

Optimization and Algorithms

Project report

Group 42

José Neves 89683, Leonardo Pedrosa 89691, and Gustavo Bakker 100660

1 Part 3

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

```
1 %% Part 3 – Task 1 (part3task1.m)
2 %% Load dataset from file data_opt.csv
3 X = csvread("./data/data_opt.csv");
4 N = size(X,1); % Get number of datapoints
5 %% Compute matrix D
6 D = zeros(N); % Initialize matrix D
7 for m = 1:N % Four each off-diagonal pair of coordinates
8     for n = m+1:N
9         D(m,n) = norm(X(m,:) - X(n,:), 2); % Compute D_{mn}
10        D(n,m) = D(m,n); % D_{nm}=D_{mn}
11    end
12 end
13 %% Check results
14 % Find maximum value of distance and repetitive indices
15 Dmax = max(max(D));
16 [mDmax, nDmax] = find(D==Dmax);
17 % Output results
18 fprintf("————— Task 1 —————\n");
19 fprintf("D(2,3) = %g | D(4,5) = %g.\n", D(2,3), D(4,5));
20 fprintf("max{D(m,n)} = %g for (m,n) = {( %d, %d), ( %d, %d) }.\n", ...
21        Dmax, mDmax(1), nDmax(1), mDmax(2), nDmax(2))
22 %% Save data
23 save("./data/distancesTask1.mat", 'D', 'Dmax', 'nDmax', 'mDmax', 'N');
```

obtaining

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

and

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

1.2 Task 2

One has

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

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

$$f_{mn}(\mathbf{y}) := \|\mathbf{y}_{m-n}\| - D_{mn}, \quad (2)$$

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 (2) as

$$f_{mn}(\mathbf{y}) = \|\mathbf{E}_m \mathbf{y} - \mathbf{E}_n \mathbf{y}\| - D_{mn} = \sqrt{\mathbf{y}^T (\mathbf{E}_m - \mathbf{E}_n)^T (\mathbf{E}_m - \mathbf{E}_n) \mathbf{y}} - D_{mn}. \quad (3)$$

Taking the jacobian of (3), one obtains

$$\begin{aligned} D_{\mathbf{y}} f_{m,n}(\mathbf{y}) &= D_u(\sqrt{u}) \Big|_{u=\mathbf{y}^T (\mathbf{E}_m - \mathbf{E}_n)^T (\mathbf{E}_m - \mathbf{E}_n) \mathbf{y}} D_{\mathbf{y}} (\mathbf{y}^T (\mathbf{E}_m - \mathbf{E}_n)^T (\mathbf{E}_m - \mathbf{E}_n) \mathbf{y}) \\ &= \mathbf{y}^T \frac{(\mathbf{E}_m - \mathbf{E}_n)^T (\mathbf{E}_m - \mathbf{E}_n)}{\sqrt{\mathbf{y}^T (\mathbf{E}_m - \mathbf{E}_n)^T (\mathbf{E}_m - \mathbf{E}_n) \mathbf{y}}} \\ &= \frac{\begin{bmatrix} \mathbf{0}_{1 \times (m-1)k} & \mathbf{y}_{m-n}^T & \mathbf{0}_{1 \times (n-m-1)k} & -\mathbf{y}_{m-n}^T & \mathbf{0}_{1 \times (N-n)k} \end{bmatrix}}{\|\mathbf{y}_{m-n}\|} \end{aligned} \quad (4)$$

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

Similarly taking the jacobian of (1), 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

$$\nabla_{\mathbf{y}} f(\mathbf{y}) = \sum_{m=1}^N \sum_{n=m+1}^N 2f_{mn}(\mathbf{y}) \nabla_{\mathbf{y}} f_{mn}(\mathbf{y}) \quad (5)$$

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 (1), (2), (4), and (5), 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 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

$$\mathbf{A} := \begin{bmatrix} D_{\mathbf{y}}f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}}f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}}f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda}\mathbf{I}_{Nk \times Nk} \end{bmatrix} \quad \text{and} \quad \mathbf{b} := \begin{bmatrix} D_{\mathbf{y}}f_{1,1}(\mathbf{y})\mathbf{y} - f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}}f_{1,2}(\mathbf{y})\mathbf{y} - f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}}f_{N-1,N}(\mathbf{y})\mathbf{y} - f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda}\mathbf{y} \end{bmatrix}. \quad (6)$$

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

$$D_{\mathbf{y}}f_{mn}(\mathbf{y})\mathbf{y} - f_{mn}(\mathbf{y}) = \mathbf{y}_{\mathbf{m}-\mathbf{n}}(\mathbf{y}_{\mathbf{m}} - \mathbf{y}_{\mathbf{n}}) - \|\mathbf{y}_{\mathbf{m}-\mathbf{n}}\| + D_{mn} = D_{mn}. \quad (7)$$

Thus, noticing that \mathbf{b} in (6) results of the concatenation of terms of the form (7), \mathbf{b} is computed using

$$\mathbf{b} = \begin{bmatrix} D_{1,1} & D_{1,2} & \dots & D_{N-1,N} & \sqrt{\lambda}\mathbf{y} \end{bmatrix}, \quad (8)$$

which is very efficiently computed since it does not have to be 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 $\mathbf{y}_{\mathbf{m}-\mathbf{n}}$. 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, $\mathbf{y}_{\mathbf{m}-\mathbf{n}}$, or $\|\mathbf{y}_{\mathbf{m}-\mathbf{n}}\|$, then this allows to reduce the computational load significantly.

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

```

1 function [costF,normG,A,b] = objectiveF(y)
2 %% objectiveF.m
3 % Input: y: vector at which the quantities are evaluated
4 % Output: costF: cost function value
5 %         normG: norm of the gradient of the cost function
6 %         A: matrix A for the application of the LM method (only the entries
7 %           that do not depend on lambda)
8 %         b: vector b for the application of the LM method (only the entries
9 %           that do not depend on lambda)
10 %% Initialize cost function dataset
11 % Load dataset in the first call to this function

```

```

12 persistent b_ N k % do not have to be recomputed between calls
13 if isempty(k) || isempty(b_) || isempty(N)
14     % Load data:
15     % D: Distance matrix
16     % N: Number of data points
17     % k: Dimension of the target space
18     load('./data/objectiveFData.mat','D','N','k');
19     % Compute the portion of b that is constant
20     b_ = nonzeros(tril(D,-1));
21     fprintf("Initializing dataset.\n");
22 end
23 %% Compute quantities
24 costF = 0; % Initialize costF
25 gradf = zeros(1,N*k); % Initialize gradf
26 A = zeros((N^2-N)/2+N*k,N*k); % Initialize A
27 b = [b_;zeros(N*k,1)]; % Compute entries of b that do not depend on lambda
28 count = 1; % Iteration count
29 for i = 1:N
30     for j = i+1:N
31         dy = (y((i-1)*k+1:(i-1)*k+k)-y((j-1)*k+1:(j-1)*k+k))'; % y_{m-n}
32         normdy = norm(dy); % ||y_{m-n}||
33         faux = normdy-b_(count); % f_{mn} = ||y_{m-n}||-D_{mn}
34         gradaux_ = [zeros(1,(i-1)*k) dy zeros(1,k*(j-i-1))...
35             -dy zeros(1,(N-j)*k)]/(normdy); % D_y(f{mn}(y))
36         costF = costF + faux^2; % f(y) += f_{mn}(y)^2
37         % D_y(f(y))(y) += 2*f_{mn}(y)*D_y(f{mn}(y))
38         gradf = gradf + 2*faux*gradaux_;
39         A(count,:) = gradaux_; % A(<->) = D_y(f{mn}(y))
40         count = count+1; % increment iteration count
41     end
42 end
43 normG = norm(gradf); % compute norm of D_y(f(y))(y)
44 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

```

1 function [xk,k,costF,normGk] = LMAlgorithm(lambda0,x0,epsl,maxIt)
2 %% LMAlgorithm.m
3 % Input: lambda0: initialization lambda of the LM algorithm
4 %         x0: initialization solution estimate
5 %         epsl: stopping criterion
6 %         maxIt: maximum number of iterations

```

```

7 % Output: xk: output of the gradient descent method (returns NaN if
8 %         stopping criterion not met after the maximum number of
9 %         iterations chosen
10 %        k: number of iterations required for convergence if a
11 %        solution was found
12 %        costF: vector of objective function value for each iteration
13 %        normGk: vector of the norm of the gradient of the objective
14 %        function for each iteration
15 %% LM algorithm
16 % Initialize variables
17 costF = zeros(maxIt,1); % vector of objective function value
18 normGk = zeros(maxIt,1); % vector of gradient norms of the objective f
19 % Initialize LM algorithm
20 k = 0; % Initialize iteration count
21 xk = x0; % Initialization solution estimate
22 lambdak = lambda0; % Initialization lambda
23 fprintf("Running LM.\n");
24 % ----- LM algorithm -----
25 while k < maxIt % iterate up to a limit of maxIt iterations
26     % ----- Compute objective function and gradients -----
27     if k % store previous A and b, which are used if the step is invalid
28         Aprev = A;
29         bprev = b;
30     end
31     % Compute objective f value, norm of the gradient, A and b for xk
32     % note that only the entries of A and b that do not depend on lambda
33     % are computed
34     [costF(k+1),normGk(k+1),A,b] = objectiveF(xk);
35     % ----- check stopping criterion -----
36     if normGk(k+1) < epsl % Stopping criterion
37         break;
38     end
39     % ----- Check validity of last step -----
40     if k && costF(k+1)<costF(k) % if step is valid
41         lambdak = 0.7*lambdak; % decrease lambda
42     elseif k % if step is invalid
43         xk = xprev; % redo step
44         A = Aprev;
45         b = bprev;
46         normGk(k+1) = normGk(k);
47         costF(k+1) = costF(k);
48         lambdak = 2*lambdak; % decrease lambda
49     end
50     % ----- New step (solve least squares problem) -----
51     % store previous estimate, which is used if the step is invalid
52     xprev = xk;
53     % Entries of A and b that depend on lambda must be computed
54     A(end-size(x0,1)+1:end,:) = sqrt(lambdak)*eye(size(x0,1));
55     b(end-size(x0,1)+1:end,1) = sqrt(lambdak)*xk;
56     xk = ((A'*A)\A')*b; % solve least squares problem Ax=b
57

```

```

58      % ----- Increment iteration count -----
59      % Display LM status
60      if k fprintf("Iteration: %d | cost = %g.\n",k,costF(k+1)); end
61      k = k+1; % increment iteration count
62  end
63  if k == maxIt
64      % No solution found within the maximum number of iterations
65      xk = NaN; % Output invalid estimate
66  else
67      % Output only relevant data
68      costF = costF(2:k+1);
69      normGk = normGk(2:k+1);
70  end
71  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

```

1  %% Part 3 – Task 3 (part3task3.m)
2  %% Initialize cost function dataset
3  % Load distances matrix of the dataset of task 1
4  load("./data/distancesTask1.mat",'D','N');
5  % Initialize variables to hold the solution and status parameters of the LM
6  % algorithm for K = 2,3
7  solLM = cell(2,1); % solution of the optimization problem
8  itLM = zeros(2,1); % number of iterations ran
9  elapsedTimeLM = zeros(2,1); % time elapsed running LM
10 costLM = cell(2,1); % vector of cost function values for each iteration
11 % vector of gradient norm of the cost function for each iteration
12 normGradLM = cell(2,1);
13
14 %% Solve optimization problem for k = 2,3
15 for k = 2:3 % target space dimension
16     % Set up parameters
17     maxIt = 200; % maximum number of iterations
18     lambda0 = 1; % initial value for lambda of the LM method
19     eps1 = k*1e-2; % stopping criterion
20     % set up data for the computation of the quantities related to the
21     % objective function in objectiveF(y)
22     save("./data/objectiveFData.mat",'D','N','k');
23     y0 = csvread(sprintf("./data/yinit%d.csv",k)); % initialization of LM
24     clear objectiveF; % clear persistent variables in objectiveF
25     fprintf("----- Task 3 ----- \n");
26     tic; % start counting LM time

```

```

27 % run LM method
28 [solLM{k-1,1},itLM{k-1,1},costLM{k-1,1},normGradLM{k-1,1}] =...
29     LMalgorithm(lambda0,y0,epsl,maxIt);
30 elapsedTimeLM(k-1,1) = toc; % save elapsed time
31 if ~isnan(solLM{k-1,1}) % if a solution was found
32     fprintf("Solution found for dataset of task 1 with k = %d "+...
33         "using LM algorithm.\n",k);
34     fprintf("— Objective function value: %g.\n",costLM{k-1,1}(end,1));
35     fprintf("— Elapsed time: %g s.\n", elapsedTimeLM(k-1,1));
36 else % if a solution was not found
37     fprintf("Solution could not be found for dataset of task 1 "+...
38         "with k = %d using\n LM algorithm with the provided "+...
39         "stopping criterion and maximum number of iterations.\n",k);
40 end
41 end
42 % Save solutions
43 save("./data/solTask3.mat",...
44     'solLM','itLM','elapsedTimeLM','costLM','normGradLM');
45 %% Plot results
46 for k = 2:3
47     figure('units','normalized','outerposition',[0 0 1 1]);
48     yyaxis left
49     plot(0:itLM(k-1,1)-1,costLM{k-1,1},'LineWidth',3);
50     hold on;
51     ylabel('$f(y)$','Interpreter','latex');
52     set(gca, 'YScale', 'log');
53     yyaxis right
54     plot(0:itLM(k-1,1)-1,normGradLM{k-1,1},'LineWidth',3);
55     ylabel('$||\nabla f(y)||$','Interpreter','latex');
56     set(gca,'FontSize',35);
57     ax = gca;
58     ax.XGrid = 'on';
59     ax.YGrid = 'on';
60     title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
61     set(gca, 'YScale', 'log');
62     xlabel('$k$','Interpreter','latex');
63     saveas(gcf,sprintf("./data/task3_LM_k_%d.fig",k));
64     hold off;
65     y = reshape(solLM{k-1},[k,N]);
66     figure('units','normalized','outerposition',[0 0 1 1]);
67     if k == 2
68         scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,...
69             'MarkerFaceColor','flat');
70     else
71         scatter3(y(1,:),y(2,:),y(3,:),100,'o','b','LineWidth',1,...
72             'MarkerFaceColor','flat');
73     end
74     hold on;
75     set(gca,'FontSize',35);
76     ax = gca;
77     ax.XGrid = 'on';

```

```

78     ax.YGrid = 'on';
79     title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
80     saveas(gcf,sprintf("./data/task3_lowerDim_k_%d.fig",k));
81     hold off;
82 end

```

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

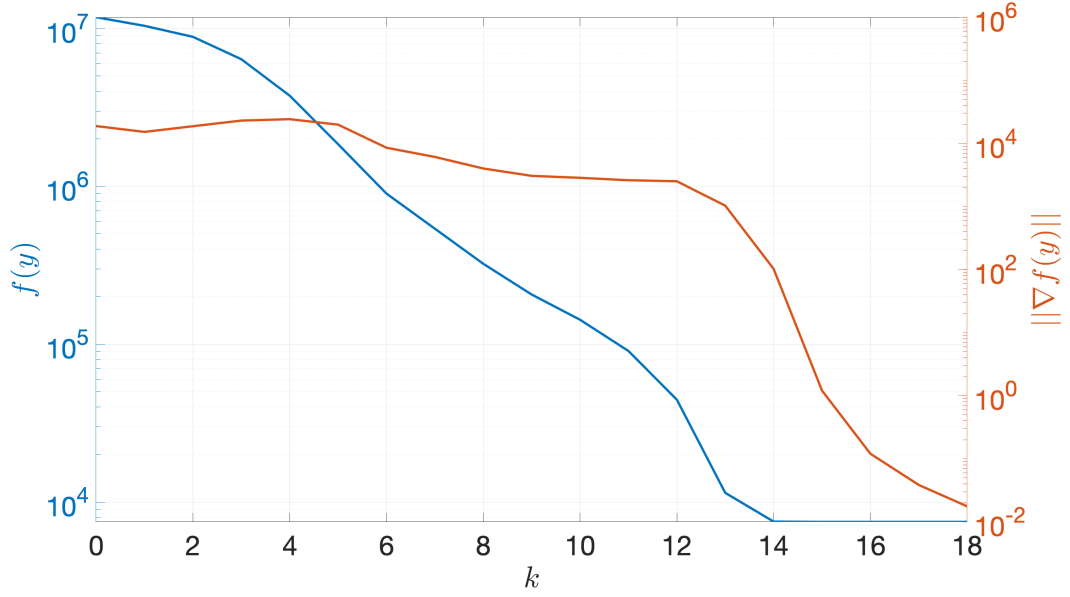


Figure 1: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

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 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.

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

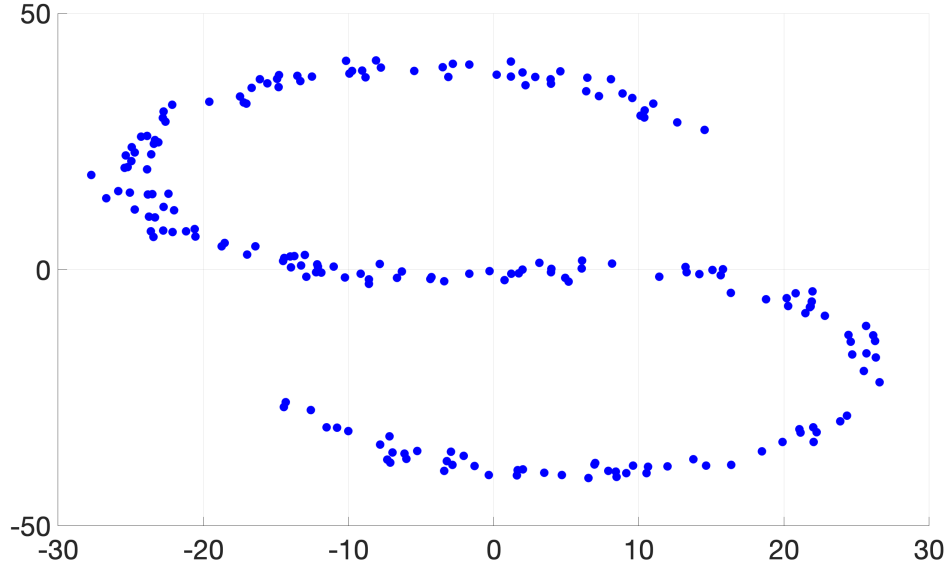


Figure 2: Solution to the optimization problem using the LM algorithm for $k = 2$.

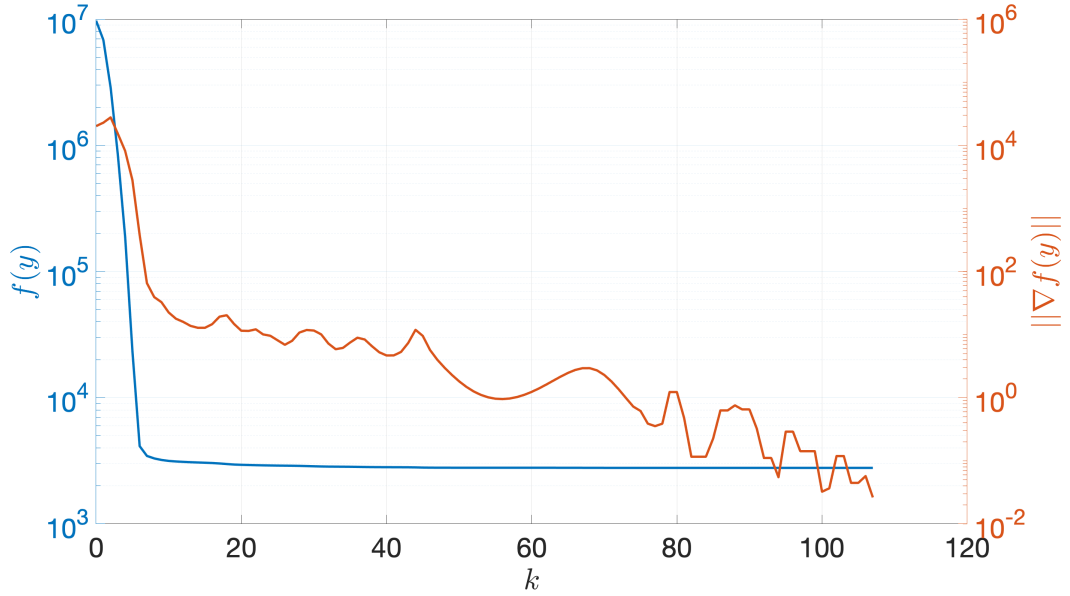


Figure 3: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

solution, since the objective function is not necessarily 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

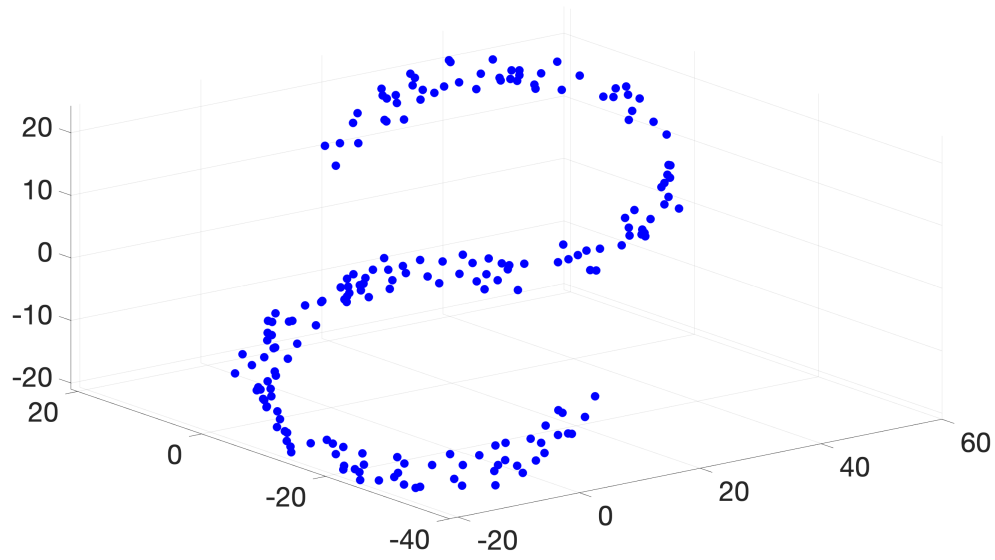


Figure 4: Solution to the optimization problem using the LM algorithm for $k = 2$.

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

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

the solutions can be run in a parallel manner, using the *Parallel* toolbox in MATLAB, for instance. The following MATLAB script solves task 4.

```

1 %% Part 3 – Task 4 (part3task4.m)
2 % Various runs can be performed. In each run the LM algorithm is run NIts
3 % times all for randomly generated initializations
4 %% Set parameters
5 RUN = 1; % run number
6 NRuns = 1; % number of runs so far
7 NIts = 12*2; % number of times the LM algorithm is called in a run
8 %% Load or compute data
9 computedD = true; % if D is already computed just load it
10 if ~computedD % C was not computed -> compute it now
11     X = csvread("./data/dataProj.csv");
12     N = size(X,1);
13     D = zeros(N);
14     for m = 1:N
15         for n = m+1:N
16             D(m,n) = norm(X(m,:) - X(n,:), 2);
17             D(n,m) = D(m,n);

```

```

18         end
19     end
20     % Save data
21     save('./data/distancesTask4.mat','D','N');
22 else % D was already computed -> compute it now
23     load('./data/distancesTask4.mat','D','N');
24 end
25 %% Run various times LM for random initializations
26 % Set up parameters
27 k = 2; % target space dimension
28 lambda0 = 1; % initial value for lambda of the LM method
29 maxIt = 200; % maximum number of iterations
30 epsl = k*1e-4; % stopping criterion
31 % Initialize variables to hold the solution and status parameters of the LM
32 % algorithm for the NIts LM calls
33 solLM = cell(NIts,1); % solution of the optimization problem
34 itLM = zeros(NIts,1); % number of iterations ran
35 elapsedTimeLM = zeros(NIts,1); % time elapsed running LM
36 costLM = cell(NIts,1); % vector of cost function values for each iteration
37 % vector of gradient norm of the cost function for each iteration
38 normGradLM = cell(NIts,1);
39 fprintf("----- Task 4 -----\n");
40 clear objectiveF; % clear persistent variables in objectiveF
41 save('./data/objectiveFData.mat','D','N','k');
42 parfor it = 1:NIts % calls of LM can be run in parallel
43     % each entry of y is randomly generated from an uniform distribution
44     % between -200 and 200
45     y0 = 200*2*(rand()-0.5)*2*(rand(N*k,1)-0.5);
46     tic; % start counting LM time
47     fprintf("----- RUN %02d - Attempt %02d -----\n",...
48         RUN,it);
49     % run LM method
50     [solLM{it,1},itLM(it,1),costLM{it,1},normGradLM{it,1}] =...
51         LMalgorithm(lambda0,y0,epsl,maxIt);
52     elapsedTimeLM(it,1) = toc; % save elapsed time
53
54     % if ~isnan(solLM{it,1}) % if a solution was found
55     %     fprintf("Solution found for dataset of task 4 with k = %d using LM algorithm.\n",k);
56     %     fprintf("- Objective function value: %f.\n",costLM{it,1}(end,1));
57     %     fprintf("- Elapsed time: %g.\n", elapsedTimeLM(it,1));
58     % else % if a solution was not found
59     %     fprintf("Solution could not be found for dataset of task 1 with "+...
60     %         "k = %d using\n LM algorithm with the provided stoppin criterion"+...
61     %         " and maximum number of iterations.\n",k);
62     % end
63 end
64 % Save whole run
65 save(sprintf('./data/RunsTask4/solRUN%02d.mat',RUN),...
66     'solLM','itLM','elapsedTimeLM','costLM','normGradLM');
67
68 %% Sort solutions found in all runs

```

```

69 solSorted = zeros(NRuns*NIts,3); % sorted list of all solutions
70 count = 0; % count number of solutions
71 for i = 1:NRuns
72     data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",i));
73     for j = 1:NIts
74         count = count+1;
75         solSorted(count,1) = i; % 1st column has run number
76         solSorted(count,2) = j; % 2nd column has attempt number within run
77         solSorted(count,3) = data.costLM{j,1}(end,1); % 2nd column has cost
78     end
79 end
80 solSorted = sortrows(solSorted,3); % sort rows ascending cost
81 save("./data/solSortedTask4.mat",'solSorted'); % save sorted solutions
82
83 % Best solution
84 % Load best solution run
85 data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",solSorted(1,1)));
86 % Get best solution data and save it
87 solLM = data.solLM{solSorted(1,2),1};
88 itLM = data.itLM(solSorted(1,2),1);
89 elapsedTimeLM = data.elapsedTimeLM(solSorted(1,2),1);
90 costLM = data.costLM{solSorted(1,2),1};
91 normGradLM = data.normGradLM{solSorted(1,2),1};
92 % Save best solution data
93 save("./data/solBestTask4.mat",...
94     'solLM','itLM','elapsedTimeLM','costLM','normGradLM');
95
96 %% Plot best solution
97 figure('units','normalized','outerposition',[0 0 1 1]);
98 yyaxis left
99 plot(0:itLM(k-1,1)-1,costLM,'LineWidth',3);
100 hold on;
101 ylabel('$f(y)$','Interpreter','latex');
102 set(gca, 'YScale', 'log');
103 yyaxis right
104 plot(0:itLM(k-1,1)-1,normGradLM,'LineWidth',3);
105 ylabel('$||\nabla f(y)||$','Interpreter','latex');
106 set(gca, 'FontSize', 35);
107 ax = gca;
108 ax.XGrid = 'on';
109 ax.YGrid = 'on';
110 title(sprintf("LM algorithm | Dataset task 4 | k = %d",k));
111 set(gca, 'YScale', 'log');
112 xlabel('$k$','Interpreter','latex');
113 saveas(gcf, "./data/task4_LM.png");
114 hold off;
115 y = reshape(solLM,[k,N]);
116 figure('units','normalized','outerposition',[0 0 1 1]);
117 scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,'MarkerFaceColor','flat');
118 hold on;
119 set(gca, 'FontSize', 35);

```

```

120 ax = gca;
121 ax.XGrid = 'on';
122 ax.YGrid = 'on';
123 title(sprintf("LM algorithm | Dataset task 4 | k = %d", k));
124 saveas(gcf, sprintf("./data/task4_sol.png"));
125 hold off;

```

The LM algorithm was run for 12 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 are identical, 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 is 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 to the global minimizer was found.

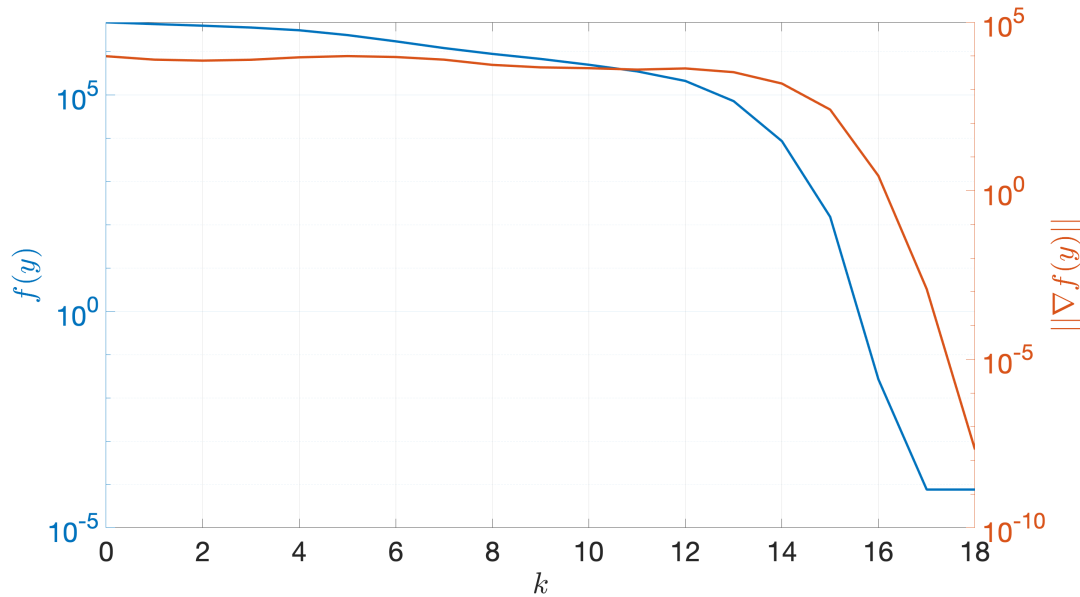


Figure 5: Objective function value and gradient norm throughout the iterations of the LM algorithm for the dataset of task 4 and $k = 2$.

1.5 Task 3 - bonus

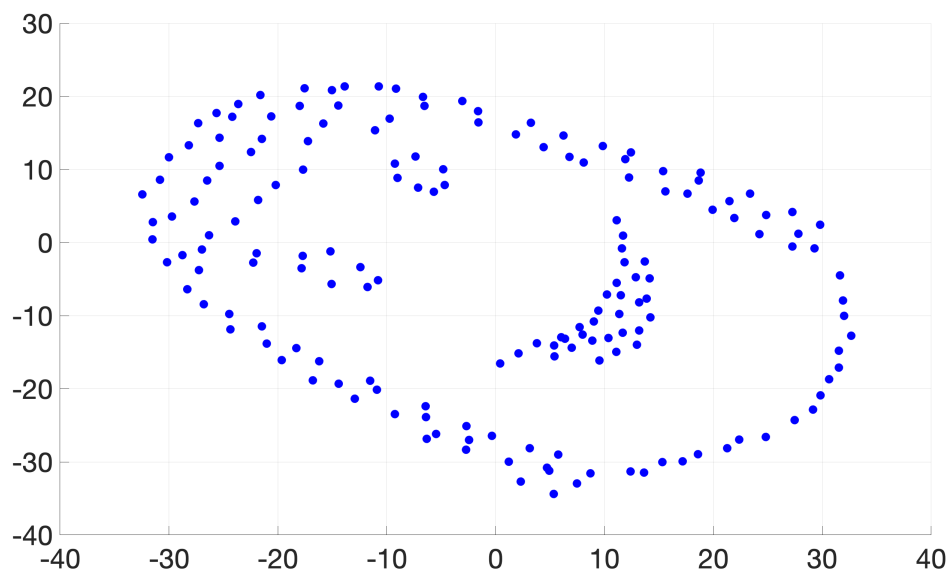


Figure 6: Solution to the optimization problem using the LM algorithm for the dataset of task 4 and $k = 2$.