Task 2 – Gradient Methods (10P)

a) Which condition must be fulfilled by a search direction $v^{(k)}$? Also formulate this condition mathematically and proof that this is always fulfilled by the steepest descent method! (4P)

b) Formulate the subproblem that must be solved in each iteration and give two methods which can be used for that. Can the simplex method according to Nelder-Mead be used for this? Give a short explanation! (6P)

a) The descent condition must be fulfilled by a search direction $v^{(K)}$,

$$(v^{(K)})^T F_x^{(K)} < 0$$

$\rightsquigarrow$ Steepest descent method: $v^{(K)} = -F_x^{(K)}$

$$(v^{(K)})^T F_x^{(K)} = (-F_x^{(K)})^T F_x^{(K)} = -\|F_x^{(K)}\| < 0.$$

b) The subproblem that must be solved at each iteration is given by

$$\min_{\alpha > 0} F(x^{(K)} + \alpha^{(K)} v^{(K)})$$

where $x^{(K+1)} = x^{(K)} + \alpha^{(K)} v^{(K)}$

The subproblem consists of an unconstrained one dimensional optimization problem to determine step size $\alpha^{(k)}$ at each iteration.

To solve this problem we could use search methods such as the golden ratio search, with multiple evaluations of the function F. Furthermore, we could use the line search method with the wolfe conditions to provide sets of allowed step sizes.

In principle, the Nelder-Mead simplex method could be used for this. But since this method is designed for multidimensional problems it is considered slow and does not guarantee the wolfe conditions. So, it is not recommended for this kind of problems.

Task 3 – Newton Methods (10P)

a) Give two equivalent interpretations of the basic idea of the Newton-Raphson method, as well as its iteration rule! (4P)

b) List three major difficulties that can occur when using the Newton-Raphson method and explain how the quasi-Newton method addresses these difficulties! (3P)

c) Compare the quasi-Newton method with the steepest descent method using the criteria convergence speed (number of iterations), computational effort per iteration and robustness! (3 P)

a) The main idea of Newton-Raphson's method is to apply Newton's method to the optimality condition $F_x(x^*) = 0$.

(i) 1st solution

Linear approximation of $F_x(x^{(k)})$ at the current point $x^{(k)}$.

$$\tilde{F}_x(x^{(k)} + \Delta x^{(k)}) = F_x(x^{(k)}) + F_{xx}(x^{(k)}) \Delta x^{(k)}$$

Finding zero of $\tilde{F}_x$ instead of F_x

$$\tilde{F}_x = 0 = F_x(x^{(k)}) + F_{xx}(x^{(k)}) \Delta x^{(k)}$$

$$\Rightarrow \Delta x^{(k)} = -\left[F_{xx}(x^{(k)})\right]^{-1} F_x(x^{(k)})$$

(ii) 2^{nd} solution

Approximation of the objective function as a quadratic function.

$$\tilde{F}(x^{(k)} + \Delta x^{(k)}) = F(x^{(k)}) + (\Delta x^{(k)})^T \cdot F_x(x^{(k)}) + \frac{1}{2} (\Delta x^{(k)})^T \cdot F_{xx}(x^{(k)}) \cdot (\Delta x^{(k)})$$

Then the optimality condition of approximation becomes:

$$\tilde{F}_{\Delta x}(\Delta x^{(k)}) = F_x(x^{(k)}) + F_{xx}(x^{(k)}) \cdot \Delta x^{(k)} = 0$$

b) Difficulties of Newton-Raphson

(i) Calculation of the Hessian matrix is very expensive for high-dimensional problems.

(ii) Inverting the Hessian matrix is also very expensive.

(iii) If the Hessian matrix is not positive definite, the Newton direction might not be a descent direction leading to divergence.

Solution is Quasi - Newton method.

(i) Uses gradient information to build an approximation of the Hessian matrix and its inverse.

(ii) Updates Hessian matrix with rank -1 or rank -2 updates, avoiding large matrix inversions from scratch.

(iii) Maintains positive - definiteness of the approximation ensuring descent directions.

c)

	nº of iterations	computational effort	robustness
Steepest descent			
Quasi Newton			

Task 5 – Optimal packing of 11 squares (56P)

11 identical **small squares**, each with a **side length of 1**, should be placed **inside one larger square** with the **side length b**. Your task is to find the **smallest side length b** that fits all 11 squares. The small squares can be rotated arbitrarily and can touch each other, but they should not overlap. To give you an idea of how a possible solution to this problem might look like: The figure below shows the optimal packing of 10 small squares.

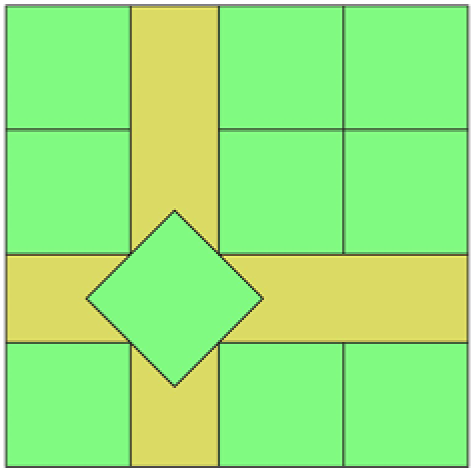

Figure 1: Example - Optimal packing of 10 small squares.

a) **Formulate** the corresponding optimization problem. Especially think about a suitable formulation of the constraints (give the mathematical formulation: objective function, optimization variables, constraints, dimensions). (12P)

b) **Perform** a mathematical optimization to **minimize the side length b** using MATLAB, Python, or CasADi. Clearly **comment** your code and **describe your approach** in a **step-by-step** manner (As a small hint: Consider starting with a simple script, such as placing the 11 small squares without overlapping. Then create the larger square that fits the smaller ones exactly). **Choose a optimization algorithm** for this problem and **discuss** your choice. Finally, **plot** your solution / your arrangement of the squares and **interpret** your result. (28P)

c) Do you expect that your solution of task b) is a **local or a global** optimum? **Perform a global optimization** to proof your expectation. (16P)

a) (i) Variables

→ $b \in \mathbb{R}$

→ For $i = 1, \dots, 11$:

↳ $c_i = (x_i, y_i) \in \mathbb{R}^2$ is the center of small square i

↳ $\theta_i \in \mathbb{R}$ is the rotation angle of small square i

Constants

(ii) The solution includes 2 steps: First, we approximate the squares as circles to get an initial guess and then close the gaps to get final solution.

I Circle approximation model

Let $p_{i,k} = c_i + R(\theta_i)\, d_k$, $k = 1, 2, 3, 4$, where $d_1 = (\frac{1}{2}, \frac{1}{2})$, $d_2 = (-\frac{1}{2}, \frac{1}{2})$, $d_3 = (-\frac{1}{2}, -\frac{1}{2})$, $d_4 = (\frac{1}{2}, -\frac{1}{2})$ and $R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ is the rotation matrix.

$$\min b$$

s.t.

$$0 \leq p_{i,k,x} \leq b \quad ; \quad i = 1, \ldots, 11 \quad ; \quad k = 1, \ldots, 4$$

$$0 \leq p_{i,k,y} \leq b \quad ; \quad i = 1, \ldots, 11 \quad ; \quad k = 1, \ldots, 4$$

$$(x_i - x_j)^2 + (y_i - y_j)^2 \geq 2 \quad ; \forall\, i < j$$

$$b \geq 1$$

$\rightsquigarrow$ Constraints 1 an 2 make sure the squares are always inside the big square. Constraint 3 is the circle approximation to avoid overlapping.

II Final model replacing the circle approximation

For every $i = 1, \ldots, 11$, define two orthonormal edge directions $e_{i1} = (\cos\theta_i, \sin\theta_i)$ and $e_{i2} = (-\sin\theta_i, \cos\theta_i)$.

For a pair $i < j$ consider the four candidate separating axes $U_{ij} = \{e_{i1}, e_{i2}, e_{j1}, e_{j2}\}$.

For each $u \in U_{ij}$ compute:

$\hookrightarrow$ $\mathrm{proj}(u) = sa((c_j - c_i)u)$, where $sa(t) := \sqrt{t^2 + \varepsilon}$

$\hookrightarrow$ $\rho_i(u) = \frac{1}{2}\left(sa(u \cdot e_{i1}) + sa(u \cdot e_{i2})\right)$

$\hookrightarrow \; p_j(u) = \frac{1}{2}\left(sa(u \cdot e_{j1}) + sa(u \cdot e_{j2}) \right)$

$\hookrightarrow \; f_{ij}^{(u)} = proj_{ij}(u) - \left(p_i(u) + p_j(u) \right)$

Now, let

$$LSE_\alpha(\{f_k\}) = \frac{1}{\alpha} \log\left(\sum_{k=1}^{4} e^{\alpha f_k} \right)$$

Model:

$$\min b$$

s.t.

$$0 \le p_{i,k,x} \le b \quad ; \; i = 1, \dots, 11 \; ; \; k = 1, \dots, 4$$

$$0 \le p_{i,k,y} \le b \quad ; \; i = 1, \dots, 11 \; ; \; k = 1, \dots, 4$$

$$LSE_\alpha\left(f_{ij}^{(e_{i1})}, f_{ij}^{(e_{i2})}, f_{ij}^{(e_{j1})}, f_{ij}^{(e_{j2})} \right) \ge 0$$

$$b \ge 1$$

The added third constraint forces smoothly the squares axis together from the stage 1 solution used as initial guess.

→ Chatgpt proposed the stage 2 solution for getting the squares close after I got stuck with the stage 1 solution.

c) I expect the solution to be a local minimum because the problem is highly nonconvex and the solver tends to converge to the closest local minimum given a initial guess.