Technical University of Crete
School of Electrical and Computer Engineering

Course: **Optimization**

Exercise 2 (130/1000)

Angelopoulos Dimitris, 2020030038

A. Let $f : \mathbb{R}^n \to \mathbb{R}$. For fixed $\mathbf{x} \in \mathbb{R}^n$ and $m > 0$, let $g_{\mathbf{x}} : \mathbb{R}^n \to \mathbb{R}$ be defined as the following

$$g_{\mathbf{x}}(\mathbf{y}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{m}{2}||\mathbf{y} - \mathbf{x}||_2^2$$

The gradient of $g_{\mathbf{x}}$ with respect to $\mathbf{y}$ is given by the following equation :

$$\nabla g_{\mathbf{x}}(\mathbf{y}) = \nabla f(\mathbf{x}) + m(\mathbf{y} - \mathbf{x})$$

To compute the optimal point, $\mathbf{y}_* = \text{argmin } g_{\mathbf{x}}(\mathbf{y})$ we will set the gradient equal to $\mathbf{0}$.

$$\mathbf{y}_* = \mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x})$$

The optimal value at the point $\mathbf{y}_*$ is :

$$g_{\mathbf{x}}(\mathbf{y}_*) = f(\mathbf{x}) - \frac{1}{2m}||\nabla f(\mathbf{x})||_2^2$$

B. In this exercise we will solve unconstrained quadratic problems using gradient algorithms.

We consider the problem :

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\, f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

Where $\mathbf{P} \in \mathbb{R}^{n\times n}$, $\mathbf{P} = \mathbf{P}^T \succ \mathbf{O}$ and $\mathbf{q} \in \mathbb{R}^n$

After constructing a $\mathbf{P}$ matrix for which we control λ_{max} and λ_{min} we solve the quadratic problem using different values of condition number.

The algorithms used are both first order gradient algorithms of the form :

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

The difference between them lies in how the step t_k is chosen, as shown below.

Algorithm 1 Gradient Algorithm with exact line-search

$\mathbf{x}_0 \in \mathbb{R}^n$, $k = 0$

While $(||\nabla f(\mathbf{x}_k)|| > \epsilon)$

1. $\Delta \mathbf{x}_k := -\nabla f(\mathbf{x}_k)$
2. $t_k := \frac{||\nabla f(\mathbf{x}_k)||_2^2}{\nabla f(\mathbf{x}_k)^T \mathbf{P} \nabla f(\mathbf{x}_k)}$
3. $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \Delta \mathbf{x}_k$
4. $k := k + 1$

Algorithm 2 Gradient Algorithm with Backtracking line-search

$\mathbf{x}_0 \in \mathbb{R}^n$, $k = 0$, $t_0 = 1$, $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$

While $(||\nabla f(\mathbf{x}_k)|| > \epsilon)$

1. $\Delta \mathbf{x}_k := -\nabla f(\mathbf{x}_k)$
2. While $(f(\mathbf{x}_k + t_k \Delta \mathbf{x}_k) > f(\mathbf{x}_k) + \alpha t_k \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x}_k)$
    1. $t_k := \beta t_k$
3. $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \Delta \mathbf{x}_k$
4. $k := k + 1$

For $n = 2$ we get the following :

(i) $\mathcal{K} = 1$

The closed form and CVX solutions are $p_* = -2.470597$ and $p_* = -2.4706$ respectively.

Now we will run the gradient algorithm. In Fig. 1, on the left is the result of the algorithm using exact line-search when choosing the step. On the right the algorithm uses backtracking to find a step that is sufficient for convergence.

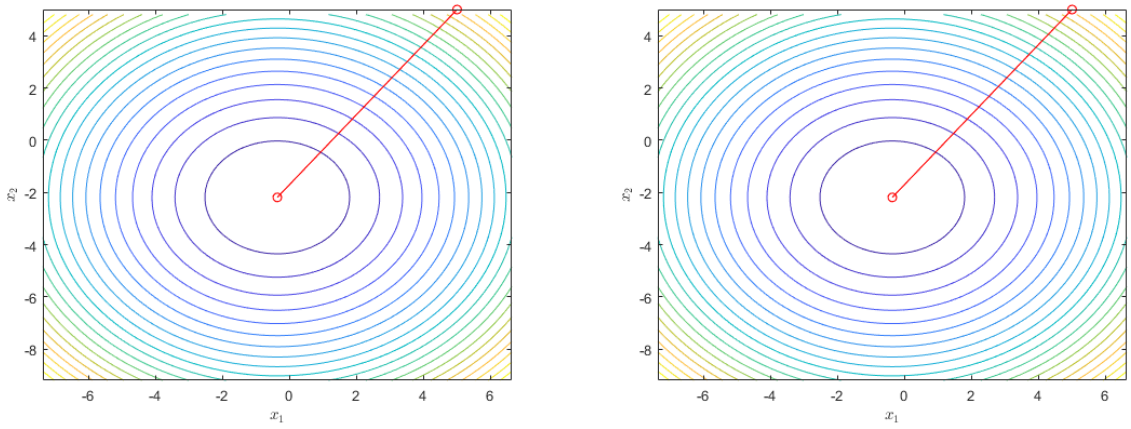

Fig. 1

In this case the iterations needed are 1 for each line-search. The optimal values are the same as they satisfy the same stopping criterion ($\epsilon = 0.01$).

(ii) $\mathcal{K} = 10$

The closed form and CVX solutions are $p_* = -0.079504$ and $p_* = -0.0795037$ respectively.

In Fig. 2, we use larger condition number and that results in more iterations for convergence for every line-search. The backtracking line-search has hyperparameters $\alpha = 0.1$ and $\beta = 0.7$.

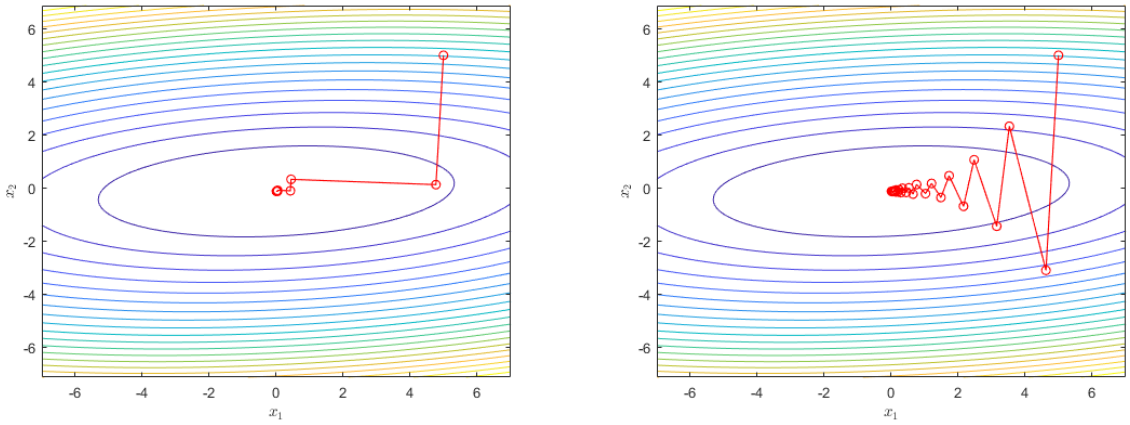

Fig. 2

For the given matrix $\mathbf{P}$ the exact line-search needs only 7 iterations to converge. On the contrary backtracking line-search needs 36. For the same stopping criterion the optimal values of each line search are $p_* = -0.079498$ and $p_* = -0.079494$ respectively.

(iii) $\mathcal{K} = 100$

The closed form and CVX solutions are $p_* = -1.204259$ and $p_* = -1.20426$ respectively.

In Fig. 3, we can see the 2 different line-search methods, but in this case for larger condition number. The hyperparameters and the stopping criterion are the same as the previous case.

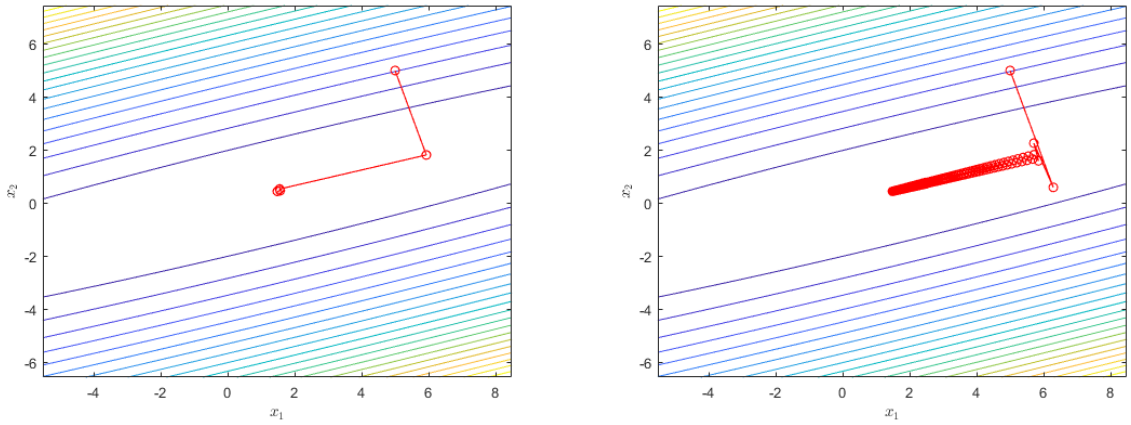

Fig. 3

We observe that the exact line-search takes only 5 iterations. On the other hand the backtracking line-search takes 351, this is the case where we can see the "zig-zag" effect.

Now we will analyze the convergence by plotting the quantity $f(\mathbf{x}_k) - p$,versus k in a common semilogy, for $n = 2$ we get :

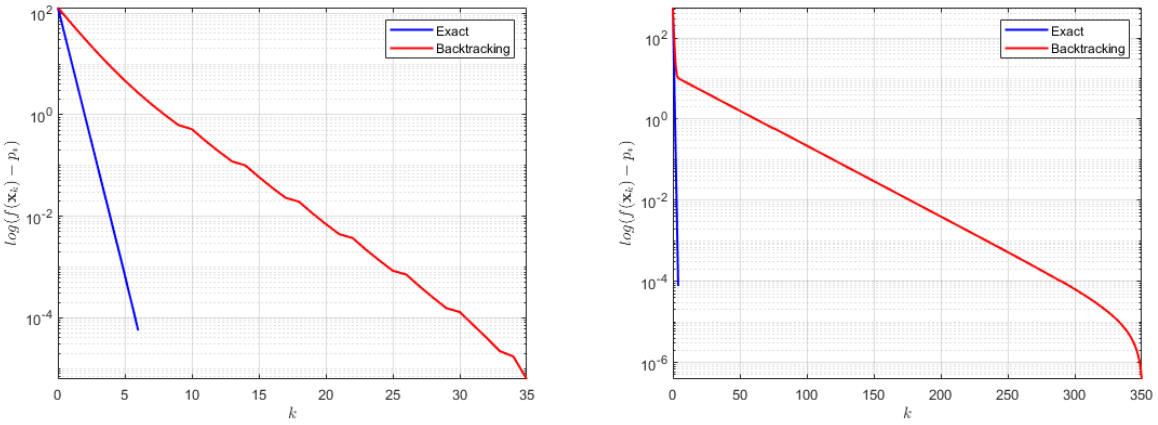

Fig. 4

We observe that as the values of the condition number get larger, the convergence rate of backtracking gets slower.

Now we will set $n = 50$, $\mathcal{K} = 10$ and run the algorithms again to get Fig. 5.

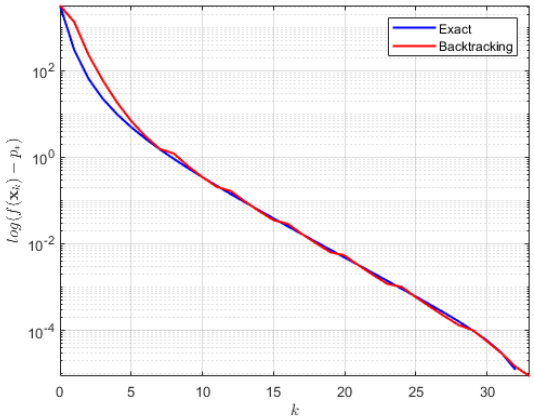

Fig. 5

For higher dimensions we observe that backtracking and exact line-search are almost identical.

Now, for different realizations of f and for different condition numbers we will compute the maximum amount of iterations(k_ϵ) needed to guarantee us ϵ-close accuracy to the closed form solution.

$$k_\epsilon \approx \mathcal{K} \log \left(\frac{f(\mathbf{x}_0) - p_*}{\epsilon} \right)$$

If we run the algorithms for sufficient realizations of f we get an estimation of how many iterations will be needed for convergence in the worst case.

(i) $\mathcal{K} = 10$

We observe that the exact line-search gradient algorithm converges much faster than the estimation for lower dimensions most of the time, $k_\epsilon \in (35, 45)$ for $n = 2$. For higher dimensions k_ϵ is increased and the exact line-search is almost identical to the backtracking line-search. If we run the algorithms a lot of times the worst case estimate might be equal to the number of iterations of either of the algorithms. But generally k_ϵ will be the upper bound.

(ii) $\mathcal{K} = 100$

In this case we observe that $k_\epsilon \in (450, 550)$ for $n = 2$. For larger dimensions the maximum amount of iterations is also increased. As in the previous case the 2 algorithms do not surpass the worst case estimation the majority of the time.

(iii) $\mathcal{K} = 1000$

Lastly $k_\epsilon \in (5000, 6500)$ for $n = 2$. As we increase the dimensions the observations above still hold for this case.

C. Let $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$ with rows $\mathbf{a}_i^T$, for $i = 1, ..., m$. Consider the function $f : \mathbf{dom} f \subseteq \mathbb{R}^n \to \mathbb{R}$, defined as :

$$f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} - \sum_{i=1}^{m} \log(b_i - \mathbf{a}_i^T\mathbf{x})$$

Firstly we will prove that the domain of f is convex. $\mathbf{dom} f$ can be written as following

$$\mathbf{dom} f = \{\mathbf{x} \in \mathbb{R}^n | b_i - \mathbf{a}_i^T\mathbf{x} > 0\}$$

Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{dom} f$ and $\theta \in [0, 1]$.

$$\left.\begin{matrix}\mathbf{a}_i^T\mathbf{x}_1 < b_i \\ \mathbf{a}_i^T\mathbf{x}_2 < b_i\end{matrix}\right\} \Rightarrow \left\{\begin{matrix}\theta\mathbf{a}_i^T\mathbf{x}_1 < \theta b_i \\ (1-\theta)\mathbf{a}_i^T\mathbf{x}_2 < (1-\theta)b_i\end{matrix}\right\} \Rightarrow$$

$$\mathbf{a}_i^T(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) < b_i$$

For $\theta = 1 \Rightarrow \mathbf{a}_i^T\mathbf{x}_1 < b_i$ and for $\theta = 0 \Rightarrow \mathbf{a}_i^T\mathbf{x}_2 < b_i$.

Thus $\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2 \in \mathbf{dom} f \ \forall \ \theta \in [0, 1]$ which means that $\mathbf{dom} f$ is convex.

Now we will prove that $f(\mathbf{x})$ is convex. First of all we will find the gradient of f.

$$\nabla f(\mathbf{x}) = \mathbf{c} + \sum_{i=1}^{m} \frac{\mathbf{a}_i}{b_i - \mathbf{a}_i^T\mathbf{x}}$$

Taking the gradient of this expression yields the Hessian as follows :

$$\nabla^2 f(\mathbf{x}) = \sum_{i=1}^{m} d_i^2 \mathbf{a}_i\mathbf{a}_i^T, \qquad d_i = \frac{1}{b_i - \mathbf{a}_i^T\mathbf{x}}, \qquad i = 1, .., m$$

Now we can verify that $\nabla^2 f(\mathbf{x}) \succeq \mathbf{O}$ because of the fact that for any nonzero vector $\mathbf{v}$ we have :

$$\mathbf{v}^T\mathbf{a}_i\mathbf{a}_i^T\mathbf{v} = (\mathbf{a}_i^T\mathbf{v})(\mathbf{a}_i^T\mathbf{v}) = (\mathbf{a}_i^T\mathbf{v})^2 \geq 0$$

Now we will minimize f using CVX for $(n, m) = (2, 20)$. The optimal value of f is $p_* = -83.6414$. Plotting the level sets of f near the minimum we get the following figure :

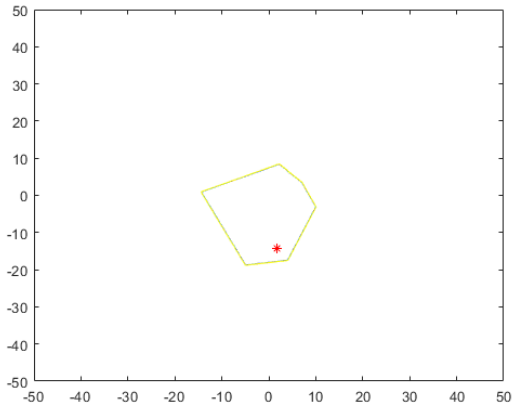

Fig. 6

To minimize the function we will implement the 2 algorithms below, both using backtracking line-search. The problem with this implementation is that there may exist a step in the algorithm that might not be feasible. If this is the case then we will use backtracking, choosing sufficient t_k that yields $\mathbf{x}_{k+1}$ feasible and then proceed. If we run into such problem a message will be printed. The first algorithm is the gradient algorithm, while the second is the Newton algorithm. Notice that the gradient algorithm is the same as the previous exercise but modified as shown below.

Algorithm 3 Gradient Algorithm with Backtracking line-search

$\mathbf{x}_0 \in \mathbf{dom} f$, $k = 0$, $t_0 = 1$, $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$

While $(||\nabla f(\mathbf{x}_k)|| > \epsilon)$

1. $\Delta \mathbf{x}_k := -\nabla f(\mathbf{x}_k)$
2. While (True)
    1. If $(\mathbf{x}_k + t_k \Delta \mathbf{x}_k \notin \mathbf{dom} f)$, Then $t_k := \beta t_k$
    2. Else, Break
3. While $(f(\mathbf{x}_k + t_k \Delta \mathbf{x}_k) > f(\mathbf{x}_k) + \alpha t_k \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x}_k)$
    1. $t_k := \beta t_k$
4. $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \Delta \mathbf{x}_k$
5. $k := k + 1$

Algorithm 4 Newton Algorithm with Backtracking line-search

$\mathbf{x}_0 \in \mathbf{dom} f$, $k = 0$, $t_0 = 1$, $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$

While $(\nabla f(\mathbf{x}_k)^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) > 2\epsilon)$

1. $\Delta \mathbf{x}_k := -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$
2. While (True)
    1. If $(\mathbf{x}_k + t_k \Delta \mathbf{x}_k \notin \mathbf{dom} f)$, Then $t_k := \beta t_k$
    2. Else, Break
3. While $(f(\mathbf{x}_k + t_k \Delta \mathbf{x}_k) > f(\mathbf{x}_k) + \alpha t_k \nabla f(\mathbf{x}_k)^T \Delta \mathbf{x}_k)$
    1. $t_k := \beta t_k$
4. $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \Delta \mathbf{x}_k$
5. $k := k + 1$

After implementing these algorithms in MATLAB we get the following plots for different (n, m). Notice that the hyperparameters α, β and ϵ are the same in both algorithms. More specifically $\alpha = 0.1$, $\beta = 0.7$ and $\epsilon = 0.001$.

(i) $n = 2$ and $m = 20$

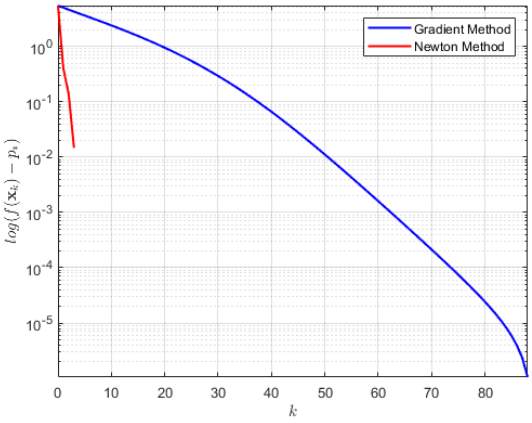

Fig. 7

The optimal values yielded by the algorithms are $p_* = -83.6413$ and $p_* = -83.6412$. We can see that all 3 tools (CVX and the 2 algorithms) find an optimal value that is the same as of to the third decimal digit.

(i) $n = 50$ and $m = 200$

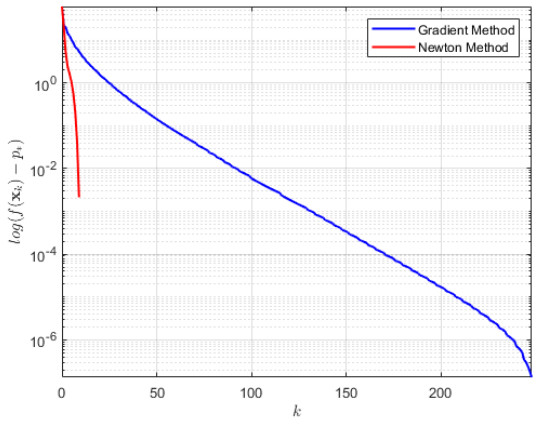

Fig. 8

(iii) $n = 300$ and $m = 800$

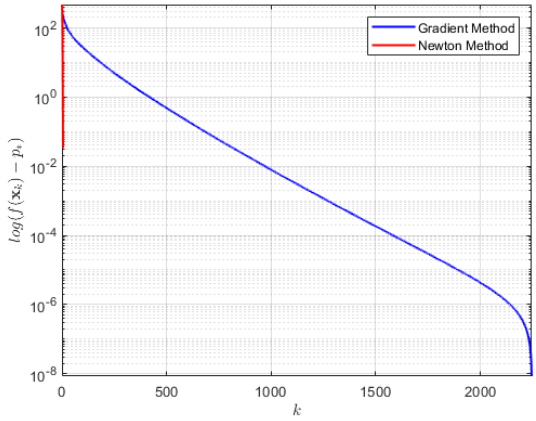

Fig. 9

In the last two cases we can see that the gradient algorithm converges a lot slower than the newton algorithm. For larger values of the pair (n, m) the gradient algorithm becomes worse. On the contrary the Newton method converges with the same rate.