Convex Optimization
Homework #03

Submitted By: Rifat Bin Rashid

RUID: 237000174

Problem 1 :-

Given, $f \in C_L^1(\mathbb{R}^n)$

Then, $\forall\ x, y \in \mathrm{dom} f$

$$f(y) \leq f(x) + \nabla f(x)^T (y-x) + \frac{L}{2} \|y-x\|_2^2$$

Let, $y \doteq x^{(k)} + t^{(k)} \Delta x^{(k)} = x^{(k+1)}$

and $x = x^{(k)}$

Then,

$$f\left(x^k + t^{(k)} \Delta x^{(k)}\right) \leq f(x^{(k)}) + \nabla f(x^{(k)})^T \left(\cancel{x^{(k)}} + t^{(k)} \Delta x^{(k)} - \cancel{x^{(k)}}\right) + \frac{L}{2} \| \cancel{x^{(k)}} + t^{(k)} \Delta x^{(k)} - \cancel{x^{(k)}} \|_2^2$$

$$\Rightarrow f\left(x^{(k+1)}\right) \leq f(x^{(k)}) + \nabla f(x^{(k)})^T t^{(k)} \Delta x^{(k)} + t^{(k)^2} \frac{L}{2} \| \Delta x^{(k)} \|_2^2$$

--- ①

Part 1:

For $\Delta x^{(k)}$ to be a descent direction,

we need to have,

$$f(x^{(k+1)}) < f(x^{(k)})$$

from (1)

$$\Rightarrow t^{(k)} \nabla f(x^{(k)})^T \Delta x^{(k)} + {t^{(k)}}^2 \frac{L}{2} \|\Delta x^{(k)}\|_2^2 < 0 \quad \text{----} \; \textcircled{2}$$

Now in these two terms,

$$ {t^{(k)}}^2 \frac{L}{2} \|\Delta x^{(k)}\|_2^2 > 0$$

because, $\begin{cases} t^{(k)} = \text{step size} > 0 \\ L > 0 \end{cases}$

so, for ② to hold: $\Longrightarrow$

$$t^{(k)} \nabla f(x^{(k)})^T \Delta x^{(k)} < 0$$

$$\Rightarrow \nabla f(x^{(k)})^T \Delta x^{(k)} < 0 \; ; [\, t^{(k)} > 0 \,]$$

$$\Rightarrow \left[-\nabla f(x^{(k)})\right]^T \cdot \Delta x^{(k)} > 0$$

so, $\Delta x^{(k)}$ will be a ~~g~~ descent direction if and only if it makes an acute angle with $-\nabla f(x^{(k)})$

But we also have to choose $t^{(k)}$ carefully so that the positive term in eq. 2 ; $t^{(k)^2} \frac{L}{2} \|\Delta x^{(k)}\|_2^2$ does not overweigh the negative term ; $t^{(k)} \nabla f(x^{(k)})^T \Delta x^{(k)}$

Part: 2

$\xrightarrow{\text{From eq } \textcircled{2}}$

$$t^{(k)} \nabla f(x^{(k)})^T \Delta x^{(k)} + t^{(k)^2} \frac{L}{2} \|\Delta x^{(k)}\|_2^2 < 0$$

$$\Rightarrow \quad t^{(k)} < \frac{-\nabla f(x^{(k)})^T \Delta x^{(k)}}{\frac{L}{2} \|\Delta x^{(k)}\|_2^2}$$

So;

$$\boxed{0 < t^{(k)} < -\frac{2}{L} \frac{\Delta f(x^{(k)})^T \Delta x^{(k)}}{\|\Delta x^{(k)}\|_2^2}}$$

Problem 2:

Given, $f(x) = \|x\|_2^4$

from chain rule,

$$\begin{cases} \text{if } h(x) = g(f(x)) \quad \text{where } x \in \mathbb{R}^n \\ D\,h(x) = D_{f(x)}\, g(f(x)) \cdot D_x f(x) \end{cases}$$

for given function,

$$f(x) = \|x\|_2^4$$

let, $g(x) = \|x\|_2^2$

So, $f(x) = \|x\|_2^4 = (g(x))^2$

so, $\nabla f(x) = 2g(x) \cdot \nabla g(x)$

Now, $g(x) = \|x\|_2^2 = x^T x$

So, $\nabla g(x) = 2x$

so, $\nabla f(x) = 2g(x) \cdot 2x = 4\|x\|_2^2 \cdot x$

Now, $\nabla f(x)$ will be lipschitz continous iff

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x-y\|_2, \forall x,y \in \text{dom} f$$

Now,

$$\|\nabla f(x) - \nabla f(y)\|_2 = \|4\|x\|_2^2 x - 4\|y\|_2^2 y\|_2$$

$$= 4\| \|x\|_2^2 x - \|y\|_2^2 y \|_2 .$$

---- ①

Now,

$$\|x\|_2^2 x - \|y\|_2^2 y = \|x\|_2^2 x - \|x\|_2^2 y + \|x\|_2^2 y - \|y\|_2^2 y$$

$$= \|x\|_2^2 (x-y) + (\|x\|_2^2 - \|y\|_2^2) y$$

Now,

$$\| \|x\|_2^2 x - \|y\|_2^2 y \|_2 = \| \|x\|_2^2 (x-y) + (\|x\|_2^2 - \|y\|_2^2) y \|_2$$

---- ②

Using triangular inequality, $\boxed{\|a+b\| \leq \|a\| + \|b\|}$

$$\| \|x\|_2^2 (x-y) + (\|x\|_2^2 - \|y\|_2^2) y \|_2 \leq \| \|x\|_2^2 (x-y)\|_2 + \|(\|x\|_2^2 - \|y\|_2^2) y\|_2$$

--- ③

$$= \|x\|_2^2 \|x-y\|_2 + \left| \|x\|_2^2 - \|y\|_2^2 \right| \|y\|_2 \quad \cdots \text{④}$$

(because, $\|cv\| = |c| \cdot \|v\|$
where c is scaler
v is vector)

<u>Now,</u>

$$\left| \|x\|_2^2 - \|y\|_2^2 \right| \|y\|_2 = \left| (\|x\|_2 + \|y\|_2)(\|x\|_2 - \|y\|_2) \right| \|y\|_2$$

From inverse triangular inequality

$$\|a-b\| \geq \left| \|a\| - \|b\| \right|$$

<u>so,</u>

$$\left| \|x\|_2^2 - \|y\|_2^2 \right| \|y\|_2 \leq \|x-y\|_2 (\|x\|_2 + \|y\|_2) \|y\|_2 \quad \cdots \text{⑤}$$

<u>Combining ③, ④ & ⑤</u>

$$\left\| \|x\|_2^2 x - \|y\|_2^2 y \right\|_2 \leq \|x\|_2^2 \|x-y\|_2 + \|x-y\|_2 (\|x\|_2 + \|y\|_2) \|y\|_2$$

$$= \|x-y\|_2 \left(\|x\|_2^2 + (\|x\|_2 + \|y\|_2) \|y\|_2 \right)$$

Now, $x, y \in \text{dom} f$.

so, $\|x\|_2^2 \leq 1$

$\|y\|_2^2 \leq 1$

$\|x\|_2^2 + \|y\|_2^2 \leq 2$

so,

$$\| \|x\|_2^2 x - \|y\|_2^2 y \|_2 \leq \|x-y\|_2 \,(1+2\cdot1)$$

$$\leq 3 \|x-y\|_2$$

From $\textcircled{1}$ $\longrightarrow$

$$\boxed{\|\nabla f(x) - \nabla f(y)\|_2 \leq 12 \|x-y\|_2}$$

So, $\nabla f(x)$ fullfills the condition of Lipshitz Continous and Comparing this to condition for Lipschitz Continous,

$$\boxed{L = 12}$$

Problem 3:

Strategy : I have completed the code in these five steps explained below.

Step_1: Defining the gradient descent function

At first, I have defined the gradient descent function with the inputs mentioned in the question, namely:

-A function computing the gradient of the objective.
– An initialization point x(0)
– A flag specifying whether to use a fixed step size or a variable step size.
– A step size value (for fixed step size).
– A maximum number of iterations.
– A tolerance ϵ for the stopping criterion.

At first, two things are checked in the function,

1. Positive step size: Gradient descent algorithm is defined with a positive step size. Zero step size will not make any progress toward the minimum, and it will waste computational resources.
2. Step size type: Step size is kept fixed for this problem.

So if positive and fixed step size is not maintained, the function will not work.

Then, until we reach the max number of iterations, as defined in the question, the function stops when the L_2 norm of the gradient falls below the tolerance value, so the stopping criteria is defined in the function as such.

And the iterate was updated using the gradient descent algorithm:

$$\mathbf{x^{(k+1)} = x^{(k)} - \alpha \cdot \nabla f(x^{(k)})}$$

Also, a global iteration count variable was declared to see exactly how many iterations it took in every case to reach the tolerance limit.
So my function for step 1 looks like this :

```
#Step_1: Defining the gradient descent function
def gradient_descent(grad_func, x0, step_flag, alpha,
max_iteration, tolerance):

   global iteration_count  # Declare the global variable first
   #Check if a non-zero step size is given.
   if alpha <= 0:
       print('Step size alpha must be greater than zero.')
       return None

   # Check if step_type is supported
   if step_flag != 'fixed':
       print('Variable step size is currently not supported.')
       return None

   # Initialize
   x = x0
   X = x.reshape(-1, 1)  # Store iterates as columns

   for k in range(max_iteration):
       grad = grad_func(x)  # Compute gradient using the
provided grad_func

       # Check stopping criterion
       grad_norm = 0.0
       for g in grad:
           grad_norm += g**2
       grad_norm = grad_norm**0.5  # Take the square root

       # Check if the gradient norm is below the tolerance,break
if not.
```

```
        if grad_norm <= tolerance:
            iteration_count = k + 1  # Update the global variable
            break

        # Gradient Descent Update
        x = x - alpha * grad

        # Store iterate
        X = np.hstack((X, x.reshape(-1, 1)))

    # If the loop completes without breaking, set iteration_count
to max_iteration
    if grad_norm > tolerance:
        iteration_count = max_iteration

    return X
```

Step_2: Taking the inputs as mentioned in question
Matrix Q1 and Q2 as mentioned in the question and their corresponding step sizes are taken.

Step_3: Function and gradient function
The given quadratic function was :

$$\mathbf{f(x) = \frac{1}{2}\, x^T Q\, x}$$

$$\textbf{So,} \quad \nabla\, \mathbf{f(x) = Q\, x}$$

This is implemented in code as:

```
#Step_3: Function and gradient function
def grad_func1(x):
    return Q1 @ x  # Gradient for Q1
```

```
def grad_func2(x):
    return Q2 @ x  # Gradient for Q2

# Functions for quadratic function values
def f1(x):
    return 0.5 * x.T @ Q1 @ x  # Quadratic function for Q1

def f2(x):
    return 0.5 * x.T @ Q2 @ x  # Quadratic function for Q2
```

Step_4: Hyperparameters
The remaining hyperparameters in the design were initial point, maximum iterations and the stopping criteria tolerance. They were chosen as:

```
#Step_4: Hyperparameters
# Initial point
x0 = np.array([1, 1])

# Maximum iterations and tolerance
max_iter = 2000
tolerance = 1e-7
```

Step_5: Run Gradient Descent and Plot Results
Finally the gradient descent function was run on the given step size and matrix and the result (Contour plot of f(x), Function value and Gradient norm) were plotted in accordance.

Full code for my implementation:

```
import numpy as np
import matplotlib.pyplot as plt
# Global variable to store the number of iterations
iteration_count = 0

#Step_1: Defining the gradient descent function
def gradient_descent(grad_func, x0, step_flag, alpha,
max_iteration, tolerance):

   global iteration_count  # Declare the global variable first
   #Check if a non-zero step size is given.
   if alpha <= 0:
       print('Step size alpha must be greater than zero.')
       return None

   # Check if step_type is supported
   if step_flag != 'fixed':
       print('Variable step size is currently not supported.')
       return None

   # Initialize
   x = x0
   X = x.reshape(-1, 1)  # Store iterates as columns

   for k in range(max_iteration):
       grad = grad_func(x)  # Compute gradient using the
provided grad_func

       # Check stopping criterion
```

```
        grad_norm = 0.0
        for g in grad:
            grad_norm += g**2
        grad_norm = grad_norm**0.5  # Take the square root

        # Check if the gradient norm is below the tolerance,break
if not.
        if grad_norm <= tolerance:
            iteration_count = k + 1  # Update the global variable
            break

        # Gradient Descent Update
        x = x - alpha * grad

        # Store iterate
        X = np.hstack((X, x.reshape(-1, 1)))

    # If the loop completes without breaking, set iteration_count
to max_iteration
    if grad_norm > tolerance:
        iteration_count = max_iteration

    return X

#Step_2: Taking the inputs as mentioned in question
# Define two different Q matrices
Q1 = np.array([[1, 0], [0, 1]])   # Case 1
Q2 = np.array([[10, 0], [0, 1]])  # Case 2

# Step sizes
alphas_Q1 = [0.1, 0.5]
alphas_Q2 = [0.01, 0.05]
```

```
#Step_3: Function and gradient function
# Functions for gradients
def grad_func1(x):
   return Q1 @ x  # Gradient for Q1

def grad_func2(x):
   return Q2 @ x  # Gradient for Q2

# Functions for quadratic function values
def f1(x):
   return 0.5 * x.T @ Q1 @ x  # Quadratic function for Q1

def f2(x):
   return 0.5 * x.T @ Q2 @ x  # Quadratic function for Q2

Q_matrices = [Q1, Q2]
grad_funcs = [grad_func1, grad_func2]
funcs = [f1, f2]
alpha_sets = [alphas_Q1, alphas_Q2]
Q_labels = ['Q1', 'Q2']

#Step_4: Hyperparameters
# Initial point
x0 = np.array([1, 1])

# Maximum iterations and tolerance
max_iter = 2000
tolerance = 1e-7

#Step_5: Run Gradient Descent and Plot Results
# Initialize figure counter
```

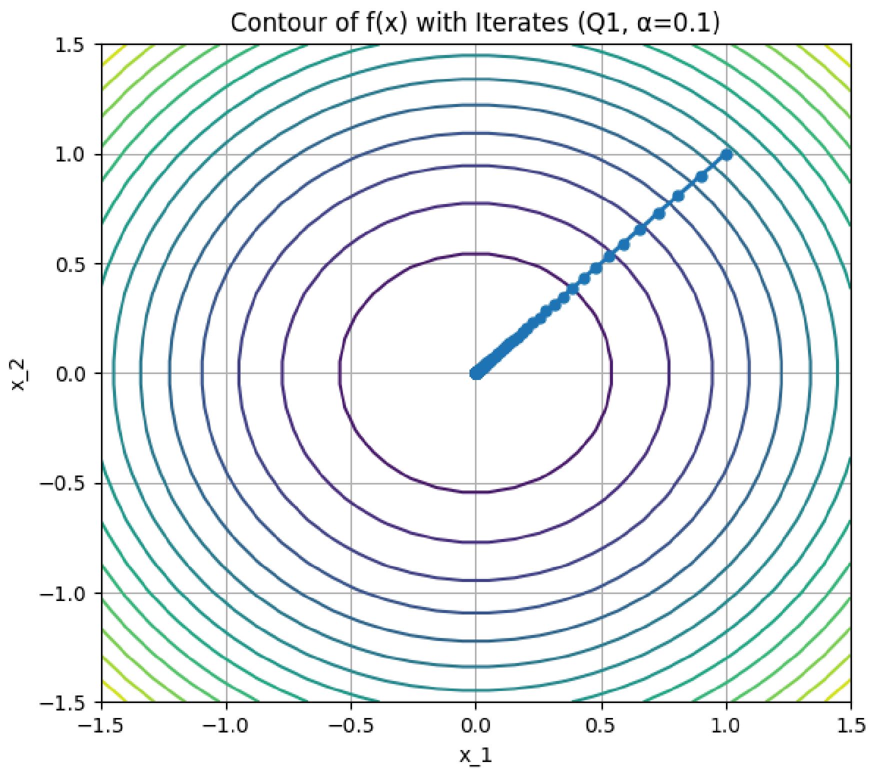
```
figure_number = 1

for i in range(len(Q_matrices)):
   Q = Q_matrices[i]
   grad_func = grad_funcs[i]
   f = funcs[i]
   alphas = alpha_sets[i]
   Q_name = Q_labels[i]

   for j in range(len(alphas)):
       alpha = alphas[j]
       # Reset the global variable before each run
       iteration_count
       iteration_count = 0

       # Run gradient descent
       X = gradient_descent(grad_func, x0, 'fixed', alpha,
max_iter, tolerance)

       # Print the number of iterations for this case
       print(f'Case: {Q_name}, Step size (α): {alpha},
Iterations: {iteration_count}')

       # Compute function values and gradient norms
       num_iters = X.shape[1]
       f_vals = np.zeros(num_iters)
       grad_norms = np.zeros(num_iters)

       for k in range(num_iters):
           f_vals[k] = f(X[:, k])
           grad_norms[k] = np.linalg.norm(grad_func(X[:, k]), 2)
```

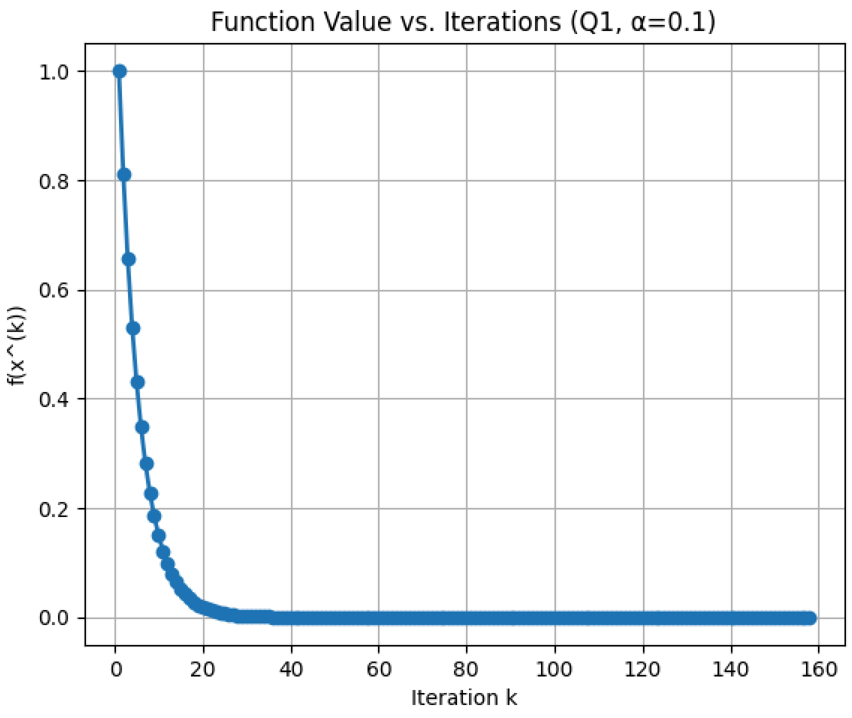
```
        # (a) Contour Plot with Iterates
        plt.figure(figure_number)
        figure_number += 1
        X1, X2 = np.meshgrid(np.linspace(-1.5, 1.5, 30),
np.linspace(-1.5, 1.5, 30))
        F_vals = np.array([[f(np.array([x1, x2])) for x1, x2 in
zip(np.ravel(X1), np.ravel(X2))]]).reshape(X1.shape)
        plt.contour(X1, X2, F_vals, 20)
        plt.plot(X[0, :], X[1, :], '-o', linewidth=2,
markersize=5)
        plt.title(f'Contour of f(x) with Iterates ({Q_name},
α={alpha})')
        plt.xlabel('x_1')
        plt.ylabel('x_2')
        plt.grid(True)
        plt.show()

        # (b) Function Value vs. Iterations
        plt.figure(figure_number)
        figure_number += 1
        plt.plot(range(1, num_iters + 1), f_vals, '-o',
linewidth=2)
        plt.title(f'Function Value vs. Iterations ({Q_name},
α={alpha})')
        plt.xlabel('Iteration k')
        plt.ylabel('f(x^(k))')
        plt.grid(True)
        plt.show()

        # (c) Gradient Norm vs. Iterations
        plt.figure(figure_number)
        figure_number += 1
```

```
       plt.plot(range(1, num_iters + 1), grad_norms, '-s',
linewidth=2)
       plt.title(f'Gradient Norm vs. Iterations ({Q_name},
α={alpha})')
       plt.xlabel('Iteration k')
       plt.ylabel('||∇f(x^(k))||_2')
       plt.grid(True)
       plt.show()
```

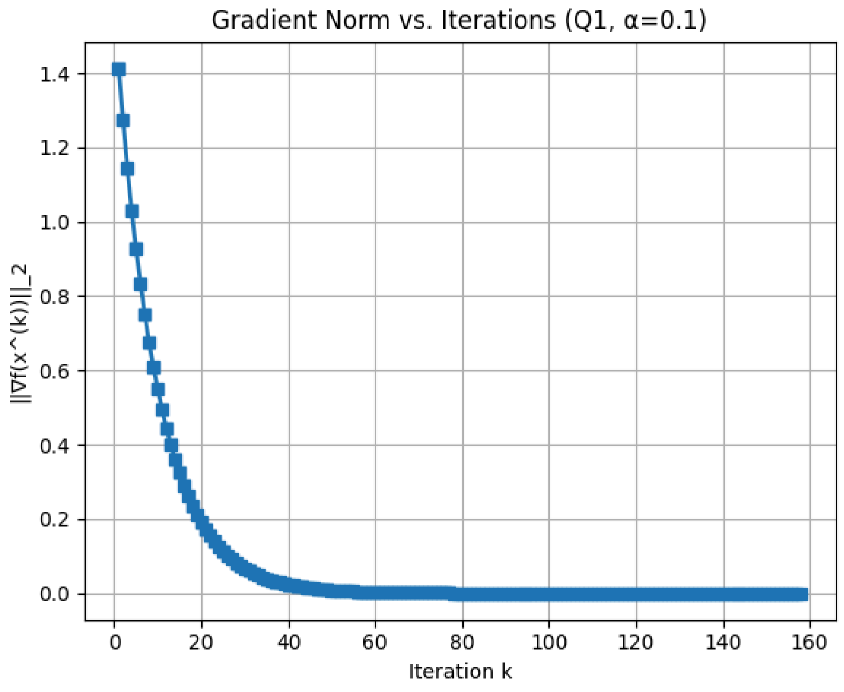
Labeled output results for each case:

Case 1: Q = [1 0 , 0 1], α = 0.1



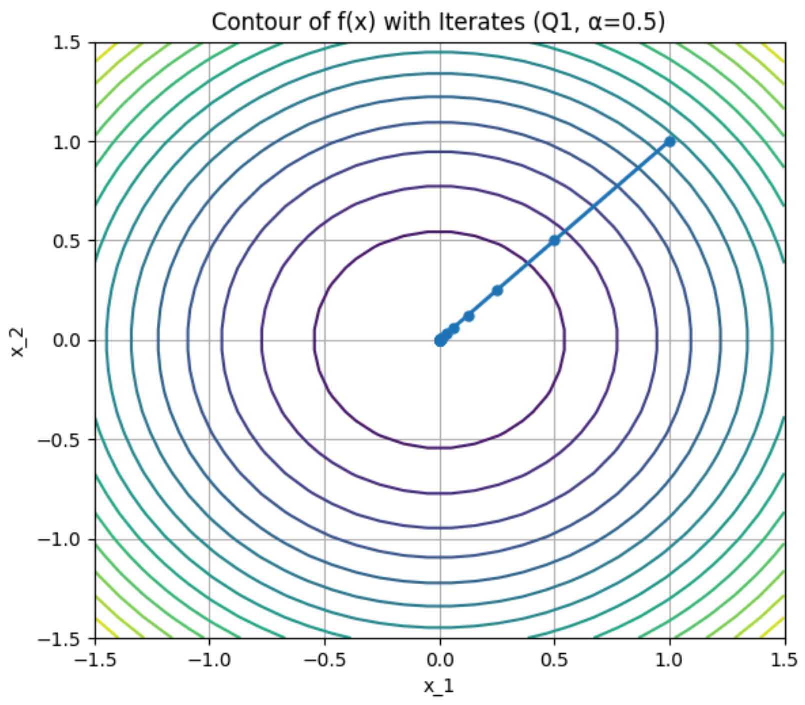


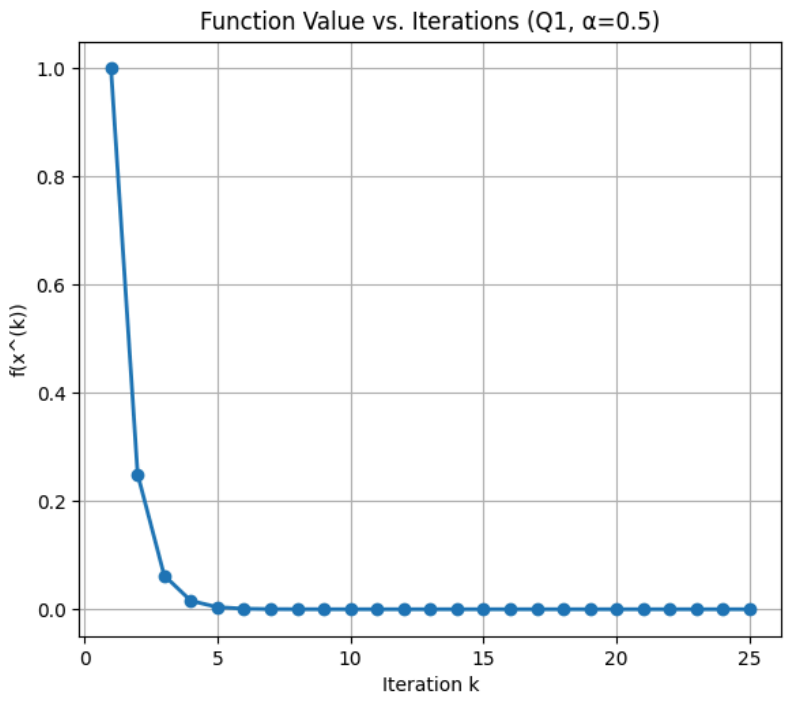
Function Value vs. Iterations (Q1, α=0.1)
f(x^(k))
Iteration k
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160

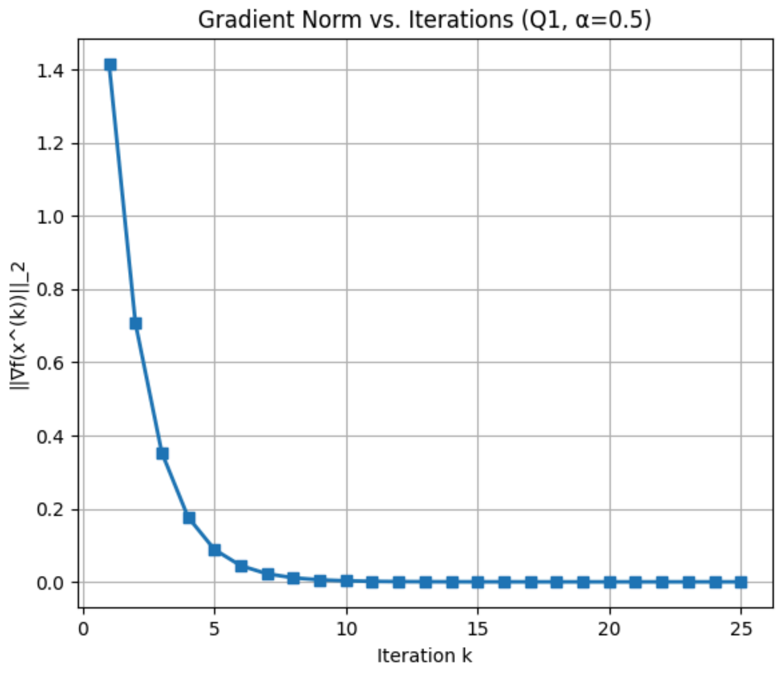


Gradient Norm vs. Iterations (Q1, α=0.1)
||∇f(x^(k))||_2
Iteration k
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140
160

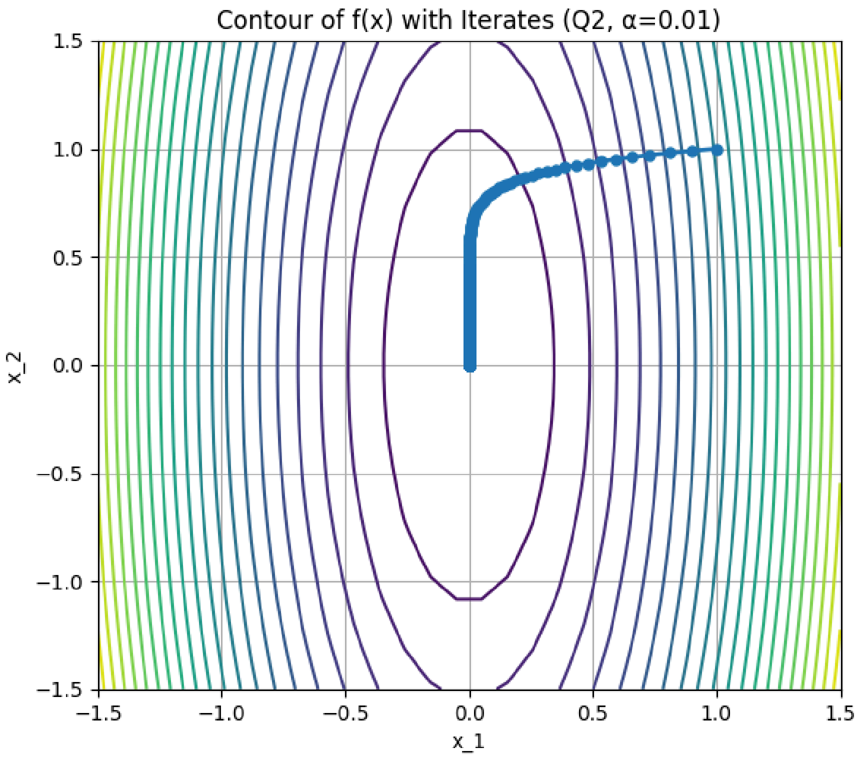
Case 2 : Q = [1 0 , 0 1], α = 0.5

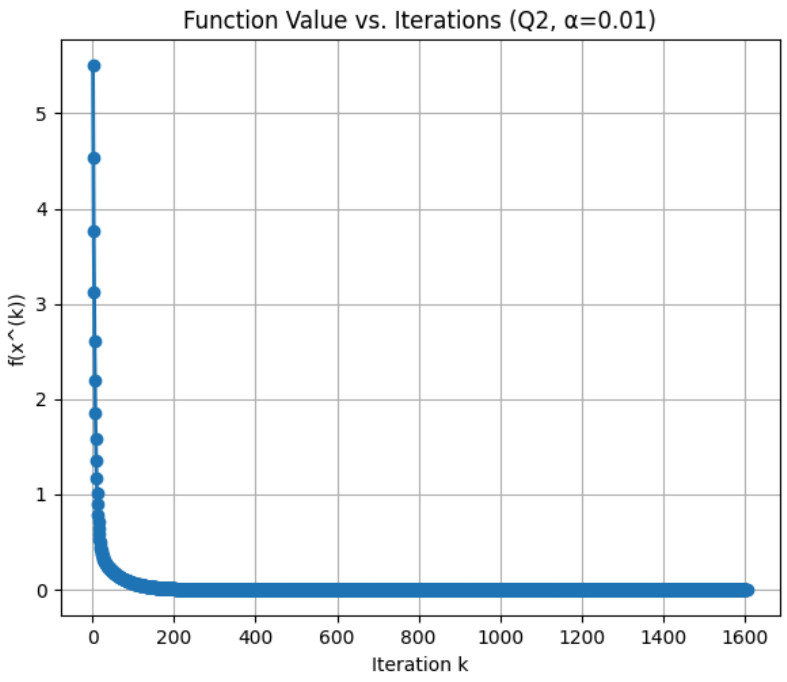


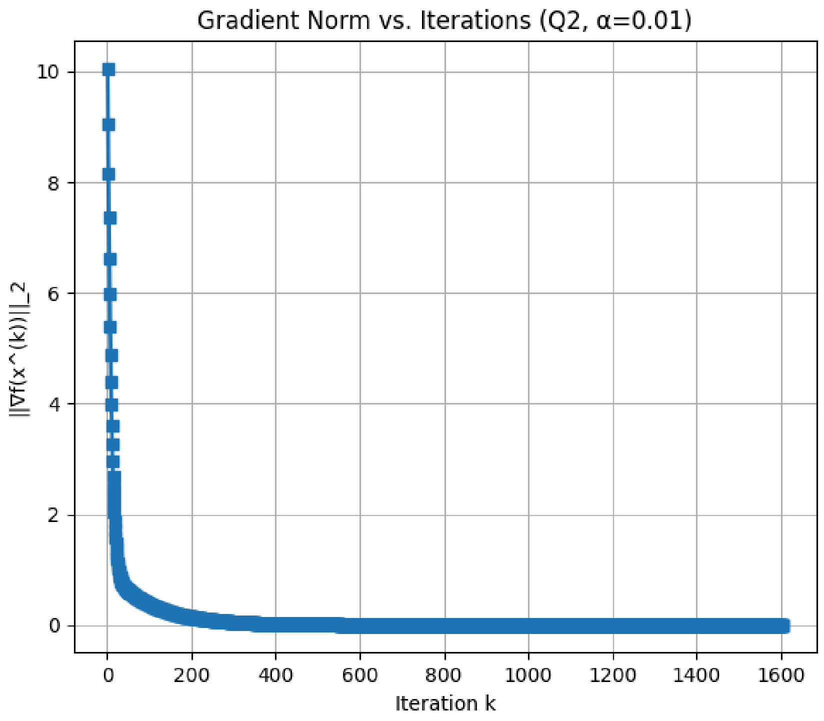




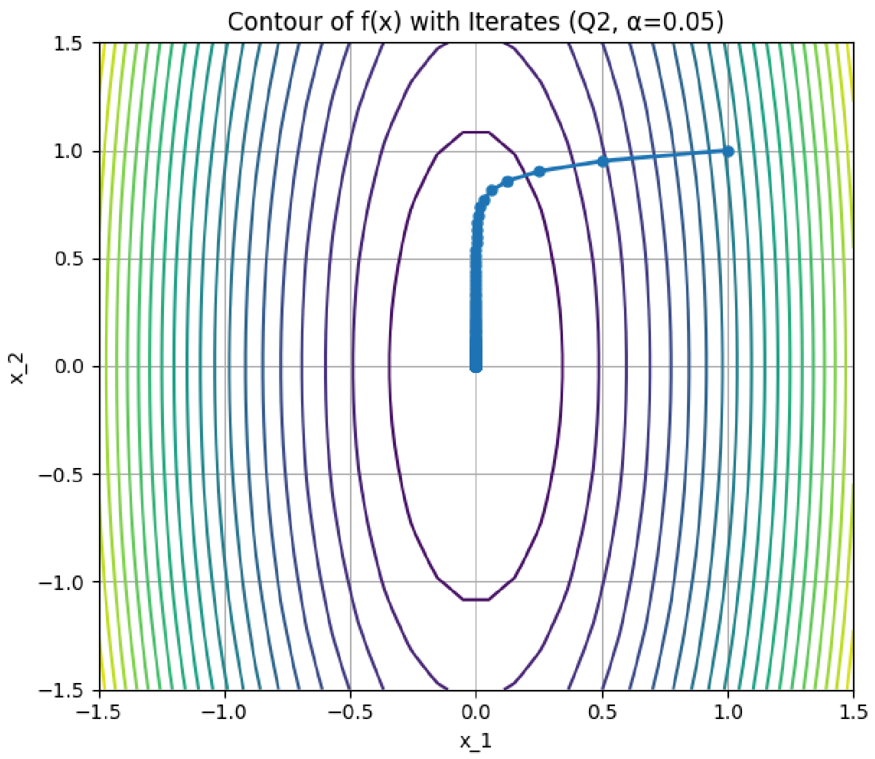
Case 3: Q = [10 0, 0 1], α = 0.01

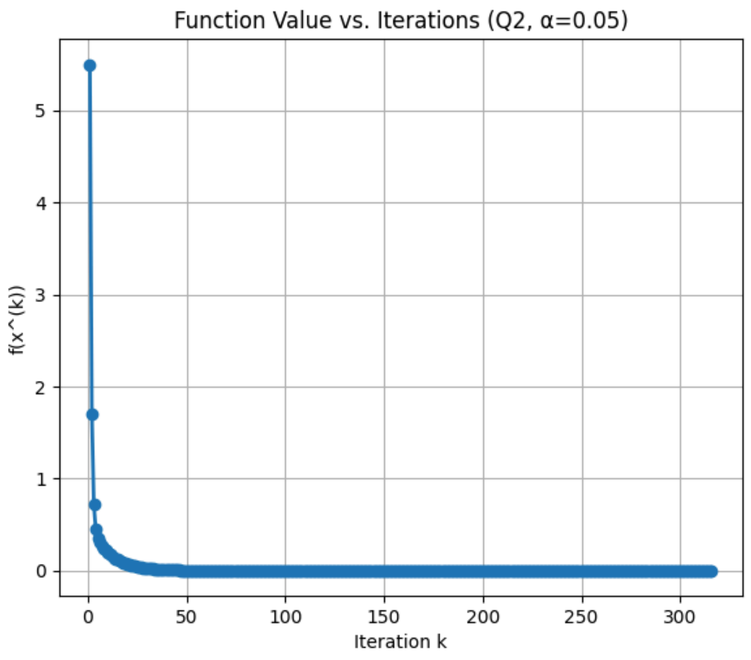


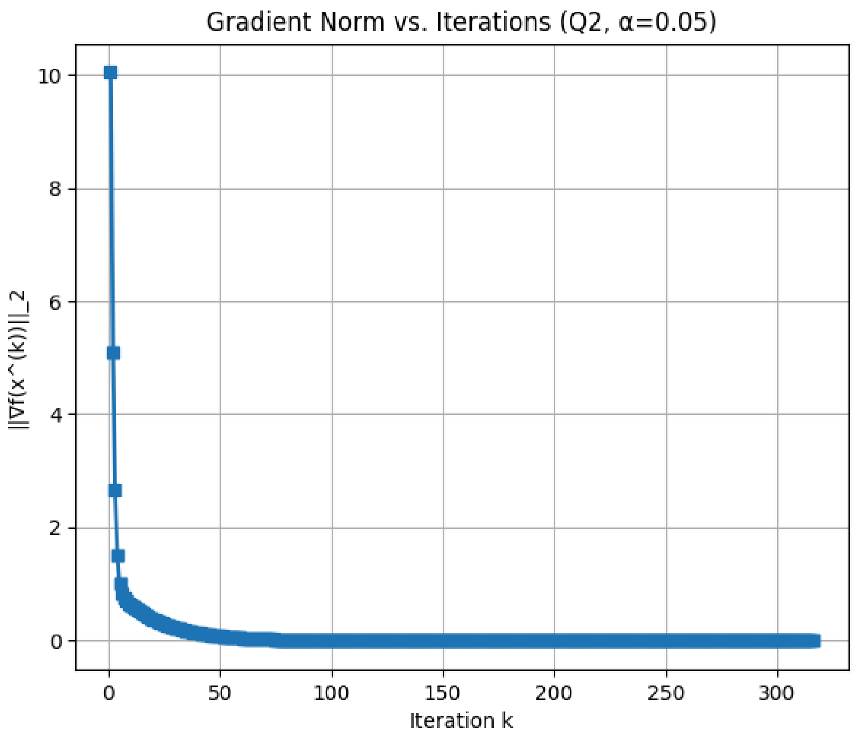

Function Value vs. Iterations (Q2, α=0.01)
f(x^(k))
Iteration k
0
1
2
3
4
5
0
200
400
600
800
1000
1200
1400
1600


Gradient Norm vs. Iterations (Q2, α=0.01)
||∇f(x^(k))||_2
Iteration k
0
2
4
6
8
10
0
200
400
600
800
1000
1200
1400
1600

Case 4 : Q = [10 0, 0 1], α = 0.05







Questions based on my results:

From my results and the plots:
Case: Q1, Step size (α): 0.1, Iterations: 158
Case: Q1, Step size (α): 0.5, Iterations: 25
Case: Q2, Step size (α): 0.01, Iterations: 1605
Case: Q2, Step size (α): 0.05, Iterations: 316

Q1. How does the choice of step size affect convergence behavior?

As we can see from the plots, for the smaller step sizes (like α=0.01 for Q2), the algorithm converges very slowly. It was expected because a small step size means the algorithm takes tiny steps toward the minimum, which slows down convergence.

On the other hand, for bigger step sizes (like α=0.5 for Q1), the algorithm converges very quickly. It was seen because a larger step size means the algorithm is now taking giant steps toward the minimum, which speeds up convergence.

However, this may not always be the case as the function can overshoot for very large step sizes.

For Moderate Step Size (like α=0.1 for Q1 or α=0.05 for Q2), the algorithm converges relatively quickly. Here, fewer iterations are needed to reach the minimum compared to a small step size. This moderate step size balances between convergence speed and stability.

Q2. How does changing the matrix Q affect convergence?

Here, The given quadratic function was :

$$\mathbf{f(x) = \frac{1}{2}\, x^T Q\, x}$$

So, changing Q changes the shape of the overall quadratic function f(x), which eventually affects the convergence.

From the contour plot, for Q1, the contour plot shows circular contours. This indicates that f(x) is equally scaled in all directions for Q1. And from the function and gradient plots for Q1, we can see that gradient descent converges quickly and smoothly for both step sizes (α=0.1 and α=0.5). Most importantly, as evident from the contour plot, the algorithm takes a direct path toward the minimum for Q1, with larger step sizes leading to faster convergence.

On the other hand, for Q2, the contour plot shows elliptical contours. This indicates that f(x) is not equally scaled in all directions for Q2. And from the function and gradient plots for Q1, we can see that gradient descent converges slowly for both step sizes (α=0.01, α=0.05). Here, the algorithm takes a curved path toward the minimum for Q2, with larger step sizes leading to faster convergence, but overall slower convergence compared to Q1.
So, for a symmetric matrix like Q1, the algorithm converges quickly and smoothly. And for stretched matrices like Q2, the algorithm converges more slowly and may zigzag toward the minimum.