Assignment 2

Name: 陈子蔚 SID: 12332440

1

a

$$
\begin{aligned}
11.60 &= x_1 + 10x_2 && (1)\\
11.85 &= x_1 + 15x_2 && (2)\\
12.25 &= x_1 + 20x_2 && (3)\\
\Leftrightarrow \begin{bmatrix}11.60\\11.85\\12.25\end{bmatrix} &= \begin{bmatrix}1 & 10\\1 & 15\\1 & 20\end{bmatrix} \begin{bmatrix}x_1\\x_2\end{bmatrix} && (4)
\end{aligned}
$$

Denote it as

$$
\mathrm{y} = \mathrm{Ax} \tag{5}
$$

b

- Choose (1)&(2), $\mathrm{x} = [11.1, 0.05]^\top$
- Choose (1)&(3), $\mathrm{x} = [10.95, 0.065]^\top$
- Choose (2)&(3), $\mathrm{x} = [10.65, 0.08]^\top$
  Not consistent obviously. There's no particular reason to prefer one of these results over the others without additional context or constraints.

c

$$
\mathrm{x} = (\mathrm{A}^\top \mathrm{A})^{-1} \mathrm{A}^\top y = [10.925, 0.065]^\top \tag{6}
$$

Comparing this result with those obtained in part (b), we see that the least squares solution gives a different result from any pair of solutions obtained by considering only two of the three points. This least squares solution is preferred when dealing with an overdetermined system because it takes into account all available data to minimize the MSE.

2

a

$$
A \in \mathbb{R}^{m \times n} \wedge m > n \tag{7}
$$

Thus,

$$
A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \Leftrightarrow \#H = \#\text{column} = 3 \tag{8}
$$

b

To avoid cancellation,

$$-\|c_1\|e_1 = [-2, 0, 0, 0]^\top \tag{9}$$

c

The first column would not change.

$$[-2, 0, 0, 0]^\top \tag{10}$$

3

$$A \in \mathbb{R}^{m \times n} \wedge m > n \tag{11}$$

Thus,

$$\begin{aligned} A = QR \wedge Q \in \mathbb{R}^{m \times n} \wedge R \in \mathbb{R}^{n \times n} \\ \Leftrightarrow Q^\top Q = I \in \mathbb{R}^{n \times n} \end{aligned} \tag{12}$$

As for the equation,

$$Ax = QRx = b \tag{13}$$

$$\Leftrightarrow Rx = Q^\top b \tag{14}$$

Output

```
# Q
[[ 5.77350269e-01  2.04124145e-01  3.53553391e-01]
 [ 0.00000000e+00  6.12372436e-01  3.53553391e-01]
 [ 0.00000000e+00  0.00000000e+00  7.07106781e-01]
 [-5.77350269e-01  4.08248290e-01 -1.17756934e-16]
 [-5.77350269e-01 -2.04124145e-01  3.53553391e-01]
 [ 0.00000000e+00 -6.12372436e-01  3.53553391e-01]]
# R
[[ 1.73205081 -0.57735027 -0.57735027]
 [ 0.          1.63299316 -0.81649658]
 [ 0.          0.          1.41421356]]
# x
[1236. 1943. 2416.]
```

4

Firstly prove the $\|A\|_2 = \max_i \sqrt{\lambda_i} = \max_i \sigma_i$,

$$\|A\|_2 = \sup_{\|x\|_2 = 1} \|Ax\|_2 \tag{15}$$

Since $A^\top A$ is semi-positive,

$$A^\top A = H^\top \mathrm{diag}(\lambda_i) H \tag{16}$$

$$\begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \end{bmatrix}$$

$$\operatorname{diag}(\lambda_i) = \begin{bmatrix} 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \tag{17}$$

$\forall x \in \{x | \|x\|_2 = 1\}$, denote

$$Hx = y = (y_1, y_2, \dots, y_n)^\top \wedge \|y\|_2 = \|x\|_2 = 1 \tag{18}$$

$$\|Ax\|_2^2 = x^\top A^\top Ax = (Hx)^\top \operatorname{diag}(\lambda_i)(Hx) = \sum_{i=1}^{n} \lambda_i y_i^2 \leq \|y\|_2^2 \max_{1 \leq i \leq n} \lambda_i = \max_{1 \leq i \leq n} \lambda_i \tag{19}$$

That is,

$$\|A\|_2 = \max \sigma(A) \tag{20}$$

$$\Leftrightarrow \|A^{-1}\|_2 = \max \sigma(A^{-1}) = \max \frac{1}{\sigma(A)} = \frac{1}{\min \sigma(A)} \tag{21}$$

5

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{22}$$

Please refer to `as2_5.py` for more information.

The iteration history for the function $f(x) = x^2 - 1$ using Newton's method with $x_0 = 2$ for 4 iterations is:

0. $x_0 = 2, h = -\frac{3}{4}$
1. $x_1 = \frac{5}{4}, h = -\frac{9}{40}$
2. $x_2 = \frac{41}{40}, h = -\frac{81}{3280}$
3. $x_3 = \frac{3281}{3280}, h = -\frac{6561}{21523360}$
4. $x_3 = \frac{21523361}{21523360}$

When we start with $x_0 = -2$, the iteration history is:

0. $x_0 = -2, h = \frac{3}{4}$
1. $x_1 = -\frac{5}{4}, h = \frac{9}{40}$
2. $x_2 = -\frac{41}{40}, h = \frac{81}{3280}$
3. $x_3 = -\frac{3281}{3280}, h = \frac{6561}{21523360}$
4. $x_3 = -\frac{21523361}{21523360}$

The sequences converge to the roots of the function, which are $x = 1$ and $x = -1$, respectively. The sequences are symmetrical because the function is symmetrical and the initial guesses are symmetrical about the origin.

6

$$x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \tag{23}$$

Please refer to `as2_6.py` for more information.

After applying the secant method to the function $f(x) = x^2 - 1$ with starting points $x_0 = 3$ and $x_1 = 2$, and performing 6 iterations, the sequence of approximations to the root is:

0. $x_1 = 2, h = -0.6$
1. $x_2 = 1.4, h = -0.2823529411764706$
2. $x_3 = 1.1176470588235294, h = -0.0989554700384827$
3. $x_4 = 1.0186915887850467, h = -0.017662252706817205$
4. $x_5 = 1.0010293360782296, h = -0.0010198100459649776$
5. $x_6 = 1.0000095260322646, h = -9.52113206547989 \times 10^{-6}$
6. $x_7 = 1.0000000049001991$

As we can see, the approximations are converging towards 1.

7

Core code

```
# Define the Newton's method function
def newton(f, df, a, b, iterations=5):
    x_n = (a + b) / 2
    for _ in range(iterations):
        fx = f(x_n)
        dfx = df(x_n)
        x_n = x_n - fx / dfx

    assert a <= x_n <= b # Ensure the root is within the interval
    return x_n

# Define the Bisection method function
def bisect(f, a, b, iterations=15):
    for _ in range(iterations):
        c = (a + b) / 2
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c
```

Output

```
# roots obtained by Bisection method function
[-0.90618896484375, -0.53846435546875, -1.2207031249888982e-05,
0.53846435546875, 0.90618896484375]
# roots obtained by Newton's method function
[-0.9062605812803672, -0.538469310105683, 0.0, 0.538469310105683,
0.9062605812803672]
```

8

a

$$\lim_{x \to -\infty} f(x) = -\infty \tag{24}$$
$$\lim_{x \to +\infty} f(x) = +\infty \tag{25}$$

Therefore, $\lim_{|x| \to +\infty} f(x) \neq +\infty$ and $f(x)$ is not coercive. Then $f(x)$ could only have local minimum local maximum on $\mathbb{R}$.

$$\frac{\partial f(x)}{\partial x} = 3x^2 + 12x - 15 = 3(x+5)(x-1) \tag{26}$$
$$\frac{\partial^2 f(x)}{\partial x^2} = 6x + 12 \tag{27}$$

Thus,

$$\frac{\partial f(x)}{\partial x} = 0 \Leftrightarrow x_1 = -5, x_2 = 1 \tag{28}$$

The critical points are $x_1 = -5, x_2 = 1$.

$$\frac{\partial^2 f(x)}{\partial x^2}_{x=x_1} = 6 \times (-5) + 12 = -18 < 0 \tag{29}$$
$$\frac{\partial^2 f(x)}{\partial x^2}_{x=x_2} = 6 \times 1 + 12 = 18 > 0 \tag{30}$$

Thus, $x_1 = -5$ is local maximum and $x_2 = 1$ is local minimum of $f(x)$ on $\mathbb{R}$.

d

$$\lim_{x \to -\infty} f(x) = 0 \tag{31}$$
$$\lim_{x \to +\infty} f(x) = +\infty \tag{32}$$

Therefore, $\lim_{|x| \to +\infty} f(x) \neq +\infty$ and $f(x)$ is not coercive. However, (31) implies that $f(x)$ could have global minimum on $\mathbb{R}$. $f(x)$ could only have local maximum on $\mathbb{R}$.

$$\frac{\partial f(x)}{\partial x} = x^2 e^x + 2xe^x = x(x+2)e^x \tag{33}$$
$$\frac{\partial^2 f(x)}{\partial x^2} = x^2 e^x + 4xe^x + 2e^x = (x^2 + 4x + 2)e^x \tag{34}$$

Thus,

$$\frac{\partial f(x)}{\partial x} = 0 \Leftrightarrow x_1 = -2, x_2 = 0 \tag{35}$$

The critical points are $x_1 = -2, x_2 = 0$.

$$\frac{\partial^2 f(x)}{\partial x^2}_{x=x_1} = -2e^{-2} < 0 \tag{36}$$
$$\frac{\partial^2 f(x)}{} = 2 > 0 \tag{37}$$

$$\partial x^2 \quad x=x_2$$

Thus, $x_1 = -2$ is local maximum and $x_2 = 0$ is local minimum of $f(x)$ on $\mathbb{R}$.

$$f(0) = 0 = \lim_{x \to -\infty} f(x) = 0 \tag{38}$$

$x_2 = 0$ is also global minimum of $f(x)$ on $\mathbb{R}$.

9

Core code

```
# Implementation of the Golden Section Search algorithm
def golden_section_search(f, a, b, tol=3e-3):
    record = []

    tau = (np.sqrt(5) - 1) / 2  # Golden ratio

    # Initial two points and their function values
    x1 = a + (1 - tau) * (b - a)
    x2 = a + tau * (b - a)
    f1 = f(x1)
    f2 = f(x2)

    # Iteration loop
    while (b - a) > tol:
        record += [[x1, f1, x2, f2]]
        if f1 < f2:
            b = x2
            x2 = x1
            f2 = f1
            x1 = a + (1 - tau) * (b - a)
            f1 = f(x1)
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = a + tau * (b - a)
            f2 = f(x2)

    # Return the approximated minimum location
    return a, b, record
```

Output

```
# minimum location, minimum value
0.7077508286455843 0.8713355239628642
# [x1, f1, x2, f2]
[0.7639320225002102, 0.8721428186736883, 1.2360679774997898,
0.9195324670920649]
[0.4721359549995794, 0.8866611708763403, 0.7639320225002102,
0.8721428186736883]
```

```
[0.7639320225002102, 0.8721428186736883, 0.9442719099991589,
0.8838605155861354]
[0.6524758424985279, 0.8721220125648489, 0.7639320225002102,
0.8721428186736883]
[0.5835921350012617, 0.8754571597011437, 0.6524758424985279,
0.8721220125648489]
[0.6524758424985279, 0.8721220125648489, 0.6950483150029442,
0.871373044567459]
[0.6950483150029442, 0.871373044567459, 0.7213595499957939,
0.871387334994435]
[0.6787870774913776, 0.8715444673539482, 0.6950483150029442,
0.871373044567459]
[0.6950483150029442, 0.871373044567459, 0.7050983124842272,
0.8713364562887198]
[0.7050983124842272, 0.8713364562887198, 0.7113095525145109,
0.8713399534896091]
[0.7012595550332279, 0.8713442394539668, 0.7050983124842272,
0.8713364562887198]
[0.7050983124842272, 0.8713364562887198, 0.7074707950635116,
0.8713354513477254]
[0.7074707950635116, 0.8713354513477254, 0.7089370699352264,
0.8713362785522915]
[0.706564587355942, 0.8713354929233407, 0.7074707950635116,
0.8713354513477254]
```

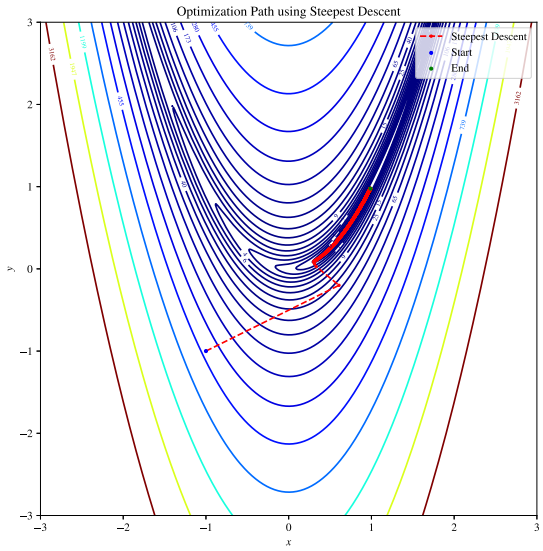
10

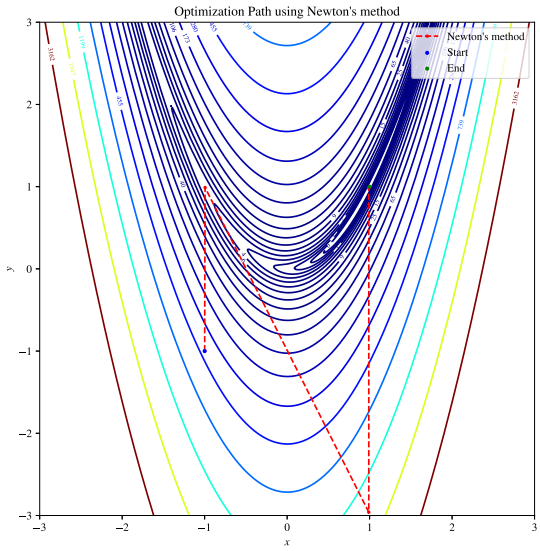
Core Code

```python
def steepest_descent_method(
    start_point, alpha=2e-3, tolerance=1e-5, max_iterations=4000
):
    # Starting point
    x, y = start_point

    # Keep track of the iterations and the path taken
    path = [(x, y)]
    for iteration in range(max_iterations):
        # Calculate the gradient at the current point
        grad = f_g(x, y)

        # Check if the magnitude of the gradient is small enough to stop
        if np.linalg.norm(grad) < tolerance:
            break

        # Update the current point by moving in the opposite direction of
the gradient
        x, y = x - alpha * grad[0], y - alpha * grad[1]

        # Save the new point
        path.append((x, y))
```

```
    # Return the final point and the path
    return (x, y), path


def newton_method(start_point, tolerance=1e-5, max_iterations=4000):
    # Starting point
    x, y = start_point

    # Keep track of the iterations and the path taken
    path = [(x, y)]
    for iteration in range(max_iterations):
        # Calculate the gradient and Hessian at the current point
        grad = f_g(x, y)
        hessian = f_g_g(x, y)

        # Check if the magnitude of the gradient is small enough to stop
        if np.linalg.norm(grad) < tolerance:
            break

        # Calculate the Newton update step
        # Note: We use `np.linalg.inv` to invert the Hessian matrix
        # This could be improved by solving the linear system instead of
inverting the Hessian
        delta = np.linalg.solve(hessian, -grad)

        # Update the current point with the Newton step
        x, y = x + delta[0], y + delta[1]

        # Save the new point
        path.append((x, y))

    # Return the final point and the path
    return (x, y), path
```

Output






```
# minimum location (x,y) path length
(0.9883758370456498, 0.976839953726074) 4001 # steepest_descent
(0.9999999999941024, 0.9999999999882049) 6 # newton
```