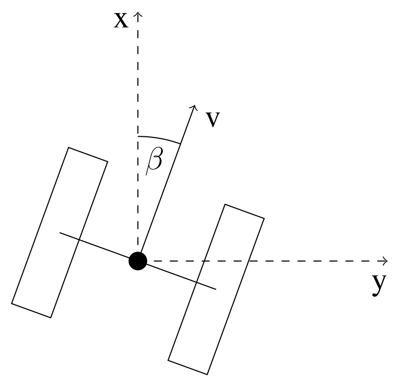
Exercises for Course on Modeling and System Identification (MSI)
Albert-Ludwigs-Universität Freiburg – Winter Term 2018-2019

Exercise 7: Recursive Least Squares

(to be returned on Dez 21, 2018, 10:00 in SR 00-010/014, or before in building 102, 1st floor, 'Anbau')

Prof. Dr. Moritz Diehl, Tobias Schöls, Katrin Baumgärtner, Alexander Petrov

In this exercise you will implement a Recursive Least Squares (RLS) estimator and a forward simulation of a robot. We will apply the RLS algorithm to position data of a 2-DOF moving in the X-Y plane, measured with a sampling time of $0.0159\,\mathrm{s}$. The movement of the robot depends on the angular velocities of the left and the right wheel ω_L and ω_R, as well as on their radii R_L and R_R. Differing radii influence the behaviour of the robot.



The system can be described by a state space model with three internal states. The state vector $\mathbf{x} = [x\ y\ \beta]^\top$ contains the position of the robot in the $X-Y$ plane and the deviation β from its initial orientation. The system can be controlled by the angular velocities of the wheels: $\mathbf{u} = [\omega_L\ \omega_R]^\top$. The output of the system is the position of the robot: $\mathbf{y} = [x\ y]^\top$. The model follows as

$$\dot{\mathbf{x}} = \begin{pmatrix} v \cdot \cos\beta \\ v \cdot \sin\beta \\ \frac{\omega_L R_L - \omega_R R_R}{L} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

with L being the length of the axis between the two wheels and the velocity v being

$$v = \frac{\omega_\mathrm{L} \cdot R_\mathrm{L} + \omega_\mathrm{R} \cdot R_\mathrm{R}}{2}.$$

1. **Recursive Least Squares applied to position data**

   It is your task to implement the RLS algorithm in MATLAB and to tune it with the appropriate *forgetting factors*. As you can see, the model for the position of the robot is nonlinear. To keep it simple, in task (1) we approximate the position data by a fourth order polynomial. You can assume that the noise on the X and Y measurements is independent. The experiment starts at $t = 0\,\mathrm{s}$.

   (a) MATLAB: Fit a 4-th order polynomial through the data using ordinary least-squares. Plot the data and the fit both in the $X - Y$ plane and separately.
   Hint: You need one estimator for each coordinate.
   PAPER: Does the fit seem reasonable? Why do you think that is? (1 point)

(b) MATLAB: Implement the RLS algorithm as described in the script to estimate 4-th order polynomials to fit the data. Do not use forgetting factors yet. Plot the result against the data.
PAPER: Compare the LS estimator from (a) with the RLS estimator you obtain after processing N measurements. Please give an explanation for your observation. (1 point)

(c) MATLAB: Add a forgetting factor α to your algorithm and try different values for α. Plot the results on the same plot as the previous question.
PAPER: How does α influence the fit? What is a reasonable value for α? (1 point)

(d) PAPER: How can you compute the covariance Σ_p of the position, if you know the covariance of the estimator $\Sigma_{\hat{\theta}}$?
Hint: For a random variable $\gamma = A\theta$, where A is a matrix, $\mathrm{cov}(\gamma) = A\mathrm{cov}(\theta)A^{\mathrm{T}}$. (0.5 point)

(e) MATLAB: Compute the *one-step-ahead* prediction at each point (i.e. extrapolate your polynomial fit to the next time step). We also provided code to plot the 1σ confidence ellipsoid around this point, and the data.

PAPER: Do the confidence ellipsoids grow bigger or smaller as you take more measurements? (1.5 points)

2. **Forward simulation of a robot's position**

In this task you will simulate the position of the two-wheel-robot using the state space model.

(a) MATLAB: Given the state space model and the system state $x = [x\ y\ \beta]^\top$, implement a function `[xdot] = robot_ode(x,u,p)` which evaluates the right-hand side of the ODE $\dot{x} = f(x, u, p)$, with parameters $p = [R_L, R_R, L]$. Use the following values: R_L = 0.2 m, R_R = 0.2 m and L = 0.6 m. (1 point)

(b) MATLAB: Implement a function

```
[x_next] = euler_step(deltaT, x0, u, ode, p)
```

which performs one Euler integration step for a general ODE $\dot{x} = f(x, u, p)$ starting at x_0, with input u, parameters p and an integration interval ΔT. (1 point)

(c) MATLAB: Implement a function

```
[x_next] = rk4_step(deltaT, x0, u, ode, p)
```

which performs one Runge-Kutta (of order 4) integration step for a general ODE $\dot{x} = f(x, u, p)$ starting at x_0, with input u, parameters p and an integration interval ΔT. (1 point)

(d) MATLAB: Write a function `[x_sim] = sim(t,x0,u, integrator, ode, p)` which simulates the robot's behaviour given a set of inputs $\mathbf{u}$, starting at $x_0 = [0\ 0\ 0]^\top$ where you use the given integrator. (1 point)

(e) MATLAB: Plot the results you obtain from the `sim` function when calling it with `euler_step` and `RK4_step`.

PAPER: Are there any difference? Why (not)? (1 point)

Hint: You might also compare your solutions to the results obtained via `ode45` [1].

This sheet gives in total 10 points

[1] de.mathworks.com/help/matlab/ref/ode45.html