MATH-329 Nonlinear optimization Homework 1: gradient descent and convexity

Benoît Müller
Instructor: Nicolas Boumal
TAs: Quentin Rebjock and Axel Séguin

October 4, 2023

Remarks on the code

Each file named questionx.m is the main file of the question number x. The other files are data or functions named after their utility. Running the main script of a question give the output detailed in this document.

We use showMNISTImage.m and showMNISTImages_many.m as it has been given in the starting file, without changing them.

Answers to the questions

1. Using the fact that $-\log(x^a y^b) = a\log(1/x) + b\log(1/y)$, we obtain

$$\begin{aligned} f(\theta) = -\log l(\theta) &= -\log \prod_{j=1}^{m} \sigma(\langle x_j, x\rangle) + b)^{y_j} \sigma(-\langle x_j, x\rangle) - b)^{1-y_j} \\ &= -\sum_{j=1}^{m} \Big(y_j \log\big(\sigma(\langle \tilde{x}_j, \theta\rangle)\big) + (1-y_j)\log\big(\sigma(-\langle \tilde{x}_j, \theta\rangle)\big)\Big) \\ &= \sum_{j=1}^{m} \Big(y_j \log\big(1 + e^{-\langle \tilde{x}_j, \theta\rangle}\big) + (1-y_j)\log\big(1 + e^{\langle \tilde{x}_j, \theta\rangle}\big)\Big). \end{aligned}$$

2. We call it $\tau : z \mapsto \log(1+e^z)$ and it is clearly twice differentiable with first derivative

$$\tau'(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} = \sigma(z),$$

and second derivative

$$\tau''(z) = \frac{e^z}{(1+e^z)^2} > 0.$$

Then by a classic result of calculus, having a positive second derivative, it must be convex.

3. We know that a linear combination (with positive factors) of convex functions is convex again. We then just have to show that $\log\big(1 + e^{\pm\langle \tilde{x}_j, \theta\rangle}\big)$ are convex in θ, since $\{y_j, 1-y_j\} = \{0,1\} \subset \mathbb{R}_+$ for all $j \leq m$.

Morevover, for two twice differentiable functions f and g, $(h\circ g)'' = ((h'\circ g)g')' = ((h''\circ g)(g')^2 + (h'\circ g)g''$ is positive if h', h'' and g'' are positive, that mean that $h \circ g$ is convex if g and h are convex too and h is non-decreasing.

In our case, the function τ is convex and has a positive derivative so it's increasing. $\pm\langle\tilde{x}_j,\theta\rangle$ are both linear in θ and so convex too. We can apply what we've just said and conclude that $\log\left(1+e^{\pm\langle\tilde{x}_j,\theta\rangle}\right)$ are convex in θ, implying that f is convex.

4. By construction, $f_\lambda(\theta) - \frac{\lambda}{2}\|\theta\|^2 = f(\theta)$, which is convex (Question 3.). By defintion, f_λ is then λ-strongly convex.

5. By the course, λ-strong convexity implies the existance of a unique global minimum and since f_λ is continuously differentiable, it's a point where the gradient vanishes because it is in particular a local minimum. We can conclude directly thanks to Question 4.

6. We claim that, with respect to the standard euclidean structure of $\mathbb{R}^m$,

$$\nabla f_\lambda(\theta) = \sum_{j=1}^{m} \log(1+e^{\langle\tilde{x}_j,\theta\rangle}) - Xy + \lambda\theta = X(\tau(X^\top\theta)) - y) + \lambda\theta$$

whith the matrix $X = (\tilde{x}_1 \mid \cdots \mid \tilde{x}_m) \in \mathbb{R}^{d\times m}$, and where we extend τ (and other functions like exp etc...) to matrix entries with $\tau(A) = (\tau(A_{ij}))_{ij}$. Let's proove it:

First, noticing that $\tau(z) = \log(1+e^z) = z + \log(e^{-z}+1) = z + \tau(-z)$, we rewrite f as follow :

$$\begin{aligned}
f(\theta) &= \sum_{j=1}^{m} \left(y_j \log\left(1+e^{-\langle\tilde{x}_j,\theta\rangle}\right) + (1-y_j)\log\left(1+e^{\langle\tilde{x}_j,\theta\rangle}\right)\right) \\
&= \sum_{j=1}^{m} \left(y_j\big(-\langle\tilde{x}_j,\theta\rangle + \log\left(1+e^{\langle\tilde{x}_j,\theta\rangle}\right)\big) + (1-y_j)\log\left(1+e^{\langle\tilde{x}_j,\theta\rangle}\right)\right) \\
&= \sum_{j=1}^{m} \left(\log\left(1+e^{\langle\tilde{x}_j,\theta\rangle}\right) - y_j\langle\tilde{x}_j,\theta\rangle\right) = \mathbb{1}^\top \log(1+exp(X^\top\theta)) - (Xy)^\top\theta \\
&= \mathbb{1}^\top\tau(X^\top\theta) - (Xy)^\top\theta.
\end{aligned}$$

with the vector collon $\mathbb{1} = (1,...,1)^\top \in \mathbb{R}^{d\times 1}$.

Now using basic properties of jacobians and gradients, $\nabla f_\lambda(\theta) = D_\theta(\mathbb{1}^\top\tau(X^\top\theta)) - Xy + \lambda\theta$. The first term is computed as follow :

$$(D_\theta(\mathbb{1}^\top\tau(X^\top\theta))) = \sum_{j=1}^{m} D_\theta\tau(\langle\tilde{x}_j,\theta\rangle) = \sum_{j=1}^{m} \tilde{x}_j\tau'(\langle\tilde{x}_j,\theta\rangle) = X\sigma(X^\top\theta).$$

Together, this give us $X\sigma(X^\top\theta)) - Xy + \lambda\theta = X(\sigma(X^\top\theta)) - y) + \lambda\theta$

7. First we implement τ which will be needed and which has bad computing precision properties when the argument become big. The value of $\tau(z)$ approach z but before the log take the value back down, the exp take the value very high and thus, is interpreted as infinity before the final value become too big to be actually stored. Our strategy is then to factorise and take a z out to have the sum of z and the small rest : $\tau(z) = \log(1+e^z) = \log(e^z(e^{-z}+1)) = \log(e^z) + \log(e^{-z}+1) = z + \log(e^{-z}+1)$. Like this, the e^{-z} can go as small as he want(even 0), it will make the log vanish and leave the z alone. We can now separate the cases where z is big or small and use the right formula for each :

```
1 function [y]=tau(x)
2 % Input any matrix
3 % compute for each index log(1+e^x) with ~stable computation
```

```
index = x>0;
y = x;
y(index) = x(index) + log(exp(-x(index)) + 1);
y(~index) = log(exp(x(~index)) + 1);
end
```

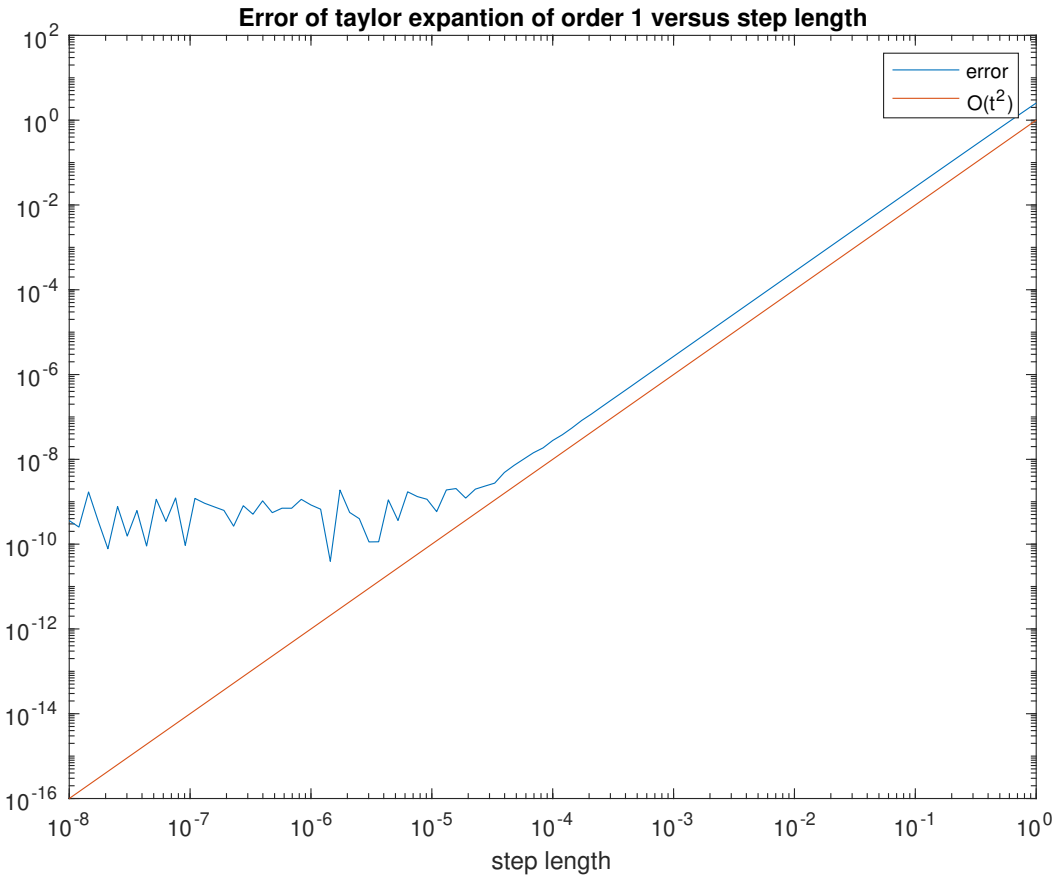
Now we can implement the function that compute f_λ and its gradient with the formulas of Question 6. We put the data in the input in order not to load it each time we call the function. Moreover the function works with multiples points stored in a matrix. In order to gain time when we only want one of the two outputs we ask for a third input to precise which output to compute.

```
function [f, g] = logistic_regression(train,theta,output)
% Input:
%
%   train : the data to train on
%   theta : A tentative parameter vector (column) for logistic regression.
%           if theta is a matrix [theta(1),...,theta(N)] it will return
%           each output concatened in a row.
%   output: 'f','g',fg', if we need f, g, or f and g respectively
%
% Output:
%
%   f: value of the logistic regression negative log-likelihood cost function
%   g: gradient of the cost function at theta, as a column vector


%  train: The training data set (see Moodle)
%          This is a structure with fields train.X and train.y
%          containing, respectively, the examples (the images) and the
%          labels to be used for training.

    X = train.X;
    y = train.y;

    lambda = 1e-4 ; % regularization parameter
    sigma = @(x) (exp(-x) + 1).^(-1);
    product = X' * theta;
    if output=='f' | output=='fg'
        f = sum(tau(product)) - y'*product + lambda/2 * sum(theta.^2);
    else
        f=[];
    end

    if output=='g' | output=='fg'
        g= X * (sigma(product) - y) + lambda * theta;
    else
        g=[];
end
```

8. Here is the code for it and the plot in Figure 1 :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 8
% Test of logistic_regression and its gradient by...
% taylor expantion of order 2...
% in a random point and direction.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
load('mnist_train_test.mat');
theta = rand(28*28+1, 1);  % random start
v  = rand(size(theta));
v = v / norm(v);           % random direction
t = logspace(-8, 0, 101);  % length of step

[f g] = logistic_regression(train, theta,'fg');
[ft, ~] = logistic_regression(train, t.*v + theta, 'f');
error = abs( ft - f - (g'*v)*t );

loglog(t, error, t, t.^2)
title('Error of taylor expantion of order 1 versus step length')
xlabel('step length')
legend('error','O(t^2)')
```



The basic idea of this is to compute a Taylor expansion of order two that involve both the gradient and different values of the function, and to see if this work. In some sense it could be a method to approximate the gradient and it's called Finite difference method. As it is a order two Taylor polynomial, the error is by definition of order two, which is what we obtain in the plot and confirm the coherence of our code. We notice that for too small values, the curve tend to be irregular and go away of the line, showing the limits and the round-off errors of the computer.

9. We compute the hessian of f_λ :

$$\nabla^2 f_\lambda(\theta) = D(\nabla f_\lambda)(\theta) = XD_\theta(\sigma(X^\top\theta)) + \lambda I = X(D\sigma)(X^\top\theta)X^\top + \lambda I = X\mathrm{diag}(\sigma'(X^\top\theta))X^\top + \lambda I$$
$$= X\mathrm{diag}(\sigma'(X^\top\theta))X^\top + \lambda I.$$

Its spectral norm can be bounded as follow, using the under-linearity and under-multiplicity :

$$\|\nabla^2 f_\lambda(\theta)\| \leq \|X\|\|\mathrm{diag}(\sigma'(X^\top\theta))\|\|X^\top\| + \lambda \leq \|X\|^2 \sup|s'| + \lambda.$$

We see that the function $|\sigma'| = \sigma'$ has a max in 0 by looking at its derivative that change of sign in 0, because

$$\sigma'(z) = \frac{e^z}{(1+e^z)^2} \text{ and } \sigma''(z) = \frac{e^z(1-e^z)}{(1+e^z)^3}.$$

The bound is then $\sup|\sigma'| = \max\sigma' = \sigma'(0) = 1/4$ Finaly, the spectral norm of the hessian is smaller than $L := 1/4\|X\|^2 + \lambda$. By a result of the course, f_λ being twice continuously differentiable, is L-strongly convex.

10. By running `1/4 * norm(X)^2` in MATLAB, we obtain $1/4\|X\|^2 \approx 198\,570$

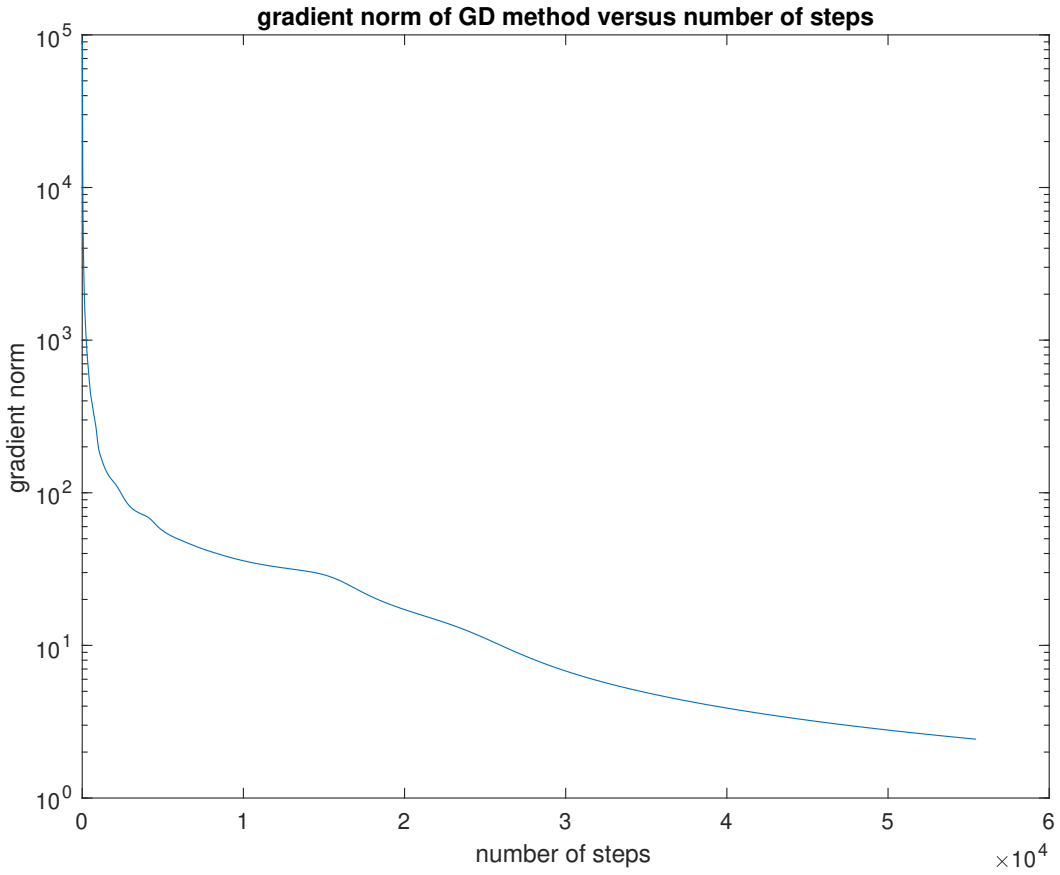
11. A classic implementation of the gradient descent :

```
function [theta, grad, time]= my_optimizer(theta0, maxtime)

    %  optimize logistic_regression by const step GD
    %
    %  INPUT  theta0 : initial point
    %         maxtime : maximal time of research
    %
    %  OUTPUT  theta : last point of search
    %          grad  : vector of succesive norm of gradient
    %          time  : vector of successive time

    %  Answer to Question 11.

    load('mnist_train_test.mat');
    f = @(theta) logistic_regression(train, theta,'g');
    L = norm(train.X)^2 /4 + 1e-4;
    alpha = 1/L;

    theta = theta0;
    grad = [];
    time = [];
    tic
    while toc < maxtime
        time = [ time toc ];
        [~, G] = f(theta);
        grad = [grad norm(G)];
        theta = theta - alpha * G;
    end
end
```

12. The code and the plot of the norm of the gradient $\|\nabla f_\lambda(\theta_k)\|$ as a function of k (log-scale on the vertical axis):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Question 12
% gradient norm of GD method versus number of steps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

theta0 = rand(785,1); % random start
maxtime = 5*60;          % maximal time

[theta, grad, ~] = my_optimizer(theta0, maxtime);
semilogy( grad )
title('gradient norm of GD method versus number of steps')
xlabel('number of steps')
ylabel('gradient norm')
```



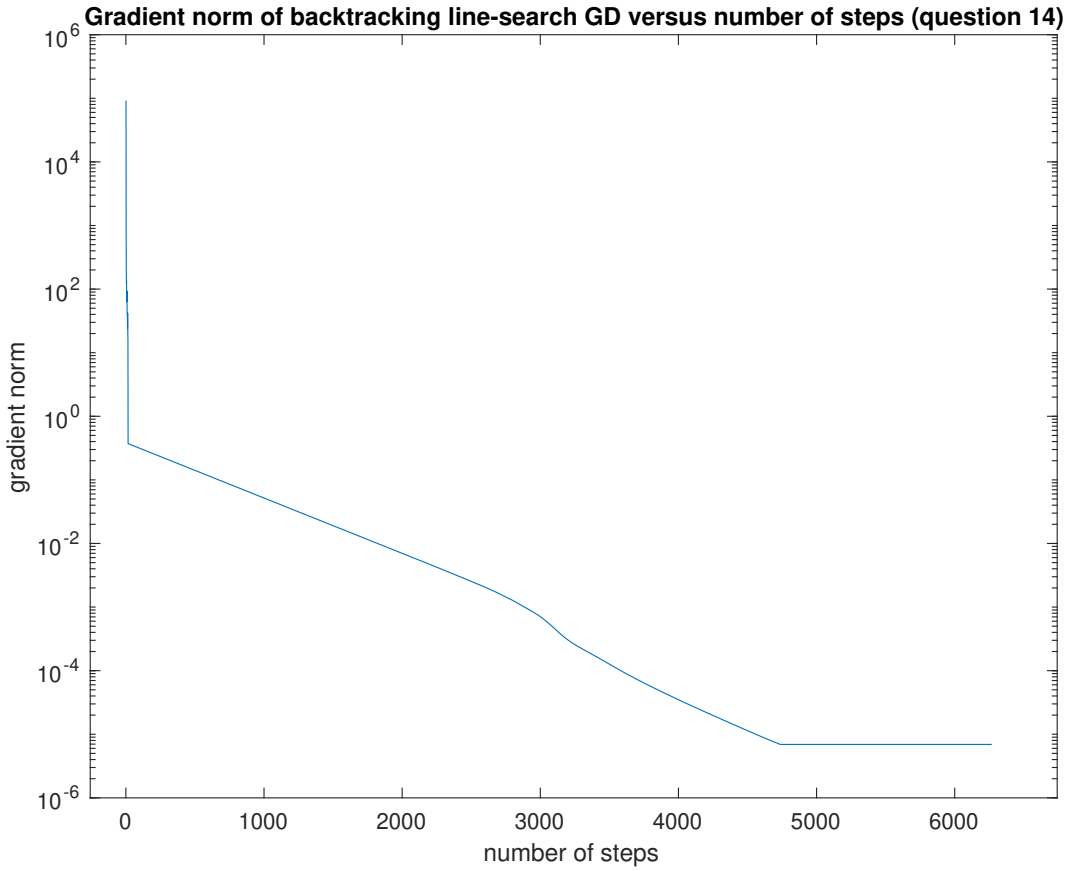
13. We have shown that f_λ is twice differentiable with Lipschitz continuous gradient, is strongly convex, and has a unique critical point with positive hessian, so by a theorem of the course, the method converge to this point. Moreover the evaluations of the function converge at least linearly to the minimum. the norm of the gradient converge at leat linearly to zero with rate $r = \sqrt{1 - 1/k}$ where k is the condition number of the hessian of the minimizer.

    In our plot the method clearly converge, very fast at first, but eventually go very slower which motivate the use of line search methods.

14. We implement the line search method. The parameters have been chosen like this : $\alpha = 20$ $\rho = 0.5$, $c = 10^{-4}$. The choice for c and ρ are usual values. We ploted the chosen values of $\rho^k\alpha$ for each step of the algoritm when we put a a big initial α and a ρ close to 1(for precision), so we can see the most frequent values chosen. All of this made us choose $\alpha = 20$, and keep $\rho = 0.5$ for short computational time.

    Here the code and the plot :

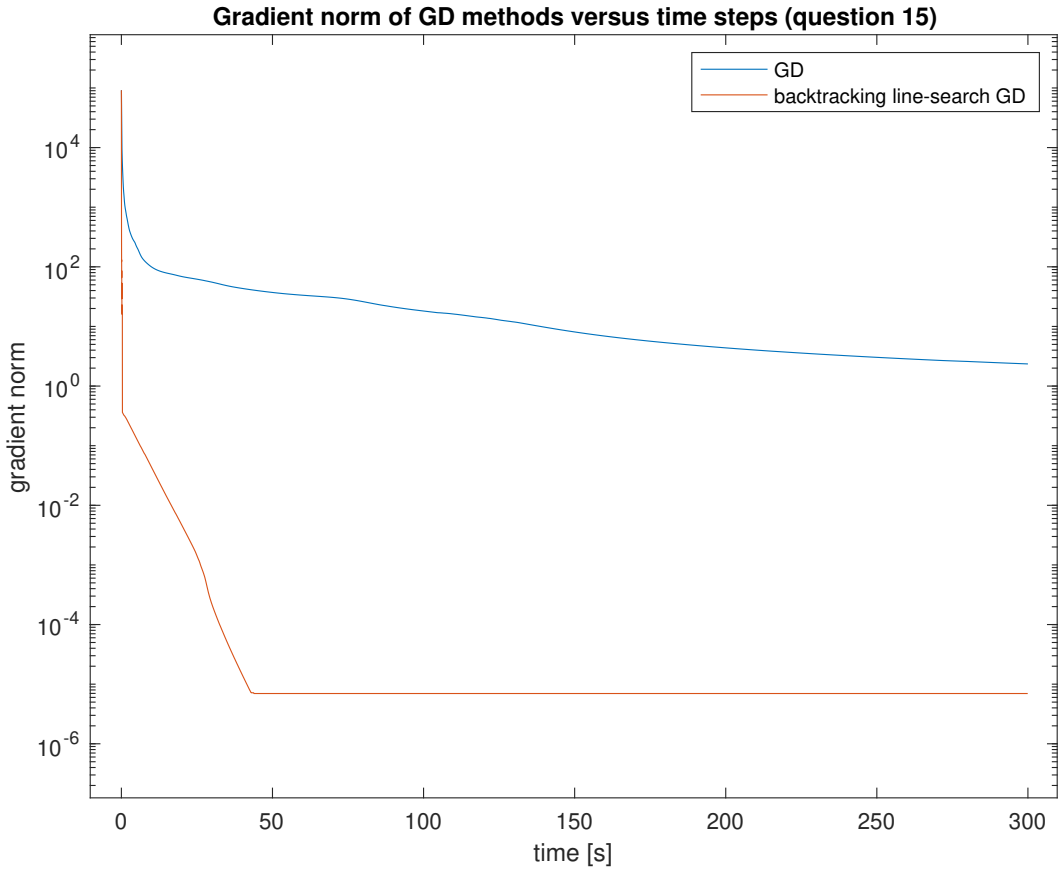
```
function [theta, grad, time]= my_optimizer2(theta0, maxtime)

    %  Optimize logistic_regression with...
    %  backtracking line-search GD method.
    %
    %  INPUT  theta0 : first point of research
    %         maxtime : maximal time of research
    %
    %  OUTPUT  theta : last point of research
    %          grad  : vector of the succesive norms of gradient
    %          time  : vector of successives times

    % Answer to Question 14.

    load('mnist_train_test.mat');
    fg= @(theta) logistic_regression(train,theta,'fg');
    f= @(theta) logistic_regression(train,theta,'f');
    g= @(theta) logistic_regression(train,theta,'g');

    alpha0=20;
    rho=0.5; % in [0.5,0.8]
    c=1e-4;

    theta=theta0;
    grad=[];
    time=[];
    tic
     while toc<maxtime
        time=[ time toc];
        [F, G]=fg(theta);
        grad=[grad norm(G)];
        alpha=alpha0;
        [Fnext, ~]=f(theta - alpha*G);
        while Fnext > F - c * alpha * G'*G
            alpha = alpha * rho;
            [Fnext, ~]=f(theta - alpha*G);
        end
        theta = theta - alpha*G;
     end
end
```



The resulting theta obtained at the end of this program is stored in a variable file (final_theta.mat) and will be used for the algorithm.

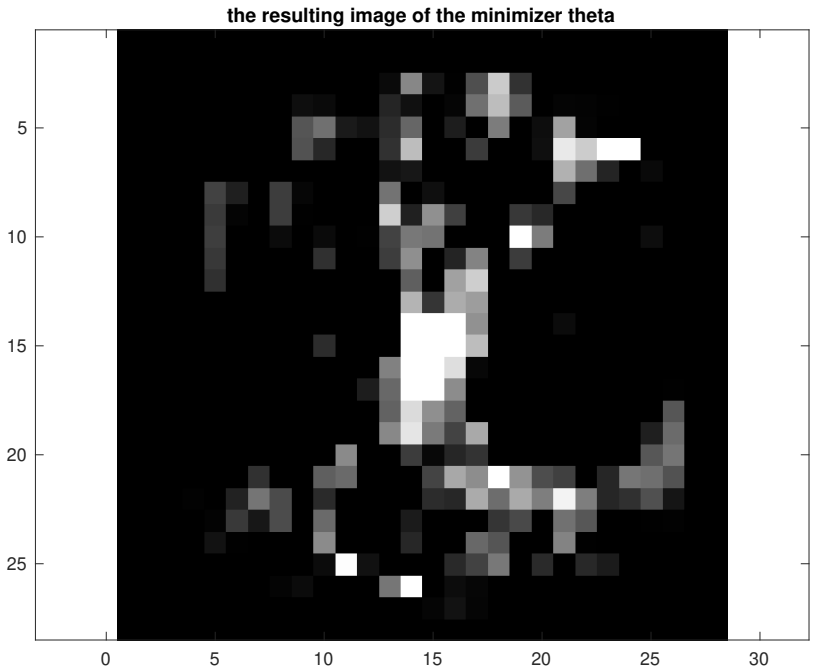
15. The code and the plot for both methods with respect to the time :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 15
% gradient norm of GD methods versus time steps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

theta0 = rand(785,1);
maxtime = 5*60;

[~, grad, time] = my_optimizer(theta0, maxtime);
[~, grad2, time2]  =my_optimizer2(theta0, maxtime);

semilogy(time, grad, time2, grad2)
title('Gradient norm of GD methods versus time steps (question 15)')
xlabel('time [s]')
ylabel('gradient norm')
legend('GD','backtracking line-search GD')
```



16. The code to show the resulting θ and the value of b:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 16
% Display of the image of the minimizer theta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('final_theta.mat')
b=final_theta(end)
showMNISTImage( final_theta )
title('the resulting image of the minimizer theta')
```




```
b = -1.1452
```

17. the final code that report the results :

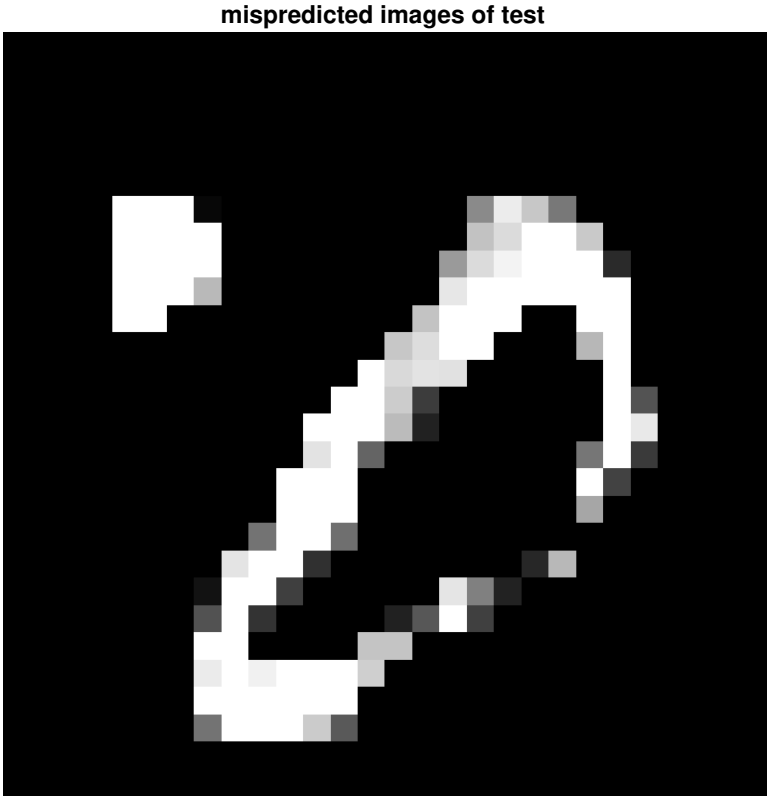
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 17
% Display the images that our theta got wrong
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('mnist_train_test.mat');
load('final_theta');

fprintf('\n\n------RESULTS ON THE TEST DATA :------\n')
results(test,final_theta);
title('mispredicted images of test')

fprintf('\n------RESULTS ON THE TRAIN DATA :------ \n')
results(train,final_theta);
title('mispredicted images of train')

function results(data,theta)
    X = data.X ;
    y = data.y' ;
    sigma = @(x) (1+exp(-x)).^(-1);
    z = sigma(theta' * X); % probability
    prediction = z > 0.5;
    error = prediction ~= y;
    m=length(y);
    % display on the command window :

    text='On the %d image(s), %d have been mispredicted, giving %.2f%% of accuracy.\n';
    fprintf( text,m, sum(error),100*(m-sum(error))/m );
```

```
29
30      text= 'The average difference beetween the probabilty and the resulting';
31      text=[text ,' prediction  is ~%.1d.\n'];
32      text= [text 'The %d mispredicted images have probability : \n'];
33      fprintf(text, sum(abs(z-prediction))/m, sum(error))
34      fails_probabilities=z(error)
35
36      showMNISTImages_many( X(:,error) )
37  end
```

With the following output in the command window :

```
1               ------RESULTS ON THE TEST DATA :------
2   On the 2115 image(s), 1 have been mispredicted, giving 99.95% of accuracy.
3   The average difference between the probability and the resulting prediction  is ~1.5e-04.
4   The 1 mispredicted images have probability :
5
6   fails_probabilities =
7
8       0.9984
9
10
11  ------RESULTS ON THE TRAIN DATA :------
12  On the 12665 image(s), 0 have been mispredicted, giving 100.00% of accuracy.
13  The average difference between the probability and the resulting prediction  is ~8.9e-08.
14  The 0 mispredicted images have probability :
15
16  fails_probabilities =
17
18    1 0  empty double row vector
```

In conclusion, the algorithm has been infallible on the train data, and has made only one mistake on the test data. Here is the mispredicted image :



We conclude that our algorithm is coherent and allows to actually read images with a certain reliability as it worked well enough on the completely new data, with respect to our overarching goal.