

### 0.0.1 Importing necessary libraries

```
[1]: import numpy as np
from sklearn.datasets import make_spd_matrix
import matplotlib.pyplot as plt
```

## 0.1 Problem 1

### 0.1.1 Problem 1 (a)

$$f(x) = x^T A x + b$$

$$\Rightarrow \nabla f(x) = 2Ax$$

$$\text{where } A = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 1 \end{bmatrix}, b = [1] \text{ By setting the gradient to zero } \Rightarrow 2Ax = 0$$

$$\Rightarrow x = 0$$

```
[2]: A_a = np.matrix([[2, -1, -1], [-1, 2, 0], [-1, 0, 1]])
b_a = np.matrix([[1]])

def f_a(x):
    return x.T @ A_a @ x + b_a

def gradient_f_a(x):
    return 2* A_a @ x

def gradient_descent(x0, learning_rate, num_iterations):
    x = x0
    for i in range(num_iterations):
        x = x - learning_rate * gradient_f_a(x)
    return x

N = 1000
t = 0.02

x0 = np.random.rand(3,1)
result_gradient_descent = gradient_descent(x0, t, N)
print("Minimizer using gradient descent:", result_gradient_descent)
print("Value of f_a at minimizer:", f_a(result_gradient_descent)[0,0])
result_gradient_zero = (1/2) * A_a.I @ np.matrix([[0],[0],[0]])
print("Minimizer using gradient zero:", result_gradient_zero)
print("Value of f_a at minimizer using gradient zero:", f_a(result_gradient_zero)[0,0])
```

Minimizer using gradient descent:  $\begin{bmatrix} 1.54960309e-04 \\ 8.59964837e-05 \\ 1.93232345e-04 \end{bmatrix}$

$\begin{bmatrix} 8.59964837e-05 \\ 1.93232345e-04 \end{bmatrix}$

$\begin{bmatrix} 1.93232345e-04 \end{bmatrix}$

Value of f\_a at minimizer: 1.000000013616153

Minimizer using gradient zero:  $\begin{bmatrix} 0. \end{bmatrix}$

```
[0.]
[0.]]
```

Value of f\_a at minimizer using gradient zero: 1.0

### 0.1.2 Problem 1 (b)

$$f(x) = \|Ax - b\|^2$$

$$\Rightarrow \nabla f(x) = 2A^T(Ax - b)$$

$$\text{where } A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \text{ By setting the gradient to zero } \Rightarrow 2A^T(Ax - b) = 0$$

$$\Rightarrow A^T(Ax - b) = 0$$

$$\Rightarrow A^T Ax = A^T b$$

$$\Rightarrow x = (A^T A)^{-1} A^T b$$

```
[3]: A_b = np.matrix([[1,2],[2,4],[3,1]])
     b_b = np.matrix([[1],[3],[1]])

     def f_b(x):
         return x.T @ A_b.T @ A_b @ x - b_b.T @ A_b @ x - x.T @ A_b.T @ b_b + b_b.T @
         ↪ b_b

     def gradient_f_b(x):
         return 2* A_b.T @ (A_b @ x - b_b)

     def gradient_descent(x0, learning_rate, num_iterations):
         x = x0
         for i in range(num_iterations):
             x = x - learning_rate * gradient_f_b(x)
         return x

     N = 1000
     t = 0.02
     x0 = np.random.rand(2,1)
     result_gradient_descent = gradient_descent(x0, t, N)
     print("Minimizer using gradient descent:", result_gradient_descent)
     print("Value of f_b at minimizer:", f_b(result_gradient_descent)[0,0])
     result_gradient_zero = (A_b.T @ A_b).I @ A_b.T @ b_b
     print("Minimizer using gradient zero:", result_gradient_zero)
     print("Value of f_b at minimizer using gradient zero:",
     ↪ f_b(result_gradient_zero)[0,0])
```

Minimizer using gradient descent: [[0.12]

[0.64]]

Value of f\_b at minimizer: 0.200000000000000107

Minimizer using gradient zero: [[0.12]

[0.64]]

Value of f\_b at minimizer using gradient zero: 0.200000000000000107

### 0.1.3 Problem 1 (c)

$$f(x) = \|Ax - b\|^2$$

$$\Rightarrow \nabla f(x) = 2A^T(Ax - b)$$

$$\text{where } A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 3 & 1 & 9 \\ 4 & 1 & 0 \\ 2 & 1 & 4 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 0 \\ 9 \end{bmatrix} \text{ By setting the gradient to zero } \Rightarrow 2A^T(Ax - b) = 0$$

$$\Rightarrow A^T(Ax - b) = 0$$

$$\Rightarrow A^T Ax = A^T b$$

$$\Rightarrow x = (A^T A)^{-1} A^T b$$

```
[4]: A_c = np.matrix([[1,2,1],[2,4,2],[3,1,9],[4,1,0],[2,1,4]])
b_c = np.matrix([[1],[3],[1],[0],[9]])

def f_c(x):
    return x.T @ A_c.T @ A_c @ x - b_c.T @ A_c @ x - x.T @ A_c.T @ b_c + b_c.T @
    ↪ b_c

def gradient_f_c(x):
    return 2* A_c.T @ (A_c @ x - b_c)

def gradient_descent(x0, learning_rate, num_iterations):
    x = x0
    for i in range(num_iterations):
        x = x - learning_rate * gradient_f_c(x)
    return x

N = 1000
t = 0.002 # learning rate is made smaller than previous problems due to out of
    ↪ bound errors
x0 = np.random.rand(3,1)
result_gradient_descent = gradient_descent(x0, t, N)
print("Minimizer using gradient descent:", result_gradient_descent)
print("Value of f_c at minimizer:", f_c(result_gradient_descent)[0,0])
result_gradient_zero = (A_c.T @ A_c).I @ A_c.T @ b_c
print("Minimizer using gradient zero:", result_gradient_zero)
print("Value of f_c at minimizer using gradient zero:",
    ↪ f_c(result_gradient_zero)[0,0])
```

```
Minimizer using gradient descent: [[0.05744939]
[0.65686105]
[0.33915902]]
Value of f_c at minimizer: 56.990482782488314
Minimizer using gradient zero: [[0.05744939]
[0.65686105]
[0.33915902]]
```

Value of  $f_c$  at minimizer using gradient zero: 56.99048278248832

## 0.2 Problem 2

Generating  $A$  randomly using `sklearn.datasets.make_spd_matrix`, and generate  $b$  and  $c$  using `np.random.rand` function of numpy.

```
[5]: A = np.matrix(make_spd_matrix(10))
     b = np.matrix(np.random.randn(10,1))
     c = np.matrix(np.random.randn(1,1))
```

$$f(x) = x^T A x - 2b^T x + c$$

$$\Rightarrow \nabla f(x) = 2Ax - 2b$$

$$\Rightarrow \nabla^2 f(x) = 2A$$

$$\nabla^2 f(x) \geq 0 \Rightarrow f \text{ is convex}$$

$$2 * A \geq 0 \Rightarrow A \geq 0 \Rightarrow A \text{ is positive semi definite} \Rightarrow f \text{ is convex}$$

So we need to check if  $A$  is positive semi definite or not to know if  $f$  is convex or not.

```
[6]: def f(x):
     return x.T @ A @ x - 2 * b.T @ x + c

     def gradient_f(x):
         return 2 * A @ x - 2 * b

     def grad_grad_f(x):
         return 2 * A

     def is_pos_sem_def(x):
         return np.all(np.linalg.eigvals(x) >= 0)
```

### 0.2.1 Problem 2 (a)

Analytical solution

```
[7]: if is_pos_sem_def(A):
     print("A is semi positive definite, hence f is convex")
```

$A$  is semi positive definite, hence  $f$  is convex

By setting the gradient of  $f$  to zero, we can find the analytical solution  $\Rightarrow \nabla f(x) = 0$

$$\Rightarrow 2Ax - 2b = 0$$

$$\Rightarrow x = A^{-1}b$$

```
[8]: result_gradient_zero = np.matrix(A.I @ b)
```

```
[9]: analytical_solution = f(result_gradient_zero)[0,0]
```

```
[10]: print("Minimizer using gradient zero:", result_gradient_zero)
      print("Analytical solution by setting gradient to zero:", analytical_solution)
```

Minimizer using gradient zero:  $\begin{bmatrix} -9.15256852 \\ 2.78083746 \\ -1.18661204 \\ 1.32064519 \\ -7.09335175 \\ 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} 2.78083746 \\ -1.18661204 \\ 1.32064519 \\ -7.09335175 \\ 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} -1.18661204 \\ 1.32064519 \\ -7.09335175 \\ 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} 1.32064519 \\ -7.09335175 \\ 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} -7.09335175 \\ 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} 13.35628316 \\ -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} -3.38601289 \\ 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} 2.42614784 \\ -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} -0.79290161 \\ 9.7378983 \end{bmatrix}$

$\begin{bmatrix} 9.7378983 \end{bmatrix}$

Analytical solution by setting gradient to zero: -28.24615806045249

## 0.2.2 Problem 2 (b)

**Gradient descent** Initial point  $x_0 = [1/10, 1/10, 1/10, 1/10, 1/10, 1/10, 1/10, 1/10, 1/10, 1/10]^T$  where  $x_0 \in \mathbb{R}^{10}$

```
[11]: x0 = np.matrix([[1/10],[1/10],[1/10],[1/10],[1/10],[1/10],[1/10],[1/10],[1/10],[1/10]])
```

Step size  $l_{GD} = \frac{1}{2\|A\| + \|b\|_2}$  where  $\|A\|$  is the spectral norm of  $A$  and  $\|b\|_2$  is the 2-norm of  $b$

```
[12]: a_norm = np.linalg.norm(A, ord=2)
b_norm = np.linalg.norm(b)
step_size = 1 / ((2*a_norm) + b_norm)
```

```
[13]: def gradient_descent(x0, step_size, num_iterations):
    """
    Gradient descent algorithm
    :param x0: np.matrix - Initial point
    :param step_size: float - step size
    :param num_iterations: int - number of iterations
    :return: value of x after num_iterations, list of objective values
    """
    x = x0
    f_list = []
    for i in range(num_iterations):
        x = x - step_size * gradient_f(x)
        f_list.append(f(x).tolist()[0][0])
    return x, f_list
```

```
[14]: result_gradient_descent, f_list = gradient_descent(x0, step_size, 1000)
```

```
[15]: optimal_solution = f(result_gradient_descent)[0,0]
```

```
[16]: print("Minimizer using gradient descent:", result_gradient_descent)
print("Optimal Solution by using gradient descent:", optimal_solution)
```

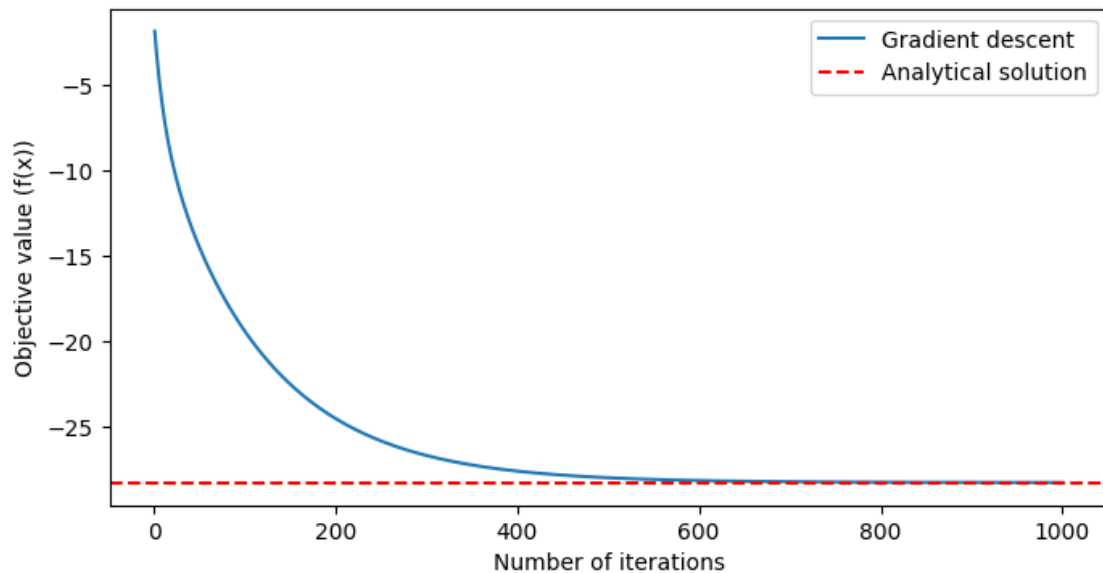
Minimizer using gradient descent:  $\begin{bmatrix} -9.03581218 \\ 2.72650626 \\ -1.14045911 \\ 1.29836243 \\ -6.98249168 \\ 13.19409598 \\ -3.3210739 \\ 2.37920814 \\ -0.77443266 \\ 9.6221097 \end{bmatrix}$

$\begin{bmatrix} 2.72650626 \\ -1.14045911 \\ 1.29836243 \\ -6.98249168 \\ 13.19409598 \\ -3.3210739 \\ 2.37920814 \\ -0.77443266 \\ 9.6221097 \end{bmatrix}$

Optimal Solution by using gradient descent: -28.242278949955974

In the following plot, the blue line shows the objective value after each iteration and the red dashed line shows the analytical solution.

```
[17]: plt.figure(figsize=(8,4))
plt.plot(range(1,1001), f_list)
plt.xlabel("Number of iterations")
plt.ylabel("Objective value (f(x))")
plt.axhline(y=analytical_solution, color='r', linestyle='--')
plt.legend(["Gradient descent", "Analytical solution"])
plt.show()
```



You can see that the objective value decreases after each iteration and converges to the analytical solution.