CONTINUOUS OPTIMIZATION

Lab 1

Giulia Frigerio
Barbora Stepankova

Contents

1 Peer Review

Peer Review by Sofi Belhaj and Sanna Åslin

- The distance between the letters varies throughout the report, could be a latex error?

  It is the use of math mode to emphasize when we are talking about variables.

- Maybe not necessary to show all optimal x* in the report

  We deleted the list of correct points.

- Some figures don't have labels, titles

  All of our figures have labels and captions.

- "The starting points X. . . " under minimizing distance could be misunderstood for the set X. Maybe call the starting points something else?

  A good point, we changed it to calling it starting points S.

- "The starting points X were evenly spaced points in lines along the top and right edge, with 10 points in each line." Confusing sentence, clarify what you mean.

  We added an explanation that we meant "along the top and right edge of the set X"

- "This non-convexity is caused by the non-convexity of the set X" is not right, in the instructions we were given that convexity of d(p) is caused by convexity of set X but not the other way around.

  We are not stating this as proof, just explaining why the proof by picture is working.

- You have missed to answer the furthest and nearest distance and corresponding point for part 1 and part 2 respectively out of all optimal points x*.

  No we haven't, that wasn't part of the task.

- No mention of strong duality in the report.

  We are using strong duality as a stopping condition when searching for an optimal x, updating the duality gap in each run and terminating when the duality gap reaches values smaller than ε.

- Nice and clear plots

- Nice report!

- Code is well commented and easy to follow

2 Task description

The pear curve C is defined as the points $(x1, x2)$ satisfying:

$$g(x) = (x_1^2+x_2^2)(1+2x_1+5x_1^2+6x_1^3+6x_1^4+4x_1^5+x_1^6-3x_2^2+2x_2^4+x_2^6-2x_1x_2^2+4x_1x_2^4+8x_1^2x_2^2+3x_1^2x_2^4+8x_1^3x_2^2+3x_1^4x_2^2)-2=0$$

which is the boundary of the sublevel set:

$$X = \{x \in R^2 : g(x) \leq 0\}$$

We define the functions $d, D : R^2 \to R$ as

$$d(p) = min_{x\in X}||p-x||$$

$$D(p) = max_{x\in X}||p-x||$$

Our task is to explore the functions $d(p)$ and $D(p)$ for points p on two lines

- $p = (t, 1.5)$ with $t = -2.0, -1.9, -1.8, \cdots, 1.0$
- $p = (1, t)$ with $t = -4.0, -3.8, -3.6, \cdots, 4.0$

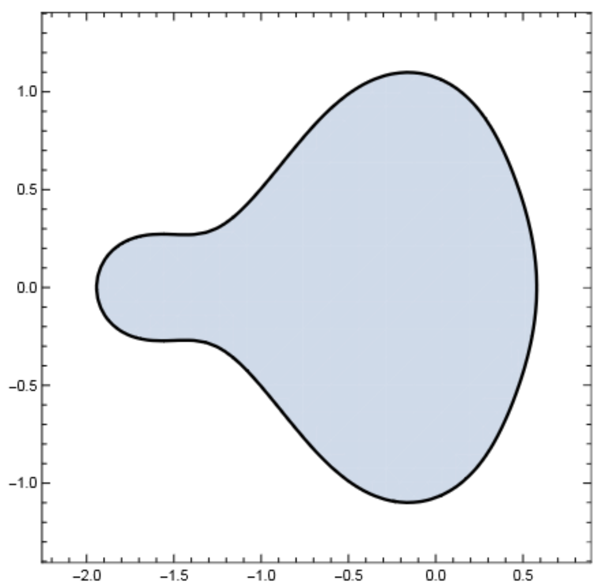

Figure 1: The set X.

3 First part - Horizontal line

We completed both figures shown in the task description and found the nearest and furthest points satisfying g from the horizontal line $p = (t, 1.5)$.

3.1 Maximizing distance

These figures were generated by running our function *solve_part* with the line p, starting points S, the negative max distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

The starting points S were evenly spaced points in lines along the bottom and right edge of the set X, with 10 points in each line.

This function is finding the optimal x for each point p, by first, by interval halving choosing a v and finding the maximum of $h(v) = \max_v \min_x Lagrange(x, v)$. To find the $\min_x Lagrange(x, v)$ with a fixed v, we use the Newton's method.

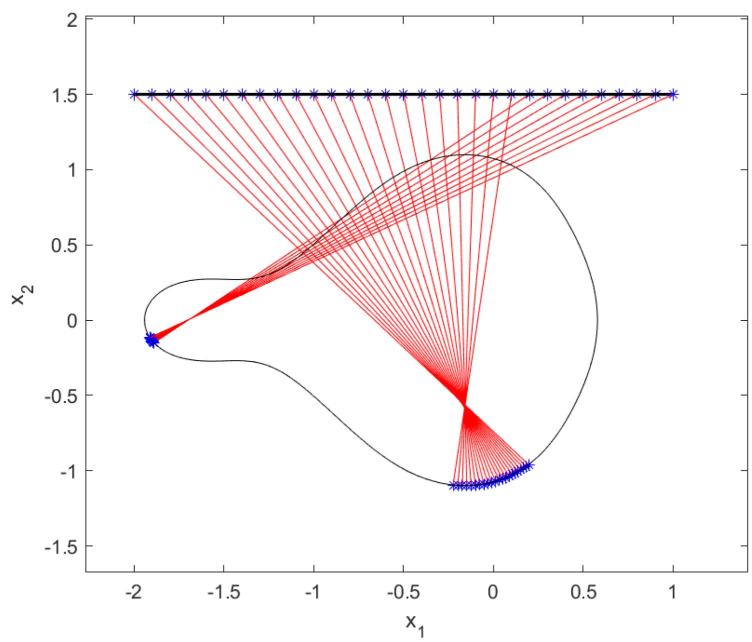

Figure 2: $x \in X$ and corresponding $p = (t, 1.5)$ with maximum distance.

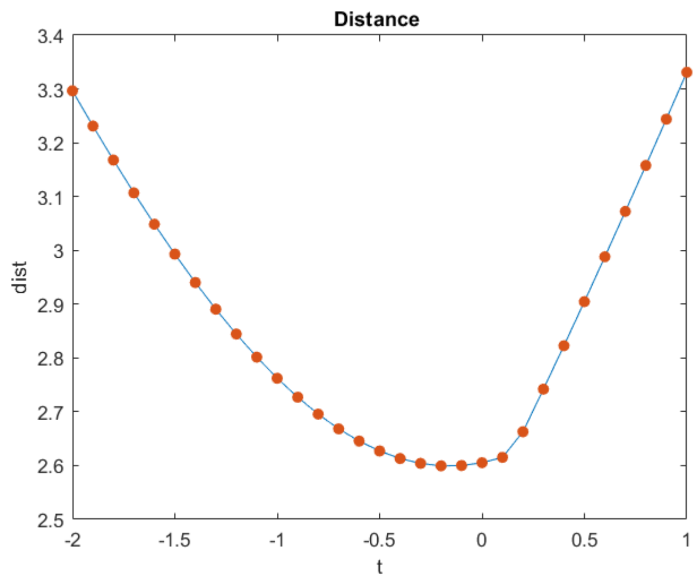

Figure 3: The maximal distance between X and $p = (t, 1.5)$ dependent on t.

3.2 Minimizing distance

We generated the following figures by running *solve_part* with the line p, starting points S, the min distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

The starting points S were evenly spaced points in lines along the top and right edge of the set X, with 10 points in each line.

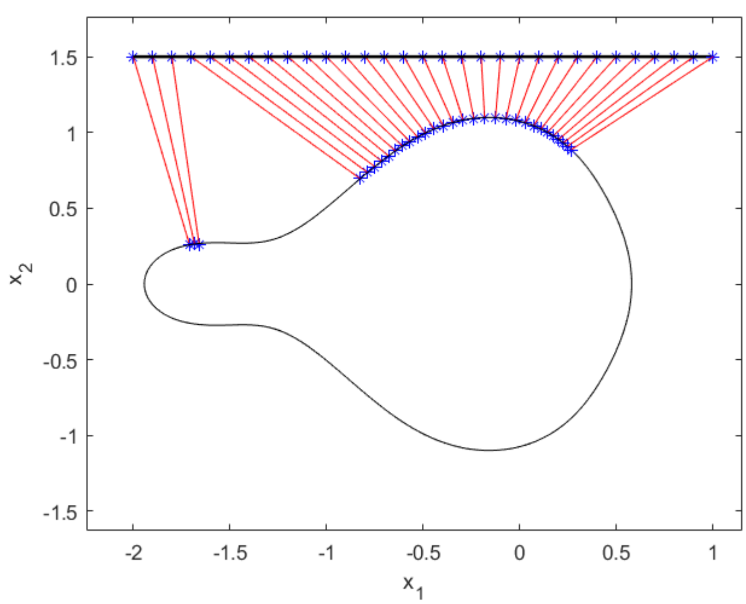

Figure 4: $x \in X$ and corresponding $p = (t, 1.5)$ with minimum distance.

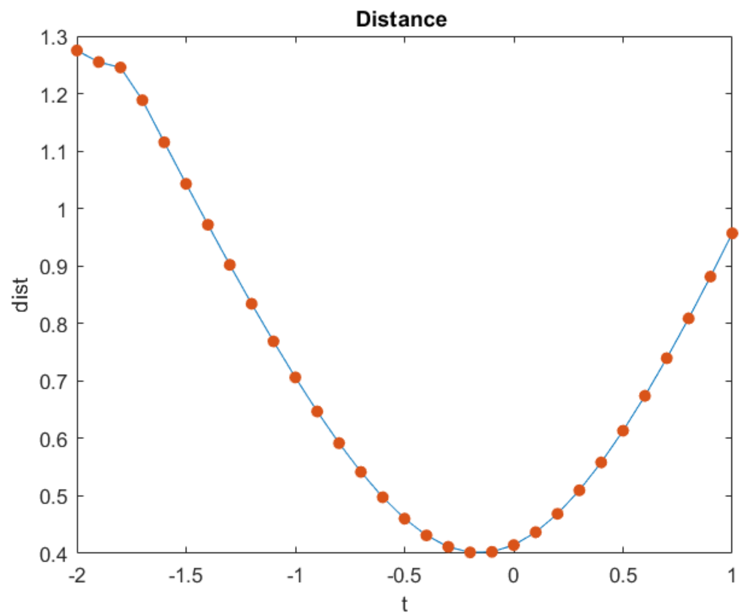

Figure 5: The minimal distance between X and $p = (t, 1.5)$ dependent on t.

As we can see in Figure 5, the function $d(p)$ is not convex. Take for example $t_a = -1.9$ and $t_b = -1.7$. The value at $t_{mid} = -1.8$ is clearly above the imaginary line connecting t_a with t_b.

This non-convexity is caused by the non-convexity of the set X.

3.3 Finding t_1

To find t_1, we run our function $find\_midpoint$ with 0 as a lower bound, 0.3 as an upper bound, 1.5 as a fixed coordinate, starting points S same as for the maximization task, the negative max distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

The function does interval halving, and based on the location of the newfound x^* sets the midpoint as a new upper or lower bound of t.

After the lower and upper bounds are closer to each other than ε, we get $t_1 = 0.1473$.

4 Second part - Vertical line

We completed both figures shown in the task description and found the nearest and furthest points satisfying g from the vertical line $p = (1, t)$.

4.1 Maximizing distance

These figures were generated by running our function *solve_part* with the line p, starting points S, the negative max distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

The starting points S were evenly spaced points in lines along the bottom and top edge of the set X, with 10 points in each line.

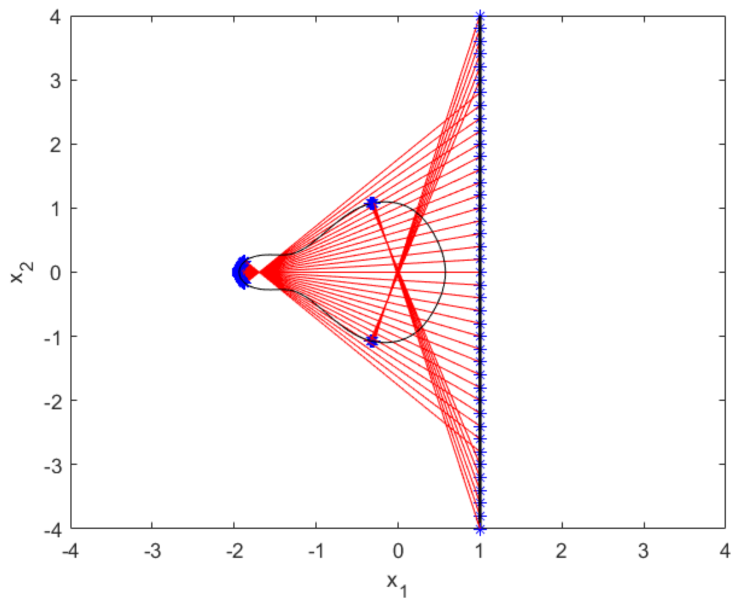

Figure 6: $x \in X$ and corresponding $p = (1, t)$ with maximum distance.

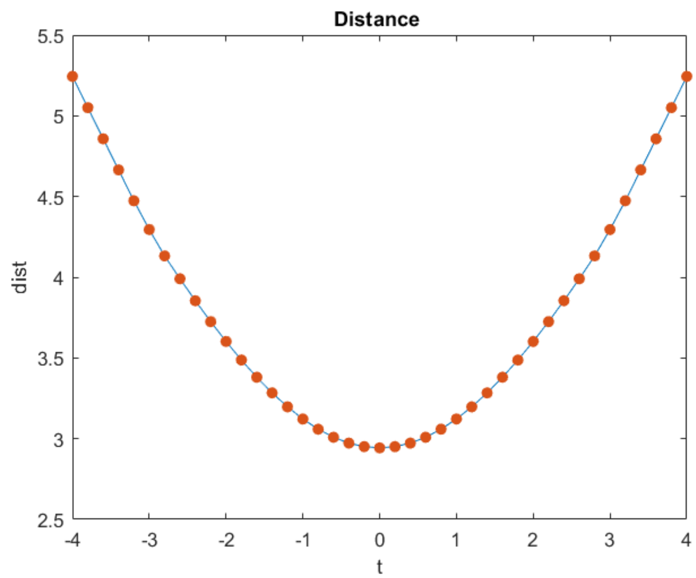

Figure 7: The maximal distance between X and $p = (1, t)$ dependent on t.

4.2 Minimizing distance

These figures were generated by running our function *solve_part* with the line p, starting points S, the min distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

The starting points S were evenly spaced points in a line along the right edge of the set X, with 10 points on the line.

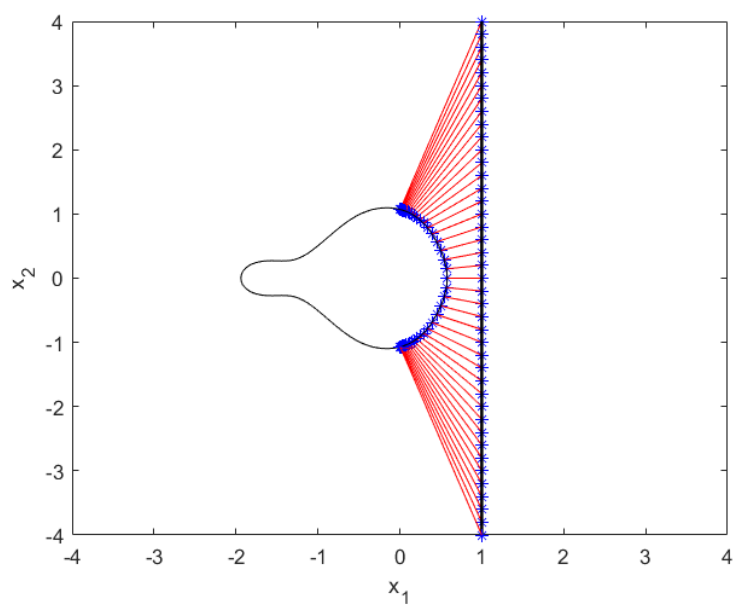

Figure 8: $x \in X$ and corresponding $p = (1, t)$ with minimal distance.

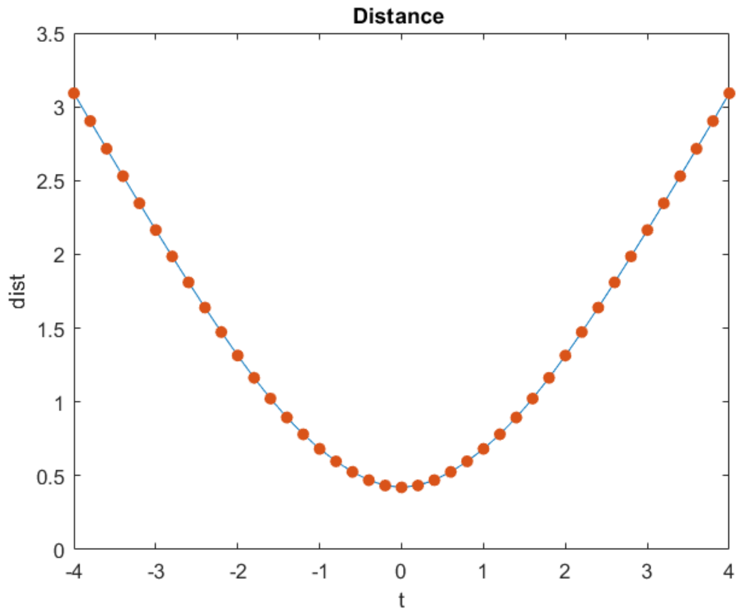

Figure 9: The minimal distance between X and $p = (1, t)$ dependent on t.

As we can see in Figure 9, function $d(p)$ is in this case convex, because the closest section of X was convex.

4.3 Finding t_2

To find t_2, we run our function $find\_midpoint$ with 2.5 as a lower bound, 3.1 as an upper bound, 1 as a fixed coordinate, starting points X same as for the maximization task, the negative max distance function, g, and an $\varepsilon = 10^{-4}$ as parameters.

We had to make minor changes to the function from using it to find t_1, as t_1 is changing the x coordinate, however, t_2 is changing the y coordinate. Other than these minor changes, the function remains the same.

The function does interval halving, and based on the location of the newfound x^* sets the midpoint as a new upper or lower bound of t.

After the lower and upper bounds are closer to each other than ε, we get $t_2 = 2.9778$.

A Main Loop

A.1 The init code necessary for every section

```
g = @(x1, x2) (x1^2 + x2^2) * (1 + 2 * x1 + 5 * x1^2 + 6 * x1^3 + 6
* x1^4 + 4 * x1^5 + x1^6 - 3 * x2^2 + 2 * x2^4 + x2^6 - 2 * x1 *
x2^2 + 4 * x1 * x2^4 + 8 * x1^2 * x2^2 + 3 * x1^2 * x2^4 + 8 * x1^3
* x2^2 + 3 * x1^4 * x2^2) - 2;

d = @(x1, x2, p1, p2) (x1 - p1)^2 + (x2 - p2)^2;
D = @(x1, x2, p1, p2) - (x1 - p1)^2 - (x2 - p2)^2;

eps = 10^(-4);
```

A.2 The code of the main section of the program

```
P_points_count = 41; % number of points from which we look for the min/max distance
X_points_per_line_count = 10; % number of starting points for the newton's method for each line

down_line = create_line(-2, 0.3, 0, -1, X_points_per_line_count);
up_line = create_line(-2, -0.3, 0, 1, X_points_per_line_count);
right_line = create_line(0.3, 0, 0.3, 1.2, X_points_per_line_count);
X_points = [up_line; down_line]; % combine lines going along the edges of g

% horizontal max
%P_points = create_line(-2, 1.5, 1, 1.5, P_points_count);
%X_opt_points = solve_part(P_points, X_points, D, g, eps)

% horizontal min
%P_points = create_line(-2, 1.5, 1, 1.5, P_points_count);
%X_opt_points = solve_part(P_points, X_points, d, g, eps)


% vertical max
P_points = create_line(1, 4, 1, -4, P_points_count);
X_opt_points = solve_part(P_points, X_points, D, g, eps)

% vertical min
%P_points = create_line(1, 4, 1, -4, P_points_count);
%X_opt_points = solve_part(P_points, X_points, d, g, eps)


t = (linspace(-4,4,P_points_count));
dplot(t, X_opt_points, P_points);
```

A.3 The *solve_part*() function

```
function x_opt_points = solve_part(P_points,X_points, f, g, eps)
%SOLVE_PART find optimum of function f satisfying g for all points P

    P_points_count = size(P_points,1);
    x_opt_points = [ones(P_points_count,1), ones(P_points_count,1)]; % prepare results array

    % iterate over points P on the line
    for i_p = 1:P_points_count
        p1 = P_points(i_p,1);
        p2 = P_points(i_p,2);

        % plot the pear and line
        fimplicit(g,'-','Color','k')
        line([P_points(1,1), P_points(P_points_count,1)],[P_points(1,2), P_points(P_points_count
        hold on

        x_opt = find_x_opt(p1, p2, X_points, f, g, eps);

        x_opt_points(i_p, 1) = x_opt(1);
        x_opt_points(i_p, 2) = x_opt(2);

        % plot the points
        plot(x_opt_points(i_p, 1),x_opt_points(i_p, 2),'* b')
        plot(p1,p2,'* b')
        line([p1, x_opt_points(i_p, 1)],[p2, x_opt_points(i_p, 2)],'Color', 'r')
        hold on
        line(P_points(1),P_points(P_points_count),'Color','k','LineWidth',1.5)
        xlim([-4,4])
        ylim([-4,4])
        xlabel('x_1')
        ylabel('x_2')
    end
    hold off
end
```

A.4 Function for finding the optimal x

```
function best_x_opt = find_x_opt(p1, p2, X_points, f, g, eps)
%FIND_X_OPT

    L = @(x1,x2,v) f(x1, x2, p1, p2) + v * g(x1,x2);
    best_x_opt = [0,0];

    % Initial bounds on v
```

```
    v_ub = 10;
    v_lb = 0;

    duality_gap = inf;

    % interval halving until the interval is epsilon or duality gap is
    % less than epsilon
    while v_ub - v_lb > eps
        if duality_gap < eps
            break;
        end
        % compute v and h(v)
        v = (v_ub + v_lb) / 2;
        [h, x_opt] = get_h_of_v(v, L, X_points, eps);

        % compute v + eps and h(v + eps)
        v_eps = v + eps;
        [h_eps, ~] = get_h_of_v(v_eps, L, X_points, eps);

        % halv the interval
        if h > h_eps
            v_ub = v;
        else
            v_lb = v;
        end

        % change optimal point if the new point gives better results
        if abs(f(x_opt(1), x_opt(2), p1, p2) - L(x_opt(1), x_opt(2), v)) < duality_gap
            duality_gap = abs(f(x_opt(1), x_opt(2), p1, p2) - L(x_opt(1), x_opt(2), v))
            best_x_opt = x_opt;
        end

    end
end
```

A.5 Function to get $h(v)$

```
function [h, x_opt] = get_h_of_v(v, L, X_points, eps)
%H_OF_V Give result of the h(v) function.

    % Lagrange with set v
    L_v = @(x1, x2) L(x1, x2, v);

    % Get optimal x for this set v
    x_opt = get_x_of_v(X_points, L_v, eps);
```

```
    % Evaluate
    h = L_v(x_opt(1), x_opt(2));
end
```

Function to get $x(v)$:

```
function x_opt = get_x_of_v(X_points, L, eps)
%GET_X_OF_V get optimal x by minimizing lagranian with set v

    % init
    x_opt = [X_points(1,1);X_points(1,2)];
    L_opt = L(x_opt(1), x_opt(2));
    X_points_count = size(X_points,1);

    % iterate over starting points x for newton
    for i_x = 1:X_points_count
        x1 = X_points(i_x,1);
        x2 = X_points(i_x,2);

        x_star = newton(L, [x1;x2], eps);

        % save the result if it is better
        if L(x_star(1), x_star(2)) < L_opt
            x_opt = x_star;
            L_opt = L(x_opt(1), x_opt(2));
        end
    end
end
```

A.6 Newton's method

```
function x_star = newton(f, x0, eps)
%NEWTON Newton's method for finding stationary points of f, starting from
%x0
    syms x1 x2

    df_sym = gradient(f, [x1,x2]);
    df = matlabFunction(df_sym,'Vars',[x1,x2]);
    ddf_sym = hessian(f, [x1,x2]);
    ddf = matlabFunction(ddf_sym,'Vars',[x1,x2]);

    df_vector = @(x) df(x(1),x(2)); % Vector version of df
    ddf_vector = @(x) ddf(x(1),x(2)); % Vector version of ddf

    % counter in case of the method not converging
    counter = 0;
```

```
    % iterate until convergence
    while norm(df_vector(x0)) > eps && counter < 500
        u = -df_vector(x0);
        t0 = (u' * u)/(u' * ddf_vector(x0) * u);
        x0 = x0 + (t0 * u); % New point
        norm(df_vector(x0));
        counter = counter + 1;
    end

    x_star = x0;
end
```

B Other functions

B.1 Function for finding t_i

```
function midpoint = find_midpoint(t_lb, t_ub, fixed_p, threshold, X_points, f, g, eps)
%FIND_MIDPOINT
    p1 = fixed_p;

    while t_ub - t_lb > eps
        p2 = (t_lb + t_ub) / 2;

        x_opt = find_x_opt(p1, p2, X_points, f, g, eps);

        if x_opt(1) < threshold
            t_lb = p2;
        else
            t_ub = p2;
        end
    end

    midpoint = (t_lb + t_ub) / 2;
end
```

B.2 Line-creating function

```
function points = create_line(a1, a2, b1, b2, number_of_points)
%CREATE_LINE Create a line of evenly spread points between
%two end points.
    x = (linspace(a1, b1, number_of_points))';
    y = (linspace(a2, b2, number_of_points))';
```

```
    points = [x,y];
end
```

B.3 Distance-plotting function

```
function dplot(t, x, p)
    distances = ones(size(p, 1),1);
    for i = 1:size(p,1)
        distances(i) = sqrt((x(i,1) - p(i,1))^2 + (x(i,2) - p(i,2))^2);
    end
    figure
    plot(t,distances)
    hold on
    scatter(t,distances,'filled')
    title('Distance ')
    xlabel('t');
    ylabel('dist')
    hold off
end
```