Assignment 5

Reid Ginoza

3/22/2020

Analytical

Exercise

This is a problem from (Douglas and Peter 2006): Verify that

$$N = N_0 \mathrm{e}^{\left(r - \frac{u^2}{2}\right)t + uB_t}$$

is the solution to

$$\frac{\mathrm{d}N}{\mathrm{d}t} = rN + uW(t)N$$

where r and u are constants.

This is a stochastic form of the population growth model, which models the population growth as propotional to the current population. This stochastic form models the stochastic term as also propotional to the current population.

My Solution

First, we write the differential equation in the form used in (Evans 2012).

$$\frac{\mathrm{d}N}{\mathrm{d}t} = rN + uWN \tag{1}$$

$$\mathrm{d}N = F\mathrm{d}t + G\mathrm{d}B_t, \tag{2}$$

where

$$F = rN \tag{3}$$

$$G = uN \tag{4}$$

$$dB_t = W\mathrm{d}t \tag{5}$$

From this form, we can apply Itô chain rule by taking the logarithm of both sides. In otherwords, we first study a function of the random variable N, $\Phi(N) = \ln(N)$. Then for our new function Φ we have the following derivatives:

$$\Phi_t = 0; \tag{6}$$

$$\Phi_N = \frac{1}{N} \tag{7}$$

$$\Phi_{NN} = -\frac{1}{N^2} \tag{8}$$

And we can proceed

$$\mathrm{d}\Phi = \mathrm{d}\ln(N) = \left(\Phi_t + \Phi_N F + \frac{1}{2}\Phi_{NN}G^2\right)\mathrm{d}t + \Phi_N G \mathrm{d}B_t \tag{9}$$

$$= \left(0 + \left(\frac{1}{N}\right)(rN) - \frac{1}{2}\left(\frac{1}{N^2}\right)(uN)^2\right)\mathrm{d}t + \left(\frac{1}{N}\right)(uN)\,\mathrm{d}B_t \tag{10}$$

$$\mathrm{d}\Phi = \left(r - \frac{u^2}{2}\right)\mathrm{d}t + u\mathrm{d}B_t \tag{11}$$

In the last line above, r and u are constants from the problem definition. By the definition of the stochastic differential, we know the solution Φ, and we can solve for Φ by integration.

$$\Phi = \ln(N) = \int_0^t \left(r - \frac{u^2}{2}\right)\mathrm{d}t + \int_0^t u\mathrm{d}B_t \tag{12}$$

$$\ln(N) = \left(r - \frac{u^2}{2}\right)t + uB_t + C \tag{13}$$

$$N = C\mathrm{e}^{\left(r-\frac{u^2}{2}\right)t+uB_t} \tag{14}$$

Lastly, when $t = 0$, $B_t = 0$ by virtue of Brownian motion. Let $N_0 = N(0)$, our initial condition, and we have:

$$N = N_0\mathrm{e}^{\left(r-\frac{u^2}{2}\right)t+uB_t}. \tag{15}$$

Simulation

This code calculates the above stochastic differential equation using the Euler-Maruyama method. This follows (Higham 2001) but vectorizes it for multiple trials. The code was written using Python 3 (Van Rossum and Drake 2009), and the NumPy (Oliphant 2006) and Matplotlib (Hunter 2007) packages.

Set up and Functions

This first code chunk establishes the packages imported and the functions I've written to run this code. I've written some docstrings and comments to help keep my code more professional and readable than in previous exercises. I've also added `assert` lines, which are used to help the developer identify any bugs, and `raise` statements, which are used to help both the developer and the user identify proper usage of the code. Note that although (Douglas and Peter 2006) and (Higham 2001) use `r` and `R` respectively, this has been changed to `b` and `M` since this was compilted with RStudio (RStudio Team 2015) and Reticulate (Allaire et al. 2017) and `r` and `R` may have already been reserved.

```python
from matplotlib import pyplot as plt
import numpy as np



def multiple_brownian_motion(end_time=1., num_tsteps=500, n_trials=1000):
    """Creates multiple 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

    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]}')
```

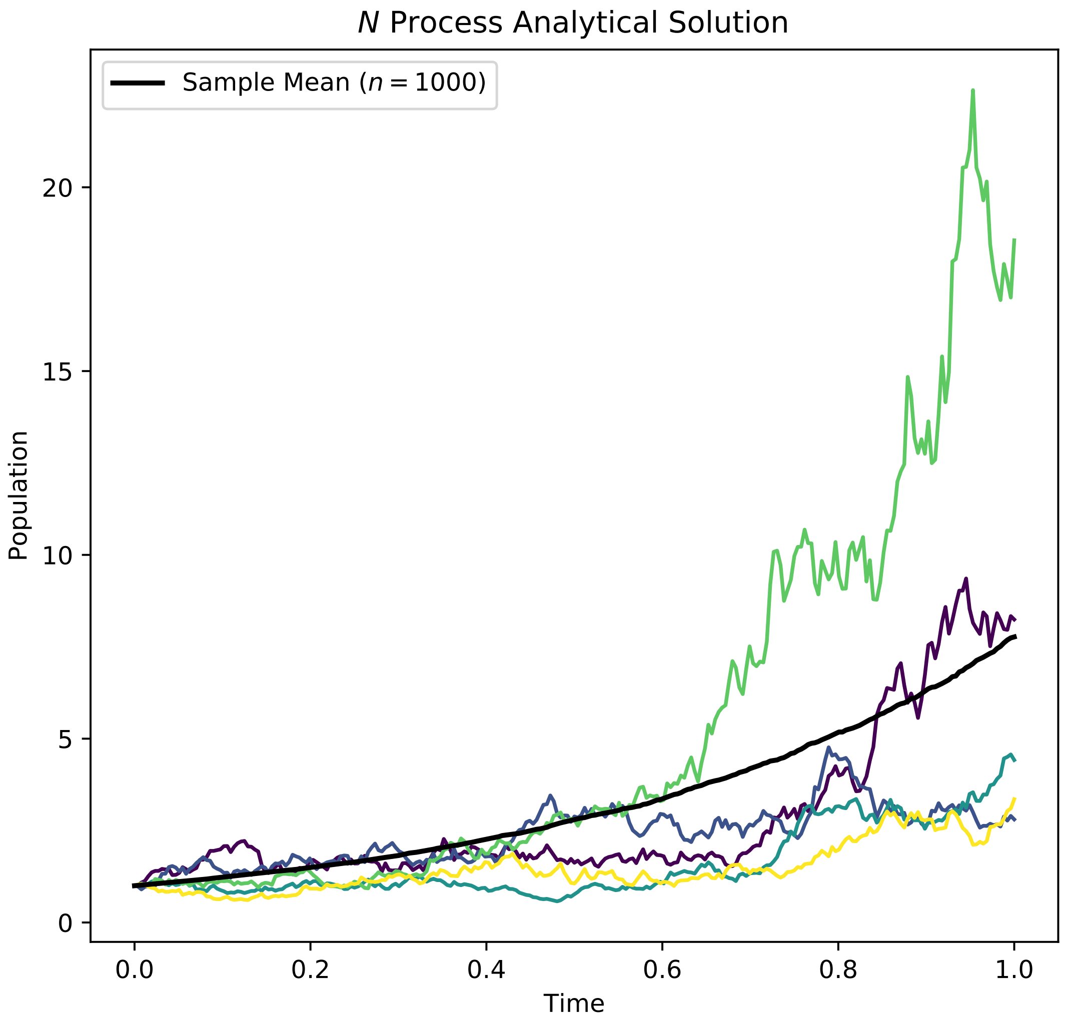
```python
    assert w.shape == dw.shape, ('position and velocity arrays are not the '
                                 'same shape: '
                                 f'w.shape: {w.shape}    dw.shape: {dw.shape}')
    return t, w, dt, dw


# --- Solve SDE
def analytical_truth_linear(b, u, N0, t, w):
    """
    Provides the true (analyatical) solution evaluated at t

    Can be vectorized for multiple trials based on the Brownian motion
    parameter ``w``.

    :param b: shift parameter in SDE
    :param u: drift parameter in SDE
    :param N0: Initial condition
    :param t: time nd-array
    :param w: brownian motion nd-array
    :param dt: step size of time array, float
    :param dw: White noise associated with the Brownian motion, ndarray

    :return: nd-array true solution
    """
    return N0 * np.exp((b - 0.5 * u ** 2) * t + u * w)


def euler_maruyama_linear(b, u, N0, t, dt, dw, M):
    """
    calculates the EM approximation on the linear SDE
    :param b: shift parameter in SDE
    :param u: drift parameter in SDE
    :param N0: Initial condition
    :param t: time nd-array
    :param dt: step size of time array, float
    :param dw: White noise/Gaussian associated with the Brownian motion, nd-array
    :param M: multiple of dt for Euler-Maruyama step size
    :return: time array and solution N array
    """

    Dt = M * dt  # EM step size
    L = (t.shape[0] - 1) / M  # number of EM steps
    assert L.is_integer(), 'Cannot handle Step Size that is not a multiple of M'
    L = int(L)  # needed for range below

    N = [N0]
    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)
        N.append(N[i-1] + Dt * b * N[i - 1] + u * N[i - 1] * DW)
        T.append(T[i-1] + Dt)
    return np.array(T), np.array(N)


def euler_maruyama_linear_vec(b, u, N0, t, dt, dw, M):
    """
    calculates the EM approximation on the linear SDE, vectorized for
    multiple trials based on the shape of ``dw``.
    :param b: shift parameter in SDE
    :param u: drift parameter in SDE
    :param N0: Initial condition
    :param t: time 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 N 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

    N = [np.full((dw.shape[1],), N0)]
    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], )
        N.append(N[i-1] + Dt * b * N[i - 1] + u * N[i - 1] * DW)
        T.append(T[i-1] + Dt)

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


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()
```

Now we are able to set up our particular problem with parameters for the population growth stochastic differential equation, the initial condition, the Brownian motion parameters, and the Euler-Maruyama multiple of the step size. For a larger program, the code chunk below would be the input deck.

```python
# Parameters for SDE
b = 2
u = 1

# Initial Condition
N0 = 1

# Brownian Motion
END_TIME = 1.
NUM_TSTEPS = 2**8
N_TRIALS = 1000

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

# Plotting Parameters
N_PATHS = 5
viridis = plt.get_cmap('viridis', lut=N_PATHS)
columns = np.arange(N_PATHS)
np.random.shuffle(columns)
plot_columns = columns[:N_PATHS]
```

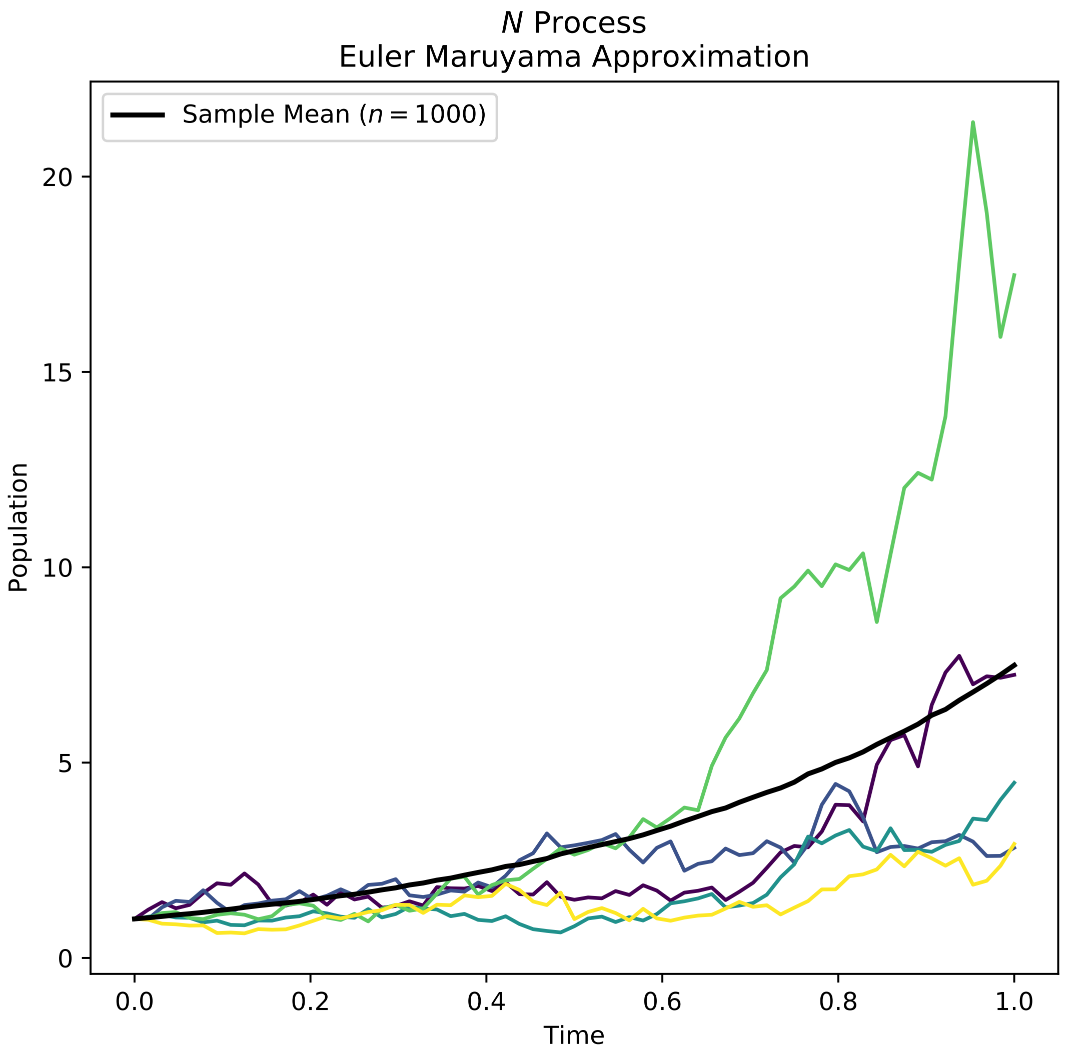
True Solution

```python
t, w, dt, dw = multiple_brownian_motion(end_time=END_TIME,
                                        num_tsteps=NUM_TSTEPS,
                                        n_trials=N_TRIALS)
N_true = analytical_truth_linear(b, u, N0, t, w)

fig, ax = plt.subplots(1, figsize=(7, 6.5))
plot_on_axis(ax, t, N_true, plot_columns, r'$N$ Process Analytical Solution',
             color_map=viridis, with_mean=True)
plt.xlabel('Time')
plt.ylabel('Population')
```



Euler-Maruyama Numerical Approximation

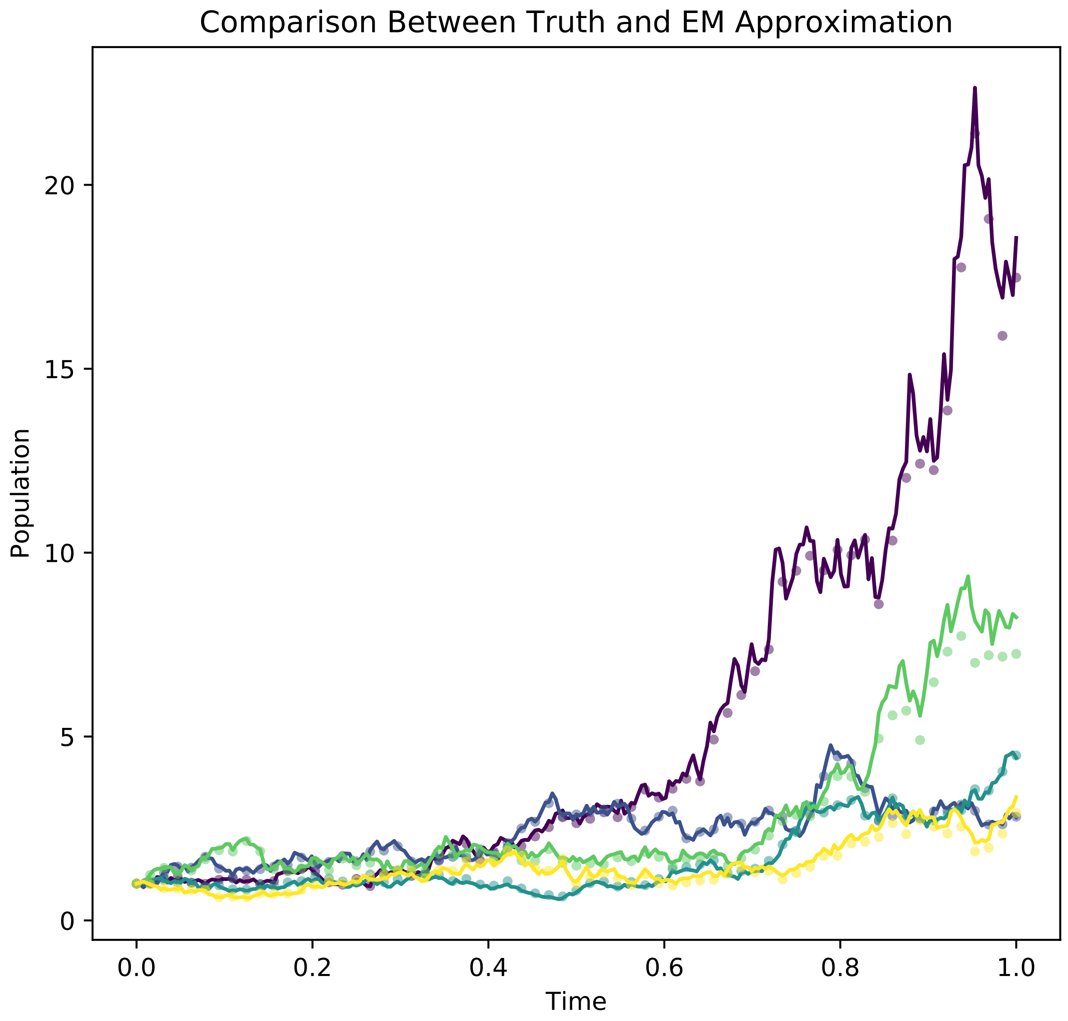
```
t_em_v, N_ems_v = euler_maruyama_linear_vec(b, u, N0, t, dt, dw, M)

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



Comparison

This plot shows the true solution with `NUM_TSTEPS = 2^8` time steps as the solid lines, and the Euler-Maruyama approximation with `NUM_TSTEPS/M = 2^6` time steps as dots.

```
fig, ax = plt.subplots(1, figsize=(7, 6.5))
for p in range(N_PATHS):
    ax.plot(t, N_true[:, p], c=viridis(p))
    ax.plot(t_em_v, N_ems_v[:, p], c=viridis(p), marker='.', ls='', alpha=0.5)
ax.set_title('Comparison Between Truth and EM Approximation')
plt.xlabel('Time')
plt.ylabel('Population')
plt.show()
```



References

Allaire, JJ, Kevin Ushey, Yuan Tang, and Dirk Eddelbuettel. 2017. *Reticulate: R Interface to Python*. https://github.com/rstudio/reticulate.

Douglas, Henderson, and Plaschko Peter. 2006. *Stochastic Differential Equations in Science and Engineering (with Cd-Rom)*. World Scientific.

Evans, Lawrence C. 2012. *An Introduction to Stochastic Differential Equations*. Vol. 82. American Mathematical Soc.

Higham, Desmond J. 2001. "An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations." *SIAM Review* 43 (3). SIAM: 525–46.

Hunter, John D. 2007. "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering* 9 (3). IEEE Computer Society: 90–95.

Oliphant, Travis. 2006. "NumPy: A Guide to NumPy." USA: Trelgol Publishing. http://www.numpy.org/.

RStudio Team. 2015. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Van Rossum, Guido, and Fred L. Drake. 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.