Computer Laboratory Exercises in Optimization 2023
Lab 2. Penalty and barrier functions

You should work in a group of 2–3 students. We will investigate the following LP problems:

$$\text{minimize } x_1 + 2x_2 \qquad \text{subject to } \begin{cases} x_1 + x_2 & \geq & 2, \\ x_2 - x_1 & \geq & 0, \\ x_1 & \geq & 0, \\ x_1 + 2x_2 & \leq & 6, \end{cases} \tag{1}$$

$$\text{minimize } x_1 + 2x_2 \qquad \text{subject to } \begin{cases} -x_1 + 3x_2 & \leq & 12, \\ x_1 - 3x_2 & \leq & 2, \\ x_1 + 2x_2 & \geq & 2, \\ 3x_1 + 5x_2 & \leq & 34, \\ x_1 & \geq & 0, \\ x_2 & \geq & 0, \end{cases} \tag{2}$$

$$\text{minimize } -2x_1 - 4x_2 - 3x_3 \quad \text{subject to } \begin{cases} x_1 + 3x_2 + 2x_3 & \leq & 30, \\ x_1 + x_2 + x_3 & \leq & 24, \\ 3x_1 + 5x_2 + 3x_3 & \leq & 60, \\ x_1, x_2, x_3, & \geq & 0. \end{cases} \tag{3}$$

These problems should each be converted to unconstrained minimization problems, which are solved by MATLAB's function `fminunc`. Each region is of the form

$$S = \{\boldsymbol{x} : g_k(\boldsymbol{x}) \leq 0, k = 1, \ldots, m\}, \quad \text{where } \boldsymbol{g}(\boldsymbol{x}) := \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}.$$

Preparations before the lab session

- Read and be prepared to give a brief explanation to the penalty and barrier function methods.
- Use MATLAB's 'search documentation' and read about `fminunc`. Which is the default optimization algorithm (see `Algorithms`)? Which stop criterion is used (see `exitflag`)? Which is the first search direction of this algorithm? Explain the purpose of the algorithm.
- Download the files `Lab2.m` and `exampleNonlinear.m` from the course homepage.
- *In the same directory* as you saved the file `Lab2.m`, implement in MATLAB a penalty function of the type

$$\alpha(\boldsymbol{x}) = \sum_{k=1}^{m} \max(0, g_k(\boldsymbol{x})) \tag{p1}$$

  in a file `penalty.m` starting with `function alfa = penalty(x,A,b)`. You will not need to supply the matrices $\boldsymbol{A}$ and $\boldsymbol{b}$ for Problems (1)–(3) – that will be done automatically by `Lab2.m`.

- Do the same for the barrier function

$$\beta(\boldsymbol{x}) = -\sum_{k=1}^{m} \frac{1}{g_k(\boldsymbol{x})}$$

  in a file `barrier.m` starting with `function beta = barrier(x,A,b)`.

During the laboratory session

1. Type `Lab2` to start the program. The Penalty method on Problem (1) should be investigated by the `System` penalty function and a predefined sequence of penalty parameters $\{\mu_k\}_1^\infty$. Choose a starting point with the mouse. Then each click anywhere in the window runs `fminunc` for the current μ_k, which usually requires several iterations to converge. *Those iterations are not shown*, but only the resulting minimizer and a dashed line from the previous minimizer (with μ_{k-1}) (or the starting point). Try different starting points.

   - Do you think the method finds the minimizer? Why?
   - Are you satisfied with the approximation? Is the final point feasible? You can zoom in the figure with the magnifying-glass button. Do not forget to click once more before restarting with another initial point.

2. Select the Barrier method (still for Problem (1)) and investigate it as above.

   - Is there a difference in how you pick a starting point in these two methods?
   - Do the methods find the same point as in the Penalty method (the function values are shown)? If not, explain why.
   - How would you compare the Barrier method with the Penalty method for this problem? What are the advantages and drawbacks?

3. Now choose `Personal` to test your own penalty function (p1) (still for Problem (1)). You choose an initial point by clicking in the plane. Then choose a μ-value and click `Iterate`.

   - What happens if you choose μ-values close to zero and why?
   - What happens for μ-values between 1 and 5 and different starting points? Are the results satisfactory? Do you find a feasible point? What happens if you increase μ? Compare with the behaviour of the default penalty function. There is at least one problem with the penalty function (p1); which? You can right-click over the problem formulation in the window to choose contours. Then investigate more and try to explain what you see.
   - Make a modification of you penalty function that improves the behaviour. Do the same test as above. Are the results satisfactory? Do you find the minimizer? Do you find a feasible point?

4. Use your modified and improved penalty function (still for Problem (1)).

   - Suggest a sequence of μ_k that you think works well.
   - Can you get fewer iterations (mouse clicks) than with the default sequence of μ_k?
   - Try with a very large μ, e.g. 10^8, from the beginning and different starting points. Is the approximation good? Does the result depend on the starting point, for example, if you start really close to/far from the minimum, inside/outside the set?

5. Now investigate your own barrier function (`Personal`) in the same way as above and compare with the `System` one (still for Problem (1)).

- Do you need to make a modification of your barrier function so that it works well for all feasible starting points?
- After the modification, try your own sequence of ε_k-values. Can you find a sequence of ε_k that gives better convergence than the default one?
- Start with a very small ε like 10^{-8} or smaller. Try different starting points. Is the result satisfactory? Does it depend on the starting point?

6. Choose `Problem 2` and run the two methods with `System`. Do they give the same solution? Explain the phenomenon by studying Problem (2) and the contours when you iterate.

7. Choose `Problem 3` to investigate Problem (3). Note that you can turn around the feasible region. Now you have to type in the starting point coordinates manually, press return to see the starting point, and then click `Iterate`. Can you find sequences of the penalty parameters that are more efficient then the `System` ones?

8. You can solve your own 2D or 3D problems of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \end{array}$$

   with a nonlinear objective function by modifying the program `exampleNonlinear.m`, which uses your penalty function. Open the program `exampleNonlinear.m` in the editor. Run the program as it is and explain what you see. Did you get a good solution? How can you get a better approximation?

   Make changes in the program in order to solve some of Exercises 5.1, 5.2, 7.13–15 in the textbook.