AI505/AI801, Optimization – Exercise Sheet 04

2026-03-07

Solutions included.

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1 * (4.1)

Find examples where each of the four termination conditions would not work individually, showing the importance of having more than one.

- Step-size termination condition: Consider running a descent method on $f(x) = 1/x$ for $x > 0$. The minimum does not exist and the descent method will forever proceed in the positive x direction with ever-increasing step sizes.
- Terminating based on gradient magnitude: a descent method applied to $f(x) = -x$ will also forever proceed in the positive x direction. The function is unbounded below, so neither a step-size termination condition nor a gradient magnitude termination condition would trigger.
- Termination condition to limit the number of iterations: always in effect.

Exercise 2 * (6.1)

What advantage does second-order information provide about the point of convergence that first-order information lacks?

Second-order information can guarantee that one is at a local minimum, whereas a gradient of zero is necessary but insufficient to guarantee local optimality.

Exercise 3 * (6.2)

When would we use Newton's method instead of the bisection method for the task of finding roots in one dimension?

We would prefer Newtons method if we start sufficiently close to the root and can compute derivatives analytically. Newton's method enjoys a better rate of convergence.

Exercise 4 * (6.4, 6.9)

Apply Newton's method to $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T H \boldsymbol{x}$ starting from $\boldsymbol{x}_0 = [1, 1]$. What have you observed? Use H as follows:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix}$$

Next, apply gradient descent to the same optimization problem by stepping with the unnormalized gradient. Do two steps of the algorithm. What have you observed? Finally, apply the conjugate gradient method. How many steps do you need to converge?

Repeat the exercise for:

$$f(\boldsymbol{x}) = (x_1 + 1)^2 + (x_2 + 3)^2 + 4.$$

starting at the origin.

Newton's method updates design points (solutions) as follows: As H is a symmetric matrix, the Hessian is simply H and $\nabla f(\boldsymbol{x}_k) = H\boldsymbol{x}_k$. Thus, Newton's update looks like:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H^{-1}(H\boldsymbol{x}_k)$$

This implies,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{x}_k = 0$$

Hence

$$\boldsymbol{x}_1 = 0$$

and the search finishes in one step because the following ones would not yield any change.
Gradient Descent update rule is as follows for unnormalized case:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)$$

and subtituting $\nabla f(\boldsymbol{x}_k) = H\boldsymbol{x}_k$ yields:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H\boldsymbol{x}_k = [I - \alpha H]\boldsymbol{x}_k$$

Let's try first to set $\alpha = 1$. This will reveal itself for being a bad choice. In this case, the first iterates yield:

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -999 \end{bmatrix}$$

$$\boldsymbol{x}_2 = \begin{bmatrix} 0 \\ -999 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix} \begin{bmatrix} 0 \\ -999 \end{bmatrix} = \begin{bmatrix} 0 \\ 998111 \end{bmatrix}$$

$$\boldsymbol{x}_3 = \begin{bmatrix} 0 \\ 998001 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix} \begin{bmatrix} 0 \\ 998001 \end{bmatrix} = \begin{bmatrix} 0 \\ -997002999 \end{bmatrix}$$

The gradient descent method diverges.
Let's backtrack and analyse the update rule more generally. We have:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H\boldsymbol{x}_k = [I - \alpha H]\boldsymbol{x}_k$$

.

We can regard this as a stochastic process where the state $\boldsymbol{x}_k$ is updated by multiplying with a matrix $I - \alpha H$. In particular, it is a Markov process of first order where the new state depends only on the current state and not on all the previous states. The theory to analyse the convergence of these processes is an application of Linear Algebra that is also behind the pagerank algorithm, see for example this class from AI511. The convergence of this process depends on the spectral radius of the matrix $I - \alpha H$. In particular, for convergence we require that all eigenvalues of $I - \alpha H$ have magnitude less than 1.
Recursing on the update rule we get:

$$\boldsymbol{x}_k = [I - \alpha H]^k \boldsymbol{x}_0$$

Now using our specific H matrix, we get:

$$\boldsymbol{x}_k = \begin{bmatrix} 1-\alpha & 0 \\ 0 & 1-1000\alpha \end{bmatrix}^k \boldsymbol{x}_0$$

It is easy to see that:

$$\boldsymbol{x}_k = \begin{bmatrix} (1-\alpha)^k & 0 \\ 0 & (1-1000\alpha)^k \end{bmatrix} \boldsymbol{x}_0$$

To have convergence (ie, $\boldsymbol{x}_{k+1} = x_k$), we require α to satisfy the following equations: $|1-\alpha| < 1$ and $|1-1000\alpha| < 1$. Solving these we get that choosing $0 < \alpha < 2/1000$ ensures GD converges to the minimizer. However, note that it does not converge as fast as Newton's method.
Conjugate gradient

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \qquad \boldsymbol{d}_k = -\boldsymbol{r}_k + \beta_k \boldsymbol{d}_{k-1}$$

$$\boldsymbol{r}_k = \nabla f(\boldsymbol{x}_k) = H\boldsymbol{x}_k \qquad \beta_k = \frac{\boldsymbol{r}_k^T H \boldsymbol{d}_{k-1}}{\boldsymbol{d}_{k-1}^T H \boldsymbol{d}_{k-1}} \qquad \alpha_k = -\frac{\boldsymbol{r}_k \boldsymbol{d}_k}{\boldsymbol{d}_k^T H \boldsymbol{d}_k}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha_0 \boldsymbol{d}_0$$

We choose the first search direction $\boldsymbol{d}_0$ to be the steepest descent direction at the initial point $\boldsymbol{x}_0$. Hence, $\boldsymbol{d}_0 = -\nabla f(\boldsymbol{x}_0) = -H\boldsymbol{x}_0$ and

$$\alpha_0 = -\frac{(H\boldsymbol{x}_0)^T(-H\boldsymbol{x}_0)}{(-H\boldsymbol{x}_0)^T H(-H\boldsymbol{x}_0)} = \frac{1+10^6}{1+10^9} \approxeq 0.001$$

The first iteration then yields:

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \alpha_0 \boldsymbol{d}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.001 \begin{bmatrix} 1 \\ 1000 \end{bmatrix} = \begin{bmatrix} 0.999 \\ 0 \end{bmatrix}$$

The second iteration:

$$\boldsymbol{x}_2 = \boldsymbol{x}_1 + \alpha_1 \boldsymbol{d}_1$$

Setting $r_1 = H\boldsymbol{x}_1$:

$$\beta_1 = \frac{\boldsymbol{r}_1^T H \boldsymbol{d}_0}{\boldsymbol{d}_0^T H \boldsymbol{d}_0} = \frac{(H\boldsymbol{x}_1)^T H (H\boldsymbol{d}_0)}{\boldsymbol{d}_0^T H \boldsymbol{d}_0} = \dots$$

$$\boldsymbol{d}_1 = -\boldsymbol{r}_1 + \beta_1 \boldsymbol{d}_0 = \dots$$

$$\alpha_1 = -\frac{(H\boldsymbol{x}_1)^T \boldsymbol{d}_0}{\boldsymbol{d}_0^T H \boldsymbol{d}_0} = \dots$$

$$\boldsymbol{x}_2 \approx \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The matrix H has eigenvalues 1 and 1000 that are both positive, hence the matrix is positive definite (and hence symmetric). In this case, the conjugate gradient method needs n iterations to find the local optimum. So $\boldsymbol{x}_2$ should be the optimal solution.
The function:

$$f(\boldsymbol{x}) = (x_1+1)^2 + (x_2+3)^2 + 4 = x_1^2 + x_2^2 + 2x_1 + 6x_2 + 9 + 4 + 1$$

Can be obtained by

$$f(\boldsymbol{x}) = 1/2\boldsymbol{x}^T A\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$$

where $\boldsymbol{x} = [x_1, x_2]^T$, $\boldsymbol{b} = [2, 6]$, $c = 14$ and

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The matrix A is positive definite since its eigenvalues are positive. Hence, this is a quadratic function where the Netwon's method finds the optimal solution in one iteration.

$$\nabla f(\boldsymbol{x}) = [2(x_1 + 1), 2(x_2 + 3)]$$

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - H^{-1}\nabla(\boldsymbol{x}_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} -1 \\ -3 \end{bmatrix}$$

Note that $H = A$, hence we could have derived A also by calculating the Hessian.

Exercise 5 * (6.5)

Compare Newton's method and the secant method on $f(x) = x^2 + x^4$, with $x_1 = -3$ and $x_0 = -4$. Run each method for 10 iterations. Make two plots:

1. Plot f vs. the iteration for each method.
2. Plot f' vs. x. Overlay the progression of each method, drawing lines from $(x_i, f'(x_i))$ to $(x_{i+1}, 0)$ to $(x_{i+1}, f'(x_{i+1}))$ for each transition.

What can we conclude about this comparison?

Note that Newton's method is used both for finding roots and for finding minima. When used for finding roots it is also known as Newton-Raphson method. Compare:

Newton-Raphson method	Solve $f(x) = 0$	First derivative $f'(x)$	$x_{k+1} = x_k - \frac{f(x)}{f'(x)}$
Newton's method	Minimize $f(x)$	First and second derivative $f'(x), f''(x)$	$x_{k+1} = x_k - \frac{f'(x)}{f''(x)}$

Its first derivative is $f'(x) = 2x + 4x^3$.
Its second derivative is $f'(x) = 2 + 12x^2$.
Newton's method updates the estimate x_k using the formula:

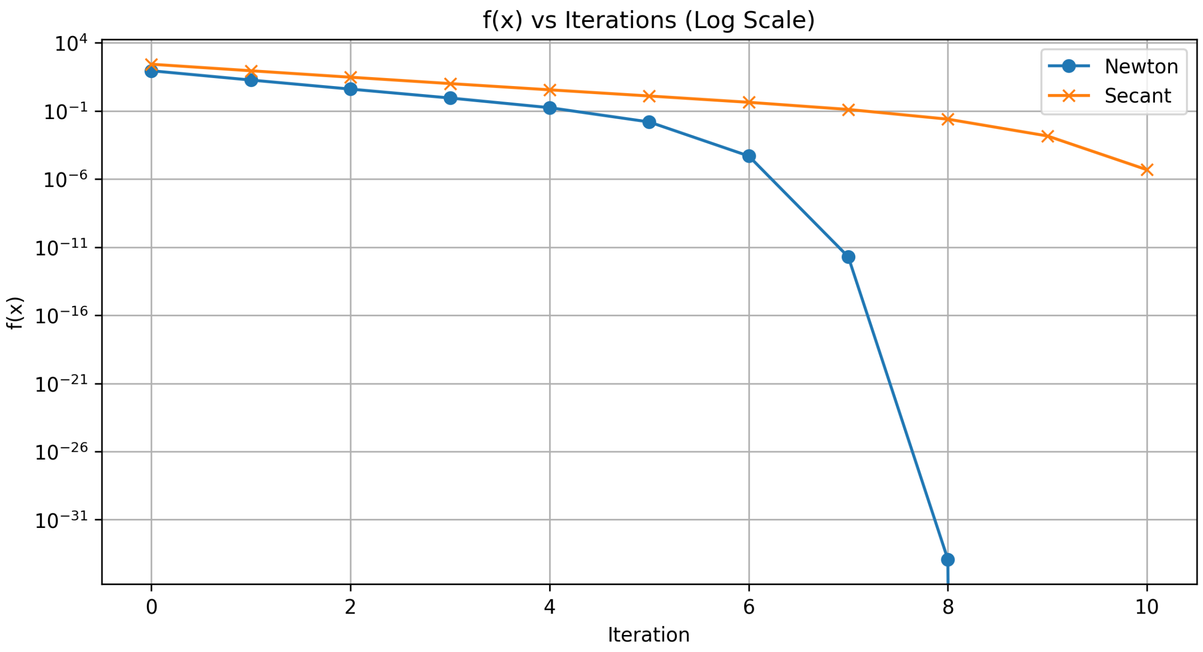
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

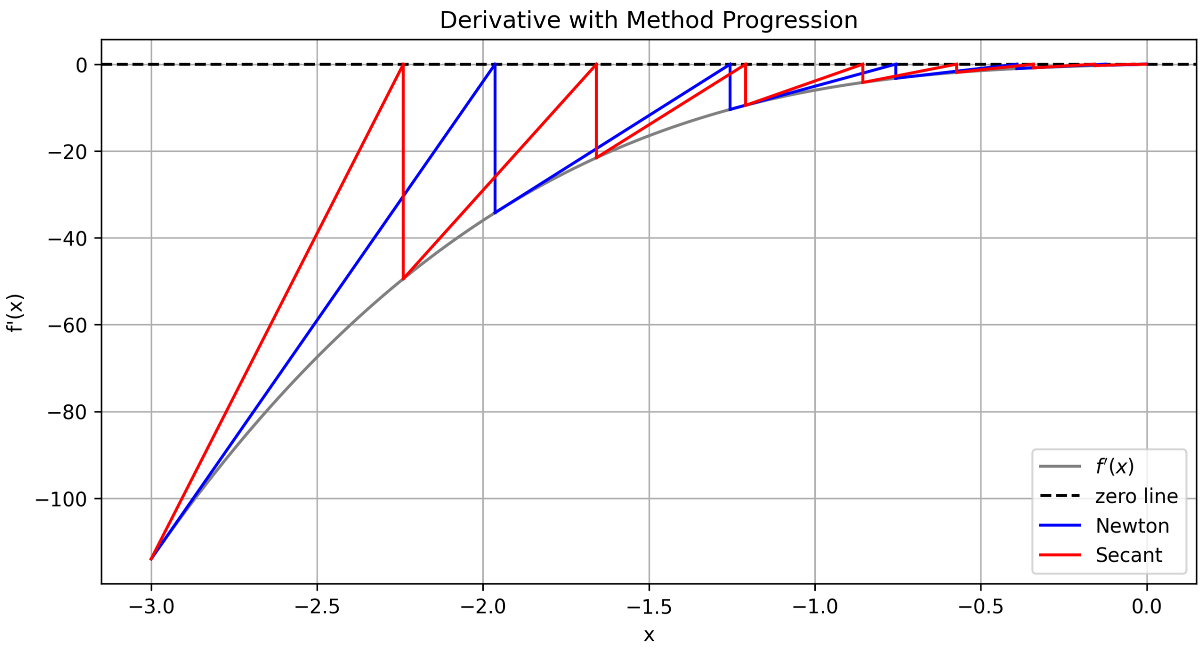
The secant method updates the estimate x_n using the formula:

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

Note that the secant method requires two initial guesses, x_0 and x_1.

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function and its derivative
def f(x):
    return x**2 + x**4

def df(x):
    return 2*x + 4*x**3

def ddf(x):
    return 2 + 12*x**2


# Newton's method
def newton_method(x0, iterations):
    xs = [x0]
    for _ in range(iterations):
        x0 = x0 - df(x0) / ddf(x0)
        xs.append(x0)
    return np.array(xs)

# Secant method
def secant_method(x0, x1, iterations):
    xs = [x0, x1]
    for _ in range(iterations - 1):
        x_new = xs[-1] - df(xs[-1]) * (xs[-1] - xs[-2]) / (df(xs[-1]) - df(xs[-2]))
        xs.append(x_new)
    return np.array(xs)

# Plot f(x) vs iteration
def plot_function_vs_iteration(x_newton: np.array, x_secant: np.array) -> None:

    plt.figure(figsize=(10, 5))
    plt.plot(range(len(x_newton) ), f(x_newton), 'o-', label="Newton")
    plt.plot(range(len(x_secant) ), f(x_secant), 'x-', label="Secant")
    plt.yscale("log")
    plt.xlabel("Iteration")
    plt.ylabel("f(x)")
    plt.title("f(x) vs Iterations (Log Scale)")
    plt.legend()
    plt.grid()
    plt.show()
    #plt.savefig("secant_function.png")


# Plot f'(x) vs x with transitions
def plot_derivative(x_newton: np.array, x_secant: np.array) -> None:
    x_vals = np.linspace(-3, 0, 1000)
    plt.figure(figsize=(10, 5))
    line_fx, = plt.plot(x_vals, df(x_vals), color='grey', label="$f'(x)$")
    line_zero = plt.axhline(0, color='black', linestyle='--', label="zero line")

    # Newton's method
    newton_lines = []
    for i in range(len(x_newton) - 1):
        l, = plt.plot([x_newton[i], x_newton[i+1]], [df(x_newton[i]), 0], 'b-')
        plt.plot([x_newton[i+1], x_newton[i+1]], [0, df(x_newton[i+1])], 'b-')
        if i == 0:
            newton_lines.append(l)

    # Secant method
```





The first plot illustrates how quickly each method converges to a local optimum.
The second plot visualizes how each method approaches the root in the context of the derivative's behavior.
Conclusions from the comparison:

- Convergence Rate: Newton's method typically exhibits quadratic convergence, meaning the error decreases exponentially with each iteration when close to the root. The secant method generally has a convergence rate of approximately 1.618 (the golden ratio), which is superlinear but slower than Newton's method.
- Initial Guess Sensitivity: Newton's method requires a good initial guess to ensure convergence, as poor initial guesses can lead to divergence or convergence to an unintended root. The secant method, while generally more robust to initial guesses, can still fail to converge if the initial guesses are not

chosen appropriately.

- Computational Efficiency: If derivative computation is costly or impractical, the secant method may be preferred despite its slower convergence rate. However, if the derivative is readily available and computationally inexpensive, Newton's method's faster convergence can lead to quicker results.

Exercise 6 Strongly Convex Functions

Consider:

$$f(x) = x^4.$$

- Show that f is convex.
- Compute $f''(x)$.
- Show that f is not strongly convex on $\mathbb{R}$.
- Explain geometrically why the function fails to be strongly convex.

Let's first compute the first and second derivatives of f:

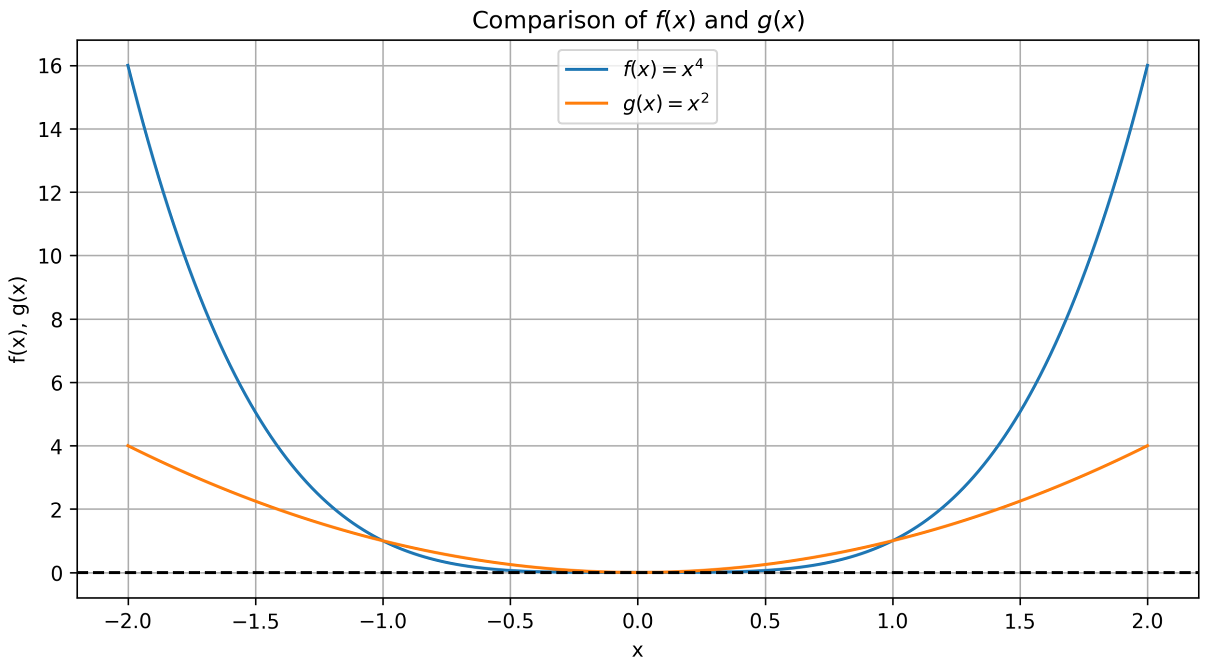
$$f'(x) = 4x^3$$

$$f''(x) = 12x^2$$

1. Convexity: A function is convex if its second derivative is non-negative for all x. Since $f''(x) = 12x^2$ is always greater than or equal to zero, f is convex.
2. $f''(x) = 12x^2$
3. Strong convexity would require:

$$ f' '(x) c > 0 for all x. But $f''(0) = 0$. Therefore no uniform positive lower bound exists. f is convex but not strongly convex.
Notice how the curvature goes to zero at the minimum. Strong convexity requires quadratic growth everywhere.
In contrast, the function $g(x) = x^2$ is strongly convex with $g''(x) = 2$ and hence positive for all x.
Let's compare these two functions graphically.

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 400)
f_x = x**4
g_x = x**2
plt.figure(figsize=(10, 5))
plt.plot(x, f_x, label='$f(x) = x^4$')
plt.plot(x, g_x, label='$g(x) = x^2$')
plt.axhline(0, color='black', linestyle='--')
plt.xlabel('x')
plt.ylabel('f(x), g(x)')
plt.title('Comparison of $f(x)$ and $g(x)$')
plt.legend()
plt.grid()
plt.show()
```



Exercise 7 Strongly Convex Functions

Recall that for a symmetric matrix A,

$$A \succeq 0$$

means that A is positive semidefinite, i.e.

$$\boldsymbol{x}^\top A\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n.$$

Hence, for Q a symmetric matrix, and I the identity matrix, the condition

$$Q \succeq cI$$

means:

$$Q - cI \succeq 0.$$

In scalar form this means:

$$\boldsymbol{x}^T(Q - cI)\boldsymbol{x} = \boldsymbol{x}^\top Q\boldsymbol{x} - c\|\boldsymbol{x}\|^2 \geq 0.$$

So, in its easiest form, the condition is:

$$\boldsymbol{x}^\top Q\boldsymbol{x} \geq c\|\boldsymbol{x}\|^2 \quad \forall \boldsymbol{x}.$$

Assume f is a twice-differentiable function (ie, $f \in C^2(\mathbb{R}^n)$). Show that:

$$f \text{ is } c\text{-strongly convex} \iff \nabla^2 f(\boldsymbol{x}) \succeq cI \text{ for all } \boldsymbol{x} \in \mathbb{R}^n.$$

(Hint: Use Taylor's theorem with integral remainder.)

($\Longrightarrow$)
If f is c-strongly convex:

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x}) + \frac{c}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2.$$

Using Taylor expansion:

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^\top \nabla^2 f(\xi)(\boldsymbol{y} - \boldsymbol{x})$$

Comparing:

$$\frac{c}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2 \leq \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^\top \nabla^2 f(\xi)(\boldsymbol{y} - \boldsymbol{x}).$$

Thus, since $\boldsymbol{x}, \boldsymbol{y}$ are arbitrarily chosen:

$$\nabla^2 f(\boldsymbol{\xi}) \succeq cI.$$

($\Longleftarrow$)
If

$$\nabla^2 f(\boldsymbol{x}) \succeq cI.$$

then substituting its equivalent scalar form $\boldsymbol{\xi}^\top \nabla^2 f(\boldsymbol{\xi})\boldsymbol{\xi} \geq c\|\boldsymbol{\xi}\|^2$ for any $\boldsymbol{\xi}$ and hence also for $\boldsymbol{\xi} = \boldsymbol{y} - \boldsymbol{x}$ in Taylor's expansion yields:

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top(\boldsymbol{y} - \boldsymbol{x}) + \frac{c}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2.$$

Thus f is strongly convex.

Exercise 8

Implement gradient descent for:

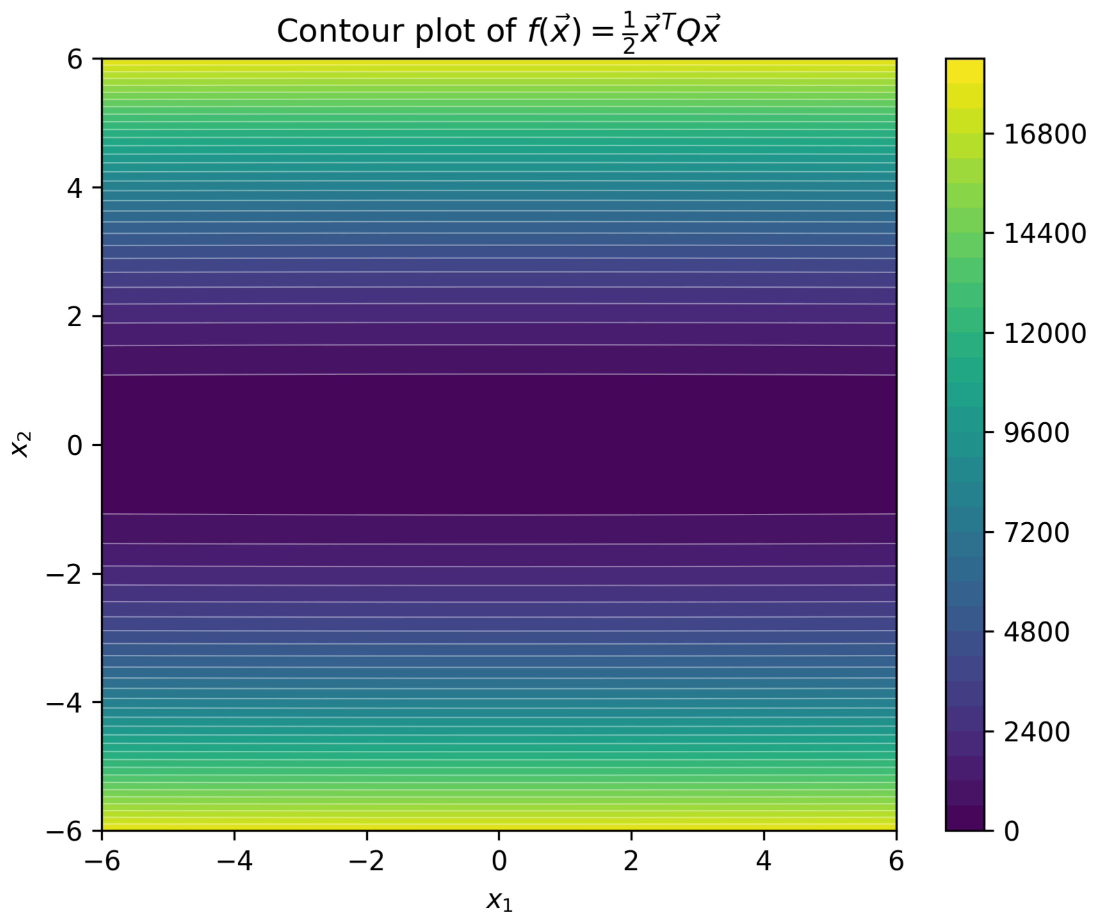
$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T Q \boldsymbol{x}$$

where

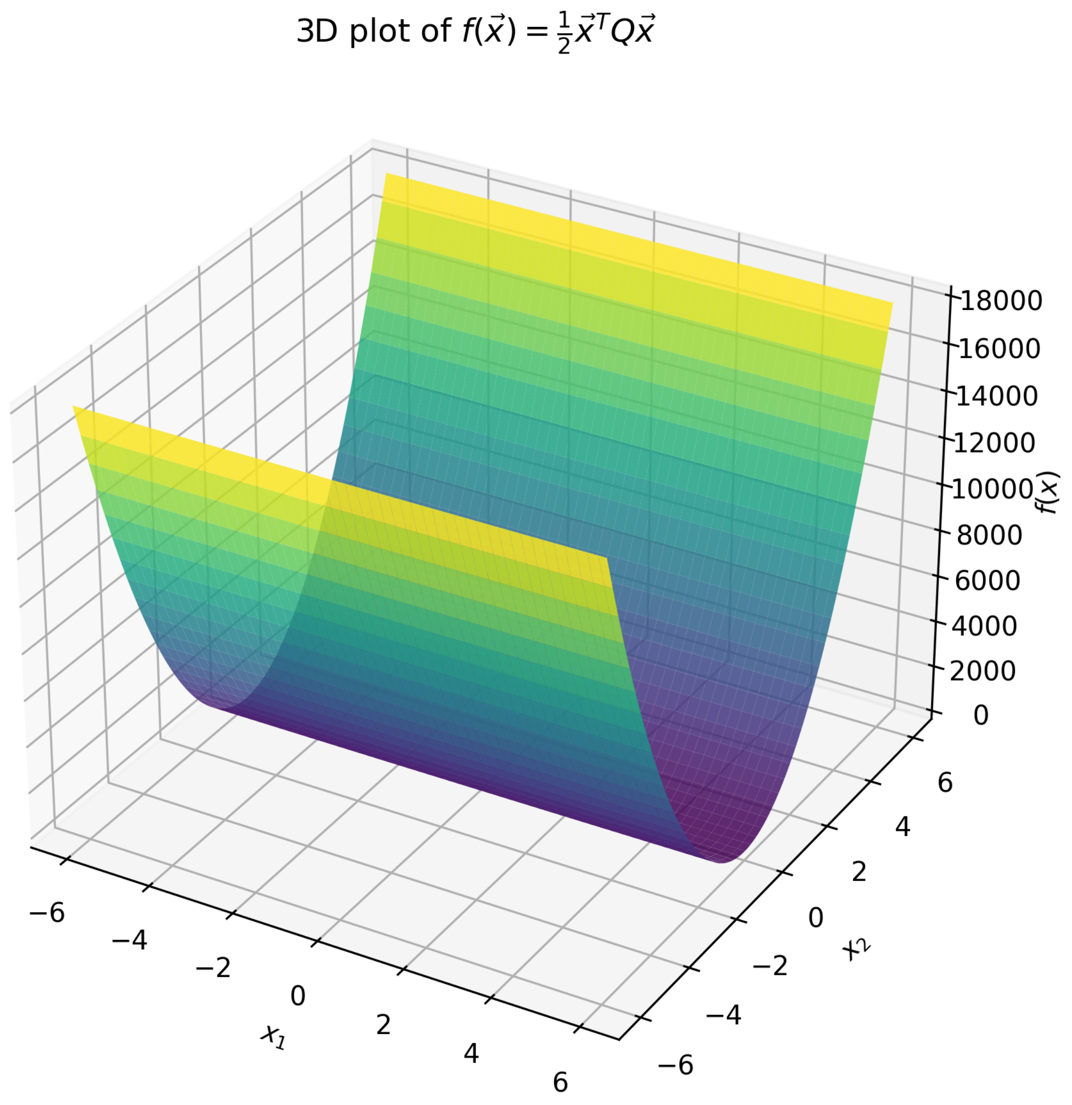
$$Q = \begin{bmatrix} 1 & 0 \\ 0 & \gamma \end{bmatrix}.$$

- Run experiments for $\gamma = 1, 1/10, 1/100, 1/1000$.
- Plot convergence.
- Relate the behavior to the condition number.
- Observe how increasing strong convexity improves convergence.

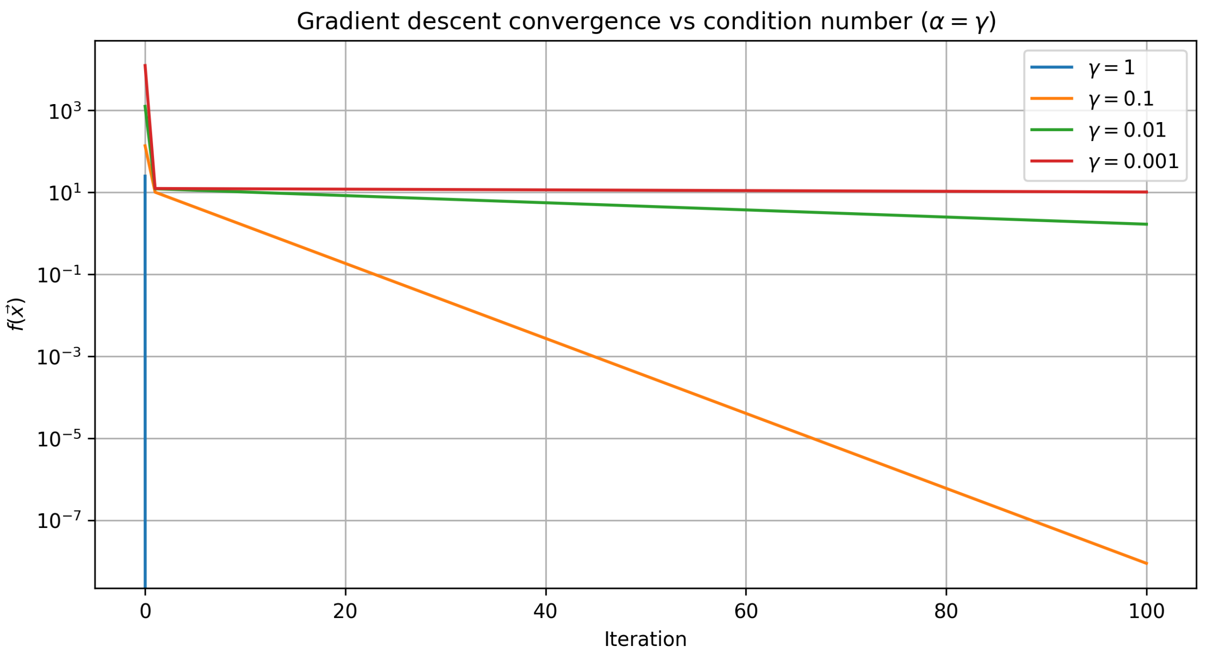
```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function and its derivative
def f(x1, x2, Q):
    x = np.array([x1, x2])
    return 0.5 * x.T @ Q @ x

def df(x1, x2, Q):
    x = np.array([x1, x2])
    return Q @ x

def ddf(x, Q):
    return np.diag(Q)


# Contour plot of f
def contour_plot(Q):
    x1 = np.linspace(-6, 6, 300)
    x2 = np.linspace(-6, 6, 300)
    X1, X2 = np.meshgrid(x1, x2)
    Z = 0.5 * (Q[0, 0] * X1**2 + Q[1, 1] * X2**2)
    plt.figure(figsize=(6, 5))
    cp = plt.contourf(X1, X2, Z, levels=30, cmap='viridis')
    plt.colorbar(cp)
    plt.contour(X1, X2, Z, levels=30, colors='white', linewidths=0.5, alpha=0.4)
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title(r'Contour plot of $f(\vec x) = \frac{1}{2}\vec x^T Q \vec x$')
    plt.tight_layout()
    plt.show()


# 3D plot of f
def plot_3d(Q):
    x1 = np.linspace(-6, 6, 200)
    x2 = np.linspace(-6, 6, 200)
    X1, X2 = np.meshgrid(x1, x2)
    Z = 0.5 * (Q[0, 0] * X1**2 + Q[1, 1] * X2**2)
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X1, X2, Z, cmap='viridis', alpha=0.85)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('$f(x)$')
    ax.set_title(r'3D plot of $f(\vec x) = \frac{1}{2}\vec x^T Q \vec x$')
    plt.tight_layout()
    plt.show()


# Gradient descent method
def gr_descent_method(x0_1, x0_2, Q, alpha=0.1, iterations=10):
    x0 = np.array([x0_1, x0_2])
    xs = [x0]
    for _ in range(iterations):
        x0 = x0 - alpha * df(x0[0], x0[1], Q)
        xs.append(x0)
    return np.array(xs)



# Plot f(x) vs iteration
```



Contour plot of $f(\vec{x}) = \frac{1}{2}\vec{x}^T Q \vec{x}$
x_1
x_2
−6
−4
−2
0
2
4
6
0
2400
4800
7200
9600
12000
14400
16800



3D plot of $f(\vec{x}) = \frac{1}{2}\vec{x}^T Q \vec{x}$
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
$f(x)$
−6
−4
−2
0
2
4
6
x_1
6
4
2
0
−2
−4
−6
x_2



Note that if we had $\gamma = 100$ then the default $\alpha = 0.1$ is larger than $2/100$ and the itereates grow exponentially since the update in the x_2 direction multiplies by $(1 - \alpha \cdot 100) = -9$ each step. With $\gamma < 1$ then $\alpha = 0.1$ satisfies $0 < \alpha < 2/\lambda_{\max} = 2$. To be on the safe side for any γ (and if we know the function is a quadratic function) we should adjust the learning rate according to the curvature of the function $L = \max\{1, \lambda_{\max}\}$. Since, for convergence we must have $0 < \alpha < 2/\lambda_{max}(Q)$, we can set $\alpha = 1/\max\{1, \gamma\} = \min\{1, \gamma\}$. If $\gamma < 1$, like in our case, then $\alpha = \gamma$.
Of course we can also always use line search to ensure that we operate within the stability range and achieve convergence.
The convergence plots show that smaller values of γ lead to slower convergence. We can relate this to the curvature, since $c \leq \lambda_{\min}(Q)$ and $\lambda_{\min}(Q) = \gamma$.
This is also reflected in the *optimization condition number* which is usually defined as $\kappa = L/c$ where L is the Lipschitz gradient constant (aka smoothness constant) and c is the strong convexity constant (aka curvature). For a quadratic function, L is the maximum eigenvalue of Q and c is the minimum eigenvalue of Q. Hence, the condition number can be expressed as: $\kappa = \lambda_{\max}/\lambda_{\min}$. For quadratic problems, the *optimization condition number* is the same as the *matrix condition number* $\kappa_2\ (Q) = \|Q\|_2/\|Q^{-1}\|_2$. For our specific case with $\gamma < 1$, $\lambda_{\max} = 1$ and $\lambda_{\min} = \gamma$. Hence:

- Small c implies flat curvature and hence an ill-conditioned problem.
- Large c implies well-conditioned problem.

This is fundamental in machine learning (ridge regression adds strong convexity).

$$f(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|^2$$

The regularization term makes the objective λ-strongly convex. This is why ridge regression behaves much better numerically than ordinary least squares when the design matrix is ill-conditioned.

Exercise 9

Show that strong convexity is equivalent to strong monotonicity of the gradient:

$$(\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}))^\top (\boldsymbol{x} - \boldsymbol{y}) \geq c\|\boldsymbol{x} - \boldsymbol{y}\|^2 \quad \forall \boldsymbol{x}, \boldsymbol{y}.$$

We prove equivalence.
Using the fundamental theorem of calculus:

$$\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) = \int_0^1 \nabla^2 f(\boldsymbol{y} + t(\boldsymbol{x} - \boldsymbol{y}))(\boldsymbol{x} - \boldsymbol{y}) dt.$$

Multiplying by $(\boldsymbol{x} - \boldsymbol{y})$:

$$(\boldsymbol{x} - \boldsymbol{y})^\top (\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})) = \int_0^1 (\boldsymbol{x} - \boldsymbol{y})^\top \nabla^2 f(\boldsymbol{y} + t(\boldsymbol{x} - \boldsymbol{y}))(\boldsymbol{x} - \boldsymbol{y}) dt.$$

If $\nabla^2 f(\cdot) \succeq cI$, then: then:

$$(\boldsymbol{x} - \boldsymbol{y})^\top (\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})) \geq c\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

This is strong monotonicity. The reverse direction follows similarly.