Python Programming for Finance

Marius A. Zoican, PhD.

Lecture 5: Stochastic processes in Python

Outline

Basic issue

1. We work with random numbers generated by functions from the `numpy.random` sublibrary.

```
import numpy as np
import numpy.random as npr
```

2. Easiest function: `npr.rand(x)` returns an array of size x from the Uniform $[0, 1]$ distribution.
3. You can extend the array to multiple dimensions: `npr.rand(x,y,z,...)`
4. To get random numbers between two different real numbers, a and b, you can transform `npr.rand(x)` as: `a+(b-a)*npr.rand(x,y,z,...)`
5. The transformation works perfectly well with multi-dimensional arrays!

Other random number functions

- `npr.randn`: standard normal random numbers.
- `npr.randint(low,high)`: random *integers* from "low" (inclusive) to "high" (exclusive)
- `npr.random_integers(low,high)`: random *integers* from "low" (inclusive) to "high" (inclusive)
- `npr.choice(v, size=x, replace=True or False)`: random sample from given vector v, of size x, with or without replacement.
- `npr.random`, `npr.ranf`, `npr.sample` all generate random floats in $[0, 1)$ (different seed methods).
  You need to specify size explicitly, i.e., `size=?`.

Applications

Standard normals

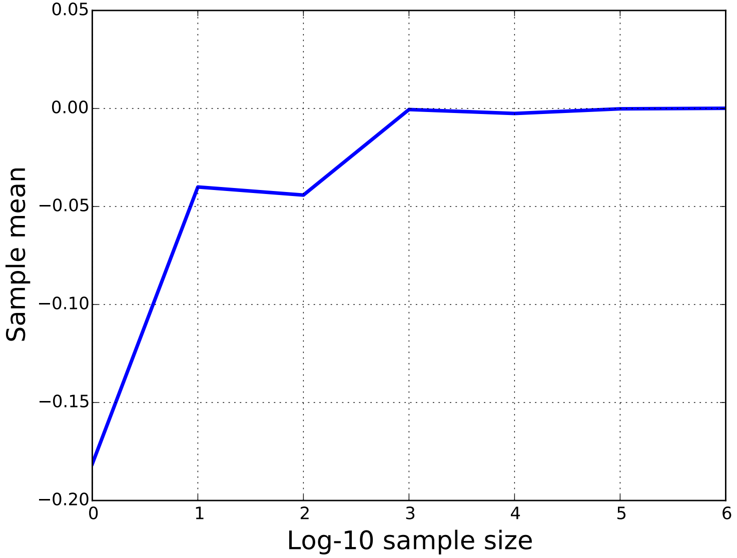
Draw from the standard normal distribution eight samples. The first sample has one observation, the second 10 observations, the third 10^2 observations, the eighth one, 10^7. Plot the means and standard deviation of each sample against the sample decade (log-10 scale). What do you observe? How do you interpret the observation?

Sampling from a population

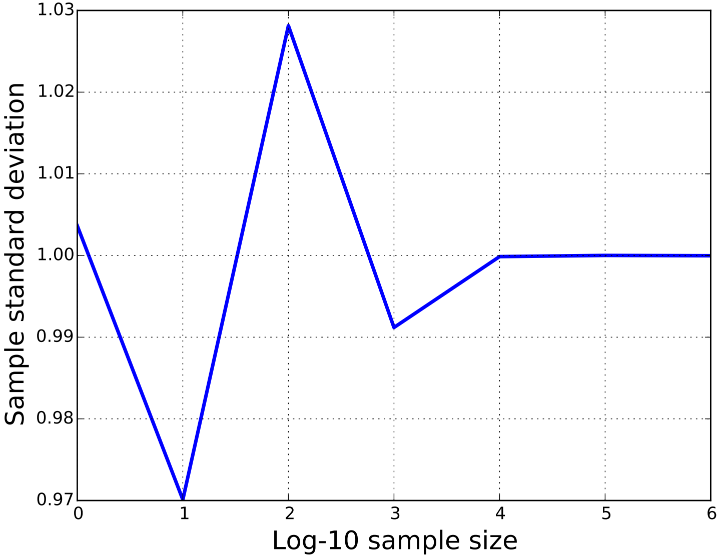
There are nine students in a class. One of them is 23, two are 24, three are 25, one is 26, and two are 27. Draw a 10,000-large sample from their age distribution. Plot a histogram of the empirical frequencies. What do you expect the height of the third column to be relative to the fifth?

Solution (1)

```python
exponents=range(0,8)
means=np.zeros(8)
stds=np.zeros(8)
for sample_size in [10**x for x in exponents]:
    r_sample=npr.randn(sample_size)
    means[np.log10(sample_size)]=r_sample.mean()
    stds[np.log10(sample_size)]=r_sample.std()

plt.plot(means,lw=2.5)
plt.xlabel("Log-10␣sample␣size",fontsize=18)
plt.ylabel("Sample␣mean",fontsize=18)
plt.grid()
```



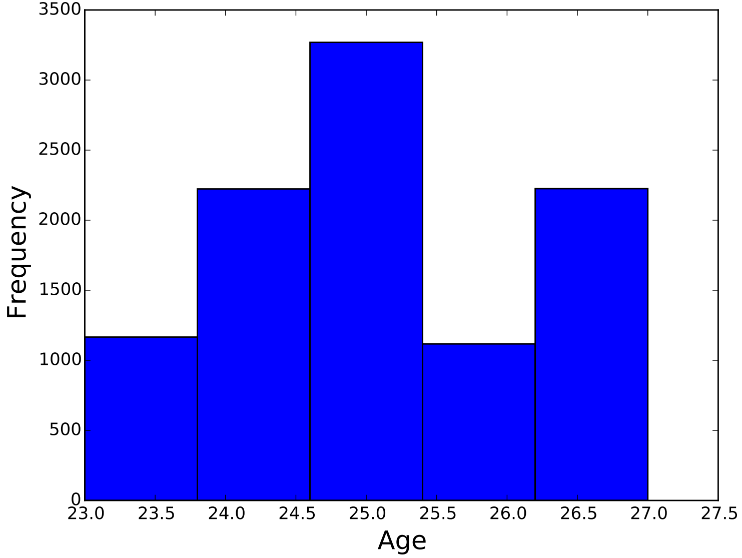
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
1
2
3
4
5
6
Sample mean
Log-10 sample size



Sample standard deviation
1.03
1.02
1.01
1.00
0.99
0.98
0.97
0
1
2
3
4
5
6
Log-10 sample size

Solution (2)

```
ages=[23,24,24,25,25,25,26,27,27]
age_sample=npr.choice(ages, 10000)
plt.hist(age_sample, bins=5)
plt.xlabel("Age", fontsize=18)
plt.ylabel("Frequency", fontsize=18)
```



Frequency
3500
3000
2500
2000
1500
1000
500
0
23.0
23.5
24.0
24.5
25.0
25.5
26.0
26.5
27.0
27.5
Age

Yet some more distributions to draw from...

Function	Parameters	Distribution
`beta`	a, b, size	beta distribution
`binomial`	n, p, size	binomial distribution
`chisquare`	df, size	χ^2 distribution
`f`	dfnum, dfden, size	F-distribution
`lognormalv`	mean, sigma, size	Log-normal distribution
`standard_t`	df, size	Student-t distribution

How to draw from any distribution

We define an arbitrary cumulative distribution function:

$$F(x) = \frac{1}{1 + \exp(-x)} \tag{1}$$

We quickly see that $1 > F(x) > 0$, also F is non-decreasing. Further,

$$\lim_{x \to -\infty} F(x) = 0$$
$$\lim_{x \to +\infty} F(x) = 1$$

Therefore, F satisfies all properties of a CDF. We want do draw a 10,000-large sample from this distribution.

How to draw from any distribution (2)

First, we note that the CDF returns always a number between zero and one:

$$F(x) = \frac{1}{1+\exp(-x)} = \mathbb{P}(X < x) = y \tag{2}$$

Moreover, the output of the CDF function is uniformly distributed:

$$\mathbb{P}(Y \leq y) = \mathbb{P}(F(X) < y) = \mathbb{P}\left(X < F^{-1}(y)\right) = F\left(F^{-1}(y)\right) = y$$

The algorithm becomes simple:

1. Draw a 10,000 sample from the uniform distribution (y values)
2. Compute x values as $F^{-1}(y)$:

$$x = -\log\left(\frac{1}{y} - 1\right) \tag{3}$$

Solution

Thanks to vectorization, the solution is literally two short lines of code:

```
y=npr.rand(10000)
x=-np.log(1.0/y-1)
```

Outline

Simulate Black-Scholes stock prices

Let the stock price S_t follow a geometric brownian motion with drift r and volatility σ, i.e.,

$$dS_t = r \times dt + \sigma \times dz_t \tag{4}$$

The stock price at time T, S_T, is:

$$S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right) T + \sigma\sqrt{T}z\right), \tag{5}$$

where z is a standard normal variable.

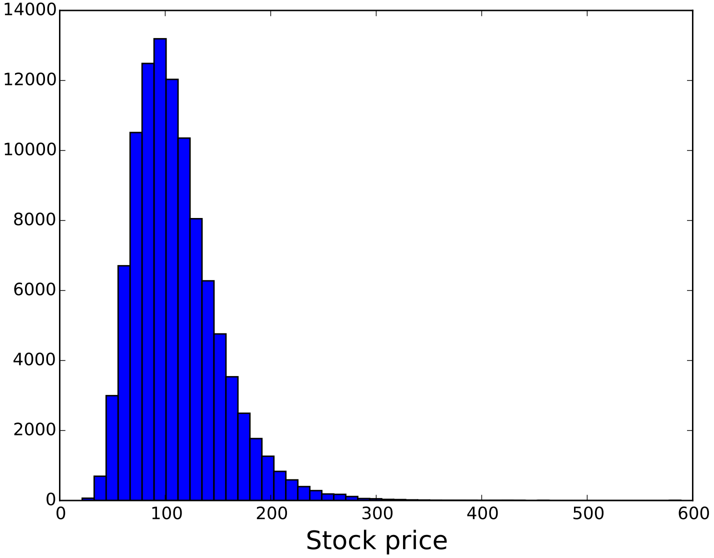
Application

Let us simulate 100,000 possible stock prices in two-years time. The current stock price is 100. The risk-free rate is 5% per annum, and the annualised volatility is 25%.

1. Plot a histogram of the simulated stock prices. What distribution do they resemble?
2. What is the mean and variance of the simulated stock prices?
3. What happens with the mean and variance if we look at a three-years window?

Solution

```
S0=100
r=0.05
sigma=0.25
T=2
I=10**5
ST1=S0*np.exp((r-0.5*sigma**2)*T+
  sigma*np.sqrt(T)*npr.randn(I))

plt.clf()
plt.hist(ST1, bins=50)
plt.xlabel("Stock␣price", fontsize=18)
```


14000
12000
10000
8000
6000
4000
2000
0
0
100
200
300
400
500
600
Stock price

Outline

Again, the Black-Scholes model..

Consider the following dynamic for the stock price:

$$dS_t = rS_t \times dt + \sigma S_t \times dZ_t \tag{6}$$

In discrete time, from $t - \Delta t$ to t, we have (almost) the exact same expression as before:

$$S_t = S_{t-\Delta t} \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}z\right), \tag{7}$$

Now though, we want a full **path** of stock prices, not just the end values.

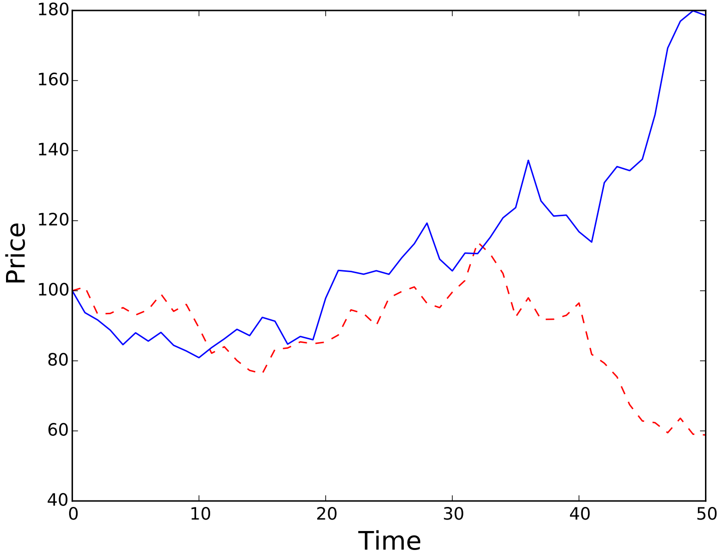
Application

Let us simulate 100,000 possible stock paths in two-years time, using 25 steps per year. The current stock price is 100. The risk-free rate is 5% per annum, and the annualised volatility is 25%.

1. Plot the first and the last path of the stock price.
2. How would the path look if the risk-free rate is 2%? or the volatility 50%?

Solution

```
S0=100
r=0.05
sigma=0.25
T=2
I=10**5
M=50
dt=np.float(T)/M
S=np.zeros((M+1,I))
S[0]=S0
for t in range(1,M+1):
        S[t]=S[t-1]*np.exp((r-0.5*sigma**2)*dt
          +sigma*np.sqrt(dt)*npr.randn(I))
```


Price
Time
180
160
140
120
100
80
60
40
0
10
20
30
40
50

Cox, Ingersoll, and Ross model

Consider the following mean-reverting dynamic for the interest rate x_t:

$$dx_t = \kappa\,(\theta - x_t) \times dt + \sigma\sqrt{x_t} \times dZ_t \tag{8}$$

The parameters: κ is the speed of mean-reversion, θ is the long-run mean, σ is the volatility parameter.

1. The discrete solution to the stochastic differential equation is a little bit more involved.
2. One nice property of this model is that the value of x always remains positive (e.g., the case of interest rates).
3. In the discrete time version, this may not hold always – so we need to adjust!

Let $s = t - \Delta t$ and $x^+ = \max(x, 0)$.

CIR process discretization

$$\tilde{x}_t = \tilde{x}_{t-\Delta} + \kappa \left(\theta - \tilde{x}_{t-\Delta}^{+}\right) \Delta t + \sigma \sqrt{\tilde{x}_{t-\Delta}^{+}} \sqrt{\Delta t} z_t$$

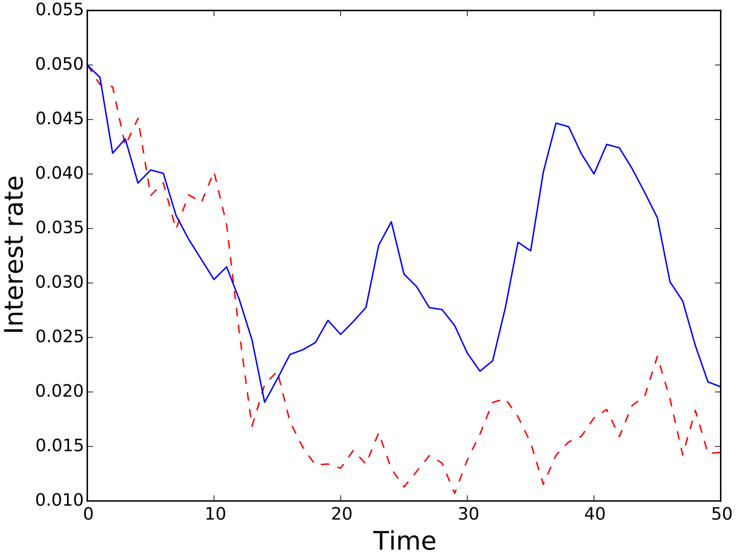
$$x_t = \tilde{x}_t^{+}$$

Application

Let us simulate 100,000 possible interest paths in two-years time, using 25 steps per year. The current interest rate is 5%.
The other parameters are: $\kappa = 3$, $\theta = 0.02$, $\sigma = 0.1$.

Solution

```
x0=0.05
kappa=3.0
theta=0.02
sigma=0.1

xh=np.zeros((M+1,I))
x1=np.zeros_like(xh)
xh[0]=x0
x1[0]=x0

for t in range(1,M+1):
    xh[t]=xh[t-1]+
        kappa*(theta-np.maximum(xh[t-1],0))*dt+
        sigma*np.sqrt(np.maximum(xh[t-1],0))*
        np.sqrt(dt)*npr.randn(I)
    x1=np.maximum(xh,0)
```


0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
Interest rate
0
10
20
30
40
50
Time

The Heston stochastic volatility model

Consider the following dynamic of the stock price:

$$dS_t = rS_t \times dt + \sqrt{v_t} S_t \times dZ_t^1, \tag{9}$$

and the following mean-reverting process for the volatility:

$$dv_t = \kappa \left(\theta - v_t\right) \times dt + \sigma \sqrt{v_t} \times dZ_t^2, \tag{10}$$

where the two Brownian motions are correlated, $dZ_t^1 dZ_t^2 = \rho dt$.

To generate two normal variables, z_1 and z_2, with correlation ρ:

1. Draw x_1 and x_2 independent normal variables.
2. Compute:
$$z_2 = \rho x_1 + \sqrt{1-\rho^2} x_2 \tag{11}$$
3. Let $z_1 = x_1$. Done!

Application

Let us simulate 100,000 possible volatility and stock price paths in two-years time, using 25 steps per year. Parameters:

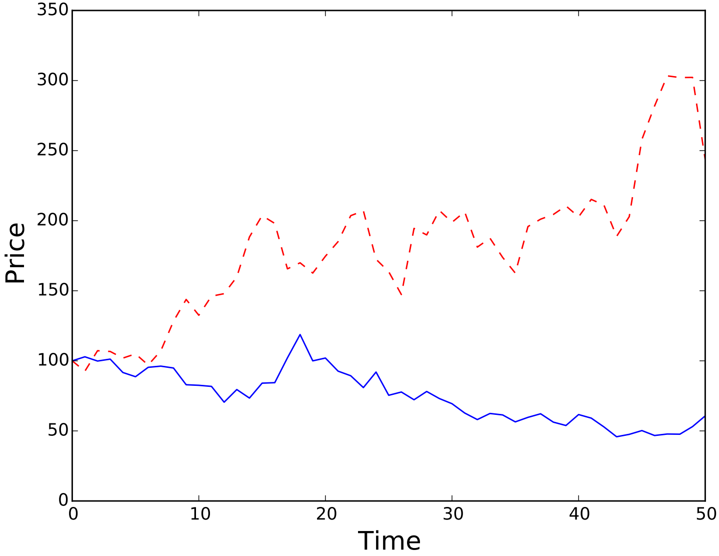
```
M=50
I=100000
S0=100
r=0.05
v0=0.1
kappa=3.0
theta=0.25
sigma=0.1
rho=0.6
T=2.0
```

Solution (1)

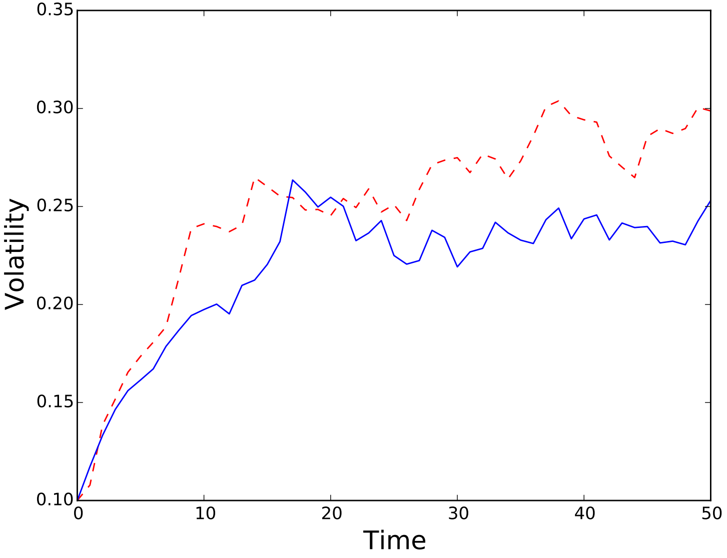
```python
dt=T/M
vh=np.zeros((M+1,I))
v1=np.zeros_like(vh)
vh[0]=v0
v1[0]=v0

z1=npr.randn(M+1,I)

for t in range(1,M+1):
 vh[t]=vh[t-1]+kappa*
     (theta-np.maximum(vh[t-1],0))*dt
     +sigma*np.sqrt(np.maximum(vh[t-1],0))
      *np.sqrt(dt)*z1[t]
 v1=np.maximum(vh,0)
```

Solution (2)

```

z2=rho*z1+np.sqrt(1-rho**2)*npr.randn(M+1,I)

S=np.zeros((M+1,I))
S[0]=S0
for t in range(1,M+1):
    S[t]=S[t-1]*np.exp((r-0.5*sigma**2)*dt
    +np.sqrt(v1[t])*np.sqrt(dt)*z2[t])
```



Price
Time
0
50
100
150
200
250
300
350
0
10
20
30
40
50


0.35
0.30
0.25
0.20
0.15
0.10
Volatility
0
10
20
30
40
50
Time

The Merton (1976) model

Consider the following dynamic of the stock price, both including a Brownian motion and a Poisson jump process with intensity λ and mean jump size μ_J:

$$dS_t = (r - r_J) \times S_t dt + \sigma \times S_t dZ_t + J_t S_t dN_t. \tag{12}$$

1. $r_J = \lambda \left(\exp\left(\mu_J + \frac{\delta^2}{2} \right) - 1 \right)$
   is a drift correction to maintain the risk-neutral measure.
2. J_t is a jump at date t. The distribution of the jump is:

$$\log(1 + J_t) \approx \text{Normal}\left(\log(1 + \mu_J) - \frac{\delta^2}{2}, \delta^2 \right) \tag{13}$$

Discretization of the Merton model

$$S_t = S_{t-\Delta t}\left(e^{\left(r-r_J-\frac{\sigma^2}{2}\right)\Delta t+\sigma\sqrt{\Delta t}z_t^1} + \left(e^{\mu_J+\delta z_t^2} - 1\right)y_t\right)$$

There are three sources of randomness, and thus we need to generate three sets of numbers:

1. The diffusion part of the stock price (Brownian motion): z_t^1.
2. The size of the jump, using normal variable: z_t^2.
3. The timing of the jump, using Poisson variable: y_t.

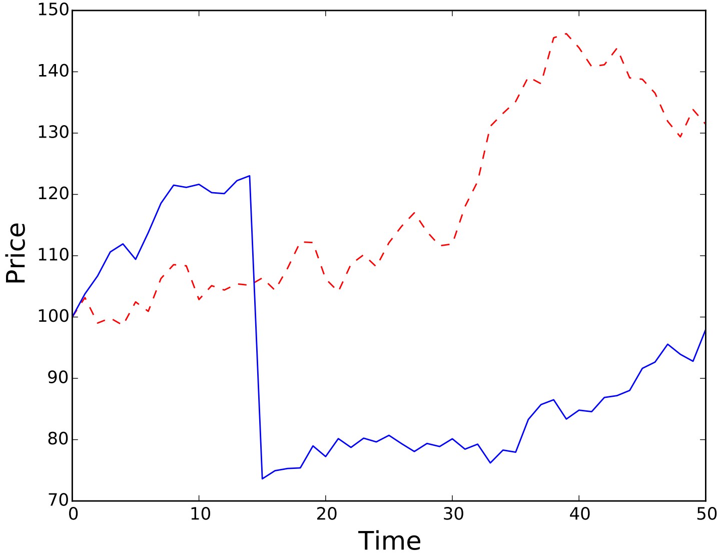
Application

Simulate the stock price from the Merton model with jumps, assuming the following parameters:

```
S0=100
r=0.05
sigma=0.2
lamb=0.75
mu=-0.6
delta=0.25
T=1.0
M=50
I=10**4
dt=T/M
```

Solution

```
rj=lamb*(np.exp(mu+0.5*delta**2)-1)
S=np.zeros((M+1,I))
S[0]=S0

z1=npr.standard_normal((M+1,I))
z2=npr.standard_normal((M+1,I))
y=npr.poisson(lamb*dt, (M+1,I))

for t in range(1,M+1):
  S[t]=S[t-1]*(np.exp((r-rj-0.5*sigma**2)*dt
  +sigma*np.sqrt(dt)*z1[t])
  +(np.exp(mu+delta*z2[t])-1)*y[t])
  S[t]=np.maximum(S[t],0)
```


Price
Time
150
140
130
120
110
100
90
80
70
0
10
20
30
40
50

Outline

Standardizing variables – mean

Let us generate 100 standard normal variables:

```
X=npr.randn(100)
```

Especially in small samples, the sample mean and variance will not be zero and one!

```
X.mean()=-0.128
X.std()=1.015
```

Mean adjustment

First, we can normalise the mean by subtracting it from each element:

```
X1=X-X.mean()
X1.mean()=5.16e-17
X1.std()=1.015
```

This solves the mean issue, not the standard deviation!

Standardizing variables – variance

Variance adjustment

Second, we can normalise the vairance by dividing each element with the standard deviation:

```
X2=X1/X.std()
X1.mean()=5.16e-17
X1.std()=1.000
```

Or, directly:

```
X3=(X-X.mean())/X.std()
```

Outline

Computing the VaR

Let us go back to the code that simulates a path of stock prices using the Geometric Brownian Motion.

1. `S[-1]` is the row of final prices for the 100,000 simulations.
2. `S[0]` is the row of initial prices.

To compute the Value-at-Risk:

- Set the desired VaR confidence level:
  `var=0.99`
- Compute returns for each stock path:
  `R_GM=(S[-1]/S[0]-1)*100`
- Use the `np.percentile` function to look at empirical quantiles:
  `np.percentile(R_GM, 100*(1-var))=-75.17`
- Interpretation: with probability 99% the loss will not exceed -75.17% of the initial stock price.

Computing Expected Shortfall

The *expected shortfall* is the expected loss conditional that we are in the “tail case,” i.e., the loss exceeds the Value-at-Risk.
In Python, it is extremely easy to compute, even without computing the VaR directly!

```
ExpShortfall=
R_GM[R_GM<np.percentile(R_GM,100*(1-var))].mean()
```

In my sample, this was -79.46%!

Application

Compute the Value-at-Risk and Expected Shortfall for the simulated Merton jump process stock price paths, with the parameters from before.
Before you start coding, how do you expect VaR and ES to be, relative to the GBM case?
What if I change a parameter: $\mu = 0.6$?
What if I change $\lambda = 2$ (and leave the old μ)?