Second Problem Set - Optimization Course

Leonidas Bakopoulos A.M. 2018030036

November 24, 2022

Introduction

In this report are solved the exercises from the second pset of the optimization class.

A

$q_x(y) = f(x) + \nabla f^\top(x)(y-x) + \frac{m}{2}\|y-x\|_2^2$ knowing that $\nabla\|x\|_2^2 = 2x$
i) $\nabla g_x(y) = \nabla f^\top(x) + \frac{m}{2}\ 2(y-x)$
(ii)In order to find y_* we can set $\nabla g_x(y) = 0 \Rightarrow y_* = -\frac{\nabla f^\top(x) + mx}{m}$
and $g_x(y_*) = f(x) + \nabla f(x)^\top(-\frac{\nabla f^\top(x)}{m}) + \frac{m}{2}\left\|-\frac{\nabla f^\top(x)}{m}\right\|_2^2 = f(x) - \frac{\left(\nabla f^\top(x)\right)^2}{m} - \frac{1}{2}\left\|\nabla^\top f(x)\right\|_2^2$

B

In the second exercise, P and q was constructed using the instructions given (P with positive eigenvalues, in order to be positive definite).

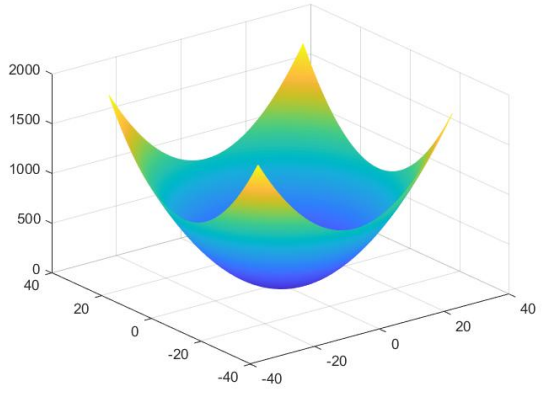


Figure 1: Mesh while K=1.

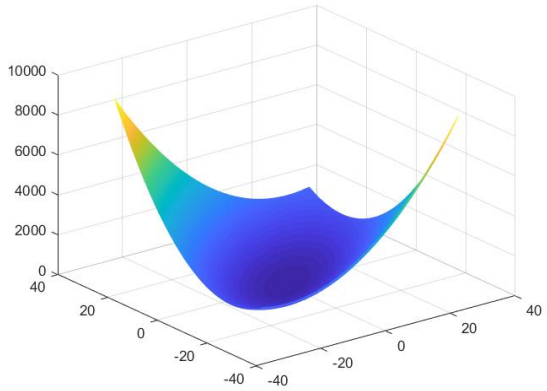


Figure 2: Mesh while K=5.

iii)

$f(x) = \frac{1}{2}x^\top Px + q^\top x$ so
$\nabla f(x) = Px + q^\top$
Using the closed form solution, we can see that:
$\nabla f(x) = 0 \Rightarrow$
$Px + q^\top = 0 \Rightarrow$
$x_* = -P^{-1}q^\top$

v)

The function described above, was minimized using the exact and the backtracking method, as they were taught in the lecture (the complete implementation can be found in the matlab scripts). Main section of the exact method

```
%t_opt was calculated outside this function
t_opt=(norm(g)^2)/(g'*P*g);
...
...
...
def exact:
    while norm(grad)>epsilon
        Dx=-grad;
        x=x+t_opt*D_x; %new value of x
        k=k+1; %counter for the iterations
        grad=g(x,P,q);
    end
```

Main section of the backtracking method (Beck's book)

```
while (norm(grad)>epsilon)
    t=s;
    while (fun_val-f(x-t*grad, P,q)<alpha*t*norm(grad)^2)
        t=beta*t;
    end
    x=x-t*grad;
    fun val=f(x,P,q);
    grad=g(x, P, q);

end
```

When K=1, the exact method, needs just one iteration to approximate the optimal value. While K is increasing, the exact method needs more iterations than the backtracking one, in order to converge. Since we are searching iteratively, in the following graphs are presented numerous results that derived

from the repetitions of the algorithm, for k=25. We can see, that both algorithms end up at the same value (that is slightly different from cvx). Also, as it was expected, the "zig zag effect" appears in the backtracking method, but not in the exact. In the exact method, we can see the expected two orthogonal edges. [1]

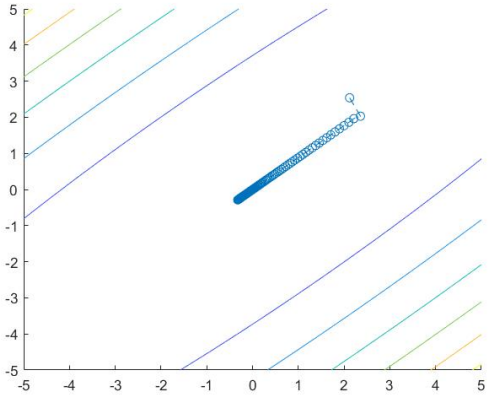


Figure 3: Contour while K=25, optimization using exact method.

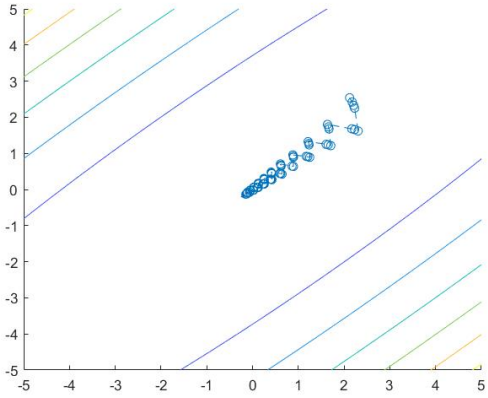


Figure 4: Contour while K=25, optimization using backtrack method.

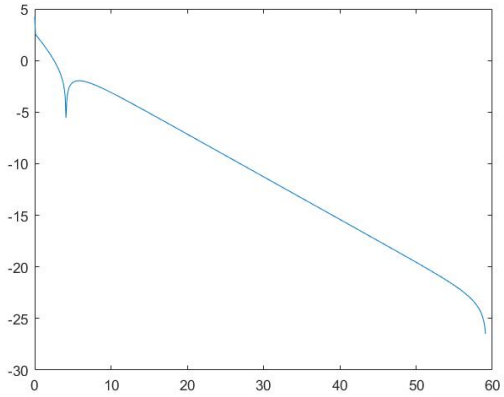


Figure 5: $\log(f(x_k) - p_*)$ while K=25, optimization using exact method.

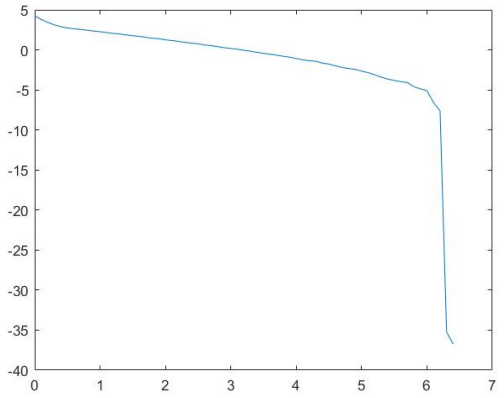


Figure 6: $\log(f(x_k) - p_*)$ while K=25, optimization using exact method.

vii)

As it was expected, the logarithmic function of the previously mentioned points minus the global minimum, is approaching negative infinite values, as the iterations continue (because in k+1 iteration, the point is closer to the global minimum).

C

a)

Dom of f is $Ax > b$ that is a convex (almosted proved in the previous pset).
Proof:
Let x_1, x_2 be any two points in halfspace $H : Ax > b$. Then $\forall \theta \epsilon [0,1]$ we can see that :
$A(\theta x_1 + (1-\theta)x_2) = \theta A x_1 + (1-\theta)Ax_2 > \theta b + (1-\theta)b = b$

b)

Note: b was assigned a greater value, in order to avoid infeasible problems.

```
A=((-1).^(randi([0,1],m,n))).*rand(m,n); %both negatives and positives values in A
c=rand(n,1);
b=10*rand(m,1);
```

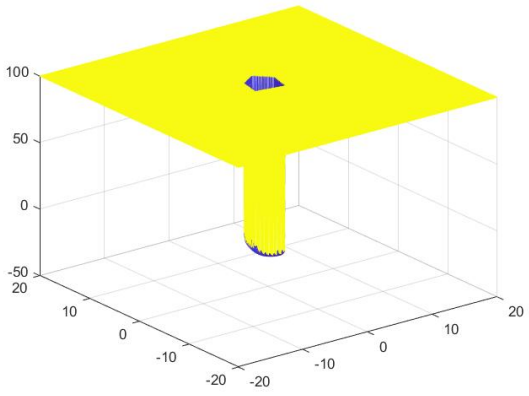


Figure 7: mesh for a 2-variable function.

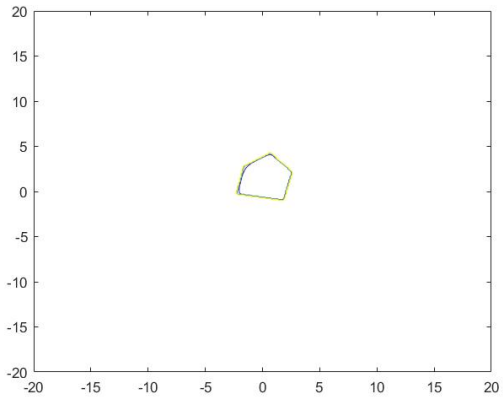


Figure 8: level sets for a 2-variable function.

c)

Below can be seen the backward algorithm as implemented in the previous exercise, with a little twist, in order to check if x belongs in dom f.

```
while (norm(grad)>epsilon)
    t=s;
    while (fun_val-f(x-t*grad,A,b,c)<alpha*t*norm(grad)^2)
        t=beta*t;
    end
    x=x+t*grad;
    if(¬less_than_zero(b-A*x)) %%just in case to prevent, that x belongs in dom(f)
        fun_val=f(x,A,b,c);
        grad=g(x, A,b,c)';
    end

end
```

Note: The less than zero method(x), returns True iff $x_i > 0 \forall i \epsilon [0, size(x)]$

d)

Newton main section for optimization

```

while (lambda>2*epsilon)
    t=s;
    while (fun_val-f(x-t*grad,A,b,c)<alpha*t*norm(grad)^2)
        t=beta*t;
    end

    grad=g(x,A,b,c)';
    h=hessian(x,A,b);
    x=x+t.*h*grad;

    if(dom(A,b,x)==1) %%just in case to prevent, that x belongs in dom(f)
        lambda=(grad'*h').*grad;
        fun_val=f(x,A,b,c);
        grad=g(x, A,b,c)';
   end


end
```

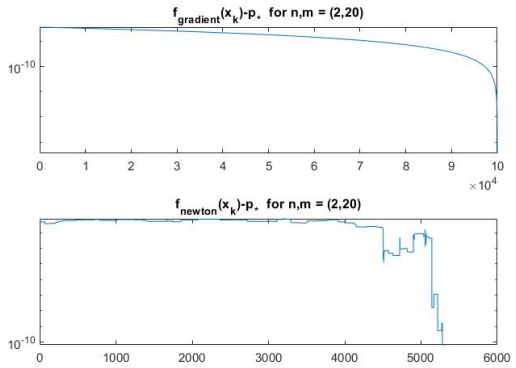
Differences between the Gradient and Newton method

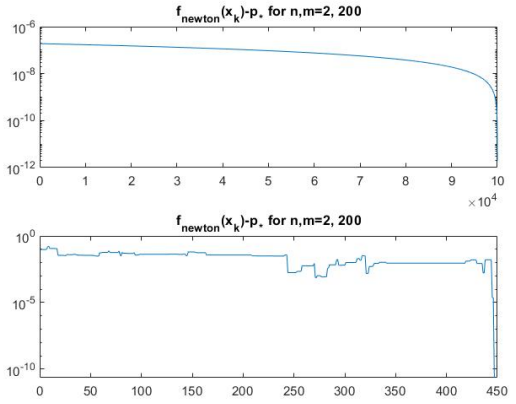
```
CVX
Status: Solved
Optimal value (cvx_optval): -26.46
...
Gradient
iter_number = 100000 norm_grad = 0.69601549154537056818270457370090 fun_val = ...
    -26.414653121659458889780580648221O8 and  point is ...
    0.000000006484287279866203155859 71, 0.000000002529190791662209549555 66
...
Newton's method
iter_number = 3296 norm_grad = 0.696462 fun_val = ...
    -26.411293217141324163321769447065 89  and optimal point is ...
    -0.013592541400463930764686004692 98, -0.013592541400463930764686004692 98
```

Above is shown a "screenshot" of the matlab console (it is attached as a screenshot for proof in the end of the report). As it is shown, both newton's and gradient converge on the same result (slightly different from cvx's result.) On the other hand, starting from the same point, the gradient method

needs more iterations than newton's for the same epsilon.
Note: For complexity reasons, (and the fact that my pc almost caught on fire...), the gradient method stopped manually at ten thousand iterations.

e)





As can be seen from the graphs above, Newton's algorithm needs lees iterations in order to satisfy the stopping criterion, but ends up having worse results than Newton. So in conclusion, from all the problem set,we figured out that if an engineer needs a fast optimization algorithm (with very satisfying (optimal) results),he should choose newton's algorithm or as a second solution, backtracking gradient decent.

References

[1]https://web.stanford.edu/class/ee364a/lectures/unconstrained.pdf

Proof (screenshot) for Exercise C (d)