Exercises for Lecture Course on Numerical Optimization (NUMOPT)
Albert-Ludwigs-Universität Freiburg – Summer Semester 2026

Exercise 4: Constrained Optimization

Léo Simpson, Prof. Dr. Moritz Diehl

The solutions for these exercises will be given and discussed during the exercise session on June the 23rd, 2026.
To receive feedback on your solutions, please hand it in during the exercise session on June the 23rd, 2026, or by e-mail to leo.simpson@imtek.uni-freiburg.de before the same date.

I Simple equality constrained optimization

In this exercise, we discuss the following simple equality constrained example (already discussed in the lecture):

$$\begin{aligned} \underset{x_1, x_2 \in \mathbb{R}}{\text{minimize}} \quad & x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 - 1 = 0. \end{aligned} \tag{1}$$

1. Is this problem convex?

> **Solution:** There is a nonlinear inequality constraint, hence the problem is not convex.

2. Write the Lagrangian $\mathcal{L}(x_1, x_2, \lambda)$ of the problem.

> **Solution:**
>
> $$\mathcal{L}(x_1, x_2, \lambda) = x_1 - \lambda(x_1^2 + x_2^2 - 1)$$

3. Derive the first order necessary conditions (FONC) of optimality for this problem.

> **Solution:** FONC:
>
> $$\nabla \mathcal{L}(x_1, x_2, \lambda) = 0$$
>
> Which implies:
>
> $$\begin{aligned} -2\lambda x_1 &= 0 \\ 1 - 2\lambda x_2 &= 0 \\ x_1^2 + x_2^2 &= 1 \end{aligned}$$

4. Solve the equations from the necessary conditions you derived.

> **Solution:** We find two solutions, $\bar{x} = (0, 1), \bar{\lambda} = 1/2$ and $\tilde{x} = (0, -1), \tilde{\lambda} = -1/2$.

5. What does it imply for the global minimizer of (1)?

Solution: We know that there exists at least one global minimzer because the feasibility set is compact. Hence, it is one of these two. Evaluating the function, we clearly see that $\tilde{x}$ is the minimizer.

6. For all stationary points, write the second order necessary conditions. For which one is it satisfied?

Solution: Write the Hessian of the Lagrangian: $\nabla^2 \mathcal{L}(x, \lambda) = \begin{bmatrix} -2\lambda & 0 \\ 0 & -2\lambda \end{bmatrix}$ Hence:

- for the stationary point $\bar{x} = (0, 1), \bar{\lambda} = 1/2$, we have $\nabla^2 \mathcal{L}(x, \lambda) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$, which is not positive semi-definite
- for the stationary point $\tilde{x} = (0, -1), \tilde{\lambda} = -1/2$, we have $\nabla^2 \mathcal{L}(x, \lambda) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, which is positive definite.

7. What does it imply for the local minimizer of (1)?

Solution: We already knew that $\tilde{x}$ was a local minimizer. Now we know that it is the only one.

8. Pick one of the stationary points. Invent an additional equality constraint, such that the linear independence constraint qualification (LICQ) is violated at this point.

Solution: We can add the constraint $x_2^2 = 1$ for example, which violates the LICQ condition for both stationary points.

II Lifted Newton method

In this exercise we compare two different approaches to solve the following nonlinear equation:

$$w^{16} = 10. \tag{2}$$

The first approach is to use Newton's method directly on the function $F : \mathbb{R} \to \mathbb{R}$ defined by:

$$F(w) = w^{16} - 10. \tag{3}$$

The second approach is to use Newton's method on the function $\widetilde{F} : \mathbb{R}^4 \to \mathbb{R}^4$ defined by:

$$\widetilde{F}(\omega) = \begin{bmatrix} \omega_2 - \omega_1^2 \\ \omega_3 - \omega_2^2 \\ \omega_4 - \omega_3^2 \\ 2 - \omega_4^2 \end{bmatrix}. \tag{4}$$

1. Complete the file `non_lifted_newton.py` to implement approach 1.

   Analyse the number of iteration for convergence with respect to the initial guess $w^{[0]}$.

   Solution: See Python file `non_lifted_newton_sol.py`
   The number of iteration required for convergence for small values of $w^{[0]}$ is very large (can get to 700 iterations).

2. Complete the file `lifted_newton.py` to implement approach 2.

   Use initial values of the form $\omega^{[0]} = \begin{bmatrix} \bar{\omega} \\ \bar{\omega} \\ \bar{\omega} \\ \bar{\omega} \end{bmatrix}$.

   Analyse the number of iteration for convergence with respect to the initial guess $\bar{\omega}$.

   Comment on the comparison of the two approaches

   Solution: See Python file `lifted_newton_sol.py`
   It seems that the second approach is more robust because it converges faster for a larger range of initial values. One interpretation is that the lifted function has more components, but each component is less nonlinear, so each linearization is more accurate.

III Control of a dynamic system

A controlled dynamical system is a system that evolves with time, according to the following law:

$$\begin{aligned} s_0 &= \bar{s}_0 \\ s_{t+1} &= \phi(s_t, u_t) \quad \text{for } t = 0, \dots, N-1 \end{aligned} \tag{5}$$

for some function $\phi(\cdot, \cdot)$, some initial state $\bar{s}_0$ and some control sequence $u_0, \dots, u_N$. In this exercise, we consider the following function $\phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$:

$$\phi(s_t, u_t) = s_t + \Delta t((s_t + 1)^3 + u_t), \tag{6}$$

with $\Delta t = 0.1$.
Here, we study a so called *optimal control problem*, where the best control sequence $u_0, \dots, u_N$ is determined by minimizing a cost function. More precisely, we will solve the following optimization problem:

$$\begin{aligned} \underset{(s_0, \dots, s_N), (u_0, \dots, u_N)}{\text{minimize}} \quad & \frac{1}{2} \sum_{t=0}^{N} u_t^2 \\ \text{subject to} \quad & s_0 = \bar{s}_0, \\ & s_{t+1} = \phi(s_t, u_t), \quad t = 0, \dots, N-1, , \\ & s_N = \bar{s}_N \end{aligned} \tag{7}$$

The objective and constraints express our aim to bring the terminal state s_N to $\bar{s}_N$ using the least amount of effort in terms of control actions u_t.

In this exercise, we will implement an algorithm to solve the optimization (7).

1. Transform the problem in the standard least-square form:

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2} \|F(x)\|^2 \qquad = \frac{1}{2} \sum_{j=1}^{p} F_j(x)^2 \\ \text{subject to} \quad & g(x) = 0 \end{aligned} \tag{8}$$

   where you have to define the dimensions n, m, p, the variable x, and the functions $g(x) \in \mathbb{R}^m$ and $F(x) \in \mathbb{R}^p$.

   Solution: $p = N + 1$, $n = 2(N + 1)$, $m = N + 2$, and:

$$x := \begin{bmatrix} s_0 \\ \vdots \\ s_N \\ u_0 \\ \vdots \\ u_N \end{bmatrix}, \quad F(x) := \begin{bmatrix} u_0 \\ \vdots \\ u_N \end{bmatrix}, \quad g(x) := \begin{bmatrix} s_0 - \bar{s}_0 \\ s_1 - \phi(s_0, u_0) \\ \vdots \\ s_N - \phi(s_{N-1}, u_{N-1}) \\ s_N - \bar{s}_N \end{bmatrix}$$

2. In the lecture, we saw a popular algorithm for solving (8): *the Gauss-Newton* algorithm. This is an iterative algorithm, where at each iteration, the functions $F(x)$ and $g(x)$ are linearized around the current guess $x^{[k]}$. Write down the optimization problem that is solved at each iteration of the algorithm in the general form.

What kind of optimization problem is this?

Solution: If our current guess is $x^{[k]}$, we compute the next guess $x^{[k+1]}$ by solving the following problem:

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2}\left\| F\left(x^{[k]}\right) + \nabla F\left(x^{[k]}\right)^\top \left(x - x^{[k]}\right)\right\|^2 \\ \text{subject to} & g\left(x^{[k]}\right) + \nabla g\left(x^{[k]}\right)^\top \left(x - x^{[k]}\right) = 0 \end{array}$$

This is a Quadratic Programming (QP) problem without any inequality constraints.

3. Write down first order the optimality conditions for the subproblem that is solved at each step in its general form.

   Hint: You may want to define $J^{[k]} := \nabla F\left(x^{[k]}\right)$.

Solution: The first order optimality conditions are given by the following conditions:

$$\begin{aligned} J^{[k]} F\left(x^{[k]}\right) + J^{[k]} J^{[k]\top}\left(x - x^{[k]}\right) - \nabla g\left(x^{[k]}\right)\lambda &= 0 \\ g\left(x^{[k]}\right) + \nabla g\left(x^{[k]}\right)^\top \left(x - x^{[k]}\right) &= 0 \end{aligned}$$

where $\lambda \in \mathbb{R}^m$ is the Lagrange multiplier associated with the equality constraint $g(x) = 0$.

4. Put the first order optimality conditions in the form of a linear system:

$$A^{[k]} w^{[k]} = b^{[k]} \tag{9}$$

   where $w^{[k]}$ is the vector of unknowns (that you also need to define), and $A^{[k]}$ and $b^{[k]}$ are matrices and vectors that depend on the current guess $x^{[k]}$.

Solution:

$$\underbrace{\begin{bmatrix} J^{[k]} J^{[k]\top} & -\nabla g\left(x^{[k]}\right) \\ -\nabla g\left(x^{[k]}\right)^\top & 0 \end{bmatrix}}_{=:A^{[k]}} \underbrace{\begin{bmatrix} x - x^{[k]} \\ \lambda \end{bmatrix}}_{=:w^{[k]}} = \underbrace{\begin{bmatrix} -J^{[k]} F\left(x^{[k]}\right) \\ g\left(x^{[k]}\right) \end{bmatrix}}_{=:b^{[k]}}$$

5. Complete the code in the file `control.py` to implement the algorithm discussed in the previous questions, and visualize the iterations of the algorithm using the code provided.

   Hint: The function $\phi(\cdot,\cdot)$ *and its derivatives are already implemented in the file* `control_phi.py`. *The visualization code is already implemented in the file* `control_animation.py`.

   We choose $N = 100$, $\bar{s}_0 = 0$, and $\bar{s}_N = 1$.

Solution: See Python file `control_sol.py`.

6. For each sequence of control actions $u_0, \ldots, u_{N-1}$, we can compute the corresponding sequence of states $\hat{s}_0, \ldots, \hat{s}_N$ using the equations (5). This is already implemented in the function `rollout` in the file `control_phi.py`.

   Use that function to also compute, for each iteration of the algorithm, the sequence of states $\hat{s}_0^{[k]}, \ldots, \hat{s}_N^{[k]}$ that would be produced by the controls $u_0^{[k]}, \ldots, u_{N-1}^{[k]}$ that are computed by the algorithm.

Add these sequences of states to the visualization. Comment what you observe, and give an interpretation of the results.

Hint: *You can use the syntax*

```
make_animation(list_s,list_u, list_other_s=my_other_list_s)
```

to visualize the trajectories of state and controls `list_s` *and* `list_u`*, and the additional trajectory of states* `list_other_s`*.*

The additional trajectory of states will be plotted in purple.

Solution: See Python file `control_sol.py`.
What we observe is that at the early iterations of the algorithm, the trajectory of states and the trajectories of controls are very different. This is due to the fact that the equality constraints are not yet satisfied.
We also see that the trajectory of states $s_0^{[k]}, \ldots, s_N^{[k]}$ is much closer to the desired goal (reaching the terminal state $\bar{s}_N$) than the trajectory of states $\hat{s}_0^{[k]}, \ldots, \hat{s}_N^{[k]}$. This is because while the algorithm tries to find a feasible solution, it is also considering the terminal constraint.