Mock Exam — Sample Answers

REMARKS

- The exam will be a combination of questions on
  - Using Python more generally
  - Financial theory
  - Applying Python to a financial application
- The balance regarding the number of questions will be similar during the real exam.

Handing in the exam

- Do not forget to save and hand in this file in TestVision Online.

Instructions

- Answers can be worked out in this notebook or on paper. Both will be graded together.
- Efficient code is required for full marks.
- Make sure that any function you write has a docstring, and comments where appropriate.
- Before submitting your work, **click on `Kernel -> Restart & Run All`** and verify that your notebook produces the desired results and does not error.
- Some questions require you to write code to obtain a numerical result (e.g., an option price). In that case, don't just give the function, but also the result of calling it with the given parameter values (i.e., the numerial value that it returns). If your function uses random numbers, then set the seed to 0 before calling it. This makes it much easier to grade the exam (at least as long as the answer is correct).

Question 1

1. Write a docstring for the function `f(z)`, that takes the list `z` input, defined in the cell below. In other words, what does the function do?

In [1]:

```
def f(z):
    '''
    Returns the index of the first positive element in the list z.
    If all elements in z are non-positive, then it throws an error.
    '''

    j = 0

    while z[j] <= 0:
        j = j + 1

    return j
```

1. We define three arrays `a`, `b` and `c`. Please do the following:

    - Select the last three elements in `a` without using that you know that `a` is of length 6.
    - Select the elements in `a` corresponding to the elements in `b` that are equal to 0.
    - Compute a matrix `d` where each element `d[i, j] = a[i] + b[j]` without using a `for`-loop.

In [2]:

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6])
b = np.array([0, -2, 0, -3, 4, 0])
c = np.array([5, 10, -2])
```

Answer

In [3]:

```
a[-3:]
```

Out[3]:

```
array([4, 5, 6])
```

In [4]:

```
a[b<0]
```

Out[4]:

```
array([2, 4])
```

In [5]:

```
d = a[:,np.newaxis] + b
d
```

Out[5]:

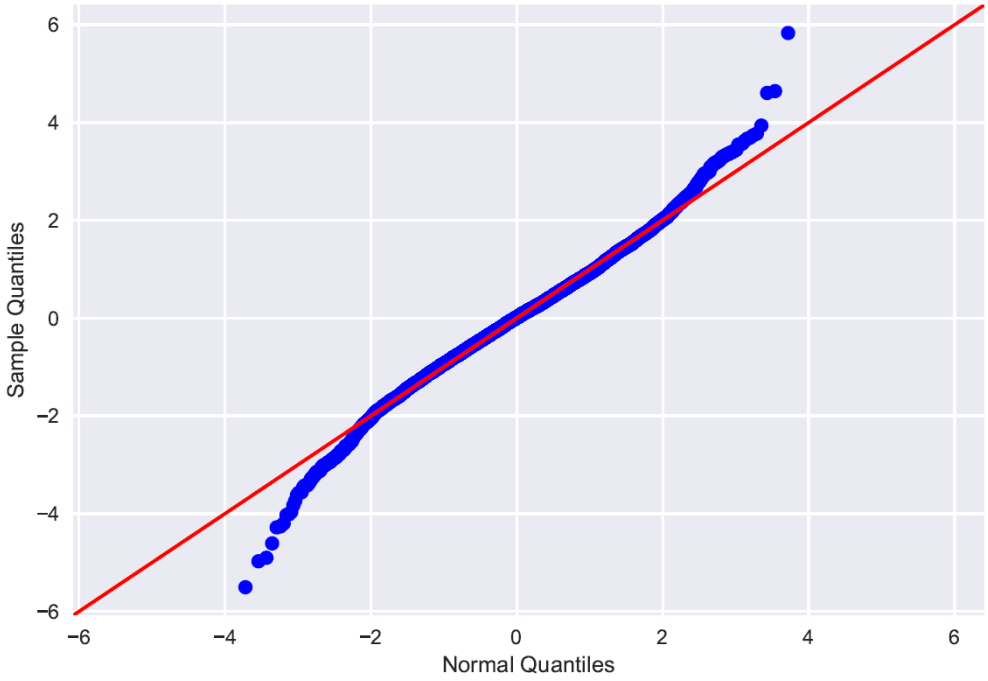
```
array([[ 1, -1,  1, -2,  5,  1],
       [ 2,  0,  2, -1,  6,  2],
       [ 3,  1,  3,  0,  7,  3],
       [ 4,  2,  4,  1,  8,  4],
       [ 5,  3,  5,  2,  9,  5],
       [ 6,  4,  6,  3, 10,  6]])
```

Question 2

You are given a set of N returns $\{R_t\}_{t=T-N+1}^{T}$.

1. Suppose that $N = 10$ and $\{R_t\} = \{-0.07, 0.87, 1.13, -0.48, 0.22, 0.94, -0.62, -0.01, 2.06, -0.08\}$. What is the historical 20\% VaR?

1. You gather some more returns and produce the graph in the figure below. Interpret it.



Answer

1. Sort the array and pick the smallest value such that at least 20\% (2 out of 10) of the values are smaller than or equal to it, i.e., -0.48 (because -0.48 and -0.62 are smaller than or equal to it). The VaR is minus that number, so 0.48 (recall that VaR should be a positive number).

1. This is a QQ plot; it plots the empirical/historical quantiles against those of a fitted parametric distribution, in this case the Normal. If the fit is good, then the points should lie on the 45 degree line. Here, the data are clearly more heavy-tailed than the normal distribution would imply. E.g., focussing on the point at the bottom left, for some p, the p\% percentile of the normal is about -4, whereas the empirical one is almost -6.

Question 3

Let

$$dX_t = \nu dt + \sigma dW_t, \quad \nu = -\frac{1}{2}\sigma^2,$$

a Brownian motion with drift.

Derive the SDE satisfied by $Y_t \equiv f(t, X_t) = tX_t$, and state whether Y_t is a martingale.

You can work out the answer on paper or in this notebook. You only get full points for a complete derivation.

Answer

$f(t, X_t) = tX_t$, so $\dot{f}(t, X_t) = X_t$, $f'(t, X_t) = t$, $f''(t, X_t) = 0$, and

$$\begin{aligned} dY_t \equiv df(t, X_t) &= \dot{f}(t, X_t)dt + f'(t, X_t)dX_t + \frac{1}{2}f''(t, X_t)\sigma^2 dt \\ &= X_t dt + t dX_t + 0 \\ &= X_t dt + t(\nu dt + \sigma dW_t) \\ &= (X_t + \nu t)dt + \sigma t dW_t, \\ &= (X_t - \frac{1}{2}\sigma^2 t)dt + \sigma t dW_t, \end{aligned}$$

which has a non-zero drift and thus isn't a martingale.

Question 4

Suppose that we are in a world where there is stock, a bond and a shout option. The shout option is an option where the holder can "shout" at one time during the lifetime of the option. If the holder shouts, then the payoff will be the either the payoff of the call option using the final value, or the value at time of shouting. At maturity, the payoff is then the largest of (i) the payoff of the usual option $\max(0, S_T - K)$ or (ii) the payoff with the price at the time of shouting $\max(0, S_\tau - K)$, where S_τ is the price at the time of shouting.

1. Show that *if the holder shouts* at time τ, the payoff at time T can be written as:
$$\max(0, S_T - S_\tau) - (S_\tau - K) \qquad (\dagger)$$

1. Use the result in equation (†) to write a function `shouttree(S0, K, T, r, sigma, N)`, which will return the estimate of the price of the shout option, where τ can be any period in the lifetime of the option. Use the following parameter values: $S_0 = 12$, $K = 11$, $T = 10$, $r = 0.03$, $\sigma = 0.4$, $N = 500$.

1. What if we restrict the period during which the holder can shout. Explain how this will impact its value.

Answer

1. We know that the payoff if the holder shouts at time τ is the maximum of $\max(0, S_T - K)$ and $\max(0, S_\tau - K)$. We can write this as

$$\begin{aligned}\max\left(\max(0, S_T - K), \max(0, S_\tau - K)\right) &= \max(S_T - K, S_\tau - K, 0) \\ &= \max(S_T - K, S_\tau - K) \\ &= \max(S_T - K - (S_\tau - K), 0) - (S_\tau - K) \\ &= \max(0, S_T - S_\tau) - (S_\tau - K)\end{aligned}$$

   where we use that $S_\tau - K > 0$ because otherwise the holder would not shout.

1. See the code below. At each node, we need to determine if we want to shout or not. If we do not shout, the value is equal to a regular call option. If we do shout, the value is equal to the expression in equation (†): a combination of a call option with strike price S_τ and cash value $S_\tau - K$. The pricing is then done similar to an American option, where we compute the value of shouting at each node and compare it to the value of working backwards through the tree.

   Note that the pricing of the call option with strike price S_τ can be done using a tree, but here we opt for the Black-Scholes formula to make the code a bit faster and easier to read since we have to compute it at each node. Both alternatives would receive full points.

1. If we restrict the period during which the holder can shout, the holder loses flexibility which is valuable. So we would expect the shout option's value to decrease.

In [6]:

```
import numpy as np
from scipy.stats import norm
```

In [7]:

```
def blackscholes(S0, K, T, r, sigma):
    """
    Price of a European call in the Black-Scholes model.
    """
    d1 = (np.log(S0)-np.log(K)+(r+sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1-sigma*np.sqrt(T)
    return S0*norm.cdf(d1)-np.exp(-r*T)*K*norm.cdf(d2)
```

In [8]:

```
def calltree_shout(S0, K, T, r, sigma, N):
    """Call price of a shout option based on an N-step binomial tree."""
    deltaT = T / float(N)
    u = np.exp(sigma*np.sqrt(deltaT))
    d = 1.0 / u
    p = (np.exp(r*deltaT)-d) / (u-d)
    piu = np.exp(-r*deltaT) * p
    pid = np.exp(-r*deltaT) * (1-p)
    SH = np.zeros((N+1, N+1)) # Define matrix
    S = S0 * u**np.arange(N+1) * d**(2*np.arange(N+1)[:, np.newaxis])
    S = np.triu(S)  #keep only the upper triangular part
    SH[:, N] = np.maximum(0, S[:, N]-K)  # Start with payoff if no shouting
    for j in range(N-1, -1, -1):
        # Discounted expected value under risk neutral measure
        SH[:j+1, j] = piu * SH[:j+1, j+1] + pid * SH[1:j+2, j+1]

        # Shouting value
        # Present value at time tau of two parts:
        # (i) European option with strike price S_tau
        C = blackscholes(S0, S[:j+1,j], T, r, sigma) # Compute for all nodes in this co
lumn/time period
        # plus (ii) Cash value of getting S_tau-K at time T, to present value (times e^
(r tau))
        B = np.exp(-r*T)*(S[:j+1,j]-K)
        # Present value (at time tau = deltaT * j)
        shoutval = np.exp(r*deltaT*j)*(C + B)

        # Compare to value of shouting.
        SH[:j+1, j] = np.maximum(SH[:j+1, j], shoutval)

    return  SH[0, 0]
```

In [9]:

```
S0=12; K=11; T=10; r=0.03; sigma=0.4; N=3; M = int(N/2);
SH = calltree_shout(S0, K, T, r, sigma, N)
SH
```

Out[9]:

```
10.235203140305945
```