AWESCO Winter School on Numerical Optimal Control with DAE - University of Freiburg

Exercise 7: Direct Single Shooting and Direct Multiple Shooting

Joel Andersson, Andrea Zanelli, Gianluca Frison,
Elena Malz, Joris Gillis, Sebastien Gros, Moritz Diehl

Consider the following simple OCP for controlling a Van-der-Pol oscillator:

$$\begin{aligned} \underset{x,u}{\text{minimize}} \quad & \int_0^T x_1(t)^2 + x_2(t)^2 + u(t)^2 \, dt \\ \text{subject to} \quad & \dot{x}_1 = (1 - x_2^2)\, x_1 - x_2 + u, \qquad x_1(0) = 0 \\ & \dot{x}_2 = x_1, \qquad\qquad\qquad\qquad x_2(0) = 1 \\ & -1 \leq u(t) \leq 1, \end{aligned} \tag{1}$$

where $T = 10$.

Tasks:

7.1 Implement a CasADi `Function` $f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2 \times \mathbb{R}$ that takes the states x and input u and returns the ODE right-hand-side $\dot{x}$ and the Lagrange objective term L. Anticipating the upcoming simplified calling syntax in CasADi, define the following helper function:

```
def easycall(fcn, *args):
  return fcn(args)[0] if len(args)==1 else fcn(args)
```

7.2 Divide the time horizon into $N = 20$ equidistant intervals, $[t_k, t_{k+1}]$, $k = 0, \ldots, N-1$ and assume a constant control u_k on each interval. Then take $M = 4$ steps with a RK4 scheme to define the discrete-time dynamics as a `Function` $F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2 \times \mathbb{R}$.

F should take $x(t_k)$ and u_k and return $x(t_{k+1})$ and $J_k = \int_{t_k}^{t_{k+1}} L(x, u_k)$, the contribution to the objective from interval k. The key rows of the integrator could look like:

```
for j in range(M):
    k1, k1_q = easycall(f, X,             U)
    k2, k2_q = easycall(f, X + DT/2 * k1, U)
    k3, k3_q = easycall(f, X + DT/2 * k2, U)
    k4, k4_q = easycall(f, X + DT   * k3, U)
    X = X + DT/6*(k1   + 2*k2   + 2*k3   + k4)
    Q = Q + DT/6*(k1_q + 2*k2_q + 2*k3_q + k4_q)
```

Evaluate the integrator with $x(t_k) = [0.2, 0.3]$ and $u_k = 0.4$.

7.3 Formulate the direct single shooting NLP and solve it with IPOPT. Construct the NLP variables step-by-step starting with empty list:

```
w = []
lbw = []
ubw = []
```

The key rows of the NLP formulation should look something along the lines of:

```
for k in range(N):
    # New NLP variable for the control
    Uk = MX.sym('U_' + str(k))
    w += [Uk];
    lbw += ...
    ubw += ...

    # Integrate till the end of the interval
    Xk, Jk = easycall(F, Xk, Uk)
    J += Jk
end
```

To plot the results, you can use

```
tgrid = [T/N*k for k in range(N+1)]
import matplotlib.pyplot as plt
plt.figure(1)
plt.clf()
plt.plot(tgrid, x1_opt, '--')
plt.plot(tgrid, x2_opt, '-')
plt.step(tgrid, vertcat((DM.nan(1), u_opt)), '-.')
plt.xlabel('t')
plt.legend(['x1','x2','u'])
plt.grid()
plt.show()
```

7.4 Modify the script to so that it implements the direct multiple shooting method. The control parametrization and the definition of the integrator should remain the same. Introduce NLP variables corresponding to the state for all discrete time points, including $k = 0$.

7.5 Compare the IPOPT output for both scripts. How did the change from direct single shooting to direct multiple shooting influence:

- The number of iterations
- The number of nonzeros in the Jacobian of the constraints
- The number of nonzeros in the Hessian of the Lagrangian

7.6 **Extra**: Introduce the additional constraint $x_1(t) \geq -0.25$. You only need to enforce this path constraint at the end of each control interval. Modify your scripts to solve the modified problem with direct multiple shooting (easy) and direct single shooting (more tricky).