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

Exercise 8: Direct Collocation

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

Collocation, in its most basic sense, refers to a way of solving initial-value problems by approximating the state trajectory with piecewise polynomials. For each step of the integrator, corresponding to an interval of time, we choose the coefficients of these polynomials to ensure that the ODE becomes exactly satisfied at a given set of time points. The time points, in turn, are chosen to get the highest possible acuracy and, possibly, to make sure that the dynamics be satisfied at the beginning and/or end of the time interval. For a discussion on different choices of these *collocation points*, cf. e.g. *Nonlinear Programming* by Biegler (2010). In the following, we will choose the *Legendre points* of order $d = 3$:

$$\tau = [0, 0.112702, 0.500000, 0.887298] \tag{1}$$

where we have assumed that the time interval is $[0, 1]$.

Using these time points, we define a Lagrangian polynomial basis for our polynomials:

$$L_j(\tau) = \prod_{r=0,\, r\neq j}^{d} \frac{\tau - \tau_r}{\tau_j - \tau_r} \tag{2}$$

Note that the Lagrangian basis satisfies:

$$L_j(\tau_r) = \begin{cases} 1, & \text{if } j = r \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Introducing a uniform time grid $t_k = k\,h$, $k = 0, \dots, N$ with the corresponding state values $x_k := x(t_k)$, we can approximate the state trajectory approximation inside each interval $[x_k, x_{k+1}]$ as a linear combination of these basis functions:

$$\tilde{x}_k(t) = \sum_{r=0}^{d} L_r\left(\frac{t - t_k}{h}\right) x_{k,r} \tag{4}$$

By differentiation, we get an approximation of the time derivative at each collocation point:

$$\dot{\tilde{x}}_k(t_{k,j}) = \frac{1}{h} \sum_{r=0}^{d} \dot{L}_r(\tau_j)\, x_{k,r} := \frac{1}{h} \sum_{r=0}^{d} C_{r,j}\, x_{k,r} \tag{5}$$

We can also get an expression for the state at the end of the interval:

$$\tilde{x}_{k+1,0} = \sum_{r=0}^{d} L_r(1)\, x_{k,r} := \sum_{r=0}^{d} D_r\, x_{k,r} \tag{6}$$

We can also integrate our approximation over the interval, giving a formula for *quadratures*:

$$\int_{t_k}^{t_{k+1}} \tilde{x}_k(t)\, dt = h \sum_{r=0}^{d} \int_0^1 L_r(t)\, dt\, x_{k,r} := h \sum_{r=1}^{d} B_r\, x_{k,r} \tag{7}$$

Tasks:

8.1 Download `collocation.m` (MATLAB) or `collocation.py` (Python) from the course website, containing an implementation of the above collocation scheme. Go through the code and make sure you understand it well. Use the code to to reproduce the result from the second task of Exercise 7.

8.2 Replace the RK4 integrator in the direct multiple shooting implementation from Exercise 7 with the above collocation integrator. Make sure that you get the same results as before.

8.3 Instead of letting the rootfinder solve the collocation equations, augment the NLP variable and constraint vectors with additional degrees of freedom corresponding to the state at the collocation points and let the NLP solver also solve the integration problem. For simplicity, only consider a single collocation finite element per control interval (i.e. $M = 1$). Compare the solution time and number of nonzeros in the Jacobian and Hessian matrices with the direct multiple shooting method.

8.4 **Extra**: Form the Jacobian of the constraints and inspect the sparsity pattern using MATLAB's or SciPy's `spy` command. The following is a hint:

```
% MATLAB
J = jacobian(vertcat(g{:}), vertcat(w{:}));
spy(sparse(DM.ones(J.sparsity())));
```

```
% Python
J = jacobian(vertcat(g), vertcat(w))
import matplotlib.pylab as plt
plt.spy(DM.ones(J.sparsity()).sparse())
plt.show()
```

Repeat the same for the Hessian of the Lagrangian function $L(x, \lambda) = J(x) + \lambda^{\mathrm{T}} g(x)$.