EE 5239 Optimization Homework 2 Cover Sheet

Instructor name: Mingyi Hong, Jiaxiang Li

Student name: Alex Ojemann

- Date assigned: Monday 9/23/2024
- Date due: Friday 10/04/2024 11:59pm
- This cover sheet must be signed and submitted along with the homework answers on additional sheets.
- By submitting this homework with my name affixed above,
  - I understand that late homework will not be accepted,
  - I acknowledge that I am aware of the University's policy concerning academic misconduct (appended below),
  - I attest that the work I am submitting for this homework assignment is solely my own, and
  - I understand that suspiciously similar homework submitted by multiple individuals will be reported to the Dean of Students Office for investigation.
- Academic Misconduct in any form is in violation of the University's Student Disciplinary Regulations and will not be tolerated. This includes, but is not limited to: copying or sharing answers on tests or assignments, plagiarism, having someone else do your academic work or working with someone on homework when not permitted to do so by the instructor. Depending on the act, a student could receive an F grade on the test/assignment, F grade for the course, and could be suspended or expelled from the University.

1 Reading

- Reading: Textbook Section 1.1 - 1.2
- Appendix A.

2 Problems

1. Exercise 1.2.1, 1.2.2, 1.2.3, 1.2.7 in the textbook (version 2, version 3)

   1.2.1(a)

   Given $f(x, y) = 3x^2 + y^4$, the gradient is:

   $$\nabla f(x, y) = (6x, 4y^3)$$

   $$\nabla f(1, -2) = (6, -32)$$

   The update step is:

   $$(x_{k+1}, y_{k+1}) = (x_k, y_k) - \beta_k \nabla f(x_k, y_k)$$

   With $\beta = 0.5$:

   $$(x_1, y_1) = (1, -2) - 0.5 \cdot (6, -32) = (-2, 14)$$

   The new point is $(-2, 14)$.

   1.2.1(b)

   Using $\beta = 0.1$:

   $$(x_1, y_1) = (1, -2) - 0.1 \cdot (6, -32) = (0.4, 1.2)$$

   The new point is $(0.4, 1.2)$.

   Tradeoffs: A smaller β results in smaller steps, which may need more iterations to reach the minimum, but can prevent overshooting. Larger β may reduce iterations but risks overshooting.

   1.2.1(c)

   The Hessian is:

   $$H(f) = \begin{pmatrix} 6 & 0 \\ 0 & 12y^2 \end{pmatrix}$$

   $$H(f)(1, -2) = \begin{pmatrix} 6 & 0 \\ 0 & 48 \end{pmatrix}$$

   Newton's update:

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

   $$H(f)^{-1} = \begin{pmatrix} \frac{1}{6} & 0 \\ 0 & \frac{1}{48} \end{pmatrix}$$

   $$(x_1, y_1) = (1, -2) - \begin{pmatrix} 1 & 0 \\ 0 & -\frac{32}{48} \end{pmatrix} = (0, -\frac{4}{3})$$

   The new point is $\left(0, -\frac{4}{3}\right)$.

   Comparison: Newton's method converges faster but involves higher computational costs due to Hessian computation and inversion.

1.2.2

For $f(x) = \|x\|^{2+\beta}$, the gradient is:

$$\nabla f(x) = (2+\beta)\|x\|^{\beta} x$$

With constant step size s, the update is:

$$x_{k+1} = x_k - s(2+\beta)\|x_k\|^{\beta} x_k$$

For convergence, s must satisfy:

$$0 < s < \frac{2}{L}$$

where L is the Lipschitz constant.

1.2.3

For $f(x) = \|x\|^{3/2}$, the gradient:

$$\nabla f(x) = \frac{3}{2}\|x\|^{1/2}\frac{x}{\|x\|}$$

does not satisfy the Lipschitz condition because the gradient blows up as $x \to 0$. Thus, steepest descent with constant step size does not guarantee convergence to $x^* = 0$.

1.2.7

The engineer's method is similar to a coordinate ascent approach. Let the current be represented by $I(x_1, x_2)$, where $x_1 \in [0, R_1]$ and $x_2 \in [0, R_2]$.

At each step: - The engineer maximizes I with respect to x_1, holding x_2 fixed. - Then she maximizes I with respect to x_2, holding x_1 fixed.

This procedure ensures that the current either increases or remains the same with each iteration. The method stops when a local maximum is reached, i.e., when no further progress can be made by adjusting x_1 or x_2.

This approach is theoretically sound because it is analogous to a steepest ascent algorithm applied in coordinate directions. The stopping criterion is when the gradient with respect to both x_1 and x_2 is zero, corresponding to a stationary point of $I(x_1, x_2)$.

Since the extreme values $0, R_1, R_2$ are not reached, it implies that the maximum current occurs at an interior point of the domain, rather than at the boundaries. This suggests that the function $I(x_1, x_2)$ has a local maximum in the interior of the feasible region.

The method is reliable if $I(x_1, x_2)$ is concave, ensuring that the local maximum found is also the global maximum.

2. Show that a second-order continuously differentiable function $f(x) : \mathbb{R} \to \mathbb{R}$ is convex *if and only if* its second-order derivative is non-negative.

   We show that $f(x)$ is convex if and only if $f''(x) \geq 0$ for all $x \in \mathbb{R}$.

   Assume $f''(x) \geq 0$ for all $x \in \mathbb{R}$.

$$f'(x) = f'(a) + \int_a^x f''(t)dt$$

If: Since $f''(t) \geq 0$, $f'(x)$ is non-decreasing, so $f'(x_1) \leq f'(x_2)$ for $x_1 < x_2$. Hence, $f(x)$ is convex.

Only if: Assume $f(x)$ is convex. For any $x_1, x_2 \in \mathbb{R}$ and $\lambda \in [0, 1]$,

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

$f''(x) \geq 0$ for all $x \in \mathbb{R}$.

3. Suppose that $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is convex. Please show the following properties

   (a) $f(\mathbf{x}) - f(\mathbf{y}) \geq \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$, $\forall\ \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

   (b) If $g(y) : \mathbb{R} \to \mathbb{R}$ is a monotonically non-decreasing **convex** function, then $g(f(\mathbf{x}))$ is also a convex function.

   (c) If $g(y) : \mathbb{R} \to \mathbb{R}$ is a decreasing **convex** function, then provide an example showing that $g(f(\mathbf{x}))$ may not be a convex function.

   Proof:

   (a) Since $f(\mathbf{x})$ is convex, by definition, for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we have:

   $$f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$$

   Thus,

   $$f(\mathbf{x}) - f(\mathbf{y}) \geq \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$$

   (b) Let $g(y)$ be a monotonically non-decreasing convex function. The composition $g(f(\mathbf{x}))$ is convex if for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ and $\lambda \in [0, 1]$:

   $$g(f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)) \leq \lambda g(f(\mathbf{x}_1)) + (1-\lambda) g(f(\mathbf{x}_2))$$

   Since $f(\mathbf{x})$ is convex, we know:

   $$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

   Applying the non-decreasing property of g:

   $$g(f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)) \leq g(\lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2))$$

   Since $g(y)$ is convex:

   $$g(\lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)) \leq \lambda g(f(\mathbf{x}_1)) + (1-\lambda) g(f(\mathbf{x}_2))$$

   Thus, $g(f(\mathbf{x}))$ is convex.

   (c) Let $g(y) = -\log(y)$ (convex and decreasing). Consider $f(\mathbf{x}) = \|\mathbf{x}\|^2$, a convex function. Then:

   $$g(f(\mathbf{x})) = -\log(\|\mathbf{x}\|^2)$$

   This function is not convex because its second derivative does not satisfy the convexity condition.

4. Let $\mathbf{A} \neq 0$ be a rank one square matrix, i.e., $\mathbf{A} = \mathbf{x}_0\mathbf{y}_0^T \neq 0$, for some $\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{R}^n$. Consider the following optimization problem

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \frac{1}{2}\|\mathbf{A} - \mathbf{x}\mathbf{y}^T\|_F^2 \\ \text{s.t.} \quad & \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \end{aligned} \tag{1}$$

Please answer the following questions.

- Show that it is **not** a convex optimization problem.
- Find the set of stationary points.
- **(bonus)** Show that every local optimum of the problem is a global optimum

Consider the optimization problem:

$$\min_{\mathbf{x},\mathbf{y}} \quad \frac{1}{2}\|\mathbf{A} - \mathbf{x}\mathbf{y}^T\|_F^2$$

where $\mathbf{A} = \mathbf{x}_0\mathbf{y}_0^T$, a rank-one matrix.

- **Non-convexity:** The objective function is:

$$f(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x}_0\mathbf{y}_0^T - \mathbf{x}\mathbf{y}^T\|_F^2$$

$$f(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\left(\|\mathbf{x}_0\mathbf{y}_0^T\|_F^2 - 2\langle \mathbf{x}_0\mathbf{y}_0^T, \mathbf{x}\mathbf{y}^T\rangle + \|\mathbf{x}\mathbf{y}^T\|_F^2\right)$$

  Since $\mathbf{x}\mathbf{y}^T$ is bilinear in $\mathbf{x}$ and $\mathbf{y}$, the function is not convex in $(\mathbf{x}, \mathbf{y})$.
- **Stationary points:** The gradient of $f(\mathbf{x}, \mathbf{y})$ with respect to $\mathbf{x}$ and $\mathbf{y}$ are:

$$\nabla_{\mathbf{x}} f = \mathbf{x}\mathbf{y}^T\mathbf{y} - \mathbf{x}_0\mathbf{y}_0^T\mathbf{y}$$

$$\nabla_{\mathbf{y}} f = \mathbf{y}\mathbf{x}^T\mathbf{x} - \mathbf{y}_0\mathbf{x}_0^T\mathbf{x}$$

  Setting these equal to zero, we get the stationary points as $(\mathbf{x}, \mathbf{y}) = (\alpha\mathbf{x}_0, \beta\mathbf{y}_0)$ for scalars α, β.
- **Bonus:** To show every local optimum is a global optimum, consider the objective function:

$$f(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x}_0\mathbf{y}_0^T - \mathbf{x}\mathbf{y}^T\|_F^2$$

  Since the error function is quadratic and the rank-one nature of $\mathbf{A}$ means the solution must match the structure of $\mathbf{A}$, any stationary point where the gradient is zero corresponds to the global minimum, implying that every local minimum is a global minimum.

3 Programming Assignment

In this assignment, you are asked to experiment with

1. Steepest descent method with Armijo step size rule
2. Diagonally scaled gradient method (using the Hessian diagonals), constant step size rule

3. Conjugate gradient method, exact minimization step size rule

to solve the following convex quadratic minimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Q x + b^T x \\ \text{s.t.} \quad & x \in \mathbb{R}^n \end{aligned} \tag{2}$$

Always use the starting point $x = (1; 1; \cdots, 1)^T$. Please see below for a Matlab script that you should use to generate your data sets.

What you need to turn in:

1. Well documented MATLAB codes (or any other codes of a scientific computing language).
2. Plots of CPU time and progression of objective values (relative to the minimum) for various choices of step size rules, algorithms, problem sizes ($n = 100, 500$), and condition numbers ($c = 10, 5000$).
3. Brief discussions of your findings relative to the convergence theory.

Code (Python):

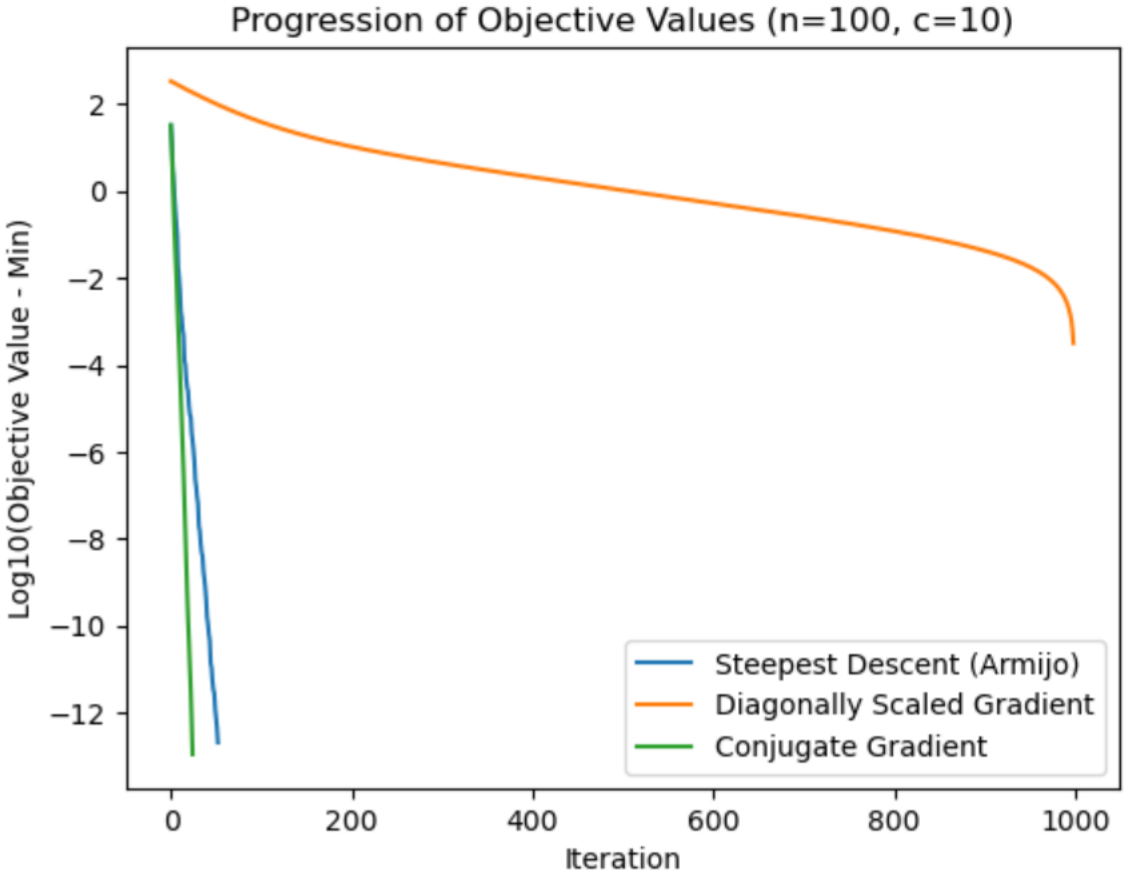
```
import numpy as np
import matplotlib.pyplot as plt
import time

# Generate problem data
def generate_problem(n, condition_number):
    Q = np.random.randn(n, n)
    Q = np.dot(Q.T, Q)  # Make Q positive definite
    u, s, vh = np.linalg.svd(Q)
    s = np.linspace(1, condition_number, n)
    Q = np.dot(u * s, vh)
    b = np.random.randn(n)
    return Q, b

# Objective function and gradient
def objective(x, Q, b):
    return 0.5 * np.dot(x.T, np.dot(Q, x)) + np.dot(b.T, x)

def gradient(x, Q, b):
    return np.dot(Q, x) + b

# Hessian for Diagonally Scaled Gradient Method
def hessian_diag(Q):
    return np.diag(Q)

# Steepest descent with Armijo step size rule
def steepest_descent_armijo(Q, b, x0, max_iter=1000, tol=1e-6):
    x = x0
```

```
    obj_values = []
    for _ in range(max_iter):
        grad = gradient(x, Q, b)
        if np.linalg.norm(grad) < tol:
            break
        step_size = armijo_step_size(Q, b, x, grad)
        x = x - step_size * grad
        obj_values.append(objective(x, Q, b))
    return x, obj_values

def armijo_step_size(Q, b, x, grad, sigma=0.1, beta=0.5, max_step=1):
    t = max_step
    while objective(x - t * grad, Q, b) > objective(x, Q, b) - sigma * t * np.dot(grad.T, grad)
        t *= beta
    return t

# Diagonally scaled gradient method with constant step size
def diagonally_scaled_gradient(Q, b, x0, step_size=0.01, max_iter=1000, tol=1e-6):
    x = x0
    diag_Q = hessian_diag(Q)
    obj_values = []
    for _ in range(max_iter):
        grad = gradient(x, Q, b)
        if np.linalg.norm(grad) < tol:
            break
        x = x - step_size * grad / diag_Q
        obj_values.append(objective(x, Q, b))
    return x, obj_values

# Conjugate gradient method
def conjugate_gradient(Q, b, x0, max_iter=1000, tol=1e-6):
    x = x0
    r = gradient(x, Q, b)
    p = -r
    obj_values = []
    for _ in range(max_iter):
        if np.linalg.norm(r) < tol:
            break
        alpha = -np.dot(r.T, p) / np.dot(p.T, np.dot(Q, p))
        x = x + alpha * p
        r_new = gradient(x, Q, b)
        beta = np.dot(r_new.T, r_new) / np.dot(r.T, r)
        p = -r_new + beta * p
        r = r_new
        obj_values.append(objective(x, Q, b))
    return x, obj_values

# Plotting function
```

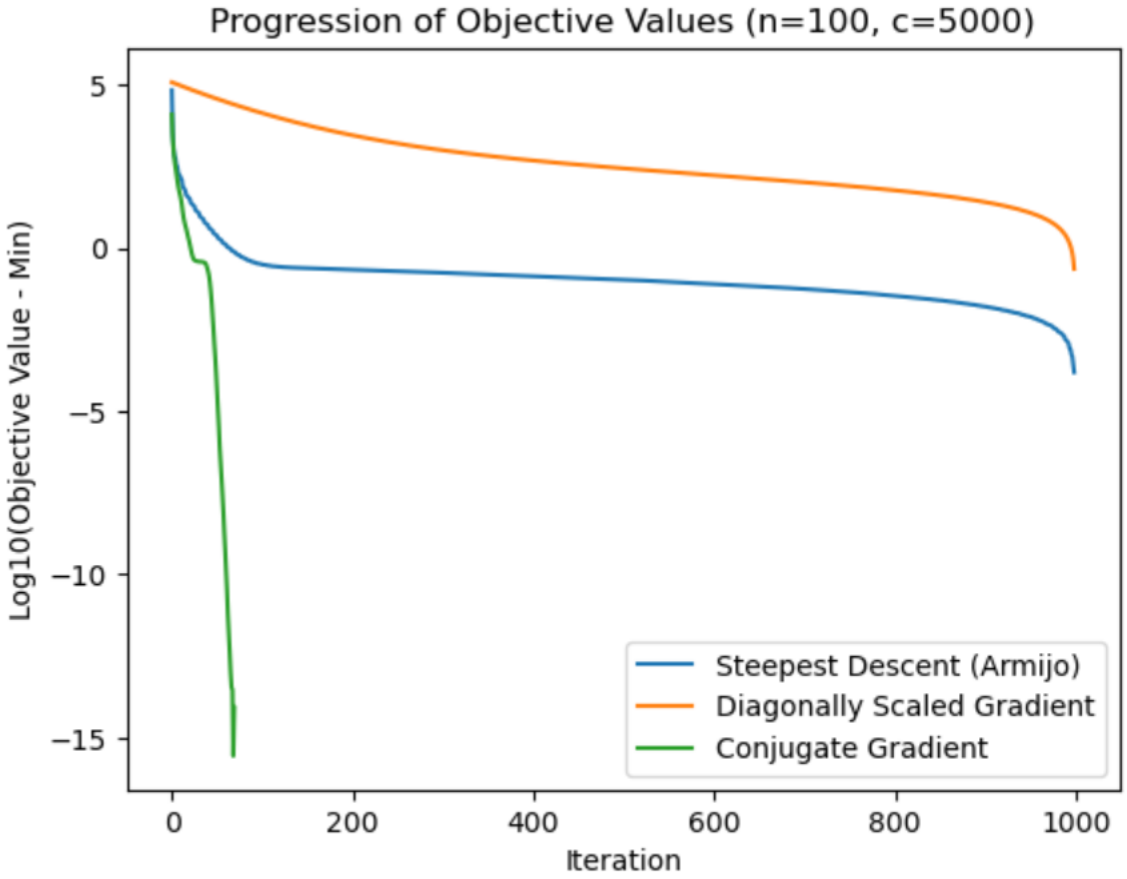
```
def plot_progression(obj_values, label):
    plt.plot(np.log10(obj_values - min(obj_values)), label=label)
    plt.xlabel('Iteration')
    plt.ylabel('Log10(Objective Value - Min)')
    plt.legend()

# Main function to run experiments
def run_experiments(n, condition_number):
    Q, b = generate_problem(n, condition_number)
    x0 = np.ones(n)

    # Steepest Descent with Armijo rule
    start_time = time.time()
    _, obj_values_sd = steepest_descent_armijo(Q, b, x0)
    time_sd = time.time() - start_time
    print(f"Steepest Descent Time: {time_sd:.4f}s")

    # Diagonally Scaled Gradient with constant step size
    start_time = time.time()
    _, obj_values_dsg = diagonally_scaled_gradient(Q, b, x0)
    time_dsg = time.time() - start_time
    print(f"Diagonally Scaled Gradient Time: {time_dsg:.4f}s")

    # Conjugate Gradient
    start_time = time.time()
    _, obj_values_cg = conjugate_gradient(Q, b, x0)
    time_cg = time.time() - start_time
    print(f"Conjugate Gradient Time: {time_cg:.4f}s")

    # Plot progression of objective values
    plt.figure()
    plot_progression(obj_values_sd, 'Steepest Descent (Armijo)')
    plot_progression(obj_values_dsg, 'Diagonally Scaled Gradient')
    plot_progression(obj_values_cg, 'Conjugate Gradient')
    plt.title(f'Progression of Objective Values (n={n}, c={condition_number})')
    plt.show()

# Run experiments for different sizes and condition numbers
run_experiments(n=100, condition_number=10)
run_experiments(n=100, condition_number=5000)
run_experiments(n=500, condition_number=10)
run_experiments(n=500, condition_number=5000)
```

Plots

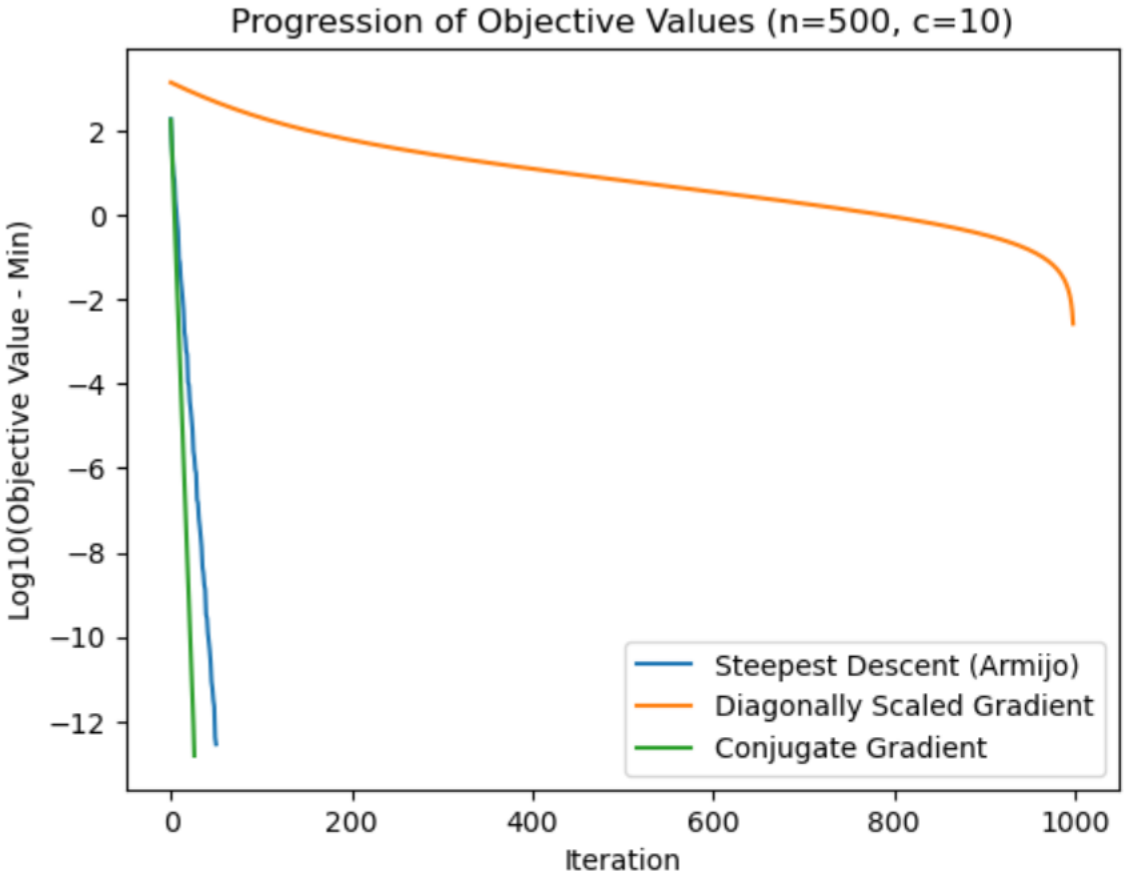
```
Steepest Descent Time: 0.0040s
Diagonally Scaled Gradient Time: 0.0140s
Conjugate Gradient Time: 0.0010s
C:\Users\alexo\AppData\Local\Temp\ipykernel_22872\3029943399.py:78: RuntimeWarning: 
ered in log10
  plt.plot(np.log10(obj_values - min(obj_values)), label=label)
```



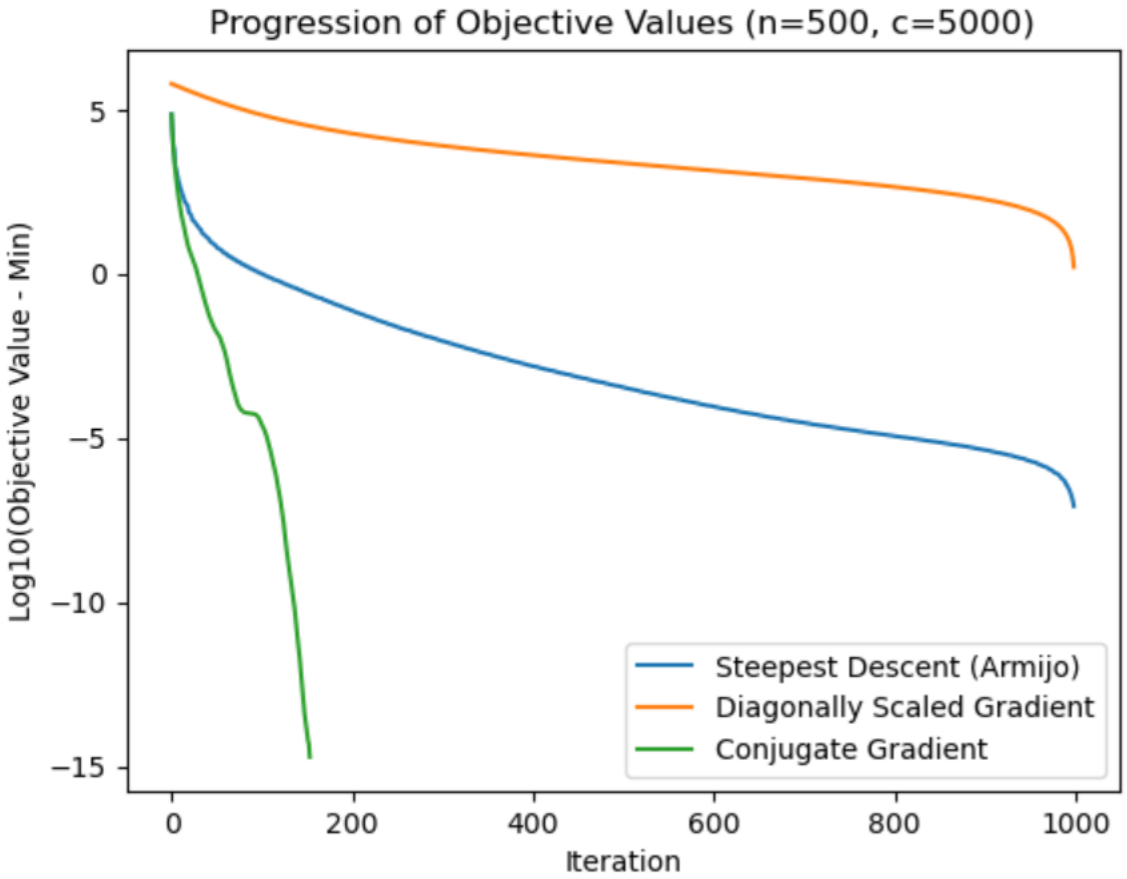
```
Steepest Descent Time: 0.1830s
Diagonally Scaled Gradient Time: 0.0116s
Conjugate Gradient Time: 0.0010s
```



```
Steepest Descent Time: 0.1309s
Diagonally Scaled Gradient Time: 0.3390s
Conjugate Gradient Time: 0.0090s
```



```
Steepest Descent Time: 2.2800s
Diagonally Scaled Gradient Time: 0.1737s
Conjugate Gradient Time: 0.0410s
```



Commentary

We see that the conjugate gradient method converges the fastest in all cases, which is expected because it can take advantage of the nature of the problem and perform exact minimization along conjugate directions.

We see the steepest descent with Amijo rule step size have very rapid convergence when the condition number is low but very slow convergence when the condition number is 5000. This is to be expected because a high condition number means the matrix Q is harder to deal with.