Exercises for Lecture Course on Numerical Optimal Control (NOC)
Albert-Ludwigs-Universität Freiburg – Summer Term 2017

Exercise 5: Algorithmic Differentiation

Prof. Dr. Moritz Diehl, Dimitris Kouzoupis, Andrea Zanelli

The aim of this exercise is to gain experience with the two modes of algorithmic differentiation discussed in the class.

1. **Forward and backward algorithmic differentiation:** Consider the following discrete-time dynamical system:

$$x_{k+1} = x_k + h((1 - x_k)x_k + u_k), \tag{1}$$

where $x_k \in \mathbb{R}$ and $u_k \in \mathbb{R}$ are the state and control input of the system respectively and h is a constant parameter (you can think of it as the time step of an explicit Euler integrator). We are interested in simulating the dynamics forward for N steps starting from the initial value $x_0 = \bar{x}_0$ and computing the derivatives of the obtained states with respect to controls:

$$\frac{\partial x_i}{\partial u_{j-1}}, \quad \forall i, j = 1, ..., N. \tag{2}$$

(a) Fix $\bar{x}_0 = 0.5$, $N = 50$, $h = 0.1$ and $u_k = 1$, $\forall k = 0, \ldots, N-1$. Using CasADi, implement the function $\Phi : \mathbb{R}^N \to \mathbb{R}^N$ that maps controls to the obtained state trajectory

$$x = \Phi(u), \tag{3}$$

where x and u denote the vector of stacked states and controls respectively. Define a CasADi function that outputs the Jacobian of x with respect to u

$$x = \frac{\partial \Phi(u)}{\partial u}. \tag{4}$$

You will use the output of this function as a reference for your implementations in the rest of the exercise.

(1 point)

(b) Implement a MATLAB function `forw_AD` that takes as input a vector containing the values for u and returns the derivative $\frac{\partial x_N}{\partial u_{N-1}}$ using forward algorithmic differentiation (AD). Check that the result provided by your implementation is equal to the corresponding entry in the output obtained with CasADi.

(2 points)

(c) Analogously, implement a MATLAB function `back_AD` that takes as input u and returns the derivative $\frac{\partial x_N}{\partial u_{N-1}}$ using backward AD. Check that the result provided by your implementation is equal to the corresponding entry in the output obtained with CasADi.

(2 points)

(d) Implement now a function `J_FAD` that takes as inputs u and a scalar m and, using forward AD, computes the last m rows of the Jacobian $\frac{\partial \Phi(u)}{\partial u}$ containing the derivatives of the last m states in the simulation with respect to the all the controls. Again, validate your results against the reference output. *Hint: notice that, because of the way forward AD builds the evaluation of $\frac{\partial x_i}{\partial u_j}$, the derivatives of previous states $\frac{\partial x_k}{\partial u_j}, \forall k < i$ are already available after having computed $\frac{\partial x_i}{\partial u_j}$.*

(2 points)

(e) Analogously, implement a function `J_BAD` that takes as inputs u and a scalar m and computes the last m rows of the Jacobian $\frac{\partial \Phi(u)}{\partial u}$ using your implementation of backward AD. *Hint: notice that, because of the way backward AD builds the evaluation of* $\frac{\partial x_i}{\partial u_j}$*, the derivatives of* x_i *with respect to previous controls* $\frac{\partial x_i}{\partial u_k}, \forall k < j$ *are already available after having computed* $\frac{\partial x_i}{\partial u_j}$.

(2 points)

(f) Which of the two implementation do you expect to be more performant for small values of m? Which one for high values of m? Why?

(1 point)

(g) Run your implementations for m ranging from 1 to N and measure the execution time using the MATLAB functions `tic` and `toc`. For this simulation choose $h = 0.01$ and $N = 500$. Plot the obtained execution times as a function of m. Do the results validate your considerations from the previous question?

(1 point)

This sheet gives in total 11 points