Optimization and Algorithms
Project report

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

1 Part 1

2 Part 2

2.1 Task 1

In this Part, the goal is to solve the optimization problem

$$\underset{(s,r)\in\mathbf{R}^n\times\mathbf{R}}{\text{minimize}} \quad \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left(\log\left(1+\exp\left(s^Tx_k-r\right)\right)-y_k\left(s^Tx_k-r\right)\right). \tag{1}$$

From (1), the objective function is $f:\mathbf{R}^n\times\mathbf{R}\to\mathbf{R}$

$$f(s,r)=\frac{1}{K}\sum_{k=1}^{K}\left(\log\left(1+\exp\left(s^Tx_k-r\right)\right)-y_k\left(s^Tx_k-r\right)\right), \tag{2}$$

which can be written as

$$f(s,r)=\sum_{k=1}^{K}\frac{1}{K}h_k(s,r)+\sum_{k=1}^{K}\frac{1}{K}l_k(s,r), \tag{3}$$

where, for $k=1,...,K$, $h_k:\mathbf{R}^n\times\mathbf{R}\to\mathbf{R}$

$$h_k(s,r)=\log\left(1+\exp\left(s^Tx_k-r\right)\right), \tag{4}$$

and $l_k:\mathbf{R}^n\times\mathbf{R}\to\mathbf{R}$

$$l_k(s,r)=-y_k\left(s^Tx_k-r\right). \tag{5}$$

From (5), the functions l_k can be written as

$$l_k(s,r)=\begin{bmatrix}-y_kx_k\\ y_k\end{bmatrix}^T\begin{bmatrix}s\\ r\end{bmatrix}+0, \tag{6}$$

Therefore, the functions l_k are affine. Since l_k are affine, they are also convex. In addition, from (4), the functions h_k can be written as

$$h_k(s, r) = (t_k \circ q_k)(s, r), \tag{7}$$

where, for $k = 1, ..., K$, $t_k : \mathbf{R} \to \mathbf{R}$

$$t_k(z) = \log(1 + \exp(z)) \tag{8}$$

and $q_k : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$

$$q_k(z) = s^T x_k - r. \tag{9}$$

From (8), it can be observed that the functions t_k are logistic functions and, therefore, convex. The functions q_k, in the other hand, can be written from (9) as

$$q_k(s, r) = \begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix} + 0,$$

being, therefore, affine. Since the functions q_k are affine and t_k are convex, it is know from (7) that each function h_k can be written as the composition of an affine function with a convex function, which means the functions h_k are all convex. Since all functions h_k and l_k are convex, it is also known from (2) that the objective function f is the sum with positive coefficients of convex functions, which means f is convex.

2.2 Task 2

In this Task, the gradient descent algorithm will be used to solve (1). To use this algorithm, it is necessary to know the gradient of f. From (3), one can write

$$\nabla f(s, r) = \frac{1}{K} \sum_{k=1}^{K} (\nabla h_k(s, r) + \nabla l_k(s, r)). \tag{10}$$

From (5), the gradient of l_k can be written as

$$\nabla l_k(s, r) = \begin{bmatrix} -y_k x_k \\ y_k \end{bmatrix} = -y_k \begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{11}$$

From (7), it can be written that

$$\nabla h_k(s, r) = \nabla t_k(q_k(s, r)) \nabla q_k(s, r), \tag{12}$$

where, from (9),

$$\nabla q_k(s, r) = \begin{bmatrix} x_k \\ -1 \end{bmatrix} \tag{13}$$

and, from (8),

$$\nabla t_k(z) = \frac{dt_k}{dz}(z) = \frac{\exp(z)}{1 + \exp(z)}. \tag{14}$$

Combining (9), (12), (13), and (14) leads to

$$\nabla h_k(s,r) = \left(1 - \frac{1}{\exp\left(\begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix}\right) + 1}\right) \begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{15}$$

Finally, combining (10), (11), and (15), one can write

$$\nabla f(s,r) = \frac{1}{K} \sum_{k=1}^{K} \left(1 - y_k - \frac{1}{\exp\left(\begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix}\right) + 1}\right) \begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{16}$$

In order to apply the gradient method, the following MATLAB script was written. This script returns the results of the method for each of the datasets given. This script implements the Gradient Descent algorithm through the MATLAB function *gradientDescent* also given below.

```matlab
% GradientMethod.m
%% Initialization
clear;
clc;
NDataSets = 4;

%% Setup parameters
epsl = 1e-6; % stopping criterion
alpha_hat = 1; %initialization of alpha_k for the backtracking routine
gamma = 1e-4; % gamma of backtraking routine
beta = 0.5; % beta of backtraking routine
maxIt = [1e4; 1e4; 1e4; 1e5]; % maximum number of iterations

%% GD for each data set
for i = 1:NDataSets
    %% Upload data
    load(sprintf("./data%d.mat",i),'X','Y'); % upload data set
    K = length(Y);
    n = size(X,1);

    %% Set up x0 (note that x = [s;r])
    x0 = [-ones(n,1); 0];

    %% Setup objetive function and gradient
    h = [X;-ones(1,K)];
    F = @(x) (1/K)*...
        sum(log(1+exp((h'*x)'))-Y.*(h'*x)');
```

```
    gradF = @(x) (1/K)*sum((exp((h'*x)')./...
        (1+exp((h'*x)'))-Y).*h,2);

    %% Run GD
    fprintf("Running gradient descent for dataset %d (n = %d | K = %d).\n",...
        i,n,K);
    tic
    [xGD,ItGD,normGradGD] = gradientDescent(F,gradF,x0,epsl,...
        alpha_hat,gamma,beta,maxIt(i));
    elapsedTimeGD = toc;
    if ¬isnan(xGD)
        fprintf("Gradient descent for dataset %d"+...
        " converged in %d iterations.\n",i,ItGD);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeGD);
        if i≤2
            fprintf("s = [%g; %g] | r = %g.\n",xGD(1),xGD(2),xGD(3));
        end
    else
        fprintf("Gradient descent for dataset %d "+...
            "exceeded the maximum number of iterations.\n",i);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeGD);
    end
    save(sprintf("./DATA/GradientDescent/GDsolDataset%d.mat",i),...
        'xGD','ItGD','normGradGD','elapsedTimeGD');

    %% Plot result
    plotResults = false;
    if plotResults
    if i≤ 2
        figure('units','normalized','outerposition',[0 0 1 1]);
        set(gca,'FontSize',35);
        hold on;
        ax = gca;
        ax.XGrid = 'on';
        ax.YGrid = 'on';
        axis equal;
        for k = 1:K
            if Y(k)
                scatter(X(1,k),X(2,k),200,'o','b','LineWidth',3);
            else
                scatter(X(1,k),X(2,k),200,'o','r','LineWidth',3);
            end
        end
        %ylim([-4 8]);
        %xlim([-4 8]);
        title(sprintf("Dataset %d",i));
        ylabel('$x_2$','Interpreter','latex');
        xlabel('$x_1$','Interpreter','latex');
        x1 = (min(X(1,:)):(max(X(1,:)-min(X(1,:))))/100:max(X(1,:)));
        plot(x1,(xGD(3)-xGD(1)*x1)/xGD(2),'--g','LineWidth',4);
        saveas(gcf,sprintf("./DATA/GradientDescent/GDsolDataset%d.fig",i));
```

```
        close(gcf);
        hold off;
    end

    figure('units','normalized','outerposition',[0 0 1 1]);
    plot(0:ItGD,normGradGD,'LineWidth',3);
    hold on;
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title(sprintf("Gradient method | Dataset %d",i));
    ylabel('$||\Delta f (s_k,r_k)||$','Interpreter','latex');
    xlabel('$k$','Interpreter','latex');
    set(gca, 'YScale', 'log');
    saveas(gcf,sprintf("./DATA/GradientDescent/GDNormGradDataset%d.fig",i));
    close(gcf);
    hold off;
    end

end
```

```
% gradientDescent.m
function [xk,k,normGk] = gradientDescent(F,gradF,x0,epsl,...
    alpha_hat,gamma,beta,maxIt)
    %% Description
    % Inputs: 1. F: objective function (as a function handle)
    %         2. gradF: gradient of teh objective function (as a function
    %         handle)
    %         3. x0: initialization
    %         4. epsl: stopping criterion
    %         5. alpha_hat: initialization of alpha_k for the backtracking
    %         routine
    %         6. gamma: gamma of backtraking routine
    %         7. beta: beta of backtraking routine
    %         8. maxIt: maximum number of iterations
    % Outputs: 1. x: output of the gradient descent method (returns NaN if
    %          stopping criterion not met after the maximum number of
    %          iterations chosen
    %          2. k: number of iterations required for convergence if a
    %          solution was found
    %          3. normGk: norm of the gradient of the objective function
    %% Gradient descent routine
    k = 0;
    xk = x0;
    normGk = zeros(maxIt,1);
    while k < maxIt
        gk = gradF(xk); % Compute gradient at xk
        normGk(k+1) = norm(gk);
        if normGk(k+1) < epsl % Stopping criterion
```

```
            break;
        end
        % ----------  backtracking routine  ----------
        alpha_k = alpha_hat;
        % It is guaranteed that there is convergence, no maximum number of
        % iterations needed (obviously for beta < 0)
        while true
            % check if F(alpha_k) < phi(0)+gamma*phi_dot(0)+alpha_k
            if F(xk-alpha_k*gk) < F(xk)-gamma*alpha_k*(gk'*gk)
                break; % alpha_k found
            else
                alpha_k = beta*alpha_k; % Update alpha_k
            end
        end
        xk = xk - alpha_k*gk; % update xk
        % --------  End backtracking routine  --------
        k = k + 1; % Increment iteration count
    end
    if k == maxIt
        % No solution found within the maximum number of iterations
        xk = NaN;
    else
        normGk = normGk(1:k+1);
    end
end
```

For Task 2, the results obtained were $s = (1.3495, 1.0540)$ and $r = 4.8815$. The dataset 1 and the line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$ are represented in Fig. 1. In Fig. 2, the norm of the gradient along iterations is represented.

2.3 Task 3

In this Task, the code used was the one presented in the previous section. The results can be observed in Fig. 3 and 4 and the values obtained for s and r were $s = (0.7402, 2.3577)$ and $r = 4.5553$.

2.4 Task 4

In this Task, the gradient method was applied to two different datasets. However, in these datasets the points are no longer two-dimensional. In the dataset 3, $x_k \in \mathbf{R}^{30}$, for $k = 1, ..., 500$, and in the dataset 4, $x_k \in \mathbf{R}^{100}$, for $k = 1, ..., 8000$, which means it is no longer possible to represent the datasets. Therefore, only the global minimizers, s and r, and the evolution of the norm of the gradient along iterations will be presented.

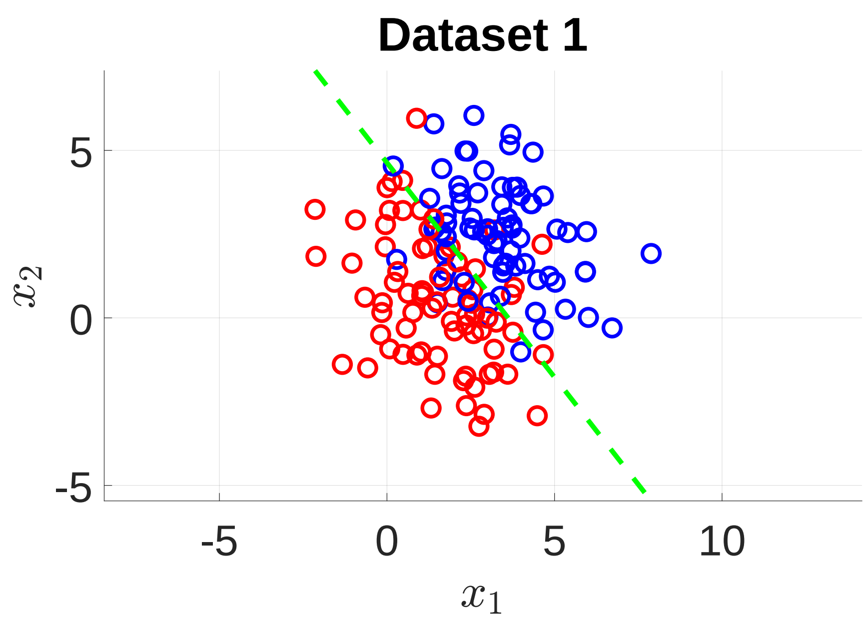

Figure 1: Dataset 1 and the corresponding line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$.

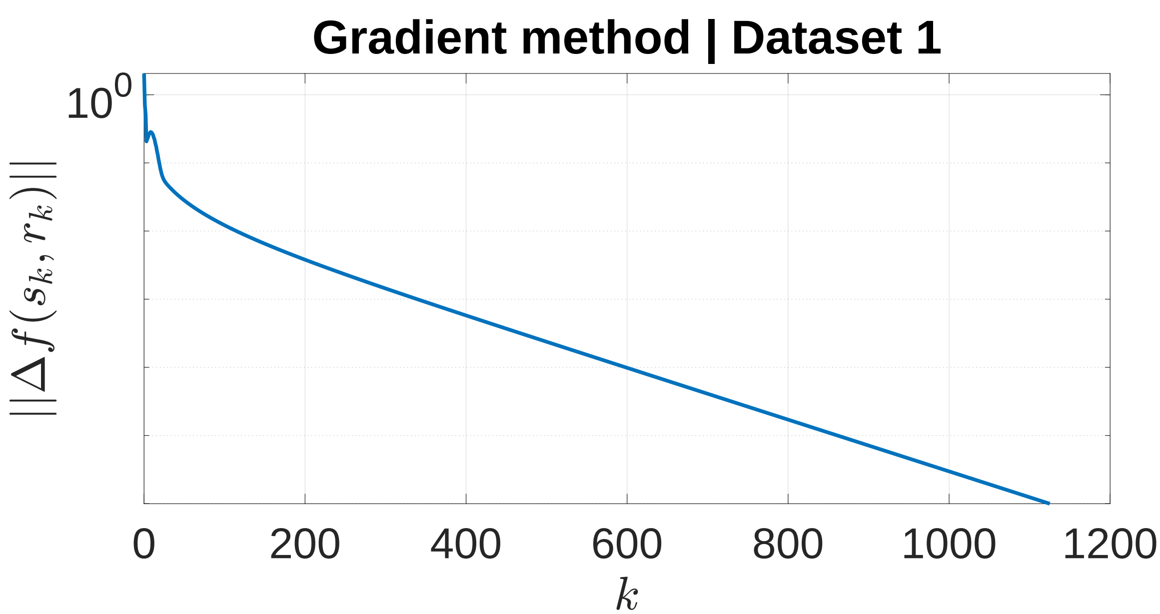

Figure 2: Norm of the gradient along iterations for the dataset 1.

For dataset 3,

$$s = [-1.3082, 1.4078, 0.8049, -1.0024, 0.5548, -0.5489, -1.1997, 0.0792, -1.8279, -0.1484, 1.9241, -0.3586, -0.2900, 0.1925, 1.0614, 0.2107, -0.0929, 1.0476, -1.1248, -1.3311, 0.7661, -0.2729, -0.5349, 0.9996, -0.4191, -0.3133, 0.4075, -0.1965, -0.7379, -0.9814],$$

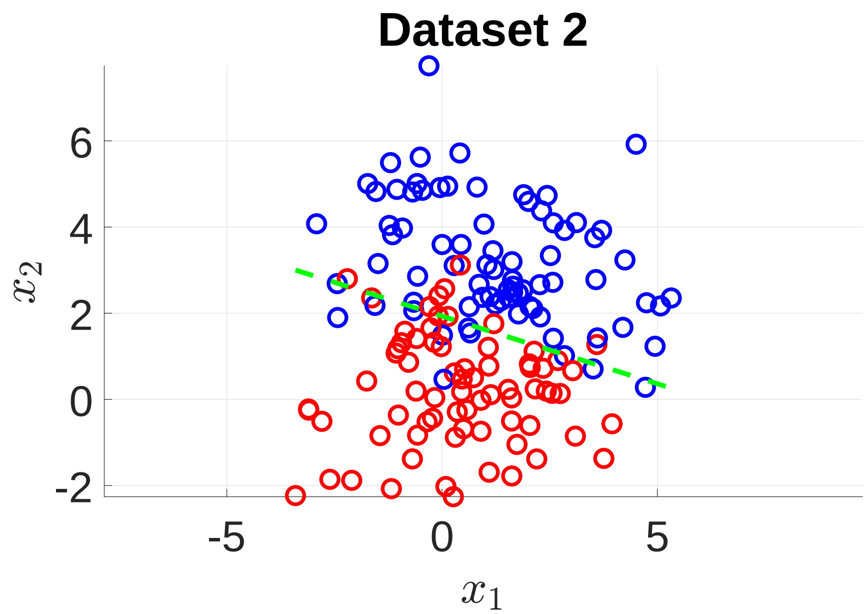

Figure 3: Dataset 2 and the corresponding line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$.

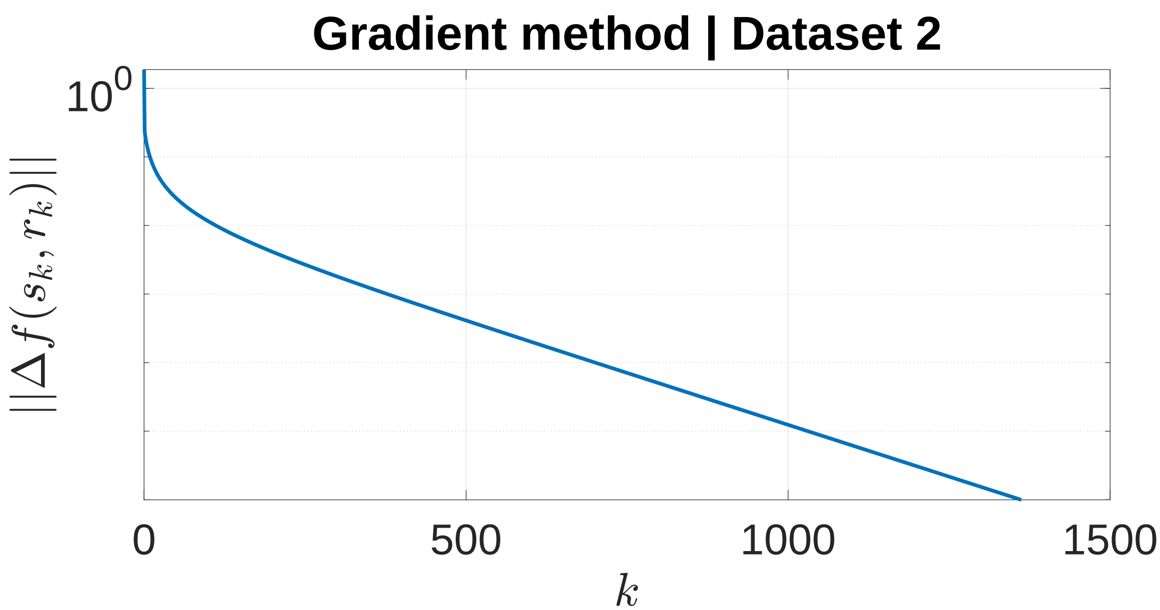

Figure 4: Norm of the gradient along iterations for the dataset 2.

$r = 4.7984$, and the evolution of the norm of the gradient along iterations is represented in

Fig. 5. For dataset 4,

$$
\begin{aligned}
s = [&0.1098, -0.6423, 0.1019, 1.2428, -1.6431, 1.0244, 0.0512, 0.8271, 0.3136, 0.7449, -0.5858,\\
&0.6267, 1.3611, 0.1534, 2.3234, -0.0840, -0.9489, 2.4699, -0.8678, -1.6516, 0.6460,\\
&-0.4779, 1.6397, 0.9034, -1.2293, -0.7587, -0.4887, 1.0306, 0.0888, -1.0917, -1.2717,\\
&-2.0333, -0.2505, -0.3518, -0.3486, -2.5610, -0.3132, -0.4902, 0.7258, 0.5774,\\
&-1.0528, 0.6400, 0.3759, -0.1547, 0.0298, 0.9547, -0.2863, 0.6364, 0.7859, 0.7584,\\
&0.2880, 0.1648, 0.6776, 2.0550, 1.0996, 0.5261, -0.5770, 1.1454, -0.5617, 0.0065, 0.4768,\\
&-2.3677, -1.1561, -2.6619, 0.0622, 0.1037, -0.6237, 0.1913, 0.6672, -1.0493, -0.3240,\\
&-0.3207, -1.0904, -0.8293, -0.3104, -0.4879, -0.1060, -0.1646, 2.2683, -1.2380,\\
&-0.8575, -2.4781, -0.4158, 0.1660, 0.7931, 0.3685, -0.0524, -0.9997, -0.5732, 0.3971,\\
&1.1911, 1.8318, -1.7287, 0.2329, -1.1921, 1.6558, 0.4612, -0.6431, 0.8295, 0.2975],
\end{aligned}
$$

$r = 7.6701$, and the evolution of the norm of the gradient along iterations is represented in Fig. 6.

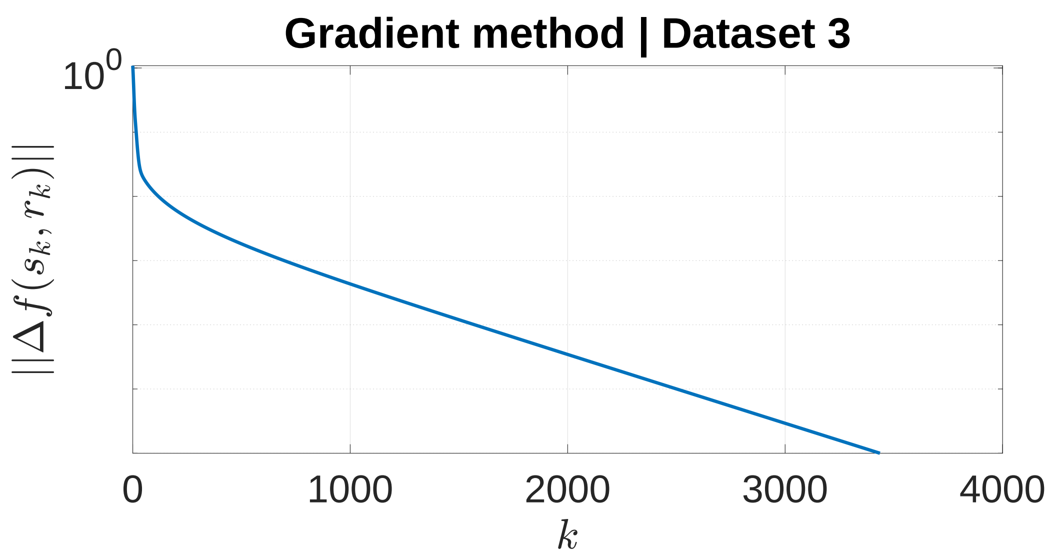

Figure 5: Norm of the gradient along iterations for the dataset 3.

2.5 Task 5

Letting $\phi : \mathbf{R} \to \mathbf{R}$ be a twice differentiable function and supposing $p : \mathbf{R}^3 \to \mathbf{R}$ is given by

$$p(x) = \sum_{k=1}^{K} \phi\left(a_k^T x\right), \tag{17}$$

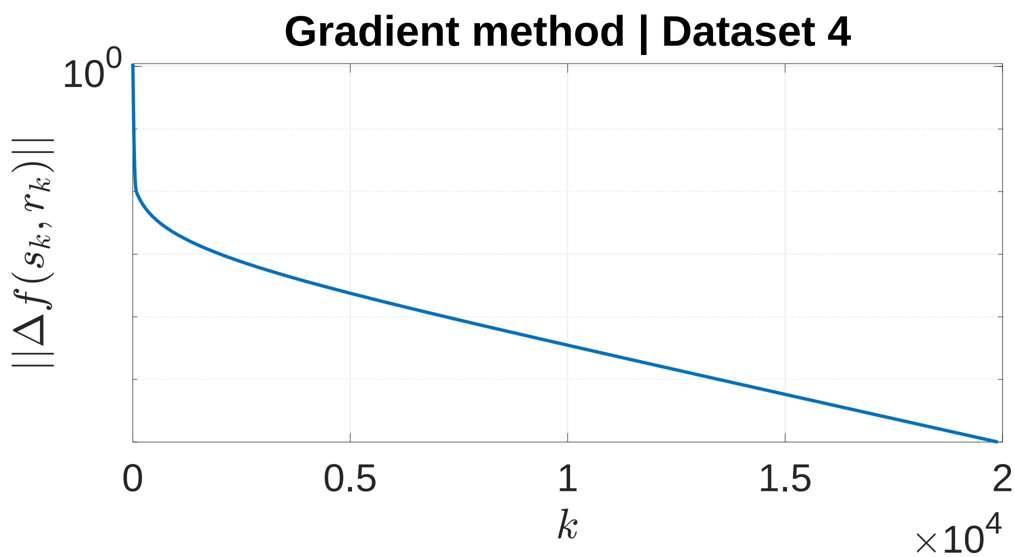

Figure 6: Norm of the gradient along iterations for the dataset 4.

where $a_k \in \mathbf{R}^3$ for $k = 1, ..., K$, one can write

$$\nabla p(x) = \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_k = Av, \tag{18}$$

where $A = [a_1 \quad a_2 \quad ... \quad a_K]$ and $v = [\phi(a_1^T x) \quad \phi(a_2^T x) \quad ... \quad \phi(a_K^T x)]^T$. In order to write the Hessian of p at x, one can define, for $k = 1, ..., K$, the functions $u_k : \mathbf{R} \to \mathbf{R}^3$

$$u_k(z) = z a_k, \tag{19}$$

which, from (18), lead to

$$\nabla p(x) = \sum_{k=1}^{K} u_k \left[\dot{\phi}(a_k^T x) \right]. \tag{20}$$

From (19) and (20), it is possible to write

$$\nabla^2 p(x) = \sum_{k=1}^{K} D u_k \left[\dot{\phi}(a_k^T x) \right] D \left[\dot{\phi}(a_k^T x) \right] D(a_k^T x) = \sum_{k=1}^{K} a_k \ddot{\phi}(a_k^T x) a_k^T = ADA, \tag{21}$$

where D is the diagonal matrix

$$D = \begin{bmatrix} \ddot{\phi}(a_1^T x) & & & \\ & \ddot{\phi}(a_2^T x) & & \\ & & ... & \\ & & & \ddot{\phi}(a_K^T x) \end{bmatrix} \tag{22}$$

2.6 Task 6

In this Task, the Newton method will be used to solve (1). To be able to use this method, it is necessary to know the gradient, which is given by (16), and the Hessian of the objective function. In order to write the Hessian, one starts by writing, from (2), (4), and (5),

$$f(x) = \sum_{k=1}^{K} \left(\phi\left(a_k^T x\right) + \frac{1}{K} \begin{bmatrix} -y_k x_k \\ y_k \end{bmatrix}^T x \right), \tag{23}$$

where $x = [s \quad r]^T$, $a_k = [x_k \quad -1]^T$ and $\phi : \mathbf{R} \to \mathbf{R}$

$$\phi(z) = \frac{1}{K} \log(1 + \exp(z)) \tag{24}$$

with second-derivative

$$\ddot{\phi}(z) = \frac{\exp(z)}{K\left[1 + \exp(z)\right]^2}. \tag{25}$$

Taking into consideration that (23) is written in the form of (17) except for the sum of affine therms whose second-derivative is null, it may be written, from (21), (22), and (25),

$$\nabla^2 f(x) = \begin{bmatrix} a_1 & a_2 & \dots & a_K \end{bmatrix} \begin{bmatrix} \ddot{\phi}(a_1^T x) & & & \\ & \ddot{\phi}(a_2^T x) & & \\ & & \dots & \\ & & & \ddot{\phi}(a_K^T x) \end{bmatrix} \begin{bmatrix} a_1 & a_2 & \dots & a_K \end{bmatrix}. \tag{26}$$

Taking into consideration (16) and (26), the Newton method was implemented according to the script below. In it, the Newton algorithm is implemented through the MATLAB function *newtonAlgorithm* also presented below.

2.7 Task 7

3 Part 3