Solutions to Practice Questions

1. (a) Broadcasting works as follows: counting from the back, the dimensions of the arrays match if they either agree, are equal to 1, or absent. In particular:

   i. The dimensions are `(2,)` and `(1,)`. The second array gets expanded to `np.array([7, 7])`, and the result is `np.array([8, 9])`.

   ii. The dimensions are `(2, 2)` and `(2,)`. The second array gets expanded to `np.array([7, 8], [7, 8])`, and the result is `np.array([[ 8, 10], [10, 12]])`.

   iii. The dimensions are `(2, 2)` and `(3,)`. The second dimension does not match (2 and 3). This errors.

   iv. The dimensions are `(3, 2)` and `(3, 1)`. The second array gets expanded to `np.array([[ 7, 7], [8, 8], [9, 9]])`, and the result is `np.array([[ 8, 9], [11, 12], [14, 15]])`.

   (b) The function loops over the elements of `b` and subtracts them, in turn, from the corresponding elements in `a`, thereby modifying `a` in place (remember that Numpy arrays are *mutable*). Thus, by returning `a`, it returns the elementwise difference of the two arrays. The variable `j` acts as a counter that tells us which element of `a` needs to be modified during the current loop iteration. An alternative way to write this function would be

```
def f(a, b):
    for j, e in enumerate(b):
        a[j] -= e
    return a
```

   or, in vectorized form[1]

```
def f(a, b):
    a[:] -= b[:]
    return a
```

   In particular:

   i. Returns `np.array([0, 1])`.

   ii. Returns `np.array([0, -1])`. Remember that these are *positional arguments*: the names of the variables that get passed don't matter. Inside the function, `a` refers to the first argument (which happened to be called `b` outside of the function), and `b` refers to the second argument (which happened to be called `a` outside of the function).

   iii. Returns `np.array([-1, -3])`. Remember that Python uses call-by-object-reference; this means that modifications that a function makes to its inputs are visible to the caller. Thus, by calling the function twice, the second vector (called `a` outside of the function) is subtracted from the first (called `b` outside) twice.

2. (a) 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). I would also have accepted -0.08, because before I updated them, the slides said "smaller than" rather than "smaller than or equal". The VaR is minus that number, so either 0.08 or 0.48 (recall that VaR should be a positive number).

[1] In fact the `[:]` isn't even needed; in Python, `a -= b` operates in place, unlike `a = a-b`, which creates a new array and binds it to the name `a`, hence leaving the original array untouched. I.e., had we written `a = a-b` instead, then the answer to Question 1(b)ii would be the same as the answer to Question 1(b)iii.

(b) i. Historical VaR assumes that the last N returns are representative for tomorrow. Solution: use a filtered method (e.g., filtered historical simulation based on the EWMA model).

ii. Only 20% of the data are really used to compute the VaR (note how the historical VaR doesn't change at all if we multiply all the positive returns by, e.g., 10,000). We could use a larger N, but that exacerbates the first problem. Solution: use a parametric distribution, such as the Student's t. To solve both problems, the methods can be combined (as in Assignment 1).

(c) 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.

(d)

$$\begin{aligned} VaR_{t+1}^p &= -F_\nu^{-1}(p; m, h) \\ &= -m - hF_\nu^{-1}(p; 0, 1) \\ &= -0.01 - .3F_4^{-1}(p; 0, 1) \\ &= -0.01 - .3(-3.747) = 1.114. \end{aligned}$$

3. (a) The condition is

$$\begin{aligned} \mathbb{E}^{\mathbb{Q}}\left[S_t|\mathcal{F}_{t-1}\right] = e^{r\delta t}S_{t-1} \Leftrightarrow \\ pS_{t-1}u + (1-p)S_{t-1}d = e^{r\delta t}S_{t-1} \Leftrightarrow \\ p(u-d) = e^{r\delta t} - d, \end{aligned}$$

so that $p = (e^{r\delta t} - d)/(u-d) = (\frac{5}{4} - \frac{2}{4})/(2 - \frac{1}{2}) = \frac{1}{2}$.

(b) The payoffs at maturity are $C_{uu} = 0$, $C_{ud} = 4$, $C_{du} = 0$, and $C_{dd} = 3$. Backwards induction yields $C_u = 1.6$, $C_d = 1.2$, and $C_0 = 1.12$.

4. (a) X_t is a martingale if

i. $\mathbb{E}[|X_t|] < \infty$

ii. $\mathbb{E}[X_t \mid \mathcal{F}_s] = X_s$, $s < t$.

This means that a martingale is a process without a drift; on average, we expect tomorrow's value to be equal to today's.

(b) $f(t, X_t) = \exp(X_t)$, so $\dot{f}(t, X_t) \equiv \frac{\partial}{\partial t} f(t, X_t) = 0$, $f'(t, X_t) \equiv \frac{\partial}{\partial X_t} f(t, X_t) = \exp(X_t)$, and $f''(t, X_t) \equiv \frac{\partial^2}{\partial X_t^2} f(t, X_t) = \exp(X_t)$, so by Ito's lemma,

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

an Ito process without a drift and hence a martingale.

(c) $f(t, X_t) = tX_t$, 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.

5. (a) The process could be discretized by the Euler approximation

$$S_i = S_{i-1} + \mu_t \delta t + \sigma_t \sqrt{\delta t} Z_i,$$

where $Z_i \overset{i.i.d.}{\sim} N(0, 1)$. Given pseudo random numbers for Z_i, this can be iterated to give a path for S_t, including S_T and hence $C_T(S_T)$. Repeating this n times with independent random numbers produces n paths.

(b) By the C.L.T.,

$$\sqrt{n}(\bar{X}_n - \theta) \xrightarrow{d} N\left(0, \sigma^2\right), \quad \sigma^2 = \mathrm{var}[X].$$

Hence,

$$\begin{aligned} \mathbb{P}[-1.96\sigma \leq \sqrt{n}(\bar{X}_n - \theta) \leq 1.96\sigma] = 0.95 \Leftrightarrow \\ \mathbb{P}[\bar{X}_n - 1.96\frac{\sigma}{\sqrt{n}} \leq \theta \leq \bar{X}_n + 1.96\frac{\sigma}{\sqrt{n}}] = 0.95. \end{aligned}$$

Hence $c_l = \bar{X}_n - 1.96\frac{\sigma}{\sqrt{n}}$ and $c_u = \bar{X}_n + 1.96\frac{\sigma}{\sqrt{n}}$ is an asymptotically valid CI. σ^2 is unknown, but we can estimate it as

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \bar{X}_n\right)^2.$$