**The Chinese University of Hong Kong, Shenzhen**
**School of Data Science**

**MAT3007 · Homework 8**
Due: 23:59, Dec 6, 2024

**Instructions:**

- Homework problems must be carefully and clearly answered to receive full credit. Complete sentences that establish a clear logical progression are highly recommended.
- You must submit your assignment in Blackboard.
- The homework must be written in English.
- Late submission will not be graded.
- Each student **must not copy** homework solutions from another student or from any other source.
- For those questions that ask you to write Matlab/Python codes to solve the problem. Please attach the code to the homework. However, you do not need to attach the outputs in the command window of Matlab/Python.

---

**Problem 1: Descent Directions (20 pts).**
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and consider $\boldsymbol{x} \in \mathbb{R}^n$ with $\nabla f(\boldsymbol{x}) \neq 0$. Verify the following statements:

a) Set $\boldsymbol{d} = -(\nabla f(\boldsymbol{x})_j) \cdot \boldsymbol{e}_j = -\frac{\partial f}{\partial x_j} \cdot \boldsymbol{e}_j$, where $\boldsymbol{e}_j \in \mathbb{R}^n$ is the $j$-th unit vector and $j \in \{1, \ldots, n\}$ is an index satisfying

$$\left|\frac{\partial f}{\partial x_j}\right| = \max_{1 \le i \le n} \left|\frac{\partial f}{\partial x_i}\right|.$$

Then, $\boldsymbol{d}$ is a descent direction of $f$ at $\boldsymbol{x}$.

b) The direction $\boldsymbol{d} = -\nabla f(\boldsymbol{x})/\|\nabla f(\boldsymbol{x})\|$ is a descent direction of $f$ at $\boldsymbol{x}$.

c) Let $f$ be twice continuously differentiable and define $d_i = -(\nabla f(\boldsymbol{x})_i)/\max\{\nabla^2 f(\boldsymbol{x})_{ii}, \varepsilon\}$ for all $i \in \{1, \ldots, n\}$ and for some $\varepsilon > 0$. Then, $\boldsymbol{d}$ is well-defined (we do not divide by zero) and it is a descent direction of $f$ at $\boldsymbol{x}$.

**Problem 2: Optimization Algorithm for Regularized Least Squares (30 pts).**
Consider the following unconstrained optimization problem:

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \;\; f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 + \frac{\lambda}{2}\|\boldsymbol{x}\|^2,$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m, \lambda \in \mathbb{R}$ are known.

a) Compute the gradient and Hessian of $f$.

b) The property of $f$ depends on the parameter $\lambda$. Assume the matrix $\boldsymbol{A}^\top \boldsymbol{A}$ is *not* positive definite. Discuss the corresponding conditions on $\lambda$ such that the optimization problem will be strongly convex, convex, and non-convex.

**Note:** An optimization problem is said to be strongly convex if its Hessian is positive definite.

c) Suppose that we want to apply *gradient descent method* to solve this unconstrained optimization problem, where the matrix $\boldsymbol{A}$, vector $\boldsymbol{b}$, and parameter $\lambda$ are defined as:

$$\boldsymbol{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda = 3.$$

We choose the initial point $x^0$ as:

$$\boldsymbol{x}^0 = \begin{bmatrix} \frac{1}{2} \\ \\ \frac{1}{2} \end{bmatrix}.$$

Now consider the following two stepsize rules to determine $\alpha_0$ for updating $\boldsymbol{x}^1$.

1) Suppose we apply Armijo/backtracking line search to select the stepsize $\alpha_0$ with Armijo parameters set as follows:

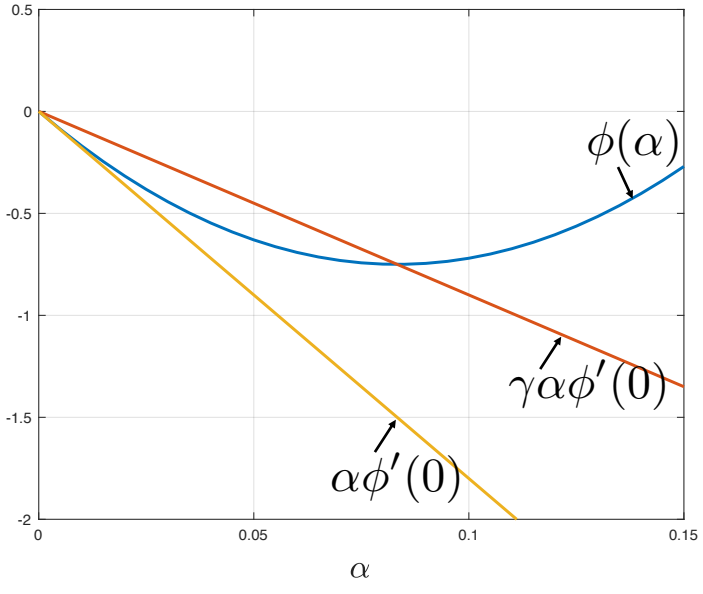
$$\gamma = \frac{1}{2}, \quad \sigma = \frac{1}{2}.$$

Recall that the search direction $\boldsymbol{d}^0 = -\nabla f(\boldsymbol{x}^0)$ in gradient descent method. Now let us define

$$\phi(\alpha) := f(\boldsymbol{x}^0 + \alpha \boldsymbol{d}^0) - f(\boldsymbol{x}^0).$$

Compute the stepsize $\alpha_0$ using the backtracking line search and use such a $\alpha_0$ to compute the next iterate $\boldsymbol{x}^1$.

**Hint:** The following figure may be helpful for applying the backtracking linesearch:



2) Suppose we apply constant stepsize $\alpha_k = \alpha = \frac{1}{12}$ for all $k \geq 0$. Compute the next iterate $\boldsymbol{x}^1$. Verify if $\boldsymbol{d}^1 = -\nabla f(\boldsymbol{x}^1)$ is a descent direction or not. If it is not a descent direction, discuss what conclusion you can draw.

**Problem 3: Gradient Descent Method (30 pts).**
Implement the gradient descent method in Matlab to solve the optimization problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \; f(x) = e^{1-x_1-x_2} + e^{x_1+x_2-1} + x_1^2 + x_1x_2 + x_2^2 + x_1 - 3x_2.$$

The following input parameters should be considered:

- $\boldsymbol{x}^0 = (0,0)^\top$ – the initial point.
- $\epsilon = 10^{-5}$ – the tolerance parameter. The method should stop whenever the current iterate $\boldsymbol{x}^k$ satisfies the criterion $\|\nabla f(\boldsymbol{x}^k)\| \leq \epsilon$.
- $\sigma, \gamma = \frac{1}{2}$ – parameters for backtracking and the Armijo condition.

Implement different methods using the given parameter.

a) Implement the gradient descent method using constant stepsize. Try two different constant stepsizes $\alpha_1 = 1$ and $\alpha_2 = 0.1$. Report the number of iterations, the final objective function value, and the point to which the methods converged, respectively.

b) Implement the gradient descent method using backtracking /Armijo line search method, report the number of iterations, the final objective function value, and the point to which the methods converged.

c) Plot figures of the solution path for each of the different step size strategies (similar to the one in the lecture slides).

Please also submit your code.

**Note:** You can also use Python to implement if you want. We have provided a prototype code in Matlab format in the folder "Code_prototype". You need to fill the parts marked as "**TODO**" by yourself. You can also write the code from scratch if you want.

**Problem 4: Newton's Method (20 pts).**
Implement the Newton's method in Matlab to solve the same problem in Problem 3. Use also the same parameter setup in Problem 3. However, use only Armijo backtracking line search for choosing stepsize in this question. Report the number of iterations, the final objective function value, the point to which the methods converged. Plot figures of the solution path.

**Note:** You can also use Python to implement if you want. We have provide a prototype the code in Matlab format in the folder "Code_prototype". You need to fill the parts marked as "**TODO**" by yourself. You can also write the code from scratch if you want.

**Problem 1: Descent Directions (20 pts).**
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and consider $\boldsymbol{x} \in \mathbb{R}^n$ with $\nabla f(\boldsymbol{x}) \neq 0$. Verify the following statements:

a) Set $\boldsymbol{d} = -(\nabla f(\boldsymbol{x})_j) \cdot \boldsymbol{e}_j = -\frac{\partial f}{\partial x_j} \cdot \boldsymbol{e}_j$, where $\boldsymbol{e}_j \in \mathbb{R}^n$ is the $j$-th unit vector and $j \in \{1, \ldots, n\}$ is an index satisfying

$$\left|\frac{\partial f}{\partial x_j}\right| = \max_{1 \leq i \leq n} \left|\frac{\partial f}{\partial x_i}\right|.$$

Then, $\boldsymbol{d}$ is a descent direction of $f$ at $\boldsymbol{x}$.

b) The direction $\boldsymbol{d} = -\nabla f(\boldsymbol{x})/\|\nabla f(\boldsymbol{x})\|$ is a descent direction of $f$ at $\boldsymbol{x}$.

c) Let $f$ be twice continuously differentiable and define $d_i = -(\nabla f(\boldsymbol{x})_i)/\max\{\nabla^2 f(\boldsymbol{x})_{ii}, \varepsilon\}$ for all $i \in \{1, \ldots, n\}$ and for some $\varepsilon > 0$. Then, $\boldsymbol{d}$ is well-defined (we do not divide by zero) and it is a descent direction of $f$ at $\boldsymbol{x}$.

$$\nabla f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

$$\nabla f(x)^T = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) & \cdots & \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

a) $d$ is a descent direction iff $\nabla f(x)^T d < 0$

$$\nabla f(x)^T d = \left(\frac{\partial f}{\partial x_j}\right)\left(-\nabla f(x)_j \cdot e_j\right)$$

$$= -\left(\frac{\partial f}{\partial x_j}\right)\left(\frac{\partial f}{\partial x_j}\right)$$

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

given $\nabla f(x) \neq 0 \Rightarrow \|\nabla f(x)\|^2 \neq 0$ and thus, $-\|\nabla f(x)\|^2 < 0$

$\Rightarrow \nabla f(x)^T d$ is $< 0$

Thus, $d$ is a descent direction.

b) $d = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$, WTS $d$ is descent dir.

$$\nabla f(x)^T d = -\frac{\nabla f(x)^T \nabla f(x)}{\|\nabla f(x)\|} = -\frac{\|\nabla f(x)\|^2}{\|\nabla f(x)\|} = -\|\nabla f(x)\| < 0$$

Hence $d$ is a descent direction

c) $d_i = -\frac{(\nabla f(x)_i)}{\max\{\nabla^2 f(x)_{ii}, \varepsilon\}}$, $\varepsilon > 0$

Since $\nabla f(x) \neq 0$, $\max\{\nabla^2 f(x)_{ii}, \varepsilon\} > 0$

$$\nabla f(x)^T d = \sum_{i=1}^{n} \left[-\frac{(\nabla f(x)_i)\cdot(\nabla f(x)_i)}{\max\{\nabla^2 f(x)_{ii}, \varepsilon\}}\right]$$

$$= -\sum_{i=1}^{n} \frac{(\nabla f(x)_i)^2}{\max\{\nabla^2 f(x)_{ii}, \varepsilon\}} \rightarrow < 0$$

$\Rightarrow \therefore d$ is a descent direction

**Problem 2: Optimization Algorithm for Regularized Least Squares (30 pts).**
Consider the following unconstrained optimization problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2 + \frac{\lambda}{2}\|\boldsymbol{x}\|^2,$$

where $\boldsymbol{A}\in\mathbb{R}^{m\times n}, \boldsymbol{b}\in\mathbb{R}^m, \lambda\in\mathbb{R}$ are known.

a) Compute the gradient and Hessian of $f$.

$\|x\|^2 = x^Tx$

$\frac{d}{dx}(x^Tx) = 2x$

a)

$$f(x) = \frac{1}{2}\|Ax-b\|^2 + \frac{\lambda}{2}\|x\|^2$$

$$\nabla f(x) = \frac{1}{2}\cdot 2(Ax-b)\cdot A^T + \frac{\lambda}{2}\cdot 2x$$

$$= A^T(Ax-b) + \lambda x$$

$$\nabla^2 f(x) = A^TA + \lambda I$$

-ve or 0 if $\lambda = 0$

$\lambda$ = +ve/0

b) The property of $f$ depends on the parameter $\lambda$. Assume the matrix $\boldsymbol{A}^\top\boldsymbol{A}$ is *not* positive definite. Discuss the corresponding conditions on $\lambda$ such that the optimization problem will be strongly convex, convex, and non-convex.

**Note:** An optimization problem is said to be strongly convex if its Hessian is positive definite.

| $\lambda$-condition | Hessian | optimization problem |
|---|---|---|
| $\lambda > 0$ | Positive Definite | Strongly Convex |
| $\lambda \geq 0$ | Positive Semi definite | Convex |
| $\lambda < 0$ | Negative Definite | non-Convex |

c) Suppose that we want to apply *gradient descent method* to solve this unconstrained optimization problem, where the matrix $\boldsymbol{A}$, vector $\boldsymbol{b}$, and parameter $\lambda$ are defined as:

$$\boldsymbol{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \lambda = 3.$$

We choose the initial point $x^0$ as:

$$\boldsymbol{x}^0 = \begin{bmatrix} \frac{1}{2} \\ \\ \frac{1}{2} \end{bmatrix}.$$

Now consider the following two stepsize rules to determine $\alpha_0$ for updating $\boldsymbol{x}^1$.

1) Suppose we apply Armijo/backtracking line search to select the stepsize $\alpha_0$ with Armijo parameters set as follows:

$$\gamma = \frac{1}{2}, \quad \sigma = \frac{1}{2}.$$

Recall that the search direction $\boldsymbol{d}^0 = -\nabla f(\boldsymbol{x}^0)$ in gradient descent method. Now let us define

$$\phi(\alpha) := f(\boldsymbol{x}^0 + \alpha\boldsymbol{d}^0) - f(\boldsymbol{x}^0).$$

Compute the stepsize $\alpha_0$ using the backtracking line search and use such a $\alpha_0$ to compute the next iterate $\boldsymbol{x}^1$.

**Problem 2: Optimization Algorithm for Regularized Least Squares (30 pts).**
Consider the following unconstrained optimization problem:

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2 + \frac{\lambda}{2}\|\boldsymbol{x}\|^2,$$

where $\boldsymbol{A}\in\mathbb{R}^{m\times n}, \boldsymbol{b}\in\mathbb{R}^m, \lambda\in\mathbb{R}$ are known.

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \left\| \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|^2 + \frac{3}{2} \|x\|^2 \quad , \quad x_0 = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

$$\phi(\alpha) := f(x^0 + \alpha d^0) - f(x^0)$$

$$\phi'(\alpha) = \nabla f(x^0 + \alpha d^0)^T d^0$$

$$\phi_k'(0) = \nabla f(x^k)^T d^k$$

when $k=0$, $\phi_0'(0) = \nabla f(x^0)^T d^0$

$$= \nabla f(x^0)^T \left( -\nabla f(x^0) \right)$$

$$= -\|\nabla f(x^0)\|^2$$

Since $\nabla f(x) = A^T(Ax - b) + \lambda x$

$\nabla f(x^0) = A^T(Ax^0 - b) + \lambda x^0$

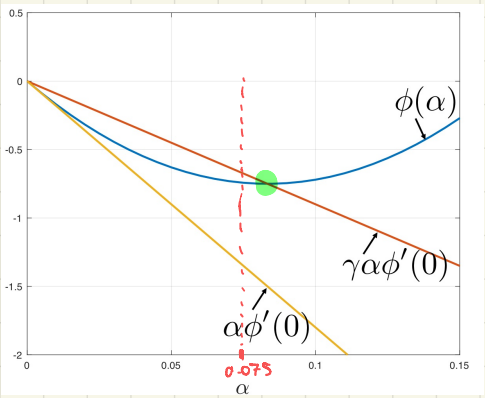
$= \begin{bmatrix} 3 \\ 3 \end{bmatrix}$

$\Rightarrow \|\nabla f(x^0)\|^2 = 3^2 + 3^2 = 18$

$\Rightarrow \phi_0'(0) = -18$ , $d^0 = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$

Armijo Condition :

find $\alpha$ s.t. $\phi_k(\alpha) < \gamma\alpha \cdot \phi_k'(0)$




Backtracking / Armijo line search procedure:

1. Start with $\alpha = 1$.
2. If $f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k) \leq f(\boldsymbol{x}^k) + \gamma\alpha \cdot \nabla f(\boldsymbol{x}^k)^\top \boldsymbol{d}^k$, choose $\alpha_k = \alpha$. Otherwise, set $\alpha = \sigma\alpha$ and repeat this step.

- $\alpha_k$ can be determined after finitely many steps once $\boldsymbol{d}^k$ is a descent direction (see next slide).

using $\alpha_0 \leftarrow \sigma^m a$ and the provided hint graph

$\Rightarrow$ At $m = 4$, $\alpha_0 \leftarrow (\frac{1}{2})^4 (1) = 0.0625$

When $\alpha_0 = 0.0625$, $\phi_0(\alpha = 0.0625) = f\left(\begin{bmatrix} -0.1875 \\ -0.1875 \end{bmatrix}\right) - f\left(\begin{bmatrix} -3 \\ -3 \end{bmatrix}\right)$

$= 2.546875 - 127$

$= -124.4531$

and since $\gamma\alpha_0 \cdot \phi_0'(0) = \frac{1}{2}(0.0625)(-18)$

$= -0.5625$

Larger then $\Rightarrow$ Armijo satisfied

Armijo condition satisfied : $\alpha_0 = 0.0625$

The next itrate, $x^1 = x^0 + \alpha_0 d^0 = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} + (0.0625)\begin{bmatrix} -3 \\ -3 \end{bmatrix} = \begin{bmatrix} 0.3125 \\ 0.3125 \end{bmatrix}$

or

$= \begin{bmatrix} 5/16 \\ 5/16 \end{bmatrix}$

2) Suppose we apply constant stepsize $\alpha_k = \alpha = \frac{1}{12}$ for all $k \geq 0$. Compute the next iterate $\boldsymbol{x}^1$. Verify if $\boldsymbol{d}^1 = -\nabla f(\boldsymbol{x}^1)$ is a descent direction or not. If it is not a descent direction, discuss what conclusion you can draw.

$\alpha = \frac{1}{12}$, $x^1 = x^0 + \left(\frac{1}{12}\right) d^0 = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} + \begin{bmatrix} -3/12 \\ -3/12 \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/4 \end{bmatrix}$ or $\begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix}$

$d^1 = -\nabla f(x^1) = -\left[A^T(Ax^1 - b) + \lambda x^1\right] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

hence, $\nabla f(x^1)^T d^1 = 0$ which is not $< 0$ $\Rightarrow$ $d^1$ is not a descent direction

Conclusion: In this case, constant step size $\alpha$ is not suitable

Using exact line search in (1) is likely to be more suitable

but might be more time-consuming

**Problem 3: Gradient Descent Method (30 pts).**
Implement the gradient descent method in Matlab to solve the optimization problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} f(x) = e^{1-x_1-x_2} + e^{x_1+x_2-1} + x_1^2 + x_1x_2 + x_2^2 + x_1 - 3x_2.$$

The following input parameters should be considered:

- $\boldsymbol{x}^0 = (0,0)^\top$ – the initial point.
- $\epsilon = 10^{-5}$ – the tolerance parameter. The method should stop whenever the current iterate $\boldsymbol{x}^k$ satisfies the criterion $\|\nabla f(\boldsymbol{x}^k)\| \leq \epsilon$.
- $\sigma, \gamma = \frac{1}{2}$ – parameters for backtracking and the Armijo condition.

Implement different methods using the given parameter.

a) Implement the gradient descent method using constant stepsize. Try two different constant stepsizes $\alpha_1 = 1$ and $\alpha_2 = 0.1$. Report the number of iterations, the final objective function value, and the point to which the methods converged, respectively.

b) Implement the gradient descent method using backtracking /Armijo line search method, report the number of iterations, the final objective function value, and the point to which the methods converged.

c) Plot figures of the solution path for each of the different step size strategies (similar to the one in the lecture slides).

Please also submit your code.

**Note:** You can also use Python to implement if you want. We have provided a prototype code in Matlab format in the folder "Code_prototype". You need to fill the parts marked as "**TODO**" by yourself. You can also write the code from scratch if you want.

READ ME:

- Problem 3 & 4 was implemented using python 3, and due to overflow limitations of numpy library, I have adjusted the values for alpha_1 & alpha_2 in order to avoid computing an exponentially large number ( >10e10 )
- alpha_1 <- 0.1
- alpha_2 <- 0.01

a)

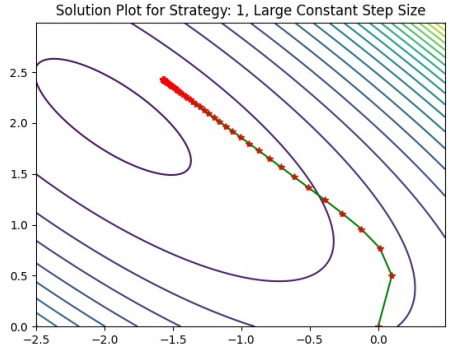
```
Which strategy to use?
 [1]: large constant stepsize
 [2]: small constant stepsize
 [3]: armijo backtracking
1
The algorithm ends after 120 iterations
 x*=[-1.57128395  2.42870314]
 f(x*)=-2.285679688702567
 gradient norm=9.133687029882485e-06
```

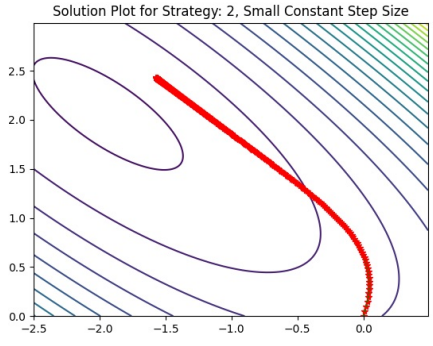
```
Which strategy to use?
 [1]: large constant stepsize
 [2]: small constant stepsize
 [3]: armijo backtracking
2
The algorithm ends after 1249 iterations
 x*=[-1.57128333  2.42870253]
 f(x*)=-2.2856796886943007
 gradient norm=9.997767795899812e-06
```

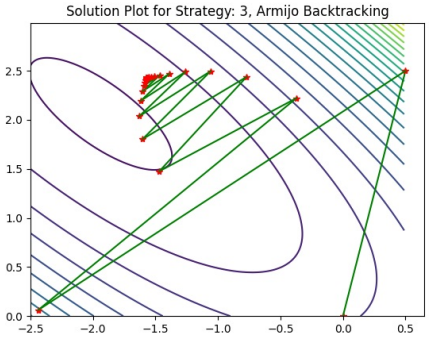
b)

```
Which strategy to use?
 [1]: large constant stepsize
 [2]: small constant stepsize
 [3]: armijo backtracking
3
The algorithm ends after 54 iterations
 x*=[-1.57129096  2.42870844]
 f(x*)=-2.285679688738994
 gradient norm=8.562720141692647e-06
```

c)



**Problem 4: Newton's Method (20 pts).**
Implement the Newton's method in Matlab to solve the same problem in Problem 3. Use also the same parameter setup in Problem 3. However, use only Armijo backtracking line search for choosing stepsize in this question. Report the number of iterations, the final objective function value, the point to which the methods converged. Plot figures of the solution path.

**Note:** You can also use Python to implement if you want. We have provide a prototype the code in Matlab format in the folder "Code_prototype". You need to fill the parts marked as "**TODO**" by yourself. You can also write the code from scratch if you want.

```
The algorithm ends after 35 iterations
 x*=[-1.5712848836594802,2.4287040754560474]
 f(x*)=-2.2856796887138024
 gradient norm=7.807084280772086e-06
```