Exercises for Lecture Course on Optimal Control and Estimation (OCE)
Albert-Ludwigs-Universität Freiburg – Summer Term 2015

Exercise 6 - Sequential Quadratic Programming (deadline: 15.6.2015, 2:15pm)

Prof. Dr. Moritz Diehl and Jonas Koenemann

In this exercise, we use the same simple pendulum system as in the previous exercise sheets. This time, we solve the non-linear problem formulated as an Non-Linear Program (NLP) by a self written Sequential Quadratic Programming (SQP) solver with Gauss-Newton Hessian.

Recall the system dynamics of the pendulum with state $x = [\phi, \omega]^\top$:

$$\dot{x} = \begin{bmatrix} \omega \\ 2\sin(\phi) + u \end{bmatrix}$$

and the optimal control problem that we want to solve

$$\begin{aligned}
\underset{x_0,\ldots,x_N,u_0,\ldots,u_{N-1}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} (\phi_k^2 + u_k^2) \\
\text{subject to} \quad & \\
\bar{x}_0 - x_0 &= 0 \\
f(x_k, u_k) - x_{k+1} &= 0, \quad \text{for} \quad k = 0, \ldots, N-1 \\
\omega_{\min} &\leq \omega_k \leq \omega_{\max}, \quad \text{for} \quad k = 0, \ldots, N \\
u_{\min} &\leq u_k \leq u_{\max}, \quad \text{for} \quad k = 0, \ldots, N-1
\end{aligned}$$

where the discrete time system dynamics are obtain by a Runge-Kutta integrator of order 4 with a timestep of $h = 0.2$. The horizon of the optimal control problem is $N = 60$, the given initial state is $\bar{x}_0 = [-\pi, 0]^\top$, and the bounds are given by $\omega_{\min} = -\pi$, $\omega_{\max} = \pi$, $u_{\min} = -1.1$, $u_{\max} = 1.1$.

The NLP with variables $y = [x_0^\top, u_0, \ldots, u_{N-1}, x_N^\top]^\top$ that we solved in the last exercise sheet with `fmincon` has the form:

$$\begin{aligned}
\underset{y}{\text{minimize}} \quad & \psi(y) \\
\text{subject to} \quad & \\
G(y) &= 0, \\
y_{\min} \leq\ & y \leq y_{\max}
\end{aligned}$$

In this exercise sheet, we will first prepare the calculation of the Jacobian that is needed in the SQP iterations. We will test the correctness of the Jacobian by passing it to the `fmincon` solver and find a faster way to compute the Jacobian.

Then we will implement the SQP solver by solving the Quadratic Programs (QP) in each iteration with then Matlab `quadprog` function.

Use the same initialization for y_0 as in Exercise Sheet 5. You can start by using the template `sqp_template.m` on the course page which includes the solution to the last exercise sheet.

1. Inside the `fmincon` function the Jacobian of the non-linear equality constraints $J_G(y) = \frac{\partial G}{\partial y}(y)$ is needed quite often and internally calculated by finite differences. It is possible to manually calculate the Jacobian and pass it to the solver.

   Calculate the Jacobian $J_G(y)$ by finite differences, perturbing all 182 directions one after the other using $\delta = 10^{-4}$. This needs in total 183 calls of G. Give your routine e.g. the name `[jac,Gy]=GJacSlow(y)`. Compute J_G for w_0 and look at the structure of this matrix by making a plot using the command `spy(J)`.

   Pass this Jacobian to your constraints function (see Matlab constraints documentation). How many iterations and how much time does the solver need to converge? Specify which algorithm you used.

   Don't forget to set the option to tell the solver to use your Jacobian:

   ```
   options=optimoptions(@fmincon,...,'GradConstr','on',...);
   ```

   (2 points)

2. By looking at the structure of J_G, we see that the matrix is very sparse can be calculated much more efficiently. The Jacobian $J_G(y) = \frac{\partial G}{\partial y}(y)$ is block sparse with as blocks either (negative) unit matrices or the partial derivatives $A_k = \frac{\partial f}{\partial x}(x_k, u_k)$ and $B_k = \frac{\partial f}{\partial u}(x_k, u_k)$. Fill in the corresponding blocks in the following matrix

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

(2 points)

3. With this knowledge you can construct the Jacobian in a computationally much more efficient way, as follows:
   - First write a function `[A,B]=RK4stepJac(x,u)` using finite differences with a step size of $\delta = 10^{-4}$. Here, $A = \frac{\partial f}{\partial x}(x, u)$ and $B = \frac{\partial f}{\partial u}(x, u)$.
   - Using this function `[A,B]=RK4stepJac(x,u)`, implement a function `[jac,Gy]=GJacFast(y)`.
   - Compare if the result is correct by taking the difference of the Jacobians you obtain by `[jac,Gy]=GJacFast(y)` and `[jac,Gy]=GJacSlow(y)`.

     Pass this Jacobian to your constraints function. How many iterations and how much time does the solver need now?

(2 points)

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

5. 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 $\bar{y}$, the next iterate is the solution of the following Quadratic Program (QP):

$$\underset{y}{\text{minimize}} \quad y^T H y \tag{1a}$$

$$\text{subject to} \quad G(\bar{y}) + J_G(\bar{y})(y - \bar{y}) = 0, \tag{1b}$$

$$y_{\min} \le y \le y_{\max}. \tag{1c}$$

Define what H_x and H_u need to be in the Hessian

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

(1 point)

6. Write a function `[ybar_next]=GNStep(ybar)` that performs one SQP-Gauss-Newton step by first calling `[jac,Gy] = GJacFast(y)` and then solving the resulting QP (1) 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`.

7. Write a loop around your function `GNStep`, initialize the GN procedure at at y_0, and stop the iterations when $\|y_{k+1} - y_k\|$ gets smaller than 10^{-4}. Plot the iterates as well as the vector G during the iterations. How many iterations do you need? How much time does your SQP solver need to converge?

   Plot the evolution of the state and the applied controls in time.

(3 points)

8. **Bonus question:** Find out how to exploit sparsity in the `quadprog` solver and solve the SQP with the sparse QP solver. How much time does your SQP solver need to converge now?

(2 bonus points)