**UMEÅ UNIVERSITET**
**Department of Mathematics and Mathematical Statistics**

**October 30, 2023**

# Laboration 2
# **Continuous Optimization**

## Disk Jockeys Dream

## version 1.0

**Name** Harald Bjurulf
**Email** habj0053@ad.umu.se

**Name** Barbora Štěpánková
**Email** barca-stepankova@seznam.cz

**Graders**
Per-Håkan Lundow
Alp Yurtsever

# Contents

# 1 Disclaimer

We decided to work together very late in the process after we had both struggled with problem 2. We were both mostly done with problems 1 and 2 but had very little time left for problems 3 and 4. We therefore decided to collaborate for problems 3 and 4 and share ideas and insights. This means that there are 2 different code bases and that the code for solving problem 3 and 4 share many ideas and inspirations but the implementation details are different.

Since we were originally supposed to peer review each other and it is Sunday it is highly unlikely that we will be able to receive peer review from another group. However, we have read the parts written by the other to find areas of improvement and possibly missed parts of the assignment.

Per-Håkan Lundow was informed of our decision to work together as soon as it was made. Hopefully this does not disqualify our report.

Sincerly,
Harald Bjurulf and Barbora Štěpánková

# 2 Introduction

The report will consider the problem of disk packing with a fixed number of disks in various regions. Four separate problems will be considered. Half of them will use disks with equal radii and the other half will use disks with variable radii.

# 3 Models and problems

Each problem has a different geometry and thus a different model. However all the models will share certain commonalities. Firstly each disk is best represented by $(x_1, x_2, r)^T$ where $(x_1, x_2)^T$ is its center and $r$ is its radius. It is preferable to use the radius instead of the radius squared because it is easier to decide if two disks overlaps using the non squared radii. A natural target function for each problem will be the sum of the area of the disks.

## 3.1 Problem 1 - Equal Radii packing in a square

The problem is to find a lower limit to the density of packing 49 disks with equal radii in a square. For ease of calculations the unit square $X = \{x \in \mathbb{R}^2 : x_1 \geq 0,\, x_1 \leq 1,\, x_2 \geq 0,\, x_2 \leq 1\}$ is chosen.

All disks have identical radii. Therefore one variable is sufficient for every radius. Every disk needs two variables for its center. Let $n$ be the number of disks. In the model $x \in \mathbb{R}^{2n+1}$ is sufficient to determine the state. Disk $i$ is centered at $(x_{2i-1}, x_{2i})^T$ with radius $x_{2n+1}$. The goal is to maximise the packing density. That is equivalent to maximising the area of all disks. However, the disks have the same radius and thus area. Therefore it is equivalent to just maximise the radius. A suitable target function for a minimisation phrasing of the problem can be seen in equation 1. The resulting packing density is $d_n = \frac{n\pi f(x)^2}{1}$.

$$\begin{aligned} f &: \mathbb{R}^{2n+1} \to \mathbb{R} \\ f(x) &= -x_{2n+1} \end{aligned} \tag{1}$$

Equation 2 constraints the circles to being within the square $X$. The conditions in equation 3 guarantees that the distance between the centres of the disks is at least twice their radius and therefore the disks at most intersect on their boundary. Equation 3 also includes that the radius is positive.

$$\begin{aligned} i &= \{1, 2, ..., 2n\} \\ -x_i + x_{2n+1} &\leq 0 \\ x_i + x_{2n+1} - 1 &\leq 0 \end{aligned} \tag{2}$$

$$\begin{aligned} i, j &\in \{1, 2, ..., n\} \\ i &\neq j \\ g_{i,j} &: \mathbb{R}^{2n+1} \to \mathbb{R} \\ g_{i,j}(x) &= (2x_{2n+1})^2 - (x_{2i-1} - x_{2j-1})^2 - (x_{2i} - x_{2j})^2 \\ g_{i,j}(x) &\leq 0 \\ -x_{2n+1} &\leq 0 \end{aligned} \tag{3}$$

Equation 4 contains the resulting model from the target function in equation 1 and the conditions in equations 2 and 3.

$$\begin{aligned} \min f(x)& \\ -x_i + x_{2n+1} \leq 0, &\quad i = 1, ..., 2n \\ x_i + x_{2n+1} - 1 \leq 0, &\quad i = 1, ..., 2n \\ -x_{2n+1} \leq 0& \\ g_{i,j}(x) \leq 0, &\quad i, j \in \{1, 2, ..., n\}, \, i \neq j \end{aligned} \tag{4}$$

## 3.2 Problem 2 - Equal radii packing in quarter disk

The assignment is to find $N(0.8)$, the lowest number of disks for which the density is at least $0.8$, for the shape $X = \{x \in \mathbb{R}^2 : x_1 \geq 0, x_2 \geq 0, x_1^2 + x_2^2 \leq 1\}$. The area of X is $A = \frac{1}{4}\pi 1^2 = \frac{\pi}{4}$.Since the problem uses fixed radii the condition in equation 3 can be reused as can the target function in equation 1.

A suitable condition for the circle part of the boundary of $X$ is derived by considering that a circle $(x_1, x_2, r)^T$ is within another circle centered at the origin with radius $1$ if $\sqrt{x_1^2 + x_2^2} + r \leq$

1 which can be rewritten as $\sqrt{x_1^2 + x_2^2} \leq 1 - r$. If an additional condition $r \leq 1$ is added then both the left and right side of the inequality is positive and can safely be squared resulting in the condition $x_1^2 + x_2^2 \leq (1-r)^2 \implies x_1^2 + x_2^2 - (1-r)^2 \leq 0$. The resulting optimisation problem can be seen in equation 5. The density of a specific packing is $d = \frac{n\pi f(x)^2}{A} = \frac{n\pi f(x)^2}{\frac{\pi}{4}} = 4nf(x)^2$

$$\begin{aligned} \min f(x) & \\ -x_i + x_{2n+1} \leq 0, &\quad i = 1, ..., 2n \\ x_{2j-1}^2 + x_{2j}^2 - (1 - x_{2n+1})^2 \leq 0, &\quad j = 1, ..., n \\ -x_{2n+1} \leq 0 & \\ x_{2n+1} - 1 \leq 0 & \\ g_{i,j}(x) \leq 0, &\quad i, j \in \{1, 2, ..., n\},\ i \neq j \end{aligned} \tag{5}$$

## 3.3 Problem 3 - Variable radii in obstructed square

The problem considers packing in the shape $X$ given by the equations below. The area of the $X$ is $A = 1 - \frac{1}{18} - \frac{5\pi}{64}$. The goal is to find a reasonable upper bound for $N(0.85)$ and for $N(0.9)$.

$$\begin{aligned} X_1 &= \left\{x \in \mathbb{R}^2 : x_1 \geq 0,\ x_1 \leq 1,\ x_2 \geq 0,\ x_2 \leq 1\right\} \\ X_2 &= \left\{x \in \mathbb{R}^2 : x_2 - x_1 \leq \frac{2}{3},\right\} \\ X_4 &= \left\{x \in \mathbb{R}^2 : (x_1 - \frac{2}{3})^2 + (x_2 - \frac{1}{4})^2 \geq \left(\frac{1}{4}\right)^2\right\} \\ X_4 &= \left\{x \in \mathbb{R}^2 : (x_1 - 1)^2 + (x_2 - 1)^2 \geq \left(\frac{1}{4}\right)^2\right\} \\ X &= X_1 \cap X_2 \cap X_3 \cap X_4 \end{aligned}$$

In the model with $n$ circles the state will consist of $x \in \mathbb{R}^{3n}$ where each triple $(x_{3i-2}, x_{3i-1}, x_{3i})^T$ is considered one circle with center at $(x_{3i-2}, x_{3i-1})^T$ and radius $x_{3i}$. An appropriate target function is seen in equation 6. The density of a specific packing is $d = \frac{-\pi f(x)}{A}$.

$$\begin{aligned} f &: \mathbb{R}^{3n} \to \mathbb{R} \\ f(x) &= -\sum_{i=1}^{n} x_{3i}^2 \end{aligned} \tag{6}$$

Equation 7 gives a condition for non overlapping disks. It achieves this by verifying that the squared distance between centres of disks is at least the square of the sum of their radii.

$$
\begin{aligned}
&i, j \in \{1, ..., n\} \\
&i \neq j \\
&g_{i,j} : \mathbb{R}^{3n} \to \mathbb{R} \\
&g_{i,j}(x) = (x_{3i} + x_{3j})^2 - (x_{3i-2} - x_{3j-2})^2 - (x_{3i-1} - x_{3j-1})^2 \\
&g_{i,j}(x) \leq 0
\end{aligned} \tag{7}
$$

Constraining the disks to the set $X$ is easily achieved by simultaneously constraining them to the sets $X_1, X_2, X_3, X_4$. $X_1$ is a simple square. $X_3$ and $X_4$ is essentially solved by equation 7 as it can be seen as non overlapping disks. $X_2$ can be solved by using the dot product with an orthogonal vector to find the directional distance from the line $x_2 - x_1 = \frac{2}{3}$. Equation 8 constrains the disks to $X$ and requires that the radius is positive.

$$
\begin{aligned}
i &\in \{1, ..., n\} \\
-x_{3i-2} + x_{3i} &\leq 0 \\
-x_{3i-1} + x_{3i} &\leq 0 \\
x_{3i-2} + x_{3i} - 1 &\leq 0 \\
x_{3i-1} + x_{3i} - 1 &\leq 0 \\
\left(\frac{1}{4} + x_{3i}\right)^2 - \left(x_{3i-2} - \frac{2}{3}\right)^2 - \left(x_{3i-1} - \frac{1}{4}\right)^2 &\leq 0 \\
\left(\frac{1}{4} + x_{3i}\right)^2 - (x_{3i-2} - 1)^2 - (x_{3i-1} - 1)^2 &\leq 0 \\
\frac{-x_{3i-2} + x_{3i-1} - \frac{2}{3}}{\sqrt{2}} + x_{3i} &\leq 0 \\
-x_{3i} &\leq 0
\end{aligned} \tag{8}
$$

Equation 9 contains the complete optimisation problem for problem 3.

$$
\begin{aligned}
\min f(x) \\
i, j \in \{1, ..., n\} \\
i \neq j \\
g_{i,j}(x) \leq 0 \\
-x_{3i-2} + x_{3i} \leq 0 \\
-x_{3i-1} + x_{3i} \leq 0 \\
x_{3i-2} + x_{3i} - 1 \leq 0 \\
x_{3i-1} + x_{3i} - 1 \leq 0 \\
\left(\frac{1}{4} + x_{3i}\right)^2 - \left(x_{3i-2} - \frac{2}{3}\right)^2 - \left(x_{3i-1} - \frac{1}{4}\right)^2 \leq 0 \\
\left(\frac{1}{4} + x_{3i}\right)^2 - (x_{3i-2} - 1)^2 - (x_{3i-1} - 1)^2 \leq 0 \\
\frac{-x_{3i-2} + x_{3i-1} - \frac{2}{3}}{\sqrt{2}} + x_{3i} \leq 0 \\
-x_{3i} \leq 0
\end{aligned}
\tag{9}
$$

## 3.4 Problem 4 - Variable radii in three quarter Disk

The problem is to determine if $d_{12} \geq 0.85$ as well as finding reasonable upper bounds on $N(0.85)$ and $N(0.88)$. This problem is somewhat trickier as it seems necessary to include a non differentiable function to handle the sharp edge at $(0, 0)^T$. Equation 7 can be reused. Containing the area to a disk is similar to problem 2.

The intepretation of $x$ from problem 3 is reused. The tricky condition is excluding a quarter disk. This can be done by the condition in equation 10 which is derived by considering the closest point to the lines making up the quadrant. The target function in equation 6 is reused. The area of the unit disk with one quadrant removed is $A = \frac{3\pi}{4}$.

$$
\begin{aligned}
& i = 1, ..., n \\
& h_i(x) = \begin{cases} x_{3i}^2 - x_{3i-2}^2 - x_{3i-1}^2, & \text{for } x_{3i-2} \leq 0 \text{ and } x_{3i-1} \geq 0 \\ x_{3i}^2 - x_{3i-2}^2, & \text{for } x_{3i-2} \leq 0 \text{ and } x_{3i-1} < 0 \\ x_{3i}^2 - x_{3i-1}^2, & \text{for } x_{3i-2} > 0 \text{ and } x_{3i-1} \geq 0 \\ x_{3i}^2 + x_{3i-2}^2 + x_{3i-1}^2, & \text{for } x_{3i-2} > 0 \text{ and } x_{3i-1} < 0 \end{cases} \\
& h_i(x) \leq 0
\end{aligned}
\tag{10}
$$

Combining it all in one model gives the problem in equation 11.

$$
\begin{aligned}
\min f(x) \\
i, j \in \{1, ..., n\} \\
i \neq j \\
g_{i,j}(x) \leq 0 \\
h_i(x) \leq 0 \\
x_{3i-2}^2 + x_{3i-1}^2 - (1 - x_{3i})^2 \leq 0
\end{aligned}
\tag{11}
$$

# 4 Method

All problems are solved by defining the various conditions in a matlab function and then trying the three inbuilt functions `MultiStart`, `ga`, and `GlobalSearch` in various configurations.

## 4.1 Problem 1

The optimisation problem is solved for $n = 49$.

## 4.2 Problem 2

An approximation of $d_n^*$ is calculated for increasing $n$. If the approximation is close to $0.8$ then a more accurate approximation is calculated. The process is slow but necessary as it is not guaranteed to be increasing in $n$. In practice it is difficult to perform the search fully automatically and the conditions and starting $n$ was tweaked over multiple runs to find an acceptable upper limit. The code seems to get stuck if matlab is not restarted every so often. Using `GlobalSearch` seems particularly unreliable and too slow to practically useful.

## 4.3 Problem 3

Problem 3 was solved partially by `GlobalSearch` and partially by `Multisearch`, with the best solution further refined by a custom method based on the mutation part of a genetic algorithm. It proved to be very efficient to repatedly randomly change the coordinates of the smallest disk in an existing packing in order to try to find an improved packing.

## 4.4 Problem 4

For solving problem 4 `GlobalSearch` was less useful for refining a solution created by `MultiSearch`. The smallest-disk-moving method described above was used instead.

# 5 Result and discussion

The behaviour of the results are highly dependant on the exact configuration of the problem. That said `ga` has reliably supplied poor results. However, that might be solvable by choosing better meta parameters.

## 5.1 Problem 1

Somewhat consistently `MultiStart` terminates faster while `GlobalSearch` gives somewhat better results. First using `MultiSearch` with a low number of starting points combined with using GlobalSearch on the solution from `MultiSearch` seems to give acceptable performance and supplies better solutions than only using `MultiSearch`.

In figure 1 a packing of a square with density $d \geq 0.79$ can be seen. Note that the packing mostly follows hexagonal packing for the leftmost part of the square and the remaining space seems unordered. The packing is almost certainly not optimal.

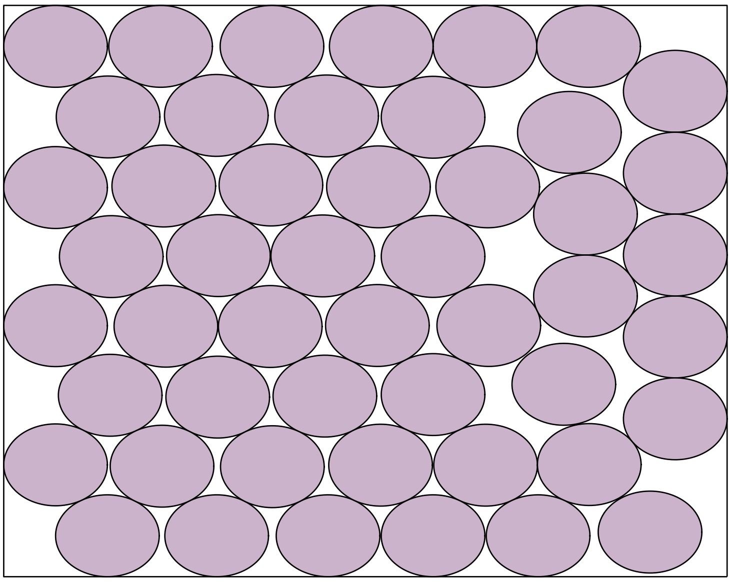


Figure 1: Packing of square with density $d = 0.7905$

## 5.2 Problem 2

Deciding $N(0.8)$ is tricky as accurately determining $d_n$ for high $n$ is slow. The method used to determine $N(0.8)$ leaves a high degree of uncertainty and the actual value could be significantly lower. The problem is fundamentaly limited by how long it is given to run. The best found approximation is $N(0.8) \leq 54$. A packing for $n = 54$ can be seen in figure 2. To find a substantially better approximation would need substantially more run time or a more efficient method.

The method used just stops for certain cases and it seems to do so without crashing inside `MultiStart`. This makes it harder to find good guesses. Multiple $n$ smaller than 54 are of interest but have failed to run. Of particular note is $n = 48, 52, 53$ which have confirmed packings with density $d = 0.7928, 0.7963, 0.7972$.

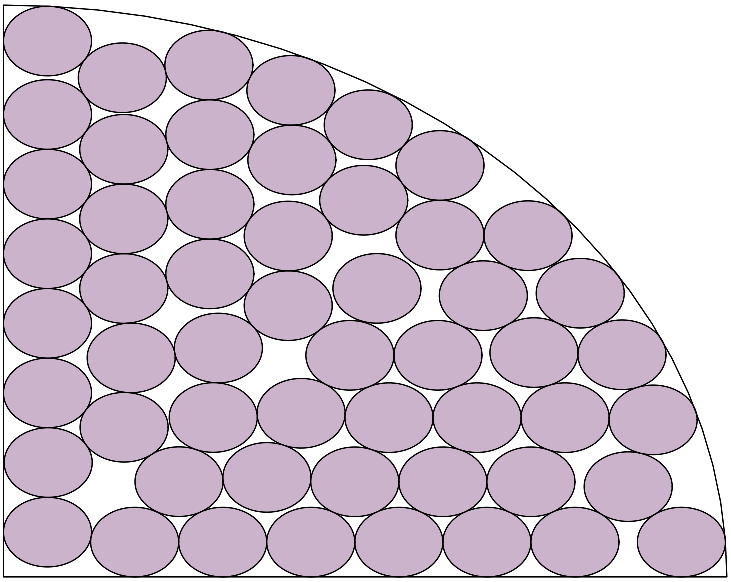


Figure 2: Packing of quarter circle using 54 disks with density $d = 0.8002$

## 5.3 Problem 3

When running the `GlobalSearch` in a loop, using the previous results as new starting points, a local optimum is quickly reached and the density of the covering stops improving.

To combat this problem, the circle with the smallest radius is found and its position is changed to random points.

At first, it seemed that a good upper bound for $N(0.85)$ could be $n = 15$, but after running the loop for a considerable amount of time, the best density was $d = 0.847875$.

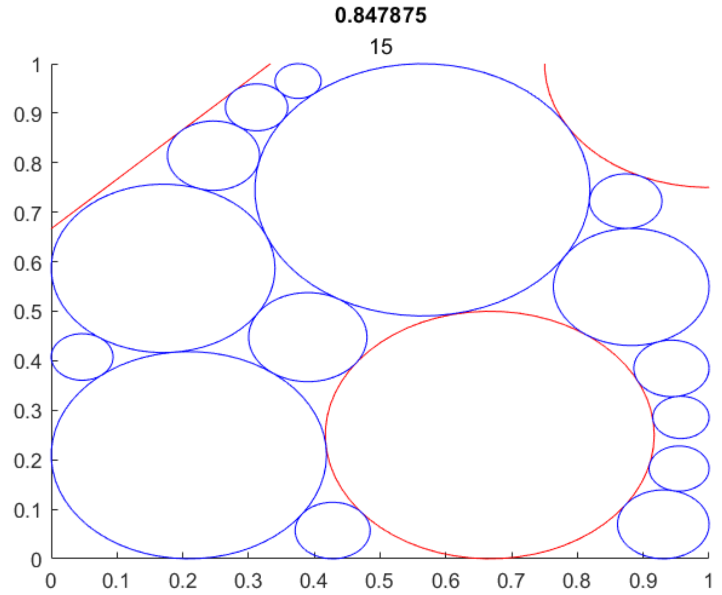
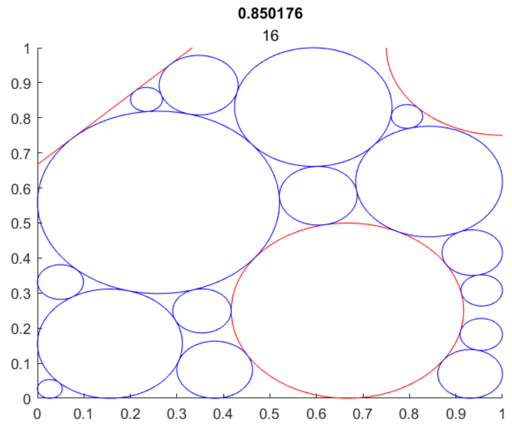
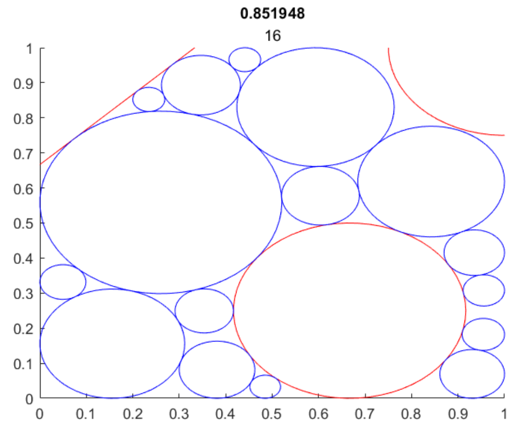



Figure 3: Packing of a constrained square using 15 disks with density $d = 0.847875$

When increasing the number of circles to 16, the best-achieved density was $d = 0.851948$. This packing was achieved by repeatedly changing the position of the smallest circle. Below, we can see a packing with $d = 0.850176$ before the change and packing with $d = 0.851948$ after the change.




(a) Packing of a constrained square with density $d = 0.850176$




(b) Packing of a constrained squarewith density $d = 0.851948$

Figure 4: Comparison of 16 discs packings before and after changing positions of smallest circle

The method using `GlobalSearch` and moving the smallest circle proved to be too slow for the scale of this problem, so `MultiStart` with $10 \cdot n$ starting points and subsequential refining of the solution with the smallest-circle-moving function `refine_solution` was used.

By this method, it seemed that a good upper bound for $N(0.90)$ could be $n = 35$, but the best achieved density for $n = 35$ was $d = 0.898259$.

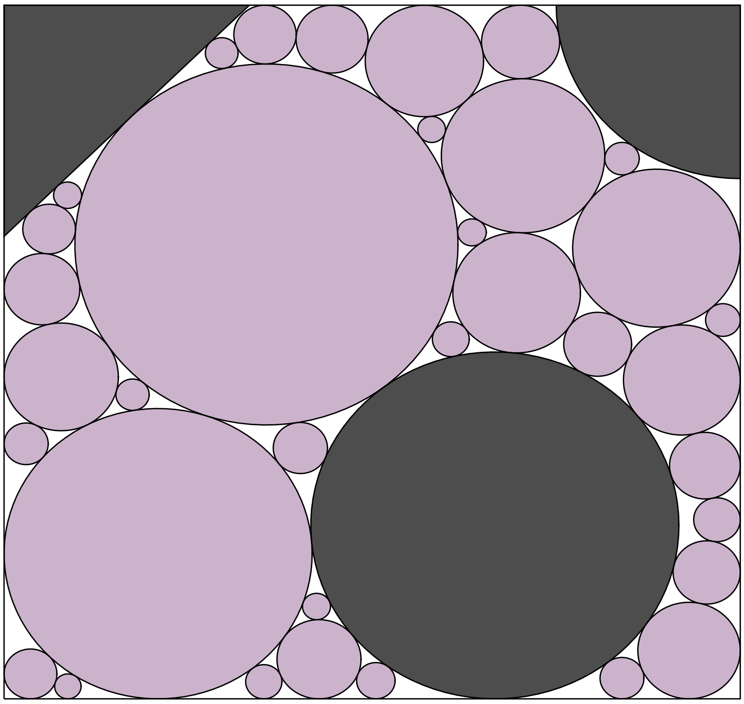


Figure 5: Packing of a constrained square using 35 disks with density $d = 0.898259$

The upper bound found for the $N(0.90)$ is $n = 36$ with the density $d = 0.902022$

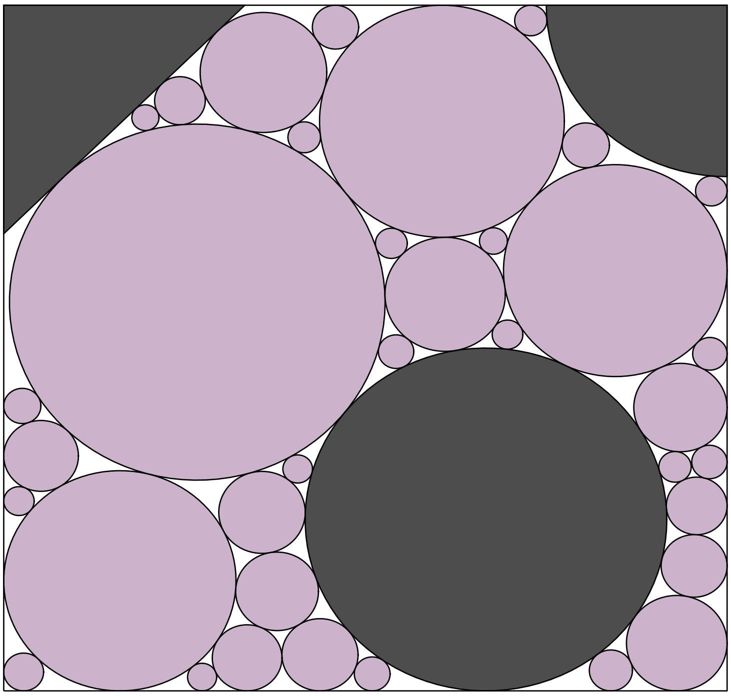
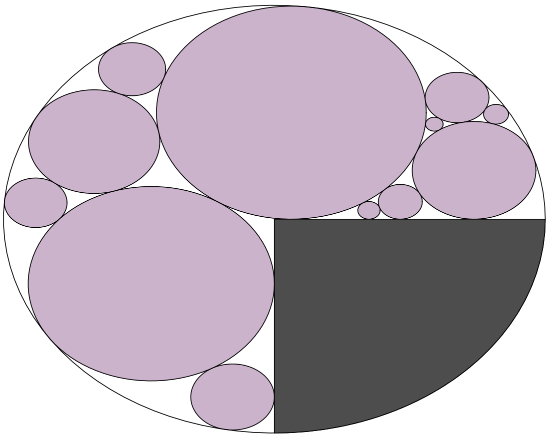


Figure 6: Packing of a constrained square using 36 disks with density $d = 0.902022$

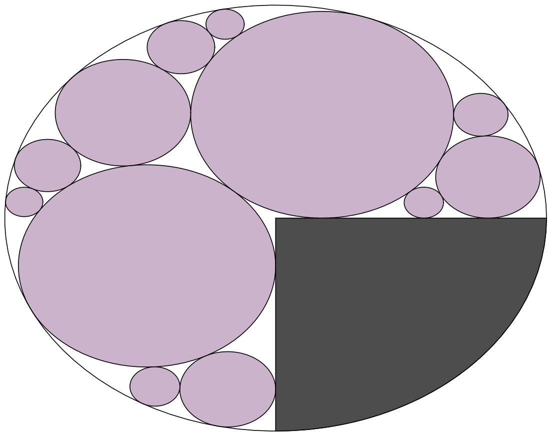
## 5.4 Problem 4

In figure 7 can be seen the profound difference, that moving the smallest circle randomly and the optimising again, has on the quality of the solution.

The packing in sub-figure 7a has been generated by `MultiStart` with 1000 starting points while the packing in sub-figure 7b just uses an extra 1000 starting points in batches while randomly moving the smallest circle in each batch. This results in a better density than `MultiStart` with 10000 starting points.



(a) Packing of 3 quarter circle non refined with $d = 0.857845$



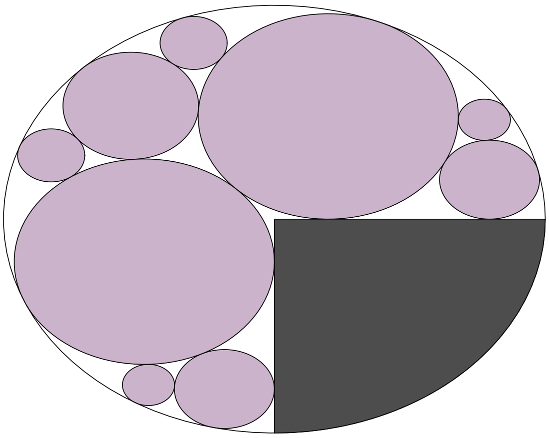
(b) Refined packing of 3 quarter circle with $d = 0.874870$

Figure 7: Refined vs unrefined packing for 3 quarter circle with 12 disks

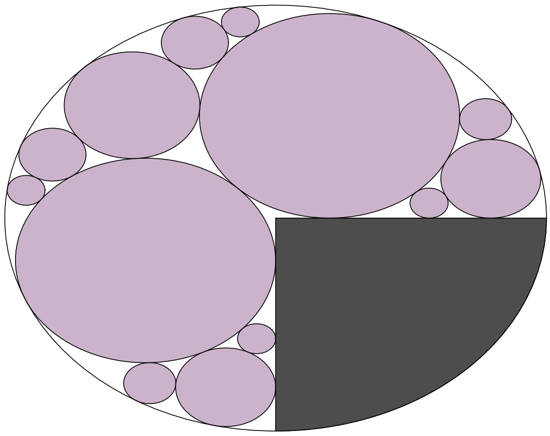
$N(0.85)$ is approximated to $N(0.85) \leq 9$ as one packing with 9 disks with $d_9 \geq 0.85$ can be seen in sub-figure 8a. No packing for $n = 8$ was found to have density 0.85 or higher.

Since $N(0.85) \leq 9$ it follows that there is a packing for $n = 12$ with density $d_{12} \geq 0.85$. The smallest $n$ found with a packing of density $d \geq 0.88$ was $n = 13$.

That packing can be seen in sub-figure 8b. Thus the approximation for $N(0.88)$ is $N(0.88) \leq 13$. By studying figure 8 it is apparent that the packings are very similar. This naturally suggests that one method of finding the starting point for $n = k$ is to first find an efficient packing for $n = k - 1$ and then adding another circle.



(a) Packing of 3 quarter circle with 9 discs with density $d = 0.855046$



(b) Packing of 3 quarter circle with 13 discs with density $d = 0.881259$

Figure 8: Comparison of packings of 3-quarter disk with different $n$ of discs.

## 5.5 Sources of error

A big problems for any conclusions and results drawn from this program is that no real thought or study have gone into analysing the size of errors. Primarily because we do not know the details of how $GlobalSearch$, $MultiSearch$, and $fmincon$ works or more importantly how the various tolerances can be interpreted.

The largest source of error is probably the fact that no efficient method was found for solving the models in problem 2 which leads to the run time for an accurate result being unreasonably long which means less accurate methods were utilised which causes a high degree of uncertainty.

# A Code

## A.1 Repository

The code used for presenting problem 1, 2, and 4 can be found at repository `https://git.cs.umu.se/c20hbf/cont-opt-lab-2`. The code used for presenting problem 3 can be found at repository
`https://github.com/stepankovab/Continuous-Optimization-Umea`.

## A.2 plot_circles_equal_radius.m

Code for plotting circles with equal radii.

```
function plot_circles_equal_radius(x,n)
% plot_circles_equal_radius Plots All the circles contained in x to the
%  current plot
%
% INPUT:
%   x   Column vector of length 2n+1. Each disk has center=(x(2i-1), x(2i))
%         with radius x(2n+1)
%   n   The total number of disks

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
%   2023-10-27 initial implementation

hold on
angles = linspace(0, 2*pi, 100)';
x_p = cos(angles);
y_p = sin(angles);
%for some reason on my machine on my verion of matlab using green gives
%rainbows around the shape
c=[0.8 0.7 0.8];
r = x(2*n + 1);
%scale to radius
x_p = x_p * r;
y_p = y_p * r;
for i = 1:n
    x_plot = x_p + x(2*i - 1);
    y_plot = y_p + x(2*i);
    fill(x_plot, y_plot, c);
end
hold off
end
```

## A.3 constant_radius_constraints.m

Code for calculating function $g_{i,j}$ in equation 3.

```
function [c,num] = constant_radius_constraints(x,n)
% constant_radius_constraints Calculates constraitns for non overlapping
```

```
% disks with equal radius
%
% INPUT:
%   x   Column vector of length 2n+1. Each disk has center=(x(2i-1), x(2i))
%         with radius x(2n+1)
%   n   The total number of disks
% Output:
%   c   A vector of size n*(n-1). Contains values that are each 0 or
%        negative if the disks defined by x are non overlapping
%   num n*(n-1) the size of c


%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
%   2023-10-27 initial implementation

num = n*(n-1);
r = x(2*n+1);
I = 1:n;
J = 1:n;
[I,J]=meshgrid(I,J);
I = reshape(I, n*n, 1);
J = reshape(J, n*n, 1);
choice = I~=J;
I=I(choice);
J=J(choice);
fourRSquared = 4*r.^2;
DeltaX1 = x(2*I - 1) - x(2*J - 1);
DeltaX2 = x(2*I) - x(2*J);
c = fourRSquared - DeltaX1.^2 - DeltaX2.^2;
end
```

## A.4 constraints_p1.m

Code for calculating the constraints in equation 4.

```
function [c,ceq] = constraints_p1(x, n)
% constraints_p1 Calculates constraint functions for problem 1 as per
%  https://www.canvas.umu.se/courses/10377/assignments/117252
%
% INPUT:
%   x   Column vector of length 2n+1. Each disk has center=(x(2i-1), x(2i))
%         with radius x(2n+1)
%   n   The total number of disks
% OUTPUT:
%   c   A vector of size n*(n-1) + 4*n + 1 where each value is <= 0 if x is
%        a valid vector for problem 1.
%   ceq An empty vector []

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
%   2023-10-27 initial implementation

%The total number of constraints will be n*(n-1) + 4*n + 1

num_constraints = n*(n-1) + 4*n + 1;
c = zeros(num_constraints, 1);
```

```
ceq = [];
%non overlapping circles
[circle_constraints, num] = constant_radius_constraints(x, n);
c(1:num) = circle_constraints;
insertion_index = num + 1;
r = x(2*n + 1);

%Disks constrained to square
for i = 1:2*n
    c(insertion_index) = -x(i) + r;
    insertion_index = insertion_index + 1;
    c(insertion_index) = x(i) + r - 1;
    insertion_index = insertion_index + 1;
end

%radius positive
c(insertion_index) = -r;
end
```

## A.5 problem1_multistart.m

Code for solving problem 1 using multistart.

```
% problem1_multistart Tries to solve problem 1 from
%  https://www.canvas.umu.se/courses/10377/assignments/117252 using the
%  multistart algorithm combined with global search to refine the found
%  solution.

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
n=49;
rng default
close all
opts = optimoptions('fmincon','Algorithm','sqp');
nlc = @(x)constraints_p1(x, n);
target = @(x)-x(2*n + 1);

x0 = rand(2*n + 1, 1);
lb = x0 * 0;
ub = lb + 1;
%The radius can at most be 1
ub(end) = 0.5;


while max(nlc(x0)) > 0
    x0(end) = x0(end) / 2;
end

problem=createOptimProblem("fmincon", "objective", target, "x0", x0, ...
    "ub", ub, "lb", lb, "nonlcon", nlc, "options", opts);
ms =  MultiStart('UseParallel',true);

tic
[x, f] = run(ms, problem, 100);
best_x = x;
best_f = f
```

```
toc

gs = GlobalSearch;
x0 = x;
x0(end) = 0;
problem=createOptimProblem("fmincon", "objective", target, "x0", x0, ...
    "ub", ub, "lb", lb, "nonlcon", nlc, "options", opts);
tic
[x, f] = run(gs, problem);
toc
if f < best_f
    best_f = f
    best_x = x;
end


plot_circles_equal_radius(best_x, n);
hold on
plot([1 1 0 0 1], [0 1 1 0 0], 'k');
axis([-0.1 1.1 -0.1 1.1])
axis off
exportgraphics(gcf, "problem1.pdf", "ContentType", "vector")
d=n*pi*best_f^2
```

## A.6 constraints_p2_nlc.m

Code for calculating non linear constraints in equation 5.

```
function [c,ceq] = constraints_p2_nlc(x, n)
% constraints_p2 Calculates non linear constrain functions for problem 2 as
%  per https://www.canvas.umu.se/courses/10377/assignments/117252.
%
% INPUT:
%   x   Column vector of length 2n+1. Each disk has center=(x(2i-1), x(2i))
%         with radius x(2n+1)
%   n   The total number of disks
% OUTPUT:
%   c   A vector of size n*(n-1) + n where each value is <= 0 if x is
%         a valid vector for problem 1.
%   ceq An empty vector []

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)

%The total number of constraints will be n*(n-1) + 4*n + 1

num_constraints = n*(n-1) + n;
c = zeros(num_constraints, 1);
ceq = [];
%non overlapping circles
[circle_constraints, num] = constant_radius_constraints(x, n);
c(1:num) = circle_constraints;
insertion_index = num + 1;
r = x(2*n + 1);

%Disks constrained to circle
```

```
for i = 1:n
    c(insertion_index) = x(2*i-1).^2 + x(2*i).^2 - (1-r).^2;
    insertion_index = insertion_index + 1;
end

end
```

## A.7 problem_p2.m

Code for defining the model in equation 5.

```
function [lb,ub, A, b, nlc, f] = problem_p2(n)
% problem_p2 Creates the proper variables for the optimisation problem as
%  defined in problem 2 in
%  https://www.canvas.umu.se/courses/10377/assignments/117252
%
% In the model each disk is centered at (x(2i-1), x(2i)) with radis x(2n+1)
% INPUT:
%   n   number of disks
% OUTPUT:
%   lb  A (2n + 1) x 1 column vector with lower bounds x
%   ub  A (2n + 1) x 1 column vector with upper bounds x
%   A   A part of the condition Ax<=b. A 2*n x (2n + 1) matrix
%   b   A part of the condition Ax<=b. A 2*n x 1 column vector of 0
%   nlc A the non linear conditions.
%   f   Target function for optimisation


%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)

%All values are at least 0
lb = zeros(2*n + 1, 1);

%All coordinates are contained <= 1
ub = lb + 1;

%Since the disks are contained in <= 1 the maximum radius is 0.5
ub(end) = 1/2;

%Disks constrained to positive quadrant
A = zeros(2*n, 2*n + 1);
for j = 1:2*n
    A(j, j) = -1;
    A(j, 2*n + 1) = 1;
end
b = zeros(2*n, 1);
nlc = @(x)constraints_p2_nlc(x, n);
f = @(x)-x(2*n + 1);
end
```

## A.8 problem2_solve.m

Code for solving problem 2. Tends to get stuck and needs to be restarted.

```
% problem2_solve Solves problem 2 as defiend in
%  https://www.canvas.umu.se/courses/10377/assignments/117252

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
close all
clear all
rng default
%We are trying to find N(0.8)
% We start by deciding on an upper bound

%Using interior point leads to terrible solutions. sqp gives better
%solutions. At least that is the case for problem 1
opts = optimoptions('fmincon','Algorithm','sqp');
delete(gcp('nocreate'))
ms =  MultiStart('UseParallel',true, "Display","iter")%,'Display','iter');
gs = GlobalSearch("Display","iter");

d=@(f, n)4*n*f^2;
%Have previously tested upto 54
%
%Of interest is
% n     d_n
% 48    0.7928
% 52    0.7963
% 53    0.7972
% 54    0.7996
% 55   >0.7950

%70 is definietly an upper limit
n = 51;
f = 0;
max_deltad = 0;
while d(f, n)<0.8
    n = n + 1;
    tic
    [lb,ub, A, b, nlc, target]=problem_p2(n);
    %x0 not guaranteed to be a valid starting point!
    x0 = rand(size(lb)) .* (ub - lb) + lb;
    x0(end)=0;
    problem = createOptimProblem('fmincon','objective', target, 'x0', ...
        x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
        'options', opts);
    [x, f] = run(ms, problem, 100);
    time = toc;
    if d(f, n) > 0.79 && d(f, n) < 0.8
        old_x = x;
        old_f = f;
        fprintf("Finer  search for n=%d, d=%1.4f, expected time = %6.1f",
         ↪ n, d(f,n),toc * 10)
        x0 = x;
        x0(end)=x0(end)/2;
        problem = createOptimProblem('fmincon','objective', target, 'x0',
         ↪ ...
            x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc,
             ↪ ...
            'options', opts);
        [x, f] = run(ms, problem, 1000);
        if old_f < f
```

```
                f = old_f;
                x = old_x;
            end
        end
        if d(f,n) > 0.796 && d(f, n) < 0.8
            fprintf("Using GlobalSearch for n=%d, d=%1.4f", n, d(f,n))
            old_x = x;
            old_f = f;
            x0 = x;
            x0(end) = x0(end) / 2;
            problem = createOptimProblem('fmincon','objective', target, 'x0',
             ↪  ...
                x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc,
                 ↪  ...
                'options', opts);
            [x, f] = run(gs, problem);
            if old_f < f
                f = old_f;
                x = old_x;
            end
        end
        fprintf("n = %d, gives %1.4f. Time took %5.1f. \n", n, d(f, n), toc);
end
N = n

%ga seems unreliable at best? Might work better for other problems or with
%better options?
%x = ga(target,2*n+1,A,b,[],[],lb,ub,nlc);
plot_circles_equal_radius(x, n);
d=4*n*f^2

hold on
angles = linspace(0, pi/2, 25);
x = cos(angles);
y = sin(angles);
x = [x, 0, 0, 1];
y = [y, 1, 0, 0];
plot(x, y, 'k');
axis([-0.1 1.1 -0.1 1.1])
axis off
exportgraphics(gcf, "problem2.pdf", "ContentType", "vector")
```

## A.9 problem2_test.m

Prototype for `problem2_solve.m`. Useful for studying specific $n$. Was used to find the packing in the report for $n = 54$.

```
%

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
close all
clear variables
clear global

rng default
```

```
%We are trying to find N(0.8)
% We start by deciding on an upper bound

%Using interior point leads to terrible solutions. sqp gives better
%solutions. At least that is the case for problem 1
opts = optimoptions('fmincon','Algorithm','sqp');
delete(gcp('nocreate'))
ms =  MultiStart('UseParallel',true,'Display','iter');
gs = GlobalSearch;

d=@(f, n)4*n*f^2;



n = 54
[lb,ub, A, b, nlc, target]=problem_p2(n);
x0 = rand(size(lb)) .* (ub - lb) + lb;
x0(end)=0;
problem = createOptimProblem('fmincon','objective', target, 'x0', ...
    x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
    'options', opts);
tic
[x, f] = run(ms, problem, 1000);
toc
x_best = x;
f_best = f;
fprintf("Finished n = %d\n", n)
fprintf("D prelim = %1.6f\n", d(f, n));
%if it is a good solution we try to refine it
%fprintf("Found candidate n = %d \n", n)
x0 = x;
x0(end) = x0(end) / 2;
problem = createOptimProblem('fmincon','objective', target, 'x0', ...
    x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
    'options', opts);
[x, f] = run(gs, problem);
if f < f_best
    f_best = f;
    x_best = x;
end
best_d = d(f_best, n);
f=f_best;
x=x_best;

%ga seems unreliable at best? Might work better for other problems or with
%better options?
%x = ga(target,2*n+1,A,b,[],[],lb,ub,nlc);
plot_circles_equal_radius(x, n);
d=4*n*f^2

hold on
angles = linspace(0, pi/2, 25);
x = cos(angles);
y = sin(angles);
x = [x, 0, 0, 1];
y = [y, 1, 0, 0];
plot(x, y, 'k');
axis([-0.1 1.1 -0.1 1.1])
```

```
axis off
if d >= 0.8
    name = sprintf("problem2_%d.pdf", n);
    exportgraphics(gcf, name, "ContentType", "vector")
end
```

## A.10 problem3_main.m

Main function for problem 3. All parameters of the run should be changed here.

```
s_points = 5;
loop_iter = 15;
start_iter = 100;
max_area = 0;
max_circles = 0;

for circles = 25:26
    xopt = zeros(1, 2 * circles + 1);
    x = starting_points_3(circles, s_points);

    for i = 1:start_iter
        for j = 1:loop_iter
            [a, x] = discs_3(circles, x);

            if a > max_area
                max_area = a
                xopt = x
                plot_3(circles, xopt);
                max_circles = circles
            end
        end

        smallest_r = inf;
        smallest_r_i = 0;
        for r = 1:circles
            if xopt(2*circles + r) < smallest_r
                smallest_r_i = r;
                smallest_r = xopt(2*circles + r);
            end
        end

        xopt(smallest_r_i) = rand();
        xopt(circles + smallest_r_i) = rand();
    end
end

aaaa = max_area
ccccc = max_circles
```

## A.11 discs_3.m

Creates parameters and runs GlobalSearch.

```
function [area, xopt] = discs_3(n_discs, start)
%DISCS_3 packs as many discs into a constrained square as computationally
↪ possible.

    options = optimoptions('fmincon','Algorithm','sqp');

    lb = zeros(1,3 * n_discs);
    ub = ones(1,3 * n_discs);

    problem = createOptimProblem('fmincon', 'objective', @f_3, 'x0', start,
    ↪ 'lb', lb, 'ub', ub, 'nonlcon', @constraints_3, 'options', options);

    xopt = run(GlobalSearch, problem);

    area = density_3(xopt);

end
```

## A.12 f_3.m

Objective function to minimize.

```
function neg_area = f_3(x)
%F_3 Objective function for problem 3.

    n = (length(x)) / 3;
    neg_area = 0;

    for i = 1:n
        neg_area = neg_area - x(2*n + i)^2;
    end

end

```

## A.13 constraints_3.m

Function to generate constraints for problem 3.

```
function [g, geq] = constraints_3(x)
%CONSTRAINTS_3 The constraints of a problem 3.

    n = (length(x)) / 3;
    m = (4 * n + (n * (n - 1)) / 2);
    geq = [];
    g = zeros(m, 1);
    k = 1;
```

```

    for i = 1:n
        for j = i+1:n
            g(k) = (x(2*n + i) + x(2*n + j))^2 - (x(i) - x(j))^2 - (x(i +
            ↪  n) - x(j + n))^2;
            k = k + 1;
        end
    end

    for i = 1:n
        g(k) = x(2*n + i) - x(i);
        k = k + 1;
        g(k) = x(2*n + i) - x(n + i);
        k = k + 1;
        g(k) = x(2*n + i) + x(i) - 1;
        k = k + 1;
        g(k) = x(2*n + i) + x(n + i) - 1;
        k = k + 1;
    end


    for i = 1:n
        g(k) = x(2*n + i) + 1/4 - sqrt((1 - x(i))^2 + (1 - x(n + i))^2);
        k = k + 1;
        g(k) = x(2*n + i) + 1/4 - sqrt((2/3 - x(i))^2 + (1/4 - x(n +
        ↪  i))^2);
        k = k + 1;
        g(k) = ((- x(i) + x(n + i) - 2/3) / sqrt(2)) + x(2 * n + i);
        k = k + 1;
    end



end


```

## A.14 starting_points_3.m

Generate random feasible starting points for problem 3.

```
function start = starting_points_3(n_discs,num_start_line)
%STARTING_POINTS_3 Generates feasible starting points for problem 3.

    start = zeros(1, 3 * n_discs);
    s = 1;
    for i = 1:num_start_line
        for j = 1:n_discs / num_start_line
            start(s) = (i) / (3 * (num_start_line + 1));
            start(n_discs + s) = (j) / (3 * (n_discs / num_start_line +
            ↪  1));
            s = s + 1;
        end
        start(2 * n_discs + i) = 1 / (3 * (max(num_start_line, n_discs /
        ↪  num_start_line) + 1));
```

```
    end
end
```

## A.15 density_3.m

Calculate the density of a solution of problem 3.

```
function den = density_3(x)
%DENSITY_3 Computes density of a packing for problem 3.

    n = (length(x)) / 3;
    circle_area = 0;

    for i = 1:n
        circle_area = circle_area + (pi * x(2*n + i)^2);
    end

    to_fill_area = 1 - (pi * (1/4)^2) - 1/4 * (pi * (1/4)^2) - ((1/3)^2 /
    ↪  2);

    den = circle_area / to_fill_area;

end
```

## A.16 constraints_p3_nlc.m

Code for calculating non linear constraints in equation 9.

```
function [c,ceq] = constraints_p3_nlc(x, n)
% constraints_p3 Calculates non linear constrain functions for problem 2 as
%  per https://www.canvas.umu.se/courses/10377/assignments/117252.
%
% INPUT:
%   x   Column vector of length 3n. Each disk has center=(x(3i-2), x(3i-1))
%         with radius x(3i)
%   n   The total number of disks
% OUTPUT:
%   c   A vector of size n*(n-1) + 2*n
%   ceq An empty vector []

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)

%The total number of constraints will be n*(n-1) + 4*n + 1

num_constraints = n*(n-1) + 2*n;
c = zeros(num_constraints, 1);
ceq = [];
%non overlapping circles
[circle_constraints, num] = variable_radius_constraints(x, n);
c(1:num) = circle_constraints;
insertion_index = num + 1;
```

```
%One circle constraint
for i = 1:n
    k = 3*i;
    c(insertion_index) = (0.25 + x(k)).^2 - (x(k-2)-2/3).^2 - (x(k-1) -
    ↪ 0.25).^2;
    insertion_index = insertion_index + 1;
end

%Other circle constraint
for i = 1:n
    k = 3*i;
    c(insertion_index) = (0.25 + x(k)).^2 - (x(k-2)-1).^2 - (x(k-1) -
    ↪ 1).^2;
    insertion_index = insertion_index + 1;
end

end
```

## A.17 target_p3.m

Target function for problem 3.

```
function [f,grad_f] = target_p3(x)
% Target function for problem 3
%
% INPUT:
%   x Column vector of length 3n. Each disk has center= (x(3i-2), x(3i-1))
%       with radius x(3i)
% OUTPUT:
%   f      function value at x
%   grad_f gradient of f at x
n = numel(x)/3;
I = (1:n)*3;
f = -sum(x(I).^2);
if nargout > 1
    grad_f = zeros(3*n, 1);
    grad_f(I) = -2*x(I);
end
end
```

## A.18 problem_p3.m

Definition of the problem 3.

```
function [lb,ub, A, b, nlc, f] = problem_p3(n)
% problem_p2 Creates the proper variables for the optimisation problem as
%  defined in problem 3 in
%  https://www.canvas.umu.se/courses/10377/assignments/117252
%
% In the model each disk is centered at (x(3i-2), x(3i-1)) with radis x(3i)
```

```
% INPUT:
%   n   number of disks
% OUTPUT:
%   lb  A (3n) x 1 column vector with lower bounds x
%   ub  A (2n + 1) x 1 column vector with upper bounds x
%   A   A part of the condition Ax<=b. A 2*n x (2n + 1) matrix
%   b   A part of the condition Ax<=b. A 2*n x 1 column vector of 0
%   nlc A the non linear conditions.
%   f   Target function for optimisation


%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)

%All values are at least 0
lb = zeros(3*n, 1);

%All coordinates are contained <= 1
ub = lb + 1;

%All radii must be <= 1 to be constrained to unit square
I = (1:n)*3;
ub(I)=0.5;



A = zeros(5*n, 3*n);
b = zeros(5*n, 1);
for j = 1:n
    A(j, 3*j-2) = -1;
    A(j, 3*j) = 1;
end
for j = 1:n
    i = j + n;
    A(i, 3*j-1) = -1;
    A(i, 3*j) = 1;
end
for j = 1:n
    i = j + 2*n;
    A(i, 3*j-2) = 1;
    A(i, 3*j) = 1;
    b(i) = 1;
end
for j = 1:n
    i = j + 3*n;
    A(i, 3*j-1) = 1;
    A(i, 3*j) = 1;
    b(i) = 1;
end
sqrt2 = sqrt(2);
for j = 1:n
    i = j + 4*n;
    A(i, 3*j-2) = -1/sqrt2;
    A(i, 3*j-1) = 1/sqrt2;
    A(i, 3*j) = 1;
    b(i) = 2 / (3*sqrt2);
end
nlc = @(x)constraints_p3_nlc(x, n);
f = @target_p3;
```

```
end
```

## A.19 test_p3.m

Code to study specific $n$ for problem 3 as well as the difference that `refine_solution` in appendix A.25 does.

```
%

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
close all
clear variables
clear global

rng default
%We are trying to find N(0.8)
% We start by deciding on an upper bound

%Using interior point leads to terrible solutions
opts = optimoptions('fmincon','Algorithm','active-set', 'SpecifyObjectiveGradient',true);
ms =  MultiStart('UseParallel',true,'Display','off');
gs = GlobalSearch;
area = 1 - 1/18 - 5*pi/64;
d=@(f, n)-pi*f/area;


tic
%40 too low
%
n = 41
[lb,ub, A, b, nlc, target]=problem_p3(n);
x0 = rand(size(lb)) .* (ub - lb) + lb;
problem = createOptimProblem('fmincon','objective', target, 'x0', ...
    x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
    'options', opts);
[x, f] = run(ms, problem, 10*n);
plot_packing_p3(x, n);

[x, f] = refine_solution(x, n, problem, n, ms, 20);
figure()
plot_packing_p3(x, n);
fprintf("Density of     refined n=%d packing is d=%1.6f\n", n, d(f,n));
toc

I=(1:n)*3;
remove = (d(f, n) - 0.9)/(min(x(I)).^2 * pi / area);
if (remove >= 1)
    fprintf("Can safely remove smallest circle %3.2f times", remove);
end
toc
%plot_packing_3quarter_circle(x, n);
name=sprintf("problem3_n_%d_d_%1.6f.pdf", n, d(f, n));
exportgraphics(gcf, name, "ContentType", "vector")
%fprintf("Density of     refined n=%d packing is d=%1.6f", n, d(f,n));
```

```
%ga seems unreliable at best? Might work better for other problems or with
%better options?
%x = ga(target,2*n+1,A,b,[],[],lb,ub,nlc);

```

## A.20 variable_radius_constraints.m

Code for calculating function $g_{i,j}$ in equation 7.

```
function [c,num] = variable_radius_constraints(x,n)
% variable_radius_constraints Calculates constraints for non overlapping
%  disks with variable radii
%
% INPUT:
%   x   Column vector of length 3n. Each disk has center=(x(3i-2), x(3i-1))
%         with radius x(3i)
%   n   The total number of disks
% Output:
%   c   A vector of size n*(n-1). Contains values that are each 0 or
%         negative if the disks defined by x are non overlapping
%   num n*(n-1) the size of c


%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
%   2023-10-27 initial implementation

num = n*(n-1);
I = 1:n;
J = 1:n;
[I,J]=meshgrid(I,J);
I = reshape(I, n*n, 1);
J = reshape(J, n*n, 1);
choice = I~=J;
I=I(choice);
J=J(choice);
I = I*3;
J = J*3;
DeltaX1 = x(I - 2) - x(J - 2);
DeltaX2 = x(I - 1) - x(J - 1);
rSum = x(I) + x(J);
c = rSum.^2 - DeltaX1.^2 - DeltaX2.^2;
end
```

## A.21 constraints_p4_nlc.m

Code for calculating non linear constraints in equation 11.

```
function [c,ceq] = constraints_p4_nlc(x, n)
% constraints_p3 Calculates non linear constrain functions for problem 4 as
%  per https://www.canvas.umu.se/courses/10377/assignments/117252.
%
% INPUT:
```

```
%   x   Column vector of length 3n. Each disk has center=(x(3i-2), x(3i-1))
%           with radius x(3i)
%   n   The total number of disks
% OUTPUT:
%   c   A vector of size n*(n-1) + 2*n
%   ceq An empty vector []

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)



num_constraints = n*(n-1) + 2*n;
c = zeros(num_constraints, 1);
ceq = [];
%non overlapping circles
[circle_constraints, num] = variable_radius_constraints(x, n);
c(1:num) = circle_constraints;
insertion_index = num + 1;


%Within circle constraint
for i = 1:n
    k = 3*i;
    c(insertion_index) = x(k-2).^2 + x(k-1).^2 - (1-x(k))^2;
    insertion_index = insertion_index + 1;
end

%Other circle constraint
for i = 1:n
    k = 3*i;
    if x(k - 2) <= 0
        if x(k - 1) >= 0
            result = x(k).^2 - x(k - 2).^2 - x(k - 1).^2;
        else
            result = x(k).^2 - x(k - 2).^2;
        end
    else
        if x(k - 1) >= 0
            result = x(k).^2 - x(k - 1).^2;
        else
            result = x(k).^2 + x(k - 2).^2 + x(k - 1).^2;
        end
    end
    c(insertion_index)=result;
    insertion_index = insertion_index + 1;
end
end
```

## A.22 model_p4.m

Code for defining constraints in equation 11.

```
function [lb,ub, A, b, nlc, f] = model_p4(n)
% model_p4 Creates the proper variables for the optimisation problem as
%  defined in problem 4 in
```

```
%  https://www.canvas.umu.se/courses/10377/assignments/117252
%
% In the model each disk is centered at (x(3i-2), x(3i-1)) with radis x(3i)
% INPUT:
%    n    number of disks
% OUTPUT:
%    lb   A (3n) x 1 column vector with lower bounds x
%    ub   A (3n) x 1 column vector with upper bounds x
%    A    An empty matrix []
%    b    An empty vector []
%    nlc A the non linear conditions.
%    f    Target function for optimisation


%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)

%All values are at least 0
lb = -ones(3*n, 1);

%All coordinates are contained <= 1
ub = ones(3*n, 1);

I = (1:n)*3;
%radius always positive
lb(I)=0;



A = [];
b = [];
nlc = @(x)constraints_p4_nlc(x, n);
f = @(x)-sum(x(I).^2);
end
```

## A.23 plot_circle_variable_radii.m

Code for plotting circles with variable radii.

```
function plot_circles_variable_radii(x,n)
% plot_circles_variable_radii Plots All the circles contained in x to the
%  current plot
%
% INPUT:
%    x    Column vector of length 3n. Each disk has center=(x(3i-2), x(3i-1))
%           with radius x(3i)
%    n    The total number of disks

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
%    2023-10-27 initial implementation

hold on
angles = linspace(0, 2*pi, 100)';
x_p = cos(angles);
y_p = sin(angles);
%for some reason on my machine on my verion of matlab using green gives
```

```
%rainbows around the shape
c=[0.8 0.7 0.8];
%scale to radius

for i = 1:n
    r = x(3*i);
    x_plot = x_p * r + x(3*i - 2);
    y_plot = y_p * r + x(3*i - 1);
    fill(x_plot, y_plot, c);
end
hold off
end
```

## A.24 plot_packing_3quarter_circle.m

Code to plot a packing for problem 4.

```
function plot_packing_3quarter_circle(x,n)
% plot_packing_3quarter_circle Plots a packing of the three quarter circle
%
% INPUT:
%   x           Column vector of length 3n. Each disk has center=
%                 (x(3i-2), x(3i-1)) with radius x(3i)
%   n           The total number of disks

%Plot packed disks
plot_circles_variable_radii(x, n);
hold on

%Plot outer circle
angles = linspace(0, 2*pi, 100);
xp = cos(angles);
yp = sin(angles);
plot(xp, yp, 'k');

%Fill quarter disk
angles = linspace(-pi/2, 0, 25);
xp = cos(angles);
yp = sin(angles);
xp = [xp, 1, 0,  0];
yp = [yp, 0, 0, -1];
c = [0.3 0.3 0.3];
fill(xp, yp, c);
axis off
end
```

## A.25 refine_solution.m

Code to improve a solution to equation 11 by moving the circle with the smallest radius.

```
function [x,f] = refine_solution(x, n, problem, iterations, ms, tries)
% refine_solution Refines a solution x to problem by randomly moving the
```

```
%   smallest disk. Each x and y coordinate must have a lb and ub
%
% INPUT:
%   x          Column vector of length 3n. Each disk has center=
%                (x(3i-2), x(3i-1)) with radius x(3i)
%   n          The total number of disks
%   problem    An OptimProblem to solve
%   iterations The number of iterations of refinement done. Each iteration
%                checks tries new starting points. All coordinates must
%                have lb and ub
%   ms         MultiStart object to run
%   tries      Generates tries points per run. Defaults to 10
% OUTPUT:
%   x The best solution found to problem
%   f The function value at x, problem.objective(x)
x_best = x;
f_best = problem.objective(x);
if nargin < 6
    tries = 10;
end
refined_solutions = tries;
for dummy = 1:iterations
    [~, small_i] = min(x((1:n)*3));
    I = [3*small_i-2, 3*small_i-1];
    start_pt_matrix = zeros(refined_solutions, 3*n) + x';
    lb = reshape(problem.lb(I), 1, 2);
    ub = reshape(problem.ub(I), 1, 2);
    start_pt_matrix(:, I) = rand(refined_solutions, 2).*(ub - lb) + lb;
    start_pts = CustomStartPointSet(start_pt_matrix);
    [x, f] = run(ms, problem, start_pts);
    if f < f_best
        f_best = f;
        x_best = x;
    end
end
x = x_best;
f = f_best;
end
```

## A.26 test_p4.m

Code to study specific $n$ for problem 4 as well as the difference that `refine_solution` in appendix A.25 does.

```
%

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
close all
clear variables
clear global

rng default
%We are trying to find N(0.8)
% We start by deciding on an upper bound

```

```
%Using interior point leads to terrible solutions
opts = optimoptions('fmincon','Algorithm','sqp');
ms =  MultiStart('UseParallel',true,'Display','off');
gs = GlobalSearch;

d=@(f, n)-pi*f/(3*pi/4);



n = 12
[lb,ub, A, b, nlc, target]=model_p4(n);
x0 = rand(size(lb)) .* (ub - lb) + lb;
problem = createOptimProblem('fmincon','objective', target, 'x0', ...
    x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
    'options', opts);
[x, f] = run(ms, problem, 1000);
plot_packing_3quarter_circle(x, n);
exportgraphics(gcf, "problem4_n12_non_refined.pdf", "ContentType",
↪  "vector")
fprintf("Density of non refined n=%d packing is d=%1.6f", n, d(f,n));

figure
[x, f] = refine_solution(x, n, problem, 100, ms);
plot_packing_3quarter_circle(x, n);
exportgraphics(gcf, "problem4_n12_refined.pdf", "ContentType", "vector")
fprintf("Density of     refined n=%d packing is d=%1.6f", n, d(f,n));
%ga seems unreliable at best? Might work better for other problems or with
%better options?
%x = ga(target,2*n+1,A,b,[],[],lb,ub,nlc);
```

## A.27 packing_3quarter_circle.m

Code to calculate a good packing for a three quarter circle.

```
function [d,x] = packing_3quarter_circle(n, samples, display)
% packing_3_quarter_circle Calculates a packing of disks for the three
%  quarter unit circle. Uses MultiSearch to find a starting point and then
%  uses refine_solution to improve it. Half of the samples will be spent in
%  refine_solution. Becomes deterministic be resetting the rng.
%
% INPUT:
%   n       Number of disks used in packing
%   samples Maximum number of points searched. A minimum of 11 samples is
%             always run
%   display Input to the "display" MultiSearch option.
% OUTPUT:
%   d Density of best found packing x
%   x Best found packing. A 3n x 1 column vector. Disk i is centered at
%         [x(3*i-2), x(3*i-1)] with radius x(3*i)

rng default

%Using interior point leads to terrible solutions
opts = optimoptions('fmincon','Algorithm','sqp');
```

```

%Paralelle computation is prefered
ms =  MultiStart('UseParallel',true,'Display',display);

%Create problem
[lb,ub, A, b, nlc, target]=model_p4(n);
x0 = rand(size(lb)) .* (ub - lb) + lb;
problem = createOptimProblem('fmincon','objective', target, 'x0', ...
    x0, 'Aineq', A, 'bineq', b, 'lb', lb, 'ub', ub, 'nonlcon', nlc, ...
    'options', opts);

iters = max(floor(samples / 20), 1);
runs = max(samples - iters * 10, 1);
[x, ~] = run(ms, problem, runs);

[x, f] = refine_solution(x, n, problem, iters, ms);
d = -pi*f/(3*pi/4);
end
```

## A.28 binary_search.m

Performs a binary search to find smallest solution to $f(x) >= k$ for increasing function $f : \{a, a+1, a+2, ..., b\} \to \mathbb{R}$ when $a, b \in \mathbb{Z}$.

```
function [n,f_n] = binary_search(a, b, minimum, f)
% binary_serch Performs binary search to find solve the lowest solution to
%  the integer equation f(x)>= minimum. Unknown behaviour if
%  f(b)< minimum. Assumes that f is increasing on [a, b]
%
% INPUT:
%   a,b      Bracket to search in [a, b]. Must have f(b)>=f(a),
%            f(b)>=minimum
%   minimum Value to find lowest x such that f(x)>= minimum
%   f        Target function, increasing on [a, b]
% OUTPUT:
%   n    The best n found
%   f_n f(n)
f_a = f(a);
f_b = f(b);
if f_a >= minimum
    b = a;
    f_b = a;
end
if f_b < minimum
    a = b;
end

while b>a+1
    c = floor((a+b)/2);
    f_c = f(c);
    if f_c >= minimum
        b=c;
        f_b=f_c;
    else
        a=c;
```

```
    end
end
n = b;
f_n = f_b;
end
```

## A.29 solve_p4.m

Solves problem 4.

```
%

%Programming by Harald Bjurulf (haraldbjurulf@hotmail.com)
close all
clear variables
clear global

rng default

display="off";
f = @(n)packing_3quarter_circle(n, 2000, display);
[N_085, ~] = binary_search(1, 15, 0.85, f);
[N_088, ~] = binary_search(1, 15, 0.88, f);

[d_085, x_085] = f(N_085);
plot_packing_3quarter_circle(x_085, N_085);
fprintf("N085=%d with density d=%1.6f\n", N_085, d_085);
exportgraphics(gcf, "problem4_N085.pdf", "ContentType", "vector")

figure
[d_088, x_088] = f(N_088);
plot_packing_3quarter_circle(x_088, N_088);
fprintf("N085=%d with density d=%1.6f\n", N_088, d_088);
exportgraphics(gcf, "problem4_N085.pdf", "ContentType", "vector")
```