# Chapter 4

# Statistics by Simulation (solutions to exercises)

# Contents

## Import Python packages

```
# Import all needed python packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as stats
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

## 4.1 Reliability: System lifetime (simulation as a computation tool)

### |||| Exercise 4.1 Reliability: System lifetime (simulation as a computation tool)

A system consists of three components A, B and C serially connected, such that A is positioned before B, which is again positioned before C. The system will be functioning only so long as A, B and C are all functioning. The lifetime in months of the three components are assumed to follow exponential distributions with means: 2 months, 3 months and 5 months, respectively (hence there are three random variables, $X_A$, $X_B$ and $X_C$ with exponential distributions with $\lambda_A = 1/2$, $\lambda_B = 1/3$ and $\lambda_C = 1/5$ resp.). A little Python-help: You will probably need (or at least it would help) to put three variables together to make e.g. a $k \times 3$-matrix – this can be done by the `cbind` function:

```
x = np.column_stack((xA,xB,xC))
```

And just as an example, in Python we can easily compute e.g. the mean of the three values for each of all the $k$ rows of this matrix by using the "axis" argument in the `np.mean` function. This argument specifies the axis along which the means are computed, so for axis=1, we take the mean for the three columns in the x. Many functions in Python have this argument, so it is a good idea to get familiar with it. Example for mean:

```
simmeans = np.mean(x, axis=1)
```

a) Generate, by simulation, a large number (at least 1000 – go for 10000 or 100000 if your computer is up for it) of system lifetimes (hint: consider how the random variable $Y$ = System lifetime is a function of the three $X$-variables: is it the sum, the mean, the median, the minimum, the maximum, the range or something even different?).

## |||| Solution

Note that the lifetime can be seen as the minimal value of the three random component lifetimes:

$$\text{"Lifetime"} = \min(X_A, X_B, X_C).$$

First, note that the generated solution below has been generated with this seed in order to get the same result each time. Note, that when a simulation analysis is carried out, this number should only be set once and set randomly (potentially it is possible to find a seed (see Remark 2.12) that gives a rare simulation result and thus showing a "wrong" result, however if $k$ is high enough this is very hard). The solution below has been generated with the following seed

```
## You might want to set the seed to achieve a particular result
np.random.seed(82719)
```

The following Python-code generates 10.000 simulated system lifetimes:

```
# Number of simulations
k = 10000
# Generating k component A lifetimes
xA = stats.expon.rvs(scale=2, size=k)
# Checking the mean of these
print(np.mean(xA))
```

```
2.0035078916253752
```

```
# Generating k component B lifetimes
xB = stats.expon.rvs(scale=3, size=k)
# Checking the mean of these
print(np.mean(xB))
```

```
3.0264519724715777
```

```
# Generating k component C lifetimes
xC = stats.expon.rvs(scale=5, size=k)
# Checking the mean of these
print(np.mean(xC))
```

```
5.047803906911606
```

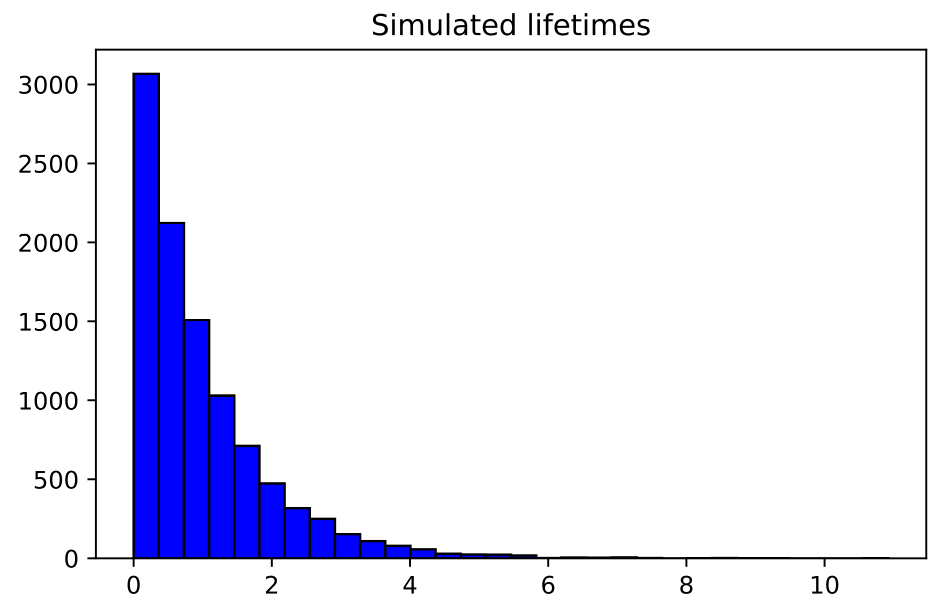
```
# Putting these three sets of k lifetimes together
# into a single k-by-3 matrix
x = np.column_stack((xA, xB, xC))

# Finding the minimum value of the three components in each of the k situations
lifetimes = np.min(x, axis=1)
```

## |||| Solution

Let us have a look at these simulated lifetimes:

```
## Histogram of the simulated lifetimes
plt.hist(lifetimes, bins=30, color='blue', edgecolor='black')
plt.title('Simulated lifetimes')
plt.show()
```



b) Estimate the mean system lifetime.

|||| **Solution**

```
## The estimated mean lifetime
print(np.mean(lifetimes))
```

```
0.9880015386365172
```

c) Estimate the standard deviation of system lifetimes.

|||| **Solution**

```
## The estimated std. dev. of the lifetime
print(np.std(lifetimes,ddof=1))
```

```
0.9959225338611547
```

d) Estimate the probability that the system fails within 1 month.

#### |||| Solution

We need to count how often the lifetimes are smaller than or equal to 1 month – this can in Python be achieved by use of a logical operator:

```
## The fraction of times the simulated lifetime was below or equal 1
print(np.mean(lifetimes <= 1))


0.6391


# or
print(np.sum(lifetimes <= 1)/k)


0.6391
```

In Python `FALSE` is a 0 and a `TRUE` is a 1 - this is why we can simply apply the `mean` function directly on the vector of TRUES and FALSES like this.

e) Estimate the median system lifetime

#### |||| Solution

```
## The estimated median lifetime
print(np.median(lifetimes))


0.6878542614451192
```

f) Estimate the 10th percentile of system lifetimes

### |||| Solution

```
## The estimated 10% quantile
print(np.quantile(lifetimes, 0.1))


0.10367217182567331
```

g) What seems to be the distribution of system lifetimes? (histogram etc)

### |||| Solution

We already made the histogram above. It appears that the minimum of the three exponential variables also has a distribution that looks like an exponential. In fact, there is a theoretical result (beoynd the syllabus of this course) that states that the distribution of the minimum of these three exponential distributions is again an exponential distribution but now with

$$\lambda_{min} = \lambda_A + \lambda_B + \lambda_C = 1/2 + 1/3 + 1/5 = 31/30.$$

Note how this matches nicely with the found mean above!

## 4.2 Basic bootstrap CI

### |||| Exercise 4.2 Basic bootstrap CI

(Can be handled without using R) The following measurements were given for the cylindrical compressive strength (in MPa) for 11 prestressed concrete beams:

$$38.43, 38.43, 38.39, 38.83, 38.45, 38.35, 38.43, 38.31, 38.32, 38.48, 38.50.$$

1000 bootstrap samples (each sample hence consisting of 11 measurements) were generated from these data, and the 1000 bootstrap means were arranged on order. Refer to the smallest as $\bar{x}^*_{(1)}$, the second smallest as $\bar{x}^*_{(2)}$ and so on, with the largest being $\bar{x}^*_{(1000)}$. Assume that

$$\begin{aligned}
\bar{x}^*_{(25)} &= 38.3818, \\
\bar{x}^*_{(26)} &= 38.3818, \\
\bar{x}^*_{(50)} &= 38.3909, \\
\bar{x}^*_{(51)} &= 38.3918, \\
\bar{x}^*_{(950)} &= 38.5218, \\
\bar{x}^*_{(951)} &= 38.5236, \\
\bar{x}^*_{(975)} &= 38.5382, \\
\bar{x}^*_{(976)} &= 38.5391.
\end{aligned}$$

a) Compute a 95% bootstrap confidence interval for the mean compressive strength.

### |||| Solution

Looking at Method box 4.10, we see that we need to find the 2.5%, and 97.5% quantiles of the 1000 bootstrap samples. According to the rule for finding the 2.5% quantile this should be the average of the 25th andn the 26th observation:

$$q_{0.025} = \frac{\bar{x}^*_{(25)} + \bar{x}^*_{(26)}}{2} = 38.3818,$$

and similarly

$$q_{0.975} = \frac{\bar{x}^*_{(975)} + \bar{x}^*_{(976)}}{2} = \frac{38.5382 + 38.5391}{2} = 38.5387,$$

and hence the 95% bootstrap confidence band is:

$$[38.3818; 38.5387].$$

b) Compute a 90% bootstrap confidence interval for the mean compressive strength.

## |||| Solution

As above we get:

$$q_{0.05} = \frac{\bar{x}^*_{(50)} + \bar{x}^*_{(51)}}{2} = \frac{38.3909 + 38.3919}{2} = 38.3914,$$

and similarly:

$$q_{0.95} = \frac{\bar{x}^*_{(950)} + \bar{x}^*_{(951)}}{2} = \frac{38.5218 + 38.5236}{2} = 38.5227,$$

and hence the 90% bootstrap confidence band is:

$$[38.3914; 38.5227].$$

## 4.3 Various bootstrap CIs

### |||| Exercise 4.3 Various bootstrap CIs

Consider the data from the exercise above. These data are entered into Python as:

```
x = np.array([38.43, 38.43, 38.39, 38.83, 38.45, 38.35,
        38.43, 38.31, 38.32, 38.48, 38.50])
```

Now generate $k = 1000$ bootstrap samples and compute the 1000 means (go higher if your computer is fine with it)

a) What are the 2.5%, and 97.5% quantiles (so what is the 95% confidence interval for $\mu$ without assuming any distribution)?

### |||| Solution

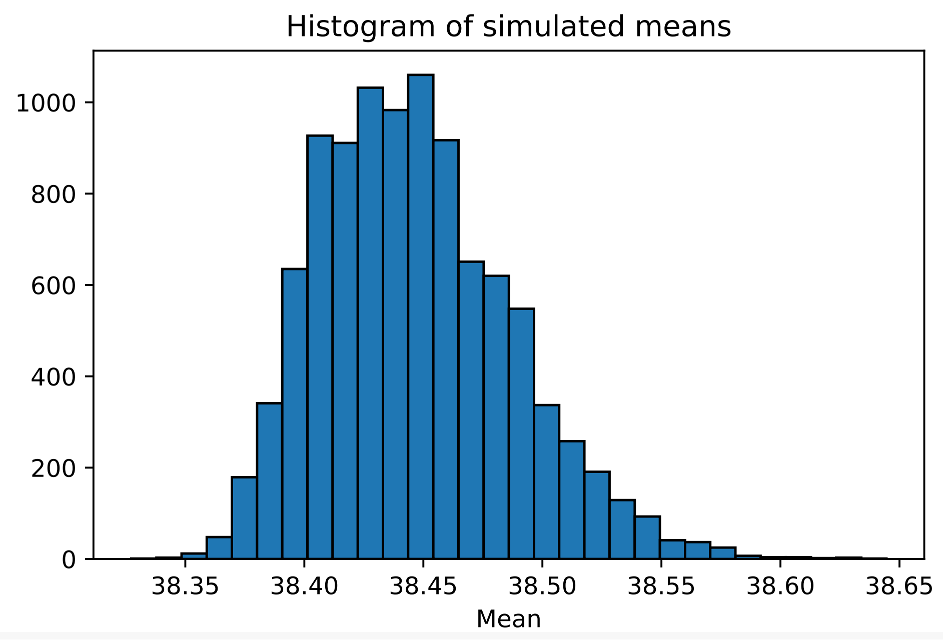
The solution below has been generated with the following seed (see Remark 2.12)

```
## You might want to set the seed to achieve a particular result
np.random.seed(6287)
```

```
x = np.array([38.43, 38.43, 38.39, 38.83, 38.45, 38.35,
       38.43, 38.31, 38.32, 38.48, 38.50])
k = 10000
n = len(x)
simsamples = np.random.choice(x, (n, k), replace=True)
simmeans = np.mean(simsamples, axis=0)
print(np.quantile(simmeans, [0.025, 0.975]))
```

```
[38.381 38.536]
```

```
plt.hist(simmeans, bins=30, edgecolor = 'black')
plt.title('Histogram of simulated means')
plt.xlabel('Mean')
plt.show()
```



b) Find the 95% confidence interval for $\mu$ by the parametric bootstrap assuming the normal distribution for the observations. Compare with the classical analytic approach based on the $t$-distribution from Chapter 2.

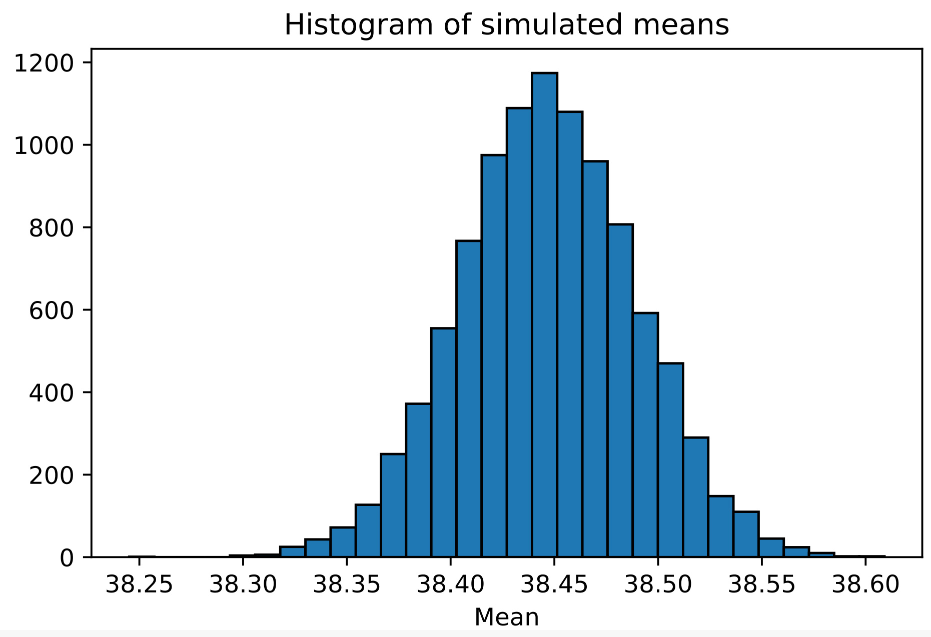
## |||| Solution

First we do the parametric bootstrap:

```
k = 10000
n = len(x)
simsamples = stats.norm.rvs(loc=np.mean(x), scale=np.std(x,ddof=1), size=(n, k))
simmeans = np.mean(simsamples, axis=0)
print(np.quantile(simmeans, [0.025, 0.975]))
```

```
[38.363 38.530]
```

```
# Histogram
plt.hist(simmeans, bins=30, edgecolor = 'black')
plt.title('Histogram of simulated means')
plt.xlabel('Mean')
plt.show()
```



And the classic *t*-based approach (without simulation):

```
t_stat,p_val = stats.ttest_1samp(x, 38.5)
print(t_stat)


-1.239610578766898


print(p_val)


0.24342150717016434


# interval directly
(CI_low,CI_high) = stats.t.interval(0.95, len(x)-1, loc=np.mean(x), scale=stats.sem(x))
print(CI_low,CI_high)


38.35249805615088 38.54204739839457
```

c) Find the 95% confidence interval for $\mu$ by the parametric bootstrap assuming the log-normal distribution for the observations. (Help: To use the `stats.lognorm.rvs` function to simulate the log-normal distribution, we face the challenge that we need to specify the mean and standard deviation on the log-scale and not on the raw scale, so compute mean and standard deviation for log-transformed data for this Python-function)

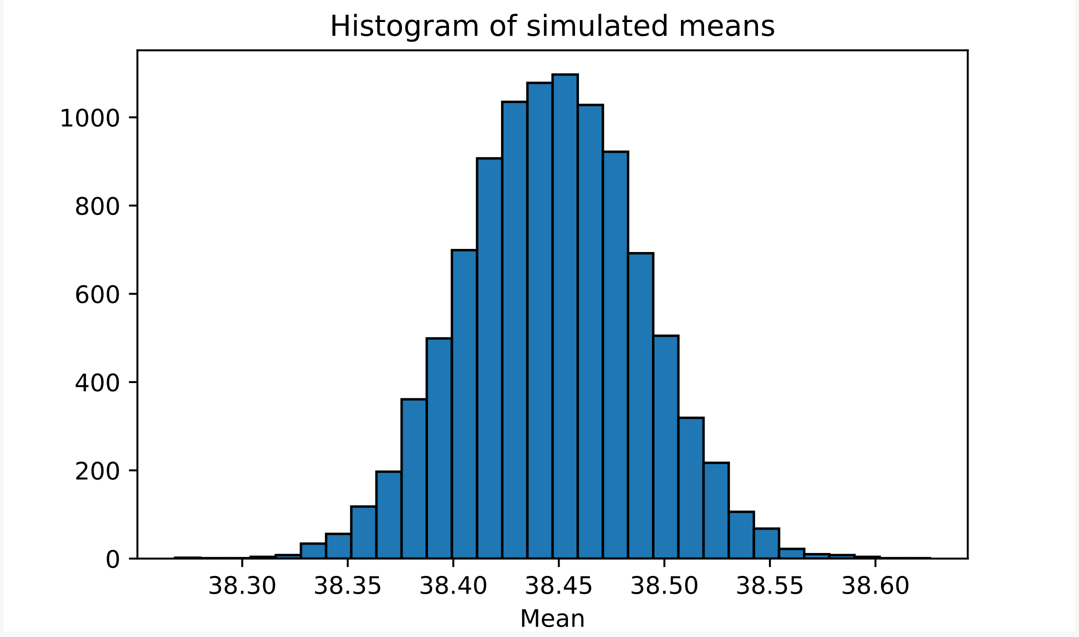
## Solution

We do the parametric bootstrap using the log-normal distribution.

```
k = 10000
n = len(x)
simsamples = stats.lognorm.rvs(s=np.std(np.log(x), ddof=1), scale=np.exp(np.mean(np.log
simmeans = np.mean(simsamples, axis=0)
print(np.quantile(simmeans, [0.025, 0.975]))
```

```
[38.365 38.529]
```

```
# Histogram
plt.hist(simmeans, bins=30, edgecolor = 'black')
plt.title('Histogram of simulated means')
plt.xlabel('Mean')
plt.show()
```



d) Find the 95% confidence interval for the lower quartile $Q_1$ by the parametric bootstrap assuming the normal distribution for the observations.

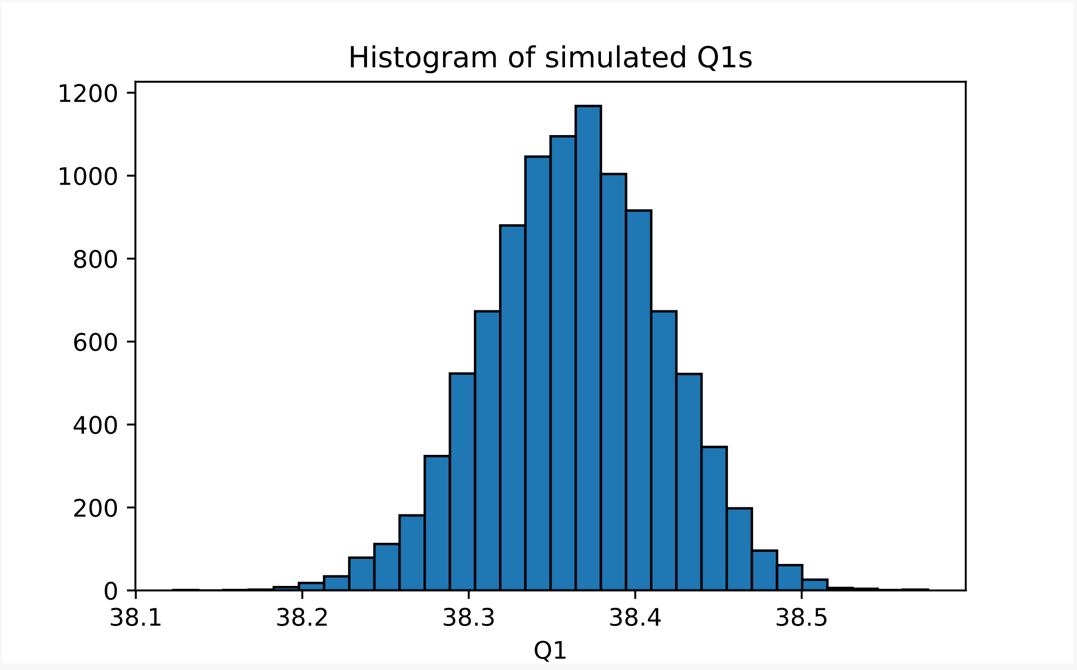
## |||| Solution

We do the parametric bootstrap of lower quartile $Q_1$ using the normal distribution by first making a $Q1$-function in Python, and then the usual stuff:

```
k = 10000
n = len(x)
simsamples = stats.norm.rvs(loc=np.mean(x), scale=np.std(x,ddof=1), size=(n, k))
simQ1s = np.quantile(simsamples, 0.25, axis=0)
print(np.quantile(simQ1s, [0.025, 0.975]))
```

```
[38.258 38.466]
```

```
# Histogram
plt.hist(simQ1s, bins=30, edgecolor = 'black')
plt.title('Histogram of simulated Q1s')
plt.xlabel('Q1')
plt.show()
```



e) Find the 95% confidence interval for the lower quartile $Q_1$ by the non-parametric bootstrap (so without any distributional assumptions)

### |||| Solution

We simply substitute the sampling line with the non-parametric version:

```
k = 10000
n = len(x)
simsamples = np.random.choice(x, (n, k), replace=True)
simQ1s = np.quantile(simsamples, 0.25, axis=0)
print(np.quantile(simQ1s, [0.025, 0.975]))
```

```
[38.315 38.430]
```

## 4.4 Two-sample TV data

### Exercise 4.4 Two-sample TV data

A TV producer had 20 consumers evaluate the quality of two different TV flat screens - 10 consumers for each screen. A scale from 1 (worst) up to 5 (best) were used and the following results were obtained:

| TV screen 1 | TV screen 2 |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 1 | 2 |
| 3 | 4 |
| 2 | 2 |
| 1 | 3 |
| 2 | 2 |
| 3 | 4 |
| 1 | 3 |
| 1 | 2 |

a) Compare the two means without assuming any distribution for the two samples (non-parametric bootstrap confidence interval and relevant hypothesis test interpretation).

## |||| Solution

The solution below has been generated with the following seed (see Remark 2.12)

```
## You might want to set the seed to achieve a particular result
np.random.seed(98273)
```

```
x1 = np.array([1, 2, 1, 3, 2, 1, 2, 3, 1, 1])
x2 = np.array([3, 4, 2, 4, 2, 3, 2, 4, 3, 2])
## Number of simulated (bootstrapped) samples
k = 10000
n = len(x1) # same as len(x2)
## Simulated samples of TV1 and TV2 groups
simx1samples = np.random.choice(x1, (n, k), replace=True)
simx2samples = np.random.choice(x2, (n, k), replace=True)
simmeandifs = np.mean(simx1samples, axis=0) - np.mean(simx2samples, axis=0)
# Confidence interval
ci = np.quantile(simmeandifs, [0.025, 0.975])
print(ci)
```

```
[-1.900 -0.500]
```

We reject the null hypothesis of $\mu_1 = \mu_2$, since zero is not included in the CI of the differences.

b) Compare the two means assuming normal distributions for the two samples - without using simulations (or rather: assuming/hoping that the sample sizes are large enough to make the results approximately valid).

#### |||| Solution

```
t_stat, p_val = stats.ttest_ind(x1, x2)
print(t_stat)
```

```
-3.157408869505305
```

```
print(p_val)
```

```
0.005449057981469947
```

We reject the null hypothesis of $\mu_1 = \mu_2$.

c) Compare the two means assuming normal distributions for the two samples - simulation based (parametric bootstrap confidence interval and relevant hypothesis test interpretation – in spite of the obviously wrong assumption).

#### |||| Solution

```
simx1samples = stats.norm.rvs(loc=np.mean(x1), scale=np.std(x1,ddof=1), size=(n, k))
simx2samples = stats.norm.rvs(loc=np.mean(x2), scale=np.std(x2,ddof=1), size=(n, k))
simmeandifs = np.mean(simx1samples, axis=0) - np.mean(simx2samples, axis=0)
# Confidence interval
print(np.quantile(simmeandifs, [0.025, 0.975]))
```

```
[-1.948 -0.442]
```

We reject the null hypothesis of $\mu_1 = \mu_2$.

## 4.5 Non-linear error propagation

### |||| Exercise 4.5 Non-linear error propagation

The pressure $P$, and the volume $V$ of one mole of an ideal gas are related by the equation $PV = 8.31T$, when $P$ is measured in kilopascals, $T$ is measured in kelvins, and $V$ is measured in liters.

a) Assume that $P$ is measured to be 240.48 kPa and $V$ to be 9.987 L with known measurement errors (given as standard deviations): 0.03 kPa and 0.002 L. Estimate $T$ and find the uncertainty in the estimate.

### |||| Solution

This is a almost direct copy of the rectangle example ($A = XY$) (Example 4.5), since $T = PV/8.31$, so since: To use the approximate error propagation rule, we must differentiate the function $f(x, y) = xy/8.31$ with respect to both $x$ and $y$:

$$\frac{\partial f}{\partial x} = y/8.31 \quad \frac{\partial f}{\partial y} = x/8.31.$$

We get the result: $\hat{T} = 240.48 \cdot 9.987/8.31 = 289.0101$, and the uncertainty is:

$$\sigma_{\hat{T}} = \sqrt{9.987^2 \times 0.03^2 + 240.48^2 \times 0.002^2}/8.31 = 0.0682.$$

b) Assume that $P$ is measured to be 240.48kPa and $T$ to be 289.12K with known measurement errors (given as standard deviations): 0.03kPa and 0.02K. Estimate $V$ and find the uncertainty in the estimate.

### |||| Solution

$$V = f(P, T) = 8.31T/P.$$

So:

$$\frac{\partial f}{\partial T} = 8.31/P \;\; \frac{\partial f}{\partial P} = -8.31\frac{T}{P^2},$$

and hence:

$$\hat{V} = 8.31 \cdot 289.12/240.48 = 9.9908.$$

and

$$\sigma_{\hat{V}} = 8.31\sqrt{1/240.48^2 \times 0.02^2 + 289.12^2/240.48^4 \times 0.03^2} = 0.00143.$$

c) Assume that $V$ is measured to be 9.987 L and $T$ to be 289.12 K with known measurement errors (given as standard deviations): 0.002 L and 0.02 K. Estimate $P$ and find the uncertainty in the estimate.

### |||| Solution

Since

$$P = f(V, T) = 8.31T/V,$$

we can simply change the roles of $P$ and $V$ in the above and find similarly

$$\frac{\partial f}{\partial T} = 8.31/V \;\; \frac{\partial f}{\partial V} = -8.31\frac{T}{V^2},$$

and hence

$$\hat{P} = 8.31 \cdot 289.12/9.987 = 240.5715,$$

and

$$\sigma_{\hat{P}} = 8.31\sqrt{1/9.987^2 \times 0.02^2 + 289.12^2/9.987^4 \times 0.002^2} = 0.0510.$$

d) Try to answer one or more of these questions by simulation (assume that the errors are normally distributed).

## |||| Solution

Let's look at 3. The following Python-code will do the job:

The solution below has been generated with the following seed (see Remark 2.12)

```
## You might want to set the seed to achieve a particular result
np.random.seed(28973)
```

```
k = 10000
Vs = stats.norm.rvs(loc=9.987, scale=0.002,size=k)
Ts = stats.norm.rvs(loc=289.12,scale=0.02, size=k)
Ps = 8.31*Ts/Vs
print(np.std(Ps, ddof=1))
```

```
0.05119900268933971
```

Rerunning this a few times will show that 0.051 is the proper result. This additional re-running gives a feeling of the error in the simulation - rather small here. Alternatively increase $k$.

Similarly 2. can be handled as:

```
k = 10000
Ps = stats.norm.rvs(loc=240.28, scale=0.03, size=k)
Ts = stats.norm.rvs(loc=289.12, scale=0.02, size=k)
Vs = 8.31*Ts/Ps
print(np.std(Vs, ddof=1))
```

```
0.0014176700426727481
```

Providing again basically the same answer as above: 0.0014.