Model Predictive Control
Checkpoints 6, 7 & 8
Date: 26 October 2020

1 Checkpoint 6

This checkpoint is about unconstrained MPC for LTI systems with quadratic cost.

1.1 Question 1

I rely on the relation $u = -Kx + v$ where K and v are the first lines of the matrices $M_2\phi$ and $-M_1h$. M_1 is the left half of the matrix $\begin{bmatrix} 2H & \Psi^T \\ \Psi & 0 \end{bmatrix}^{-1}$, from column 1 to $N(nx + nu)$, and M_2 is the other half of the same matrix. This is just a linear state feedback controller, with the gains already calculated using optimization.

1.2 Question 2

This is done by first initializing the continuous state-space representation `sysC` (A and B as given, $C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and $D = 0$), then discretizing `sysC` with a sampling period $\tau = 0.25s$ using the `c2d` command to obtain the discrete-time system `sysD`, and give the matrices `sysD.A` and `sysD.B` to the feedback+feed-forward function.
The matrices are: $K = \begin{bmatrix} 0.748505598867654 & 1.90606218479175 & 2.04938903897148 \end{bmatrix}$ and $v = 1.49701119773531$.
I would like to **check the stability of the closed-loop system**, using the command `abs(eig(sysD.A-K*sysD.B))` and it gives 0.5386, 0.9167, 0.9167, which means that all the three poles of the close-loop discrete system lie inside the unit circle (magnitude smaller than 1), so it is stable.

1.3 Question 3

While the gains are the same, the state x is different each time. In each iteration, I calculate the new control using the relation above, and then update the state uisng $x_{k+1} = Ax_k + Bu_k$, and repeat this for the next iteration. This is the result of unconstrained MPC:

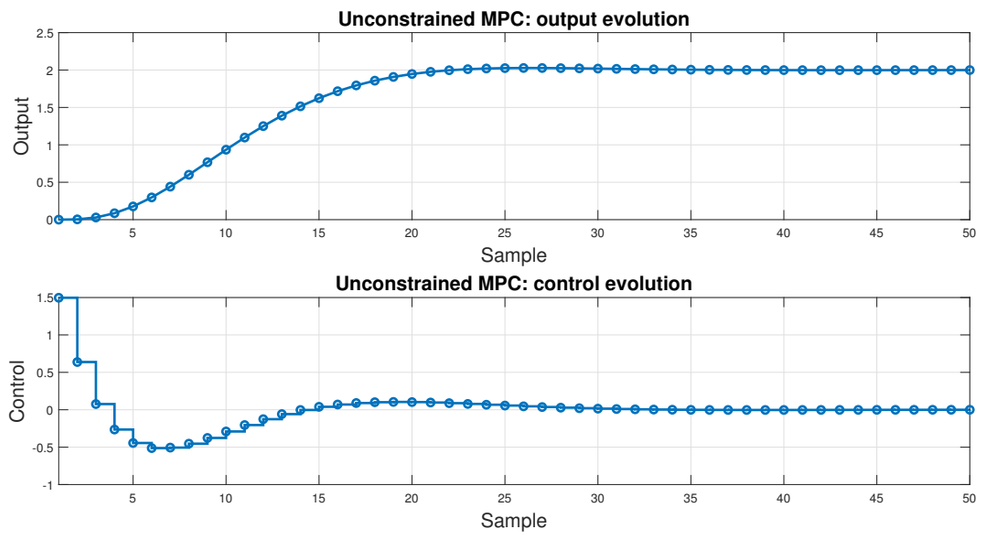

Figure 1: Results of unconstrained MPC.

The output tracks the setpoint $x_d = 2$ after 20 samples (5 seconds) and the control goes to 0, which shows the effectiveness of unconstrained MPC for this problem.

2 Checkpoint 7

This checkpoint is about constructing the inequality constraint matrices for constrained MPC.

2.1 Question 1

Some explanations on the code:

- For this exercise, the limits are considered as constants.
- The `Sk` matrices are constructed using a sub-matrix where the number 1 is run through it, and kronecker product is performed.
- In each iteration, the matrices `Aineq` and `Bineq` (initiated as empty) are stacked with newly calculated values.

2.2 Question 2

Dimensions of matrices: `Gk` is $2(nx+nu)\times(nx+nu)$ and `Sk` is $(nx+nu)\times N(nx+nu)$. So `GkSk` is $2(nx+nu)\times N(nx+nu)$ and it follows that `Aineq` is $2N(nx+nu)\times N(nx+nu)$. Also `dk` is $2(nx+nu)\times 1$ and so `Bineq` is $2N(nx+nu)\times 1$.
Here $nx=4$, $nu=2$, and $N=12$ give `Aineq` being 144×72 and `Bineq` being 144×1.
I verified it with MATLAB and indeed it returned the matrices with exactly those dimensions!

3 Checkpoint 8

This checkpoint is about constrained MPC for LTI systems with quadratic cost. We will see that when there are constraints, optimal controllers like LQR do not work and we need MPC to do this.

3.1 Question 1

This is done using the `quadprog` command in MATLAB **once only**, using the inequality constraints as formed in Checkpoint 7 and the equality constraint $\Phi x_0+\Psi z=0$ where x_0 is a chosen initial condition where I put as $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$. The results are as follows:

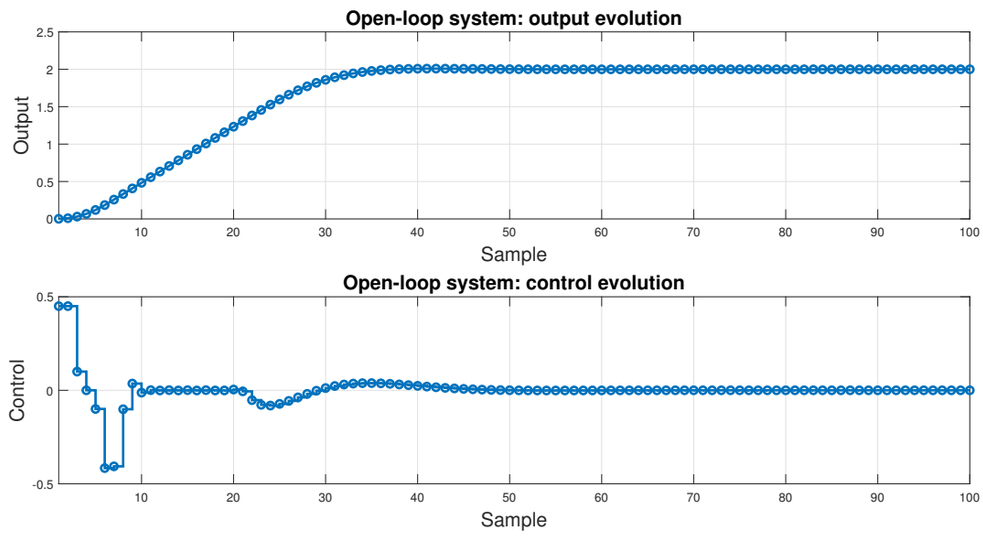

Figure 2: Results of open-looped constrained MPC (prediction horizon = 100 samples).

Remarks and analysis:

- This shows that the open-loop trajectory is stable and the output can track the set-point.
- Note that **this can only be calculated up to the prediction horizon**. Here I extended the horizon to 100 samples to watch the output converge.

3.2 Question 2

This is done by performing constrained optimization iteratively. Here the prediction horizon is set back to $N = 20$ samples. In each iteration, the `quadprog` command is called with the inequality constraints, but **the equality constraint is updated with the current state**: $\Phi x + \Psi z = 0$. The results are as follows:

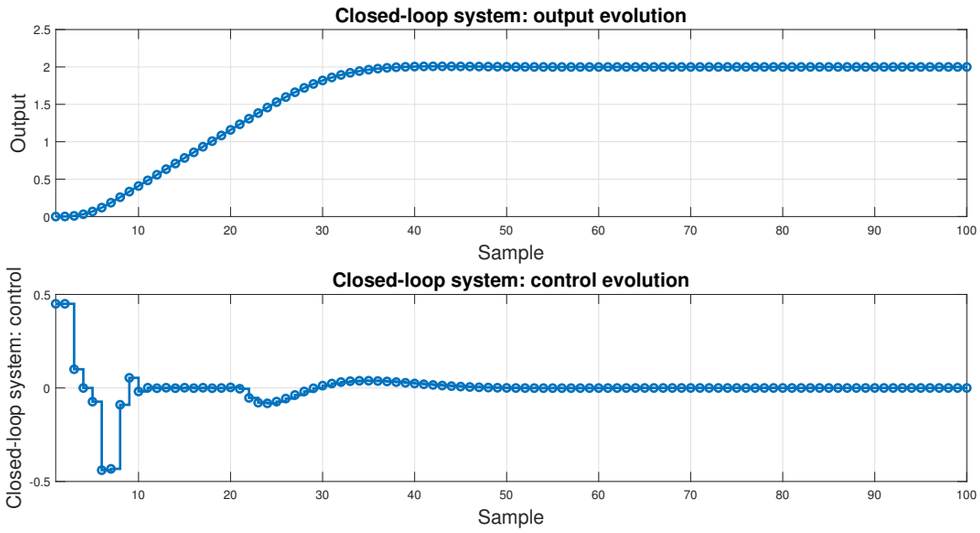

Figure 3: Results of closed-looped constrained MPC (prediction horizon = 20 samples).

Remarks and analysis:

- The control strategy works well and we can track the setpoint.
- Here, we re-optimize with the actual state as constraints to the optimization problem, so this **takes in account how the system behaves after the previous sample** and therefore is adaptive.
- This closed-loop system, unlike the open-loop one, **can be performed beyond the prediction horizon, up to infinity**.
- If, for some particular case, **the inequality constraints can also change** (for example we want to limit energy consumption more and more with time), this can easily be integrated into the problem by just recalling the function for `Aineq` and `Bineq` **inside the loop** with the new values, right before the `quadprog` command. In this exercise with constant constraints, I call it once outside the loop to save computation time.

Now, I would like to compare this strategy under different values of constraint. An outer loop was formed where I put in 0.3, 0.25 and 0.15 as constraints of the second state:

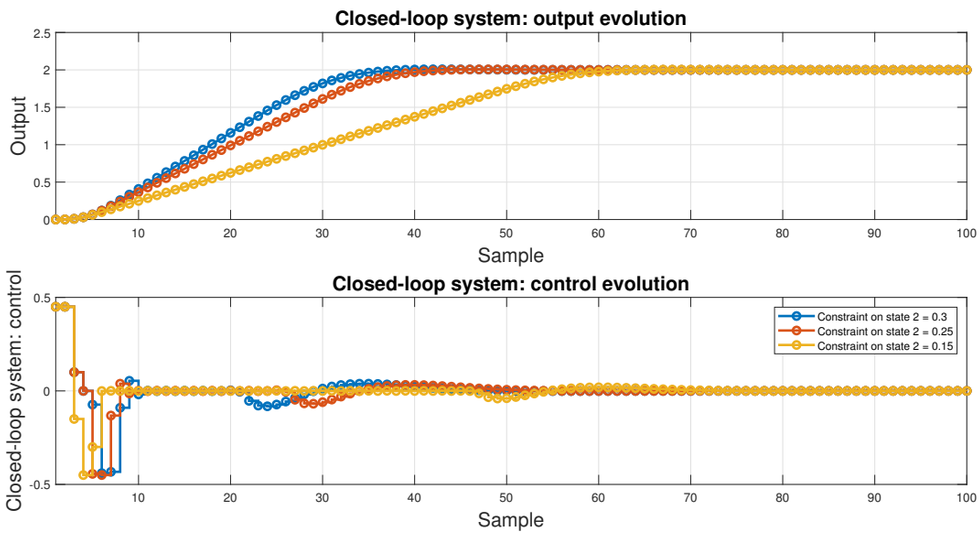

Figure 4: Comparison across cases of constrained MPC with different constraint values.

We see that when one of the states is limited, then the rise time of course must be reduced as well. The system is slower as the first state (output) cannot change too fast (its derivative is the second state is now more limited). This also means that the control input must work harder to raise the output: we see that at the beginning, u of the third case is the largest.

3.3 Extension: What if we do saturated unconstrained MPC instead of constrained MPC?

This part is my experiment to see why when there are constraints, **we cannot just use unconstrained MPC and then let it saturate**. We have to do constrained optimization. To do this, I go back to the unconstrained MPC code and add some saturation in it, using the `min` and `max` functions. Here is the same system where the constraint on u is still 0.45:

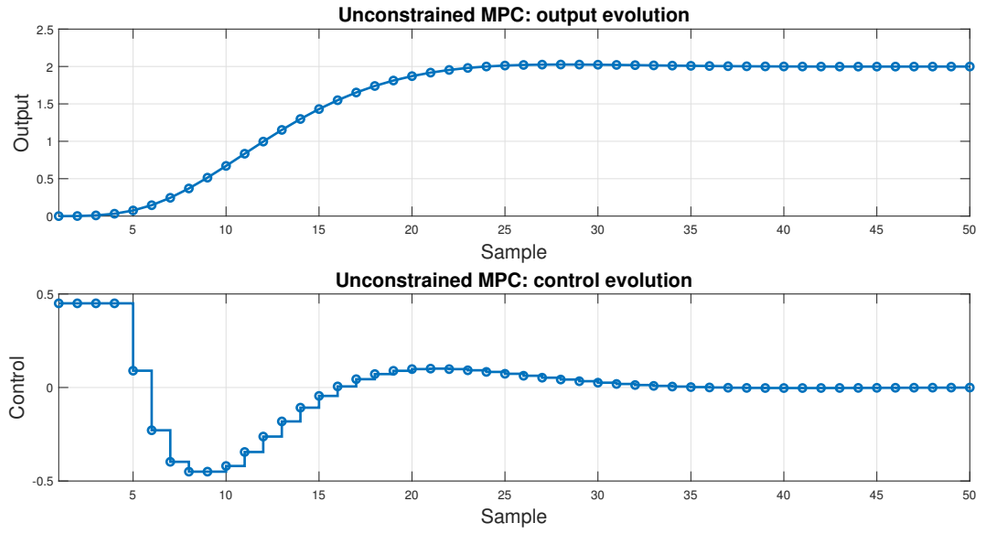

Figure 5: Saturated unconstrained MPC instead of constrained MPC (case 1)

Fortunately, it still works. But that is when we are lucky: the control is not very limited so it can still do the job in this case. But this method is **not guaranteed to work**. This is the result when we have the constraint on u equal 0.1:

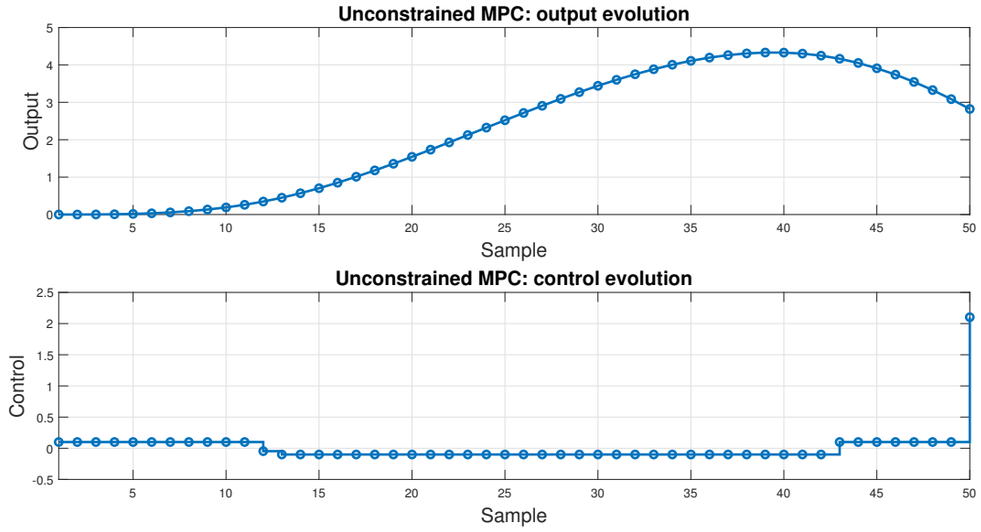

Figure 6: Saturated unconstrained MPC instead of constrained MPC (case 2)

We see that it goes wrong. This controller does not work because the solution given by the unconstrained optimization is not application in the constrained case. We absolutely need constrained MPC here, which is guaranteed to work, as shown in the last part where we have different values of constraint.