Optimization and Algorithms
Project report

Group 42
José Neves 89683, Leonardo Pedroso 89691, and Gustavo Bakker 100660

1 Part 3

In this part, the goal is to solve

$$\underset{\mathbf{y}\in\mathbf{R}^{Nk}}{\text{minimize}} \quad \sum_{m=1}^{N}\sum_{n=m+1}^{N}\left(||\mathbf{y_m}-\mathbf{y_n}||_2 - D_{mn}\right)^2, \tag{1}$$

where $\mathbf{D}\in\mathbb{R}^{N\times N}$.

1.1 Task 1

The dataset in file `data_opt.csv` is loaded and the corresponding matrix D is computed according to $D_{mn} = ||\mathbf{x_m}-\mathbf{x_n}||_2$. The following MATLAB script solves task 1

```
%% Part 3 - Task 1 (part3task1.m)
%% Load dataset from file data_opt.csv
X = csvread("./data/data_opt.csv");
N = size(X,1); % Get number of datapoints
%% Compute matrix D
D = zeros(N); % Initialize matrix D
for m = 1:N % Four each off-diagonal pair of coordinates
    for n = m+1:N
        D(m,n) = norm(X(m,:)-X(n,:),2); % Compute D_{mn}
        D(n,m) = D(m,n); % D_{nm}=D_{mn}
    end
end
%% Check results
% Find maximum value of distance and repective indices
Dmax = max(max(D));
[mDmax,nDmax] = find(D==Dmax);
% Output results
fprintf("--------------------------- Task 1 ---------------------------\n");
fprintf("D(2,3) = %g | D(4,5) = %g.\n", D(2,3),D(4,5));
fprintf("max{D(m,n)} = %g for (m,n) = {(%d,%d),(%d,%d)}.\n",...
    Dmax,mDmax(1),nDmax(1),mDmax(2),nDmax(2))
```

```
%% Save data
save("./data/distancesTask1.mat",'D','Dmax','nDmax','mDmax','N');
```

obtaining

$$D_{2,3} = 5.8749, \quad D_{4,5} = 24.3769$$

and

$$\max(D_{mn}) = 83.003 \quad \text{for} \quad (m,n) \in \{(134,33),(33,134)\} .$$

1.2 Task 2

One has

$$f(\mathbf{y}) = \sum_{m=1}^{N}\sum_{n=m+1}^{N} (||\mathbf{y_m} - \mathbf{y_n}|| - D_{mn})^2 = \sum_{m=1}^{N}\sum_{n=m+1}^{N} f_{mn}(\mathbf{y})^2 , \tag{2}$$

where $\mathbf{y_m} \in \mathbb{R}^m$, k is the dimension of the target space, $\mathbf{y} = \text{col}(\mathbf{y_1}, \dots, \mathbf{y_N}) \in \mathbb{R}^{Nk}$ is the optimization variable, and

$$f_{mn}(\mathbf{y}) := ||\mathbf{y_{m-n}}|| - D_{mn} , \tag{3}$$

defining $\mathbf{y_{m-n}}$ as $\mathbf{y_{m-n}} := \mathbf{y_m} - \mathbf{y_n}$.

Note that one can write $\mathbf{y_m} = \mathbf{E_m}\mathbf{y}$, where $\mathbf{E_m} \in \mathbb{R}^{k\times Nk}$ is defined as

$$\mathbf{E_m} := \begin{bmatrix} \mathbf{0}_{k\times k(m-1)} & \mathbf{I}_{k\times k} & \mathbf{0}_{k\times k(N-m)} \end{bmatrix} ,$$

thus, it is possible to rewrite (3) as

$$f_{mn}(\mathbf{y}) = ||\mathbf{E_m y} - \mathbf{E_n y}|| - D_{mn} = \sqrt{\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}} - D_{mn} . \tag{4}$$

Taking the jacobian of (4), one obtains

$$\begin{aligned} D_{\mathbf{y}} f_{m,n}(\mathbf{y}) &= D_u(\sqrt{u})\Big|_{u=\mathbf{y}^T(\mathbf{E_m}-\mathbf{E_n})^T(\mathbf{E_m}-\mathbf{E_n})\mathbf{y}} D_{\mathbf{y}}(\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}) \\ &= \mathbf{y}^T \frac{(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})}{\sqrt{\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}}} \\ &= \frac{\begin{bmatrix} \mathbf{0}_{1\times(m-1)k} & \mathbf{y_{m-n}}^T & \mathbf{0}_{1\times(n-m-1)k} & -\mathbf{y_{m-n}}^T & \mathbf{0}_{1\times(N-n)k} \end{bmatrix}}{||\mathbf{y_{m-n}}||} \end{aligned} \tag{5}$$

therefore the gradient $\nabla_{\mathbf{y}} f_{mn}(\mathbf{y}) = (D_{\mathbf{y}} f_{mn}(\mathbf{y}))^T$.

Similarly taking the jacobian of (2), one obtains

$$D_{\mathbf{y}} f(\mathbf{y}) = \sum_{m=1}^{N}\sum_{n=m+1}^{N} D_u(u^2)\Big|_{u=f_{mn}(\mathbf{y})} D_{\mathbf{y}} f_{mn}(\mathbf{y}) = \sum_{m=1}^{N}\sum_{n=m+1}^{N} 2f_{mn}(\mathbf{y}) D_{\mathbf{y}} f_{mn}(\mathbf{y})$$

therefore the gradient $\nabla_{\mathbf{y}} f(\mathbf{y}) = (D_{\mathbf{y}} f(\mathbf{y}))^T$ is given by

$$\nabla_{\mathbf{y}} f(\mathbf{y}) = \sum_{m=1}^{N}\sum_{n=m+1}^{N} 2f_{mn}(\mathbf{y}) \nabla_{\mathbf{y}} f_{mn}(\mathbf{y}) \tag{6}$$

In conclusion, $f(\mathbf{y})$, $f_{mn}(\mathbf{y})$,$\nabla_{\mathbf{y}} f_{mn}(\mathbf{y})$, and $\nabla_{\mathbf{y}} f(\mathbf{y})$ can be computed making use of (2),(3), (5), and (6), respectively. Also note that each of these four quantities may be computed making use of differences of the optimization vector exclusively, *i.e.*, the $N(N/2-1)$ vectors $\mathbf{y_{m-n}}$. As it is explored herein this property allows for considerable optimization of the computational load required to solve the optimization problem.

For the implementation of the Levenberg-Marquardt (LM) method is is required to compute, for each new iteration, $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, matrix $\mathbf{A}$, and vector $\mathbf{b}$, defined by

$$\mathbf{A} := \begin{bmatrix} D_{\mathbf{y}} f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}} f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}} f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda}\mathbf{I}_{Nk\times Nk} \end{bmatrix} \quad \text{and} \quad \mathbf{b} := \begin{bmatrix} D_{\mathbf{y}} f_{1,1}(\mathbf{y})\mathbf{y} - f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}} f_{1,2}(\mathbf{y})\mathbf{y} - f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}} f_{N-1,N}(\mathbf{y})\mathbf{y} - f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda}\mathbf{y} \end{bmatrix} . \tag{7}$$

For that purpose a MATLAB function independent of the LM algorithm is devised. This allows to implement the LM algorithm separately, which can then be applied to any optimization problem of suitable form, and not being constrained to the problem at hand in this part. To allow for a computationally efficient algorithm to compute, at each iteration, the relevant quantities related to the objective function, first note that

$$D_{\mathbf{y}} f_{mn}(\mathbf{y})\mathbf{y} - f_{mn}(\mathbf{y}) = \frac{\mathbf{y_{m-n}}(\mathbf{y_m} - \mathbf{y_n})}{||\mathbf{y_{m-n}}||} - ||\mathbf{y_{m-n}}|| + D_{mn} = D_{mn} \, . \tag{8}$$

Thus, noticing that $\mathbf{b}$ in (7) results of the concatenation of terms of the form (8), $\mathbf{b}$ is computed according to

$$\mathbf{b} = \begin{bmatrix} D_{1,1} & D_{1,2} & \dots & D_{N-1,N} & \sqrt{\lambda}\mathbf{y} \end{bmatrix} , \tag{9}$$

which is very efficiently computed since it does not have to carried out iteratively. Furthermore, a significant portion of $\mathbf{b}$ is constant, which has to be computed just once in the LM algorithm. Second, the computation of $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, and $\mathbf{A}$ is performed iteratively, running one iteration for each of the $N(N/2-1)$ vectors $\mathbf{y_{m-n}}$. Therefore, it is more efficient to compute all of these quantities at once. Third, λ is a variable of the LM method, which does not depend directly on the objective function, thus, it was chosen that the entries of $\mathbf{A}$ and $\mathbf{b}$ which are dependent on λ are computed in the LM algorithm function. Fourth, an effort was made so that there is not replication of computations. Given that, the quantities computed depend essentially on each other, $\mathbf{y_{m-n}}$, or $||\mathbf{y_{m-n}}||$, then this allows to reduce the computational load significantly.

Having the aforementioned optimization guidelines in mind the following MATLAB function was designed

```
function [costF,normG,A,b] = objectiveF(y)
%% objectiveF.m
```

```
% Input: y: vector at which the quatities are evaluated
% Ouput: costF: cost function value
%        normG: norm of the gradient of the cost function
%        A: matrix A for the application of the LM method (only the entries
%        that do not depend on lambda)
%        b: vetor b for the application of the LM method (only the entries
%        that do not depend on lambda)
%% Initialize cost function dataset
% Load dataset in the first call to this function
persistent b_ N k % do not have to be recomputed between calls
if isempty(k) ||isempty(b_) || isempty(N)
    % Load data:
    % D: Distance matrix
    % N: Number of data points
    % k: Dimension of the target space
    load("./data/objectiveFData.mat",'D','N','k');
    % Compute the portion of b that is constant
    b_ = nonzeros(tril(D,-1));
    fprintf("Initializing dataset.\n");
end
%% Compute quantities
costF = 0; % Initialize costF
gradf = zeros(1,N*k); % Initialize gradf
A = zeros((N^2-N)/2+N*k,N*k); % Initialize A
b = [b_;zeros(N*k,1)]; % Compute entries of b that do not depend on lambda
count = 1; % Iteration count
for i = 1:N
   for j = i+1:N
       dy = (y((i-1)*k+1:(i-1)*k+k)-y((j-1)*k+1:(j-1)*k+k))'; % y_{m-n}
       normdy = norm(dy); % ||y_{m-n}||
       faux = normdy-b_(count); % f_{mn} = ||y_{m-n}||-D_{mn}
       gradaux_ = [zeros(1,(i-1)*k) dy zeros(1,k*(j-i-1))...
           -dy zeros(1,(N-j)*k)]/(normdy); % D_y(f{mn}(y))
       costF = costF + faux^2; % f(y) += f_{mn}(y)^2
       % D_y(f(y))(y) += 2*f_{mn}(y)*D_y(f{mn}(y))
       gradf = gradf + 2*faux*gradaux_;
       A(count,:) = gradaux_; % A(<->) = D_y(f{mn}(y))
       count = count+1; % increment iteration count
   end
end
normG = norm(gradf); % compute norm of D_y(f(y))(y)
end
```

This implementation allows for a decrease of computation time of roughly two orders of magnitude compared with a first naive implementation.

1.3 Task 3

In this task the optimization problem is solved for the dataset loaded in Task 1, for $k \in \{2, 3\}$, using the LM algorithm. For that reason, a generic implementation of this algorithm was implemented in MATLAB

```
function [xk,k,costF,normGk] = LMAlgorithm(lambda0,x0,epsl,maxIt)
%% LMAlgorithm.m
% Input: lambda0: initialization lambda of the LM algorithm
%         x0: initialization solution estimate
%         epsl: stopping criterion
%         maxIt: maximum number of iterations
% Output: xk: output of the gradient descent method (returns NaN if
%         stopping criterion not met after the maximum number of
%         iterations chosen
%         k: number of iterations required for convergence if a
%         solution was found
%         costF: vector of objective function value for each iteration
%         normGk: vector of the norm of the gradient of the objective
%         function for each iteration
%% LM algorithm
% Initialize variables
costF = zeros(maxIt,1); % vector of objective function value
normGk = zeros(maxIt,1); % vector of gradient norms of the objective f
% Initialize LM algorithm
k = 0; % Initialize iteration count
xk = x0; % Initialization solution estimate
lambdak = lambda0; % Initialization lambda
fprintf("Running LM.\n");
% ---------------------------- LM algorithm ----------------------------
while k < maxIt % iterate up to a limit of maxIt iterations
    % ----------- Compute objective function and gradients -----------
    if k % store previous A and b, which are used if the step is invalid
        Aprev = A;
        bprev = b;
    end
    % Compute objective f value, norm of the gradient, A and b for xk
    % note that only the entries of A and b that do not depend on lambda
    % are computed
    [costF(k+1),normGk(k+1),A,b] = objectiveF(xk);
    % ------------------ check stopping criterion --------------------
    if normGk(k+1) < epsl % Stopping criterion¥
            break;
    end
    % ---------------- Check validity of last step ------------------
    if k && costF(k+1)<costF(k) % if step is valid
        lambdak = 0.7*lambdak; % decrease lambda
    elseif k % if step is invalid
        xk = xprev; % redo step
        A = Aprev;
        b = bprev;
        normGk(k+1) = normGk(k);
        costF(k+1) = costF(k);
        lambdak = 2*lambdak; % decrease lambda
```

```
    end
    % ------------- New step (solve least squares problem) -------------
    % store previous estimate, which is used if the step is invalid
    xprev = xk;
    % Entries of A and b that depend on lambda must be computed
    A(end-size(x0,1)+1:end,:) = sqrt(lambdak)*eye(size(x0,1));
    b(end-size(x0,1)+1:end,1) = sqrt(lambdak)*xk;
    xk = ((A'*A)\A')*b; % solve least squares problem Ax=b

    % ------------------- Increment iteration count --------------------
    % Display LM status
    if k fprintf("Iteration: %d | cost = %g.\n",k,costF(k+1)); end
    k = k+1; % increment iteration count
end
if k == maxIt
    % No solution found within the maximum number of iterations
    xk = NaN; % Output invalid estimate
else
    % Output only relevant data
    costF = costF(2:k+1);
    normGk = normGk(2:k+1);
end
end
```

Note that this implementation relies on the fact that $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, matrix $\mathbf{A}$, and vector $\mathbf{b}$ are computed at once. It is important to remark, however, that each time a step is invalid, the new $\mathbf{A}$ and $\mathbf{b}$ that were computed are useless, which represents a waste of computational power. It was verified that, for this particular optimization problem, reduction that is achieved computing all quantities at once is greater than that obtained if $\mathbf{A}$ and $\mathbf{b}$ are only computed when necessary.

The following MATLAB script was then run to solve the optimization problem

```
%% Part 3 - Task 3 (part3task3.m)
%% Initialize cost function dataset
% Load distances matrix of the dataset of task 1
load("./data/distancesTask1.mat",'D','N');
% Initialize variables to hold the solution and status parameters of the LM
% algorithm for K = 2,3
solLM = cell(2,1); % solution of the optimization problem
itLM = zeros(2,1); % number of iterations ran
elapsedTimeLM = zeros(2,1); % time elapsed running LM
costLM = cell(2,1); % vector of cost function values for each iteration
% vector of gradient norm of the cost function for each iteration
normGradLM = cell(2,1);

%% Solve optimization problem for k = 2,3
for k = 2:3 % target space dimension
    % Set up parameters
    maxIt = 200; % maximum number of iterations
```

```
    lambda0 = 1; % initial value for lambda of the LM method
    epsl = k*1e-2; % stopping criterion
    % set up data for the compuattion of the quatities related to the
    % objective function in objectiveF(y)
    save("./data/objectiveFData.mat",'D','N','k');
    y0 = csvread(sprintf("./data/yinit%d.csv",k)); % initialization of LM
    clear objectiveF; % clear persistent variables in objectiveF
    fprintf("------------------------- Task 3 -------------------------\n");
    tic; % start counting LM time
    % run LM method
    [solLM{k-1,1},itLM(k-1,1),costLM{k-1,1},normGradLM{k-1,1}] =...
        LMAlgorithm(lambda0,y0,epsl,maxIt);
    elapsedTimeLM(k-1,1) = toc; % save elapsed time
    if ¬isnan(solLM{k-1,1}) % if a solution was found
        fprintf("Solution found for dataset of task 1 with k = %d "+...
            "using LM algorithm.\n",k);
        fprintf("- Objective function value: %g.\n",costLM{k-1,1}(end,1));
        fprintf("- Elapsed time: %g s.\n", elapsedTimeLM(k-1,1));
    else % if a solution was not found
        fprintf("Solution could not be found for dataset of task 1 "+...
            "with k = %d using\n LM algorithm with the provided "+...
            "stopping criterion and maximum number of iterations.\n",k);
    end
end
% Save solutions
save("./data/solTask3.mat",...
    'solLM','itLM','elapsedTimeLM','costLM','normGradLM');
%% Plot results
for k = 2:3
    figure('units','normalized','outerposition',[0 0 1 1]);
    yyaxis left
    plot(0:itLM(k-1,1)-1,costLM{k-1,1},'LineWidth',3);
    hold on;
    ylabel('$f(y)$','Interpreter','latex');
    set(gca, 'YScale', 'log');
    yyaxis right
    plot(0:itLM(k-1,1)-1,normGradLM{k-1,1},'LineWidth',3);
    ylabel('$||\nabla f (y)||$','Interpreter','latex');
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
    set(gca, 'YScale', 'log');
    xlabel('$k$','Interpreter','latex');
    saveas(gcf,sprintf("./data/task3_LM_k_%d.fig",k));
    hold off;
    y = reshape(solLM{k-1},[k,N]);
    figure('units','normalized','outerposition',[0 0 1 1]);
    if k == 2
        scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,...
```

```
                'MarkerFaceColor','flat');
        else
            scatter3(y(1,:),y(2,:),y(3,:),100,'o','b','LineWidth',1,...
                'MarkerFaceColor','flat');
        end
        hold on;
        set(gca,'FontSize',35);
        ax = gca;
        ax.XGrid = 'on';
        ax.YGrid = 'on';
        title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
        saveas(gcf,sprintf("./data/task3_lowerDim_k_%d.fig",k));
        hold off;
    end
```

.

The results obtained for $k = 2$ are shown in Figs. 1 and 2, and for $k = 3$ in Figs. 3 and 4.

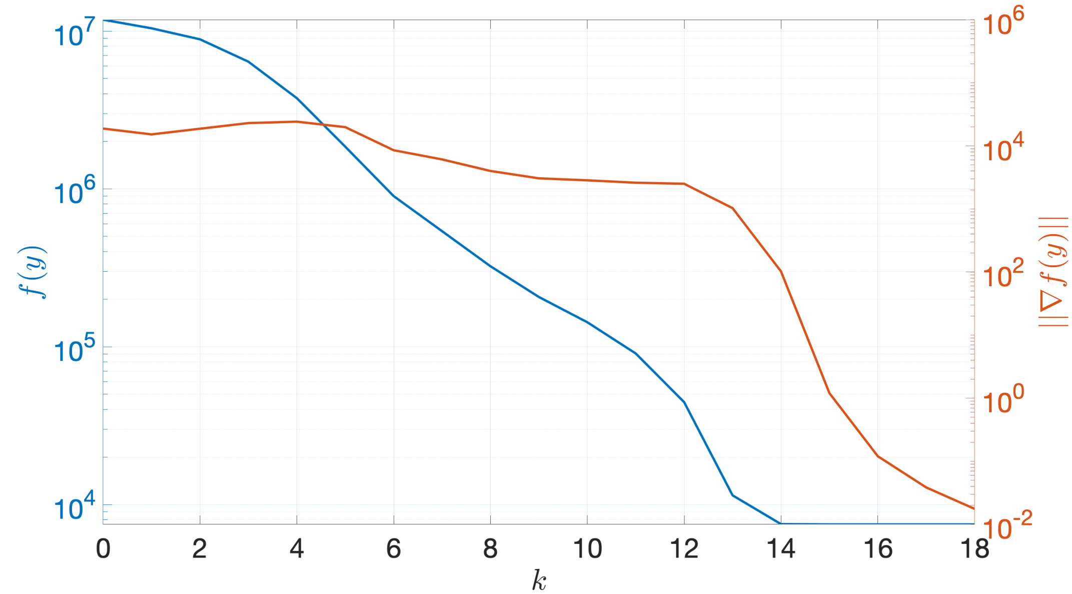

Figure 1: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

First, it is noticeable that the algorithm converges and reaches the expected solution for both values of k. Second, the value of the objective function of both solutions is shown in Table 1. It is visible that the value of the cost function for $k = 3$ decreases by a factor of roughly 2.7 in relation to the solution with $k = 2$. Thus, $k = 3$ fits much better to the dataset. Third, note that the solution using $k = 3$ requires more iterations of the LM algorithm than what is presented in the provided results. In fact, observing the evolution of the norm of the gradient of the objective function, visible in Fig. 3, it is possible to detect

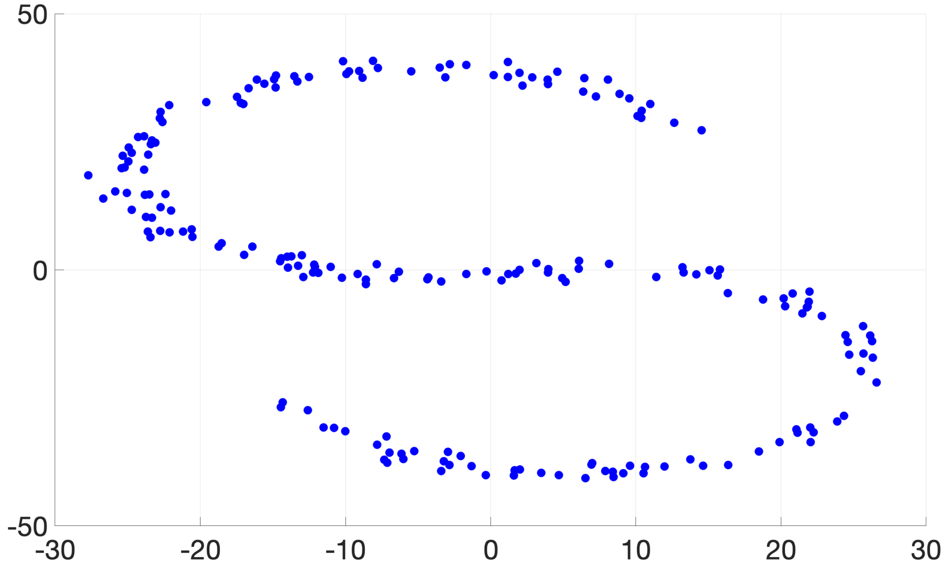

Figure 2: Solution to the optimization problem using the LM algorithm for $k = 2$.

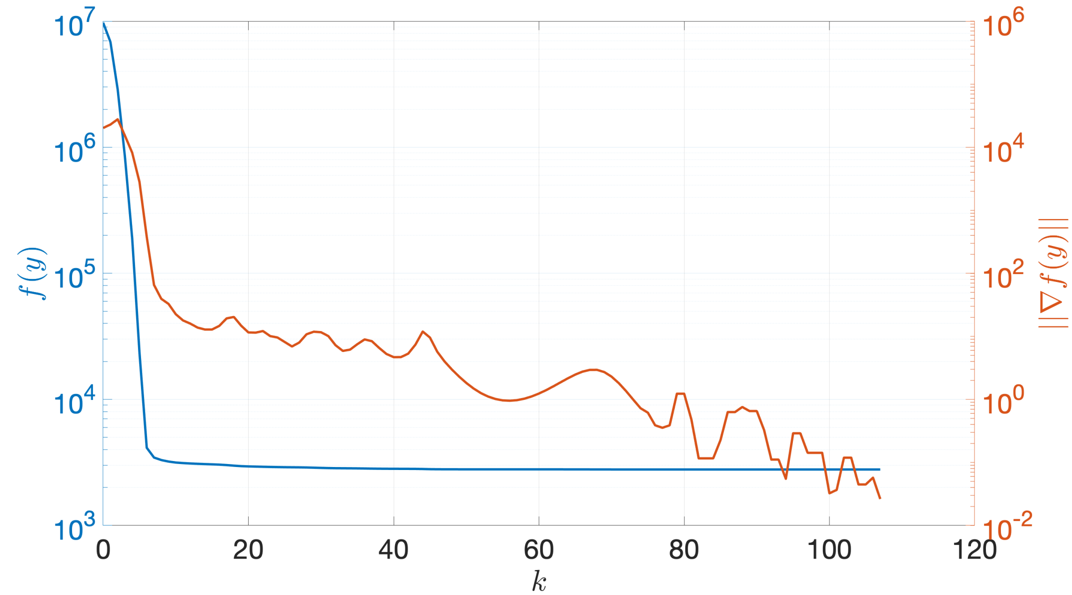

Figure 3: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

the presence of numerical error stating at the 70-th iteration, which arises using MATLAB 2018a. For this reason, although the solution is identical, it takes more iterations to reach the stopping criterion.

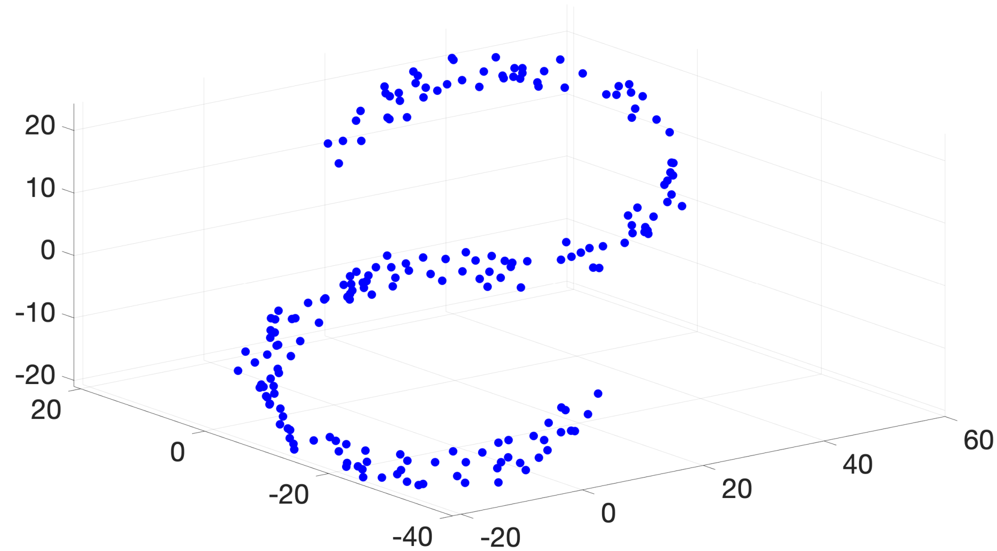

Figure 4: Solution to the optimization problem using the LM algorithm for $k = 2$.

Table 1: Value of the objective function of both solutions.

k	$f(\mathbf{y})$
2	7486.6
3	2779.2

1.4 Task 4

The LM method is now applied to dataset`dataProj.csv`, and we are not provided with an initialization $\mathbf{y}_0$. There is no guarantee that a solution found by the LM method is the global solution, since the objective function is not convex. For this reason, to find a good suboptimal solution, the method has to be run several times for different randomly generated initializations. The solutions are then sorted according to the value of the objective function, of which the best is chosen. It is very important to remark that the computation of each of the solutions can be run in a parallel manner, using the *Parallel toolbox* in MATLAB, for instance. The following MATLAB script solves task 4.

```
%% Part 3 — Task 4 (part3task4.m)
% Various runs can be performed. In each run the LM algorithm is run NIts
% times all for randmly generated initializations
%% Set parameters
RUN = 1; % run number
NRuns = 1; % number of runs so far
NIts = 12*2; % number of times the LM algorith is called in a run
%% Load or compute data
```

```
computedD = false; % if D is already computed just load it
if ¬computedD % C was not computed -> compute it now
    X = csvread("./data/dataProj.csv");
    N = size(X,1);
    D = zeros(N);
    for m = 1:N
        for n = m+1:N
           D(m,n) = norm(X(m,:)-X(n,:),2);
           D(n,m) = D(m,n);
        end
    end
    % Save data
    save("./data/distancesTask4.mat",'D','N');
else % D was already computed -> compute it now
    load("./data/distancesTask4.mat",'D','N');
end
%% Run various times LM for random initializations
% Set up parameters
k = 2; % target space dimension
lambda0 = 1; % initial value for lambda of the LM method
maxIt = 200; % maximum number of iterations
epsl = k*1e-4; % stopping criterion
% Initialize variables to hold the solution and status parameters of the LM
% algorithm for the NIts LM calls
solLM = cell(NIts,1); % solution of the optimization problem
itLM = zeros(NIts,1); % number of iterations ran
elapsedTimeLM = zeros(NIts,1); % time elapsed running LM
costLM = cell(NIts,1); % vector of cost function values for each iteration
% vector of gradient norm of the cost function for each iteration
normGradLM = cell(NIts,1);
fprintf("------------------------ Task 4 ------------------------\n");
clear objectiveF; % clear persistent variables in objectiveF
save("./data/objectiveFData.mat",'D','N','k');
parfor it = 1:NIts % calls of LM can be run in parallel
    % each entry of y is randomly generated from an uniform distribution
    % between -200 and 200
    y0 = 200*2*(rand()-0.5)*2*(rand(N*k,1)-0.5);
    tic; % start counting LM time
    fprintf("------------------ RUN %02d - Attempt %02d -----------------\n",...
        RUN,it);
     % run LM method
    [solLM{it,1},itLM(it,1),costLM{it,1},normGradLM{it,1}] =...
        LMAlgorithm(lambda0,y0,epsl,maxIt);
    elapsedTimeLM(it,1) = toc; % save elapsed time

%     if ¬isnan(solLM{it,1}) % if a solution was found
%         fprintf("Solution found for dataset of task 4 with k = %d using LM algorithm.\
%         fprintf("- Objective function value: %f.\n",costLM{it,1}(end,1));
%         fprintf("- Elapsed time: %g.\n", elapsedTimeLM(it,1));
%     else % if a solution was not found
%         fprintf("Solution could not be found for dataset of task 1 with "+...
```

```
%               "k = %d using\n LM algorithm with the provided stoppin criterion"+...
%               " and maximum number of iterations.\n",k);
%       end
end
% Save whole run
save(sprintf("./data/RunsTask4/solRUN%02d.mat",RUN),...
    'solLM','itLM','elapsedTimeLM','costLM','normGradLM');

%% Sort solutions found in all runs
solSorted = zeros(NRuns*NIts,3); % sorted list of all solutions
count = 0; % count number of solutions
for i = 1:NRuns
   data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",i));
   for j = 1:NIts
       count = count+1;
       solSorted(count,1) = i; % 1st column has run number
       solSorted(count,2) = j; % 2nd column has attempt number within run
       solSorted(count,3) = data.costLM{j,1}(end,1);  % 2nd column has cost
   end
end
solSorted = sortrows(solSorted,3); % sort rows ascending cost
save("./data/solSortedTask4.mat",'solSorted'); % save sorted solutions

% Best solution
% Load best solution run
data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",solSorted(1,1)));
% Get best solution data and save it
solLM = data.solLM{solSorted(1,2),1};
itLM = data.itLM(solSorted(1,2),1);
elapsedTimeLM = data.elapsedTimeLM(solSorted(1,2),1);
costLM = data.costLM{solSorted(1,2),1};
normGradLM = data.normGradLM{solSorted(1,2),1};
% Save best solution data
save("./data/solBestTask4.mat",...
     'solLM','itLM','elapsedTimeLM','costLM','normGradLM');

%% Plot best solution
figure('units','normalized','outerposition',[0 0 1 1]);
yyaxis left
plot(0:itLM(k-1,1)-1,costLM,'LineWidth',3);
hold on;
ylabel('$f(y)$','Interpreter','latex');
set(gca, 'YScale', 'log');
yyaxis right
plot(0:itLM(k-1,1)-1,normGradLM,'LineWidth',3);
ylabel('$||\nabla f (y)||$','Interpreter','latex');
set(gca,'FontSize',35);
ax = gca;
ax.XGrid = 'on';
ax.YGrid = 'on';
title(sprintf("LM algorithm | Dataset task 4 | k = %d",k));
```

```
set(gca, 'YScale', 'log');
xlabel('$k$','Interpreter','latex');
saveas(gcf,"./data/task4_LM.png");
hold off;
y = reshape(solLM,[k,N]);
figure('units','normalized','outerposition',[0 0 1 1]);
scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,'MarkerFaceColor','flat');
hold on;
set(gca,'FontSize',35);
ax = gca;
ax.XGrid = 'on';
ax.YGrid = 'on';
title(sprintf("LM algorithm | Dataset task 4 | k = %d",k));
saveas(gcf,sprintf("./data/task4_sol.png"));
hold off;
```

.

The LM algorithm was run for 24 different randomly generated initialization vectors, in a parallel manner. The best solution, obtained for $k = 2$, is shown in Figs. 5 and 6, achieving an objective function value of $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$. First, the parallel computation of the various solutions allowed for a significantly faster computation. Second, it was verified that all solutions obtained have identical objective function values, which on its own does not imply that it is the global solution. Notice that the objective function is nonnegative. Furthermore, it is verified that if the stopping criteria parameter is lowered then this method yields an objective function value that it closer to zero. For this reason, as the order of magnitude of the entries of D is substantially greater than the order of magnitude of the objective function value at the solutions, either the problem is ill-conditioned or one of the global minimums (or the only global minimum) was found. Assuming the problem is not ill-conditioned, then, even though this claim is not theoretically correct, it is possible to assume in a practical sense that one approximation very close the global minimizer was found.

It is now important to analyze the uniqueness of the solutions. In fact consider

$$\bar{\mathbf{y}} = \mathrm{col}(\mathcal{T}\mathbf{y_1} + \mathbf{w}, \ldots, \mathcal{T}\mathbf{y_N} + \mathbf{w})\,,$$

where $\mathcal{T} \in \mathbb{R}^{k\times k}$ is a rotation matrix and $\mathbf{w} \in \mathbb{R}^k$. It is easily verified that

$$||\bar{\mathbf{y}}_\mathbf{m} - \bar{\mathbf{y}}_\mathbf{n}|| = ||\mathbf{y_m} - \mathbf{y_n}||$$

for every pair $(m, n) : m \in \{1, \ldots, N\}, n \in \{1, \ldots, N\}$. Therefore

$$f(\bar{\mathbf{y}}) = f(\mathbf{y})\,.$$

It is, then, evident that if a global minimum is found, there are infinitely many other global minimums obtained via a rotation and translation of every $\mathbf{y_i}$, $i \in \{1, \ldots, N\}$. For this reason the previously shown solution, conjectured to be an estimate of a global solution, is not unique. In fact, analyzing the solution of the second best solution found, out of the 24 computation, shown in Fig. 7. In fact, even though both solutions achieve the same objective

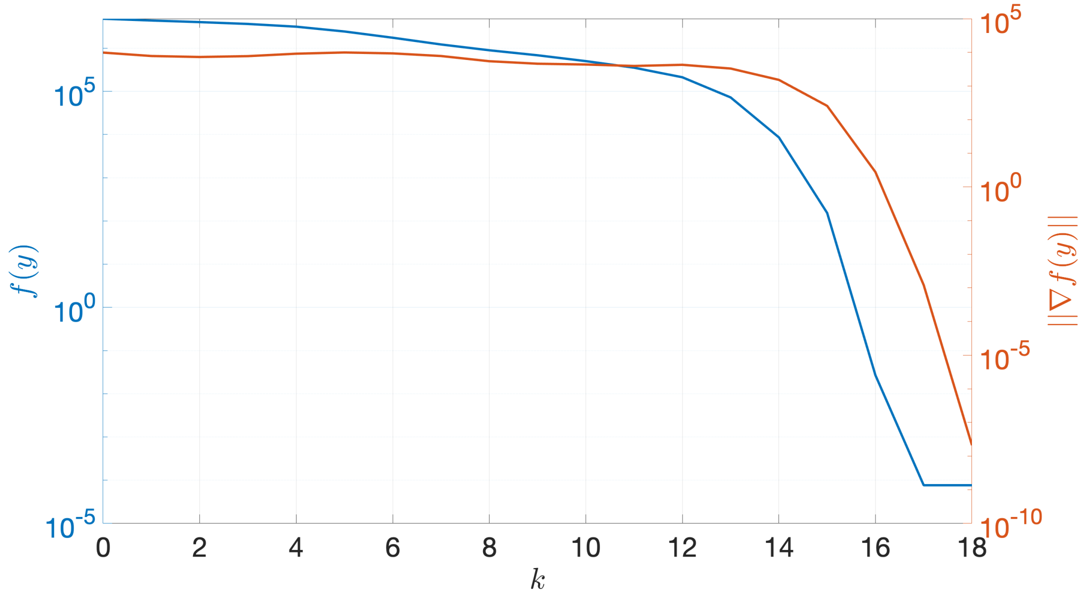

Figure 5: Objective function value and gradient norm throughout the iterations of the LM algorithm for the dataset of task 4 and $k = 2$.

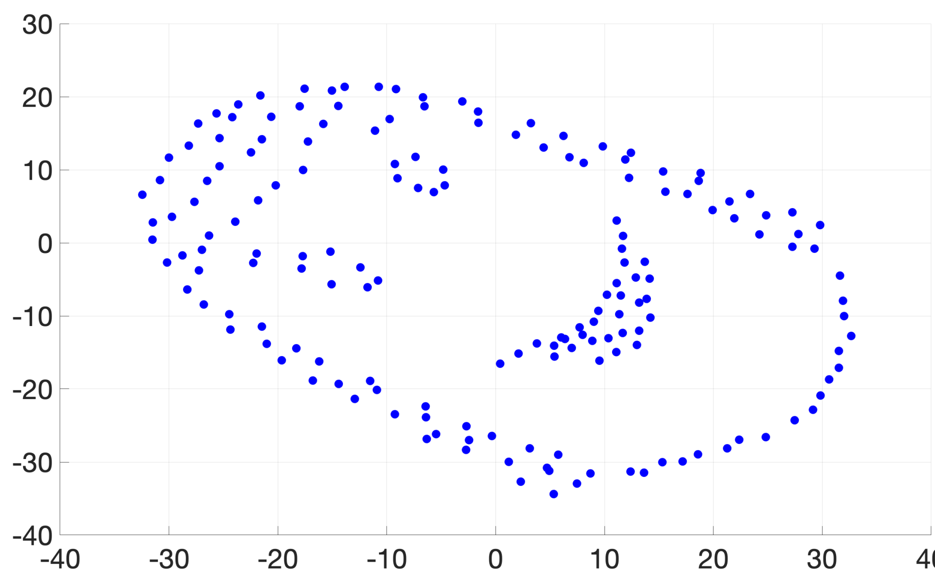

Figure 6: Best solution to the optimization problem using the LM algorithm for the dataset of task 4 and $k = 2$, obtaining $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$.

function value $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$, they are the result of a rotation and translation of each other.

The following question now arises naturally: if the additional degrees of freedom (d.o.f.)

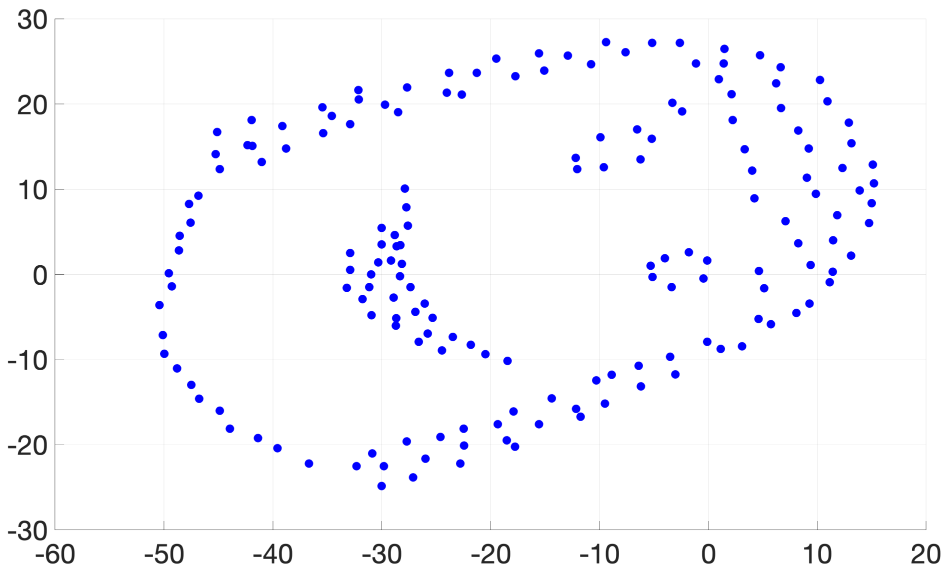

Figure 7: Second best solution to the optimization problem using the LM algorithm for the dataset of task 4 and $k = 2$, obtaining $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$.

of the rotation ($k-1$ d.o.f.) and of the translation (k d.o.f) are suppressed, constraining the optimization problem, is a solution obtained unique? First, the optimization problem can be constrained imposing

$$\mathbf{y_1} = \mathbf{0}_{k\times 1} \quad \text{and} \quad \mathbf{y_2} = \alpha \mathrm{col}(1, \mathbf{0}_{(k-1)\times 1})\,,$$

with $\alpha \in \mathbb{R}$. The optimization problem becomes

$$\begin{array}{cl} \underset{(\alpha, \mathbf{y_3}, \ldots, \mathbf{y_N}) \in \mathbb{R} \times \mathbf{R}^k \times \ldots \times \mathbf{R}^k}{\text{minimize}} & (\alpha - D_{1,2})^2 + \sum_{n=3}^{N} \left(||\mathbf{y_n}||_2 - D_{mn} \right)^2 \\ & + \sum_{n=3}^{N} \left(||\alpha \mathrm{col}(1, \mathbf{0}_{(k-1)\times 1}) - \mathbf{y_n}||_2 - D_{mn} \right)^2 \\ & + \sum_{m=3}^{N} \sum_{n=m+1}^{N} \left(||\mathbf{y_m} - \mathbf{y_n}||_2 - D_{mn} \right)^2 \end{array} , \tag{10}$$

with $\mathbf{y} = \mathrm{col}\left(\mathbf{0}_{k\times 1}, \alpha \mathrm{col}(1, \mathbf{0}_{(k-1)\times 1}), \mathbf{y_3} \ldots, \mathbf{y_N}\right) \in \mathbb{R}^{Nk}$, which is still nonconvex. In fact, it is easily proven that if a global solution is found, it is not necessarily unique. As a matter of fact, considering

$$\mathbf{D} = \begin{bmatrix} 0 & 4 & \sqrt{5} \\ 4 & 0 & 5 \\ \sqrt{5} & 5 & 0 \end{bmatrix}$$

both

$$\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \\ 1 \\ 4 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \\ 1 \\ -4 \end{bmatrix}$$

are solutions to (10), with $f(\mathbf{y}) = 0$. Notice that can not be obtained from each other via a rotation and translation. Therefore, even if the additional $2k-1$ degrees of freedom of the solutions to the original problem (1) are suppressed, there is no guarantee that if a global solution is found, it is unique. In fact, inspired by this example it is not difficult to notice that if every $\mathbf{y_m}$ is reflected on the axis corresponding to the first coordinate, then the objective function value remains unchanged. Even if this d.o.f. is suppressed via an additional constraint, we could not find any uniqueness guarantees.

In conclusion, given the thorough analysis conducted, the solution found is not unique. In fact, infinitely many solutions can be found via a translation, rotation, and/or reflection on the axis corresponding to the first coordinate.