WASP Autonomous Systems Course – Hand-in Exercises

Daniel Axehill and colleagues

September 9, 2016

1 Recommended software

- Matlab including Control System Toolbox and Optimization Toolbox
- Yalmip (`http://users.isy.liu.se/johanl/yalmip/`)
- Gurobi (`http://www.gurobi.com/`)
- Lab files: `https://gitlab.ida.liu.se/wasp_cdm/exercises.git`

2 Tasks

1. **Basic LQ control with Kalman filter** In this task, basic LQ control with a Kalman filter will be considered. It is recommended to have a look in Chapter 9 in [3] or [2]. Consider the system

$$G(s) = \frac{1}{s-1}$$

represented on state-space form with noise as

$$\begin{aligned} \dot{x}(t) &= x(t) + u(t) + v_1(t) \\ z(t) &= x(t) \\ y(t) &= x(t) + v_2(t) \end{aligned}$$

The noises $v_i(t)$ are white with intensities R_i. We use the criterion

$$V = \int Q_1 x^2(t) + Q_2 u^2(t)\, dt,$$

and want to find the LQG controller.

   (a) Show that the controller is a function of $\alpha = Q_1/Q_2$ and $\beta = R_1/R_2$ only using the algebraic expressions for the relevant algebraic Riccati equations.

(b) Compute the poles of the closed-loop system as a function of α and β.

(c) Assume $Q_1 = 1$, $Q_2 = 2$, $R_1 = 3$, and $R_2 = 4$. Verify the result of your analytical expressions above in (a) with those you obtain if you use Matlab for this particular choice of penalty matrices and noise intensities. Useful Matlab commands are `lqr` and `kalman`. For more info, the command `help control` lists all commands in Control System Toolbox.

2. **Riccati recursions** In Model Predictive Control (MPC), the optimization problem that is solved at each sample in the control loop is a so-called Constrained Finite-Time Optimal Control (CFTOC) problem. This inequality constrained optimization problem can be solved using various optimization methods, where two common types are Interior-point (IP) methods and Active-set (AS) methods. In both these types of solvers, the main computational effort is often spent on computing the Newton step (i.e. the search direction). This often corresponds to solving a sequence of Unconstrained Finite-Time Optimal Control (UFTOC) problems in the following form:

$$\begin{aligned} \underset{x_t, u_t,\, \forall t}{\text{minimize}} \quad & \sum_{t=0}^{N-1} \left(\frac{1}{2} \begin{bmatrix} x_t^T & u_t^T \end{bmatrix} \begin{bmatrix} Q_{x,t} & Q_{xu,t} \\ Q_{xu,t}^T & Q_{u,t} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} l_{x,t}^T & l_{u,t}^T \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} \right) \\ & + x_N^T Q_{x,N} x_N \\ \text{subject to} \quad & x_0 = \bar{x} \\ & x_{t+1} = A_t x_t + B_t u_t + a_t, \quad t \in \{0, 1, \ldots, N-1\}, \end{aligned} \tag{1}$$

where $x_t \in \mathbb{R}^n$ and $u_t \in \mathbb{R}^{p_t}$ are the optimization variables, $\bar{x} \in \mathbb{R}^n$ is the initial state and

$$\begin{bmatrix} Q_{x,t} & Q_{xu,t} \\ Q_{xu,t}^T & Q_{u,t} \end{bmatrix} \in \mathbb{S}_+^{n+p_t}, \quad Q_{u,t} \in \mathbb{S}_{++}^{p_t}, \quad \text{for} \quad t \in \{0, 1, \ldots, N-1\}. \tag{2}$$

and

$$Q_{x,N} \in \mathbb{S}_+^n. \tag{3}$$

Here $\mathbb{S}_{++}^n (\mathbb{S}_+^n)$ denotes symmetric positive (semi) definite matrices of size n, and "unconstrained" indicates that there are no inequality constraints.

(a) Write down the Karush-Kuhn-Tucker (KKT) system for the UFTOC problem (1). Furthermore, show that it can be written in a tri-block-diagonal form by making a suitable re-arrangement of the equations and variables. In this task, [1] (in particular Section 5.5.3), [6] and [5] might be useful.

(b) Show, using induction, that the primal and dual solutions to the UFTOC problem can be described by affine state feedback functions.

Hint: Find P_t and Ψ_t such that $P_t x_t - \lambda_t = \Psi_t$ holds for all t. Use this relation to show that the feedback matrices can be computed using a backward (in time) factorization and recursion. Finally, use the dynamics equations to compute the solution using a forward (in time) recursion.

3. **Numerical Optimal Control** The following tasks are parts of the course Numerical Methods for Embedded Optimization and Optimal Control held at LiU by Prof. Moritz Diehl (`http://www.vehicular.isy.liu.se/Edu/Courses/NumericalOptimalControl/`).

   (a) **Nonlinear Programming and Single Shooting** The purpose with this task is to solve a simple optimal control problem. The system considered in this task is the controlled harmonic oscillator described by

$$\frac{d}{dt}\begin{bmatrix} p(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t), \quad t \in [0, T]\,. \tag{4}$$

   1. Log in and start MATLAB and open an editor of your choice. Use `help fmincon` to learn what is the syntax of a call of `fmincon`. Alternatively, YALMIP can be used to formulate the optimization problems and call an appropriate solver (e.g., `fmincon`).
   2. We now want to solve an optimal control problem using MATLAB. The aim is to bring a harmonic oscillator to rest with minimal control effort. For this aim we apply Euler integration to the continuous time model in (4) and get the following linear discrete time dynamic system:

$$\begin{bmatrix} p_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} p_k \\ v_k \end{bmatrix} + \Delta t \left(\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} p_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \right), \quad k = 0, \ldots, N-1 \tag{5}$$

      where p and v are the position and velocity of the harmonic oscillator respectively and u is the control signal. Choose the fixed values $p_0 = 10$, $v_0 = 0$, $\Delta t = 0.2$, $N = 50$ and denote for simplicity from now on $x_k = (p_k, v_k)^T$. Now write a MATLAB routine `[xN] = oscisim(U)` that computes the final state x_N as a function of the control inputs $U = (u_0, \ldots, u_{N-1})^T$. Mathematically, we will denote this function by $f_{\text{oscisim}} : \mathbb{R}^N \to \mathbb{R}^2$.
   3. To verify that your routine does what you want, plot the simulated positions $p_0, \ldots, p_N$ within this routine for the input $U = 0$.
   4. Now we want to solve the optimal control problem

$$\underset{U \in \mathbb{R}^N}{\text{minimize}} \quad ||U||_2^2 \qquad \text{subject to} \qquad f_{\text{oscisim}}(U) = 0$$

      Formulate and solve this problem with `fmincon`. Plot the solution vector U as well as the trajectory of the positions in the solution.

5. Now add inequalities to the problem, limiting the inputs u_k in amplitude by an upper bound $|u_k| \leq u_{\max}, k = 0, \ldots, N-1$. This adds $2N$ inequalities to your problem. Which?
6. Formulate the problem with inequalities in `fmincon`. Experiment with different values of $u_{\max}$, starting with big ones and making it smaller. If it is very big, the solution will not be changed at all. At which critical value of $u_{\max}$ does the solution start to change? If it is too small, the problem will become infeasible. At which critical value of $u_{\max}$ does this happen?
7. Both of the above problems are convex, i.e., each local minimum is also a global minimum. Note that the equality constraint of the optimal control problems is just a linear function at the moment. Now, try to make this constraint nonlinear and thus make the problem nonconvex. One way is to add a small nonlinearity into the dynamic system (5) by making the spring nonlinear, i.e., replacing the term -1 in the lower left corner of the system matrix by $-(1 + \mu p_k^2)$ with a small μ, and solving the problem again. At which value of μ does the solver `fmincon` need twice as many iterations as before?

(b) **Runge-Kutta Integrator and Gauss-Newton Algorithm** In this part we compare two methods for numerical integration of ordinary differential equations (ODE) and program our first own optimization code, a Gauss-Newton algorithm.

1. Throughout this part, we consider again the controlled harmonic oscillator in (4). We abbreviate this ODE as $\dot{x} = f(x, u)$ with $x = (p, v)^T$. Let us choose again the fixed initial value $x_0 = (10, 0)^T$ and $T = 10$.
   In the next questions we are interested in comparing the simulation results for $u(t) = 0$ that are obtained by two different integration schemes, namely the Euler integrator from Task 3.a, and a Runge-Kutta integrator. We regard in particular the value $p(10)$, and as the ODE is explicitly solvable, we know it exactly, which is useful for comparisons. What is the analytical expression for $p(10)$?
2. First run again your explicit Euler method from Task 3.a, e.g., again with $N = 50$ integrator steps, i.e. with a stepsize of $\Delta t = 10/50 = 0.2$. The central line in the Euler code reads

$$x_{k+1} = x_k + \Delta t \cdot f(x_k, u_k). \tag{6}$$

   Plot your trajectories $\{(t_k, x_k)\}_0^N$ for $U = 0$.
3. Now exchange in your Euler simulation code the line that generates

the step (6) by the following five lines:

$$\begin{aligned} k_1 &= f(x_k, u_k) \\ k_2 &= f(x_k + \frac{1}{2}\Delta t \cdot k_1, u_k) \\ k_3 &= f(x_k + \frac{1}{2}\Delta t \cdot k_2, u_k) \\ k_4 &= f(x_k + \Delta t \cdot k_3, u_k) \\ x_{k+1} &= x_k + \Delta t \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \end{aligned}$$

This is the classical Runge Kutta method of order four (RK4). Note that each integrator step is four times as expensive as an Euler step. What is the advantage of this extra effort? To get an idea, plot your trajectories $\{(t_k, x_k)\}_0^N$ for the same N number of integrator steps.

4. To make the comparison of Euler and RK4 quantitative, regard the different approximations of $p(10)$ that you obtain for different stepsizes, e.g., $\Delta t = 10^{-k}$ with $k = 0, \ldots, 5$. We call these approximations $\tilde{p}(10; \Delta t)$. Compute the errors $|p(10) - \tilde{p}(10; \Delta t)|$ and plot them doubly logarithmic, i.e., plot $\log_{10}\left(|p(10) - \tilde{p}(10; \Delta t)|\right)$ on the y-axis and $\log_{10} \Delta t$ on the x-axis, for each of the integrators. You should see a line for each integrator. Can you explain the different slopes?
5. We consider again the optimal control problem from Task 3.a. We had previously used the Euler integrator, but here we use our new RK4 integrator because it is more accurate. We do again $N = 50$ integrator steps to obtain the terminal state as a function of the control inputs $U = (u_0, \ldots, u_{N-1})^T$, and denote the function here by $g_{\text{sim}} : \mathbb{R}^N \to \mathbb{R}^2$. The nonlinear program we want to solve is again

$$\underset{U \in \mathbb{R}^N}{\text{minimize}} \quad ||U||_2^2 \qquad \text{subject to} \qquad g_{\text{sim}}(U) = 0.$$

In this task, we do not solve this problem with `fmincon`, but we write our own Newton-type optimization method. To prepare for the next step, make sure you have the routine g_{sim} as a function of the 50 controls with the two terminal states as outputs.

6. Motivating background information (you might skip to the next question): The necessary optimality conditions (KKT conditions, see Section 5.5.3 in [1]) for the above problem are

$$\begin{aligned} 2U^\star + \frac{\partial g_{\text{sim}}}{\partial U}(U^\star)^T \lambda^\star &= 0 \\ g_{\text{sim}}(U^\star) &= 0. \end{aligned}$$

Let us introduce a shorthand for the Jacobian matrix

$$J_{\text{sim}}(U) = \frac{\partial g_{\text{sim}}}{\partial U^\star}(U).$$

By linearization of the constraint at some given iterate (U_k, λ_k) and neglecting its second order derivatives, we get the following (Gauss-Newton) approximation of the KKT conditions:

$$\begin{bmatrix} 2U_k \\ g_{\text{sim}}(U_k) \end{bmatrix} + \begin{bmatrix} 2\mathbb{I} & J_{\text{sim}}(U_k)^T \\ J_{\text{sim}}(U_k) & 0 \end{bmatrix} \begin{bmatrix} U_{k+1} - U_k \\ \lambda_{k+1} \end{bmatrix} = 0.$$

This system can easily be solved by a linear solve in order to obtain a new iterate U_{k+1}. But in order to do this, we first need to compute the Jacobian $J_{\text{sim}}(U)$.

7. Implement a routine that uses finite differences, i.e., calls the function g_{sim} $(N+1)$ times, once at the nominal value and then with each component slightly perturbed by, e.g., $\delta = 10^{-4}$ in the direction of each unit vector e_k, so that we get the approximations

$$\frac{\partial g_{\text{sim}}}{\partial U}(U) \approx \frac{g_{\text{sim}}(U + \delta e_k) - g_{\text{sim}}(U)}{\delta}.$$

We denote the resulting function that gives the full Jacobian matrix of g_{sim} by $J_{\text{sim}} : \mathbb{R}^N \to \mathbb{R}^{2\times N}$.

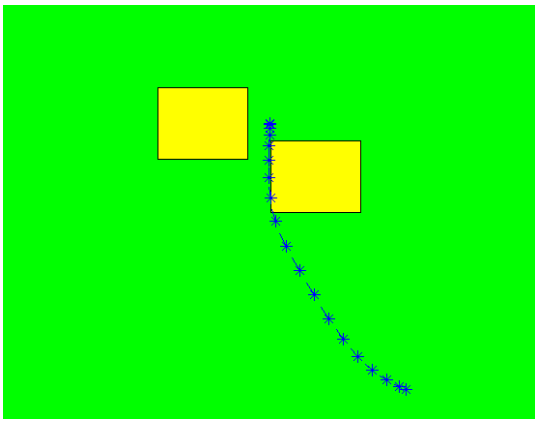
8. Now, we implement the Gauss-Newton scheme from above, but as we are not interested in the multipliers we just implement it as follows:

$$U_{k+1} = U_k - \begin{bmatrix} \mathbb{I} & 0 \end{bmatrix} \begin{bmatrix} 2\mathbb{I} & J_{\text{sim}}(U_k)^T \\ J_{\text{sim}}(U_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2U_k \\ g_{\text{sim}}(U_k) \end{bmatrix}.$$

Choose an initial guess for the controls, e.g., $U = 0$, and start your iteration and stop when $\|U_{k+1} - U_k)\|$ is very small. How many iterations do you need to converge? Do you have an idea why?

4. **Model Predictive Control using YALMIP** In the following exercises, we will progressively build a fairly advanced simulation of a nonconvex model predictive control (MPC) problem using YALMIP. The objective with the task is to park a robot on a predefined spot (the origin), while avoiding certain dangerous areas (the boxes) and staying in the playing field (green area). First, make sure you have downloaded YALMIP and Gurobi. Furthermore, make sure that Gurobi is available in the Matlab path.

   We model the robot as a simple point-mass which can be independently controlled in the up/down and left/right direction. After discretizing the



system, the 4-state 2-input model is given by

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.1 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0.1 & 0 \\ 0.005 & 0 \\ 0 & 0.1 \\ 0 & 0.005 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(k) \end{aligned}$$

The output $y(k)$ is the position of the robot. The robot is only allowed to move in the region $-2 \leq y(k) \leq 2$, and the input is constrained $-1 \leq u(k) \leq 1$.

(a) As a start, open the file `robotlqr`. In this file, the robot is controlled using a linear quadratic controller. No constraints are handled. Run the code and make sure you understand the general layout of the code and simulation

(b) In MPC, we minimize the following finite-horizon performance objective

$$\sum_{k=1}^{N} \ell(x(k), u(k))$$

The minimization is performed under control and state constraints. In this first MPC controller, we use a quadratic performance objective $y(k)^T Q y(k) + u(k)^T R u(k)$, in order to mimic the linear quadratic controller.

Open the file `robotmpcskeleton`. In this file, you have the basics for setting up an MPC problem using YALMIP. The code assumes that you want to solve the problem in implicit form, i.e. optimize over both the states and the inputs, and introduce the dynamical model by adding equality constraints.

(c) Start by changing your controller to use a linear norm in the objective function, such as a 1-norm or ∞-norm (see `help norm` if you are unfamiliar with the norm command). Can you see any conceptual difference in the simulations?

Since we are solving the optimization problem repeatedly during the simulation, we do not want to waste too much time on YALMIP overhead. We can overcome this by using the `optimizer` command. This command allows us to precompile an optimization problem where only some parameters change and these parameters enter the problem affinely. In our case, we can interpret our problem as a model where we would like to find `u{1}` for varying `x{1}`. We thus essentially need the function $u\{1\} = f(x\{1\})$.

Hence, remove the whole `optimize` part and replace it with an computation using an `optimizer` object. You create the `optimizer` object outside the loop. The optimizer creation should take `x{1}` as the fourth argument, and `u{1}` as the fifth. The third input is an options structure (created using `sdpsettings`) but you can leave it empty. Start by reading the help on `optimizer`.

(d) The controller now hopefully actively respects the green playing-field constraints. The robot still passes through the red regions though. Adding so called avoidance constraints will turn the problem much harder, as it is a nonconvex constraint. It can however be modelled using mixed-integer constraints (essentially, either you go left, or you go right). This can be a bit tricky to model, but YALMIP can do a lot of the modelling for you. For instance, if you use linear programming representable norm operators (which are convex), but use them in a nonconvex fashion (norm larger than something), YALMIP will automatically setup the mixed-integer problem that arise.

Try to add avoidance constraints to your model! Since we now solve a much harder problem (and easily make mistakes), it is not unlikely that the optimization problem goes infeasible. Add a check in your code to warn and terminate the simulation if the controller is infeasible.

Now play! Try to come up with other interesting ideas and features. How does the prediction horizon relate to performance? Can you get the robot stuck behind a box? Please note though that once you start allowing nonconvex constructions, you open up the possibility for a lot of problems, both in terms of modelling, and in terms of actually solving the problems.

Literature

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[2] Torkel Glad and Lennart Ljung. *Control Theory – Multivariable and Nonlinear Methods*. Taylor & Francis, 2000.

[3] Torkel Glad and Lennart Ljung. *Reglerteori – Flervariabla och olinjära metoder*. Studentlitteratur, 2003.

[4] Torkel Glad and Lennart Ljung. *Reglerteknik – Grundläggande teori*. Studentlitteratur, 2006.

[5] Isak Nielsen. *On Structure Exploiting Numerical Algorithms for Model Predictive Control*. Licentiate's thesis, Linköping University, 2015. `http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-121711`.

[6] Lieven Vandenberghe, Stephen Boyd, and Mehrdad Nouralishahi. Robust linear programming and optimal control. Technical report, Department of Electrical Engineering, University of California Los Angeles, 2002. `http://web.stanford.edu/~boyd/papers/pdf/robustlp.pdf`.