ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Optimization

Gradient Descent and Newton methods

School of Electrical and Computer Engineering

Michailidis Stergios - 2020030080

November 2023

1 A.

(i) Begin by computing the gradient:

$$
\begin{aligned}
\nabla g_{\mathbf{x}}(\mathbf{y}) = \frac{\partial}{\partial \mathbf{y}} g_{\mathbf{x}}(\mathbf{y}) &= \frac{\partial}{\partial \mathbf{y}} f(\mathbf{x}) + \frac{\partial}{\partial \mathbf{y}} \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} \|\mathbf{y} - \mathbf{x}\|_2^2 \\
&= \nabla f(\mathbf{x}) \frac{\partial}{\partial \mathbf{y}} (\mathbf{y} - \mathbf{x}) + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} [(\mathbf{y} - \mathbf{x})^T \mathbb{I}_n (\mathbf{y} - \mathbf{x})] \\
&= \nabla f(\mathbf{x}) + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} [(\mathbf{y}^T - \mathbf{x}^T)(\mathbf{y} - \mathbf{x})] \\
&= \nabla f(\mathbf{x}) + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} [\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{x} - \mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{x}] \\
&= \nabla f(\mathbf{x}) + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} [\|\mathbf{y}\|_2^2 - 2\mathbf{x}^T\mathbf{y} + \|\mathbf{x}\|_2^2] \\
&= \nabla f(\mathbf{x}) + \frac{m}{2} [\frac{\partial}{\partial \mathbf{y}} \|\mathbf{y}\|_2^2 - 2\frac{m}{2}\frac{\partial}{\partial \mathbf{y}} \mathbf{x}^T\mathbf{y} + \frac{m}{2}\frac{\partial}{\partial \mathbf{y}} \|\mathbf{x}\|_2^2] \\
&= \nabla f(\mathbf{x}) + m\mathbf{y} - m\mathbf{x}
\end{aligned}
$$

(ii) The optimal point y_* is found by solving:

$$
\nabla g_{\mathbf{x}}(\mathbf{y}_*) = 0 \iff \nabla f(\mathbf{x}) + m\mathbf{y}_* - m\mathbf{x} = 0 \iff \mathbf{y}_* = \mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) \tag{1}
$$

then we can calculate the optimal function value by plugging in relation (1) in the given formula:

$$
\begin{aligned}
g_{\mathbf{x}}(\mathbf{y}_*) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y}_* - \mathbf{x}) + \frac{m}{2} \|\mathbf{y}_* - \mathbf{x}\|_2^2 \\
&= f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) - \mathbf{x}) + \frac{m}{2} \left\|\mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) - \mathbf{x}\right\|_2^2 \\
&= f(\mathbf{x}) - \frac{1}{m}\|\nabla f(\mathbf{x})\|_2^2 + \frac{m}{2}\left\|\frac{1}{m}\nabla f(\mathbf{x})\right\|_2^2 \\
&= f(\mathbf{x}) - \frac{2}{2m}\|\nabla f(\mathbf{x})\|_2^2 + \frac{1}{2m}\|\nabla f(\mathbf{x})\|_2^2 \\
&= f(\mathbf{x}) - \frac{1}{2m}\|\nabla f(\mathbf{x})\|_2^2
\end{aligned}
$$

2 B.

2.1 (a)

Lets begin by observing that U is unitary: $U^T U = UU^T = \mathbb{I}$

2.2 (b),(c)

See relevant matlab code. Note that the selection of hyperparameters has been from beck's notes: $\alpha = 0.1, \beta = 0.7$

2.3 (b) iii.

Closed form solution:

$$\begin{aligned}\nabla f(x) &= \nabla(\frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}) \\ &= \nabla(\frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x}) + \nabla(\mathbf{q}^T\mathbf{x}) \\ &= \mathbf{P}\mathbf{x} + \mathbf{q}\end{aligned} \tag{2}$$

And the hessian is:

$$H(x) = \nabla^2 f(x) = \nabla(\nabla f(x)) = \mathbf{P} \tag{3}$$

with $\mathbf{P} \succ 0$ by definition. Thus the function is convex, so it attains a minimum at $\mathbf{x}_*$ which is the following:

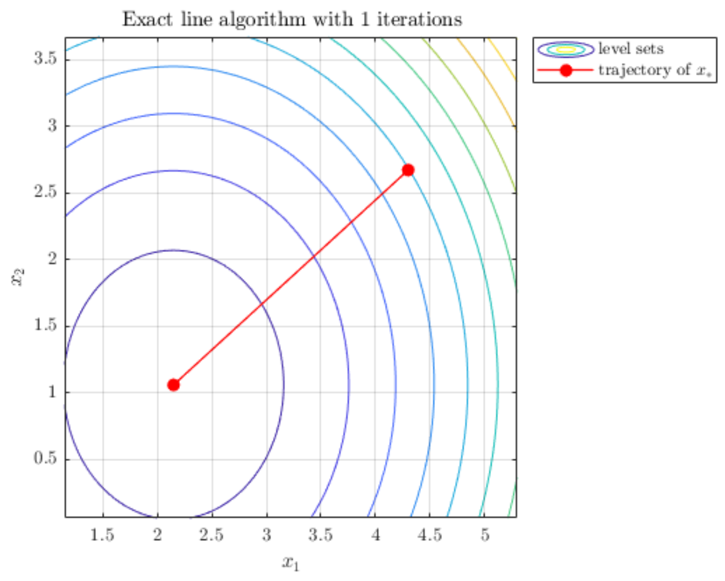
$$\nabla f(\mathbf{x}_*) = 0 \iff \mathbf{x}_* = -\mathbf{P}^{-1}\mathbf{q} \tag{4}$$

2.4 iv.

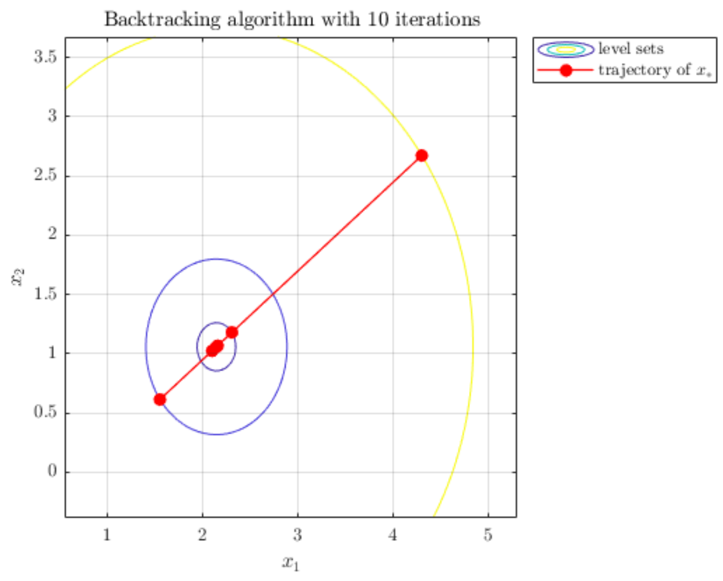
CVX gives us the following solution, for $K = 1 : p_* = -0.806458$

2.5 v.

For $K = 1$ the exact line search requires only 1 iteration to complete. See the figure below:



The backtracking method has to execute more iterations, as seen here:



As we will see next , the exact line method is much faster than the backtracking one.

2.6 vi.

Let us now see the contours and the trajectories for larger and larger K:

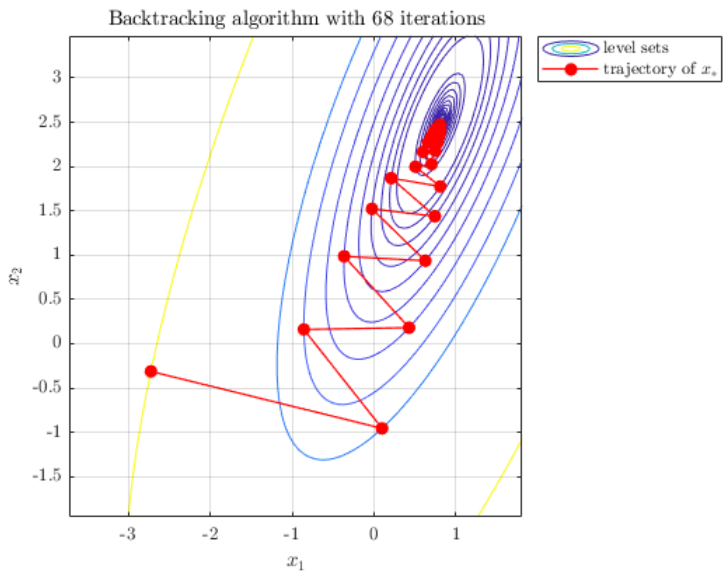


Figure 1: Backtracking for K = 10

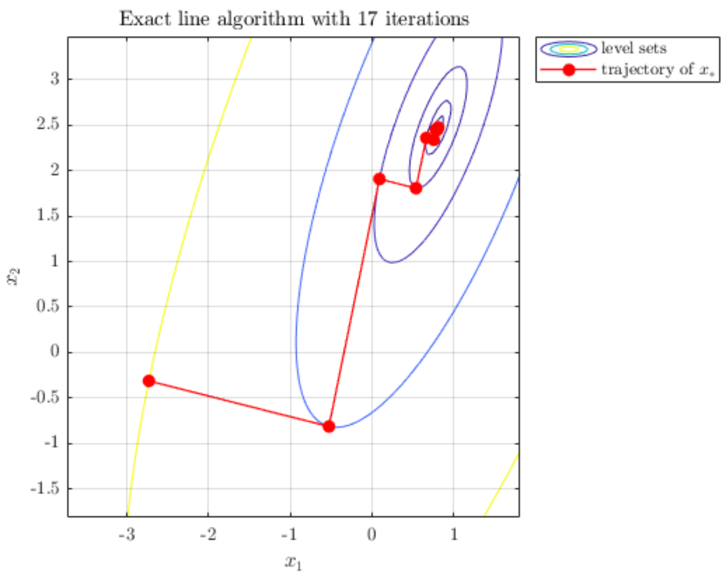


Figure 2: Exact line for K = 10

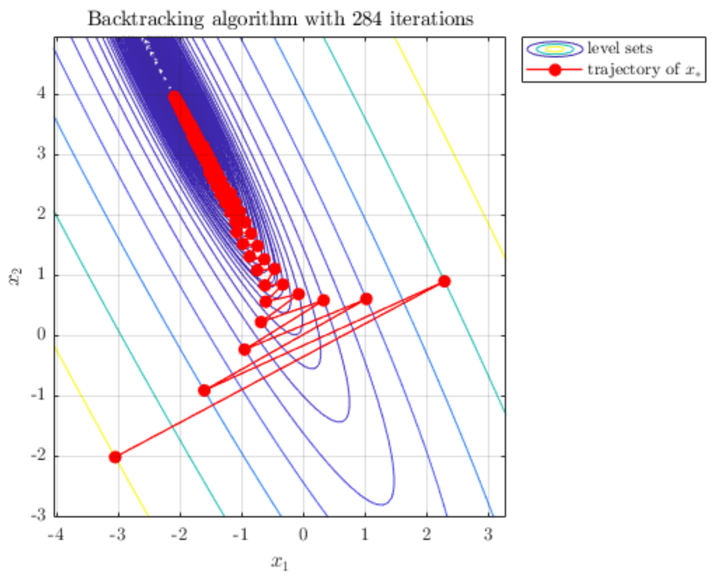


Figure 3: Backtracking for K = 50

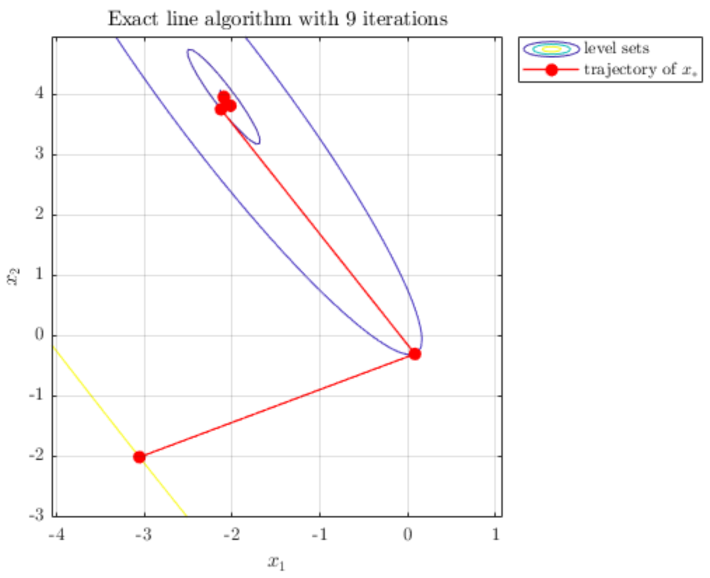


Figure 4: Exact line for K = 50

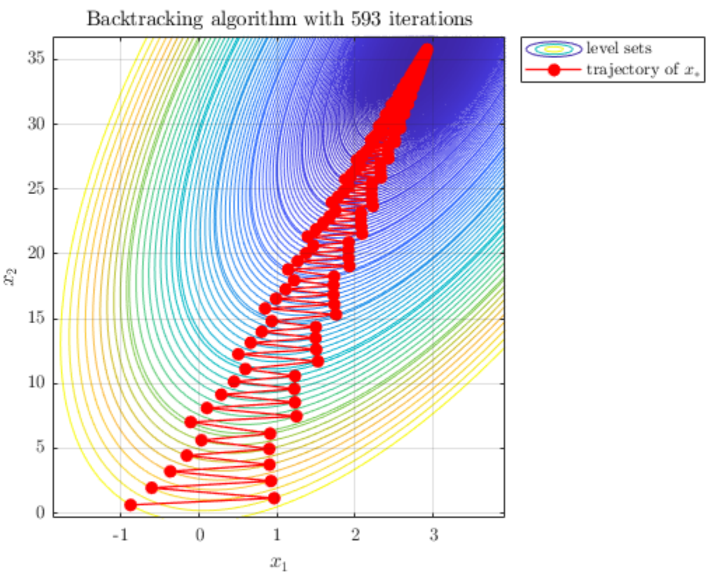


Figure 5: Backtracking for K = 100

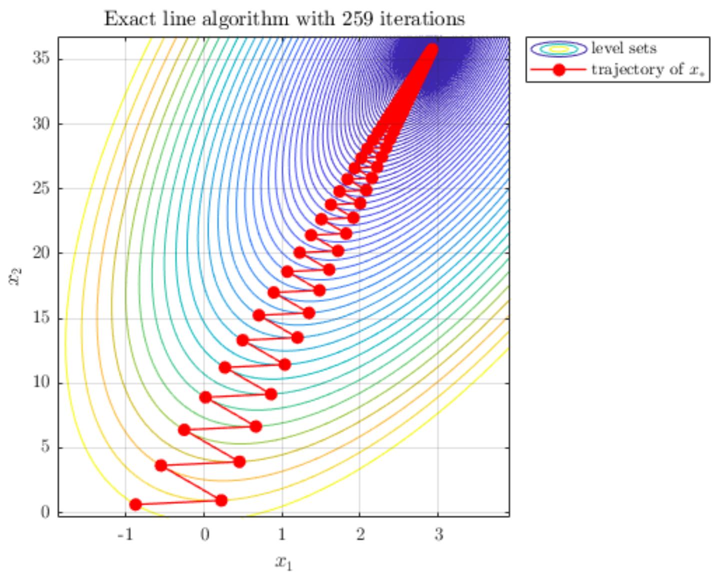


Figure 6: Exact line for K = 100

and now lets observe a rare case :

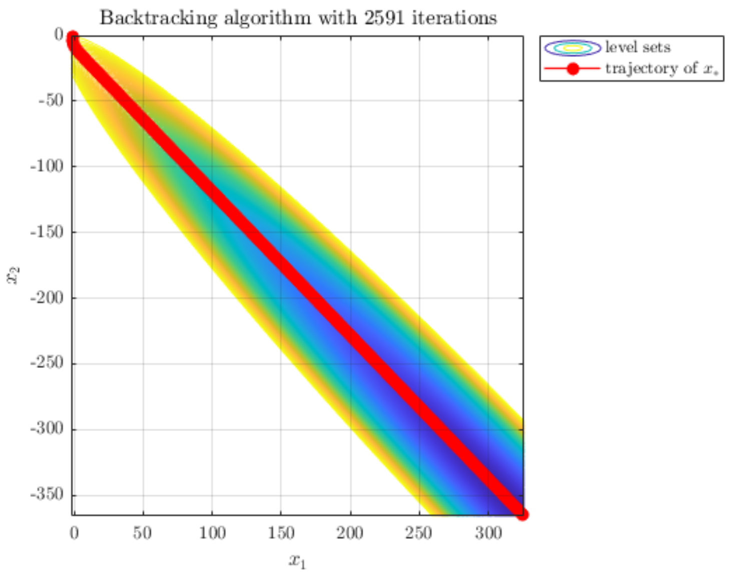


Figure 7: Backtracking for K = 100 rare case

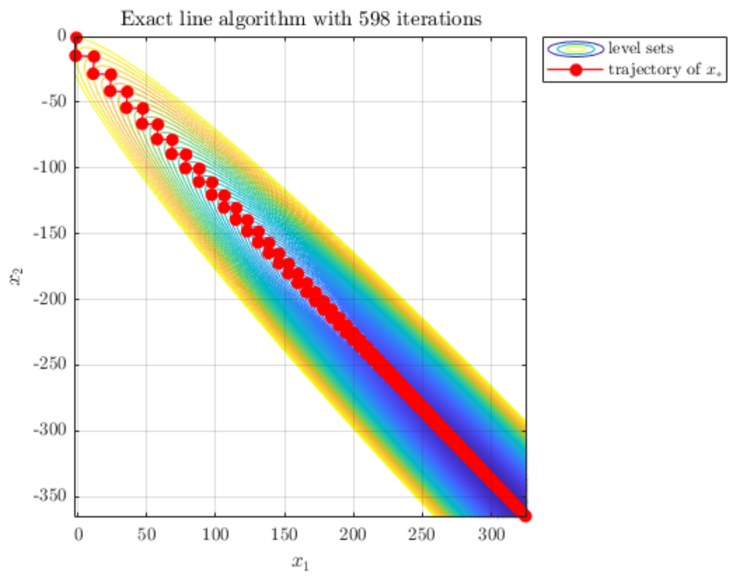


Figure 8: Exact line for K = 100 rare case

2.7 vii.

Generally we can conclude that increasing K and the dimension will increase the number of iterations needed for the algorithms to converge, as shown by the graphs:

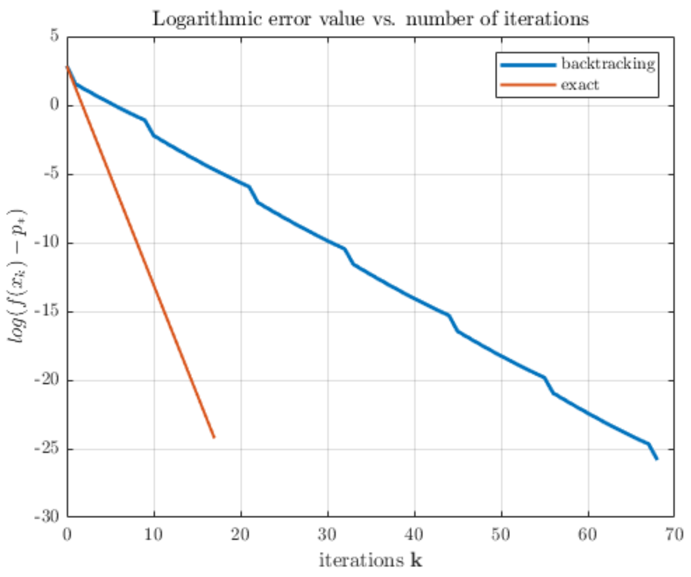


Figure 9: K = 10

lastly we have both cases of $K = 100$

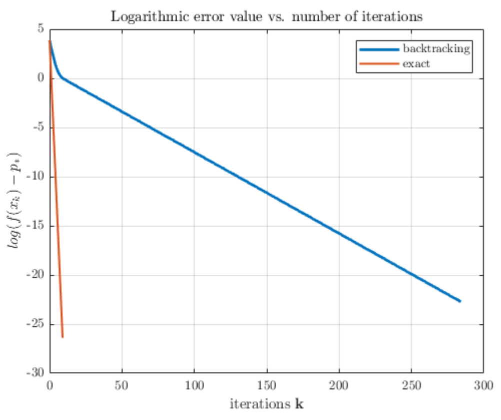


Figure 10: K = 50

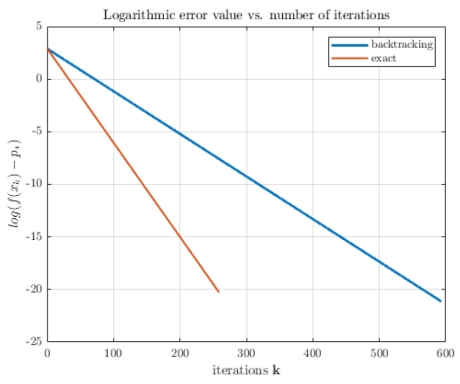
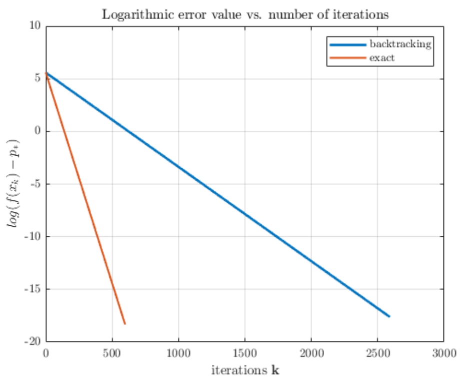


Figure 11: K = 100 both cases

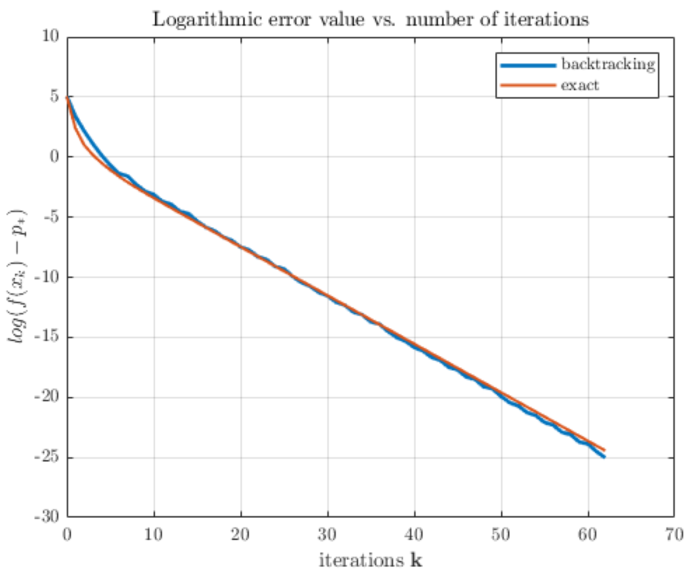


Figure 12: K = 10 — n = 15

note that for higher dimension where the computational complexity increases, the backtracking and exact line algorithms are practically the same.

2.8 viii.

see corresponding convergence analysis matlab code. NOTE: this section of the code seems to have trouble executing in some machines for $K > 50$. To avoid having issues and/or to test the other sections quicker, you can set flag $= 0$ in the section of the code to bypass it.
Lets now compare the theoretical and the practical amount of iterations of the gradient descent algorithms:
In the Convergence analysis Lecture notes, we have defined:

$$c = \begin{cases} 1 - \frac{m}{M} \\ 1 - min2ma, \frac{2\beta\alpha m}{M} \end{cases}$$

The upper bounds for the two algorithms used are as follows:

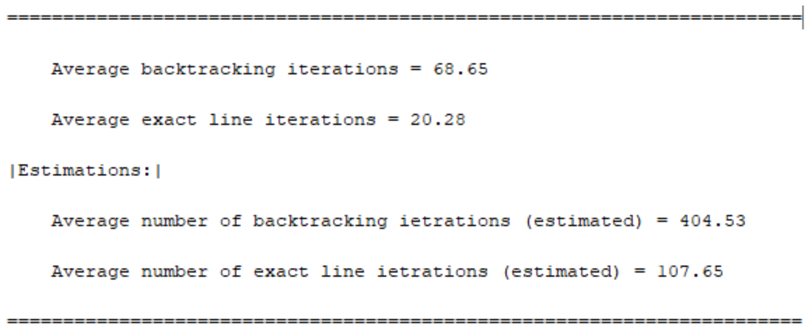
$$k_\epsilon \approx Klog(\frac{f(x_0) - p_*)}{\epsilon}) \tag{5}$$

for the exact line , and

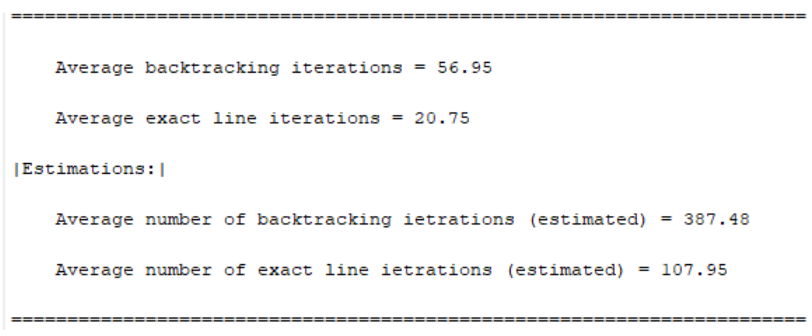
$$k \geq k_\epsilon = \frac{log(\frac{f(x_0) - p_*))}{\epsilon}}{log(1/c)} \tag{6}$$

for the backtracking method.

Lets take a look at the statistics:



```
========================================================================

    Average backtracking iterations = 68.65

    Average exact line iterations = 20.28

|Estimations:|

    Average number of backtracking ietrations (estimated) = 404.53

    Average number of exact line ietrations (estimated) = 107.65

========================================================================
```



```
========================================================================

    Average backtracking iterations = 56.95

    Average exact line iterations = 20.75

|Estimations:|

    Average number of backtracking ietrations (estimated) = 387.48

    Average number of exact line ietrations (estimated) = 107.95

========================================================================
```

We conclude that the actual iteration number is much smaller than the estimated iteration number needed to converge.

3 C.

3.1 (a)

Let's begin the proof:
The domain of f is:

$$\mathbb{D}_f = \{x \in \mathbb{R}^n | a_i^T < b_i, \forall i = 1, 2, 3, ..., m\}$$

Pick 2 arbitrary points $x_1, x_2 \in \mathbb{D}_f$ in the domain and $\theta \in [0, 1]$

$$a^T x_1 \leq b \iff \theta a^T x_1 \leq \theta b$$
$$a^T x_2 \leq b \iff (1-\theta) a^T x_2 \leq (1-\theta) b$$

adding these two inequalities, we have

$$\theta a^T x_1 + (1-\theta) a^T x_2 \leq (1-\theta) a^T x_2 \leq b$$

thus the point $(x_1, x_2) \in \mathbb{D}_f$, so the set $\mathbb{D}_f$ is convex.

3.2 (b)

Lets compute the function's gradient:

$$\begin{aligned}\nabla f(x) &= \nabla[c^T x - \sum_{i=1}^{m} log(b_i - a_i^T x)] \\ &= \nabla c^T x - \sum_{i=1}^{m} \nabla log(b_i - a_i^T x) \\ &= c + \sum_{i=1}^{m} \frac{1}{b_i - a_i^T x} \nabla(b_i - a_i^T x) \\ &= c + \sum_{i=1}^{m} \frac{a_i}{b_i - a_i^T x}\end{aligned} \tag{7}$$

and the hessian:

$$\begin{aligned}\mathbf{H}(x) = \nabla^2 f(x) = \nabla(\nabla f(x)) &= \nabla[c + \sum_{i=1}^{m} \frac{a_i}{b_i - a_i^T x}] \\ &= 0 + \sum_{i=1}^{m} \nabla \frac{a_i}{b_i - a_i^T x} \\ &= (-1) \sum_{i=1}^{m} \frac{a_i^T}{(b_i - a_i^T x)^2} \nabla(b_i - a_i^T x) \\ &= \sum_{i=1}^{m} \frac{a_i^T a_i}{(b_i - a_i^T x)^2}\end{aligned} \tag{8}$$

where $a_i^T a_i = \|a_i\|_2^2$. It is clear that the hessian matrix is p.s.d: $\mathbf{H} \succ \mathbf{0}$. Therefore the function f is convex.

3.3 a.

see corresponding matlab code for cvx minimization.

3.4 b.

function f and its level sets are plotted together using the command *meshc*: (optimal $f(x_*)$ shown in red)

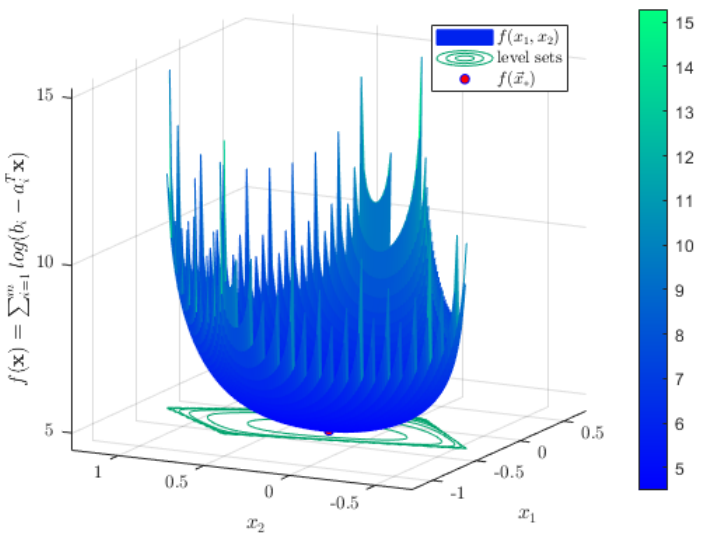


Figure 13: surface of f with its contours — n=2 m=5

and the contours separately:

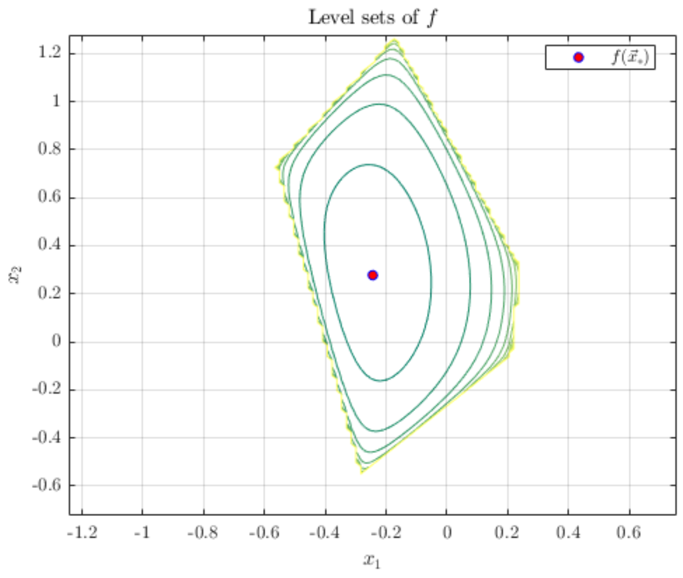


Figure 14: contours of f — n=2 m=5

3.5 c. d. e.

Newton's method computes the hessian thus making it very computationally demanding, especially in higher dimensions. But this also its advantage as the hessian contain much more information making it more efficient. In conclusion, newton's method is better from the backtracking line search, especially in higher dimensions:

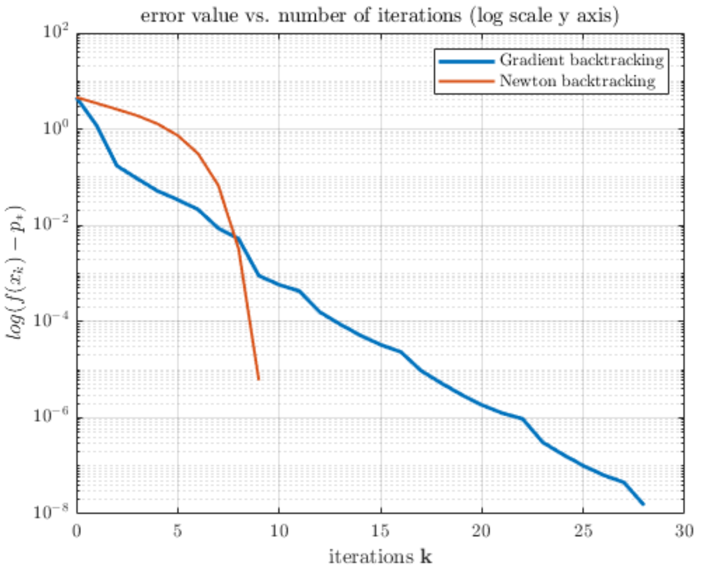


Figure 15: Semilog plot — n=2 m=5

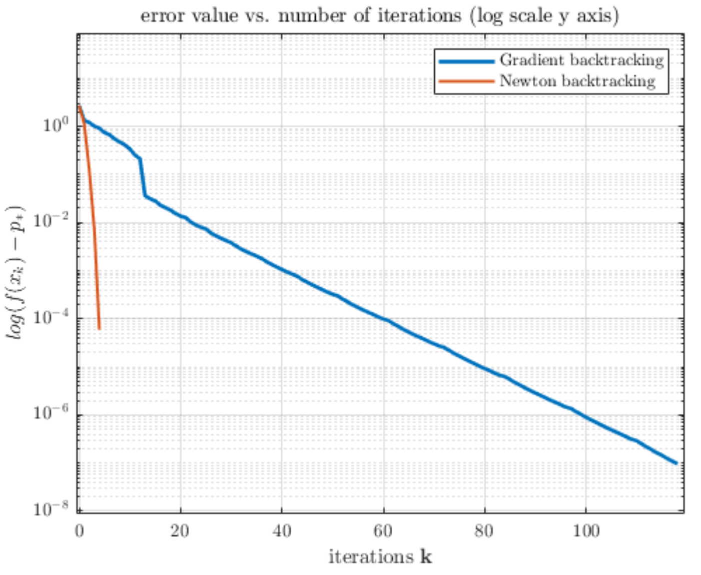


Figure 16: Semilog plot — n=2 m=5

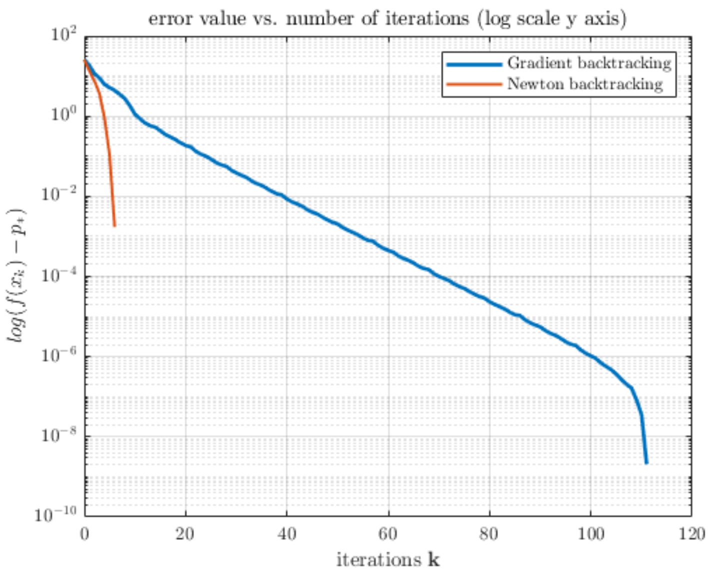


Figure 17: Semilog plot — n=20 m=200

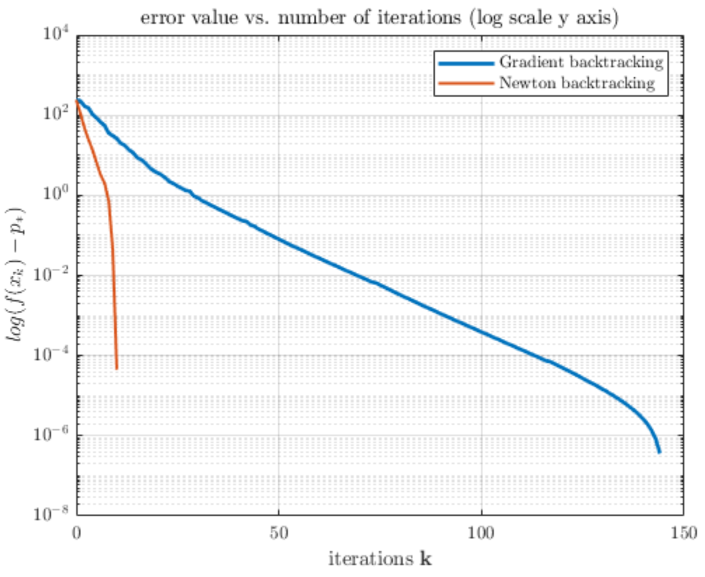


Figure 18: Semilog plot (k power of 10) — n=200 m=3000