School of Electronics and Computer Science
University of Southampton

COMP6247(2020/21): Reinforcement and Online Learning

Kalman Filter

Issue	16/02 2020
Deadline	01/03 (10:00 AM) ’

Introduction

The aim of this assignment (second for this module, worth 10% of your assessment) is to study online learning with Kalman filtering. We build on the Recursive Least Squares (RLS) algorithm covered in the first assignment, learning a framework in which inference about an underlying state and uncertainty on it can be quantified. Snippets of code are provided in Appendix to help you get started, but these should not be taken as complete working programs.

Generating Synthetic Data

Generating a Simple Autoregressive Time Series

An autoregressive time series (AR Process) of order p is given by the generating model:

$$s(n) = \sum_{k=1}^{p} a_k \, s(n-k) + v(n), \tag{1}$$

where n is index over time, $a_k,\ k = 1, .., p$ are the parameters of the process and $v(n)$ is Gaussian random noise, usually assumed to have zero mean and varince σ_v^2.
Figure 1 shows a random excitation signal and a second order autoregressive process obtained from Eqn. (1) for which $p = 2$.

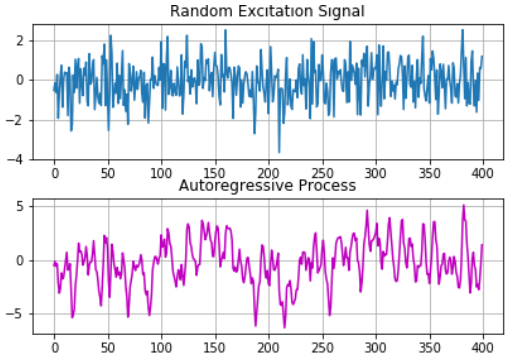

Figure 1: Random noise excitation signal and a second order autoregressive process.

Generating a Non-stationary Signal

We could study a time varying signal by slowly changing the parameters of the AR process over time. This generates a non-stationary signal. An example of slow variation in parameters and the resulting signal are shown in Fig. 2.

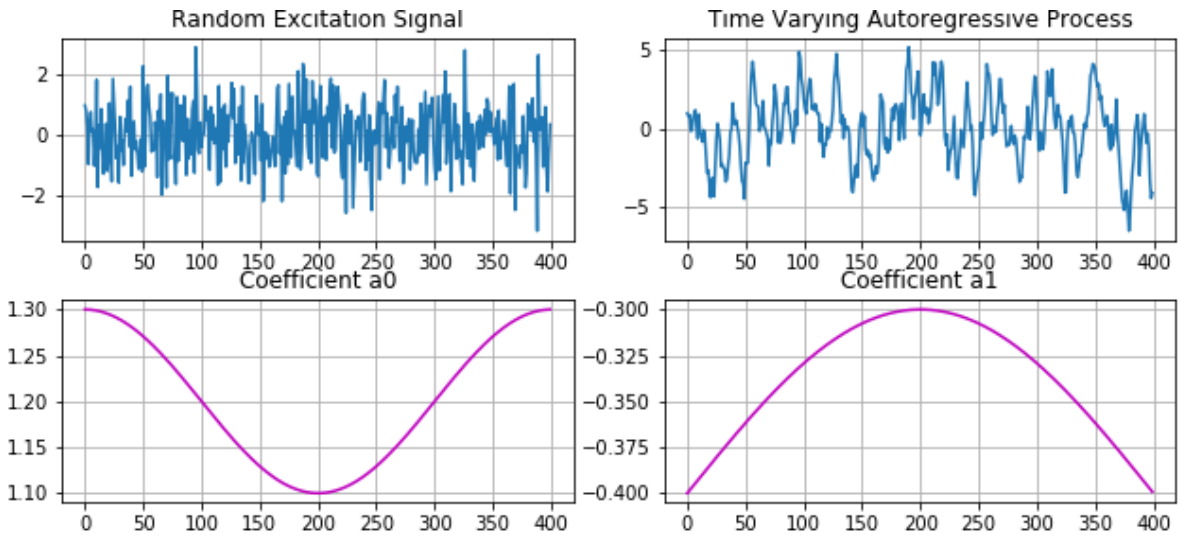

Figure 2: Slowly changing parameters of an AR process

Kalman Filter

A state space model for sequential estimation of parameters of a time series model is as follows:

$$\begin{aligned} \boldsymbol{\theta}(n) &= \boldsymbol{\theta}(n-1) + \boldsymbol{w}(n) \\ y(n) &= \boldsymbol{\theta}^T \boldsymbol{x}_n + v(n), \end{aligned}$$

where we have assumed a random walk model on the unknown parameters $\boldsymbol{\theta}$, a univariate observation and p past samples are held in the vector $\boldsymbol{x}(n) = [x_{n-1}, x_{n-2}, \ldots x_{n-p}]$. At every step in time, our task is to make the best estimate $\boldsymbol{\theta}$ from observation $y(n)$ and input $\boldsymbol{x}(n)$. The Kalman filter equtions for the above are given below. For the random walk, our predictions of the state and uncertainty on the state (error covariance matrix) are:

$$\begin{aligned} \boldsymbol{\theta}(n|n-1) &= \boldsymbol{\theta}(n-1|n-1) \\ P(n|n-1) &= P(n-1|n) + Q \end{aligned}$$

At time n, we have new data $\{\boldsymbol{x}(n),\, y(n)\}$. We predict the target signal as $\hat{y}(n) = \boldsymbol{x}(n)^T \boldsymbol{\theta}(n|n-1)$, using the predicted parameters. The innovation (or error) signal is:

$$e(n) = y(n) - \boldsymbol{x}(n)^T \boldsymbol{\theta}(n|n-1)$$

We now make posterior updates to the state estimates and the corresponding error covariance matrix:

$$\begin{aligned} \boldsymbol{\theta}(n|n) &= \boldsymbol{\theta}(n|n-1) + \boldsymbol{k}(n)\, e(n) \\ P(n|n) &= \left(I - \boldsymbol{k}(n)\, \boldsymbol{x}(n)^T\right) P(n|n-1) \end{aligned}$$

where the Kalman gain $\boldsymbol{k}(n)$ is given by

$$\boldsymbol{k}(n) = \frac{P(n|n-1)\, \boldsymbol{x}(n)}{R + \boldsymbol{x}(n)^T P(n|n-1)\, \boldsymbol{x}(n)}$$

Task

Following the snippets of code provided, implement Kalman filter to estimate the parameters of a synthetic second order AR process. Show how the parameters converge for five different initial conditions. Try various values for the process noise covariance and measurement noise variance and comment on their effects on convergence speeds. Show the convergence behaviour of the filter for the AR process parameters being constant and time varying.
Initially, set the process noice covariance Q to a small value, say $0.01I$ and the measurement noise variance to be of the same order of magnitude as the variance of the excitation signal of the AR synthetic process. Vary these hyperparameters and observe the effect on the convergence of the filter.

Report

Upload a report of **no more than four pages** describing your work. Please make sure you include your name or email in the report.

Appendix: Snippets of Code

Generating a Second Order Autoregressive Process

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Length of time series
#
N=400

# Gaussian random numbers as an excitation signal
#
ex = np.random.randn(N)

# Second order AR Process
#
a = np.array([1.2, -0.4])

S = ex.copy();
for n in range(2, N):
    x = np.array([S[n-1], S[n-2]])
    S[n] = np.dot(x, a) + ex[n]

fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(6,4))
plt.tight_layout()

ax[0].plot(range(N), ex)
ax[0].grid(True)
ax[0].set_title("Random Excitation Signal")

ax[1].plot(range(N), S, color='m')
ax[1].grid(True)
ax[1].set_title("Autoregressive Process")
```

Generating a Time-varying AR process

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Length of time series
#
N=400

# Gaussian random numbers as an excitation signal
#
ex = np.random.randn(N)

# Second order AR Process with coefficients slowly changing in time
#
a0 = np.array([1.2, -0.4])
A  = np.zeros((N,2))
omega, alpha = N/2, 0.1

for n in range(N):
    A[n,0] = a0[0] + alpha * np.cos(2*np.pi*n/N)
    A[n,1] = a0[1] + alpha * np.sin(np.pi*n/N)

S = ex.copy();
for n in range(2, N):
    x = np.array([S[n-1], S[n-2]])
    S[n] = np.dot(x, A[n,:]) + ex[n]

fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(9,4))
plt.tight_layout()

ax[1,0].plot(range(N), A[:,0])
ax[1,0].grid(True)
ax[1,0].set_title("Coefficient a0", color='m')

ax[1,1].plot(range(N), A[:,1], color='m')
ax[1,1].grid(True)
ax[1,1].set_title("Coefficient a1", color='m')

ax[0,0].plot(range(N), ex)
ax[0,0].grid(True)
ax[0,0].set_title("Random Excitation Signal")

ax[0,1].plot(range(N), S, color='m')
ax[0,1].grid(True)
ax[0,1].set_title("Time Varying Autoregressive Process")

plt.savefig("arChange.png")
```

Kalman Filter Estimates of second order AR parameters

```
# Time series data  y
# th_n_n:  estimate at time n using all data upto time n
# th_n_n1: estimate at time n using all data upto time n-1
#

# Initialize
#
x = np.zeros((2,1))
th_n1_n1 = np.random.randn(2,1)
P_n1_n1 = 0.001*np.eye(2)

# Noise variances -- hyperparameters (to be tuned)
# Set measurement noise as fraction of data variance (first few samples)
# Guess for process noise
#
R    = 0.2*np.std(ex[0:10])
beta = 0.0001
Q    = beta*np.eye(2)

# Space to store and plot
#
th_conv = np.zeros([2, N])

# First two estimates are initial guesses
#
th_conv[0,0] = th_n1_n1[0]
th_conv[0,1] = th_n1_n1[1]
th_conv[1,0] = th_n1_n1[0]
th_conv[1,1] = th_n1_n1[1]

# Kalman Iteration Loop (univariate observation, start from time step 2)
#
for n in range(2, N):
    # Input vector contains past velues
    x[0] = y[n-1]
    x[1] = y[n-2]

    # Prediction of state and covariance
    th_n_n1 = th_n1_n1.copy()
    P_n_n1  = P_n1_n1 + Q

    yh = th_n_n1.T @ x
    en = y[n] - yh
    ePlot[n] = en

    # Kalman gain (kn) and innovation variance (den)
    #
```

```
    den = x.T @ P_n1_n1 @ x + R
    kn = P_n1_n1 @ x / den

    # Posterior update
    #
    th_n_n = th_n_n1 + kn * en
    P_n_n  = (np.eye(2) - kn @ x.T) @ P_n_n1

    # Save
    th_conv[0,n] = th_n_n[0]
    th_conv[1,n] = th_n_n[1]

    # Remember for next step
    #
    th_n1_n1 = th_n_n.copy()
    P_n1_n1  = P_n_n.copy()

print(a)
print(th_n_n)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8,3))
ax[0].plot(th_conv[0])
ax[0].set_xlim(0,500)
ax[0].axhline(y=a[0], color='r')

ax[1].plot(th_conv[1])
ax[1].set_xlim(0,500)
ax[1].axhline(y=a[1], color='r')
ax[1].set_title("R = %4.3f, Q = %6.5f I"%(R, beta))
```