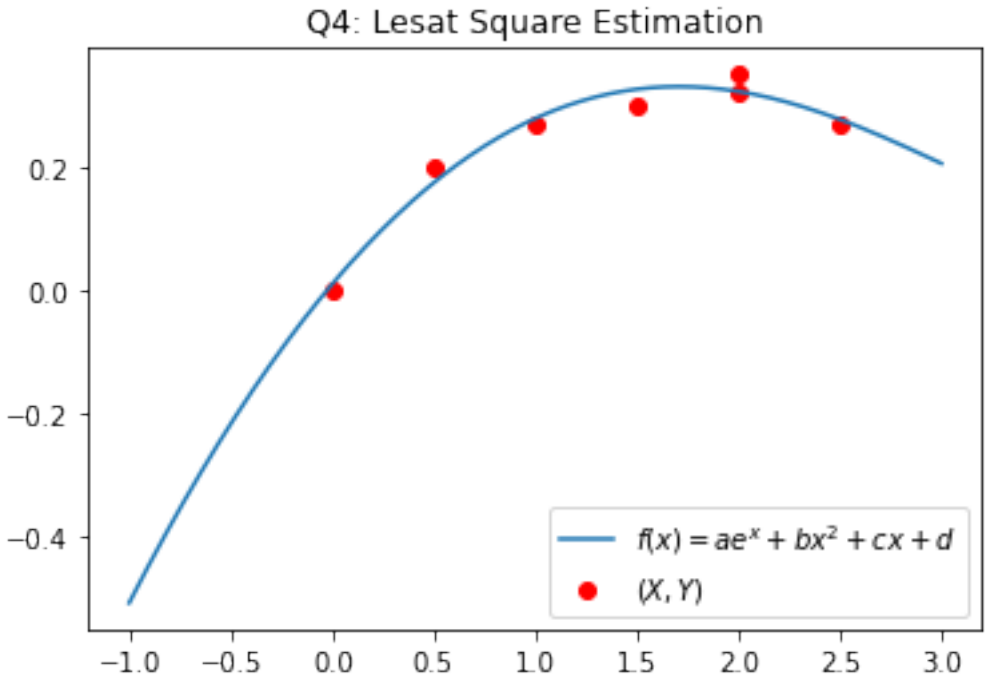
```
X_hat = X_hat.T
```

Let $\theta := [a, b, c, d]^T$. The normal equation turns out to be

$$(\hat{X}^T \hat{X})\theta = \hat{X}^T Y$$

[7]:
```
# Least Squaee estimation
# theta = np.linalg.inv(X_hat.T @ X_hat)@X_hat.T@Y
theta = np.linalg.solve(X_hat.T @ X_hat, X_hat.T@Y)
```

[8]:
```
f = lambda z: theta[0]*np.exp(z) + theta[1]*(z**2) + theta[2]*z + theta[3]
x_space = np.linspace(-1, 3, 1000)
plt.plot(x_space, f(x_space), label = r"$f(x) = ae^x + b x^2 + cx + d$")
plt.scatter(X, Y, c="r", label=r"$(X, Y)$")
plt.title("Q4: Lesat Square Estimation")
plt.legend()
plt.show()
```



0.3 Question 5

0.3.1 Clarification

Since the runtime for $N = 2^{25}$ and more is huge, with the approval of the professor, we only do the part from 2^1 to 2^{25}

For $N \leq 25$ it only takes

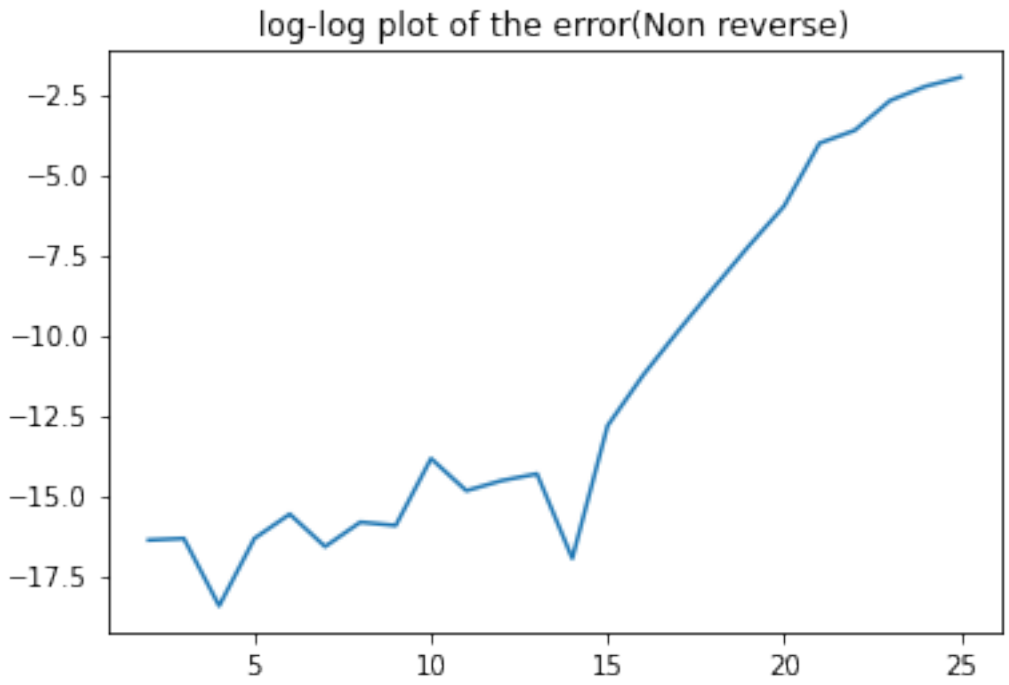
[9]:
```
def H(N):
    result = np.single(0)
    for i in range(1, N+1):
        result += np.single(1)/np.single(i)
    return result


def H_ex(N):
    return digamma(N+1) - digamma(1)


# def relative_err(h, h_ex):
#     return abs(h - h_ex) / abs(h_ex)
```

[10]:
```
h_ex_20 = np.zeros(25, dtype=np.double)

for i in tqdm(range(1, 26)):
    h_ex_20[i-1] = H_ex(2**i)

h_20 = np.zeros(25, dtype=np.single)

for i in tqdm(range(1, 26)):
    h_20[i-1] = H(2**i)

err_1 = abs(h_20 - h_ex_20) / abs(h_ex_20)
```

```
100%|      | 25/25 [00:00<00:00, 47339.77it/s]
100%|      | 25/25 [01:04<00:00,  2.59s/it]
```

[11]:
```
x_space = list(range(1, 26))
plt.plot(x_space, np.log(err_1))
plt.title("log-log plot of the error(Non reverse)")
plt.show()
```

```
<ipython-input-11-b18cacf9f3a0>:2: RuntimeWarning: divide by zero encountered in
log
  plt.plot(x_space, np.log(err_1))
```



Here we implememt the reversed order addition

[12]:
```
def H_reverse(N):
    result = np.single(0)
    for i in range(N, 0, -1):
        result = np.single(1)/np.single(i) + result
    return result
```

[13]:
```
h_20_reverse = np.zeros(25, dtype=np.single)

for i in tqdm(range(1, 26)):
    h_20_reverse[i-1] = H_reverse(2**i)

err_2 = abs(h_20_reverse - h_ex_20) / abs(h_ex_20)
```

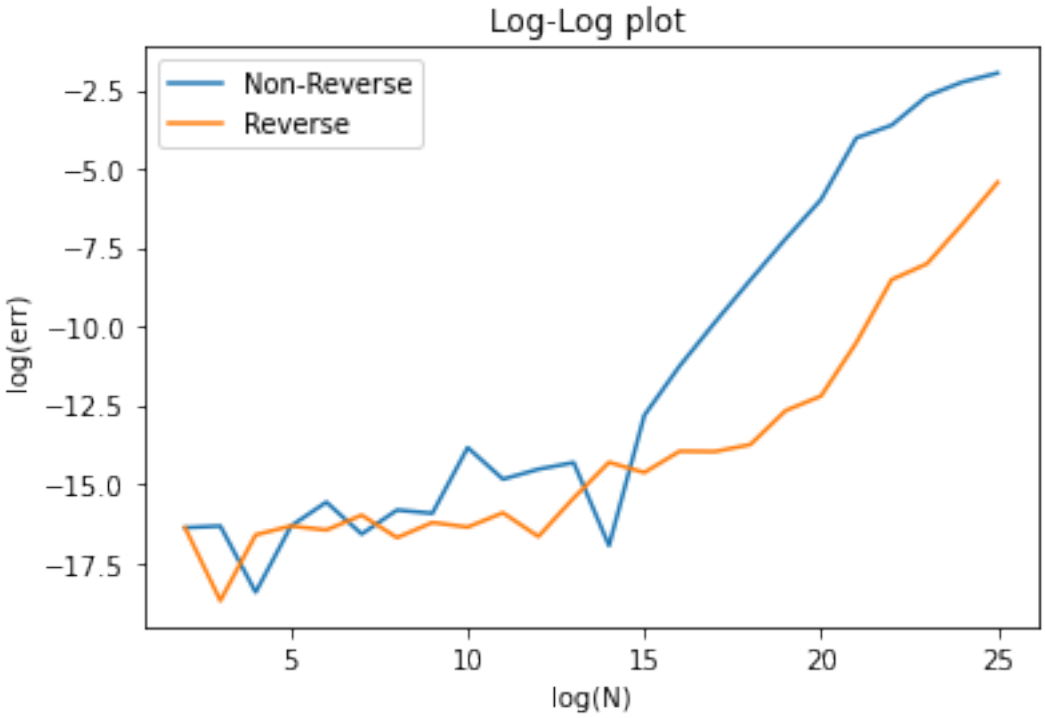
```
100%|      | 25/25 [01:03<00:00,  2.56s/it]
```

0.3.2 Visualization

[15]:
```
x_space = list(range(1, 26))
plt.plot(x_space, np.log(err_1), label="Non-Reverse")
plt.plot(x_space, np.log(err_2), label="Reverse")
plt.xlabel("log(N)")
plt.ylabel("log(err)")
plt.title("Log-Log plot ")
```

```
plt.legend()
plt.show()
```

```
<ipython-input-15-be5f1d5a1fec>:2: RuntimeWarning: divide by zero encountered in
log
  plt.plot(x_space, np.log(err_1), label="Non-Reverse")
<ipython-input-15-be5f1d5a1fec>:3: RuntimeWarning: divide by zero encountered in
log
  plt.plot(x_space, np.log(err_2), label="Reverse")
```



Here we can see the error with reversed order addition is much smaller.