In this exercise we use the direct multiple shooting approach with SQP and a Gauss-Newton Hessian to solve a simple optimal control problem.

1. First, we use multiple shooting to define a discrete time system

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

that is generated by \( m = 5 \) Euler steps applied to a controlled nonlinear pendulum. The state of the pendulum is \( x = (p, v)^T \) and the ODE \( \dot{x} = f(x, u) \) is given as

\[ f(x, u) = \begin{bmatrix} v \\ -\sin p \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{1} \]

We use a time step of \( \Delta t = 0.04 \) seconds and start with the internal state

\[ \xi_0 := x_k \]

and then obtain the output \( x_{k+1} := \xi_m \) using the loop

\[ \xi_{i+1} := \xi_i + \Delta t f(\xi_i, u_k), \quad i = 0, \ldots, m - 1. \tag{2} \]

Write the discrete time function \( \Phi(x_k, u_k) \) as a MATLAB code encapsulated in a single function \([x_{\text{new}}]=\Phi(x,u)\). Note that each call of this function simulates the system for a duration of \( 5 \cdot 0.04 = 0.2 \) seconds for a fixed control input.

2. Now we choose the initial value \( \bar{x}_0 = (1, 0)^T \) and \( N = 50 \) multiple shooting stages (i.e. a total time horizon of 10 seconds). We also define bounds on the sizes of \( p, v, \) and \( u \), namely \( p_{\text{max}} = 10, v_{\text{max}} = 10 \), i.e. \( x_{\text{max}} = (p_{\text{max}}, v_{\text{max}})^T \), and \( u_{\text{max}} = 3 \). The
optimization problem we want to solve is given by

\[
\begin{align*}
\text{minimize} & \quad x_0, u_0, x_1, \ldots, u_{N-1}, x_N \\
\text{subject to} & \quad \sum_{k=0}^{N-1} \|u_k\|_2^2 \\
& \quad \bar{x}_0 - x_0 = 0, \\
& \quad \Phi(x_k, u_k) - x_{k+1} = 0, \quad \text{for } k = 0, \ldots, N-1, \\
& \quad x_N = 0, \\
& \quad -x_{\text{max}} \leq x_k \leq x_{\text{max}}, \quad \text{for } k = 0, \ldots, N-1, \\
& \quad -u_{\text{max}} \leq u_k \leq u_{\text{max}}, \quad \text{for } k = 0, \ldots, N-1.
\end{align*}
\] (3a) (3b) (3c) (3d) (3e) (3f)

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 \( w_{\text{max}} \) such that the above problem is an NLP of the following form:

\[
\begin{align*}
\text{minimize} & \quad w \in \mathbb{R}^{152} \\
& \quad w^T H w \\
\text{subject to} & \quad G(w) = 0, \\
& \quad -w_{\text{max}} \leq w \leq w_{\text{max}}.
\end{align*}
\] (4a) (4b) (4c)

Define what \( H_x \) and \( H_u \) need to be in the Hessian

\[
H = \begin{bmatrix} H_x & H_u \\ \vdots & \ddots \end{bmatrix}, \quad H_x = \begin{bmatrix} & \end{bmatrix}, \quad H_u = \begin{bmatrix} \end{bmatrix}.
\]

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

\[
G(w) = \begin{bmatrix} \vdots \end{bmatrix}, \quad w_{\text{max}} = \begin{bmatrix} \end{bmatrix}
\]

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

3. Check if your function \( G(w) \) does what you want by writing a forward simulation function \([w]=\text{simulate}(x_0,U)\) 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 \( x_0 = (0.5, 0) \) 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:

\[
\begin{align*}
\text{minimize} & \quad w^T H w \\
\text{subject to} & \quad G(\bar{w}) + J_G(\bar{w})(w - \bar{w}) = 0, \\
& \quad -w_{\max} \leq w \leq w_{\max}.
\end{align*}
\]

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

\[
J_G(w) = \begin{bmatrix}
\ddots & \ddots & \ddots \\
\vdots & \ddots & \ddots \\
\end{bmatrix}
\]

5. Today we compute the Jacobian \( J_G(w) \) just by finite differences, perturbing all 152 directions one after the other, using a step size \( \delta = 10^{-4} \). 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 this matrix, e.g. using the command \texttt{spy(J)}.

6. Now learn how to use the MATLAB QP solver \texttt{quadprog} e.g. by calling \texttt{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 (5) using \texttt{quadprog}. Note that the QP is a very sparse QP but that this sparsity is not exploited in full during the call of \texttt{quadprog}.

8. Write a loop around your function \texttt{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?