Optimization and Optimal Control (WS 22/23) Computer Lab 3

Prof. Dr.–Ing. habil. Thomas Meurer, Chair of Automatic Control

Exercise 1 (Optimal control). Consider the pole on a cart system depicted in Fig. 3.1.

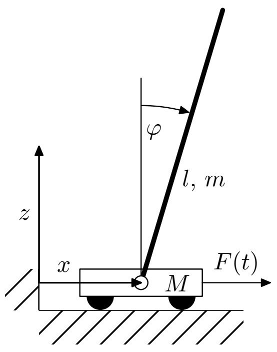

Figure 3.1: Single pole on a cart.

The system states $\boldsymbol{x} = [x_1,\ x_2,\ x_3,\ x_4]^\top = [s,\ \dot{s},\ \theta,\ \dot{\theta}]^\top$ describe the cart position and velocity w.r.t. the x axis, and angle and angular velocity of the pole w.r.t. the z axis. Note that $\theta = \varphi - \pi$. In the following, consider the energy-optimal "swing-up" optimal control problem

$$\min_u J(u) = \int_0^{t_\mathsf{f}} \frac{1}{2} u^2(t)\mathrm{d}t \tag{3.1a}$$

s.t.

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u), \quad \boldsymbol{x}(0) = \hat{\boldsymbol{x}}_0, \quad \boldsymbol{x}(t_\mathsf{f}) = \boldsymbol{x}_\mathsf{f}, \tag{3.1b}$$

where $\boldsymbol{x} \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, and

$$\hat{\boldsymbol{x}}_0 = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}^\top,$$
$$\boldsymbol{x}_\mathsf{f} = \begin{bmatrix}0 & 0 & \pi & 0\end{bmatrix}^\top,$$
$$\boldsymbol{f}(\boldsymbol{x}, u) = \begin{bmatrix} x_2 \\ -\dfrac{\left(l^3 m^2 + Jlm\right)\sin(x_3) {x_4}^2 + g\, l^2 m^2 \cos(x_3)\sin(x_3) + \left(-b\, l^2 m - Jb\right) x_2 + \left(l^2 m + J\right) u}{l^2 m^2 \cos(x_3)^2 - l^2 m^2 + (-M\, l^2 - J)m - JM} \\ x_4 \\ \dfrac{l^2 m^2 \cos(x_3)\sin(x_3){x_4}^2 + \left(gl\, m^2 + Mglm\right)\sin(x_3) + (lmu - blm\, x_2)\cos(x_3)}{l^2 m^2 \cos(x_3)^2 - l^2 m^2 + (-M\, l^2 - J)m - JM} \end{bmatrix},$$

and $l(\boldsymbol{x}, u) = \frac{1}{2}u^2(t)$ is the Lagrangian density. Solve (3.1) for $t_\mathsf{f} = 1\,\mathrm{s}$ using a *direct simultaneous method*, namely, *direct multiple shooting* see, e.g., [3], by discretizing the problem (3.1) using a trapezoidal scheme, i.e.,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{t_\mathsf{f}}{2N}\Big[\boldsymbol{f}(\boldsymbol{x}_k, u_k) + \boldsymbol{f}(\boldsymbol{x}_{k+1}, u_{k+1})\Big], \tag{3.2}$$

and assuming that $u(t)$ is a piece-wise constant function for each time step $\frac{t_\mathsf{f}}{N}$ which leads to a nonlinear equality-constrained static program (NLP)

$$\min_{\boldsymbol{w}} c(\boldsymbol{w}) \tag{3.3}$$

s.t.

$$\boldsymbol{g}(\boldsymbol{w}) = \begin{bmatrix} \boldsymbol{g}^{\mathsf{bc}}(\boldsymbol{w}) \\ \boldsymbol{g}^{\mathsf{ode}}(\boldsymbol{w}) \end{bmatrix} = \boldsymbol{0} \tag{3.4}$$

where $\boldsymbol{w} = [\boldsymbol{w}_x^\top,\, \boldsymbol{w}_u^\top]^\top = [\boldsymbol{x}_0^\top,\, \dots,\, \boldsymbol{x}_N^\top,\, u_0,\, \dots,\, u_N]^\top$ are the decision variables and $N = 50$ is the number of discretization steps,

$$\boldsymbol{g}^{\mathsf{bc}}(\boldsymbol{w}) = \begin{bmatrix} \boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0 \\ \boldsymbol{x}_N - \boldsymbol{x}_\mathsf{f} \end{bmatrix} \tag{3.5}$$

is the vector of boundary constraints, and

$$\boldsymbol{g}^{\mathsf{ode}}(\boldsymbol{w}) = \begin{bmatrix} \boldsymbol{g}_0^{\mathsf{ode}}(\boldsymbol{x}_1, \boldsymbol{x}_0, u_0, u_1) \\ \vdots \\ \boldsymbol{g}_{N-1}^{\mathsf{ode}}(\boldsymbol{x}_N, \boldsymbol{x}_{N-1}, u_{N-1}, u_N) \end{bmatrix}, \tag{3.6}$$

is the vector of ODE (defect) constraints, where

$$\boldsymbol{g}_k^{\mathsf{ode}}(\cdot) = -\boldsymbol{x}_{k+1} + \boldsymbol{x}_k + \frac{t_\mathsf{f}}{2N}\Big[\boldsymbol{f}(\boldsymbol{x}_k, u_k) + \boldsymbol{f}(\boldsymbol{x}_{k+1}, u_{k+1})\Big], \qquad k = 0, \dots, N-1 \tag{3.7}$$

realizes the trapezoidal integration of the ODE. Since this leads to an equality-constrained large-scale NLP, we will use `fmincon` to obtain a solution, which offers a variety of different algorithms such as a global SQP, interior-point, active-set, or trust-region method. Note that although `fmincon's` different algorithms require gradient information, we will dispense the calculation so that `fmincon` will approximate them numerically (thereby largely increasing computation times and function evaluations). In the following use the template provided in [1] and

1. determine the total number of decision variables $n_w = n_{w_x} + n_{w_u}$, where $\boldsymbol{w}_x \in \mathbb{R}^{n_{wx}}$ and $\boldsymbol{w}_u \in \mathbb{R}^{n_{wu}}$ in terms of n_x, n_u, and N.

2. determine the total number of equality constraints. Which dimension does $\nabla_w \boldsymbol{g}(\boldsymbol{w})$ have?

3. provide an initial guess $\boldsymbol{w}_0$ to `fmincon`. The performance of any direct method is highly dependent on a good initial guess. Therefore, given the input $u(t) = a_0 \cos(2\pi f_0 t), \quad t \in [0, t_\mathsf{f}]$, integrate the ODE $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u), \quad \boldsymbol{x}(0) = \hat{\boldsymbol{x}}_0$ for $a_0 = 10\,\mathsf{N}$, $f_0 = 1.5\,\mathsf{Hz}$ numerically using (3.2) in order to obtain $\boldsymbol{w}_x$. Therefore,

   - implement a function `x_=odeTrapz(odefun_,tspan_,N_,x0_)` that integrates an ODE numerically, where `odeFun_` is a function handle to the right-hand-side of the ODE, `tspan_` gives the start and end-time (as a 2D vector), `N_` is the number of discretization steps, and `x0_` is the initial state $\hat{\boldsymbol{x}}_0$. The output argument `x_` should have N rows and n_x columns (thus mimicking an `ode45` call).

   - implement yet another function `x_=newton(fFun_,x0_,dfdxFun_)` that solves a zero-finding problem $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$, where `fFun_` is a function handle that evaluates $\boldsymbol{F}(\boldsymbol{x})$, `x0_` is the initial guess for the solution, `dfdxFun_` provides the Jacobian of $\boldsymbol{F}(\boldsymbol{x})$, and the output `x_` contains the approximate solution of the zero-finding problem. This is

needed since the trapezoidal scheme is an implicit method, for which a nonlinear system of equations with $\boldsymbol{F}(\boldsymbol{x}_{k+1}) := \boldsymbol{g}_k^{\text{ode}}(\cdot) = \boldsymbol{0}$ must be solved at each k in order to determine $\boldsymbol{x}_{k+1}$.
Hint: implement the Newton method in such a way that `dfdxFun_` is an optional argument (using `nargin`). Therefore, if no Jacobian information is passed as an argument, a function `dfdx_=finiteDifferencesJacobian(fFun_,x_)` is needed that calculates the Jacobian numerically, see, e.g. [2].
Note: in principle you could also modify your SQP code from Computer Lab 2 since it implements a Newton-type SQP method.

4. given that the integral term in (3.1) will also be evaluated using the trapezoidal scheme, show that the discretized integral term can be simplified as the cost function

$$c(\boldsymbol{w}) = \frac{1}{2}(u_0^2 + u_N^2) + \sum_{k=1}^{N-1} u_k^2. \tag{3.8}$$

without changing the optimal $\boldsymbol{w}^*$ and implement it in `cost` which is passed as the cost function to `fmincon`.
Hint: you can use `reshape` along with the dimensions obtained in (i) in order to extract $\boldsymbol{w}_u$ from the function's argument `w_` (which contains all decision variables) so that you can calculate $c(\boldsymbol{w})$ easily.

5. implement the equality constraints (3.4) in `nonlcon` which is passed to `fmincon` in [1].
Hint: you can use `reshape` along with the dimensions obtained in (i) in order to extract $\boldsymbol{w}_x$ and $\boldsymbol{w}_u$ from the function's argument `w_` so that you can loop through $\boldsymbol{g}_k^{\text{ode}}(\cdot)$, $k, \dots, N-1$ easily.
Note: within the optimization, you do not need to solve a system of nonlinear equations in order to determine $\boldsymbol{x}_{k+1}$ for implicit integration schemes since the states are part of the optimization problem's decision variables!

6. solve the NLP and simulate the system with the optimal input $\boldsymbol{w}_u^*$ using `ode45` and compare its output with the optimal states $\boldsymbol{w}_x^*$ from the optimization by plotting the states and inputs over time. Are there differences? What can you conclude?
Hint: consult the ODE45 help (an explicit fourth-order Runge Kutta integrator) for time-dependent inputs in order to understand the function handle that you need to pass. Use `plot` to illustrate the states and `stairs` for the input.

(extra) modify the functions `cost` and `nonlcon` so that they also return the gradients of the respective return values. Subsequently, implement the gradient computation of those said quantities. Verify your gradients by setting `fmincon`'s option `CheckGradients` to `true`.
Hint: Consult the fmincon help for nonlinear constraints in order to understand how to pass gradient information to `fmincon`.

(extra) instead of the trapezoidal scheme (3.2), experiment with/implement different numerical integration methods such as the forward Euler, backward Euler, fourth-order Runge-Kutta, Heun, ... How does this affect the function evaluations, computation time, and accuracy of the NLP solution?

(extra) solve (3.1) using an indirect collocation method. Therefore, formulate the Euler-Lagrange-Equations, extract the optimal input $u^* = k(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ from $\nabla_u H(\cdot) = \boldsymbol{0}$, and solve the arising

two-point boundary value problem using `bvp4c` (Matlab).
Hint: Use `wxMaxima` or another CAS of your choice to obtain the expression for u^*.
Note: `bvp4c` is not part of `Octave`.

References

[1] J. Andrej, D. Siebelts, and S. Helling. *oocLab3*. `https://cau-git.rz.uni-kiel.de/ACON/opt/optimization-and-optimal-control`. 2020 (cit. on pp. 2, 3).

[2] B. Chachuat. *Nonlinear and Dynamic Optimization*. `https://www.researchgate.net/publication/37452197_Nonlinear_and_Dynamic_Optimization_From_Theory_to_Practice`. Sec. 5.2.2. 2009 (cit. on p. 3).

[3] T. Meurer. *Optimization and Optimal Control WS 20/21*. `https://www.control.tf.uni-kiel.de/en/teaching/winter-term/optimization-and-optimal-control/fileadmin/opt_ws2021_full`. 2020 (cit. on p. 1).