Exercises for Lecture Course on Numerical Optimization (NUMOPT)
Albert-Ludwigs-Universität Freiburg – Winter Term 2019-2020

Exercise 4: Calculation of Derivatives, Equality Constrained Optimization

Prof. Dr. Moritz Diehl, Dimitris Kouzoupis, Andrea Zanelli and Florian Messerer

Aim of this exercise is to gain experience with all derivative computation methods discussed in the class. You will then explore the concepts of equality constrained optimization discussed in the lecture.

Exercise Tasks

1. **Control of a dynamic system:** Our goal is to drive the state $x_k \in \mathbb{R}$ of a discrete time system to the origin using controls $u_k \in \mathbb{R}$ in N time intervals, where subscript k denotes discrete time. More precisely, we are interested in solving the optimization problem:

$$\underset{u,x}{\text{minimize}} \sum_{k=0}^{N-1} u_k^2 + q x_N^2 \tag{1a}$$

$$\text{subject to: } x_0 = \bar{x}_0 \tag{1b}$$

$$x_{k+1} = x_k + \frac{T}{N}((1 - x_k)x_k + u_k), \quad k = 0, \dots N-1, \tag{1c}$$

with initial condition $\bar{x}_0$, control trajectory $u = [u_0, \dots, u_{N-1}]^\top \in \mathbb{R}^N$, state trajectory $x = [x_0, \dots, x_N]^\top \in \mathbb{R}^{N+1}$ and terminal time T (corresponding to discrete time N). The objective (1a) expresses our aim to bring the terminal state x_N to zero, using the least amount of effort in terms of control actions u_k. Weighting factor $q \in \mathbb{R}$ defines the trade-off between these two aims. The equality constraints (1b) and (1c) uniquely determine the state trajectory $x_0, \dots x_N$ given controls $u_0, \dots, u_{N-1}$. Therefore we can write (1) in the equivalent unconstrained form:

$$\underset{u}{\text{minimize}} \sum_{k=0}^{N-1} u_k^2 + \Phi(u), \tag{2}$$

using the constraints to eliminate all states x_k, and to define the nonlinear function $\Phi(\cdot) : \mathbb{R}^N \to \mathbb{R}$. This function is implemented for you in MATLAB (`Phi.m`). You can call it as `f = Phi(u, param)` where $u \in \mathbb{R}^N$ is a control trajectory and param a structure with the problem parameters, similar to the previous exercise sheet. You will now implement different methods for obtaining derivatives of Φ and compare their results. We will use a random control trajectory u_{rand} to evaluate the derivatives. This has already been implemented for you in `test_derivatives.m`.

   (a) Use your code from last week to differentiate $\Phi(u)$ at u_{rand} with finite differences.

   (1 point)

   (b) Using the same syntax, write a function `[F,J] = i_trick(fun,x,param)` that calculates the Jacobian of $\Phi(u)$ using the imaginary trick.

   (1 point)

   (c) Now let's implement both forward and backward modes of Automatic Differentiation. Before you start coding, which of the two you think would perform faster in our example and why?

   (1point)

(d) Write a MATLAB function `[F,J] = Phi_FAD(u,param)` that returns the function evaluation and the Jacobian of $\Phi(u)$ using the forward mode of AD. You can start by copying the code from the given function `Phi`.

(2 points)

(e) Write a MATLAB function `[F,J] = Phi_BAD(u,param)` that implements the backward mode of AD.

(2 points)

(f) The 'AD' in 'CasADi' stands for Algorithmic Differentiation, since this is how CasADi computes derivatives. Using CasADi we can build the Jacobian of our nonlinear function as a symbolic expression within a few lines only.

Complete the template `casadi_script.m` to compute the Jacobian of $\Phi(u)$. Note that this time we are not using the `Opti()` environment, since we are interested in derivatives only.

(1 point)

(g) Once you have everything implemented, run the script `test_derivatives.m` to check (and demonstrate) that your results are correct. If you did not implement all methods, comment out or delete the corresponding lines.

(0 points)

(h) Use MATLAB's `tic toc` to measure the total time spent in the derivative calculations for the different functions you have implemented, with $N = 200$. For CasADi make sure you are only measuring the function evaluation time, i.e., without the setup time / time for running `casadi_script.m`. How do the timings change if you set $N = 1000$? Give a short reason for this behaviour. Report the time values for all methods and both values of N.

Remark: Depending on the performance of your CPU you may adapt the given values of N for purpose of better demonstration / to decrease the runtime. The cost of calling one of your derivative functions should be in the order of magnitude from approx. 10^{-5} to 1 seconds.

(1 point)

(i) **Extra:** Solve the optimization problem in (1) using the BFGS method with globalization similarly to the previous exercise (you can simply adapt your code from the last exercise sheet). Plot the state and controls as a function of time to confirm that the system behaves as expected. Don't forget to add the derivative of the quadratic term $\sum_{k=0}^{N-1} u_k^2$ to your result for $\nabla\Phi(u)^\top$ when computing the Jacobian of your objective function. Use $N = 50$, $x_0 = 2$, $T = 5$ and $q = 50$.

(2 bonus points)

2. **Optimal perturbation for finite differences:** Assume we have a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ and we want to evaluate its derivative $f'(x_0)$ at x_0 with finite differences. Further assume that in a neighborhood $\mathcal{N}(x_0)$ it holds:

$$|f''(x)| \leq f''_{\max}, \quad |f(x)| \leq f_{\max} \tag{3}$$

with $\mathcal{N}(x_0) := \{x \,|x_0 - \delta \leq x \leq x_0 + \delta\}$, $\delta > t$ and t the perturbation in the finite difference approximation.

The function $f(x)$ can be represented on a computing system with accuracy ϵ_{mach}, i.e., it is perturbed by noise $\epsilon(x)$:

$$\tilde{f}(x) = f(x)(1 + \epsilon(x)) \quad |\epsilon(x)| \leq \epsilon_{\text{mach}}. \tag{4}$$

(a) Compute a bound ψ on the error of the finite difference approximation of $f'(x_0)$

$$\left| \frac{\tilde{f}(x_0 + t) - \tilde{f}(x_0)}{t} - f'(x_0) \right| \leq \psi(t, f_{\max}, f''_{\max}, \epsilon_{\text{mach}}). \tag{5}$$

(2 points)

(b) Which value t^* minimizes this bound and which value has the bound at t^*?

(1 point)

(c) **Extra:** Do a similar analysis for the central differences where $\tilde{f}'(x_0) = \frac{\tilde{f}(x_0+t)-\tilde{f}(x_0-t)}{2t}$.
Hint: you can assume that also the third derivative is bounded in $[x_0 - t, x_0 + t]$.

(2 bonus points)

3. **Simple equality constrained optimization:** Recall the simple equality constrained example discussed in the lecture,

$$\underset{x_1,x_2}{\text{minimize}}\ x_2 \tag{6}$$

$$\text{subject to: } x_1^2 + x_2^2 - 1 = 0 \tag{7}$$

which consists of a linear objective and a nonlinear equality constraint.

(a) Study the template file `fmincon_example.m` to understand how `fmincon` works. You will need to use it yourself in the next task. Type `doc fmincon` or `help fmincon` in the command window for more details on the function.

(0 points)

(b) On paper derive the first order necessary conditions for this problem. Code a simple Newton method of your choice to find the optimal solution. To which point (or points) does your method converge?

(2 points)

(c) Check whether you have found the minimum point using the Second Order Sufficient Conditions or Second Order Necessary Conditions (on paper).

(1 point)

4. **Hanging chain, re-revisited:** Let us consider once again the example of the hanging chain that we will now solve with `fmincon`. This time we fix the distance between two adjacent masses to a constant length $l = L/(N + 1)$. Removing the potential energy of the springs from our objective function we are left with the following optimization problem:

$$\underset{y,z}{\text{minimize}} \quad \sum_{i=0}^{N+1} m\, g_0\, z_i \tag{8a}$$

$$\text{subject to} \quad (y_0, z_0) \ = \ (-2, 1) \tag{8b}$$

$$(y_{N+1}, z_{N+1}) = \ (2, 1) \tag{8c}$$

$$(y_i - y_{i+1})^2 + (z_i - z_{i+1})^2 - l^2 = 0, \ \text{for} \ \ i = 0, \dots N \tag{8d}$$

For your experiments use the parameter values $N = 21$, $L = 5$ m, $m = 0.2$ kg and $g = 9.81\ \frac{\text{m}}{\text{s}^2}$.

(a) Write a function `[f] = chain_objective(x,param)` that implements the objective and a second function `[C,Ceq] = chain_constraints(x,param)` that implements the $N+1$ nonlinear equality constraints. Order the decision vector x as $x = [y_1, z_1, \dots, y_N, z_N]^\top$ and eliminate the edge variables y_0, z_0, y_{N+1}, z_{N+1} using the constraints (8b) and (8c), similar to the last exercise sheet.

(2 points)

(b) Use `fmincon` to solve the equality constrained optimization problem. You may need to tune the options to allow the algorithm to fully converge to an optimal solution. You can illustrate your results with the plotting function `plot_chain(x,param)`.

(2 points)

(c) Evaluate the Jacobian of your constraints at the optimal solution (e.g. by using the `i_trick` function from the first exercise) and check whether LICQ holds. *Hint: Use the MATLAB command* `rank`.

(2 points)

(d) Extend your function `[C,Ceq] = chain_constraints(x,param)` to fix a third point of the chain somewhere between the other two, as demonstrated in Figure 1.

(1 point)

(e) Assuming you can fix only one mass point at a time (apart from the edges), fixing which mass points would lead to infeasibility or LICQ violation? Why? Choose a feasible set of constraints for your problem and confirm your intuition numerically. For fixing mass point k to position $(\bar{y}, \bar{z})$ add the equality constraints $y_k = \bar{y}$ and $z_k = \bar{z}$ to your NLP.

(2 points)

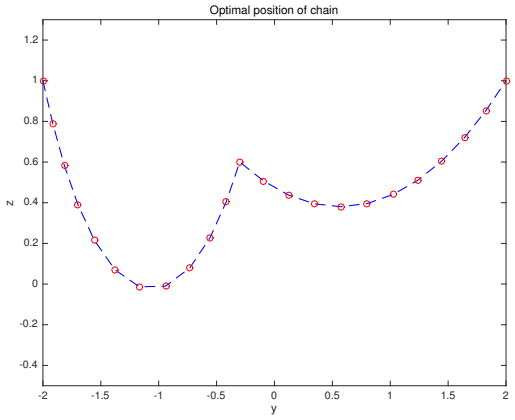

Figure 1: Hanging chain with three fixed masses.

5. **LICQ and Newton method:** Consider the following nonlinear equality constrained optimization problem:

$$\begin{aligned} \min_x \quad & f(x) \\ s.t. \quad & g(x) = 0, \end{aligned}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ and the linear system associated with the k-*th* iteration of the Newton method:

$$\begin{bmatrix} B & A^T \\ A & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ -\Delta\lambda \end{pmatrix} = - \begin{pmatrix} \nabla f(x^k) - \nabla g(x^k)\lambda \\ g(x^k) \end{pmatrix} \tag{9}$$

with $B := \nabla_x^2 f(x^k) - \sum_{i=1}^p \lambda_i \nabla_x^2 g_i(x^k)$ and $A = \nabla_x g(x^k)^\top$.

Prove that if A has full row rank and $Z^T B Z \succ 0$, with the columns of $Z \in \mathbb{R}^{n\times(m-n)}$ forming a basis for the null space of A, the iteration matrix in (9) is invertible. *Remark: this provides a sufficient condition under which a search direction can be obtained.* (2 points)

This sheet gives in total 26 points and 4 bonus points.