Homework 2

Noé Hollande, Yujun Kim, Thomas Renard

April 15, 2022

Theoretical Part

Problem 1

Check the MATLAB file attached together on moodle.

```
%% Problem 1
%Definition of h
global sigma;
sigma = 5;
function h = h(x)
    global sigma;
    h = exp(-norm(x)^2/sigma^2);
end

%Definition of other functions
function y = phi(x, P)
    y = h(P-x);
end

function y = Phi(X, P)
    p = @(x)phi(x, P);
    y = p(X)*ones(K);
end

function y = f(X, P, y)
    y = norm(Phi(X, P)-y)^2/2;
end
```

Problem 2

$$
\begin{aligned}
h(x) =& e^{-||x||^2/\sigma^2} \\
\varphi(x) =& \begin{bmatrix} h(p_1 - x) \\ \vdots \\ h(p_{n^2} - x) \end{bmatrix} \\
\Phi(X) =& \Sigma_{k=1}^{K} \varphi(x_k)
\end{aligned}
$$

Thus,

$$\begin{aligned}
f(X) =& \frac{1}{2}||\Phi(X) - y||^2 \\
=& \frac{1}{2}||\Sigma_{k=1}^{K}\varphi(x_k) - y||^2 \\
=& \frac{1}{2}\left|\left|\begin{bmatrix} h(p_1 - x) \\ \vdots \\ h(p_{n^2} - x) \end{bmatrix} - y\right|\right|^2 \\
=& \frac{1}{2}\Sigma_{i=1}^{n^2}\left(\Sigma_k e^{-||p_i - x_k||^2/\sigma^2} - y_i\right)^2 \\
\leq& \frac{1}{2}\Sigma_{i=1}^{n^2}(K + |y_i|)^2 \\
\leq& \frac{n^2}{2}(K + max|y_i|)^2
\end{aligned}$$

Hence, f is bounded and not constant, which shows that f is not convex. You can also check this by follwing figure

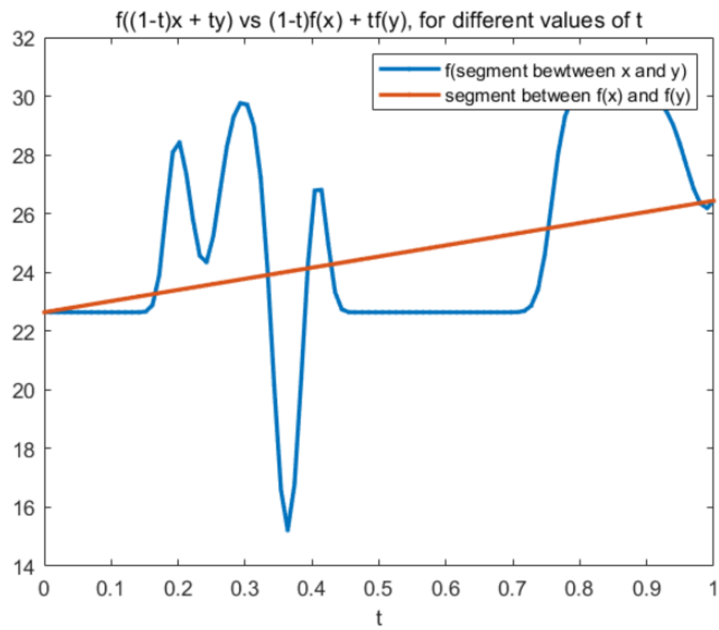

Figure 1: Proof of Non-2convexity

Problem 3

i-th element of $\varphi(x)$ is $\varphi_i = e^{-||p_i - x||^2/\sigma^2}$. Thus, i-th row of $D\varphi(x)$ is

$$\nabla\varphi_i(x) = \frac{2}{\sigma^2}(p_i - x)e^{-||p_i - x||^2/\sigma^2}$$

and so,

$$D\varphi(x) = \begin{bmatrix} \nabla\varphi_1(x) \\ \vdots \\ \nabla\varphi_{n^2}(x) \end{bmatrix} = \frac{2}{\sigma^2}\begin{bmatrix} (p_1 - x)e^{-||p_1 - x||^2/\sigma^2} \\ \vdots \\ (p_{n^2} - x)e^{-||p_{n^2} - x||^2/\sigma^2} \end{bmatrix}$$

which is $n^2 \times 2$ matrix. This can be represented simpler as following.

$$D\varphi(x) = \frac{2}{\sigma^2}\varphi(x). * (p - x\mathbb{1})$$

where $.*$ operation on RHS is MATLAB notation.

Problem 4

As adjoint of linear transformation can be obtained by the transpose of the representing matrix,

$$D\varphi^T(x) = \frac{2}{\sigma^2}[\varphi(x). * (p - x\mathbb{1})]^T$$

which is $2 \times n^2$ matrix.

Problem 5

Note that $f : \mathbb{R}^{2\times K} \longrightarrow \mathbb{R}$. $\nabla f(X)$ is unique matrix satisfying

$$Df(X)[U] =< \nabla f(X), U >$$

where $< \cdot, \cdot >$ is *Frobenius inner product.* Recall that

$$\begin{aligned} f(X) =& \frac{1}{2}||\Phi(X) - y||^2 \\ =& \frac{1}{2}\Sigma_{i=1}^{n^2}\left(\Sigma_k e^{-||p_i - x_k||^2/\sigma^2} - y_i\right)^2 \end{aligned}$$

By chain rule,

$$\begin{aligned} Df(X) &= D\left(\frac{1}{2} < \Phi(X) - y, \Phi(X) - y >\right) \\ &= D\left(\Phi(X) - y\right)^T\left(\Phi(X) - y\right) \\ &= D\Phi(X)^T\left(\Phi(X) - y\right) \\ &= \begin{bmatrix} D\varphi^T(x_1) \\ \vdots \\ D\varphi^T(x_K) \end{bmatrix}\left(\Phi(X) - y\right) \end{aligned}$$

Problem 6

The following figure 2 shows that gradient of f is well implemented.

Question 8:

When our results don't make sense, we get a picture which is either fully blue or fully teal. This is due to the fact that our parameters are not optimal. But when they're good, we see one or two stars well positioned, and one or two that are missing. It is a coherent result, and the TR algorithm will help fix the issue of missing stars.

Refer figure 3,4.

Question 9:

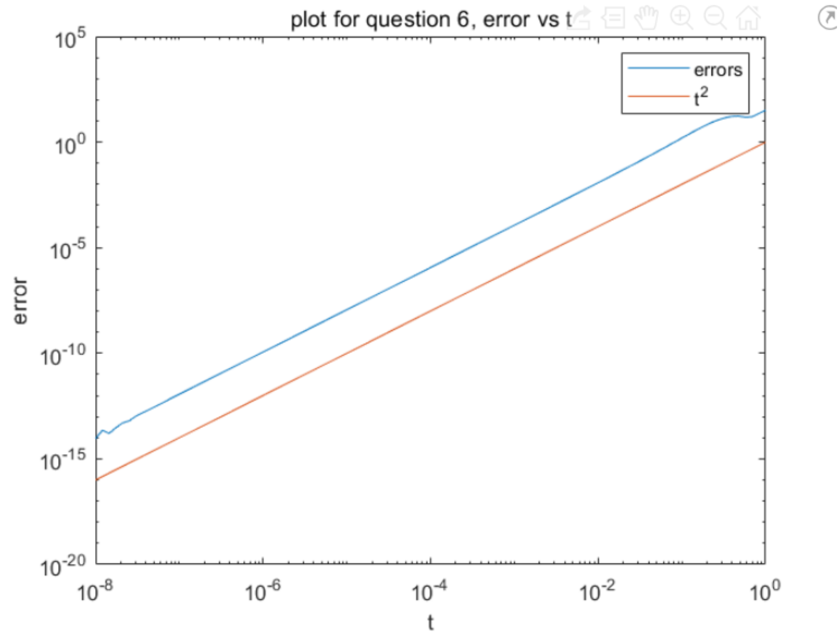

Figure 2:

The objective values are mostly different, and pretty high. This is due to the fact that f measures the mismatch between tentative positions X and the observations y. The smaller f(x) is, the more accurate X is. Thus, as depicted in 8), here using LS GD; we get positions that are not all accurate, which leads to f(x) being high and different as the stars position that are incorrect are not always the same.

Refer figure 5.

Question 11

Note that $\nabla f(X) = D\Phi(X)^T(\Phi(X) - y)$. Also,

$$\Phi(X + tU) = Phi(X) + tD\Phi(X)[U] + o(t^2)D\Phi(X + tU) = D\Phi(X) + tD^2\Phi(X)[U] + o(t^2)$$

Thus,

$$\begin{aligned}
\nabla^2 f(X)[U] &= lim_{t\to\infty}\frac{\nabla f(X + tU) - \nabla f(X)}{t} \\
&= lim_{t\to\infty}\frac{D\Phi(X + tU)^T(\Phi(X + tU) - y) - D\Phi(X)^T(\Phi(X) - y)}{t} \\
&= lim_{t\to\infty}\frac{(D\Phi(X) + tD^2\Phi(X)[U] + o(t^2))^T(\Phi(X) + tD\Phi(X)[U] + o(t^2) - y)}{t} \\
&\quad - \frac{D\Phi(X)^T(\Phi(X) - y)}{t} \\
&= \nabla\Phi(X)^T\nabla\Phi(X)
\end{aligned}$$

where the last line comes after canceling common terms and taking limit to zero for $o(t^2)$ terms in numerator.

Question 12:

The plot almost looks like a line with slope 3, except that with t under a low value, it's

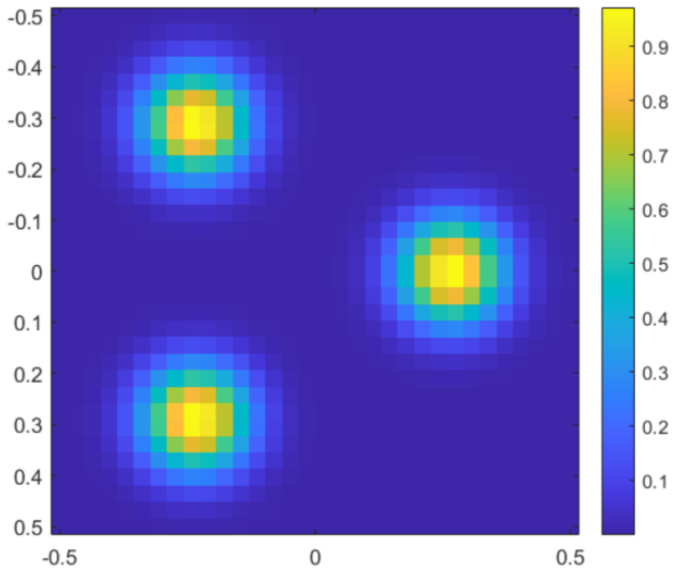

Figure 3:

more like a constant instead of $O(t^3)$, with small perturbations. This is due to the fact that we implemented a a complicated form of the Hessian, which results in having round-off errors when t is really small.

Refer figure 6.

Question 14: Refer figure 7,8

Question 15:

This looks reasonable using the same argument as in question 8.

Question 16: Refer figure 9.

Question 17:

We found X^* by running the TR algorithm and analyzing the image , to get X^* as close as we can get to y. Indeed, TR gives us a minimum but we just have to modify it a bit such that it is not a local minimum, but close to the global minimum which is supposed to result in to an image close to the one given by y. Basically, the images we get are almost the same, we just have small differences which are the result of some small imprecision of TR.

1. Voici le code et les réponses aux questions sur le matlab :

```
    %Question 2
%plot to show that f is not convex
clc
clear all
load data-toy.mat
t = linspace(0,1, 100);
X1 = randn(2,K);
X2 = randn(2,K);
   fseg = zeros(1, length(t));
segf = zeros(1, length(t));

```

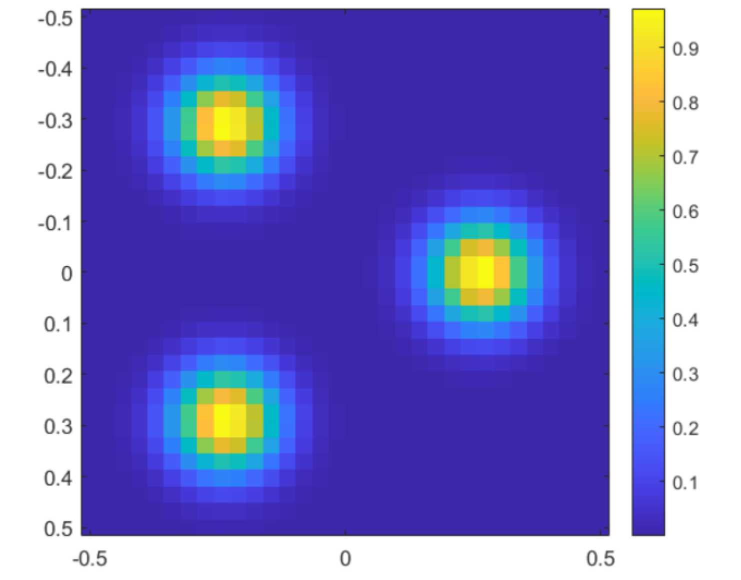

Figure 4:

```
for i = 1:length(t)
    fseg(i) = f(((1-t(i))*X1+t(i)*X2),P,y);
    segf(i) = (1-t(i))*f(X1,P,y) + t(i)*f(X2,P,y);
end

figure
plot(t, fseg, '.-', 'LineWidth', 2)
hold on
plot(t, segf, '.-', 'Linewidth', 2)
hold off
title('f((1-t)x + ty) vs (1-t)f(x) + tf(y), for different values of t')
legend('f(segment bewtween x and y)', 'segment between f(x) and f(y)')
xlabel('t')

%Question 6
X = -0.5+(0.5+0.5).*rand(2,K);
X = X/3;
v = -0.5+(0.5+0.5).*rand(2,K);
v = v/norm(mat2vec(v));
checkgradient(X, P, y, v);
%Question 7
params.maxiters = 30000;
params.maxtime = 20;
params.tolgradnorm = 1e-5;

lsparams.alphabar = 1e-4;
lsparams.c = 1e-4;
lsparams.rho = 0.7;
lsparams.alphamin = 1e-8;

%Question 8
clear K n y P
load data-toy.mat
[Xk, gradnorms, times] = lsgradientdescent(X, P, y, params, lsparams);
fprintf('position that we obtain after running GD : ')
Xk
plotimage(vec2mat(Phii(Xk,P)));
```

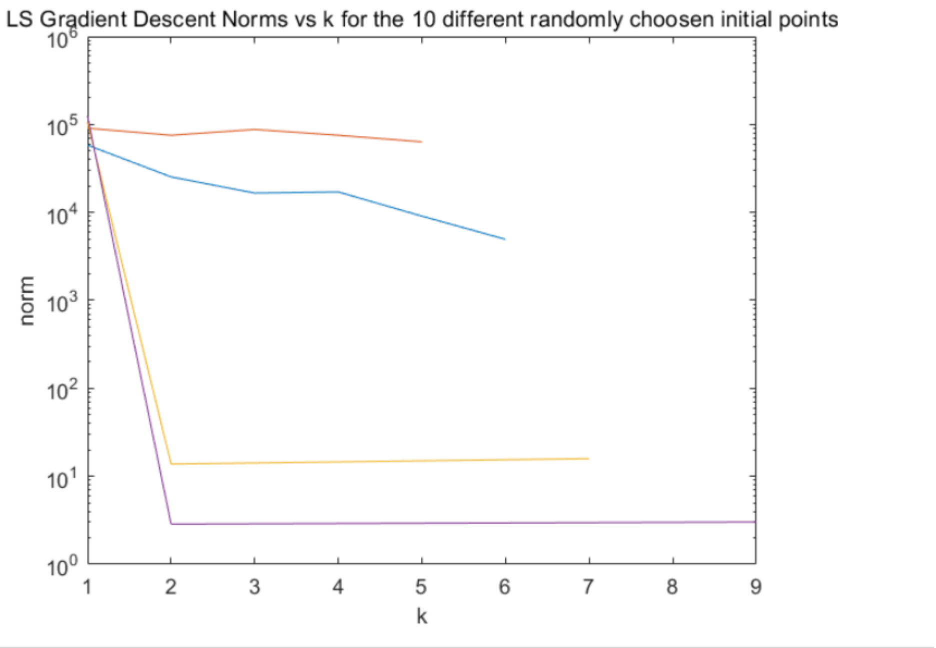

Figure 5:

```
plotimage(vec2mat(Phii(Xstar,P)));

%Question 9 and 10
clear K n y P
load data.mat
%it might take some time as we plot for 5 points sorry :'(
%feel free to reduce the max time of running, I put it that high to get
%some coherent results and easy to understand
%or reduce the number of random points
objf = zeros(5,1);
params.maxtime = 60;

for i = 1:5
    Xi = -0.5+(0.5+0.5).*rand(2,K);
    [Xki, gradnormsi, timesi] = lsgradientdescent(Xi, P, y, params, ...
        lsparams);
    objf(i) = f(Xki, P, y);
    semilogy(gradnormsi);
    hold on
end
hold off
title('LS Gradient Descent Norms vs k for the 10 different randomly ...
    choosen initial points')
xlabel('k')
ylabel('norm')

fprintf('objective value for 10 different randomly choosen initial ...
    points')
objf

%Question 12
clear K n y P
load data-toy.mat
X = -0.5+(0.5+0.5).*rand(2,K);
U = -0.5+(0.5+0.5).*rand(2,K);
```

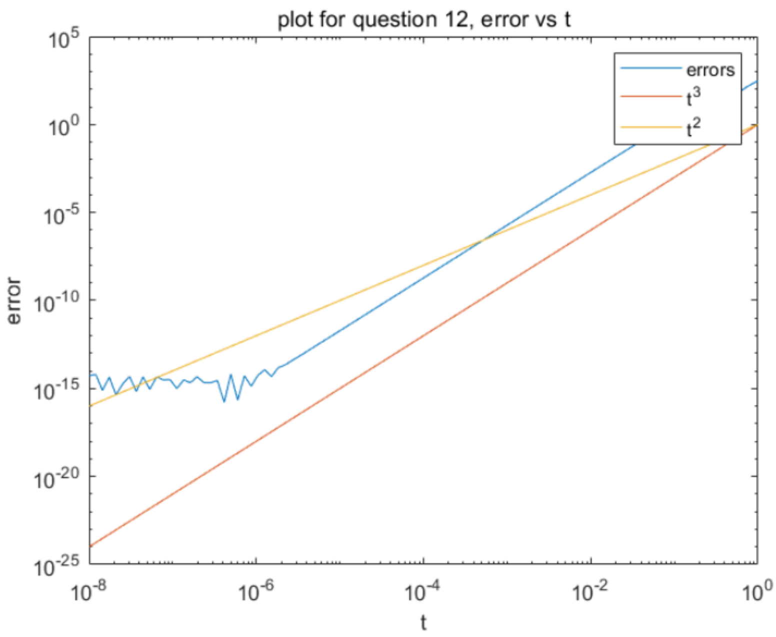

Figure 6:

```
U = U/norm(mat2vec(U));
checkhessian(X,P,y,U);

%Question 14
clear K n y P
load data-toy.mat
Δ = sqrt(2*K);
Δ0 = Δ/8;
rho = 0.1;
maxiter = 5000;
tol = 1e-8;
X = -0.5+(0.5+0.5).*rand(2,K);
X = X/3;
[thetaTR, gradnormsTR, Xk] = TR(X, P, y, rho, Δ, Δ0, tol, maxiter);
fprintf('position that we obtain after running TR : ')
Xk
plotimage(vec2mat(Phii(Xk,P)));
plotimage(vec2mat(Phii(Xstar,P)));

%Question 16
clear K n y P Xstar
load data3.mat
Δ = sqrt(2*K);
Δ0 = Δ/8;

rho = 0.1;
maxiter = 100;
tol = 1e-8;

params.maxiters = 30000;
params.maxtime = 20;
params.tolgradnorm = 1e-5;

lsparams.alphabar = 1e-4;
lsparams.c = 1e-4;
```

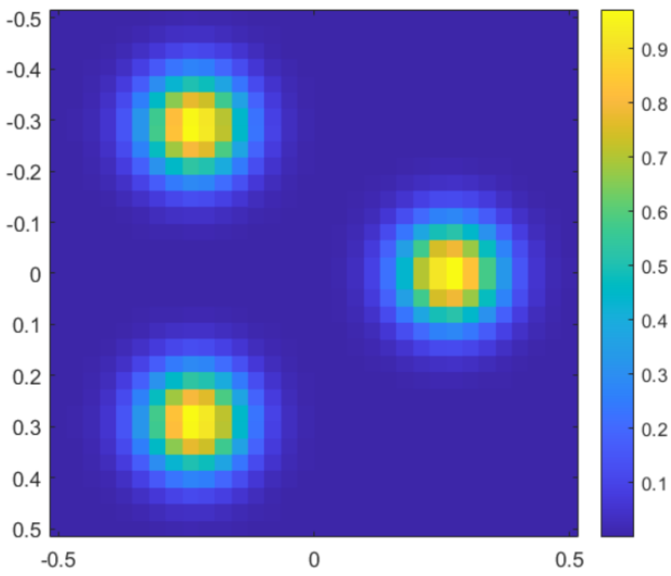

Figure 7:

```
lsparams.rho = 0.7;
lsparams.alphamin = 1e-8;

X = -0.5+(0.5+0.5).*rand(2,K);
X = X/3;

[Xk, gradnorms, times] = lsgradientdescent(X, P, y, params, lsparams);
[thetaTR, gradnormsTR, XkTR] = TR(X, P, y, rho, Δ, Δ0, tol, maxiter);

figure
semilogy(gradnorms)
hold on
semilogy(gradnormsTR)
hold off
title('LS Gradient Descent Norms, and TR Norms vs k for the same ...
    randomly choosen initial point')
legend('LS GD', 'TR')
xlabel('k')
ylabel('norm')
```

2. Voici le code matlab :

```
    function X = vec2mat(x, m, n)
% Reshapes a vector x into a m*n matrix.
% If m and n are not given the it tries to make a square matrix.
    if nargin == 1
        m = sqrt(numel(x));
        if mod(m, 1) ≠ 0
            error('vec2mat: Cannot make a square matrix from the ...
                input vector.')
        end
        n = m;
    end
    X = reshape(x, m, n);
end
```

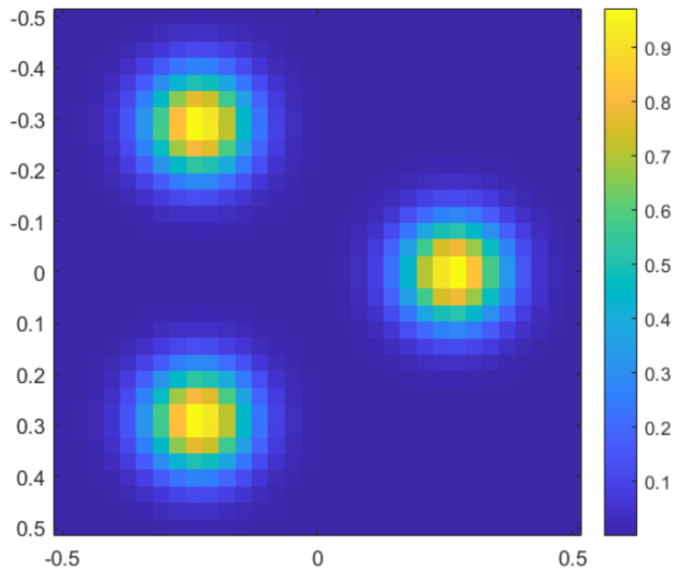

Figure 8:

---

3. Voici le code matlab de la question :

```
    function checkgradient(X, P, y, v)
n = 100;
t = logspace(-8,0,n);
g = fgrad(X,P,y);
fv = f(X,P,y);
fv = fv*ones(1,n);
dots = dot(mat2vec(v),mat2vec(g));
dots = t*dots;

fs = zeros(1,n);
for i = 1:n
    fs(i) = f(X+t(i)*v,P,y);
end
errors = abs(fs - fv - dots);

slope = @(x) x.^2;
    h = slope(t);

    figure
    loglog(t, errors)
    hold on
    loglog(t,h)
    hold off

    title('plot for question 6, error vs t')
    legend('errors', 't^2')
    xlabel('t')
    ylabel('error')

end
```

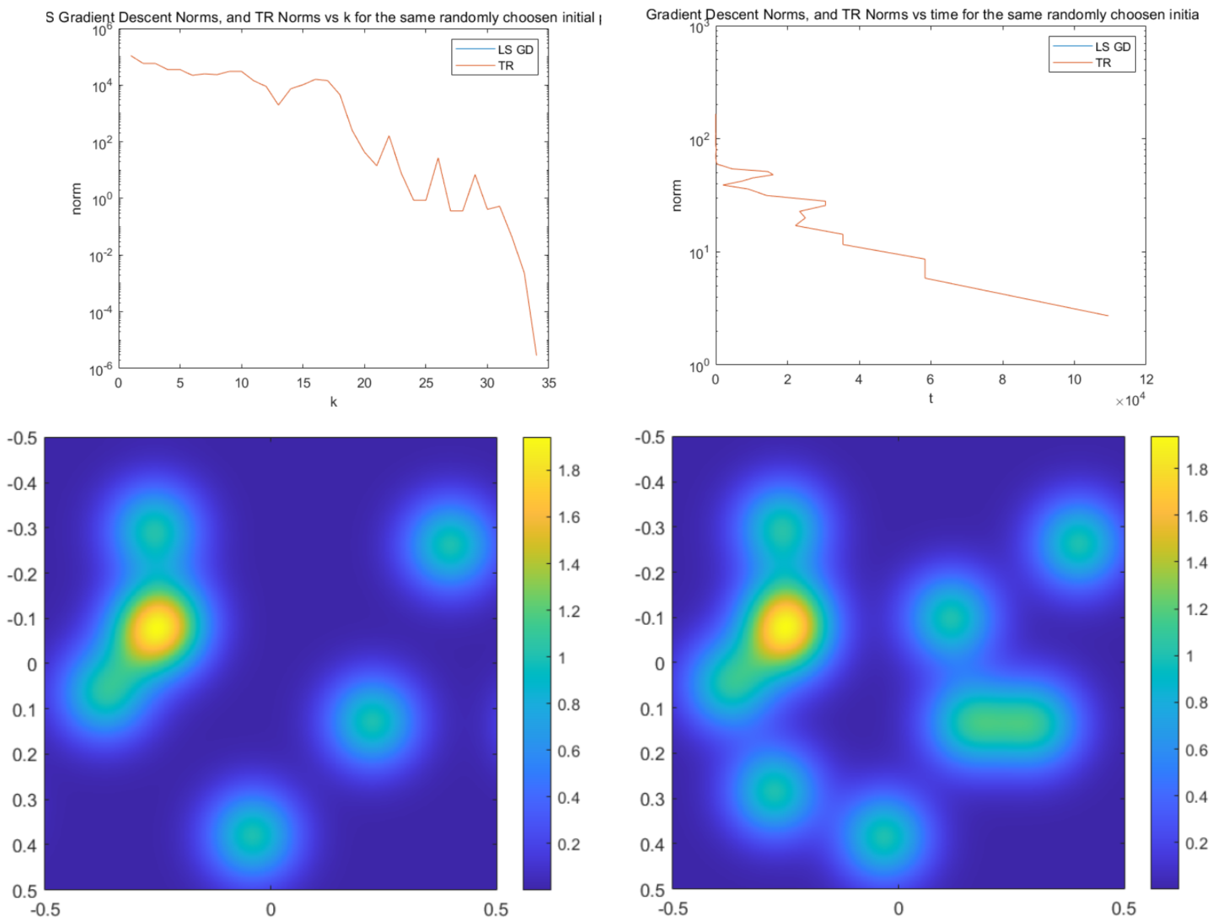
S Gradient Descent Norms, and TR Norms vs k for the same randomly choosen initial
LS GD
TR
norm
k
Gradient Descent Norms, and TR Norms vs time for the same randomly choosen initia
LS GD
TR
norm
t

Figure 9:

4. Voici le code matlab :

```
    function checkhessian(X,P,y,U)
n = 100;
t = logspace(-8,0,n);
fx = f(X,P,y);
gx = fgrad(X,P,y);
g2x = fhess(X,P,y,U);
fs = zeros(1,n);
for i = 1:n
    fs(i) = f(X+t(i)*U,P,y);
end
dots1 = dot(mat2vec(U),mat2vec(gx));
dots1 = t*dots1;
dots2 = dot(mat2vec(U),mat2vec(g2x));
dots2 = 0.5*(t.^2)*dots2;

error = abs(fs - fx - dots1 - dots2);

slope = @(x) x.^3;
    h = slope(t);
    slopeb = @(x) x.^2;
    hb = slopeb(t);

    figure
    loglog(t, error)
    hold on
    loglog(t,h)
    loglog(t, hb)
    hold off

    title('plot for question 12, error vs t')
    legend('errors', 't^3','t^2')
    xlabel('t')
    ylabel('error')

end
```

5. Voici le code matlab :

```
    function f = f(X, P, y)
f = 0.5*norm(Phii(X,P) - mat2vec(y))^2;
end
```

6. Voici le code matlab :

```
    function g = fgrad(X,P,y)
sigma = 0.1;
Dphi = @(x) ((2/sigma^2)*(P-x).*phi(x,P)');
g = zeros(2, max(size(X)));
for i = 1:max(size(X))
    g(:,i) = Dphi(X(:,i))*(Phii(X,P)-mat2vec(y));
end
end
```

7. Voici le code matlab :

```matlab
    function g2 = fhess(X, P, y, U)
t = 1e-4;
g2 = (fgrad(X+t*U,P,y)-fgrad(X,P,y))/t;
end
```

8. Voici le code matlab :

```matlab
    function h = h(x)
sigma = 0.1;
h = exp((- vecnorm(x).^2)/sigma^2);
h = h';
end
```

9. Voici le code matlab :

```matlab
    function alpha = linesearch(x, P, y, lsparams)
% Performs linesearch to find a step size.

% fhandle: Function handle returning the function value and the ...
   gradient at
% a given point.
% x: Point where to perform the linesearch.
% v: Search direction.
% lsparams: Structure containing parameters for the algorithm.

    alphabar = lsparams.alphabar;
    c = lsparams.c;
    rho = lsparams.rho;
    alphamin = lsparams.alphamin;

    alpha = alphabar;

    % ...
    f1 = f(x,P,y);
    g = fgrad(x,P,y);
    f2 = f(x,P,y);



    % the condition used for the while loop is the one from the ...
       Algorithm
    % 3.2 in the lecture notes
    while 0 > (f1-f2)-(c*alpha*norm(mat2vec(g))^2);
        % ...

            alpha = rho * alpha;
            f2 = f(x-alpha*g, P, y);

    end

    %if(rho * alpha < alphamin)
        %warning('Backtracking minimum step size reached!');
    %end
```

```
end
```

10. Voici le code matlab :

```
    function [Xk, gradnorms, times] = lsgradientdescent(x0, P, y, ...
        params, lsparams)
% Gradient descent with linesearch to find the step size.
% For this function you can reuse most of your code from the fixed step
% sizes version. The main difference is that you will call ...
    'linesearch' to
% find the step size.

% fhandle: Function handle returning the function value and the ...
    gradient at
% a given point.
% x0: Initial point.
% params: Structure containing parameters for algorithm.
% lsparams: Structure containing parameters for the linesearch.

    maxiters = params.maxiters;
    maxtime = params.maxtime;
    tolgradnorm = params.tolgradnorm;

    % it is basicaly the same code from the constant GD but we added the
    % line-search for alpha at each iteration of the while loop

    gradnorms = zeros(1, maxiters + 1);
    times = zeros(1, maxiters + 1);
    iter = 0;
    xk = x0;
    alpha = linesearch(x0, P, y, lsparams);
    f0 = f(x0, P, y);
    g0 = fgrad(x0, P, y);
    init_grad_norm = norm(mat2vec(g0));

    tic
    while (iter < maxiters && toc < maxtime)
        iter = iter + 1;
        fk = f(xk, P, y);
        gk = fgrad(xk, P, y);
        xk = xk - alpha*gk;
        alpha = linesearch(xk, P, y, lsparams);
        grad_norm = norm(mat2vec(gk));
        gradnorms(iter) = grad_norm;
        times(iter) = toc;
        if grad_norm < init_grad_norm * tolgradnorm
            break
        end
    end

    Xk = xk;
    gradnorms = gradnorms(1:iter);
    times = times(1:iter);
end
```

11. Voici le code matlab :

```matlab
    function [thetaTR, gradnormsTR, xk] = TR(x0, P, y, rho, Δ, Δ0, ...
        tol, maxiter)
gradnormsTR = zeros(1, maxiter + 1);
thetaTR = zeros(1, maxiter + 1);
xk = x0;
iter = 0;
Δk = Δ0;
%fx = f(xk,P,y);
g = fgrad(xk,P,y);



init_grad_norm = norm(g, 2);

while (iter < maxiter)
    iter = iter+1;
    fx = f(xk,P,y);
    g = fgrad(xk,P,y);
    handle = @(u) fhess(xk, P, y, u);
    n = maxiter;
    [uk, Huk] = tCG(handle, -g, Δk, n);
    xk_plus = xk +uk;
    m = @(v) fx + mat2vec(g)'*mat2vec(v) + ...
        0.5*mat2vec(v)'*mat2vec(handle(v));
    f_plus = f(xk_plus,P,y);
    %g_plus = fgrad(xk_plus,P,y);

    rhok = (fx-f_plus)/(m(zeros(2,max(size(xk))))-m(uk));
    if rhok > rho
        xk = xk_plus;
    else
        xk = xk;
    end
    if rhok < 1/4
        Δk = 0.25*Δk;
    elseif ((rhok > 3/4) && (norm(mat2vec(uk)) == Δk))
        Δk = min(2*Δk, Δ);
    else
        Δk = Δk;
    end
    gradnormsTR(iter) = norm(mat2vec(g));
    thetaTR(iter) = f(xk,P,y);
    if gradnormsTR(iter) < init_grad_norm * tol
            break
    end
end
gradnormsTR = gradnormsTR(1:iter);
thetaTR = thetaTR(1:iter);
end
```

12. Voici le code matlab :

```matlab
    function x = mat2vec(X)
% Stacks the columns of a matrix X to make a vector.
    [m, n] = size(X);
    x = reshape(X, [m*n 1]);
end
```

13. Voici le code matlab :

```
    function y = phi(x,P)
Pminus = P-x;
y = zeros(max(size(P)),1);
y = h(Pminus);

end
```

14. Voici le code matlab :

```
    function y = Phii(X, P)
y = zeros(max(size(P)),1);
for i = 1:max(size(X))
    y = y+phi(X(:,i),P);
end
end
```

15. Voici le code matlab :

```
    function plotimage(X)
    [m, n] = size(X);
    if m ≠ n
        warning('plotimage: Input matrix is not square.');
    end
    x = linspace(-.5, .5, m);
    y = linspace(-.5, .5, n);
    figure;
    imagesc(x, y, X);
    colorbar;
    axis equal;
    axis tight;
    drawnow;
end
```

16. Voici le code matlab :

```
    function x = mat2vec(X)
% Stacks the columns of a matrix X to make a vector.
    [m, n] = size(X);
    x = reshape(X, [m*n 1]);
end
```

17. Voici le code matlab :

```
    function [u, Hu] = tCG(handle, b, Δ, maxiter)
v = zeros(2,length(b));
r = b;
p = r;

for i=1:maxiter
    Hp = handle(p);
```

```
    dots = dot(mat2vec(p), mat2vec(Hp));
    alpha = norm(mat2vec(r))^2/dots;
    v_plus = v + alpha*p;
    if ((dots ≤ 0)||(norm(mat2vec(v_plus)) ≥ Δ))
        t = ...
            (-2*dot(mat2vec(p),mat2vec(v))+sqrt(4*dot(mat2vec(p),mat2vec(v))^2-4*no
        v = v + t*p;
        u = v;
        Hu = b-r+t*Hp;
        break
    else
        v = v_plus;
    end
    r_min = r;
    r = r_min - alpha*Hp;
    if norm(mat2vec(r)) ≤ norm(mat2vec(b))*min(norm(mat2vec(b)),0.1)
        u = v;
        Hu = b-r;
        break
    end
    beta = norm(mat2vec(r))^2/norm(mat2vec(r_min))^2;
    p = r+beta*p;
end
u = v;
Hu = b-r;
end
```

18. Voici le code matlab :

```
    function X = vec2mat(x, m, n)
% Reshapes a vector x into a m*n matrix.
% If m and n are not given the it tries to make a square matrix.
    if nargin == 1
        m = sqrt(numel(x));
        if mod(m, 1) ≠ 0
            error('vec2mat: Cannot make a square matrix from the ...
                input vector.')
        end
        n = m;
    end
    X = reshape(x, m, n);
end
```