TEMPO Summer School on Numerical Optimal Control and Embedded Optimization
University of Freiburg, July 27 - August 7, 2015

Exercise 9: Direct Multiple Shooting and Real-Time Iterations with YALMIP

Rien Quirynen Dimitris Kouzoupis Moritz Diehl

http://syscop.de/teaching/numerical-optimal-control/

SQP and Gauss-Newton for direct multiple shooting Using the RK4 integrator from previous exercises with sensitivity propagation, we will implement a simple SQP type method to solve a classical optimal control problem from the NMPC literature [1]. For this aim, we will perform a direct multiple shooting discretization in combination with the use of a Gauss-Newton Hessian approximation.

9.1 Consider the dynamic system described by the ODE

$$\dot{x} = f(x,u), \quad \text{with} \quad f(x,u) = \begin{bmatrix} x_2 + u(\mu + (1-\mu)x_1) \\ x_1 + u(\mu - 4(1-\mu)x_2) \end{bmatrix}$$

and assume $\mu = 0.5$. Write a MATLAB function `ode(t,x,u)` to evaluate the function $f(x,u)$, which will be provided to the RK4 integrator to simulate the system for a time of $T_s = 0.1$ units by using 2 integrator steps. The resulting nonlinear function $\Phi(\cdot)$ then forms the discrete time system $x_{k+1} = \Phi(x_k, u_k)$.

9.2 Now we want to solve the optimal control problem. After performing direct multiple shooting with $N = 15$ time steps of length $T_s = 0.1$, the resulting NLP has the form

$$\begin{aligned} \underset{X,U}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N-1} T_s \left(x_k^\top Q\, x_k + u_k^\top R\, u_k\right) + \frac{1}{2} x_N^\top P\, x_N \\ \text{subject to} \quad & x_0 = [-0.683; -0.864]^\top, \\ & x_{k+1} = \Phi(x_k, u_k), \qquad \forall k = 0,\dots,N-1, \\ & -2 \le u_k \le 2, \qquad \forall k = 0,\dots,N-1, \\ & x_N^\top P x_N \le \alpha. \end{aligned}$$

The parameters in the latter problem will be defined as

$$Q = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad R = \begin{bmatrix} 1.0 \end{bmatrix}, \quad P = \begin{bmatrix} 16.5926 & 11.5926 \\ 11.5926 & 16.5926 \end{bmatrix} \quad \text{and} \quad \alpha = 0.7.$$

Derive on paper the linearization of the nonlinear equality constraints $x_{k+1} = \Phi(x_k, u_k)$ at a trajectory $\bar{X} = (\bar{x}_0, \dots, \bar{x}_N)$ and $\bar{U} = (\bar{u}_0, \dots, \bar{u}_{N-1})$.

9.3 Now use the environment YALMIP to formulate and solve the convex subproblem that results from linearization at a certain trajectory guess $\bar{X}$ and $\bar{U}$. For now, let us **leave out** the terminal constraint $x_N^\top P x_N \le \alpha$ such that the resulting subproblem is a QP. Note that the objective function can be taken directly from the above OCP formulation in case of a Gauss-Newton approximation of the Hessian of the Lagrangian, since the objective is already a quadratic function. For this and the following tasks, you can start from the provided template code in `GN_SQP.m`. TIP: The following commands create an appropriate cell array of sdpvar objects for the states as well as the controls:

```
    u = sdpvar(nu*ones(1,N),ones(1,N));
    x = sdpvar(nx*ones(1,N+1),ones(1,N+1));
```

This way, it is for example easy to access the states `x{k}` and controls `u{k}` at a specific node k.

9.4 Write the full SQP algorithm, i.e. update the linearization point in each iteration with the most current solution guess. Note that most constraints do not change in each SQP iteration, so you could restructure the code to only update the linearizations of the nonlinear (dynamic) constraints within the loop. Decide on a suitable stopping criterion or simply perform a fixed number of SQP iterations. Plot the solution trajectory in each iteration.

9.5 We will now close the Nonlinear MPC loop by applying the first control input u_0 to the "real" system, which can be simulated again for the duration of T_s using the RK4 integrator (with possibly a different amount of steps). Let us therefore write an outer loop around the previous algorithm, which in each iteration can now start from the previous solution trajectories. At this point, we already trust the SQP method which we wrote just now so we are more interested in the closed-loop state trajectories and the applied control inputs, instead of the individual SQP iterations. You should therefore plot and try to interpret the closed-loop behavior. Does the NMPC scheme indeed manage to control the system to the origin?

9.6 As a final step, we will implement an online variant of the latter SQP type algorithm which is known as the Real-Time Iteration (RTI) scheme [2]. Due to the facts that we already implemented the direct multiple shooting method with Gauss-Newton Hessian and that the *online data vector* x_0 enters the optimization problem formulation linearly, this only requires the following simple change to our existing code:

- we will perform only one SQP iteration per time step, i.e. we change the problem specific *online data* x_0 in each iteration

We will first implement this "RTI without shift" (for another variant, see the first extra question). Plot the closed-loop state trajectories and the control inputs applied by the RTI scheme and compare them with the ideal (fully converged) NMPC scheme. Is there a noticeable difference? If yes, the following points are known to be important in order to make the RTI scheme work well:

(a) choose a good *first initialization* for the state and control trajectory when you start the real-time iterations

(b) sample your system *fast enough* and make sure that the optimization problem data do not make very large jumps from one problem to the next

(c) perform a good *transition* from one time step to the next. So far, we used "no shift", but alternatively, we can use a *shift transition*, which is important for time varying systems

9.7 **Extra 1 (shift transition)**: Additionally, after each RTI, *shift* the current state and control trajectories as an initial guess for the solution in the next time step.[1] Implement the RTI with shift and compare to the solution without shift.

9.8 **Extra 2 (sequential convex programming)**: As a second extension to the RTI scheme, you can implement a version of Sequential Convex Programming (SCP) by keeping the terminal constraint $x_N^\top P x_N \leq \alpha$ in the convex subproblem. Thus, a Quadratically Constrained Quadratic Program (QCQP) needs to be solved in each iteration of this SCP-type algorithm, which is easily implemented with YALMIP.

References

[1] H. Chen and F. Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1218, 1998.

[2] M. Diehl, H.G. Bock, J.P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[1]The last node is just doubled by the shift. Alternatively, you could additionally initialize the last node using a call to the RK4 integrator: `input.x = X(:,N); input.u = U(:,N); output = RK4_integrator(@ode, input); X(:,N+1) = output.value;`