UMEÅ UNIVERSITET **October 8, 2025**
Department of mathematics and mathematical statistics
Project

Laboration 2

Continuous optimization

version 1.0

Name Sam Moeini, Oscar Lindquist & Johan Meurk

Graders
Per-Håkan Lundow & Alp Yurtsever

Contents

1 Introduction

Introduction

1.1 Problem 1

Our first problem involves packing non overlapping disks of equal radius within a square. With 36 disks, the optimal packing density becomes $d = \pi/4 \approx 0.785398$. However, for 49 disks we need to find a better packing arrangement with a density of at least 0.79.

1.2 Problem 2

Problem 2 involves packing disks of equal radius into a circular quadrant. In this scenario as a reference we have arranged 12 disks to achieve a density of 0.7553 which is notably close to optimal. The task at hand is to determine 'n':

The minimum number of disks required to attain a packing density of 0.80 or higher.

1.3 Problem 3

In this challenge we pack disks into a unit square while considering three forbidden zones. In this scenario we've successfully arranged 11 disks, achieving a density of $d = 0.7990$.

The third problem consists of two parts:

a) Find upper bounds for N(0.85), representing the minimum number of disks required to achieve a density of 0.85 or higher.

b) Find upper bounds for N(0.90), representing the minimum number of disks needed to attain a density of 0.90 or higher.

1.4 Problem 4

This problem involves packing disks into three quadrants of a circle. In this scenario we've successfully arranged 12 disks to achieve a density of $d = 0.84926$.

The fourth problem consists of two parts:

a) Determine N(0.85), finding the minimum number of disks needed to attain a density of 0.85 or higher.

b) Find upper bounds for N(0.88), representing the minimum number of disks required to achieve a density of 0.88 or higher.

2 Method

Method

2.1 Problem 1

In the first problem we have 49 non overlapping disks with equal radius, where the task is to fit them optimally in a square. For this problem we have formulated a target function to maximize the density within the square as following:

$$f(x) = \pi \sum_{i=1}^{n} r^2$$

Linear constraints:

$$a_i + r_i \leq 1 \tag{1}$$
$$b_i + r_i \leq 1 \tag{2}$$
$$r - a_i \leq 0 \tag{3}$$
$$r - b_i \leq 0 \tag{4}$$

Non-linear constraint:

$$\|(a_i, b_i) - (a_j, b_j)\| \geq 2r \tag{5}$$

for all $i \leq j \leq n$, where $i, j = 1, 2, \ldots, n$.

Linear Constraints (1-4):

Constraints (1) and (2) are in place to restrict the distance in the x and y directions within the unit square to be less than or equal to the radius of the disk respectively. These constraints essentially bound the placement of the disk in both the x and y dimensions.

Constraints (3) and (4) guarantee that the radius is always less than or equal to the values of a_i and b_i. These constraints play a crucial role in making sure that the disk remains within the unit square. Combining all four constraints guarantees that the disk's position and radius are properly constrained within the boundaries of the square. The same reasoning applies when designing the linear constraints in Problems 2 and 3.

Non-linear Constraint (5):

Non-linear Constraint (5) takes a different approach. It ensures that for all $i \leq j \leq n$ (where i and j range from 1 to n), the Euclidean distance between any pair of points (a_i, b_i) and (a_j, b_j) is greater than or equal to 2 times the value of r. It enforces a minimum separation between any pair of points making certain that no two disks overlap in the arrangement. This constraint ensures that the disks are positioned in a way that avoids any collisions.

The vector of variables x is of length $2n + 1 = 99$. It represents the coordinates of the disk centers denoted as (a, b) as well as the radius for each of the n circles.

Within this vector the initial n elements correspond to a_i where i ranges from 1 to n and the subsequent n elements correspond to b_i where i also spans from 1 to n. The final element of the vector represents the radius of the circles.

2.2 Problem 2

As mentioned above the same constraints apply for problem 2 with the same objective function. However there is a second non linear constraint added as following:

$\sqrt{a_i^2 + b_i^2} - r \leq 1$

This constraint ensures that the disks are inside our circular quadrant.

2.3 Problem 3

In problem 3 and 4 the radius of the disks are allowed to vary. The free variable vector x is organized for each circle i in the following manner: the value at the i-th index corresponds to the a-coordinate and the value at the (n + i)-th index corresponds to the b-coordinate, and the value at the (2n + i)-th index defines the radius of the disks.

Additionally three forbidden zones are within the square. Therefore a 5th linear constraint is formulated (giving the topleft forbidden zone shown in **figure 3**):

$-a_i + b_i + \sqrt{2r_i} \leq \frac{2}{3}$

As shown in **figure 3** the other two forbidden zones are circles. These circles are formed using these two non linear constraints:

$\sqrt{\left(a_i - \frac{2}{3}\right)^2 + \left(b_i - \frac{1}{4}\right)^2} \leq \frac{1}{4} + r_i$

$\sqrt{(a_i - 1)^2 + (b_i - 1)^2} \leq \frac{1}{4} + r_i$

Both circles have the radius 1/4 and center points (2/3, 1/4) and (1, 1) respectively.

Further, a modified non linear constraint is formulated ensuring we do not have any overlapping disks:

$\|(a_i, b_i) - (a_j, b_j)\| \geq r_i + r_j$

2.4 Problem 4

In problem 4 we are allowed to have varying radius in the disks. The area is restricted to 3 quadrants of a circle with the radius 1 and center in (0,0).

Considering that the restricted area is a circle we have the following non linear constraints:

$\sqrt{a_i^2 + b_i^2} \leq 1 - r_i$

$(a_i \geq 0 \ \& \ b_i \geq 0) \cdot (r_i + b_i) \leq 0$

$(a_i \leq 0 \ \& \ b_i \geq 0) \cdot -(a_i^2 + b_i^2 - r_i^2) \leq 0$

$(a_i \leq 0 \ \& \ b_i \leq 0) \cdot (a_i + r_i) \leq 0$

$(a_i \geq 0 \ \& \ b_i \leq 0) \leq 0$

$\|(a_i, b_i) - (a_j, b_j)\| \geq r_i + r_j$

The first constraint guarantees that each circle is contained within a circle of radius 1 effectively restricting their positions within a unit circle. The second constraint ensures that if the center of a circle is located in the first quadrant it does not encroach into the fourth quadrant. Constraint 3 and 4 is similar to constraint 2, but they apply to circles whose centers are in the second and third quadrants, respectively. Constraint 5 prevents circles from having centers in the fourth quadrant maintaining their positioning exclusively in the first three quadrants. The last constraint is for the circles not overlapping.

3 Results

Results

3.1 Problem 1

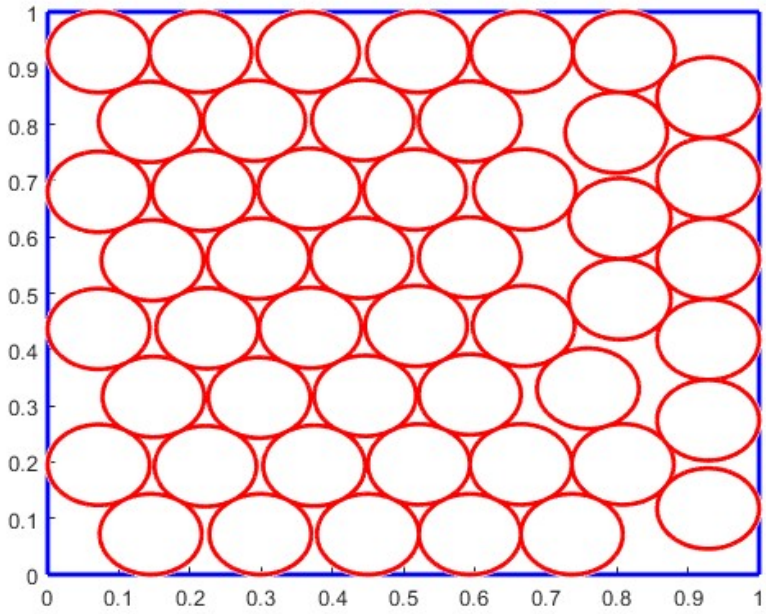

Figure 1: *Plot of an improved packing of 49 identical disks inside a unit-square, density 0.7905*

3.2 Problem 2

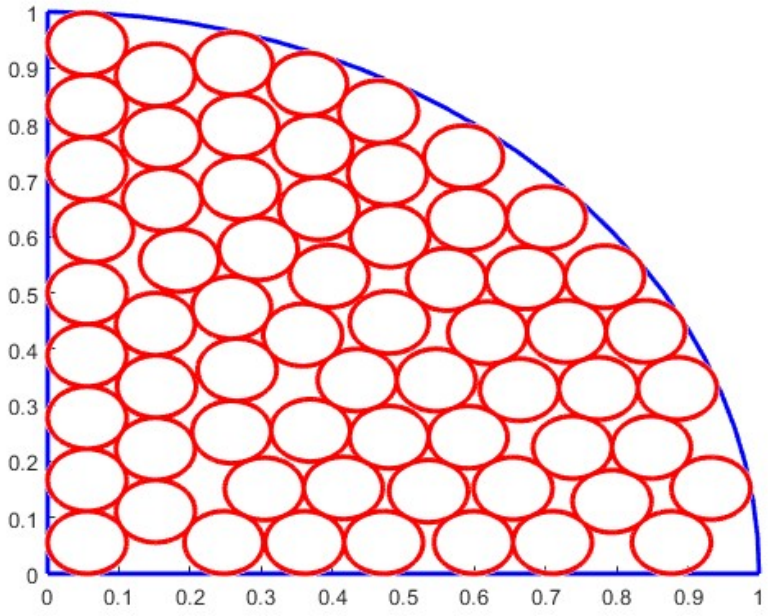

Figure 2: *Plot of 64 identical disks inside a circular quadrant, density 0.8010*

3.3 Problem 3

3.3.1 Sub task 1

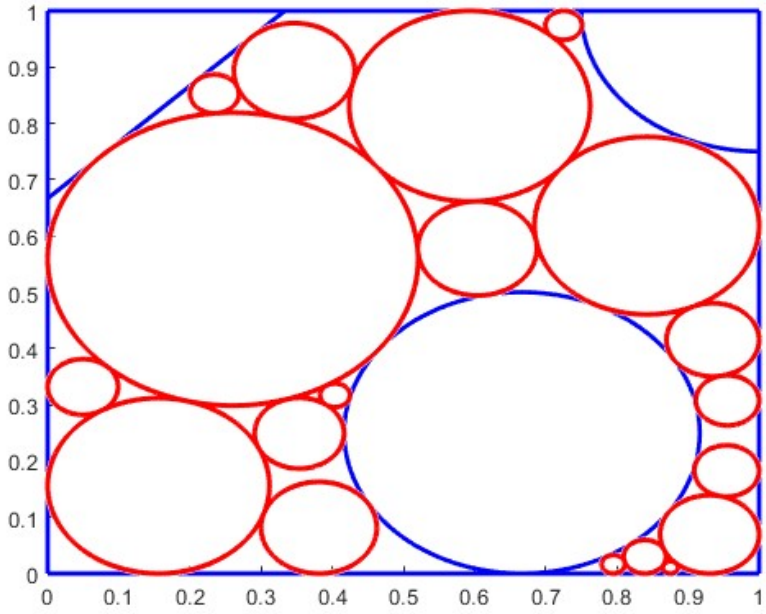

Figure 3: *Plot of 19 disks with free radius inside a unit-square with 3 forbidden zones, density 0.8525*

3.3.2 Sub task 2

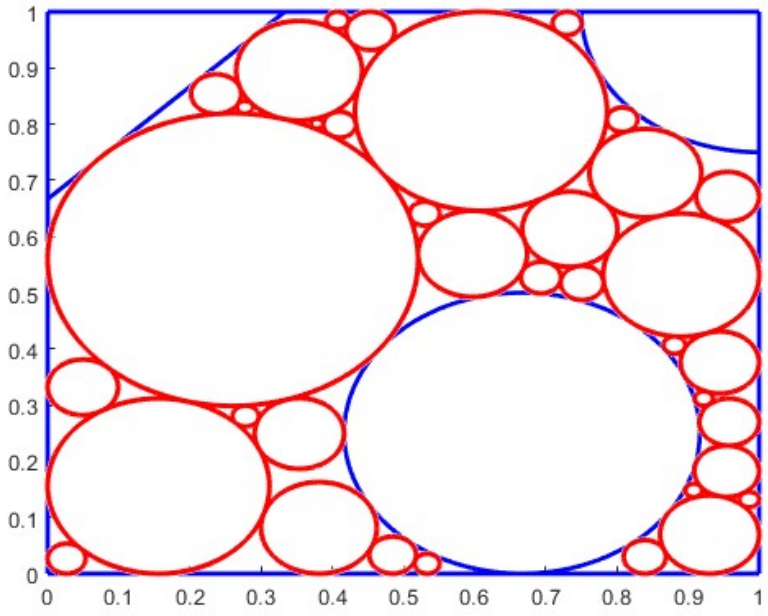

Figure 4: *Plot of 41 disks with free radius inside a unit-square with 3 forbidden zones, density 0.8863*

3.4 Problem 4

3.4.1 Sub task 1

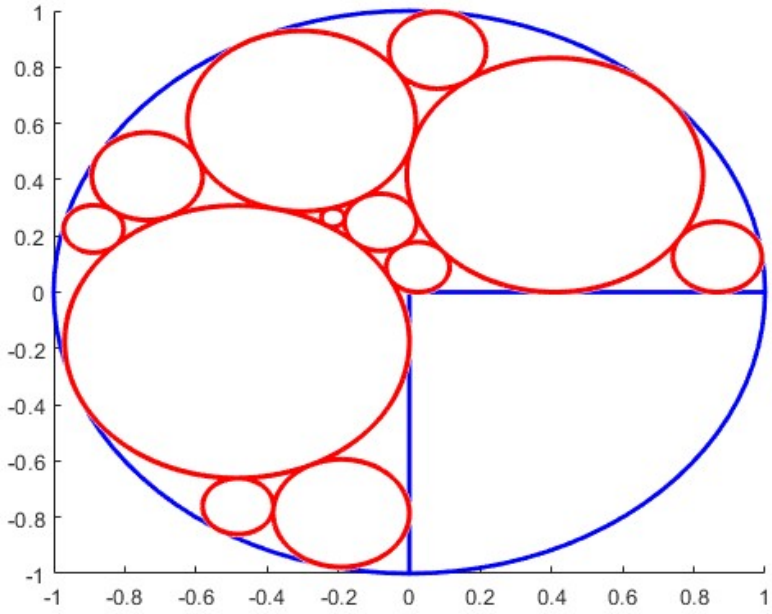

Figure 5: *Plot of 12 disks with free radius inside three quadrants of a circle, density 0.8507*

3.4.2 Sub task 2

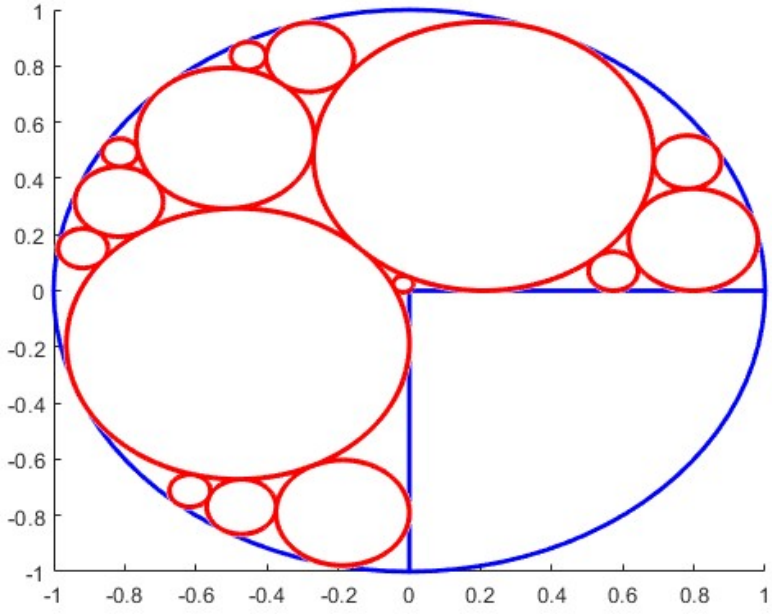

Figure 6: *Plot of 15 disks with free radius inside three quadrants of a circle, density 0.8803*

4 Conclusion

Conclusion

In order to solve these optimization problems, a lot of time was dedicated to coding in Matlab. Sometimes it was difficult to implement the problem to Matlab, but after some help from the teachers and handwritten ideas on how to solve the problems we managed to get reasonable results. The only problem in which we did not manage to get the demanded density was on problem 3 sub task 2, where the density was supposed to be 0.9 or higher with the least amount of disks. After many iterations, the best result we were able to get was at density = 0.8863 with 41 disks. This was mostly due to not enough computer power since we ran the code for hours after hours. Apart from that problem, we received a reasonable amount of disks for the demanded density on the other problems.

Something that should be clarified as well is the fact that these solutions are just feasible, but not optimal. We cannot prove that there is not a better optimal way to pack the disks inside the constraints, or if we could have used less disks but still receive the demanded density. In order to do that we would've had to optimize the running time for the code and also have better computer power.

Lastly, another aspect of the results are that they are not able to recreate. In our program, we run with random initial starting points and haven't saved neither them or radius. Instead if we rerun the code, we could get a different result with different amount of circles as our optimal result.

5 Review

Review

Reviewers: Elias Charalambidis, Johan Malmberg and Simon Styrefors

Well written report, in a condense fashion with clear and easy to understand objectives. Pleasing layout of sections, making the report easy to follow and understand.

The conclusion summed up the results as well as obstacles very well.

2.1 Describing the distance between the disk as a "safe distance" may be perceived wrong, because many disks have a arbitrarily small distance between them.

2.3 When formulating the problem by using the last problem's constraints and "adding" more constraints it seems hard to know exactly which constraints are used for the problem. For example in problem 3, you have to reformulate the non-linear constraint to be able to use different radii, but in the report it could be perceived as you just "add" an additional non-linear constraint.

6 Response

Response

The review given was constructive and gave good inputs. In the updated version of the report all comments were used to modify our method and change the formatting.

7 Appendix

Appendix

```
$$Problem 1
clc
clear all
fopt = 0;
n = 49;
f = @(x) -49.*pi.*x(2*n+1).^2;
while fopt < 0.79
    x0 = rand(2.*n+1, 1);
    lb = [zeros(n,1); zeros(n,1); zeros(n,1)];
    ub = [ones(n,1); ones(n,1); ones(n,1)./2];
    A = makeMatrixA(n);
    b = makeVectorA(n);
    options = optimoptions('fmincon','Display','off','
        Algorithm','sqp');
    problem = createOptimProblem('fmincon', 'objective
        ', f, 'x0', x0, 'lb', lb, 'ub', ub, 'Aineq', ...
    A, 'bineq', b, 'nonlcon', @constraintsA, 'options'
        , options);
    xopt = run(GlobalSearch, problem);
    fopt = densitycount(xopt(2*n+1),1,n);
end
plotProblemA(xopt,n)

function [g, geq] = constraintsA(x) % Fix for all
    constraints
    n = (length(x)-1)/2;
    geq = [];
    m = 2*n + n*(n-1)/2;
    g = zeros(m,1);
    k = 0;

    % This is for all ai, bi, when a1, a2... an, b1,
        b2 .. bn, r
    for i = 1:2*n
        k = k + 1; % r <= ai
        g(k) = x(2.*n+1) - x(i);
    end

    % For the distance constraint
    for i = 1:n-1
        for j = i+1:n
            k = k + 1;
            d = (x(i) - x(j))^2 + (x(n+i) - x(n+j))^2;
            g(k) = 4*x(2*n+1)^2 - d;
        end
    end
end

% Matrix A
```

```
function A = makeMatrixA(n)
    A = zeros(4.*n, 2*n+1);
    for i = 1:n
        A(i, i) = 1;
        A(n+i, n+i) = 1;
        A(2*n+i, i) = -1;
        A(3*n+i, n+i) = -1;
        A(:, 2*n+1) = 1;
    end
end

% Vector a
function [b] = makeVectorA(n)
    b = [ones(2*n,1); zeros(2*n,1)];
end

function plotProblemA(xopt, n) % A square
    line([0 1], [0 0], 'LineWidth', 2, 'Color', 'blue'
       )
    hold on
    line([0 0], [0 1], 'LineWidth', 2, 'Color', 'blue'
       )
    line([0 1], [1 1], 'LineWidth', 2, 'Color', 'blue'
       )
    line([1 1], [0 1], 'LineWidth', 2, 'Color', 'blue'
       )
    viscircles([xopt(1:n),xopt(n+1:2*n)], xopt(2*n+1))
end

function fopt = densitycount(r, tot, amount)
    circlearea = pi.*r.^2;
    totcircle = amount.*circlearea;
    fopt = totcircle/tot;
end

$$Problem 2
clear all
clc
n = 13;
dens = 0;
f = @(x) -pi.*x(2*n+1).^2;
k = 0;
while dens < 0.8
x0 = rand(2.*n+1, 1);
lb = [zeros(n,1); zeros(n,1); zeros(n,1)];
ub = [ones(n,1); ones(n,1); ones(n,1)./2];
A = makeMatrixA(n);
b = makeVectorA(n);
options = optimoptions('fmincon','Display','off','
   Algorithm','sqp');
problem = createOptimProblem('fmincon', 'objective', f
   , 'x0', x0, 'lb', lb, 'ub', ub, 'Aineq', ...
A, 'bineq', b, 'nonlcon', @constraintsB, 'options',
   options);
```

```
xopt = run(GlobalSearch, problem);
dens = densitycountB(xopt(2*n+1),pi/4,n);
if k > 10
n = n +1;
k = 0;
end
end
plotProblemB(xopt,n)
disp(dens)
% Matrix A
function A = makeMatrixA(n)
A = zeros(4.*n,2*n+1);
for i = 1:n
A(i, i) = 1;
A(n+i,n+i) = 1;
A(2*n+i,i) = -1;
A(3*n+i,n+i) = -1;
A(:,2*n+1) = 1;
end
end
%Vector a
function [b] = makeVectorA(n)
b = [ones(2*n,1);zeros(2*n,1)];
end
function dens = densitycountB(r, tot,amount)
circlearea = pi.*r.^2;
totcircle = amount.*circlearea;
dens = totcircle/tot;
end
function plotProblemB(xopt,n) %A square
radius = 1;
center = [0,0];
line([0 0], [0 1], 'LineWidth',2,'Color','blue')
hold on
line([0 1], [0 0], 'LineWidth',2,'Color','blue')
rectangle('Position', [center-radius, 2*radius, 2*
    radius],...
'Curvature', [1,1], 'EdgeColor', 'blue', 'LineWidth'
    ,2)
xlim([0,1])
ylim([0,1])
viscircles([xopt(1:n),xopt(n+1:2*n)],xopt(2*n+1))
end
function [g, geq] = constraintsB(x) %Fix for all
    constraints
n = (length(x)-1)/2;
geq = [];
m = 2*n + n*(n-1)/2;
g = zeros(m,1);
k=0;
%This is for all ai, bi, when a1, a2... an, b1, b2 ..
    bn, r
for i=1:2*n
k=k+1; % r<=ai
```

```
g(k)= x(2.*n+1)-x(i);
end
%For the distance constraint
for i=1:n-1
for j=i+1:n
k = k+1;
d= (x(i)-x(j))^2 +(x(n+i)-x(n+j))^2;
g(k) = 4*x(2*n+1)^2 - d;
end
end
%Circle constraint
for i=1:n
d= sqrt(x(i)^2 + x(n+i)^2);
g(end+1) = d + x(end) -1;
end
end

%% Problem 3
clear all
clc
n = 12;
f = @(x) -pi.*sum(x(2*n+1:3*n).^2);
dens = 0;
triangle = (1/3* 1/3)/2;
quartcirc = (pi * (1/4)^2)/4;
cir = pi * (1/4)^2;
total = 1-triangle-quartcirc-cir;
k=0;
while dens < 0.85 %Change to 0.9 for second part
x0 = rand(3*n, 1);
lb = [zeros(n,1); zeros(n,1); zeros(n,1)];
ub = [ones(n,1); ones(n,1); ones(n,1)./2];
A = makeMatrixA(n);
b = makeVectorA(n);
options = optimoptions('fmincon','Display','off','
    Algorithm','sqp');
problem = createOptimProblem('fmincon', 'objective', f
    , 'x0', x0, 'lb', lb, 'ub', ub, 'Aineq', ...
A, 'bineq', b, 'nonlcon', @constraintsC, 'options',
    options);
xopt = run(GlobalSearch, problem);
dens = densitycountC(xopt(2*n+1:3*n),total);
if k == 15
n =+1;
k = 0;
end
end
plotProblemC(xopt,n)
function [g, geq] = constraintsC(x)
n = (length(x))/3;
geq = [];
g = zeros(1,1);
%This is for all ai, bi, when a1, a2... an, b1, b2 ..
    bn, r
```

```
for i=1:n
g(end+1)= x(2.*n+i)-x(i);
end
for i=n+1:2*n
g(end+1)= x(n+i)+x(i)-1;
end
%Circle constraint
for i=1:n
d= sqrt((x(i)-(2/3))^2 + (x(n+i)-(1/4))^2);
g(end+1) = -(d - x(2*n+i) -1/4);
end
%Quarter circle
for i=1:n
d= sqrt((x(i)-(1))^2 + (x(n+i)-(1))^2);
g(end+1) = -(d - x(2*n+i) -1/4);
end
%For the distance constraint
for i=1:n-1
for j=i+1:n
d= sqrt((x(i)-x(j))^2 +(x(n+i)-x(n+j))^2);
g(end+1) = x(2*n+i)+x(2*n+j) - d;
end
end
end
function plotProblemC(xopt,n)
line([0 1],[0 0], 'LineWidth',2,'Color','blue')
hold on
line([0 0], [0 1], 'LineWidth',2,'Color','blue')
line([0 1], [1 1], 'LineWidth',2,'Color','blue')
line([1 1], [0 1], 'LineWidth',2,'Color','blue')
line([0 1/3], [2/3 1], 'LineWidth',2,'Color','blue')
radius = 1/4;
center = [2/3,1/4];
rectangle('Position', [center-radius, 2*radius, 2*
   radius],...
'Curvature', [1,1], 'EdgeColor', 'blue', 'LineWidth'
   ,2)
xlim([0,1])
ylim([0,1])
radius = 1/4;
center = [1,1];
rectangle('Position', [center-radius, 2*radius, 2*
   radius],...
'Curvature', [1,1], 'EdgeColor', 'blue', 'LineWidth'
   ,2)
xlim([0,1])
ylim([0,1])
viscircles([xopt(1:n),xopt(n+1:2*n)],xopt(2*n+1:3*n))
end
function fopt = densitycountC(r, tot) %Fixa circlarna
    r  ju olika
circlearea = sum(pi.*r.^2);
totcircle = circlearea;
fopt = totcircle/tot;
```

```
end
% Matrix A
function A = makeMatrixA(n)
A = zeros(4*n, 3*n);
k = 0;
for i = 1:n
k = k+1;
A(k,n+i)=1;
A(k,2*n + i) = 1;
end
for i = 1:n
k = k+1;
A(k,n+i)= -1;
A(k,2*n + i) = 1;
end
for i = 1:n
k = k+1;
A(k,i)= -1;
A(k,2*n + i) = 1;
end
for i = 1:n
k = k+1;
A(k,i)=1;
A(k,2*n + i) = 1;
end
for i = 1:n
k = k+1;
A(k,i)= -1;
A(k,n + i) = 1;
A(k, 2*n+i) = sqrt(2);
end
for i = 1:n
k = k+1;
A(k,i)= -1;
A(k,n + i) = 1;
end
end
%Vector a
function [b] = makeVectorA(n)
b = [ones(2*n,1); zeros(2*n,1); (2/3).*(ones(2*n,1))
   ];
end

%% Problem 4
clear all
clc
n = 12;
f = @(x) -pi.*sum(x(2*n+1:3*n).^2);
total = pi*(3/4);
dens = 0;
k=0;
while dens <0.85 %Change to 0.88 on the second part
k=+1;
x0 = -1 + 2.*rand(3*n,1);
```

```
lb = [-ones(n,1); -ones(n,1); zeros(n,1)];
ub = [ones(n,1); ones(n,1); ones(n,1)./2];
options = optimoptions('fmincon','Display','off','
    Algorithm','sqp');
problem = createOptimProblem('fmincon', 'objective', f
    , 'x0', x0, 'lb', lb, 'ub', ub,...
'nonlcon', @constraints, 'options', options);
xopt = run(GlobalSearch, problem);
dens = densitycountD(xopt(2*n+1:3*n),total);
if k == 15
k = 0;
n =+1;
end
end
plotProblemD(xopt,n)
function [g, geq] = constraints(x) %a1, a2, .. b1, b2
    ... r1, r2 .. rn
n = (length(x))/3;
geq = [];
m = 2*n + n*(n-1)/2;
g = zeros(m, 1);
k = 0;
for i=1:n-1
for j =i+1:n
k=k+1;
d=sqrt((x(i)-x(j))^2 + (x(n+i)-x(n+j))^2);
g(k) = (x(2*n+i)+(x(2*n+j)-d));
end
end
% Circle constraint
for i=1:n
k=k+1;
g(k) = x(2*n+i) + sqrt(x(i)^2 + x(n+i)^2)-1;
end
% Constraints to skip the fourth quadrant
for i = 1:n
k=k+1;
g(k) = (x(i)<=0 && x(n+i)<=0)*(x(i)+x(2*n+i));
k=k+1;
g(k) = (x(i) >= 0 && x(n+i)>=0)*(x(2*n+i)-x(n+i));
k=k+1;
g(k) = (x(i) >= 0 && x(n+i)<=0);
k=k+1;
g(k) = -x(i)^2 - x(n + i)^2 + x(2 * n + i)^2;
end
end
function plotProblemD(xopt,n)
line([0 1], [0 0], 'LineWidth',2,'Color','blue')
line([0 0], [-1 0], 'LineWidth',2,'Color','blue')
radius = 1;
center = [0,0];
rectangle('Position', [center-radius, 2*radius, 2*
    radius],...
```

```
'Curvature', [1,1], 'EdgeColor', 'blue', 'LineWidth'
    ,2)
xlim([-1,1])
ylim([-1,1])
viscircles([xopt(1:n),xopt(n+1:2*n)],xopt(2*n+1:3*n))
end
function fopt = densitycountD(r, tot)
circlearea = sum(pi.*r.^2);
fopt = circlearea/tot;
end
```