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

Exercise 7: Interior Point Methods and FORCES Pro

Alexander Domahidi Juan Jerez

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

Preliminaries

1. In this problem set you will create controllers implementing the following MPC problem:

$$u^\star(x) = \arg\min_{\mathbf{u}} \quad \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + x_N^\top P x_N \tag{1a}$$

$$\text{s.t.} \quad u_{\min} \le u_k \le u_{\max} \tag{1b}$$

$$x_{k+1} = A x_k + B u_k \tag{1c}$$

$$x_{\min} \le x_k \le x_{\max} \tag{1d}$$

where

$$A = \begin{bmatrix} 0.7115 & -0.4345 \\ 0.4345 & 0.8853 \end{bmatrix}, \quad B = \begin{bmatrix} 0.2173 \\ 0.0573 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 0 \\ 6 \end{bmatrix},$$

$$u_{\min} = -5, \quad u_{\max} = 5, \quad x_{\min} = \begin{bmatrix} -\infty \\ -\infty \end{bmatrix}, \quad x_{\max} = \begin{bmatrix} \infty \\ \infty \end{bmatrix},$$

$$N = 10, \quad Q = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, \quad R = 1, \quad P = P_{LQR},$$

and in case of state constraints

$$x_{\min,2} = 0 \ . \tag{2}$$

Subsequently, you will add reference tracking and time-varying dynamics.

2. Installing FORCES Pro: In this exercise you will be using the tool FORCES Pro to design controllers and to generate the underlying C code (in form of a customized library). FORCES Pro is a web service, so you will need a client to use it. To install the client:
   (a) Register as a new user on `https://www.embotech.com/Register`, or login at `https://www.embotech.com/Login` if you are a registered user.
   (b) Navigate to the `https://www.embotech.com/FORCES-Pro/Download` and click "Download FORCES Pro". A pop-up window appears. On the bottom of the page you will find the Download button.
   (c) Save the zip file in a convenient location.
   (d) Go to that directory in Matlab and unpack the archive by double clicking or by typing in the command window: `unzip FORCES_PRO_v1_3.zip`
   (e) Add FORCES PRO to the Matlab path, e.g. by right click to `FORCES` → Add to Path → Selected Folders.
   (f) Download the example at `https://www.embotech.com/FORCES-Pro/How-to-use/MATLAB-Interface/Simple-MPC-Example`, and run the file to make sure everything is correctly installed.

Graphical Optimal Control Design in Simulink (2012b or higher) In this part of the exercise, you will create the controller by using the Simulink block of FORCES Pro.

1. Basic regulation controller implementing (1):

(a) Open the Simulink model `myFirstController_sim.slx`. It is configured such that it automatically loads the problem data (the A, B, C matrix of the system).

Note: Whenever you press Start, the model will execute the FORCES Pro client function `configure_block()`, which looks for FORCES Pro blocks in the model and updates them accordingly.

(b) From your FORCES Pro installation folder, open the Simulink model `MPC_lib_2012b.mdl`. Drag&drop the "MPC" block into your model. You might want to rename it at this point.

(c) Double click on the block to open its configuration dialog. You see different panes. Go through them and configure your controller to implement (1). Note that the matrices A, B and $C = I_2$ are present in your workspace. Do not bother with the "Estimator" and "Solver options" panes at this point. If you are done configuring, click OK, and press Start. FORCES Pro will now generate a controller, and the block will change its in- and output ports.

(d) Connect the ports - state feedback from the system to the controller, and output of the controller to the system input - and run a closed-loop simulation. Verify by using the scopes that the system is regulated to zero.

(e) Add nonnegativity constraints on the second state: $x_{2,k} \geq 0 \; \forall k$. Use soft constraints to ensure that the problem is always feasible.

Note: You can change the starting point for the simulation by double clicking the state-space system block and changing the *Initial condition* property.

2. Getting solver information: Reopen the configuration dialog, and navigate to the "Settings" pane. Tick the "Get solve information" checkbox, and close the dialog by clicking OK. Press Start to update the block. You will see 2 new output ports: "exitflag" and "solverinfo". Connect the latter to the corresponding channel. Press Start again. The simulation now outputs the solvetimes and the number of conducted iterations in the workspace variable `solveinfo`. Obtain this information from the Simulink scope display. Report the displayed maximum solvetime and iterations.

   Solvetime: ☐ Iterations: ☐

3. Reference tracking controller: create a controller such that state 2 tracks a given reference.

   (a) Re-open the configuration dialog of the Simulink block and navigate to the "Control objectives" pane. Activate the "Reference tracking" checkbox, and make sure to select "Provide state and input reference" in the drop-down list. Click OK and press Start. After code generation, the ports of your block will have been changed.

   (b) Connect the output ports of the "Reference calculator" block with the respective input ports of the controller. For your convenience, attach them also to scopes to see how the references are changing.

   (c) Run the simulation and observe the behavior of the closed-loop system. You can also attach a slider gain to the input port of the reference calculator adjust the reference signal during simulation.

4. **Extra:** Create a controller with the lower bound on state 2 marked as a parameter, i.e. it can be changed while the simulation is running. Connect a slider to the newly created input port of the FORCES Pro controller, and observe the input and output trajectories when changing the lower bound.

FORCES Pro Matlab interface For the remainder of the exercise, we will be using the Matlab API of FORCES Pro to create controller (1). (You can also use the Python interface, but the templates provided for system simulation are in Matlab).

1. Basic regulation controller:

   (a) Open the file `myforcespro.m`. This script generates a controller for the case with $P := Q$ and no state constraints. In section "FORCES multistage form", locate the definitions for the final MPC stage and adjust accordingly to use P_{LQR} as the terminal cost.

   Hint: Have a look at `https://www.embotech.com/FORCES-Pro/How-to-use/MATLAB-Interface` to familiarize yourself with the syntax.

   (b) Test your controller described in `myforcespro.m` by running the script `testmyforcespro.m`. This will call your controller for a test problem. You should get the solution $u_1^\star = 1.6959$ within 9 iterations.

   (c) In the file `mympc.m` in `case 'myforcespro'`, add code to call your controller.

   The file `simCLoop.m` provides a closed-loop simulation. Run it once for solver `'quadprog'` and once for solver `'myforcespro'` (see section "Solver Selection"). Compare the plots. The print output `max_time` is the maximum solver time over all but the first time step, and `max_iter` the maximum number of conducted iterations over all but the first time step (see code). Change the solver options in `myforcespro.m` to use the ADMM solver (you will learn about ADMM in the coming week) and repeat the procedure. See `https://www.embotech.com/FORCES-Pro/How-to-use/MATLAB-Interface/Solver-Options`. Report the times and iterations.

Input constrained problem	Solvetime	Iterations
`'quadprog'`		-
`'myforcespro'` PDIP		
`'myforcespro'` ADMM		

2. Add nonnegativity constraints on the second state (i.e. $x_{2,k} \geq 0 \; \forall k$) to your controller.

   (a) Reopen the file `myforcespro.m` to add a lower bound for the second state. You will need to adapt the

      i. number of bounds (`stages(i).dims.l, stages(i).dim.u`),

      ii. indices of the stage variables that are bounded (`stages(i).ineq.b.lbidx, stages(i).ineq.b.ubidx`), and

      iii. values for the bounds at those indices (`stages(i).ineq.b.lb, stages(i).ineq.b.ub`).

   (b) Test your controller again. This time run the script `testmyforcespro2.m` (note the '2'!). You should get the solution $u_1^* = 2.0777$ within 18 iterations for the primal-dual interior point method.

   (c) Run the closed loop simulation in `simCLoop.m` and report the times and iterations. Make sure you uncomment the state constraint in Line 25.

State constraints	Solvetime	Iterations
`'quadprog'`		-
`'myforcespro'` PDIP		
`'myforcespro'` ADMM		

   (d) **Extra:** What happens to the closed-loop behaviour when you limit the number of ADMM iterations to 30?

   (e) **Extra:** Repeat these tests by tuning ADMM well - a good choice of the algorithm parameter Rho is $\rho = 30$. In `myforcespro.m`, use the following code to set this parameter, and repeat the experiments. How does the solver behave now?

```
1 codeoptions.ADMMrho = 30;
```

3. Quadratic constraints: Add the quadratic terminal constraint $x_N^T P_{LQR} x_N \leq 1$ to your controller. Use the primal-dual interior-point ('PDIP') method as a solver.

   Hint: See `https://www.embotech.com/FORCES-Pro/How-to-use/MATLAB-Interface/Quadratic-Constraints`

4. Time-varying dynamics: when solving nonlinear MPC problems by SQP methods (which you will do tomorrow), A and B (among other matrices) in (1) are changing at each sampling time. To create a controller that allows for varying dynamics,

   (a) Identify which matrices in the FORCES formulation contain the dynamics A and B.

   (b) Use the FORCES Pro client function `help newParam` to define these matrices as parameters.

   Hint: Type `help newParam` to learn how to define matrices and vectors as parameters.

   (c) Change the closed-loop simulation to load in the matrices A, B at each solve time. You can now change the dynamics before each QP solve.

   Hint: See also `https://www.embotech.com/FORCES-Pro/Examples/HOW-TO-Time-Varying-MPC` for documentation.

   Do you experience a performance penalty for one of the solvers? Give an explanation.

Parametric dynamics	Solvetime	Iterations
`'myforcespro'` PDIP		
`'myforcespro'` ADMM		

5. **Extra:** Make use of parameters to implement a reference tracking controller (hint: the linear term of the cost becomes a parameter - why?).

Your own optimization routine based on the Primal Barrier Interior-Point Method Your goal is to implement the barrier method to solve the quadratic program:

$$\begin{aligned} \min_z \quad & \frac{1}{2} z^T H z + q^T z \\ \text{s.t.} \quad & Gz \leq d \end{aligned}$$

The barrier method in pseudo-code:

1: **repeat** {*Outer loop*}
2: ***repeat*** {*Inner centering loop*}
3: *Compute search direction* Δz

$$\left(H + \kappa \sum_{i=1}^{m} \frac{1}{(d_i - g_i z)^2} g_i^T g_i \right) \Delta z = -Hz - q - \kappa \sum_{i=1}^{m} \frac{1}{d_i - g_i z} g_i^T$$

4: *Line search: Choose* $0 \leq t \leq 1$
5: *Update:* $z := z + t\Delta z$
6: ***until*** *stopping criterion is satisfied*
7: *Decrease barrier parameter:* $\kappa := \kappa / \mu$
8: ***until*** $\kappa < \epsilon$

1. Fill in the missing code in file `barrieripm.m` to compute the search direction as described in the above pseudo-code. Note that g_i is a *row vector* of G.

2. Install MPT and other tools as described in the following. They are needed by `barrieripm.m` to make the plot (see 3).

   (a) Create a directory `tbxmanager` and go to that directory in Matlab. Then, run the following code in your Matlab command window to install the tool `tbxmanager`:

   ```
   urlwrite('http://www.tbxmanager.com/tbxmanager.m', 'tbxmanager.m');
   tbxmanager
   ```

   (b) To install MPT and the other needed tools via `tbxmanager`, run:

   ```
   tbxmanager install mpt cddmex yalmip
   mpt_init
   ```

3. Run `barrieripm.m` and you should see something like the figure below (you get a random problem each time, so it will be a little different).

   - The black dots are the iterates of the inner loop (the centering step). Note the first phase where the solution moves from the initial point (0) to the central path (the analytic center is the point on the central path as κ goes to infinity).
   - The red circles are the outer iterates, and are on the central path.

4. Sensitivity.

   (a) In `barrieripm.m`, change line 12 to read `speed = 'fast';`. This changes the tuning parameters (μ, β, α, etc) to 'normal' values. The algorithm will now converge so quickly that you won't be able to see the central path. Run the file to see this.

   (b) Your goal in this exercise is to compute the sensitivity of solve-time to the choice of the parameter μ. Generate a plot of μ versus the total number of Newton steps (i.e., the number of times the search direction is computed). Consider the range of μ between `1.1` and `1e4`. The command `logspace` may be useful.

   (c) Change the dimension (line 11) to 50. How does the number of iterations change?