Worksheet: Linear Dynamical Systems

Joaquin Rapela and Aniruddh Galgali

March 10, 2024

1 Sampling from a linear dynamical system

Sample (i.e., simulate) observations from a linear dynamical system for tracking. Follow the equations on slide 16 of the Linear Dynamical Systems lecture. That is, first sample an initial state $\mathbf{x}_1$, then sample the remaining states $\mathbf{x}_2, \ldots, \mathbf{x}_N$, and finally simple the observations $\mathbf{y}_1, \ldots, \mathbf{y}_N$.

You may use the script doSimulateTrajectoryDWPA.py in the class repository, but you will need to complete a few missing parts in function `simulatLDS` of the module simulation.py (Listing1). First, in line 7 you need to draw all samples (N) of the observation noise. Second on line 15 you need to set the value of the state at time n. Finally, on line 16 you need to calculate the observation from the previously computed states and noise.

```
1  def simulateLDS(N, A, Q, H, R, mu0, P0):
2      M = A.shape[0]
3      P = H.shape[0]
4      # sample state noise
5      w = np.random.multivariate_normal(np.zeros(M), Q, N).T
6      # sample measurement noise
7      v = ...
8      # sample initial state
9      x0 = np.random.multivariate_normal(mu0, P0, 1).flatten()
10     # sample states
11     x = np.empty(shape=(M, N))
12     y = np.empty(shape=(P, N))
13     x[:, 0] = x0
14     for n in range(1, N):
15         x[:, n] = ...
16     y = ...
17     return x0, x, y
```

Listing 1: function `simulateLDS` in module `simulation.py`

You should obtain a plot similar to that in Figure 1.

2 Estimating kinematics of a foraging mouse

Here we will estimate kinematics of a foraging mouse using a linear dynamical system. Notes on how to design the matrices of a linear dynamical system for kinematics inference appear here. The state of this linear dynamical system is six dimensional, $\mathbf{x} \in \mathbb{R}^6$. $x[0]$ and $x[3]$, give the inferred position components along the horizontal and vertical direction, respectively. Similarly,

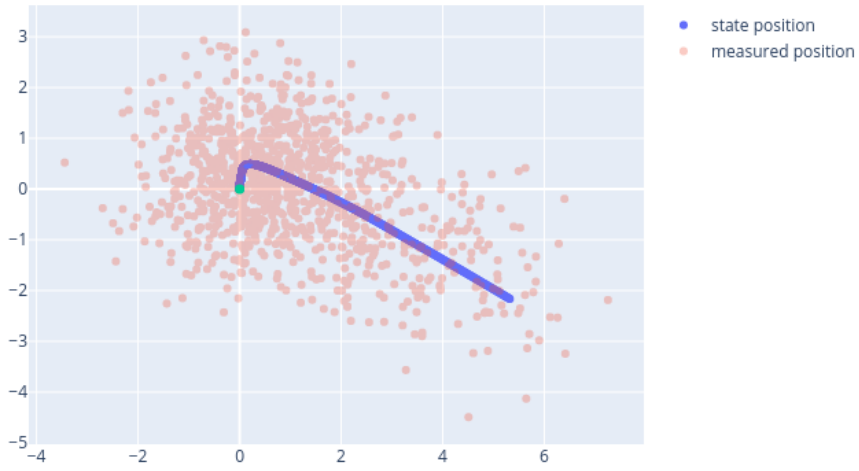

Figure 1: Simulated observations and state position components using the Discrete Wiener Process Acceleration model [Bar-Shalom et al., 2004, Section 6.3.3].

$x[1]$ and $x[4]$, give inferred velocity components, and $x[2]$ and $x[6]$, give the inferred acceleration components.

The video for this example was generously provided by the Sainsbury Wellcome Centre Foraging Behaviour Working Group (2023). Aeon: An open-source platform to study the neural basis of ethological behaviours over naturalistic timescales, `https://doi.org/10.5281/zenodo.8413142`.

To infer mouse kinematics from this video you may want to use the script doTrackMouse.py. This script reads a video file and using the OpenCV library, it estimates the centre of mass of the mouse and plots a red point at the position of this estimate.

Before running this script install OpenCV by running the following command from the root directory of your repository

```
cd worksheets/07_linearDynamicalSystems/code/scripts
pip install -r requirements.txt
```

You also need to download this video to the folder `worksheets/07_linearDynamicalSystems/data` of your repo.

The OpenCV estimates are noisy and sometimes they are not available due to occlusions. Here we use the Kalman filter to remove this noise, to fill the missing position values, and to estimate mouse velocities and accelerations.

At each sample time the Kalman filter provides the mean and covariance of the posterior distribution of the state. The script plots a green point on the video frame at the location of the mean position and a 95% confidence ellipse summarising the variability of the position posterior distribution. The brightness of this ellipse is modulated by the speed of the mouse, with a darker ellipse for slower speeds and a brighter ellipse for higher speeds. Text in the top left of the video prints the mouse speed and acceleration. Figure 2 plots one example frame of the video generated by the script. Click on the figure to obtain the full video.

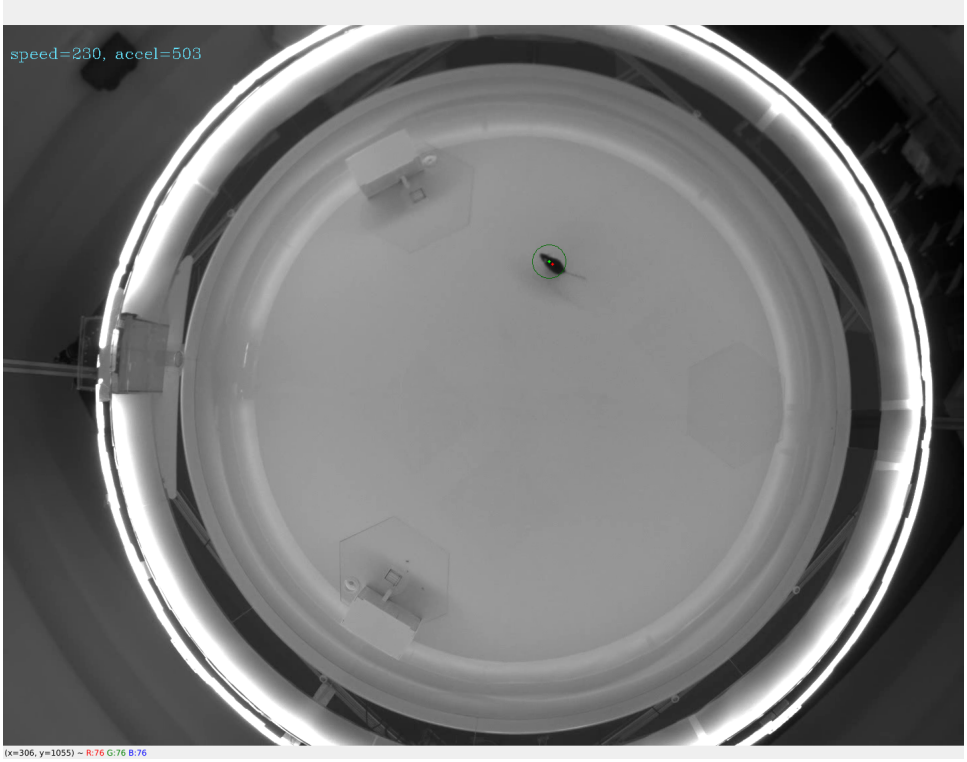

Figure 2: Example frame of the video generated by the doTrackMouse.py script. The OpenCV estimate of the mouse position is plotted in red, the Kalman filter estimated mean of the position is plotted in green, and the 95% confidence ellipse of the estimated position is drawn in green with brightness modulated by speed. Click on the figure to obtain the full video.

To run this script you will need to complete two methods in the class `OnlineKalmanFilter` in module inference.py. First, you will need to complete the method `predict` (Listing 2). When calling this method `self.x` contains the filtered mean at time t-1 (i.e., $\mu_{t-1|t-1}$ in slide 25 from the LDS lecture). In line 8 of Listing 2 you need to set `self.x` to the predicted mean at time t (i.e., $\mu_{t|t-1}$ in slide 25 from the LDS lecture). Similarly, in line 9 of Listing 2 you need to set the predicted covariance at time t (i.e., $P_{t|t-1}$ in slide 25 from the LDS lecture).

```
1     def predict(self):
2         """Predicts the next state.
3
4         :return: (mean, covariance): tuple containing the predicted mean and
    covariance matrix.
5
6         """
7
8         self.x = ...
9         self.P = ...
10        return self.x, self.P
```

Listing 2: method `predict` in class `OnlineKalmanFilter` in module `inference.py`

Second, you will need to complete the method `update` (Listing 3). When calling this method `self.x` contains the filtered mean at time t-1 (i.e., $\mu_{t-1|t-1}$ in slide 25 from the LDS lecture). Lines 11 and 12 of Listing 2 calculate the observation residuals ($y_t - H\mu_{t|t-1}$ in the expression of $\mu_{t|t}$ in slide 25 from the LDS lecture). Lines 13-15 of Listing 2 calculate the inverse of the observation residuals covariance matrix ($(HP_{t|t-1}H^\intercal + R)^{-1}$ in the expression of K_t in slide 25 from the LDS lecture). Line 16 computes Kalman gain (K_t in slide 25 from the LDS lecture). You will need to calculate the filtered mean in line 17 and the filtered covariance in line 18.

```
1     def update(self, y):
2         """Calculates the filtered mean vector and covariance matrix.
3
4         :param y: observation :math:`\in\Re^M`
5         :return: (state, covariance): tuple containing the filtered mean vector
     and covariance matrix.
6
7         """
8         if y.ndim == 1:
9             y = np.expand_dims(y, axis=1)
10        if not np.isnan(y).any():
11            pred_obs = self.H @ self.x
12            residual = y - pred_obs
13            Stmp = self.H @ self.P @ self.H.T + self.R
14            S = (Stmp + Stmp.T) / 2
15            Sinv = np.linalg.inv(S)
16            K = self.P @ self.H.T @ Sinv
17            self.x = ...
18            self.P = ...
19        return self.x, self.P
```

Listing 3: method `update` in class `OnlineKalmanFilter` in module `inference.py`

3 Adjusting the parameters of the kinematics model (optional)

You may have noticed that the filtered estimate of positions is sometimes sluggish. This may happen because the parameters `sigma_a`, `sigma_x` and/or `sigma_y` of the kinematics models are not set to optimal values. Adjust these parameters to eliminate this sluggish tracking. You may also want to automatically learn optimal parameter values using the Expectation Maximisation algorithm or by gradient ascent, as shown here.

References

Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.