Chapter 4

Statistics by Simulation (solutions to exercises)

Contents

Import Python packages

```python
# 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)

In this exercise we will practice to do our own simulation.

You should use Python to solve this exercise and getting help from internet and generative AI is allowed. In an exam situation you should be able to read and understand code that carries out a simulation. Therefore it is also a good idea to study the solution for this exercise (after you have made your own solution) - you do not need to make your own code in the exact same way, but it is good to practice reading and understanding code from solutions (and from the book in general).

A device comprising of three components; "A", "B" and "C", is only functioning as long as all three components are functioning individually. The lifetime (in months) of the individual components are assumed to follow exponential distributions: The average lifetime of components of type "A" is 2 months, the average lifetime of components of type "B" is 3 months and the average lifetime of components of type "C" is 5 months. Hence we can describe the lifetime of each component type with the random variables, X_A, X_B and X_C that follow exponential distributions:

$$
\begin{aligned}
X_A &\sim Exp(\lambda = 1/2) \\
X_B &\sim Exp(\lambda = 1/3) \\
X_C &\sim Exp(\lambda = 1/5)
\end{aligned}
$$

Describing the overall lifetime of devices is however not as straight forward. We will therefore simulate the situation.

a) Simulate a large number of devices (at least 1000 – go for 10000 or 100000 if your computer is up for it). For each device you will need to simulate the lifetimes of component A, B and C. Store this information in a DataFrame. In your simulation, how long does the first three devices stay functioning?

|||| Solution

First, note that the solution below has been generated with the following seed:

```
np.random.seed(82719)
```

If you want to simulate the exact same pseudo-random numbers as shown in this solution, you will need to use the same seed (and generate the random variables in the same order).

The following Python-code generates 10.000 simulated devices:

```
np.random.seed(82719)

# Number of simulations
k = 10000
# Generating k component A lifetimes
xA = stats.expon.rvs(scale=2, size=k)
# Generating k component B lifetimes
xB = stats.expon.rvs(scale=3, size=k)
# Generating k component C lifetimes
xC = stats.expon.rvs(scale=5, size=k)

# Store the simulated lifetimes in a DataFrame, where each row is one
device and each column is a simulated "measurement":
df = pd.DataFrame({
 'Lifetime_A': xA,
 'Lifetime_B': xB,
 'Lifetime_C': xC
 })

# Print the top part of your DataFrame containing simulated data:
print(df.head())

   Lifetime_A  Lifetime_B  Lifetime_C
0    0.320960    4.895210   11.179703
1    0.491043    3.006795    4.386929
2    1.608826    1.485084    0.929437
3    2.285528    4.850018    9.774913
4    2.409813    2.420582    1.164840



# From the simulated data we see that "device 0" stays functioning for
0.32 months (after which component A fails).
# We also see that "device 1" stops functioning after 0.49 months and
"device 2" stops functioning after 0.93 months.
```

b) To check our simulation, compute the average lifetime of each component type. Do these values match your expectations?

|||| Solution

```
print(df.mean())

Lifetime_A    2.003508
Lifetime_B    3.026452
Lifetime_C    5.047804
dtype: float64
```

These values match our expectations as they are very close to the theoretical population means used for the simulation (which were 2, 3 and 5 months).

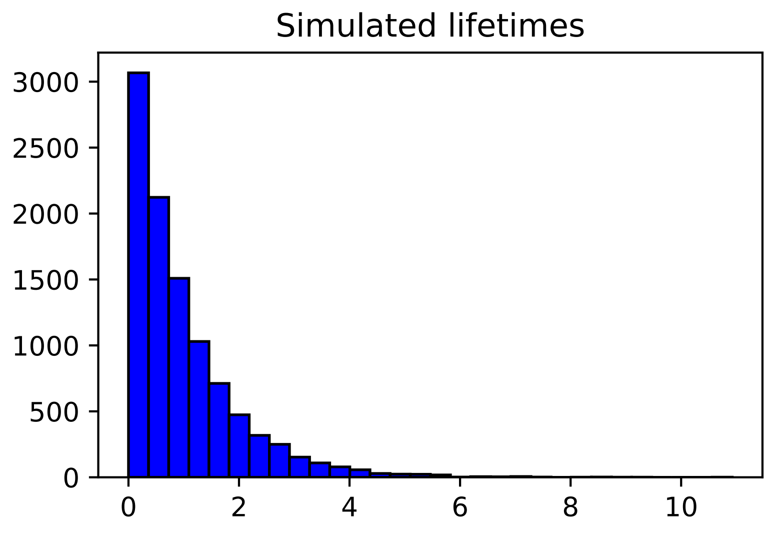
We now have a simulated dataset that we can play with. We can compute the overall lifetime for each simulated device, and investigate how this *simulated random variable* behaves:

c) For each device calculate the device lifetime and store this value in a new column in your DataFrame. Make a histogram of the device lifetimes (hence investigating the distribution of this *simulated random variable*).

   (hint: consider how the random variable Y = "device lifetime" is a function of the three component lifetimes: is it the sum, the mean, the median, the minimum, the maximum, the range or something else?)

|||| Solution

```
# Add a new column that is the minimum of the three original columns:
df['Lifetime_device'] = df[['Lifetime_A', 'Lifetime_B',
'Lifetime_C']].min(axis=1)

print(df.head())

   Lifetime_A  Lifetime_B  Lifetime_C  Lifetime_device
0    0.320960    4.895210   11.179703         0.320960
1    0.491043    3.006795    4.386929         0.491043
2    1.608826    1.485084    0.929437         0.929437
3    2.285528    4.850018    9.774913         2.285528
4    2.409813    2.420582    1.164840         1.164840


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



d) Estimate the mean device lifetime from the simulated data.

|||| Solution

```
## The estimated mean lifetime
print(df['Lifetime_device'].mean())

0.9880015386365172
```

e) Estimate the standard deviation of system lifetimes from the simulated data.

|||| Solution

```
## The estimated std. dev. of the lifetime
print(df['Lifetime_device'].std(ddof=1))

0.9959225338611547
```

f) 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:

```
# Add a new column with an "indicator vairable":
df['below_1_month'] = df['Lifetime_device'] <= 1

print(df.head())

   Lifetime_A  Lifetime_B  Lifetime_C  Lifetime_device  below_1_month
0    0.320960    4.895210   11.179703         0.320960           True
1    0.491043    3.006795    4.386929         0.491043           True
2    1.608826    1.485084    0.929437         0.929437           True
3    2.285528    4.850018    9.774913         2.285528          False
4    2.409813    2.420582    1.164840         1.164840          False

## The fraction of times the simulated lifetime was below or equal 1
print(df['below_1_month'].mean())

0.6391
```

In Python `FALSE` is 0 and `TRUE` is 1 in calculations - this is why we can simply apply the `mean` function directly on the column of `TRUE`s and `FALSE`s like this.

g) Estimate the median system lifetime from the simulated data.

|||| Solution

```
## The estimated median lifetime
print(np.median(df['Lifetime_device']))

0.6878542614451192
```

h) Estimate the 10th percentile of system lifetimes from the simulated data.

|||| Solution

```
## The estimated 10% quantile
print(np.quantile(df['Lifetime_device'], 0.1,
method='averaged_inverted_cdf'))

0.10348342692625076
```

i) 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

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

Corresponding to the average lifetime:

$$1/\lambda_{min} = 30/31 \approx 0.9677$$

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 μ without assuming any distribution)?

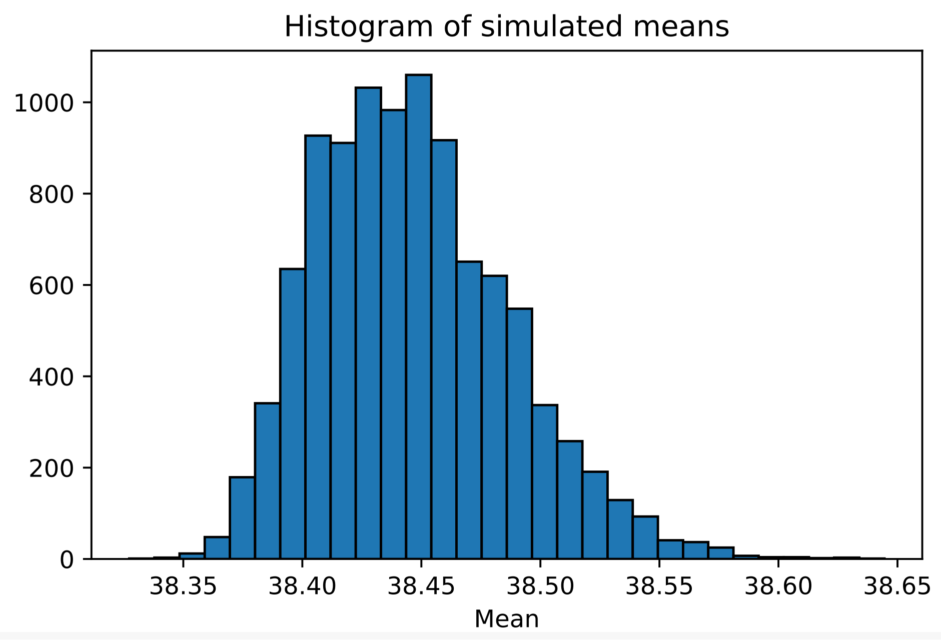
|||| Solution

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

```
## 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]))

[38.38090909 38.53636364]
```

```
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 μ by the parametric bootstrap assuming the normal distribution for the observations. Compare with the classical analytic approach based on the t-distribution from Chapter **??**.

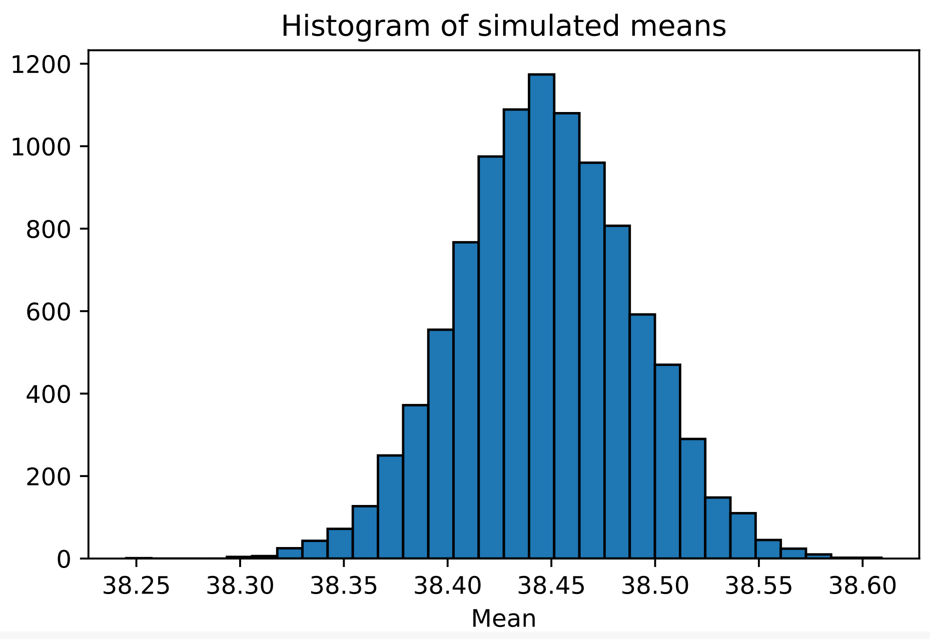
Solution

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]))
```

```
[38.36349979 38.53043896]
```

```
# 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 μ 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)

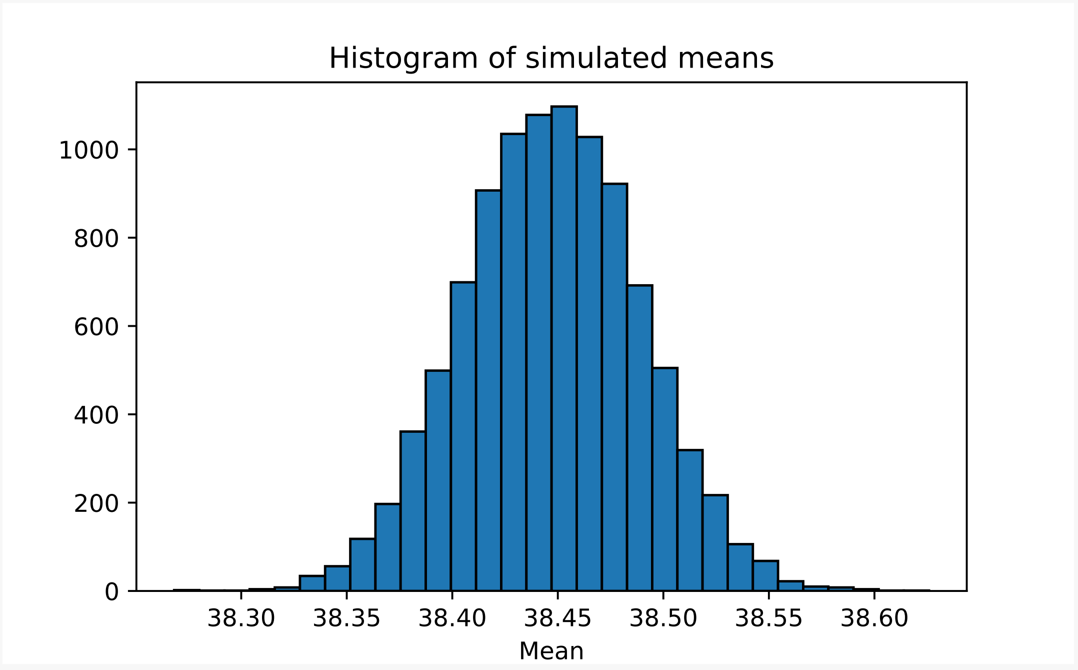
Solution

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(x))), size=(n, k))
simmeans = np.mean(simsamples, axis=0)
print(np.quantile(simmeans, [0.025, 0.975]))
```

```
[38.36528859 38.52855203]
```

```
# 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.

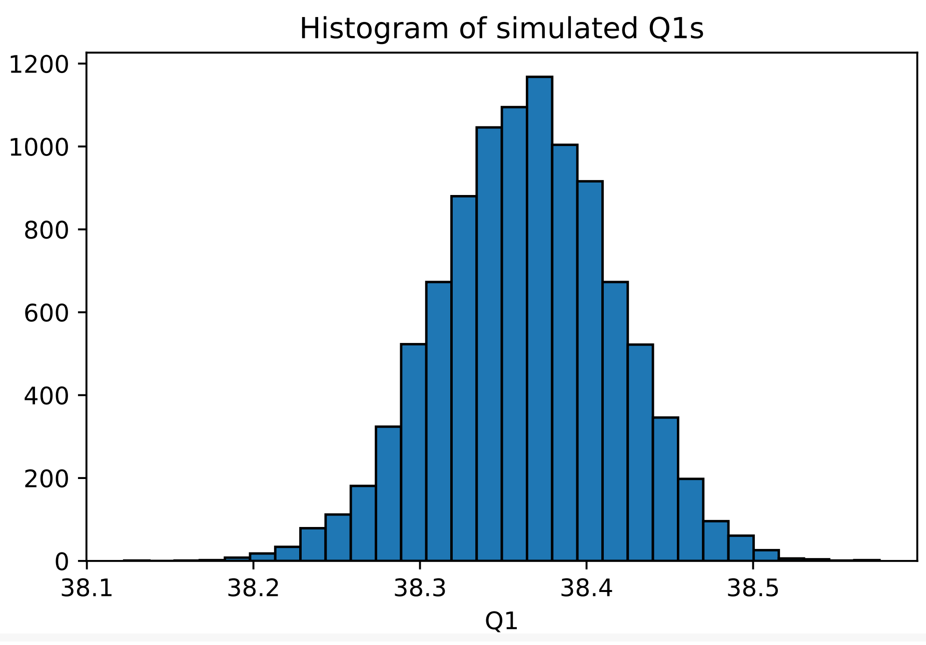
|||| Solution

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]))
```

```
[38.2581283  38.46561179]
```

```
# 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 38.43 ]
```

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 **??**)

```
## 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)

[-1.9 -0.5]
```

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]))

[-1.94849307 -0.44157426]
```

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

4.5 Non-linear error propagation

|||| Exercise 4.5 Non-linear error propagation

Solved question (a)-(c) without using Python, but use Python to simulate in question (d).

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 from the book ($A = XY$), since $T = PV/8.31$.

To use the approximate error propagation rule, we must differentiate the function $T = f(P, V) = PV/8.31$ with respect to both P and V:

$$\frac{\partial f}{\partial P} = V/8.31 \quad \frac{\partial f}{\partial V} = P/8.31.$$

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:

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

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

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

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 (c). The following Python-code will do the job:

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(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 (b) 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.

4.6 Simulation of statistical power

|||| Exercise 4.6 Simulation of statistical power

This exercise should be solved without using Python.

A company wants to measure the effect of a newly developed coating, that may improve shelf-life of their product. Without the coating the product has an average shelf-life of 27 days. They decide that the new coating will be worth the extra expense if the average shelf-life is increased by 1.5 days or more. But they also know that the shelf-life of their products are subject to rather large variability and they expect the standard deviation to be of order $\sim$ 5 days.

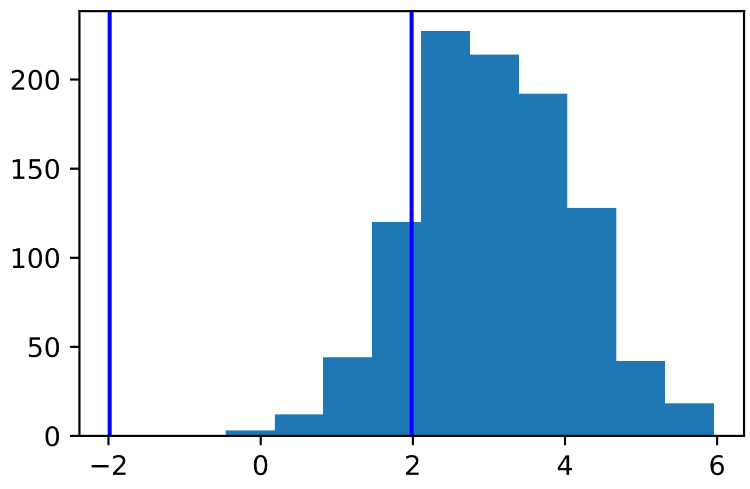
The company plans to make an experiment where they apply the new coating to 100 products and measure the shelf-life of these. But before they initiate the actual experiment they decide to simulate the risk of not detecting any significant improvement, if the true improvement is only 1.5 days (or less). They assume that the measured shelf-life, X, follows a normal distribution:

$$X \sim N(\mu = 28.5, \sigma^2 = 5^3)$$

The company carries out the following simulation in Python:

```
np.random.seed(412)

# Number of simulations
k = 1000

# sample size
n = 100

# Simulation
shelf_life_samples = stats.norm.rvs(loc=28.5, scale=5, size=(k,n))

# Simulation statistics:
sim_means = shelf_life_samples.mean(axis=1)
sim_SEM = shelf_life_samples.std(ddof=1, axis=1)/np.sqrt(n)

# Simulation results
sim_tobs = (sim_means - 27)/sim_SEM
tcritical = stats.t.ppf(0.975, df=n-1)
```

```
reject_null_hyp = np.abs(sim_tobs) >= tcritical

plt.hist(sim_tobs)
plt.axvline(-tcritical, linestyle='-', color="blue", ymin=0, ymax=1)
plt.axvline(tcritical, linestyle='-', color="blue", ymin=0, ymax=1)
plt.show()
```



```
n_rejected_null_hyp = np.sum(reject_null_hyp)
print(n_rejected_null_hyp)

858
```

Hint: Notice the variable `reject_null_hyp` is a boolean variable (`True` or `False`). Here we print the first 10 values in the variable `reject_null_hyp` and the first 10 values of the variable `sim_tobs`:

```
print(reject_null_hyp[:10])

[ True  True  True  True  True  True  True False False  True]


print(sim_tobs[:10])

[2.0167269  2.44449889 2.86036866 2.37276581 3.84951488 2.97859893
 2.83082468 1.67866428 1.37293584 3.29996005]
```

a) How many experiments (samples) were simulated?

|||| Solution

The number of samples simulated is $k = 1000$. (Each sample is of size $n = 100$.)

b) What is "`shelf_life_samples`"? (what are the dimensions of this variable?)

|||| Solution

The variable "`shelf_life_samples`" is a 2D array containing simulated shelf-lives of products. The array has 1000 rows and 100 columns. Each row represents one simulated sample (experiment) of 100 measurements.

c) What is "`sim_SEM`"?

|||| Solution

"`sim_SEM`" is an arrays containing the "standard error of the mean" for each of the 1000 simulated samples.

d) Explain the hypothesis test carried out in the simulation (and the histogram).

|||| Solution

In the simulation a hypothesis test is carried out for *each* simulated sample. The hypothesis test is a 1-sample t-test, where the null hypothesis ($\mu = \mu_0 = 27$ days), corresponds to the situation of the coating having no effect (since the known shelf-life of products without coating is also 27 days).

The hypothesis test is carried out by computing the test statistic $t_{obs} = \frac{\bar{x}-\mu_0}{SE_{\bar{x}}}$ and comparing the absolute value of this to the critical value $t_{0.975}$ in a t-distribution with $n-1 = 99$ degrees of freedom. This also means the test uses a significance level $\alpha = 0.05$.

The histogram displays all the 1000 calculated values of t_{obs} and the vertical lines indicate the critical values for the t-test. From the histogram we can see that some of the simulated experiments result in t_{obs} values that fall within the range of accepting the null hypothesis, but most values of t_{obs} are greater than the critical value $t_{0.975}$.

e) What is the conclusion from the simulation - regarding the risk of not detecting any significant improvement?

|||| Solution

The simulation shows that out of 1000 simulated experiments 858 will reult in rejecting the null hypothesis and hence the statistical power is expected to be ~ 0.86. This also means that there is a risk of ~ 0.14 of accepting the null hypothesis of "no effect" if the experiment is made with a sample size of $n = 100$ measurements and if the true effect size is 1.5 days (using a significance level of $\alpha = 0.05$ to test the null hypothesis). Hence the risk of making a type II error is ~ 0.14

f) Perform the theoretical calculation of needed sample size if the company wants a power of 0.90 (and using $\alpha = 0.05$).

For this calculation you will need one or more of the following values:

```
print(stats.norm.ppf(0.025))

-1.959963984540054 5

print(stats.norm.ppf(0.05))
```

```
-1.6448536269514729

print(stats.norm.ppf(0.10))

-1.2815515655446004

print(stats.norm.ppf(0.80))

0.8416212335729143

print(stats.norm.ppf(0.90))

1.2815515655446004

print(stats.norm.ppf(0.95))

1.6448536269514722

print(stats.norm.ppf(0.975))

1.959963984540054
```

|||| Solution

We use the formula

$$n = \left(\sigma \frac{z_{1-\alpha/2} + z_{1-\beta}}{\mu_0 - \mu_1}\right)^2 = \left(5 \frac{z_{0.975} + z_{0.90}}{1.5}\right)^2 = \left(5 \frac{1.96 + 1.28}{1.5}\right)^2 = 116.6$$

So we round up to $n = 117$.