MATH-329 Nonlinear optimization Homework 2: Trust-region and Gaussian mixture models

Anya-Aurore Mauron & Benoît Müller

October 4, 2023

Remarks

Each file named questionX.m is the main script file of the question number X. Some of them are live scripts, run each sections separately as needed.

The other files are data or functions named after their utility. Running the main scripts of the questions give the outputs detailed in this document.

Answers

1. We directly compute

$$f(\mu) = -\log\big(\prod_{n=1}^{N}\phi(x_n,\mu,\Sigma)\big) = \sum_{n=1}^{N}\log\left(\frac{1}{\sigma^d(2\pi)^{d/2}}\exp\left(\frac{\|x_n-\mu\|^2}{2\sigma^2}\right)\right)$$
$$= \sum_{n=1}^{N}\left(\log(\sigma^d(2\pi)^{d/2}) + \frac{\|x_n-\mu\|^2}{2\sigma^2}\right) = dN\log(\sigma\sqrt{2\pi}) + \frac{1}{2\sigma^2}\sum_{n=1}^{N}\|x_n-\mu\|^2.$$

2. The function f is a quadratic function (C^∞) so the convexity is entirely determined by its hessian. The gradient is

$$\nabla f(\mu) = \nabla_\mu\big(dN\log(\sigma\sqrt{2\pi}) + \frac{1}{2\sigma^2}\sum_{n=1}^{N}\|x_n-\mu\|^2\big) = \frac{1}{2\sigma^2}\sum_{n=1}^{N}\nabla_\mu\|x_n-\mu\|^2$$
$$= \frac{1}{2\sigma^2}\sum_{n=1}^{N}2(\mu-x_n) = \frac{N}{\sigma^2}\Big(\mu - \frac{1}{N}\sum_{n=1}^{N}x_n\Big),$$

and the hessian is then $\frac{N}{\sigma^2}I_d$. Since N/σ^2 is strictly positive, the hessian is obviously strictly positive definite and the function is strictly convex. For a positive scalar l, if we substract $l/2\|\mu\|^2 = l/2$ from $f(\mu)$ and still want it to be convex, we must have $N/\sigma^2 - l \geq 0$ because it is again a quadratic function. This give us $l \leq N/\sigma^2$ as a necessary and sufficient condition for f to be l-strongly convex.

3. The only critical point is the mean $\frac{1}{N}\sum_n x_n$ and it's the global minimum, because f is N/σ^2-strongly convex. That is quiet what we could have intuitively expected, μ being the mean of the random variables x_n.

4. We consider the particular case $N = 1$, $K = 2$, $d = 1$ and plot the function for some data point x_1 and parameters σ, π_1, π_2 chosen randomly. We obtain a non-convex plot, as we can see on the four non convex edges of Figure [1].

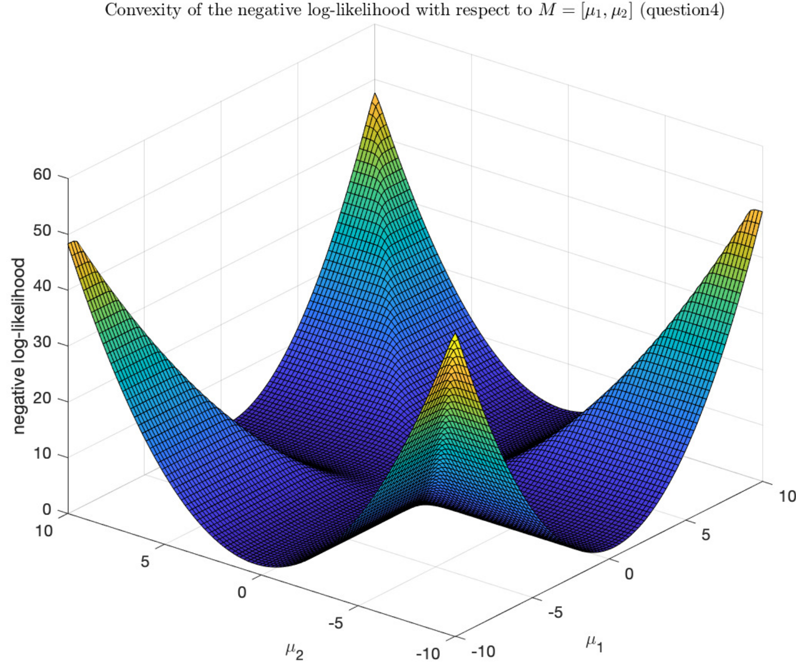

Figure 1: The function f with its four non convex edges

However, if we zoom enough we can see a possibly local convex minimum in Figure [2].

5. We develop the formula of f:

$$
\begin{aligned}
f(M) &= -\log\Big(\prod_{n=1}^{N}\sum_{k=1}^{K}\pi_k\phi(x_n,\mu_k,\Sigma)\Big) = -\sum_{n=1}^{N}\log\Big(\sum_{k=1}^{K}\pi_k\phi(x_n,\mu_k,\Sigma)\Big) \\
&= -\sum_{n=1}^{N}\log\left(\sum_{k=1}^{K}\frac{\pi_k}{\sigma^d(2\pi)^{d/2}}\exp\Big(-\frac{\|x_n-\mu_k\|^2}{2\sigma^2}\Big)\right) \\
&= Nd\log(\sigma\sqrt{2\pi}) - \sum_{n=1}^{N}\log\left(\sum_{k=1}^{K}\pi_k\exp\Big(-\frac{\|x_n-\mu_k\|^2}{2\sigma^2}\Big)\right) \qquad (1)
\end{aligned}
$$

Here we see that the expression (1) look like the log-sum-exp function, i.e. $LSE(\alpha) = \log(\sum_j e^{\alpha_j})$. It is a smooth approximation of $\max_j(\alpha_j)$. Moreover, the expression can lead to overflow because it involves computing to small exponentials of possibly smalls numbers before the log turn them back to actually stockable values.

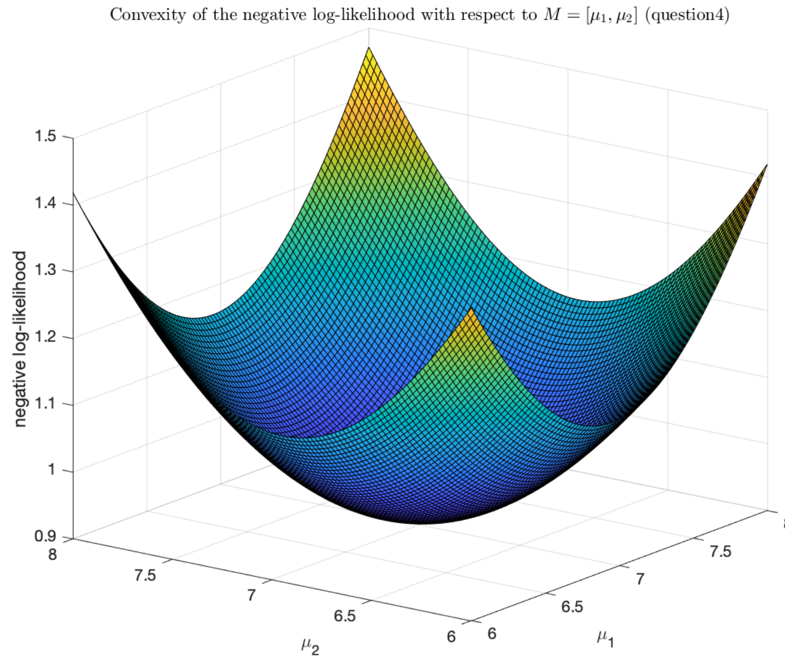

Figure 2: The function f with a zoom on a possible local convex minimum

The resulting computation done by the machine would be "$\log(\sum e^{\text{small}}) = \log(0) = -\infty$", while actually "$\log(\sum e^{\text{small}}) > \log(e^{\text{small}}) = \text{small}$". We deal with this by rearranging the formula:
We set $\alpha_k^n = -\|x_n - \mu_k\|^2/(2\sigma^2) + \log(\pi_k)$, and f is then $Nd\log(\sigma\sqrt{2\pi}) - \sum_n LSE(\alpha_n)$. A best way to write LSE for numeric stability, is to factorise $\exp(\bar{\alpha}_n)$to the sum, with $\bar{\alpha}_n = \max_k \alpha_n^k$:

$$LSE(\alpha_n) = \log(\sum_k e^{\bar{\alpha}_n} e^{\alpha_n^k - \bar{\alpha}_n}) = \bar{\alpha}_n + \log(\sum_k e^{\alpha_n^k - \bar{\alpha}^n})$$

Like this we can accurately compute the main small part of the sum, as the exp vanishes with the log. In the sum remain the exponentials of the differences to the maxima which sure are less likely to become to smalls if the values are near from each others. In all cases, there will always be a $e^{\alpha_n^k - \bar{\alpha}^n} = e^{\alpha_n^k - \alpha_n^k} = 1$ term in the sum, assuring that even when values are far from each other and zeros comes in the sum, the argument form the log stay near over 1 and the log will likely vanish, leaving the max as a approximation of the result, which is good.

The formula is then

$$\begin{aligned} f(M) &= Nd\log(\sigma\sqrt{2\pi}) - \sum_n LSE(\alpha_n) = Nd\log(\sigma\sqrt{2\pi}) - \sum_n \left(\bar{\alpha}_n + \log(\sum_k e^{\alpha_n^k - \bar{\alpha}^n})\right) \\ &= Nd\log(\sigma\sqrt{2\pi}) - \sum_n \left(\bar{\alpha}_n + \log(\sum_k e^{\alpha_n^k - \bar{\alpha}^n})\right). \end{aligned}$$

To implement it and avoid slow loops, we use multidimensional arrays. The idea is to have a dimension for the k-sum, a dimension for the n-sum and a last for the dimension d of the space.

With X of dimension $1 \times N \times d$ and M of dimension $K \times 1 \times d$, `X-M` is then the 3D matrix with $(X-M)_{kn\delta} = (x_n - \mu_k)_\delta$, which permit us to compute $\|x_n - \mu_k\|^2$ with `sum((X-M).^2,3)` We use this to compute α with `-sum( (X - M).^2 , 3 ) / (2*sigma^2) + log(Pi')` and continue computing the elements with array operations paying attention to dimensions compatibility.

The detailed code with arrays sizes is the following :

```
1  function [f] = log_likelihood(M, X, sigma, Pi)
2
3  % Compute the negative log-likelihood function
4  % of the data points X=[x1,...,xN] sampled randomly
5  % with probability repartition Pi=[pi1,...,piK]
6  % from K clusters of multivrariable normal laws with means M=[mu1,...,muK]
7  % and all with covariance sigma*I.
8  % Matrices shapes: d for first dimension and N or K for second.
9
10 % Concern Questions 4-5
11
12     [d, N] = size(X);
13     X = permute(X,[3 2 1]);  % 1*N*d
14     M = permute(M, [2 3 1]); % K*1*d
15
16     % alpha(k,n) = ||x_n - mu_k||^2 / (2*sigma^2) + log(Pi_k) :
17     alpha = -sum( (X - M).^2 , 3 ) / (2*sigma^2) + log(Pi'); % K*N
18     beta = max(alpha); % 1*N
19     LSE =  beta + log(sum(exp(alpha - beta))); %1*N
20     f = d*N*log(sigma*sqrt(2*pi)) - sum(LSE);
21 end
```

6. Let's rewrite the value of $\phi_n(\mu)$, using a new scalar function ψ:

$$\phi_n(\mu) = \frac{1}{\sigma^d (2\pi)^{d/2}} \exp\Big(-\frac{\|x_n-\mu\|^2}{2\sigma^2}\Big) =: \psi(\|x_n-\mu\|^2).$$

We can then compute its gradient with the composition formula:

$$\nabla\phi_n(\mu) = \psi'(\|x_n-\mu\|^2)\nabla_\mu\|x_n-\mu\|^2 = -\frac{1}{2\sigma^2}\psi(\|x_n-\mu\|^2)(-2(x_n-\mu)) = \frac{\phi_n(\mu)}{\sigma^2}(x_n-\mu).$$

7. Let's compute the value of $\nabla_{\mu_j} f(M)$ using the linearity of the gradient, the composition formula with respect to the log, and the previous question:

$$\begin{aligned}
\nabla_{\mu_j} f(M) &= -\nabla_{\mu_j}\sum_{n=1}^N \log\Big(\sum_{k=1}^K \pi_k\phi_n(\mu_k)\Big) = -\sum_{n=1}^N \nabla_{\mu_j}\log\Big(\sum_{k=1}^K \pi_k\phi_n(\mu_k)\Big)\\
&= -\sum_{n=1}^N \log'\Big(\sum_{k=1}^K \pi_k\phi_n(\mu_k)\Big)\nabla_{\mu_j}\sum_{k=1}^K \pi_k\phi_n(\mu_k)\\
&= -\sum_{n=1}^N \Big(\sum_{k=1}^K \pi_k\phi_n(\mu_k)\Big)^{-1}\sum_{k=1}^K \pi_k\nabla_{\mu_j}\phi_n(\mu_k)\\
&= -\sum_{n=1}^N \Big(\sum_{k=1}^K \pi_k\phi_n(\mu_k)\Big)^{-1}\pi_j\frac{\phi_n(\mu_j)}{\sigma^2}(x_n-\mu_j) = \frac{\pi_j}{\sigma^2}\sum_{n=1}^N \gamma_n\phi_n(\mu_j)(\mu_j-x_n)
\end{aligned}$$

8. As a result of the previous question, the gradient of f is then $\nabla f(M) = (\nabla_{\mu_1} f(M), \dots, \nabla_{\mu_K} f(M))$

9. We rewrite the formula of $\nabla f(M)$ obtained in question 7. :

$$\nabla_{\mu_j} f(M) = \frac{\pi_j}{\sigma^2} \sum_{n=1}^{N} \gamma_n \phi_n(\mu_j)(\mu_j - x_n)$$

Now we see that γ_n and $\phi_n(\mu_j)$ have a factor $\sigma^d(2\pi)^{d/2}$ that cancel out. Moreover they both contain exponentials that can be very small, so we decide for computational stability to distribute the exp of $\phi_n(\mu_j)$ in the sum of γ_n and put their exponents together :

$$\begin{aligned}\nabla_{\mu_j} f(M) &= \frac{\pi_j}{\sigma^2} \sum_{n=1}^{N} \gamma_n \phi_n(\mu_j)(\mu_j - x_n) = \frac{\pi_j}{\sigma^2} \sum_{n=1}^{N} \frac{1}{\sum_k \pi_k \frac{\phi_n(\mu_k)}{\phi_n(\mu_j)}}(\mu_j - x_n) \\ &= \frac{\pi_j}{\sigma^2} \sum_{n=1}^{N} \frac{1}{\sum_k \pi_k \exp\left(\frac{\|x_n-\mu_j\|^2 - \|x_n-\mu_k\|^2}{2\sigma^2}\right)}(\mu_j - x_n) = \frac{\pi_j}{\sigma^2} \sum_{n=1}^{N} \left(\sum_{k=1}^{K} \pi_k e^{(\beta_n^j - \beta_n^k)}\right)^{-1}(\mu_j - x_n)\end{aligned}$$

with $\beta_n^k = \|x_n - \mu_k\|^2/(2\sigma^2)$. Like this we compute the differences before doing the exponentials which is better. There is no danger to divide by a infinitesimal k-sum as the term $\pi_j e^{(\beta_n^j - \beta_n^j)} = \pi_j > 0$ is supposed to be not as small.

Again, we use multidimensional array to compute the gradient of f. The idea is to have a dimension for the dimension d of the space or the change of column of the gradient along j; a dimension for the k-sum; and a dimension for the n-sum. We compute β similarly as question question 5., by reshaping X and M judicially. We set then a 3D matrix B with $B_{kjn} = \beta_n^j - \beta_n^k$ using reshaped versions of β. We can then write the k-sum as a particular matrix product of Pi and e^B, and so on . . . The choices of matrix shapes are detailed in the following code of `grad_log_likelihood.m` :

```
1  function [g] = grad_log_likelihood(M, X, sigma, Pi)
2
3  % Compute the gradientof the negative log-likelihood function
4  % of the data points X=[x1,...,xN] sampled randomly
5  % with probability repartition Pi=[pi1,...,piK]
6  % from K clusters of multivrariable normal laws with means M=[mu1,...,muK]
7  % and all with covariance sigma*I.
8  % Matrices shapes: d for first dimension and N or K for second.
9
10 % Concern Question 9
11
12     [d, N] = size(X) ;
13     X = permute(X,[1, 3, 2]) ; % d*1*N
14     % M is d*K
15     diff = M-X ; % d*K*N
16     alpha = sum(diff.^2) / (2*sigma^2) ; % 1*K*N
17     A = alpha - permute(alpha, [2,1,3]) ; %K*K*N
18     g = Pi .* sum( (M-X) ./ sum(Pi'.*exp(A)) , 3) / sigma^2 ; % d*K
19
20 end
```

```
1  function [f , g] = funct_grad(M, X, sigma, Pi)
2
3  % Compute the negative log-likelihood function and its gradient
4  % of the data points X=[x1,...,xN] sampled randomly
5  % with probability repartition Pi=[pi1,...,piK]
6  % from K clusters of multivrariable normal laws with means M=[mu1,...,muK]
```

```
% and all with covariance sigma*I.
% Matrices shapes: d for first dimension and N or K for second.

% Concern Question 9

    % for clarity, we follow the current matrix dimensions by mentioning
    % them as commentaries like this: d1*d2*...*dj .

    % The gradient g :
    [d, N] = size(X) ;
    X = permute(X,[1 3 2]) ; % d*1*N
    % M is d*K
    diff = M-X ; % d*K*N
    alpha = sum(diff.^2) / (2*sigma^2) ; % 1*K*N
    A = alpha - permute(alpha,[2,1,3]) ; %K*K*N
    g = Pi .* sum( (M-X) ./ sum(Pi'.*exp(A)) , 3) / sigma^2 ; % d*K

    X = permute(X,[3 2 1]);  % 1*N*d
    M = permute(M, [2 3 1]); % K*1*d

    % The function f :
    alpha = -squeeze(alpha) + log(Pi'); % K*N
    beta = max(alpha); % 1*N
    LSE =  beta + log(sum(exp(alpha - beta))); %1*N
    f = d*N*log(sigma*sqrt(2*pi)) - sum(LSE);
end
```

10. We write a classic backtracking line-search gradient descent algorithm of f :

```
function [M, grad, time,obj_value]= BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c)

    % Backtracking Line-Search Gradient Descent of
    % the negative log-likelihood function "log_likelihood.m"
    %
    %  INPUT maxtime : maximal time of research
    %        M0      : initial point
    %        X       : data
    %        pis     : probabiliy repartition of clusters
    %        sigma   : matrix of covariance = sigma*I
    %        alpha0  : initial time step
    %        rho     : factor of discrease of the current alpha
    %        c       : exigence of approvment
    %
    %  OUTPUT  M         : last point of research
    %          grad      : vector of the succesive norms of gradient
    %          time      : vector of successives times
    %          obj_value : objective value of M


    X = X';
    Pi = pis';

    f  = @(M)      log_likelihood(M, X, sigma, Pi);
    fg = @(M)          funct_grad(M, X, sigma, Pi);


    M=M0;
    grad=[];
    time=[];
    tic
```

```
32      while toc < maxtime
33         time = [ time, toc];
34         [F, G] = fg(M);
35         grad = [grad norm(G,'fro')];
36         alpha = alpha0;
37         Fnext=f(M - alpha*G);
38         while Fnext > F - c * alpha * sum(G.*G,'all')
39             alpha = alpha * rho;
40             Fnext = f(M - alpha*G);
41         end
42         M = M - alpha*G;
43      end
44      obj_value=f(M);
45  end
```

11. Here is the function `display_data_means.m` to plot the data points and the K cluster means (μ_k):

```
1  function display_data_means(X, M)
2  X=X';
3  scatter(X(1,:),X(2,:),'MarkerEdgeAlpha',0.1) %for transparency
4  hold on
5  plot(M(1,:),M(2,:),'ro')
6  title('Data points and K clusters means')
7  end
```

Then running the line-search algorithm on `data-toy.mat` (see the implementation in `question11.m`), we obtain the graph in Figure [3] after a few trials.

```
1  clc
2  clear
3  close
4
5
6  load('data-toy.mat');
7  maxtime=4;
8  alpha0=1e-2;
9  rho=0.5; % in [0.5,0.8]
10 c=1e-4;
11 M0 =  mean(X',2) .* ones(1,K) + 1e-4*rand(d,K);
12
13 [M_1,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
14 display_data_means(X,M_1)
```

Figure 3: Plot of the data points in blue and the K cluster means (μ_k) in red.

The four clusters in red are in the middle of the four groups of data, in blue. We can conclude that our algorithm indeed outputs reasonable results.

12. First, we run the algorithm with different initializations M_0 and observe how the objective value changes (see the first plot in Figure [4]). Then, with a fixed random initialization, we try different $\bar{\alpha} \in [10^{-2}, 1]$, $\rho \in [0.5, 0.8]$ and $c \in [10^{-5}, 10^{-3}]$. See the implementation below.

    We observe by Figure [4] that starting with a random initialization can vary the results. The value of $\bar{\alpha}$ has an even more noticeable impact on the objective value. On the other hand, we note that varying ρ and c in their respective intervals does not change the objective value.

    The values ρ and c don't have a big impact on the objective values because of their respective roles; ρ is the the decreasing speed of α-candidates, and c is an "accuracy" constant on the estimation of the progress that each step produces.

    On the other hand, $\bar{\alpha}$ effects the final result more because its role is to give the biggest step one can take in the direction of the gradient.

    We choose $\bar{\alpha} = 10^{-2}$, $\rho = 0.5$ and $c = 10^{-4}$ for the next questions, according to the results above and the initial value given in the homework for ρ and c.

    We can explain the unstable behavior of the objective value by varying the initial parameter by the fact that the gradient descent algorithm depends a lot on where the algorithm starts. Also, we do not have the theory that gradient descent actually converges in this case, since most results we established were for convex and strongly convex functions.

    Note also that f has 2 axes where the function seems to be quite flat. If the starting point lies on one of them, the algorithm might be trapped in a very slow procedure. Moreover we observe on the plot that the function f is bounded below, but it still does not guarantee that f has a minimum.

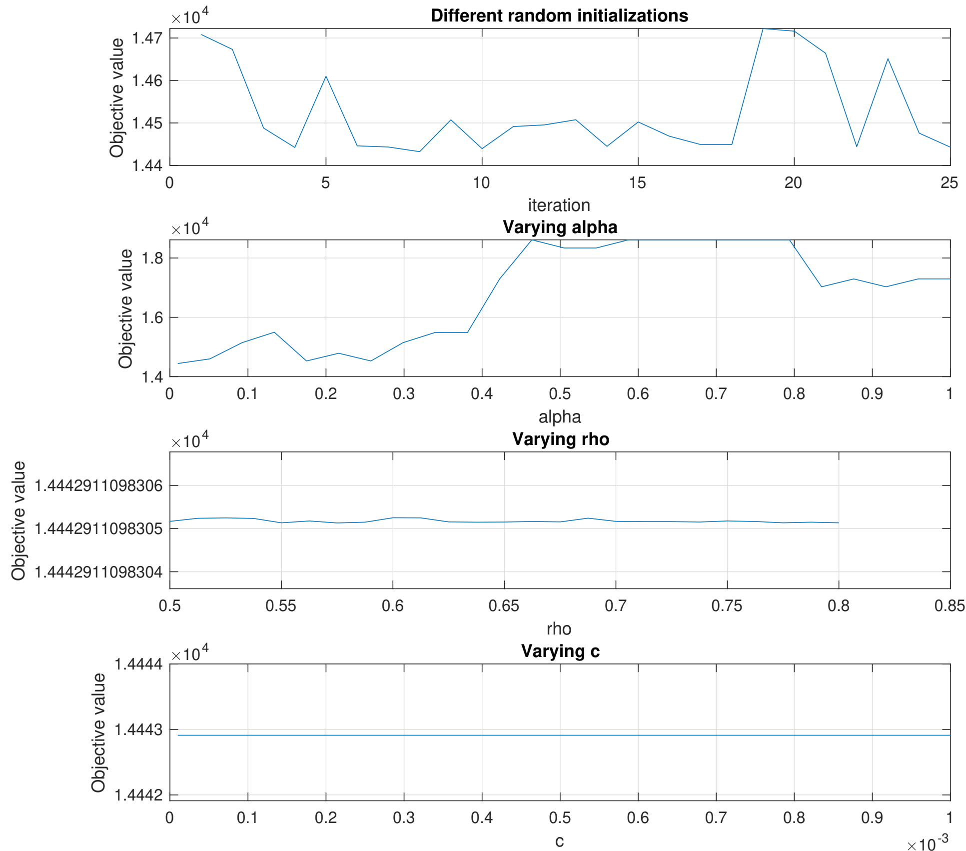

Figure 4: Plots of the objective value with respect to different parameters. When not varying, $\bar{\alpha} = 10^{-2}$, $\rho = 0.5$ and $c = 10^{-4}$.

We observe then that the backtracking line search method does converge on a stationary point, by the various trials we ran, see next question. We see on the graph that the stationary points are the set of the 2 axes, so we can conclude that BLSGD must reach a point belonging to it.

```
clc
clear
close


load('data.mat');
maxtime=4;

N=25;
I=1:1:N;
Alphas=linspace(1e-2,1,N);
Rhos=linspace(0.5,0.8,N);
```

```
13 Cs=linspace(1e-5,1e-3,N);
14 
15 %% Running with different random initializations
16 %We run the algorithm several times, and see if the results vary.
17 alpha0=1e-2;
18 rho=0.5;
19 c=1e-4;
20 obj_values_rand_init=zeros(1,N);
21 for i=I
22     M0 =  mean(X',2) .* ones(1,K) + 1e-4*rand(d,K);
23     [M,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
24     obj_values_rand_init(i)=obj_value;
25 end
26 
27 %% Varying alpha0
28 obj_values_Alphas=zeros(1,N);
29 rho=0.5;
30 c=1e-4;
31 
32 for i=I
33 alpha0=Alphas(i);
34 [M,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
35 obj_values_Alphas(i)=obj_value;
36 end
37 
38 %% Varying rho
39 obj_values_Rhos=zeros(1,N);
40 alpha0=1e-2;
41 c=1e-4;
42 
43 for i=I
44 rho=Rhos(i);
45 [M,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
46 obj_values_Rhos(i)=obj_value;
47 end
48 
49 %% Varying c
50 obj_values_Cs=zeros(1,N);
51 alpha0=1e-2;
52 rho=0.5;
53 
54 for i=I
55 c=Cs(i);
56 [M,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
57 obj_values_Cs(i)=obj_value;
58 end
59 
60 %% Comparing results
61 subplot(4,1,1)
62 plot(I,obj_values_rand_init)
63 title('Different random initializations');
64 grid on
65 xlabel('iteration');
66 ylabel('Objective value');
67 
68 subplot(4,1,2)
69 plot(Alphas,obj_values_Alphas)
70 title('Varying alpha');
71 grid on
72 xlabel('alpha');
73 ylabel('Objective value');
74 
```

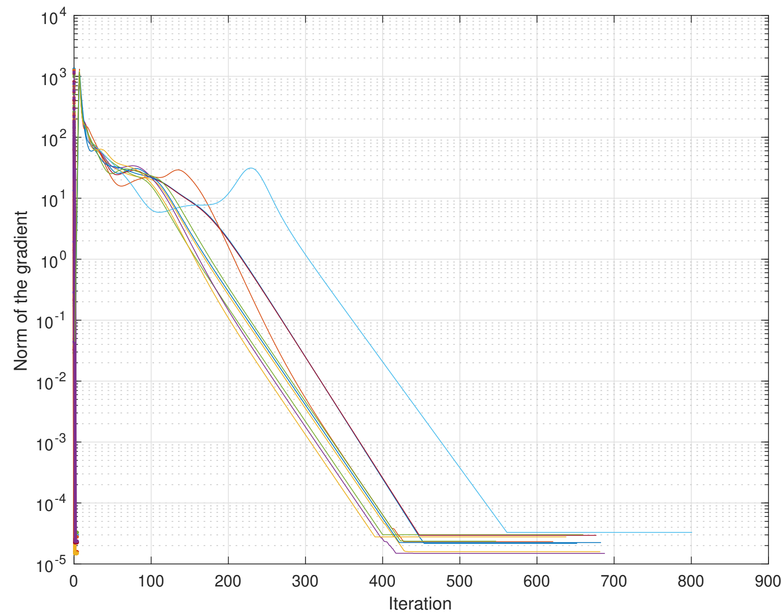

Figure 5: Graph for question 13. The norm of the gradient of the 10 trials done with different initialisation with the backtracking line-search method.

```
75  subplot(4,1,3)
76  plot(Rhos,obj_values_Rhos)
77  title('Varying rho');
78  grid on
79  xlabel('rho');
80  ylabel('Objective value');
81
82  subplot(4,1,4)
83  plot(Cs,obj_values_Cs);
84  title('Varying c');
85  grid on
86  xlabel('c');
87  ylabel('Objective value');
```

13. We run the backtracking line search method a few times to observe several trials, each trial having a different initialization. We decided to run the algorithm 10 times, see the implementation in `question13.m` below. On Figure [5], we notice that the gradient reaches an important norm of of more than 100 in the beginning of the method. Then we observe that the norm of the gradient rapidly converges to zero. After about 400 iterations, it is already below 10^{-4} for all trials.

    The gradient converges well to zero. Hence we have chosen good initial values at the previous point.

```
1  clc
2  clear
3  close
4
5
```

```
6  load('data.mat');
7  maxtime=4;
8  alpha0=1e-2;
9  rho=0.5; % in [0.5,0.8]
10 c=1e-4;
11
12 for i=1:10
13     M0 =  mean(X',2) .* ones(1,K) + 1e-4*rand(d,K);
14     [M,grad,time,obj_value]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
15     iterations=1:1:size(grad,2);
16     semilogy(iterations,grad)
17     hold on
18 end
19
20 xlabel('Iteration'); ylabel('Norm of the gradient');
21 grid on
```

14. We use the definition of the gradient of f obtained in question 7.:

$$\nabla_{\mu_i} f(M) = \frac{\pi_i}{\sigma^2}\sum_{j=1}^{N}\gamma_j\phi_j(\mu_i)(\mu_i - x_j).$$

In order to calculate the hessian of f evaluated at V, we use the definition:

$$\nabla^2 f(M)[V] = \lim_{t\to 0}\frac{\nabla f(M+tV) - \nabla f(M)}{t}$$

with $M = [\mu_1, \cdots, \mu_K]$ and $V = [v_1, \cdots, v_K] \in \mathbb{R}^{d\times K}$.
Let's calculate $\nabla^2 f(M)[V]$ for each of its component. For $i = 1, ..., K$:

$$\begin{aligned}\nabla_{\mu_i} f(M+tV) - \nabla_{\mu_i} f(M) &= \frac{\pi_i}{\sigma^2}\sum_{j=1}^{N}\gamma_j^{M+tV}\phi_j(\mu_i+tv_i)(\mu_i+tv_i-x_j) - \gamma_j^M\phi_j(\mu_i)(\mu_i-x_j)\\ &= \frac{\pi_i}{\sigma^2}\sum_{j=1}^{N}(\gamma_j^{M+tV}-\gamma_j^M)\phi_j(\mu_i+tv_i)(\mu_i+tv_i-x_j)\\ &+ \gamma_j^M\big(\phi_j(\mu_i+tv_i)(\mu_i+tv_i-x_j) - \phi_j(\mu_i)(\mu_i-x_j)\big)\end{aligned}$$

with $\gamma_j^M = \frac{1}{\sum_{k=1}^K \pi_k\phi_j(\mu_k)}$. Let's call the first part of the sum

$$A_i^j(t) := (\gamma_j^{M+tV}-\gamma_j^M)\phi_j(\mu_i+tv_i)(\mu_i+tv_i-x_j)$$

and the second part

$$B_i^j(t) := \gamma_j^M\big(\phi_j(\mu_i+tv_i)(\mu_i+tv_i-x_j) - \phi_j(\mu_i)(\mu_i-x_j)\big).$$

Let's look at the first part $A_i^j(t)$. We can express the value γ_j^{M+tV}:

$$\begin{aligned}\gamma_j^{M+tV} &= \frac{1}{\sum_{k=1}^K \pi_k\phi_j(\mu_k+tv_k)} = \frac{1}{\sum_{k=1}^K \pi_k\phi_j(\mu_k) + \sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top tv_k + O(t^2)}\\ &= \frac{1}{\sum_{k=1}^K \pi_k\phi_j(\mu_k)} - \frac{t\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k + O(t^2)}{\big(\sum_{k=1}^K \pi_k\phi_j(\mu_k)\big)^2} + O(t^2)\\ &= \gamma_j^M - \gamma_j^2\big(t\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k + O(t^2)\big) + O(t^2)\end{aligned}$$

by two expansions of Taylor of order 1. Indeed, the Taylor expansion at $t = 0$ can be generically expressed as :

$$f(a + tb) = f(a) + f'(a)tb + O(t^2)$$

The first expansion is applied on $\sum_{k=1}^K \pi_k\phi_j(\mu_k+tv_k)$ at the second equality, with $f(x) = \sum_{k=1}^K \pi_k\phi_j(x)$, $a = \mu_k$ and $b = v_k$.

The second expansion of Taylor is applied at the third equality, with $f = \frac{1}{x}$, $a = \sum_{k=1}^K \pi_k\phi_j(\mu_k)$ and $b = \sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k + O(t)$.

Now, taking the limit:

$$\begin{aligned}
\lim_{t\to 0}\frac{A_i^j(t)}{t} &= \lim_{t\to 0}\frac{\gamma_j^{M+tV} - \gamma_j^M}{t}\phi_j(\mu_i + tv_i)(\mu_i + tv_i - x_j)\\
&= \lim_{t\to 0}\frac{-\gamma_j^2\big(t\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k + O(t^2)\big) + O(t^2)}{t}\phi_j(\mu_i + tv_i)(\mu_i + tv_i - x_j)\\
&= -\gamma_j^2\big(\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k\big)\phi_j(\mu_i)(\mu_i - x_j)\\
&= \gamma_j^2\big(\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k\big)\phi_j(\mu_i)(x_j - \mu_i)
\end{aligned}$$

Let's look at $B_i^j(t)$ now:

$$\begin{aligned}
B_i^j(t) &= \gamma_j^M\big[\big(\phi_j(\mu_i + tv_i) - \phi_j(\mu_i)\big)(\mu_i + tv_i - x_j) + \phi_j(\mu_i)\big((\mu_i + tv_i - x_j) - (\mu_i - x_j)\big)\big]\\
&= \gamma_j^M\big[\big(\phi_j(\mu_i + tv_i) - \phi_j(\mu_i)\big)(\mu_i + tv_i - x_j) + \phi_j(\mu_i)tv_i\big]
\end{aligned}$$

Then, taking the limit:

$$\begin{aligned}
\lim_{t\to 0}\frac{B_i^j(t)}{t} &= \lim_{t\to 0}\gamma_j^M\big[\frac{\phi_j(\mu_i + tv_i) - \phi_j(\mu_i)}{t}(\mu_i + tv_i - x_j) + \frac{\phi_j(\mu_i)tv_i}{t}\big]\\
&= \gamma_j^M\big[\langle\nabla\phi_j(\mu_i), v_i\rangle(\mu_i - x_j) + \phi_j(\mu_i)v_i\big]\\
&= \gamma_j\big[\frac{\phi_j(\mu_i)}{\sigma^2}(x_j - \mu_i)^\top v_i(\mu_i - x_j) + \phi_j(\mu_i)v_i\big]\\
&= \gamma_j\phi_j(\mu_i)\big[\frac{1}{\sigma^2}(x_j - \mu_i)^\top v_i(\mu_i - x_j) + v_i\big]
\end{aligned}$$

The second equality comes from the definition of the gradient. The third one comes from the definition of the scalar product and from the result found in question 6.

From now we will write $\gamma_j^M = \gamma_j$ as defined in the homework. Hence, we obtain, for $i = 1, ..., K$:

$$\begin{aligned}
(\nabla^2 f(M)[V])_i &= \lim_{t\to 0}\frac{\nabla_{\mu_i}f(M + tV) - \nabla_{\mu_i}f(M)}{t} = \frac{\pi_i}{\sigma^2}\sum_{j=1}^N \lim_{t\to 0}\frac{A_i^j(t)}{t} + \lim_{t\to 0}\frac{B_i^j(t)}{t}\\
&= \frac{\pi_i}{\sigma^2}\sum_{j=1}^N \gamma_j^2\big[\sum_{k=1}^K \pi_k\nabla\phi_j(\mu_k)^\top v_k\big]\phi_j(\mu_i)(x_j - \mu_i) + \gamma_j\phi_j(\mu_i)\big[\frac{1}{\sigma^2}(x_j - \mu_i)^\top v_i(\mu_i - x_j) + v_i\big]
\end{aligned}$$

By developping the gradient of ϕ_j and rearranging the terms, we get such an expression for the hessian of f with direction V, for $i = 1, ..., K$:

$$(\nabla^2 f(M)[V])_i = \frac{\pi_i}{\sigma^2}\sum_{j=1}^{N}\big[\sum_{k=1}^{K}\pi_k\frac{\gamma_j^2\phi_j(\mu_k)\phi_j(\mu_i)}{\sigma^2}(x_j-\mu_k)^\top v_k\big](x_j-\mu_i)+\gamma_j\phi_j(\mu_i)\big[\frac{1}{\sigma^2}(x_j-\mu_i)^\top v_i(\mu_i-x_j)+v_i\big].$$

We easily check that for $i = 1, ..., K$, $(\nabla^2 f(M)[V])_i$ is in $\mathbb{R}^d$. Indeed x_j, μ_i and v_i are all in $\mathbb{R}^d$. The other terms are scalar in $\mathbb{R}$. The products $(x_j-\mu_k)^\top v_k$ and $(x_j-\mu_i)^\top v_i$ are also in $\mathbb{R}$.

Since each component of $\nabla^2 f(M)[V]$ is in $\mathbb{R}^d$ and that there are K components of $\nabla^2 f(M)[V]$, then $\nabla^2 f(M)[V] \in \mathbb{R}^{d\times K}$.

15. For the implementation of $\nabla^2 f(M)[V]$, we choose to use the same method as to calculate the gradient of f, i.e. using the function `sum` in matlab and multidimensions.

    In the following, we repeat the notation we use:

$$\beta_n^k = \frac{\|x_n-\mu_k\|^2}{2\sigma^2}$$

So considering $\gamma_j\phi_j(\mu_i)$, for $j = 1, ..., N$, $i = 1, ..., K$:

$$\begin{aligned}\gamma_j\phi_j(\mu_i) &= \frac{\phi_j(\mu_i)}{\sum_{k=1}^K \pi_k\phi_j(\mu_k)} = \frac{1}{\sum_{k=1}^K \pi_k\frac{\phi_j(\mu_k)}{\phi_j(\mu_i)}} = \frac{1}{\sum_{k=1}^K \pi_k\exp(-\beta_j^k+\beta_j^i)}\\ &= \big(\sum_{k=1}^K \pi_k\exp(-\beta_j^k+\beta_j^i)\big)^{-1}\end{aligned}$$

Similar calculations for $\gamma_j^2\phi_j(\mu_k)\phi_j(\mu_i)$, for $j = 1, ..., N$, $k = 1, ..., K$, and $i = 1, ..., K$:

$$\begin{aligned}\gamma_j^2\phi_j(\mu_k)\phi_j(\mu_i) &= \frac{\phi_j(\mu_k)\phi_j(\mu_i)}{\big(\sum_{l=1}^K\pi_l\phi_j(\mu_l)\big)^2} = \frac{1}{\big(\sum_{l=1}^K\pi_l\phi_j(\mu_l)\big)^2\frac{1}{\phi_j(\mu_k)\phi_j(\mu_i)}}\\ &= \frac{1}{\big(\sum_{l=1}^K\pi_l\frac{\phi_j(\mu_l)}{\sqrt{\phi_j(\mu_k)\phi_j(\mu_i)}}\big)^2} = \big(\sum_{l=1}^K\pi_l\frac{\phi_j(\mu_l)}{\sqrt{\phi_j(\mu_k)\phi_j(\mu_i)}}\big)^{-2}\\ &= \big(\sum_{l=1}^K\pi_l\exp(-\beta_j^l+\frac{1}{2}\beta_j^k+\frac{1}{2}\beta_j^i)\big)^{-2}\end{aligned}$$

We implement such a form for $(\nabla^2 f(M)[V])$ in `hessian_log_likelihood.m`, for $i = 1, ..., K$:

$$\begin{aligned}(\nabla^2 f(M)[V])_i = \frac{\pi_i}{\sigma^2}\sum_{j=1}^N\big[\sum_{k=1}^K\frac{\pi_k}{\sigma^2}\big(\sum_{l=1}^K\pi_l\exp(-\beta_j^l+\frac{1}{2}\beta_j^k+\frac{1}{2}\beta_j^i)\big)^{-2}(x_j-\mu_k)^\top v_k\big](x_j-\mu_i)\\ +\big(\sum_{k=1}^K\pi_k\exp(-\beta_j^k+\beta_j^i)\big)^{-1}\big[\frac{1}{\sigma^2}(x_j-\mu_i)^\top v_i(\mu_i-x_j)+v_i\big]\end{aligned}$$

In the code, the so called "fist part of the Hessian" is the first line of the formula above and, respectively, the "second part of the Hessian" is the second line.

```
function H=hessian_log_likelihood(M,v,X,sigma,Pi)
%Compute the Hessian of the negative log-likelihood function of the
%data points X=[x_1,...,x_N] sampled randomly
% with probability repartition Pi=[pi_1,...,pi_K]
% from K clusters of multivariable normal laws with means M=[mu_1,...,mu_K]
% and all with covariance sigma*I, in the direction v.

%Concern Questions 15

    X = permute(X,[1, 3, 2]) ; % d*1*N
    % M is d*K
    diff = M-X;  % d*K*N
    beta = sum(diff.^2) ./ (2*sigma^2) ; % 1*K*N in order to calculate phi

%we write the components in the following order : (i,j,k,l)
%then we can note that the dimensions should be : (K_i,N_j,K_k,K_l)
%sometimes the d-dimension is on the last or the fifth component.
%% first part of the Hessian:
res1=permute(X-M,[4,3,2,1]); %1xNxK_kxd which is (X_j-M_k)'
res2=sum(res1.*permute(v,[4,3,2,1]),4); % 1xNxK_k which is (X_j-M_k)'*V_k
res3=res1; %1xNxK_ixd
res3=permute(res3,[3,2,1,5,4]); %K_ixNx1x1xd
res4=res2.*res3; %K_ixNxK_kx1xd which is (X_j-M_k)'*V_k*(X_j-M_i)


%in exponential:
B= -permute(beta,[1,3,4,2]);
B=B+permute(1/2.*beta,[1,3,2]);
B=B+permute(1/2.*beta,[2,3,1]); %K_ixNxK_kxK_l

%continue summing:
res5=sum(permute(Pi,[1,4,3,2]).*exp(B),4); %K_ixNxK_k
res5=res5.^(-2); %K_ixNxK_k which is the sum over l to the -2
res6=res5.*res4; %K_ixNxK_kxd
res6=squeeze(res6);

res7=1/sigma^2.*sum(permute(Pi,[1,3,2]).*res6,3);
res7=squeeze(res7); %K_ixNxd;


res8=sum(res7,2);
res8=squeeze(res8); %K_ixd
H=1/sigma^2.*Pi'.*res8; %K_ixd

%% second part of the Hessian:
res10=permute(X-M,[2,3,4,1]); %K_ixNx1xd which is (X_j-Mi)
res11=sum(res10.*permute(v,[2,4,3,1]),4); %K_ixN which is (X_j-M_i)'*V_i
res12=-res10; %K_ixNx1xd which is (Mi-Xj)
res13=res11.*res12; %K_ixNx1xd which is (X_j-M_i)'*V_i*(Mi-Xj)
res13=res13./sigma^2; %which is 1/sigma^2*(X_j-M_i)'*V_i*(Mi-Xj)
res14=res13+permute(v,[2,4,3,1]); %K_ixNx1xd which is inside []

%the exponential:
C=-permute(beta,[1,3,2])+permute(beta,[2,3,1]); %K_ixN_K_k
res15=sum(permute(Pi,[3,1,2]).*exp(C),3);%K_ixN
res16=squeeze(sum(res15.^(-1).*res14,2)); %K_ixd sum over j=1,...,N

%% combine both part:
H=H+Pi'.*res16./sigma^2;
H=H';
end
```

In `question15.m`, we check that our Hessian is correct, see Figure [6].

```
clc
clear
close
%QUESTION 15: check that our hessian is correct.
load('data-toy.mat');
X=X';
Pi=pis';

f = @(M) log_likelihood(M, X, sigma, Pi); % function
g = @(M) grad_log_likelihood(M, X, sigma, Pi);    % gradient

M = 3*rand(d, K);             % random start

v  = rand(size(M));
v = v / norm(v,'fro');% random direction
t = logspace(-8, 0, 100);

Ft = zeros(size(t));
F = f(M);
G = g(M);

Hv=hessian_log_likelihood(M,v,X,sigma,Pi);

for i=1:length(t)
    Ft(i) = f(M + t(i)*v);
end

error = abs( Ft - F - trace(v'*G)*t-1/2*trace(v'*Hv)*t.^2 );
loglog(t, error, t, t.^3)
title('Error of taylor expansion of order 2 of f versus step length (question15)')
xlabel('step length')
grid on
legend('error','$O(t^3)$','Interpreter','latex','Location','Southeast')
```

We indeed obtain a slope of 3, which confirms that our error is in $O(t^3)$ and that our hessian is correct. Nevertheless, we observe some discrepancies for $t < 10^{-4}$. The reason of this phenomena is that the limit of the computer's reliability has been reached. We are too close to this limit, which causes numerical errors.

16. By exercise 1 of serie 6, we calculate which t must be used if we enter the `if` condition in the algorithm (see line 42 of the code `truncatedCG.m`). We have that t must be such as :

$$\|v_n\|^2 = \Delta^2 \iff \|v_{n-1} + tp_{n-1}\|^2 = \Delta^2 \iff \langle v_{n-1} + tp_{n-1}, v_{n-1} + tp_{n-1}\rangle = \Delta^2.$$

This is equivalent to find t such as :

$$\langle p_{n-1}, p_{n-1}\rangle t^2 + 2\langle v_{n-1}, p_{n-1}\rangle t + \langle v_{n-1}, v_{n-1}\rangle - \Delta^2 = 0$$

which is a quadratic equation. By simple calculation, we find that the roots are given by:

$$t = \frac{-\langle v_{n-1}, p_{n-1}\rangle \pm \sqrt{\langle v_{n-1}, p_{n-1}\rangle^2 - \|p_{n-1}\|^2(\|v_{n-1}\|^2 - \Delta^2)}}{\|p_{n-1}\|^2}$$

By the course, we have to choose $t \geq 0$. To guarantee that, we take the positive root, i.e.

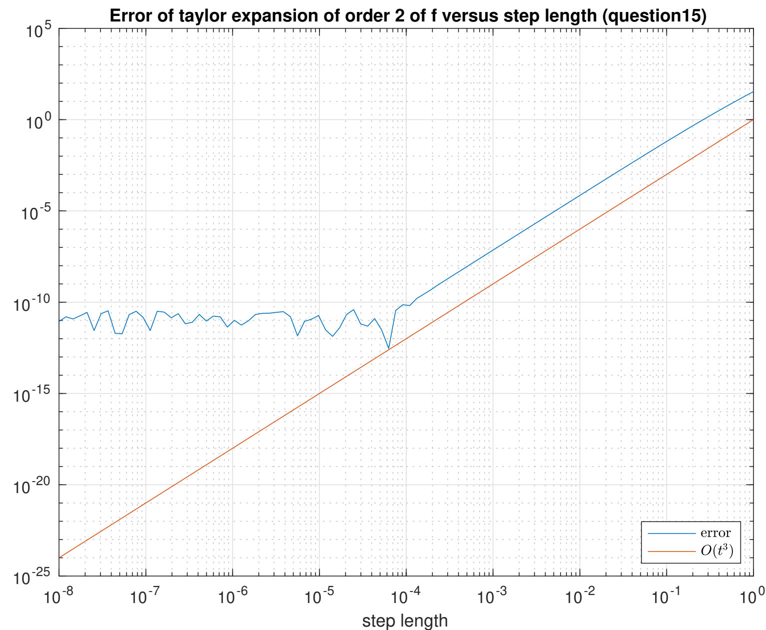

Figure 6: Graph for question 15. We observe that the slope is indeed of 3. See discrepancies for $t < 10^{-4}$.

$$t = \frac{-\langle v_{n-1}, p_{n-1}\rangle + \sqrt{\langle v_{n-1}, p_{n-1}\rangle^2 - \|p_{n-1}\|^2(\|v_{n-1}\|^2 - \Delta^2)}}{\|p_{n-1}\|^2}.$$

We implemented a code `give_t.m` which gives the positive root t from the calculation above:

```
function t=give_t(p,v,delta)
%%INPUT:
% p=p_{n-1}
% v=v_{n-1}
% delta=radius of TR
% OUTPUT:
% t= positive root

norm_p_square=trace(p'*p);
prod=trace(v'*p);
res=prod^2-norm_p_square*(trace(v'*v)-delta^2);
t=1/norm_p_square*(-prod+sqrt(res));
end
```

You'll notice the function at line 43 of `truncatedCG.m`.

By remark 6.16 of the course, we implement truncated CG such that it returns a boolean indicating if the norm of u_k is equal to Δ_k (true) or not (false). This happens when v_{n-1}^+ is outside of the region defined by the sphere of radius Δ_k, or when the hessian is not positive definite. This will be useful when we will implement the trust region method. See below the code for the truncated gradient. Moreover, the purpose of the value `small` is to terminate the procedure if the norm of r_n

is smaller than `small`. In the homework:

$$\texttt{small} = \|r_0\| \min(\|r_0\|, \kappa)$$

with $\kappa = 1/10$.

```
1  function [u,Hu,delta_attained]=truncatedCG(M,delta,g,H)
2  %Truncated gradient descent method
3  %INPUT:   M: the current value M_k
4  %         rad: the current delta
5  %         g: the current gradient
6  %         H: the current hessian (not evaluate in a direction yet)
7  % OUTPUT: u (dimension dxK) and Hu (same dimension), u being the minimizer
8  % of the function m(v)=1/2 <v,Hv> - <b,v>. Output delta_attained which is a
9  % boolean indicating if norm(u)=delta (true) or not (false). The latter
10 % will help for a condition in trust_region_limit.
11
12 %Question16
13
14
15 [d,K]=size(M);
16
17 b=-g;
18 r_0=b;
19 v_prev=zeros(d,K); %dxK
20 r_prev=r_0; %dxK
21 p_prev=r_prev; %dxK
22
23 maxit=500;
24 u=zeros(d,K);
25 Hu=zeros(d,K);
26
27 v_next=v_prev;
28 r_next=r_prev;
29 p_next=p_prev;
30
31 k=1/10;
32
33 small=norm(r_0,'fro')*min([norm(r_0,'fro') k]);
34
35 for i=1:maxit
36     Hp=H(p_prev);
37     prod=trace(p_prev'*Hp);
38
39     alpha=trace(r_prev'*r_prev)/prod;
40     v_prev_plus=v_prev+alpha*p_prev;
41
42     if (prod<=0 || norm(v_prev_plus,'fro')>=delta)
43         t=give_t(p_prev,v_prev,delta);
44         v_next=v_prev+t*p_prev;
45         u=v_next;
46         delta_attained=true; %see remark 6.16
47         Hu=b-r_prev+t*Hp;
48         break
49     else
50         v_next=v_prev_plus;
51     end
52
53     r_next=r_prev-alpha*Hp;
54
55     if norm(r_next,'fro')<=small
```

```
56          u=v_next;
57          Hu=b-r_next;
58          delta_attained=false;
59          break
60      end
61
62      beta=trace(r_next'*r_next)/trace(r_prev'*r_prev);
63      p_next=r_next+beta*p_prev;
64
65      v_prev=v_next;
66      r_prev=r_next;
67      p_prev=p_next;
68  end
69
70  if i==maxit
71      delta_attained=false;
72  end
73  end
```

17. We implement a trust region solver. We decided to implement the Cauchy step method as well, given by the function below (see `cauchy_step.m`).

```
1   function [u,delta_attained]=cauchy_step(rad,g,H)
2   % Cauchy step method
3   % INPUT:  rad: the current delta
4   %         g: the current gradient
5   %         H: the current hessian (not evaluate in a direction yet)
6   %
7   % OUTPUT: u: the minimizer of m(v)
8   %         delta_attained: boolean, true if norm(u)=rad, otherwise is false
9
10  prod=trace(g'*H(g));
11  norm_g_square=trace(g'*g);
12  norm_g=sqrt(norm_g_square);
13  if prod>0
14      if norm_g_square/prod < rad/norm_g
15         t=norm_g_square/prod;
16         delta_attained=false;
17      else
18         t=rad/norm_g;
19         delta_attained=true;
20      end
21  else
22      t=rad/norm_g;
23      delta_attained=true;
24  end
25  u=-t*g;
26
27  end
```

In the function of the trust region method (see `trust_region.m` below), the argument `tCG` is a boolean indicating if we want to use the truncated CG method (true) or the Cauchy step method (false).

We choose for $\bar{\Delta}$, Δ_0 and ρ' the values given in the homework, i.e. $\bar{\Delta} = \sqrt{dK}$, $\Delta_0 = \bar{\Delta}/8$ and $\rho' = 1/10$.

```
function [M,grad,time,obj_value]=trust_region(maxtime,M0,X,pis,sigma,tCG)
%INPUT: maxtime for the loop, M0 starting point, X the data, pis the prob.
%, sigma, and a boolean tCG which is true if we want to use the truncated
%conjugate gradient method to find u, and which is false if we want to use
%the Cauchy step method to find u.
%OUTPUT  M : last point of research
%          grad  : vector of the successive norms of gradient
%          time  : vector of successive times
%          obj_value : objective value of M
%Question17

%default values
[d,K]=size(M0);
max_rad=sqrt(d*K);
rad_0=max_rad/8;
rho2=1/10;

X=X';
Pi=pis';

M=M0;
rad=rad_0;
grad=[];
time=[];
f  = @(P)      log_likelihood(P, X, sigma, Pi);
grad_function = @(P) grad_log_likelihood(P, X, sigma, Pi);

tic
while toc<maxtime
    time=[time toc];

    g=grad_function(M);
    H= @(v) hessian_log_likelihood(M,v,X,sigma,Pi);

    grad=[grad norm(g,'fro')];

    m=@(v) f(M)+trace(g'*v)+1/2*trace(v'*H(v));

    %find min of m by tCG or Cauchy step method:
    if tCG==true
        [u,~,delta_attained]=truncatedCG(M,rad,g,H);
    else
        [u,delta_attained]=cauchy_step(rad,g,H);
    end

    M_maybe=M+u;
    rho=(f(M)-f(M_maybe))/(f(M)-m(u));

    %accept or reject:
    if rho>rho2
        M=M_maybe;
    end

    %update the trust region radius:
    if rho<1/4
        rad=1/4*rad;
    else
        if rho>3/4 && delta_attained
            rad=min([2*rad max_rad]);
        end
    end
end
```

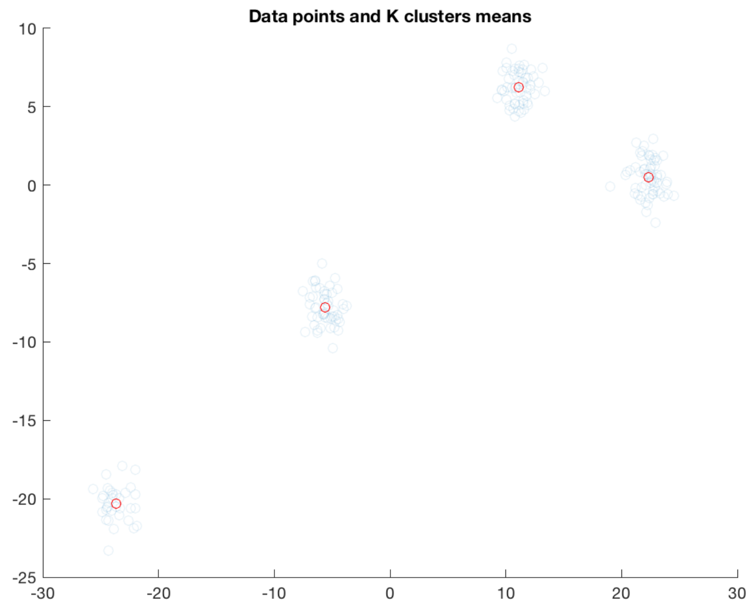

Figure 7: Graph for question 18. We see the K cluster means in red and the data points in blue.

```
63
64 obj_value=log_likelihood(M, X, sigma, Pi);
65 end
```

18. Let's run the trust region method with truncated CG on `data-toy.mat`, with the file `question18.m` below.

```
1  clc
2  clear
3  close
4
5  load('data-toy.mat');
6  maxtime=4;
7  M0 =  mean(X',2) .* ones(1,K) + 1e-4*rand(d,K);
8  tCG=true;
9
10 [M_1,grad,time,obj_value]=trust_region(maxtime,M0,X,pis,sigma,tCG);
11 display_data_means(X,M_1)
```

We obtain the graph on Figure [7].

The result is reasonable and good, as the cluster means lie in the middle of the clusters.

19. See `question19.m` below for the implementation. We decided to indicate the results obtain by the trust-region method with Cauchy step as well. The last part of the code is for the next and last question.

On Figure [8], you can see the norm of the gradient as a function of the iteration on the first graph, and then as a function of time on the second graph. We observe that the trust-region method with

truncated conjugate gradient (TR with tCG) gives much better results in less iterations. It gives better results also in 1.5 second, compared to the two other methods considered. The backtracking line-search gradient descent (BLSGD) method gives a correct gradient, as its norm is at 10^{-5} at the end of the execution of the program. It needs more iterations to attain it (about 400). Moreover, this method is having better results than the trust region methods on the first second. But then it does not get below 10^{-5}, as does TR with tCG.

```
1  clc
2  clear
3  close
4
5  %% Load the data and initialize
6  load('data.mat');
7  maxtime=4;
8  M0 =  mean(X',2) .* ones(1,K) + 1e-4*rand(d,K);
9
10 %% BLSGD
11 alpha0=1e-2;
12 rho=0.5; % in [0.5,0.8]
13 c=1e-4;
14
15 [M_BLSGD,grad_BLSGD,time_BLSGD,obj_value_BLSGD]=BLSGD(maxtime,M0,X,pis,sigma,alpha0,rho,c);
16 I_BLSGD=1:1:size(time_BLSGD,2);
17
18 %% Trust Region with Cauchy step
19 tCG=false;
20 [M_CS,grad_CS,time_CS,obj_value_CS]=trust_region(maxtime,M0,X,pis,sigma,tCG);
21 I_CS=1:1:size(time_CS,2);
22
23 %% Trust Region with Truncated Conjugate Gradient
24 tCG=true;
25 [M_tCG,grad_tCG,time_tCG,obj_value_tCG]=trust_region(maxtime,M0,X,pis,sigma,tCG);
26 I_tCG=1:1:size(time_tCG,2);
27
28 %% Plot the norm of the gradient
29 figure
30 subplot(2,1,1)
31 semilogy(I_BLSGD,grad_BLSGD,I_CS,grad_CS,I_tCG,grad_tCG);
32 legend('BLSGD method','TR with CS','TR with tCG');
33 xlabel('Iteration'); ylabel('norm of the gradient');
34 grid on;
35 title('Comparing the norm of the gradient as a function of the iteration');
36
37 subplot(2,1,2)
38 semilogy(time_BLSGD,grad_BLSGD,time_CS,grad_CS,time_tCG,grad_tCG);
39 legend('BLSGD method','TR with CS','TR with tCG');
40 xlabel('Time'); ylabel('norm of the gradient');
41 grid on;
42 title('Comparing the norm of the gradient as a function of time');
43
44 X=X';
45 %% Question 20 : clusters obtained
46 figure
47 %to plot on the same figure with different colors, we'll simply rewrite the
48 %code of display_data_means(X,M):
49
50 scatter(X(1,:),X(2,:),'MarkerEdgeAlpha',0.1) %for transparency
51 hold on
52 plot(M_BLSGD(1,:),M_BLSGD(2,:),'*')
```

```
53 hold on
54 plot(M_CS(1,:),M_CS(2,:),'kx')
55 hold on
56 plot(M_tCG(1,:),M_tCG(2,:),'s')
57 hold on
58 title('Data points and K clusters means')
59 legend('Data','BLSGD','TR with CS','TR with tCG')
```

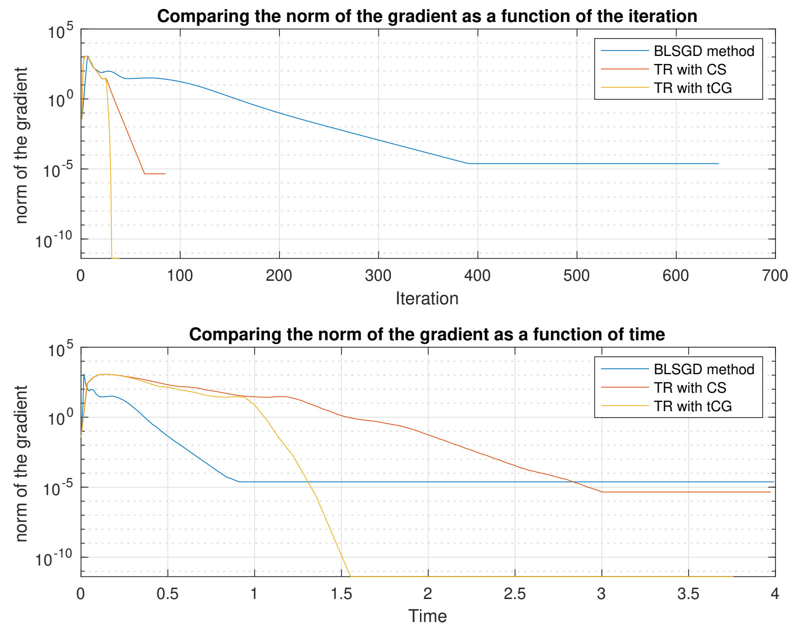

Figure 8: Graphs for question 19. Here we compare the norms of the gradient, depending on the methods chosen.

20. The results are given by `question19.m`, at the end of the code. We obtain the plot in Figure [9]. Note that we don't use `display_data_means.m` because we want to display the means of the different methods on the same plot. We simply adapt the code to our setting.

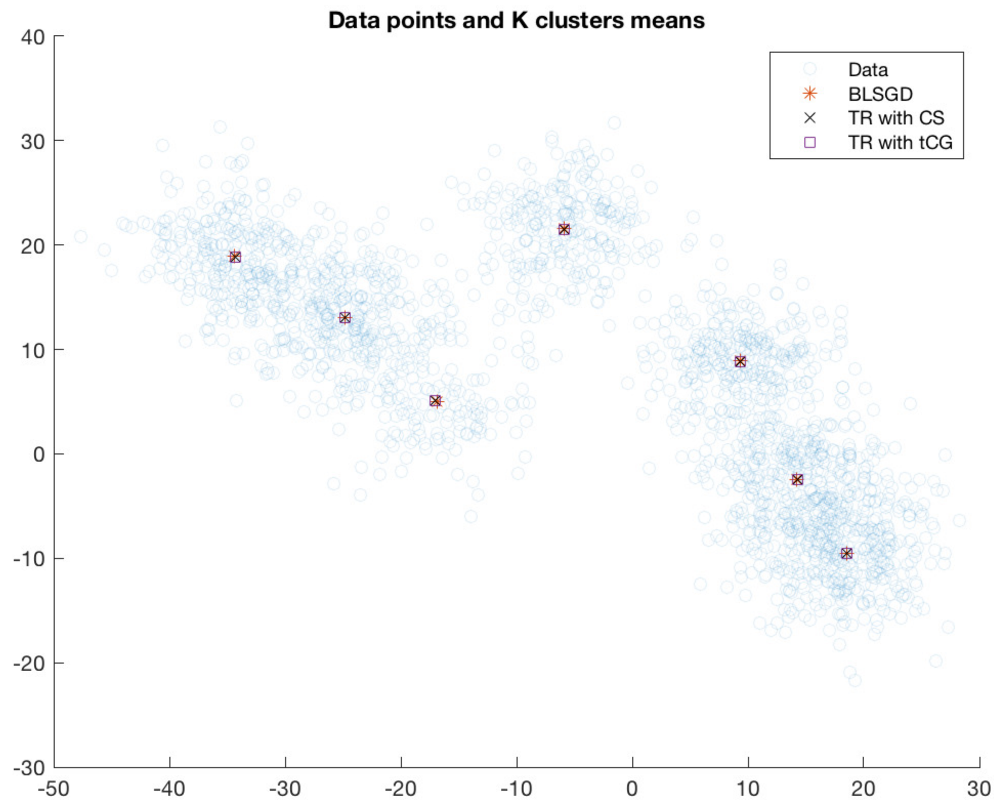

Figure 9: Graph for question 20. All cluster means coincide.

We notice that all methods give the same mean clusters, which seem correct. We obtain such results after running a few times. Sometimes BLSGD cluster means are a bit off the other cluster means given by the trust region methods, but the offset is very small, see Figure [10] below.

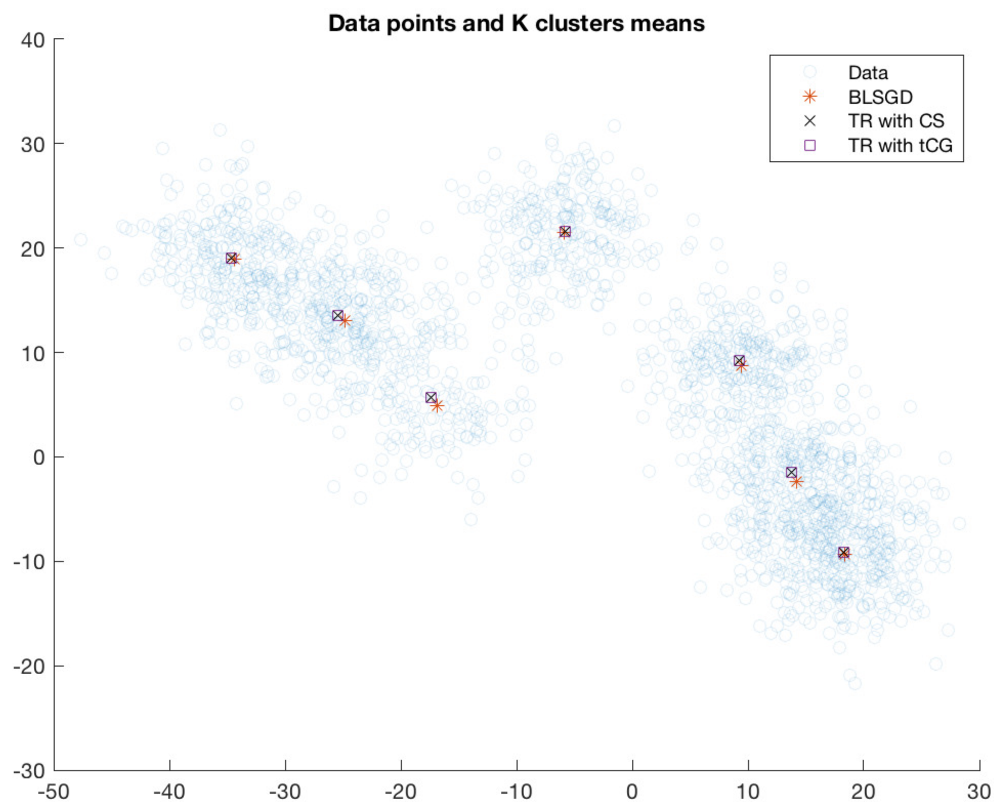

Figure 10: Graph for question 20. See that BLSGD shifts a little bit from the two other methods.

To conclude, question 19 confirms that TR with tCG gives us better results in little time. The code is efficient enough to give good results in less than two seconds. The homework has been successful: we have implemented a code which, from the observations $x_1, ..., x_N$, retrieves the cluster means $\mu_1, ..., \mu_K$.