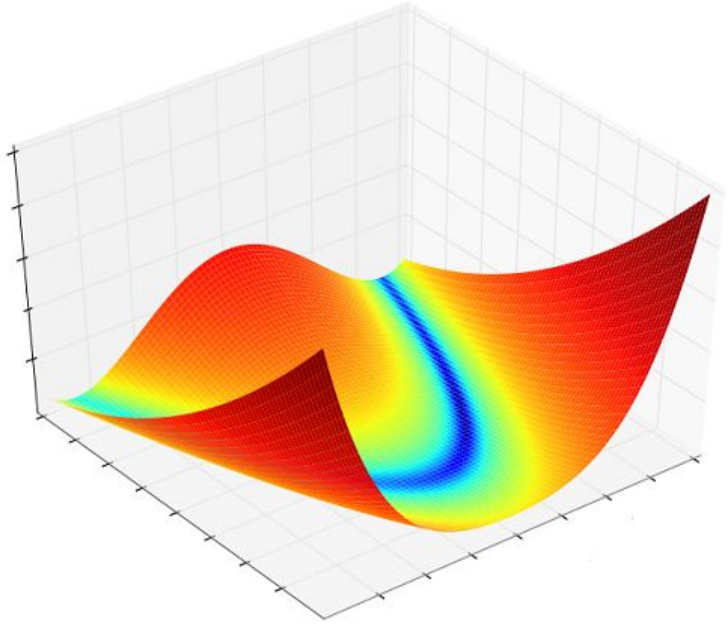
# EE553
# Optimization
# Take Home Exam Report



JeanPiere Demir
1936632

# Contents

## Problem Definition

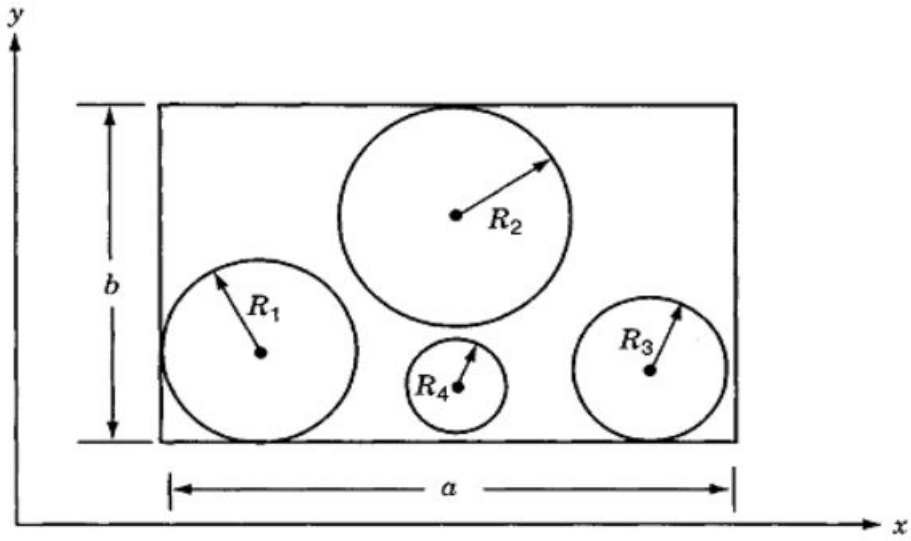



Figure 1: Locations of circular disks in a rectangular plate

In this work, it is required to place four circular disks shaped factory machines in a smallest rectangular shaped factory to minimize the scrap environments. The radius of the circular disks is given. Firstly, the required task is formulated as an optimization problem, and objective function constructed. The objective function can be defined as:

$$J = a \cdot b - \pi \cdot R_1^2 + R_2^2 + R_3^2 + R_4^2$$

This objective function tries to minimize the scrap environment inside the factory. After transforming the physical problem to a mathematical problem, constraints have to be added to the problem to model physical environment effects and satisfy the requirements.

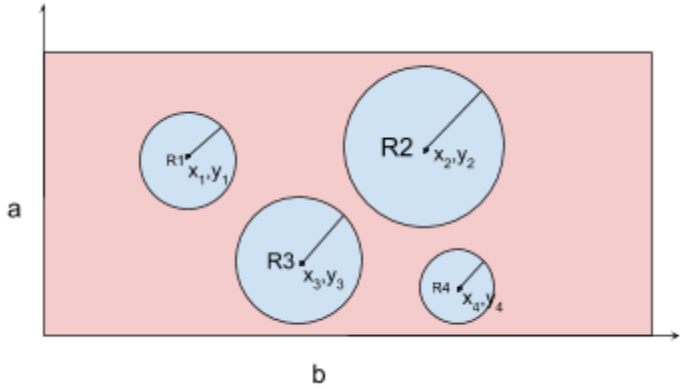



Figure 2: Reconstruction of the figure 1 to highlight design variables

Design variables of the optimization problem are ordered as:

| | | | | |
|---|---|---|---|---|
| x(1) → a | x(3) → x1 | x(5) → x2 | x(7) → x3 | x(9) → x4 |
| x(2) → b | x(4) → y1 | x(6) → y2 | x(8) → y3 | x(10) → y4 |

The constraints are limiting the algorithm according to physical properties. The given constraints are preventing collision between circular disks shaped factory machines.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \geq R_1 + R_2$$
$$\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} \geq R_1 + R_3$$
$$\sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2} \geq R_1 + R_4$$
$$\sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2} \geq R_2 + R_3$$
$$\sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2} \geq R_2 + R_4$$
$$\sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2} \geq R_3 + R_4$$

Moreover, one set more constraints will be defined to place the circular disks shaped factory machines inside the rectangular shaped factory. The last set of constraints are given as:

$$b \geq x_i + R_i$$
$$a \geq y_i + R_i$$
$$x_i \geq R_i$$
$$y_i \geq R_i$$
$$i = 1, 2, 3, 4$$

As one can see that the defined problem is constrained optimization problem. However, one has to convert constrained optimization problem to unconstrained optimization problem to apply unconstrained optimization algorithm easily. Penalty function method is chosen to convert constrained optimization problem to unconstrained optimization problem. Penalty function method introduces an artificial penalty for violating the constraint and embed them to the objective function. The penalty added as quadratic loss function.

The reformed version of the design problem can be seen as:

$$
\begin{aligned}
J = {} & a \cdot b - \pi \cdot R_1^2 + R_2^2 + R_3^2 + R_4^2 \\
& + f_1 * (R_1 + R_2 - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2})^2 \\
& + f_2 * (R_1 + R_3 - \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2})^2 \\
& + f_3 * (R_1 + R_4 - \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2})^2 \\
& + f_4 * (R_2 + R_3 - \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2})^2 \\
& + f_5 * (R_2 + R_4 - \sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2})^2 \\
& + f_6 * (R_3 + R_4 - \sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2})^2 \\
& + f_{6+i} * (x_i + R_i - b) \quad (i = 1, 2, 3, 4) \\
& + f_{10+i} * (y_i + R_i - a) \quad (i = 1, 2, 3, 4) \\
& + f_{14+i} * (R_i - x_i) \quad (i = 1, 2, 3, 4) \\
& + f_{18+i} * (R_i - y_i) \quad (i = 1, 2, 3, 4)
\end{aligned}
$$

As one can see all constraints added as quadratic loss function and $f_j$ is a flag which indicates whether the constraint violated or not and its value is 1 or 0 respectively [1].

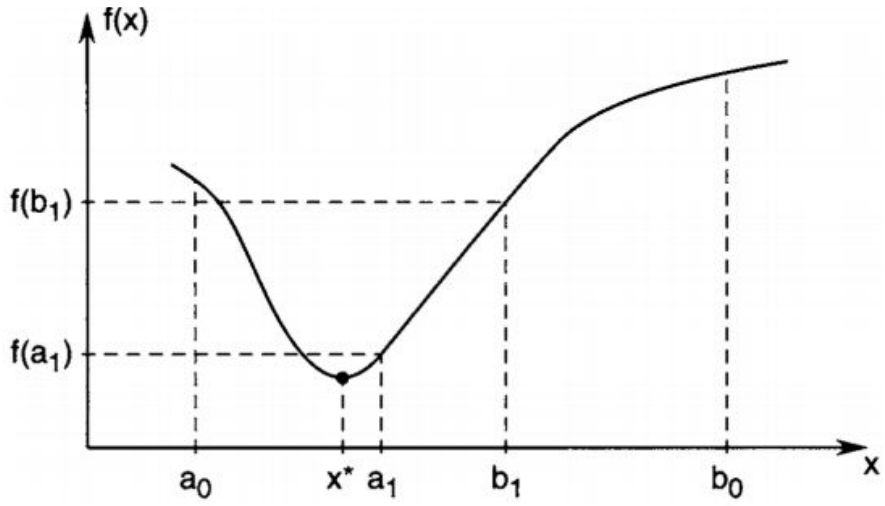
## 1D Search

The 1D Search methods allow to determine the minimizer of a function in a closed interval for 1D parameter. In this work, considered 1D search problem is minimizing and finding most appropriate learning rate for iterative processes. Most of the optimization algorithms have given generic structure:

$$\alpha_{i+1} = \alpha_i - \eta * \nabla J$$

where η is learning rate, $\nabla J$ is the gradient of the objective function. 1D search tries to optimize learning rate and objective function is given generic structure. There is different search methods to optimize learning rate. They are given is Golden Section Method, Fibonacci Search, Bisection, Dichotomous Search, etc. . I chose Golden Section Method as a 1D search method because it narrows search interval by less than half for each iteration and its computational efficiency is better than other search algorithms.

### Golden Section Search




- If $f(a_1) < f(b_1)$ then $x^* \in [a_0, b_1]$;
- If $f(a_1) > f(b_1)$ then $x^* \in [a_1, b_0]$;

The Golden section search method chose the boundaries according to above relations and it terminates when the interval is less then given tolerance value [6].

## Optimization Algorithm

### Gradient Calculation

The gradient can be calculated in two different way which are numerically and analytically.

- If the user selected to calculate the gradient numerically, following equation is applied:

$$\vec{\nabla} J = \frac{J(x_i + \vec{z}) - J(x_i)}{|\vec{z}|}$$

- If the user selected to calculate gradient analytically, derivative of the objective function is taken according design variables.

### Steepest Descent Algorithm

The generic structure was given before and can be seen as:

$$\alpha_{i+1} = \alpha_i - \eta * \nabla J$$

where ∇J is used directly according to user input, the learning rate is optimized by using Golden Section Search method. The accuracy of this algorithm is the worst one among the other algorithms which can be seen in the Results section [2].

### Fletcher-Reeves (Conjugate Gradient) Algorithm

The generic structure of the algorithm is given as:

$$\alpha_{i+1} = \alpha_i - \eta \cdot p_i$$

The learning rate is optimized by using Golden Section Search method. The equation to calculate $p_i$ is given as:

$$p_{i+1} = |\nabla J_{i+1}| + \beta_{i+1} \cdot p_i$$

$$\beta_{i+1} = \frac{|\nabla J_{i+1}|}{|\nabla J_i|}$$

One can see that $p_i$ is calculated by using current gradient and previous gradient. So, the accuracy of this method is better then Steepest Descent Method [3].

### Newton Algorithm

The generic structure of the algorithm is given as:

$$\alpha_{i+1} = \alpha_i - \cdot H_i^{-1} \cdot \nabla J$$

There is not any learning rate so 1D search is not used. One can see that the direction algorithm directly depends on hessian and gradient. The hessian is calculated by taking derivatives of the gradient vector according to design variables [5].

### Davidon-Fletcher-Powell (Newton Rank 2 ) Algorithm

The generic structure of the algorithm is given as:

$$\alpha_{i+1} = \alpha_i - \eta \cdot \tilde{H}_i^{-1} \cdot \nabla J$$

The learning rate is optimized by using Golden Section Search method. The equation to calculate approximate inverse hessian is:

$$\tilde{H}_{i+1}^{-1} = \tilde{H}_i^{-1} + \frac{p_i' * p_i}{|p_i|} - \frac{\tilde{H}_i^{-1} * (q_i * q_i') * \tilde{H}_i^{-1}}{q_i' * \tilde{H}_i^{-1} * q_i}$$

$$p_{i+1} = x_{i+1} - x_i$$

$$q_{i+1} = \nabla J_{i+1} - \nabla J_i$$

As one can see that this algorithm does not just depend on gradient based direction vector. The direction vector is calculated by using approximate inverse hessian and gradient. This algorithm is the best algorithm among the other optimization algorithms [4].

## Results

MatLab GUI developed to test algorithms easier. The usage of GUI is explained in the Appendix A, please follow the steps which are described in Appendix A. The generic structure of the GUI can be seen as:

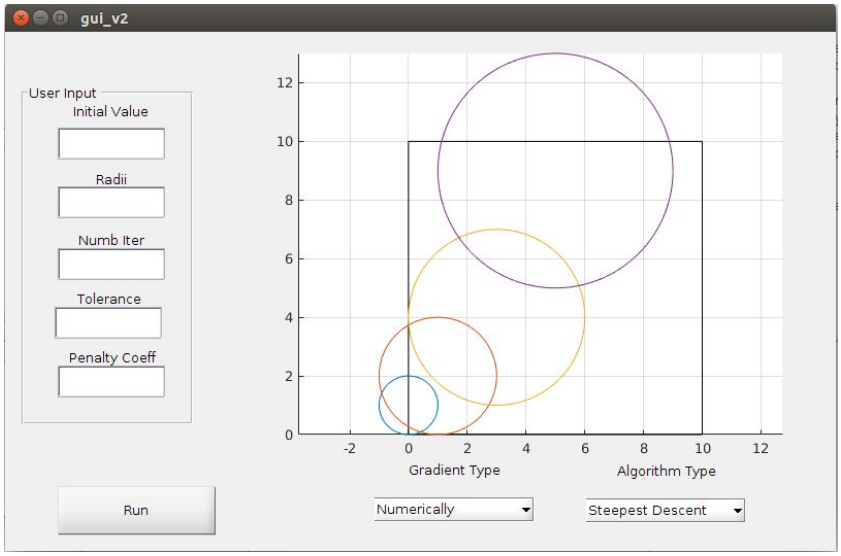



Figure 3: The General Layout of the GUI

## Steepest Descent Algorithm

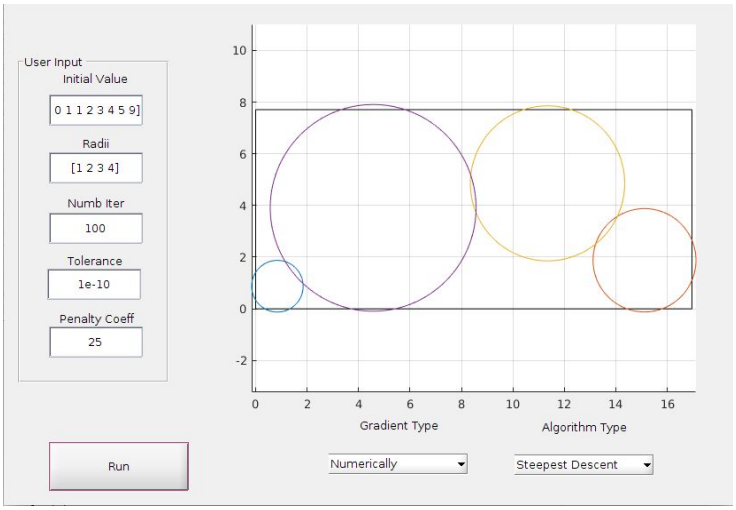



Figure 4: Steepest Descent Algorithm with numeric gradient when given parameters are applied.

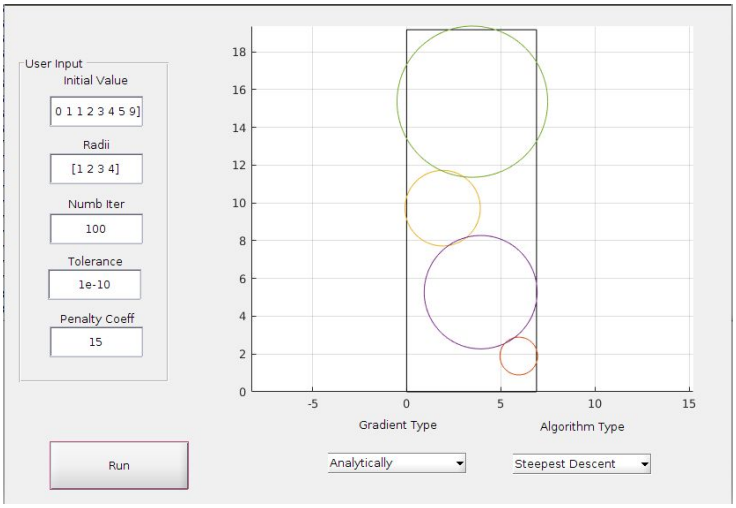



Figure 5: Steepest Descent Algorithm with analytic gradient when given parameters are applied.

The comparison has been done according to given figures and one can see that for the steepest descent both gradient methods are approximately same in the manner of results.

## Fletcher-Reeves Algorithm

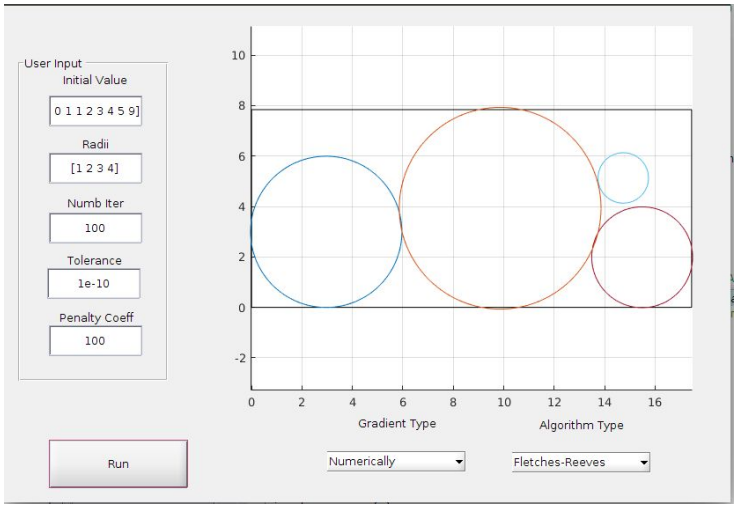



Figure 6: Fletcher-Reeves Algorithm with numeric gradient when given parameters are applied

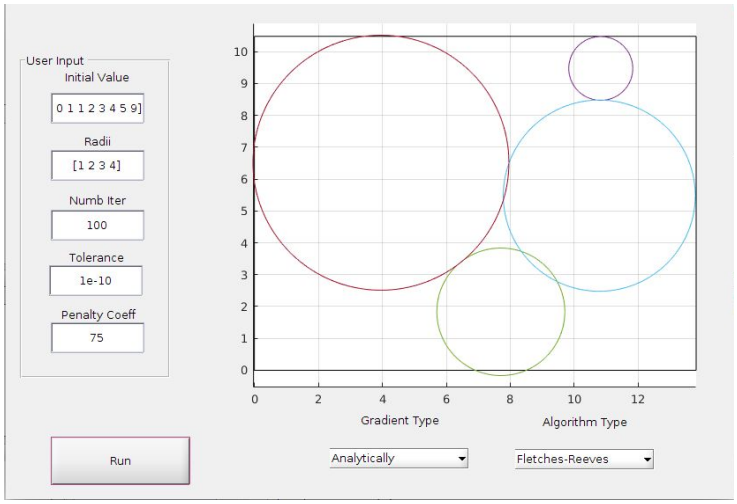



Figure 7: Fletcher-Reeves Algorithm with analytic gradient when given parameters are applied

One can see that, analytical gradient minimize the problem better than numerical gradient. Moreover, the seen result is better than Steepest Descent result because in this algorithm direction vector does not depend on only current gradient on the contrary it depends on current and old gradient.

## Davidon-Fletcher-Powell (Newton Rank 2 ) Algorithm

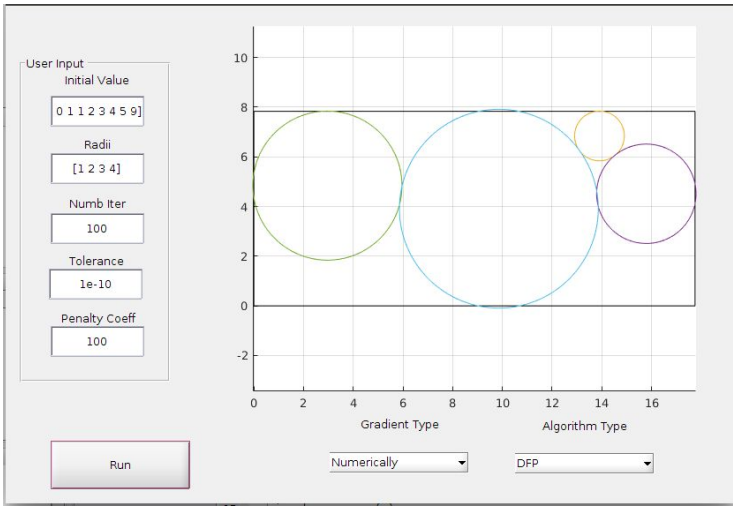



Figure 8: DFP Algorithm with numeric gradient when given parameters are applied

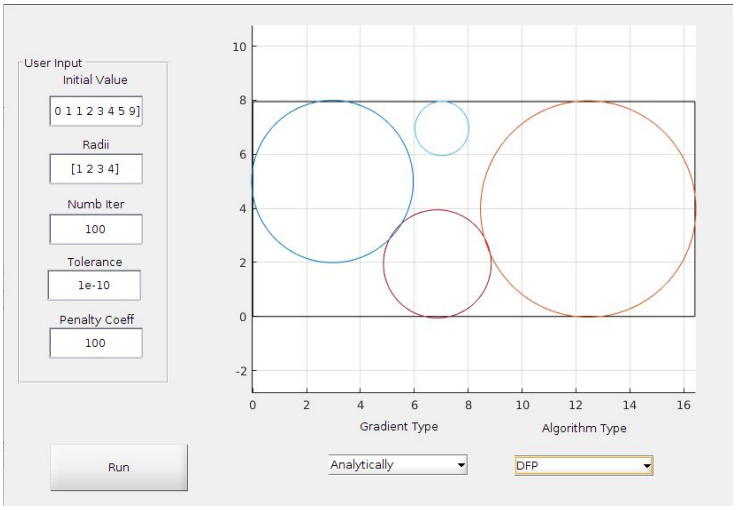



Figure 9: DFP Algorithm with analytic gradient when given parameters are applied

If one will compare this method with Fletcher-Reeves, one can see that in this algorithm the effect of penalties on the cost function considered more. So, the resultant scrap area is less and penalties do not violated.

## Newton Algorithm

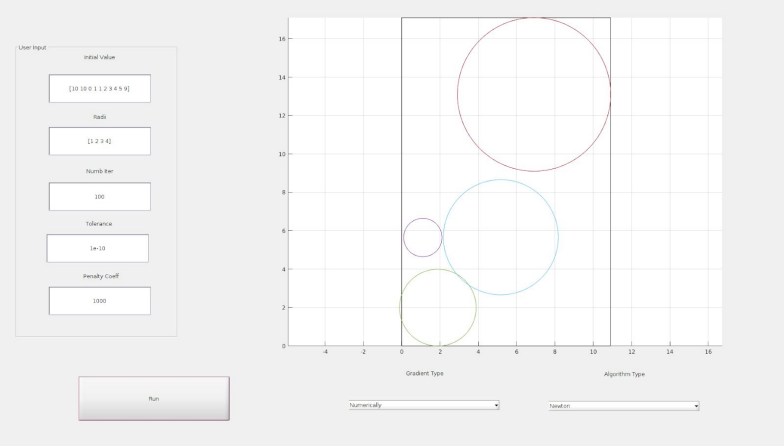



Figure 10:Newton Algorithm with numerical gradient when given parameters are applied

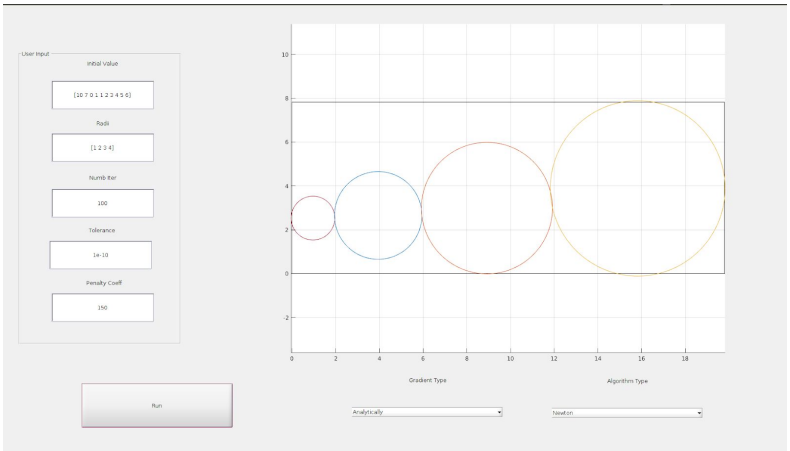



Figure 11:Newton Algorithm with analytical gradient when given parameters are applied

Theoretically, in the beginning I expected best result from this algorithm due to pure hessian calculation. However, the hessian has some singularities due to penalties, so the result is not as expected. The best results is obtained via DFP algorithm.

## Conclusion

In this work, I tried to optimize a constrained optimization problem via different unconstrained optimization algorithms such as Gradient Descent, Fletcher-Reeves, Newton Rank 2 correction (Davidon-Fletcher-Powell) and Newton algorithm. Firstly, I transformed constrained optimization problem to unconstrained optimization algorithm by using Penalty function method. Later on, different algorithms are tested to see result. To test different algorithms easily and see the visual results, a gui was developed. The usage manual of the GUI can be found in Appendix A.

As a result, if the direction vector richens with old and current gradient/hessian information the optimization algorithm converges better and the result is more appropriate.

## References

**[1]** Chong, E. K. P., & Żak, S. H. (2013). *An introduction to optimization*.

**[2]** Jonathan Barzilai and Jonathan M. Borwein, "Two-point step size gradient methods", in: IMA journal of numerical analysis, 8.1 (1998), pp. 141-148

**[3]** Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems". *Journal of Research of the National Bureau of Standards*.

**[4]** Davidon, W. C. (1991), "Variable metric method for minimization", SIAM Journal on Optimization

**[5]** Süli, Endre; Mayers, David (2003). *An Introduction to Numerical Analysis*. Cambridge University Press

**[6]** Kiefer, J. (1953), "Sequential minimax search for a maximum", Proceedings of the American Mathematical Society,

# Appendix A (User Manual)

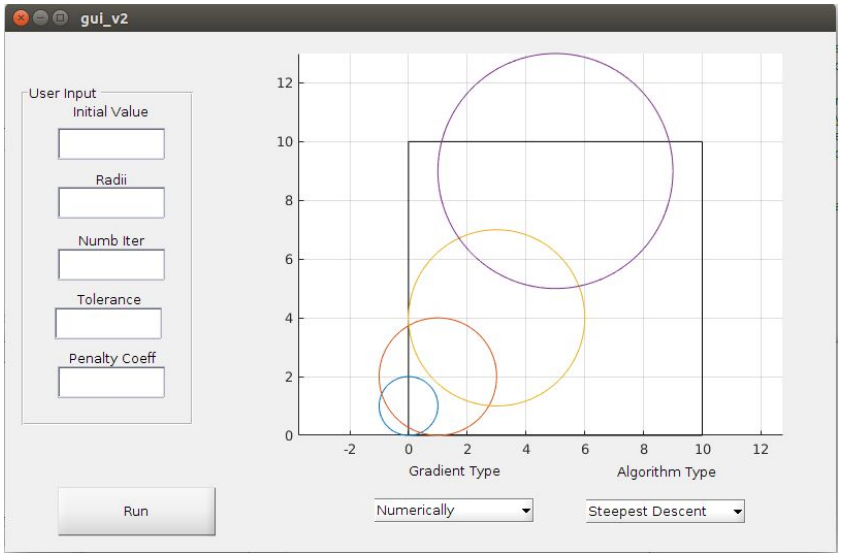



Figure 12: General Structure of the GUI

The general layout of the GUI can be seen in Figure 12. As one can see, user input part is divided from the other parts to simplify and richen the user interaction. Default values of inputs are:

**Initial Value: [10 10 0 1 1 2 3 4 5 9]**
**Radii: [1 2 3 4]**
**Number of Iteration: 1000**
**Tolerance: 1e-10**
**Penalty Coefficient: 10**
**Gradient Calculation: Analytically**
**Algorithm: Gradient Descent**

## User Input Selection

When user specify the inputs has to obey some specific rules. Entering the input to the input box is enough for the GUI to proceed. The rules are :

- When initial value is entered the input has to have 10 elements between two square brackets
    - E.g. [1 2 3 4 5 6 7 8 9 10 ]
- Also, the structure of the initial value is given as:
    - [r_x r_y c1_x c_1_y c_2_x c_2_y c_3_x c_3_y c_4_x c_4_y] where
        - r_x : Length of the horizontal side of the rectangle
        - r_y : Length of the vertical side of the rectangle
        - c_i_x: X Position of center of $i^{th}$ circle
        - c_i_y: Y Position of center of $i^{th}$ circle (i = 1,2,3,4)

- When radii is entered the input has to have 4 elements between two square brackets
    - E.g. [1 2 3 4]
- Also the structure of the initial value is given as:
    - [R_1 R_2 R_3 R_4]
        - R_i : Radius of $i^{th}$ circular disk
- When Number of iteration, tolerance and penalty coefficient is entered values like the example is enough for the GUI.
- If user want to have proper investigation about the GUI and obtain better results, s/he can increase the penalty coefficient for each run
- Optimization Algorithms and Gradient Calculation methods can be chosen via pop-up menu.
- Please restart the GUI if the any error is faced.