Exercises for Lecture Course on Optimal Control and Estimation (OCE)
Albert-Ludwigs-Universität Freiburg – Summer Term 2014

Exercise 5 - Direct Multiple Shooting

Prof. Dr. Moritz Diehl and Greg Horn

In this exercise, we use the simultaneous approach with SQP and a Gauss-Newton Hessian to solve an optimal control problem.

1. Throughout this exercise sheet, we regard the discrete time system

$$x_{k+1} = \Phi(x_k, u_k)$$

that is generated by one RK4 step applied to a controlled nonlinear pendulum. Its state is $x = (p, v)^T$ and the ODE $\dot{x} = f(x, u)$ is with $C := 180/\pi/10$ given as

$$f(x,u) = \begin{bmatrix} v(t) \\ -C\sin(p(t)/C) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t). \tag{1}$$

We use a time step $\Delta t = 0.2$ and recall that one RK4 step is given by

$$\begin{aligned} k_1 &= f(x_k, u_k) && (2)\\ k_2 &= f(x_k + \frac{1}{2}\Delta t \cdot k_1, u_k) && (3)\\ k_3 &= f(x_k + \frac{1}{2}\Delta t \cdot k_2, u_k) && (4)\\ k_4 &= f(x_k + \Delta t \cdot k_3, u_k) && (5)\\ x_{k+1} &= x_k + \Delta t \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) && (6) \end{aligned}$$

Write the function $\Phi(x_k, u_k)$ as a MATLAB code encapsulated in a single function `[xnew]=RK4step(x,u)`

2. Now we choose the initial value $\bar{x}_0 = (10, 0)^T$ and $N = 50$ time steps. We also define bounds on the sizes of p, v, and u, namely $p_{\max} = 10$, $v_{\max} = 10$, i.e. $x_{\max} = (p_{\max}, v_{\max})^T$, and $u_{\max} = 3$. The optimization problem we want to solve is given by

$$\begin{aligned} \underset{x_0, u_0, x_1, \ldots, u_{N-1}, x_N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \|u_k\|_2^2 && (7a)\\ \text{subject to} \quad \bar{x}_0 - x_0 &= 0, && (7b)\\ \Phi(x_k, u_k) - x_{k+1} &= 0, \quad \text{for} \quad k = 0, \ldots, N-1, && (7c)\\ x_N &= 0, && (7d)\\ -x_{\max} \le x_k &\le x_{\max}, \quad \text{for} \quad k = 0, \ldots, N-1, && (7e)\\ -u_{\max} \le u_k &\le u_{\max}, \quad \text{for} \quad k = 0, \ldots, N-1. && (7f) \end{aligned}$$

Summarizing the variables of this problem in a vector $w = (x_0, u_0, \ldots, u_{N-1}, x_N) \in \mathbb{R}^n$ of dimension $n = 152$, formulate in the space below the nonlinear function $G(w)$, Hessian matrix H, and bounds $w_{\max}$ such that the above problem is an NLP of the following form:

$$\begin{aligned} \underset{w \in \mathbb{R}^{152}}{\text{minimize}} \quad & w^T H w && (8a)\\ \text{subject to} \quad G(w) &= 0, && (8b)\\ -w_{\max} \le w &\le w_{\max}. && (8c) \end{aligned}$$

Define what H_x and H_u need to be in the Hessian

$$H = \begin{bmatrix} H_x & & & \\ & H_u & & \\ & & \ddots & \\ & & & H_x \end{bmatrix}, \quad H_x = \begin{bmatrix} \quad \end{bmatrix} \quad , H_u = \begin{bmatrix} \quad \end{bmatrix} \quad .$$

For G, use the same ordering as in the OCP above:

$$G(w) = \begin{bmatrix} \\ \vdots \\ \\ \end{bmatrix} \qquad w_{\max} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

Construct the matrix H and vector $w_{\max}$ in MATLAB, and write a MATLAB function `[G]=Gfunc(w)`.

3. Check if your function $G(w)$ does what you want by writing a forward simulation function `[w]=simulate(x0,U)` that simulates, for a given initial value x_0 and control profile $U = (u_0, \ldots, u_{N-1})$, the whole trajectory $x_1, \ldots, x_N$ and constructs from this the full vector $w = (x_0, u_0, x_1, \ldots, x_N)$. If you generate for any x_0 and U a vector w and then you call your function $G(w)$ with this input, nearly all your residuals should be zero. Which components will not be zero?

   As a test, simulate e.g. with $x_0 = (5, 0)$ and $u_k = 1$, $k = 0, \ldots, N-1$ in order to generate w, and then call $G(w)$, to test that your function G is correct.

4. The SQP with Gauss-Newton Hessian (also called *constrained Gauss-Newton method*) solves a linearized version of this problem in each iteration. More specific, if the current iterate is $\bar{w}$, the next iterate is the solution of the following QP:

$$\begin{aligned} \underset{w \in \mathbb{R}^{152}}{\text{minimize}} \quad & w^T H w && && \text{(9a)} \\ \text{subject to} \quad & G(\bar{w}) + J_G(\bar{w})(w - \bar{w}) &=& 0, & \text{(9b)} \\ & -w_{\max} \leq w &\leq& w_{\max}. & \text{(9c)} \end{aligned}$$

   Important for implementing the Gauss-Newton method is the computation of the Jacobian $J_G(w) = \frac{\partial G}{\partial w}(w)$, which is block sparse with as blocks either (negative) unit matrices or the partial derivatives $A_k = \frac{\partial \Phi}{\partial x}(x_k, u_k)$ and $B_k = \frac{\partial \Phi}{\partial u}(x_k, u_k)$. Fill in the corresponding blocks in the following matrix

$$J_G(w) = \begin{bmatrix} & \ddots & \ddots & \ddots & \\ & & & & \\ & & & & \end{bmatrix}$$

5. Today we compute the Jacobian $J_G(w)$ just by finite differences, perturbing all 152 directions one after the other. This needs in total 153 calls of G. Give your routine e.g. the name `[G,J]=GfuncJacSlow(w)`. Compute J_G for a given w (e.g. the one from above) and look at the structure of this matrix, e.g. using the command `spy(J)`.

6. Now learn how to use the MATLAB QP solver `quadprog` e.g. by calling `help quadprog`.

7. Write a function `[wplus]=GNStep(w)` that performs one SQP-Gauss-Newton step by first calling `[G,J]=GfuncJac(w)` and then solving the resulting QP (9) using `quadprog`. Note that the QP is a very sparse QP but that this sparsity is not exploited in full during the call of `quadprog`.

8. Write a loop around your function `GNStep`, initialize the GN procedure at at $w = 0$, and stop the iterations when $\|w_{k+1} - w_k\|$ gets smaller than 10^{-4}. Plot the iterates as well as the vector G during the iterations. How many iterations do you need?

9. * [OPTIONAL] A different algorithm is obtained if we overwrite before each call of the function `GNStep` the values for the states within `w` by the result of a forward simulation, using the corresponding controls and the initial value $\bar{x}_0$. Run it again with this modification, using the same zero initialization for the controls. How many iterations do you need now? Do you know to which of the algorithms from the previous exercises this new method is equivalent?

10. * [OPTIONAL] If you like, you can also construct the Jacobian in a computationally much more efficient way, as follows:

    - First write a function `[xnew,A,B]=RK4stepJac(x,u)` using finite differences with a step size of $\delta = 10^{-4}$. Here, $A = \frac{\partial \Phi}{\partial x}(x, u)$ and $B = \frac{\partial \Phi}{\partial u}(x, u)$.
    - Using this function `[xnew,A,B]=RK4stepJac(x,u)`, implement a function `[G,J]=GfuncJacFast(w)`.
    - Compare if the result is correct by taking the difference of the Jacobians you obtain by `[G,J]=GfuncJacFast(w)` and `[G,J]=GfuncJacSlow(w)`.