Assignment 7

Reid Ginoza

3/29/2020

Analytical

This problem studies the Beneš stochastic differential equation, as defined in (Särkkä and Solin 2019). Part (a): Write down the FPK for the Beneš Equation and check that the following probability density solves it:

$$p\left(x,t\right)=\frac{1}{\sqrt{2\pi t}}\cosh\left(x\right)\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2t}x^{2}\right) \tag{1}$$

My solution

The Beneš stochastic differential equation is as follows:

$$\mathrm{d}x=\tanh\left(x\right)\mathrm{d}t+\mathrm{d}\beta \tag{2}$$

The Fokker-Planck-Kolmogorov Equation (FPK) of the Beneš Equation, Eqn. 2, is as follows:

$$\frac{\partial p\left(x,t\right)}{\partial t}=-\frac{\partial}{\partial x}\left[\tanh\left(x\right)p\left(x,t\right)\right]+\frac{1}{2}\frac{\partial^{2}}{\partial x^{2}}\left[p\left(x,t\right)\right] \tag{3}$$

The solution of the FPK gives the probability density function of $x\left(t\right)$, the solution to the Beneš Equation, Eqn. 2.

Verifying Solution

Left-Hand Side

We'll take the partial derivative of p with respect to time, but first I'll rewrite p.

$$p\left(x,t\right) = \frac{\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right)}{\sqrt{2\pi t}}\cosh\left(x\right) \tag{4}$$

$$\text{Let } A\left(t\right) := \exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right)\text{, then} \tag{5}$$

$$\frac{\partial A}{\partial t} = \left(\frac{1}{2}x^2t^{-2}\right)\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right) - \frac{1}{2}\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right) \tag{6}$$

$$\frac{\partial p\left(x,t\right)}{\partial t} = \frac{\sqrt{2\pi t}\left(\frac{\partial A}{\partial t}\right) - \left(\frac{1}{2\sqrt{2\pi t}}\right)A}{2\pi t}\cosh\left(x\right) \tag{7}$$

$$\begin{aligned}&= \Bigg[\Bigg[\left(\frac{\sqrt{2\pi t}}{4\pi t}x^2t^{-2}\right)\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right) - \frac{\sqrt{2\pi t}}{4\pi t}\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right)\Bigg]\\ &\quad - \left[\frac{1}{4\pi t\sqrt{2\pi t}}\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right)\right]\Bigg]\cosh\left(x\right)\end{aligned} \tag{8}$$

$$= \left[\left(\frac{\sqrt{2\pi t}}{4\pi t}x^2t^{-2}\right) - \frac{\sqrt{2\pi t}}{4\pi t} - \frac{1}{4\pi t\sqrt{2\pi t}}\right]\left[\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2}x^2t^{-1}\right)\right]\cosh\left(x\right) \tag{9}$$

$$= \frac{1}{2\sqrt{2\pi t}}\left(\frac{x^2}{t^2} - \frac{1}{t} - 1\right)\exp\left(-\frac{t}{2}\right)\exp\left(-\frac{x^2}{2t}\right)\cosh\left(x\right) \tag{10}$$

$$= p\left[\frac{1}{2}\left(\frac{x^2}{t^2} - \frac{1}{t} - 1\right)\right] \tag{11}$$

Right-Hand Side

First derivative of p with respect to x

Remember, $p\left(x,t\right) = \frac{1}{\sqrt{2\pi t}}\cosh\left(x\right)\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{1}{2t}x^2\right)$.

$$\frac{\partial p}{\partial x} = \frac{1}{\sqrt{2\pi t}}\exp\left(-\frac{1}{2}t\right)\left[-\frac{1}{t}x\cosh\left(x\right)\exp\left(-\frac{1}{2t}x^2\right) + \sinh\left(x\right)\exp\left(-\frac{1}{2t}x^2\right)\right] \tag{12}$$

Second partial derivative of p with respect to x

$$\frac{\partial^2 p}{\partial x^2} = \frac{1}{\sqrt{2\pi t}}\exp\left(-\frac{1}{2}t\right)\exp\left(-\frac{x^2}{2t}\right)\left[\left(-\frac{x}{t}\sinh\left(x\right) - \frac{1}{t}\cosh\left(x\right) + \frac{x^2}{t^2}\cosh\left(x\right)\right) + \left(\cosh\left(x\right) - \frac{x}{t}\sinh\left(x\right)\right)\right] \tag{13}$$

First Term of Right-Hand Side

$$\text{Let } B := \tanh\left(x\right)p\left(x,t\right), \tag{14}$$

$$\text{Then } \frac{\partial B}{\partial x} = \frac{1}{\sqrt{2\pi t}}\exp\left(-\frac{t}{2}\right)\exp\left(-\frac{x^2}{2t}\right)\left[\cosh\left(x\right) - \frac{x}{t}\sinh\left(x\right)\right] \tag{15}$$

Complete Right-Hand Side

$$
\begin{aligned}
\frac{\partial p(x,t)}{\partial t} &= -\frac{\partial B}{\partial x} + \frac{1}{2}\frac{\partial^2 p}{\partial x^2} && (16)\\
&= \frac{1}{\sqrt{2\pi t}} \exp\left(-\frac{t}{2}\right) \exp\left(-\frac{x^2}{2t}\right) \left[-\cosh(x) + \frac{x}{t}\sinh(x)\right] \\
&\quad + \frac{1}{2}\left[\left(-\frac{x}{t}\sinh(x) - \frac{1}{t}\cosh(x) + \frac{x^2}{t^2}\cosh(x)\right) + \left(\cosh(x) - \frac{x}{t}\sinh(x)\right)\right] && (17)\\
&= \frac{1}{\sqrt{2\pi t}} \exp\left(-\frac{t}{2}\right) \exp\left(-\frac{x^2}{2t}\right) \left[-\frac{1}{2}\cosh(x) - \frac{1}{2t}\cosh(x) + \frac{x^2}{2t^2}\cosh(x)\right] && (18)\\
&= \frac{1}{\sqrt{2\pi t}} \exp\left(-\frac{t}{2}\right) \exp\left(-\frac{x^2}{2t}\right) \cosh(x) \left[-\frac{1}{2} - \frac{1}{2t} + \frac{x^2}{2t^2}\right] && (19)\\
&= p\left[\frac{1}{2}\left(\frac{x^2}{t^2} - \frac{1}{t} - 1\right)\right] && (20)
\end{aligned}
$$

and thus we see the left-hand side, Eqn. 11 is equal to the right-hand side above, Eqn. 20.

Numerical Problem

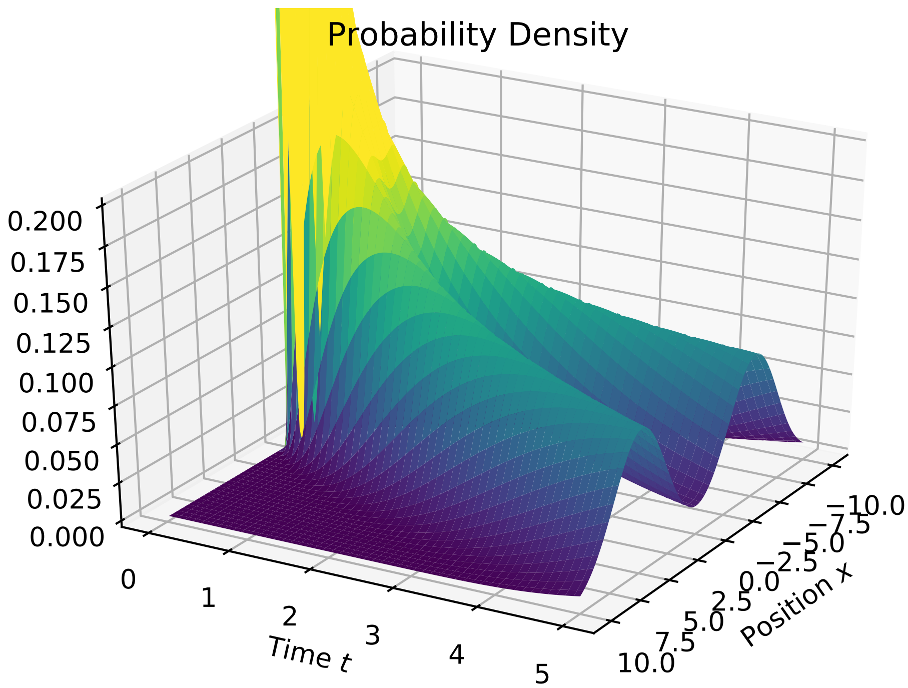
We will plot the probability density function of the solution to the Beneš stochastic differential equation, Eqn. 2.

```
from matplotlib.colors import Normalize
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
from scipy.stats import kde
import seaborn as sns
```

```
# --- Plot Known density
def p_analytical(x, t):
    return 1 / np.sqrt(2 * np.pi * t) * np.cosh(x) * np.exp(-t/2) * np.exp(-x**2/(2*t))


x_plot = np.linspace(-10, 10, 200)
t_plot = np.linspace(0, 5, 201)[1:]

xx, tt = np.meshgrid(x_plot, t_plot)

norm = Normalize(vmin=0,vmax=0.2)

fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')
ax1.set_zlim(top=0.2)
```

```
## (0.0, 0.2)
```

```
ax1.plot_surface(xx, tt, p_analytical(xx, tt), cmap=plt.cm.get_cmap('viridis'),
                 norm=norm)
ax1.view_init(30, 30)
plt.xlabel('Position $x$')
plt.ylabel('Time $t$')
plt.title('Probability Density')
```



Now we will set up the code to run the Euler-Maruyama stepping method on the Beneš stochastic differential equation. The Euler-Maruyama solver has been updated to accept functions (instead of constants) for the shirt and drift/dispersion parameters. First is the numerical and post-processing code.

```
def multiple_brownian_motion(end_time=1., num_tsteps=500, n_trials=1000):
    """Creates multiple 1-D Brownian motion with time as the row index and
    each column as a separate path of Brownian motion.

    This assumes that all Brownian motion starts at 0. Currently only
    implements one-dimensional Brownian motion. This also assumes all
    step sizes are the same size.

    The steps of Brownian motion, ``dw``, are modeled with a Gaussian
    distribution with mean 0 and variance ``sqrt(dt)``, where ``dt``
    is the constant time step size.

    Parameters
    ----------
    end_time : float
```

```python
    num_tsteps : int
        The number of steps to take. Will calculate the step
        size dt internally. The number of rows in the output of
        Brownian motion will be num_tsteps + 1.

    n_trials : int
        The number of sample paths to create. This will be the number
        of columns in the output.

    Returns
    -------
    t : ndarray
        One-dimensional time ndarray from 0 to ``end_time`` with
        shape (``num_tsteps``+1,)

    w : ndarray
        Two-dimensional ndarray representing ``n_trials`` number of
        sample paths of one-dimensional Brownian motion.
        This will be of shape (``num_tsteps``+1, ``n_trials``).

    dt : float
        The value indicating the step size of t. This is only implemented
        with constant step size.

    dw : ndarray
        Two-dimensional ndarray representing the steps of Brownian motion.
        The first row is all zeros. Each i-th row of ``dw``, ie. dw[i, :]
        indicates the change in ``w`` from w[i-1, :] to w[i, :]
        This will be the same shape as ``w``, (``num_tsteps``+1, ``n_trials``).

    """

    dt = (end_time - 0) / num_tsteps
    dw = np.random.normal(scale=np.sqrt(dt), size=(num_tsteps+1, n_trials))
    # Brownian motion must start at time 0 with value 0
    dw[0] = np.zeros_like(dw[0])
    w = dw.cumsum(axis=0)
    # t is not used in calculations, but returned to allow user to keep track
    # of points in time
    t = np.linspace(0, end_time, num=num_tsteps+1).reshape((num_tsteps+1, 1))
    assert w.shape[0] == t.shape[0], ('time and position arrays are not the '
                                      'same length. w.shape[0] - t.shape[0] = '
                                      f'{w.shape[0] - t.shape[0]}')
    assert w.shape == dw.shape, ('position and velocity arrays are not the '
                                 'same shape: '
                                 f'w.shape: {w.shape}    dw.shape: {dw.shape}')
    return t, w, dt, dw


def euler_maruyama_nonlinear_vec(f, g, x0, t, dt, dw, M):
    """
    calculates the EM approximation on the nonlinear one-dimensional SDE,
    vectorized for multiple trials based on the shape of ``dw``.
```

```
    SDE is of the form:
    dX = f(X)dt + G(X)dW; X(0) = X0

    :param f: shift function in SDE. Must be passed as a function of x
    :param g: drift/dispersion function in SDE
    :param x0: Initial condition
    :param t: time one dimensional nd-array
    :param dt: step size of time array, float
    :param dw: White noise associated with the Brownian motion, ndarray
    :param M: multiple of dt for Euler-Maruyama step size. Do not make this too large.
    :return: time array and solution x array
    """

    if M < 1:
        raise ValueError('M must be greater than or equal to 1')

    Dt = M * dt  # EM step size
    L = (t.shape[0] - 1) / M  # number of EM steps

    if not L.is_integer():
        raise ValueError('Cannot handle Step Size that is not a multiple of M')

    L = int(L)  # needed for range below

    x = [np.full((dw.shape[1],), x0)]
    T = [0]
    for i in range(1, L+1):
        # DW is the step of Brownian motion for EM step size
        DW = (dw[M * (i - 1) + 1:M * i + 1, :]).sum(axis=0).reshape(dw.shape[1], )
        x.append(x[i-1] + Dt * f(x[i-1]) + g(x[i-1]) * DW)
        T.append(T[i-1] + Dt)

    return np.array(T), np.array(x)


def plot_on_axis(ax, time, pos, cols, title, color_map, with_mean=False):
    for idx, col in enumerate(cols):
        ax.plot(time, pos[:, col], c=color_map(idx))
    ax.set_title(title)
    if with_mean:
        ax.plot(time, pos.mean(axis=1), color='black',
            label=r'Sample Mean $(n={})$'.format(pos.shape[1]), linewidth=2)
        ax.legend()
```

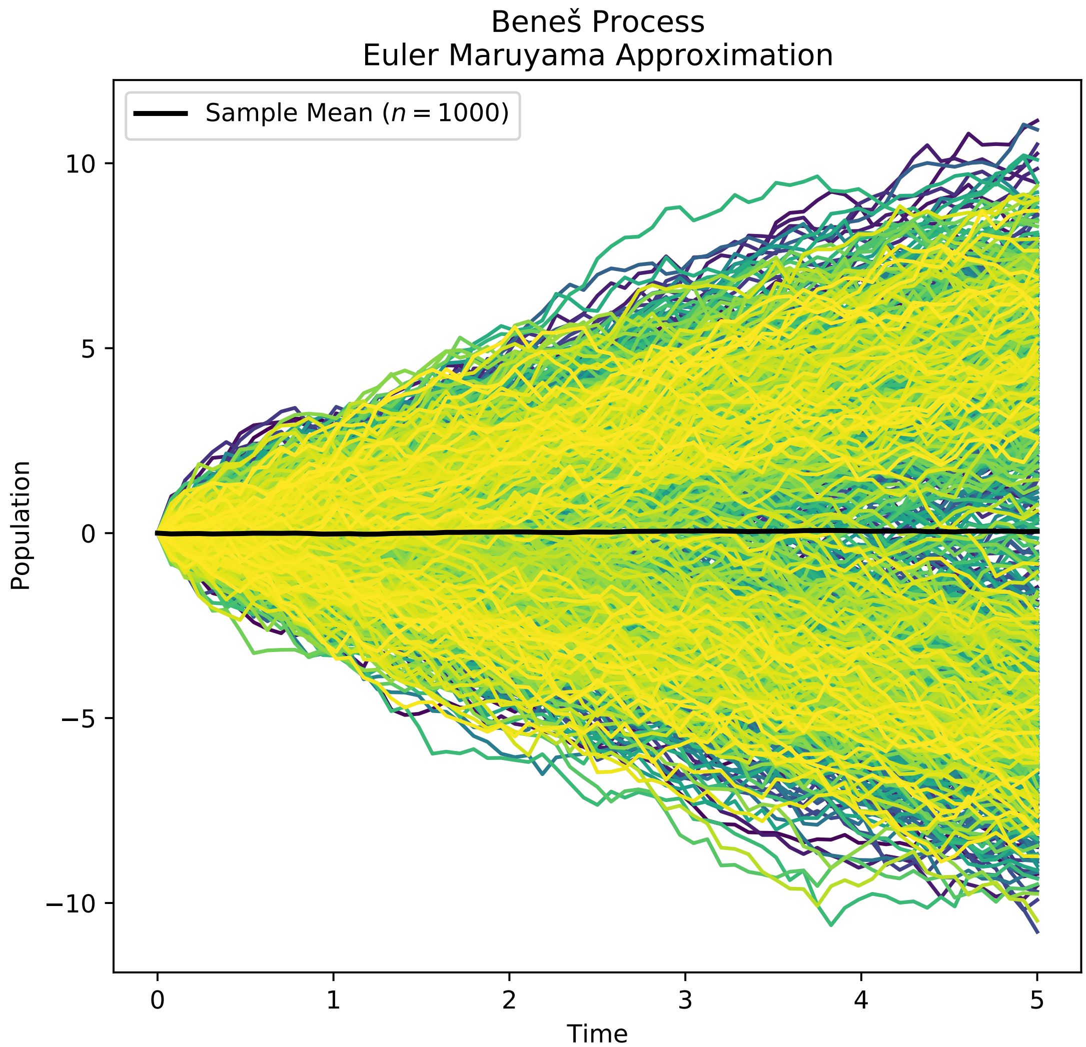
What follows would be considered the input deck for this problem.

```
# Parameters for SDE
def f(x):
    return np.tanh(x)

def g(x):
    del x  # unused
    return 1
```

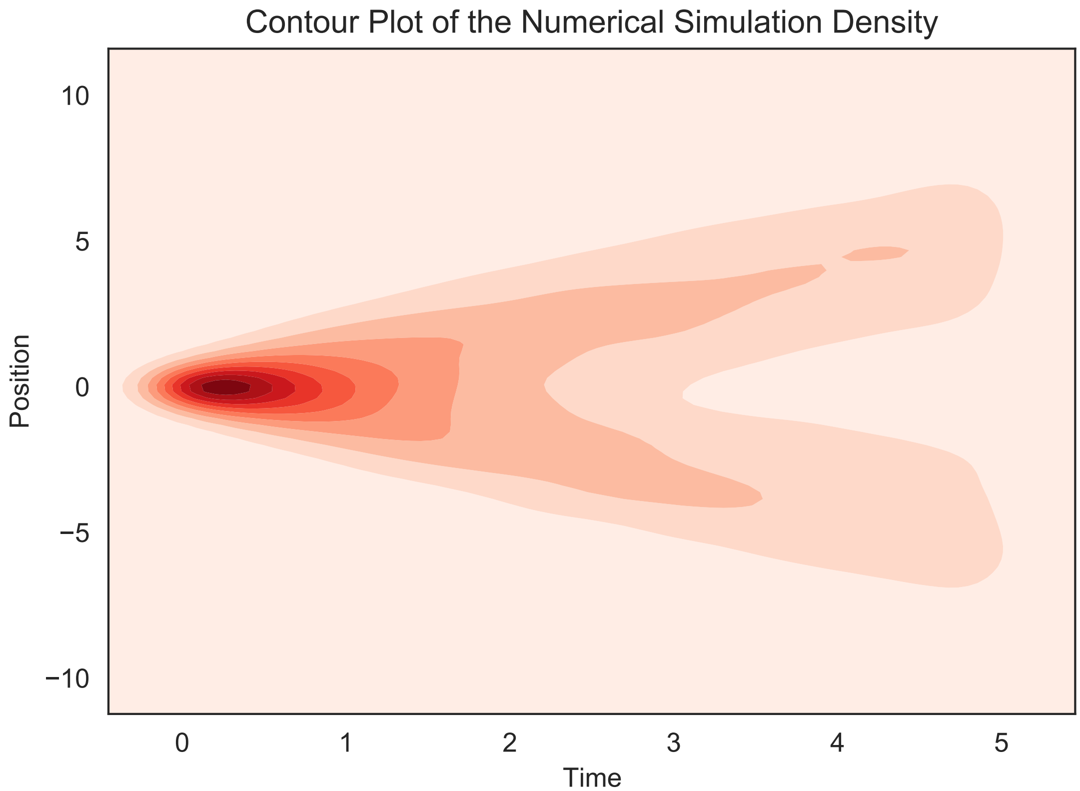
```python
# Initial Condition
x0 = 0

# Brownian Motion
END_TIME = 5.
NUM_TSTEPS = 2**8
N_TRIALS = 1000

# Euler-Maruyama
M = 4  # multiple of step size

# Plotting Parameters
N_PATHS = 1000
viridis = plt.get_cmap('viridis', lut=N_PATHS)
columns = np.arange(N_PATHS)
np.random.shuffle(columns)
plot_columns = columns[:N_PATHS]
```

Now we have the calculation and postprocessing.

```python
# -- Calculate
t, w, dt, dw = multiple_brownian_motion(end_time=END_TIME,
                                        num_tsteps=NUM_TSTEPS,
                                        n_trials=N_TRIALS)

t_em, x_em = euler_maruyama_nonlinear_vec(f, g, x0, t, dt, dw, M)

fig, ax = plt.subplots(1, figsize=(7, 6.5))
plot_on_axis(ax, t_em, x_em, plot_columns, r'Beneš Process'+'\nEuler Maruyama Approximation',
             color_map=viridis, with_mean=True)
plt.xlabel('Time')
plt.ylabel('Population')
```




```
plt.figure()
data_points = pd.DataFrame({
    'Time': np.tile(t_em, N_TRIALS),
    'Position': x_em.flatten(order='F')
})

sns.set_style("white")
sns.kdeplot(data_points['Time'], data_points['Position'], cmap="Reds", shade=True, bw=.15)
plt.title('Contour Plot of the Numerical Simulation Density')
```



References

Särkkä, Simo, and Arno Solin. 2019. *Applied Stochastic Differential Equations*. Vol. 10. Cambridge University Press.