Probability distributions EBP038A05 Assignments

Nicky van Foreest
Joost Doornbos, Wietze Koops, Mikael Makonnen

January 26, 2022

CONTENTS

Introduction 3
1 Assignment 1 4
1.1 Why is the Exponential Distribution so important? 4
1.2 BH.5.6.5 5
1.3 BH.7.1 7
1.4 BH.7.9 10
2 Assignment 2 13
2.1 BH.7.48 13
2.2 BH.7.53 14
2.3 Bayesian priors, Testing for rare deceases, Making the plot of Exercise 7.86 16
2.4 Redoing the computation of the Example 2.3.9 16
3 Assignment 3 17
3.1 BH.8.4 Figure 17
3.2 BH-8-4-3. Sum of two gamma rvs 18
3.3 BH.8.1 19
3.4 BH.8.18 19
4 Assignment 4 22
4.1 BH.8.27 22
4.2 BH.8.54 27
5 Assignment 5 29
5.1 BH.9.1.7 29
5.2 BH.9.1.7.The mystery box 31
5.3 BH.9.1.9 32
5.4 BH.9.6.1 33
5.5 BH.9.1 36
6 Assignment 6 38
6.1 BH.9.25 38
6.2 BH.9.37 40
6.3 BH.9.50 41
6.4 BH.10.2.3 43
7 Assignment 7 46
7.1 BH.10.9 46
7.2 BH.10.28 47
7.3 BH.10.30 49
7.4 BH.10.39 50

INTRODUCTION

Here we just provide the exercises of the assignments. For information with respect to grading we refer to the course manual.

Each assignment contains code in Python and R that shows how to implement an example or the solution of an exercises of BH. You have to run the code, read the output, and explain how the code works.

We include python and R code, and leave the choice to you what to use. In the exam we will also include both languages in the same problem, so you can stay with the language you like. You should know, however, that many of you will need to learn multiple languages later in life. For instance, when you have to access databases to obtain data about customers, patients, clients, suppliers, inventory, demand, lifetimes (whatever), you often have to use `sql`. Once you have the raw data, you process it with `R` or python to do statistics or make plots.[1] For your interest, based on the statistics here or here, python scores (much) higher than R in popularity; if you opt for a business career, the probability you have to use python is simply higher than to have to use R.

You should become familiar with looking up documentation on coding on the web, no matter your programming language of choice. Invest time in understanding the, at times, rather technical and terse, explanations. Once you are used to it, the core documentation is faster to read, i.e., less clutter. In the long run, it pays off.

The rules:

1. For each assigment you have to turn in a pdf document typeset in LaTeX. Include a title, group number, student names and ids, and date.
2. We expect brief answers, just a few sentences, or a number plus some short explanation. The idea of the assignment is to help you studying and improve your coding skills by showing good code, not to turn you in a writer.
3. When you have to turn in a graph, provide decent labels and a legend, ensure the axes have labels too.

[1] (While I (= NvF) worked at a bank, I used Fortran for numerical work, awk for string parsing and making tables, excel, SAS to access the database, and matlab for other numerical work, all next to each other. I got extremely tired of this, so I went to using python as this can do all of this stuff, but within one language.

1 ASSIGNMENT 1

1.1 *Why is the Exponential Distribution so important?*

At the Paris metro, a train arrives every 3 minutes on a platform. Suppose that 50 people arrive between the departure of a train and an arrival. It seems entirely reasonable to me to model the arrival times of the individual people as distributed on the interval $[0,3]$. What is the distribution of the inter-arrival times of these people? It turns out to be exponential!

You might want to compare your final result to Figure BH.13.1 (It is not forbidden to read the book beyond what you have to do for this course!). In this exercise we use simulation to see that clustering of arrival times.

Python Code

```python
import numpy as np

np.random.seed(3)


num = 5 # small sample at first, for checking.
start, end = 0, 3
labda = num / (end - start)  # per minute
print(1 / labda)

A = np.sort(np.random.uniform(start, end, size=num))
print(A)
print(A[1:])
print(A[:-1])
X = A[1:] - A[:-1]
print(X)

print(X.mean(), X.std())
```

R Code

```r
set.seed(3)


num = 5
start = 0
end = 3
labda = num / (end - start)
print(1 / labda)

A = sort(runif(num, min = start, max = end))
print(A)
print(A[-1])
```

```
13 print(A[-length(A)])
14 X = A[-1] - A[-length(A)]
15 print(X)
16
17 print(mean(X))
18 print(sd(X))
```

Ex 1.1. Explain the result of line P.12 (R.13) This are the arrival times of 5 passengers within the time interval of 3 minutes (sorted increasingly).

Ex 1.2. Compare the result of line P.13 and P.14 (R.12, R.13); explain what is `A[1:]` (`A[-1]`) `A[1:]` is an array of all elements of `A` except the first one.

Ex 1.3. Compare the result of line P.12 and P.14 (R.11 and R.13); explain what is `A[:-1]` (`A[-length(A)]`). `A[:-1]` is an array of all elements of `A` except the last one.

Ex 1.4. Explain what is `X` in P.15 (R.14) `X` consists of the interarrival times.

Ex 1.5. Why do we compare $1/\lambda$ and `X.mean()`? $1/\lambda$ is the expected interarrival time. `X.mean()` is the sample average of the interarrival times.

Ex 1.6. Recall that $\mathsf{E}[X] = \sigma(X)$ when $X \sim \text{Exp}(\lambda)$. Hence, what do you expect to see for `X.std()`? For `X.std()` we expect to see $1/\lambda = 0.6$ too (if X is indeed exponentially distributed with parameter λ).

Ex 1.7. Run the code for a larger sample, e.g. 50, and discuss (very briefly) your results. For a sample of size 50, we expect an average interarrival time of 0.06 and an equal standard deviation if the distribution of the interarrival times is indeed exponential. We indeed observe a sample mean and sample average that are very close to this value.

1.2 *BH.5.6.5*

Read this example of BH first. We chop up the exercise in many small exercises..

For the python code below, run it for a small number of samples; here I choose `samples=2`. Read the print statements, and use that to answer the questions below.

Python Code

```
1 import numpy as np
2 from scipy.stats import expon
3
4 np.random.seed(10)
5
6 labda = 6
7 num = 3
8 samples = 2
9
```

```
10 X = expon(scale=labda).rvs((samples, num))
11 print(X)
12 T = np.sort(X, axis=1)
13 print(T)
14 print(T.mean(axis=0))
15
16 expected = np.array([labda / (num - j) for j in range(num)])
17 print(expected)
18 print(expected.cumsum())
```

R Code

```
1  set.seed(10)
2
3  labda = 6
4  num = 3
5  samples = 2
6
7  X = matrix(rexp(samples * num, rate = 1 / labda), nrow = samples, ncol = num)
8  print(X)
9  bigT = X
10 for (i in 1:samples) {
11   bigT[i,] = sort(bigT[i,])
12 }
13 print(bigT)
14 print(colMeans(bigT))
15
16 expected = rep(0, num)
17 for (j in 1:num) {
18   expected[j] = labda / (num - (j - 1))
19 }
20 print(expected)
21 print(cumsum(expected))
```

Ex 1.8. In line P.11[2] we print the value of `X` in line P.10, R.7 and R.8, respectively. What is the meaning of `X`? X is a matrix of i.i.d. draws from an exponential distribution with parameter λ.

Ex 1.9. What is the meaning of `T` in line P.12 (R.11)? T is a sorted version of X, where we sort each row increasingly.

Ex 1.10. What do we print in line P.14, R.14? We print the mean value of each column of T.

Ex 1.11. What is meaning of the variable `expected`? This is an array with expected values of the ith order statistic $X_{(i)}$ (see B.H.5.6.5 afor a proof of this result).

[2] Line P.x refers to line x of the Python code. Line R.x refers to line x of the R code.

Ex 1.12. What is the `cumsum` of `expected`? The cumsum is the cumulative sum up to and including the current index. So the final entry indicates the expected value of the sum of all three entries of `expected`.

Ex 1.13. Now that you understand what is going on, rerun the simulation for a larger number of samples, e.g., 1000, and discuss the results briefly. The result of `print(T.mean(axis=0))` should be close to that of `print(expected.cumsum())`.

1.3 *BH.7.1*

First read and solve BH.7.1. Answers and hints are in the study guide.

Ex 1.14. Sketch, on paper, the 2D area that corresponds to the event that Alice and Bob meet. Just make a photo with your mobile phone and include it. (Like this you learn how to include jpeg or png files in LaTeX.)

Ex 1.15. Write A and B for the rvs corresponding to the arrival times of Alice and Bob. Explain you have to compute the expectation of $I_{A<B+0.25} \cdot I_{B-0.25<A}$ to compute the probability that Alice and Bob meet. Then, explain that is this equal to

$$\int_0^1 \int^1 I_{a<b+0.25} \cdot I_{b-0.25<a} \, \mathrm{d}a \, \mathrm{d}b. \tag{1.1}$$

Ex 1.16. What is the probability that they meet (i.e., solve the integral by hand).

Ex 1.17. Explain this code to compute numerically the probability of overlap. Check the web on what is an anonymous function in python or R, and explain why we use that here. (Hint: anonymous functions are useful "when it's not worth the effort to give it a name". Do we use elswhere the anonymous function we define here?)

Python Code

```python
from scipy.integrate import dblquad

area = dblquad(lambda a, b: (b - 0.25 < a) * (a < b + 0.25), 0, 1, 0, 1)
print(area)
```

Ex 1.18. Explain the output, i.e., the results of the print statements, of this piece of code.

Python Code

```python
import numpy as np
from scipy.stats import uniform

np.random.seed(3)

n = 5
alice = uniform(0, 1).rvs(n)
bob = uniform(0, 1).rvs(n)
```

```python
print(alice)
print(bob)
print((alice < bob))
```

I tend to set the seed of the random number generator so that I always get the same results. This helps a lot when debugging, because the numbers don't change time and again.

Ex 1.19. Run this piece of code, and then check that you get different output every time.

Python Code

```python
import numpy as np
from scipy.stats import uniform

n = 5
alice = uniform(0, 1).rvs(n)
bob = uniform(0, 1).rvs(n)
print(alice)
print(bob)
overlap = (alice < bob) # this
print(overlap)
```

Explain that the line marked as 'this' implements an indicator function.

Ex 1.20. Run this piece of code, and explain the output.

Python Code

```python
import numpy as np
from scipy.stats import uniform

np.random.seed(3)

n = 5
alice = uniform(0, 1).rvs(n)
bob = uniform(0, 1).rvs(n)
overlap = (bob - 0.1 < alice) * (alice < bob + 0.1)
print(alice)
print(bob)
print(overlap)  # this line
```

Ex 1.21. To find the probability of overlap, we have to modify in the code above the line marked as 'this line' to

Python Code

```python
print(overlap.mean())
```

Change this in your code and explain what it does.

Ex 1.22. Modify the code of the previous exercise so that you sample 1000 times an appointment between Alice and Bob, and such that they only meet when 15 minutes appart. Include your code and show the output.

Ex 1.23. Contrary to [1.14] suppose Bob is not prepared to wait longer than 5 minutes, while Alice is still prepared to wait for 15 minutes. Write down the integral that corresponds to the proability that they meet. Solve it by hand. Modify the above code such that you can evaluate it numerically. Include your code in the answers.

Ex 1.24. Contrary to [1.14] suppose Bob is prepared to wait for 20 minutes when he arrives before 1230, but just 10 when he arrives after 1230, while Alice is still prepared to wait for 15 minutes. Write down the integral that corresponds to the proability that they meet. Explain that this is the code to compute this.

Python Code

```python
from scipy.integrate import dblquad
area1 = dblquad(lambda a, b: (b - 1 / 3.0 < a) * (a < b + 0.25), 0, 1, 0, 0.5)
area2 = dblquad(lambda a, b: (b - 1 / 6.0 < a) * (a < b + 0.25), 0, 1, 0.5, 1)
print(area1[0] + area2[0])
```

Ex 1.25. Contrary to [1.14] suppose Bob's arrival time is Exp(λ) with $\lambda = 1/2$, while Alice's arrival time is still uniform. Explain the code below.

Python Code

```python
import numpy as np
from scipy.integrate import dblquad
from scipy.stats import uniform, expon

np.random.seed(3)

labda = 1.0 / 2
end = 5

value = dblquad(
    lambda a, b: (b - 0.25 < a) * (a < b + 0.25) * np.exp(-labda * b), 0, 1, 0, end
)
print(value)

area = value[0] * labda
print(area)

n = 100000
alice = uniform(0, 1).rvs(n)
bob = expon(scale=1 / labda).rvs(n)
overlap = (bob - 0.25 < alice) * (alice < bob + 0.25)
print(overlap.mean())
```

Ex 1.26. Changing the value of the variable end to 10 0 in the code above, and run it again. Then chang it to 50, and finally to 1000. When I do this, I get a very strange answers for 50 and 1000. Try to explain what goes wrong.

Hopefully you got by now the point: modeling is an interesting, but quite difficult, activity. Once you have the model, the computer can help solve the problem.

You should memorize that integration in multiple dimensions quickly becomes unmanageable. In fact, most higher D integrals are approximated with simulation, like we do here in the simulation part.

You should also memorize that computers are extremely dumb, and that they can live with any answer that comes out of a computation. Here, since we compare a simulation with a direct integration, we see a difference, and that makes us suspicious about the correctness of both. Mind, we humans can do pretty smart things, and we should typically distrust the output of computers. (That is why I often include tests.)

1.4 *BH.7.9*

First read and solve BH.7.9. Here we provide a simulation.

Ex 1.27. Run this code, and use the documentation of scipy.stats.geom to explain why it fails. Perhaps you can also use the wikipedia page on the geometric distribution. (Hint: is the first success distribution of the geometric distribution?)

Python Code

```python
import numpy as np
from scipy.stats import randint, geom

np.random.seed(3)

p, num = 0.1, 10000
q = 1 - p
rv = geom(p)  # This line
X = rv.rvs(num)

if X.min() != 0:
    raise ValueError("The minimal value of X is not 0")
if not np.isclose(q / p, X.mean(), 0.1):
    raise ValueError("The mean of X is not ok")
```

Ex 1.28. The code of the previous exercise failed, on purpose. It will work if we change the line marked as “this line”, into this:

Python Code

```python
rv = geom(p, loc=-1)
```

Explain the difference, and why it now works. Relate this location-scale transformation to one of the distributions you know from BH.

Memorize from this exercise that with such checks you can build code that fails when it gets unexpected input.

Ex 1.29. Run this code and explain in detail the output. Make, and include, some simple example code to show that you are correct.

Python Code

```python
import numpy as np

N = np.array([3, 4, 4, 8, 4])
X = np.array([8, 7, 6, 5, 3])
print(np.argwhere(N == 4))
Xn = X[np.argwhere(N == 4)]
print(Xn)
```

Ex 1.30. Explain the lines marked with ‘this in the code of listing Listing 1. You already explained the other parts so you don’t have to do that again.

Ex 1.31. Explain the results. Do they match with the theoretical results?

Python Code

```python
import numpy as np
from scipy.stats import randint, geom
import matplotlib.pyplot as plt

np.random.seed(3)

p, num = 0.1, 100000
q = 1 - p
rv = geom(p, loc=-1)
X = rv.rvs(num)

Y = rv.rvs(num)
N = X + Y

fig, axes = plt.subplots(3, 3, sharey="all", figsize=(6, 3))  # this
n = 7  # gives nice results, after some experimentation
for ax in axes.flatten():  # this
    Xn = X[np.argwhere(N == n)]
    y, bins = np.histogram(Xn, bins=np.linspace(0, n + 1, n + 2)) # this
    ax.plot(range(n + 1), y, 'bo', ms=3, label='N=8')  # this
    ax.vlines(range(n + 1), 0, y, colors='b', lw=2, alpha=0.5) # this
    ax.set_title(f'{n=} ({len(Xn)})')
    ax.set_ylim(0) # this
    n += 1
```

Listing 1: BH.7.9, python code.

2 ASSIGNMENT 2

2.1 *BH.7.48*

Read this exercise first and solve it. Then consider the code below.

Python Code

```python
import numpy as np

np.random.seed(3)


def find_number_of_maxima(X):
    num_max = 0
    M = -np.infty
    for x in X:
        if x > M:
            num_max += 1
            M = x
    return num_max


num = 10
X = np.random.uniform(size=num)
print(X)

print(find_number_of_maxima(X))

samples = 100
Y = np.zeros(samples)
for i in range(samples):
    X = np.random.uniform(size=num)
    Y[i] = find_number_of_maxima(X)

print(Y.mean(), Y.var(), Y.std())
```

R Code

```r
set.seed(3)

find_number_of_maxima = function(X) {
  num_max = 0
  M = -Inf
  for (x in X) {
    if(x > M) {
      num_max = num_max + 1
9        M = x
10     }
11   }
12   return(num_max)
13 }
14
15
16 num = 10
17 X = runif(num, min = 0, max = 1)
18 print(X)
19
20 print(find_number_of_maxima(X))
21
22 samples = 100
23 Y = rep(0, samples)
24 for (i in 1:samples) {
25   X = runif(num, min = 0, max = 1)
26   Y[i] = find_number_of_maxima(X)
27 }
28
29 print(mean(Y))
30 print(var(Y))
31 print(sd(Y))
```

Ex 2.1. Explain how the small function in lines P.6 to P.13 (R.4-R.12) works. (You should know that `x += 1` is an extremely useful abbreviation of the code `x = x + 1`). It iterates through the elements of X and checks how often the current value is larger than any of the previous values.

Ex 2.2. Explain the code in lines P.25 and P.26 (R.25, R.26). We draw a sample of a $U[0,1]$ distribution of size `num` and compute the corresponding number of maxima (or "records").

2.2 *BH.7.53*

Ex 2.3. Read and solve BH.7.53.

Ex 2.4. The code in Listing 2 or Listing 3 simulates this exercise. Run the code. (To improve your understanding, just change some parameters here and there, and check the output.) Explain the lines marked with `#*`. (In particular, why do we do a `runs` number of runs to estimate the covariance?) Finally, explain the output.

Ex 2.5. Modify the code so that the drunkard makes steps of size 2 in when moving left or right; the stepsizes up or down remain the same. What happes with $\mathsf{E}\left[R_n^2\right]$? The line for the x-steps becomes

Python Code

```python
import numpy as np
from scipy.stats import randint

np.random.seed(3)


num = 4
steps = randint(0, 4).rvs(num) # *
print(steps)
X = (2 * steps - 1) * (steps < 2) # *
Y = (2 * steps - 5) * (steps >= 2) # *
print(X)
print(Y)
R2 = X * X + Y * Y
print(f"{num=}, {R2.sum()=}")


def cov(X, Y):
    return (X * Y).mean() - X.mean() * Y.mean()  # *


num = 10
runs = 300
Ss = np.zeros(runs)  # *
Ts = np.zeros(runs)
for i in range(runs):  # *
    steps = randint(0, 4).rvs(num)
    X = (2 * steps - 1) * (steps < 2)
    Y = (2 * steps - 5) * (steps >= 2)
    Ss[i] = X.sum() # *
    Ts[i] = Y.sum() # *

print(f"{cov(Ss,Ts)=}") # *
```

Listing 2: BH.7.53, python code.

R Code

```r
a <- 3;
```

Listing 3: BH.7.53, R code.

Python Code

```python
X = (4 * steps - 2) * (steps < 2)  # *
```

Ex 2.6. Optional: Add a third dimension.

2.3 *Bayesian priors, Testing for rare deceases, Making the plot of Exercise 7.86*

In line with Exercise BH.8.33, we are now going to analyze the effect on $\mathsf{P}\{D \mid T\}$ when the sensitivity is not known exactly. So, why is this interesting? In Example 2.3.9 the sensitivity is given, but in fact, in 'real' experiments, this is not always known as accurately as assumed in this example. For example, in this paper: False-positive COVID-19 results: hidden problems and costs it is claimed that 'The current rate of operational false-positive swab tests in the UK is unknown; preliminary estimates show it could be somewhere between 0·8\Hence, even though it is claimed that PCR tests 'have analytical sensitivity and specificity of greater than 95\Simply put, the specificity and sensitivity are not precisely known, hence this must affect $\mathsf{P}\{D \mid T\}$.

To help you, we show how to make one graph. Then we ask you to make a few on your own, and comment on them.

2.4 *Redoing the computation of the Example 2.3.9*

I write p_D_g_T$ for $\mathsf{P}\{D \mid T\}$. Here is how this can be implemented in python.

Python Code
```python
sensitivity = 0.95
specificity = 0.95
p_D = 0.01

p_T = sensitivity * p_D + (1-specificity)*(1-p_D)
p_D_g_T  = sensitivity * p_D/p_T
p_D_g_T
```

1. Make a plot of $\mathsf{P}\{D \mid T\}$ in which you vary the sensitivity from 0.9 to 0.99. Explain what you see.
2. Make a plot of $\mathsf{P}\{D \mid T\}$ in which you vary the specificity from 0.9 to 0.99.
3. Make a plot of $\mathsf{P}\{D \mid T\}$ in which you vary $\mathsf{P}\{D\}$ from 0.01 to 0.5. Explain what you see.
4.

3 ASSIGNMENT 3

3.1 *BH.8.4 Figure*

The challenge is to make Figure BH.8.4 for a specific case.

We throw a coin a number of times. We take some beta distribution as prior for the probability p that Heads appears. Then we update the PDF of p according to outcomes of the throws.

Ex 3.1. Run the follow code and explain the output.

We define the variables H and T as these outcomes are easier to memorize than 1 and 0. Like this, there is less possibility for confusion, hence less bugs.

Python Code

```python
H, T = 1, 0
outcomes = [H, T]

h, t = 1, 1  # this
h += outcomes[0] == H
t += outcomes[0] == T
print(f"{h=}, {t=}")

h += outcomes[1] == H
t += outcomes[1] == T
print(f"{h=}, {t=}")
```

Ex 3.2. Explain the code below. Why does the line marked as 'this' relate to the prior distribution? What is that prior distribution?

Python Code

```python
import numpy as np
from scipy.stats import beta
import matplotlib.pyplot as plt

support = np.arange(0, 1, 0.03)

H, T = 1, 0
outcomes = [H, H, T, H, H, H, T, H, H]
h, t = 1, 1  # this

fig, axes = plt.subplots(3, 3, sharey="all", figsize=(6, 3))  # this
for i, ax in enumerate(axes.flatten()):  # this
    h += outcomes[i] == H
    t += outcomes[i] == T
    X = beta(h, t).pdf(support)
    ax.plot(support, X)
    ax.set_title(f'h={h-1}, t={t-1}')

plt.tight_layout()
plt.savefig("beta.pdf")
```

Ex 3.3. What do the PDFs actually model? Explain why the PDFs behave as they do. For instance, why does the location of the peak move from right to left to right?

Ex 3.4. Which conjugacy relation between prior and data do we use here to update the prior to the posterior?

Ex 3.5. Adapt the code above such that the prior is $a = 3.5, b = 1.5$, the outcomes are T, H, T, H, T, H, and the output is a figure with 3 x 2 subfigures. Besides the explanation, include the figure.

3.2 *BH-8-4-3. Sum of two gamma rvs*

Let us investigate BH.8.4.3. The rvs X and Y are independent, $X \sim \text{Exp}(\lambda)$ and $Y \sim \text{Exp}(\mu)$. We want to compare $X + Y$ to $\Gamma 2, \mu$ and $\Gamma 2, \lambda$, just to see what happens.

Ex 3.6. Run the code below. Explain the lines marked as 'this'. In particular, why do we simulate outcomes for X and Y? Then, why are the shape and scale variables as they are? Then explain the three graphs.

Python Code

```python
import numpy as np
from scipy.stats import expon, gamma
import matplotlib.pyplot as plt

labda, mu = 3, 5

n = 1000
X = expon(scale=1 / labda)
Y = expon(scale=1 / mu)
Z = X.rvs(n) + Y.rvs(n)          # this
G1 = gamma(2, scale=1 / labda)   # this
G2 = gamma(2, scale=1 / mu)      # this

support = np.arange(0, 5, 0.03)

plt.hist(Z, label="Z", density=True)
plt.plot(support, G1.pdf(support), label="G1")
plt.plot(support, G2.pdf(support), label="G2")
plt.legend()
plt.tight_layout()
plt.savefig("figures/gamma1.pdf")
```

Ex 3.7. Modify the above code such that X and Y are both Exp(λ). Then make a histogram of a simulation of $X + Y$ and compare it to the PDF of the correct gamma distribution. Discuss your result, and include the figure of course.

3.3 *BH.8.1*

Ex 3.8. Here is the code to solve BH.8.1. Explain the relevant steps, i.e., the lines marked as 'this'. Is the result according to your expectation?

Python Code

```python
import numpy as np
from scipy.stats import expon
import matplotlib.pyplot as plt

X = expon(scale=1) # this
Y = np.exp(-X.rvs(1000)) # this
plt.hist(Y, density=True, label="sim")

x = np.linspace(0, 1, 30)
dxdy = np.exp(x)  # this
pdf_y = X.pdf(x) * dxdy # this
plt.plot(x, pdf_y, label="f_Y")
plt.legend()
plt.tight_layout()
plt.savefig("figures/bh-8-1-fig.pdf")
```

Ex 3.9. What error would we make if we would use this code: `dxdy = np.exp(-x)`?

3.4 *BH.8.18*

Use python uncertainties module.[3].

We can demonstrate the use of Taylor's equation to see how uncertaintly should propagate[4].

How many ping pong balls fit into an Airbus Beluga? One way to answer this is as follows. According to this wiki-page the cargo volume V of this airplane is $1500\,\text{m}^3$. But this number is based on the physical dimensions that is available to store containers, tanks, and so on. So, I estimate the volume as about twice that amount, i.e., $V = 2500\,\text{m}^3$. The volume of a ping pong ball is $v = 4\pi r^3/3 = 33.49333333333333\,\text{cm}^3$ with $r = 2$ cm. A plain

[3] https://pythonhosted.org/uncertainties/
[4] https://en.wikipedia.org/wiki/Propagation_of_uncertainty

division gives 74.6268656716418 ping pong balls. Note, I left out the 10^6 conversion from meters to cm, and I do not take into the sphere packing factor. Besides that, I hope you agree with me that providing an result with the precision as given here is plain ridiculous.[5]

In fact, I know that the volumes of an air plane and a ping pong ball is an estimate, rather than a precise number as assumed above. It seems to be better to approximate V and v as rvs. Let's assume that

$$V \sim N(2500, 500^2), v \sim N(33.5, 0.5^2),$$

where the variances express my trust in my guess work. What is now the mean of $N = V/v$ and its std? Can we get the CDF? From BH8.18 you know that finding the closed form expression for the distribution of N is not entirely simple. However, with simulation it's easy to get an estimates for the mean and the std.

Ex 3.10. Explain lines marked as 'this'.

Python Code

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

num = 500

np.random.seed(3)

V = norm(loc=2500, scale=500) # this
v = norm(loc=33.5, scale=0.5) # this

print(V.mean(), V.std()) # just a check

N = V.rvs(num) / v.rvs(num) # this
print(N.mean(), N.std())

print(2500/33.5)
print(np.sqrt(500*0.5))
```

R Code

```r
num <- 500

set.seed(3)

V = rnorm(num, 2500, 500) # this
v = rnorm(num, 33.5, 0.5) # this

```

[5] For reasons incomprehensible to me, even professional econometricians sometimes report results with 10 digits or more, without questioning the precision.

```
N = V / v  # this
paste(mean(N), sd(N))

2500/33.5
sqrt(500*0.5)
```

Ex 3.11. Contrary to BH.7.1.25 if you run the code above, you'll see that $\mathsf{E}[N] < \infty$, and is very near to the deterministic answer. But isn't this strange: we divide two normal random variables, just like BH.7.1.25, but there the expectation is infinite. Explain why it works for the Beluga case, but not for the case discussed in BH.7.1.25. If we divide two $\sim$ Norm(0,1) r.v.s we obtain a Cauchy distributed rv. Note that here both rvs have a mean of 0. In contrast, in the Beluga case, the normally distributed rvs have positive expectation.

Ex 3.12. Include code to make a histogram of the PDF of N.

Remark 3.13. Here is a final, and optional, question. The numerical results suggest the interesting guess $\mathsf{V}[N] \approx \mathsf{V}[V] * \mathsf{V}[v]$, but is this true more generally?

4 ASSIGNMENT 4

4.1 *BH.8.27*

We start from BH.8.27 (which you have to read now). We are interested in the difference between the distribution of $X+Y+Z$ and the normal distribution. But why the normal distribution? As it turns out, the central limit law, see BH.10, states that the distribution of sums of r.v.s converge to the normal distribution (in a specific sense)

Here is some code to simulate.

Python Code

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

import matplotlib.pylab as plt
import seaborn as sns

sns.set()

np.random.seed(3)

k = 3
Zexact = norm(loc=k / 2, scale=np.sqrt(k / 12))
X = np.arange(0, 3, 0.1)

XYZ = np.random.uniform(size=(4000, k))
# print(XYZ)  # if you want to see it.
Z = XYZ.sum(axis=1)
sns.distplot(Z)
plt.plot(X, Zexact.pdf(X))
plt.show()
```

R Code

```r
set.seed(3)

k = 3
X <- seq(0, 3, by = 0.1)
Zexact <- dnorm(X, mean = k / 2, sd = sqrt(k / 12))


XYZ <- matrix(NA, 4000, k)
for (i in 1:k) {
  XYZ[,i] <- runif(4000, min = 0, max = 1)
}
Z <- rowSums(XYZ)
```

```

par()
hist(Z, prob = TRUE, breaks = 31)
lines(X, Zexact, type = "l", col = "orange")
lines(density(Z), col = "blue")
```

Ex 4.1. What is the shape of `XYZ` in the code above, i.e., how many rows and columns does it have? If you don't know, run the code, and print it.

Ex 4.2. What is the shape (rows and columns) of `Z`?

Ex 4.3. Explain the values for `loc` and `shape` in `Zexact`. (Read the documentation of scipy.stats.norm on the web is necessary.) To which definition in BH does this loc-scale transformation relate?

Ex 4.4. Change the seed to your student id, or any other number you like, run the code, and include the graph produced by your simulation. Explain what you see.

Now we do an exact computation.

Python Code

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

import matplotlib.pylab as plt
import seaborn as sns

sns.set()

N = 200
x = np.linspace(0, 2, 2 * N)
fx = np.ones(N) / N
f2 = np.convolve(fx, fx)
f3 = np.convolve(f2, fx)

k = 3

x = np.linspace(0, k, len(f3))
Zexact = norm(loc=k / 2, scale=np.sqrt(k / 12))


plt.plot(x, N * f3, label="conv")
plt.plot(x, Zexact.pdf(x))
plt.legend()
plt.show()
```

R Code

```r
N = 200
x = seq(0, 2, length.out = 2 * N)
fx = rep(1, N) / N
f2 = convolve(fx, fx, type = "open")
f3 = convolve(f2, fx, type = "open")

k = 3

x = seq(0, k, length.out = length(f3))
Zexact = dnorm(x, mean = k/2, sd = sqrt(k / 12))

par()
plot(x, N * f3, col = "blue", type = "l", ylim = c(0, 0.8))
lines(x, Zexact, type = "l", col = "orange")
legend("topright", legend = "conv", bty = "n",
       lwd = 2, cex = 1.2, col = "blue", lty = 1)
```

Ex 4.5. Read the documentation of np.convolve . Why is it called like this?

Ex 4.6. In the code, what is `f2`?

Ex 4.7. What is `f3`?

Ex 4.8. Why do we set `k=3`?

Ex 4.9. A bit harder, why do we plot `N*f3`, i.e., why do have to multiply with `N`? Relate this to the meaning of $f(x)\,\mathrm{d}x$, where f the density of some random variable. (To understand why is very important. Think hard, and read the solution when it becomes available.) Suppose we chop up the area under some arbitrary function g in blocks of height $g(x)$ and length Δx. Then the area of such a block is $g(x)\Delta x$.

In our case, we chop up the interval in parts with length $\Delta x = 1/N$. The elements of $f3$ are such that $f3[i] = f_3(x_i)\Delta x$, where x_i lies in the ith interval and f_3 is the density of the sum of the three r.v.s. But then, $f_3(x_i) = f3[i]/\Delta x = Nf3[i]$.

Forgetting to scale with $\Delta x = 1/N$ is a common error when dealing with densities. Hence, recall that, notationally, $f(x)\,\mathrm{d}x$ means a block of height $f(x)$ and length $\mathrm{d}x$. Don't forget to deal with the $\mathrm{d}x$!

Ex 4.10. Yet a tiny bit harder, consider `f4 = np.convolve(f3, fx)` and `g4 = np.convolve(f2, f2)`. Why are they, numerically speaking, equal? Take four r.v.s U,V,X,Y. Then

$$(U+V+X)+Y = (U+V)+(X+Y). \tag{4.1}$$

Thus the density of $(U+V+X)+Y$ must be the same as the density of $(U+V)+(X+Y)$.

Ex 4.11. When you would compute the maximum of `np.abs(f4 -g4)` you would see that this is about 10^{-10}, or so. Hence, a small number. This is not equal to 0, but we know that this is due to rounding effects.

How can we use the function `np.isclose()` to get around this problem? (You should memorize from this question that you should take care when testing on whether floating point numbers are the same or not.)

We simulate the post office part of BH.8.36. Read it now, i.e., before reading the text below. Then read the code below. In the questions we ask you to explain what the code does. There are lots of print statements that have been commented out, but we left them in for you to include while experimenting with the code to see how the code works. (I often use print statements of intermediate results when writing a program, just to see whether I am still on track. Once I checked, I remove them, because they clutter the looks of the code.)

Python Code

```python
import numpy as np
from scipy.stats import expon

np.random.seed(3)

labdas = np.array([3, 4])

N = 10

T1 = expon(scale=1 / labdas[0]).rvs(N)
# print(T1.mean())
T2 = expon(scale=1 / labdas[1]).rvs(N)
T = np.zeros([2, N])
T[0, :] = T1
T[1, :] = T2
# print(T)
server = np.argmin(T, axis=0)
# print(server)

# BH.8.36.b
print((server == 1).mean())

# BH.8.36.c
# print(labdas[server])
S = expon(scale=1).rvs(N) / labdas[server]
# print(S)

D = np.min(T, axis=0) + S
# print(D)

print(D.mean(), D.std())
```

R Code

```
1  set.seed(3)
2
3  labdas = 3:4
4
5  N = 10
6
7  T1 = rexp(N, rate = labdas[1])
8  T2 = rexp(N, rate = labdas[2])
9
10
11 bigT = matrix(0, nrow = 2, ncol = N)
12 bigT[1,] = T1
13 bigT[2,] = T2
14
15 server = rep(0, ncol(bigT))
16
17 for (i in 1:ncol(bigT)) {
18   server[i] = which.min(bigT[,i])
19 }
20
21 print(mean((server == 1)))
22
23 S = rexp(N, rate = 1) / labdas[server]
24 print(S)
25
26 D = apply(bigT, MARGIN = 2, FUN = min) + S
27
28 print(mean(D))
29 print(sd(D))
```

Ex 4.12. Explain why `T1` corresponds to a number of service times of the first server.

Ex 4.13. To what do the rows of T (`bigT`) correspond?

Ex 4.14. What is the content of `server`? Why do we compute this?

Ex 4.15. Explain how we use the fundamental bridge in line P.21 (R.21) to answer BH.8.36.b.

Ex 4.16. Alice is taken into service by the server that finishes first. We need to simulate the service time that Alice needs at that server. Explain how we do this in line P.25 (R.23). Hint, reread BH.5.5 on the exponential. BTW, this is a good time to reread BH.5.3.

Ex 4.17. Why is `np.min(T, axis=0)` in the python code (`apply(bigT, MARGIN=2, FUN=min)` in R) the time Alice spends waiting in queue, i.e., the time Alice spends in the post office before her service starts?

Ex 4.18. Why is `D` the departure time of Alice, i.e., the time Alice spends in the post office?

Ex 4.19. Set `N` to 1000 or so, or any other large number to your liking, but not so large that your computer will keep simulating for a month... Compare the values of the simulation to the theoretical result that you have to compute in BH.8.36.c.

Ex 4.20. Run the code for $\lambda_1 = 1$ fixed, and take λ_2 equal to $0.5, 1, 1.5$ and 2 successively. Compute the mean time waiting time and mean sojourn time of Alice, and put your results in a table. Compare the results of the simulation to the theoretical values.

There is an important lesson to learn here. With simulation it is, often, reasonably simple to get numerical answers, but it requires many simulations to see a pattern in the numbers. For patterns we can better use theory, as theory gives us formulas that show how the output of some model depends on the input.

4.2 *BH.8.54*

Read and solve BH.8.54. Perhaps you should read the hints in the study guide on this problem too. First we consider the case with $r = 1$, and we need to tackle some technical details.

Ex 4.21. Just as in the problem, let $Y = pX/q$. Why is $\mathsf{E}[Y] = 1$?

Ex 4.22. If $Z \sim \text{Exp}(1)$, then explain that $M_Z(s) = 1/(1-s)$ for $s < 1$.

Ex 4.23. Use the mathematical solution of the problem to explain that $M_Y(s) \to M_Z(s)$ as $p \to 0$.

Ex 4.24. Here is code to compare $M_Y(s)$ to $M_Z(s)$. Explain what the function $M(s)$ does. Then explain why in the line marked as 'this' we multiply with $1 - s_i$. Explain why we limit s to the interval $[0, 0.8]$. Finally, explain why the graph goes through the point $x = 0, y = 1$.

Python Code

```python
import numpy as np
from scipy.stats import geom, gamma
import matplotlib.pyplot as plt

np.random.seed(3)

p = 0.1
q = 1 - p
n = 10000

```

```python

def M(s):
    X = geom(p).rvs(n)
    Y = p * X / q
    return np.exp(s * Y).mean()


num = 30
m = np.zeros(num)
s = np.linspace(0, 0.8, num)
for i in range(num):
    m[i] = M(s[i]) * (1 - s[i])  # this

plt.plot(s, m, label="MY")
plt.tight_layout()
plt.savefig("figures/moments.pdf")
```

Ex 4.25. The match is not terrric when $p = 0.1$. Rerun the code, but now with $p = 1/1000$. Which line to you have to change for this? Include your graph. Is the match better now?

Ex 4.26. The problem asks for the general r. What lines of code have to change to deal with general r? Modify the code appropriately, and explain why your code is correct. The limiting mgf is the mgf of the gamma distribution. This is $(1/(1-s))^r$. Also the mgf of the geom must be raised to the power r.

Python Code

```python
    m[i] = (M(s[i]) * (1 - s[i]))**r
```

5 ASSIGNMENT 5

5.1 *BH.9.1.7*

The aim is to use a simulator to analyze the expected profit for the game of BH.9.1.7 but for a different prior than the uniform. First, however, we build BH.9.1.7 with the uniform prior so that you we our code.

First read and solve BH.9.1.7.

Ex 5.1. Run the code below. What are the values of `accepted`? What does it represent. How is the gain computed? Why do we take the mean?

Python Code

```python
import numpy as np
from scipy.stats import uniform

np.random.seed(3)

alpha = 2 / 3
n = 100
V = uniform(0, 1).rvs(n)

bid = 0.5
accepted = bid > alpha * V
gain = (V - bid) * accepted
print(gain.mean())
```

Ex 5.2. Here is some code to compute the gain for multiple bids. Explain how the function f works, in other words, how it handles the formatting of floating point numbers. The rest of the code must be clear to you, so you don't have to explain that.

Pretty printing floating points is something you really have to learn. If you unquote the line marked as 'unquote this line' and rerun the program, you'll see why people like to see formatted numbers.

Hint, search the web (python or R documentation) on formatting floating poinat numbers.

Python Code

```python
import numpy as np
from scipy.stats import uniform


def f(x):  # format floating point numbers
    # return f"{x}"  # unquote this line
    return f"{x:3.3f}"

```

```python

np.random.seed(3)


alpha = 2 / 3
n = 100
V = uniform(0, 1).rvs(n)

for b in np.arange(0.1, 1, 0.1):
    accepted = b > alpha * V
    gain = (V - b) * accepted
    print(f"{f(b)}, {f(gain.mean())}")
```

Ex 5.3. Use the code of the previous exercise as a starting point to make a graph of the gain as a function of the bid. Compute the bids for 0.05, 0.1, 0.15, etc, i.e., in steps of 0.05. Include your code, a figure, and comment on the graph. Use your student id as seed.

Ex 5.4. It is very easy to compute the expected gain by means of numerical integration. Run the code below and explain how the `res` variable is computed. Is the result similar to the simulation? Once again (so that you really remember what an anonymous function is!), explain very briefly why we use an anonymous function here.

Python Code

```python
import numpy as np
from scipy.integrate import quad

alpha = 2 / 3
n = 100
b = 0.5

res = quad(lambda v: (v - b) * (b > alpha * v), 0, 1)
print(res)
```

Ex 5.5. Modify the numerical integrator such that the prior on V is Beta(8, 2). Include your code and give the result for a bid of 1/2.

Python Code

```python
import numpy as np
from scipy.integrate import quad
from scipy.stats import beta

alpha = 2 / 3
n = 100
b = 0.5
prior = beta(8, 2)

```

```python
res = quad(lambda v: (v - b) * (b > alpha * v) * prior.pdf(v), 0, 1)
print(res)
```

5.2 *BH.9.1.7.The mystery box*

We use simulation to solve BH.9.7. Read it now, i.e., before reading the text below, then read the code below. Note how short this code is; amazing, isn't it?

Python Code

```python
import numpy as np
from scipy.stats import uniform
import matplotlib.pyplot as plt

np.random.seed(3)


N = 1000
a, b = 0, 1000_000
V = uniform(a, b).rvs(N)

x_range = np.linspace(b / 5, b / 2, num=50)
y = np.zeros_like(x_range)

for i, b in enumerate(x_range):
    payoff = (V - b) * (b >= V / 4)
    y[i] = payoff.mean()


plt.plot(x_range, y)
plt.show()
```

R Code

```r
set.seed(3)

N = 1000
a = 0
b = 1000000
V = runif(N, min = a, max = b)

x_range = seq(b / 5, b / 2, length.out = 50)
y = rep(0, length(x_range))


i = 1
```

```
for (b in x_range) {
  payoff = (V - b) * (b >= V / 4)
  y[i] = mean(payoff)
  i = i + 1
}

plot(x_range, y, type = "l", col = "blue")
```

Ex 5.6. For the python code use the scipy documentation to explain why $V \sim \text{Unif}([0,10^6])$. For R, explain the save fo `runif`.

Ex 5.7. What are the smallest and the largest value of `x_range`?

Ex 5.8. Run the code above and make a graph. Include the graph in your report, and explain what you see in the graph. For instance, is there a maximum? If so, can you explain where the maximum occurs? Can you explain how the maximum should be?

Ex 5.9. Suppose after seeing the graph of the payoffs, and this graph would only increase, or decrease, how would you change `x_range`? Do you expect to see a maximum?

Ex 5.10. For N small, e.g. N=10, you can get quite strange values. Why is that?

Ex 5.11. Change the acceptance threshold from to $V/4$ to $V/5$ (or $V/6$, or some other value you like), and make a graph of the payoffs. Include the graph in your report.

Ex 5.12. Change the payoff function to e.g $\sqrt{V-b}$, or some weird function that you like particularly such as $\sin|V-b|$ (any non-trivial function goes). Make a graph of the mean and std of the payoff. Can you explain your graph?

5.3 *BH.9.1.9*

This is a nice variation of one-step analysis of the coin throwing example of BH.9.1.9.

We have a mouse that sits on one corner of a cube whose edges are made of wire, and diagonally opposite the mouse there is some cheese. The mouse chooses at random (uniform) any edge and moves to the next corner. How many steps does it take, on average, for the mouse to reach the cheese (assuming the cheese does not move)?

For ease, we label assemble all edges with the same minimal number of edges into states. Thus, the starting corner will be called state 3, then we have a state 2, and 1, and finally state 0 is the corner with the cheese.

Ex 5.13. Explain in the code below how the while loop simulates the moves of the mouse. To understand the line marked as 'this', read what is a, so-called, ternary operator.

Python Code

```python
import numpy as np
from scipy.stats import uniform
```

```python
import matplotlib.pyplot as plt

move = uniform()


def do_run():
    state = 3
    n_steps = 0
    while state != 0:
        n_steps += 1
        p = move.rvs()
        if state == 3:
            state = 2
        elif state == 2:
            state = 1 if p < 2 / 3 else 3 # this
        elif state == 1:
            state = 2 if p < 2 / 3 else 0
    return n_steps


n = 200
steps = np.zeros(n)
for i in range(n):
    steps[i] = do_run()

print(steps.mean(), steps.std())
plt.hist(steps, density=True)
plt.tight_layout()
plt.savefig("figures/mouse.pdf")
```

5.4 *BH.9.6.1*

Read BH.9.6.1 first. We will redo the computations and check the results with simulation. We will also cover many other topics as we proceed.

Ex 5.14. We need to model the distribution of an individual demand X. As an example, we can use the beta distribution. In the example μ and σ are assumed as given, but for the beta distribution we need an a and b. One way is to solve by hand for a and b for given μ and σ. However, we can also be lazy and let the computer do the work. Here is the code to do that.

Explain how this code works. (You can use the relevant documentation on the web on how the function fsolve works.) Why do I use $\mu = 1/2$ and $\sigma^2 = 1/12$ as test case? Are the a and b that comes out of the solver correct? What does the final check do?

Python Code

```python
from scipy.stats import beta
from scipy.optimize import fsolve

mu, var = 1 / 2, 1 / 12  # why this test case?


def func(x):
    a, b = x[0], x[1]
    m = a / (a + b)
    v = m * (1 - m) / (a + b + 1)
    return [m - mu, v - var]


a, b = fsolve(func, [2, 2])
print(a, b)
X = beta(a, b)
print(X.stats(moments='mv')) # final check.
```

Ex 5.15. What do you expect to get when you would choose $\mu = 2$? If it fails, why is that?

Ex 5.16. What do you expect to get when you would choose $\sigma > 1/2$? If it fails, why is that? Hint, when an rv has a support on $[0,1]$, why is the variance at most $1/4$?

Remark 5.17. When setting up simulations and choosing variables, make sure that the variables you choose actually make any sense.

Now we know how to find a and b for given μ and σ, we henceforth assume that $a = 1$ and $b = 2$, as this is easier to work with.

Ex 5.18. Now that we know how to simulate individual demands, we can compute the mean and the variance with the formulas of the book and compare this to simulation.

Explain the lines marked as 'this' of the code. (The rest must be clear for you. If not, then you have to be seriously concerned about your understanding of programming.)

Python Code

```python
import numpy as np
from scipy.stats import poisson, beta
from scipy.optimize import fsolve

np.random.seed(3)

labda = 3
N = poisson(labda)
# print(N.rvs(100).mean()) # why this line?

```

```python
X = beta(a=1, b=2)
mu, var = X.stats(moments='mv')

days = 1000
demands = np.zeros(days)
for i in range(days):
    demands[i] = X.rvs(N.rvs()).sum()  # this

print(demands.mean(), demands.var())
print(labda * mu, var * labda + mu * mu * labda) # this
```

Ex 5.19. Here is the code to compute the mean and variance with Adam and Eve's laws. Explain how it works. In particular:

1. explain the 'this' lines;
2. what are the contents of the demand matrix;
3. what is the P;
4. how are EX,EV and VE computed;
5. Note in particular the difference between EXn and EXN, and similarly for the variance. Use BH.9.2.1 to explain why I use a small n and a large N in the naming of the variables.

Remark: I realize that this is a hard piece of code, but your understanding of Eve's law will increase by a lot. Some subtle points about the code that you *don't* have to explain:

1. Suppose the largest amount of customers (jobs) that we saw during the simulation is 7. Then on any day we can have $0,\dots,7$ demands, i.e, 8 possible outcomes. Thus, per row in the demands matrix we need to have 8 columns.
2. The numpy function that computes the mean gives a warning when it is applied to an empty array. (Just try it to see how it works.) We prevent this by computing the histogram (i.e., the empirical PDF) for $n = 0$ separately. The same reasoning applies to the case when idx has zero length.

Python Code

```python
import numpy as np
from scipy.stats import poisson, beta

np.random.seed(3)

labda = 3
N = poisson(labda)

```

```python
X = beta(a=1, b=2)
mu, var = X.stats(moments='mv')

days = 1000
n_jobs = N.rvs(days)
n_max = n_jobs.max() + 1  # see the comments
demands = np.zeros((days, n_max))
for day in range(days):
    n = n_jobs[day]
    demands[day, :n] = X.rvs(n)  # this


P = np.zeros(n_max)
EXn = np.zeros(n_max)
VXn = np.zeros(n_max)
P[0] = np.argwhere(n_jobs == 0).size / days
for n in range(1, n_max):
    idx = np.argwhere(n_jobs == n)
    if idx.size == 0:
        continue
    P[n] = idx.size / days  # this
    EXn[n] = demands[idx, :n].mean() * n  # this
    VXn[n] = demands[idx, :n].var() * n  # this

EXN = rv_discrete(values=(EXn, P))  # this

print("mean:", labda * mu, EXN.mean())

VXN = rv_discrete(values=(VXn, P))  # this
print("EV: ", var * labda, VXN.mean())

VE = EXN.var()
print("VE: ", mu * mu * labda, VE)
```

5.5 *BH.9.1*

Here is the code.

Python Code

```python
import numpy as np
from scipy.stats import randint, norm

mus = np.array([2, 5, 8])
stds = np.array([0.1, 0.5, 1])

```

```python

m2 = stds ** 2 + mus ** 2 # this
v = m2.mean() - (mus.mean()) ** 2 # this
print(mus.mean(), v)

routes = []
for i in range(3):
    routes.append(norm(mus[i], stds[i])) # this


num = 200
options = randint(0, 3).rvs(num) # this

times = np.zeros(num)  # this
for i in range(num):
    times[i] = routes[options[i]].rvs() # this

print(times.mean(), times.var())
```

Ex 5.20. Explain the lines marked as 'this', i.e., how does the program work?

Ex 5.21. Run the code and explain the results. Are the simulated results in line with the theory?

Ex 5.22. Modify the code so that you can plot the histogram of the `times` vector. Include your code and a figure. Set also a seed equal to, e.g., the day of the month in which you do the assignment.

Ex 5.23. Set μ_8 to 80 instead of 8. Rerun the code and explain what happens to the mean and the variance. Then change to 800 and include a histogram of the PDF of the times.

Ex 5.24. Change σ_1 from 0.1 to 10. Run the code and describe what you get. What's wrong with such a large std? The route lengths can become negative.

6 ASSIGNMENT 6

6.1 *BH.9.25*

This simulation exercise is based on BH.9.25. Please read the exercise first, and then the code below.

Ex 6.1. How does this code work? Use the print statements to find out what S is, and explain why we multiply by 2 and subract 1 in one of the lines marked as 'this'.

Python Code

```python
import numpy as np
from scipy.stats import bernoulli

np.random.seed(3)

n = 5
num = 10

p = 0.5
S = bernoulli(p).rvs([num, n]) * 2 - 1 # this
# print(S)

x = np.zeros([num, n])
x[:, 0] = 100  # this
# print(x)
f = 0.25

for i in range(1, n):
    x[:, i] = x[:, i - 1] * (1 + f * S[:, i]) # this

print(x.mean(axis=0), x.std(axis=0))
```

R Code

```r
set.seed(3)

n = 5
num = 10

p = 0.5
S = matrix(rbinom(n * num, size = 1, prob = p), num, n) * 2 - 1

x = matrix(0, num, n)
x[, 1] = 100
f = 0.25
```

```

for (i in 2:n) {
  x[, i] = x[, i - 1] * (1 + f * S[, i])
}

print(colMeans(x))
print(apply(x, MARGIN = 2, sd))
```

Ex 6.2. 1. Run the code below.

2. Check the values of `x`. What do you notice?

3. Then compare both figures. You should see that in `Kelly-no-jitter.pdf` many of the markes (the x below the y-axis) fall on top of each other. To repair for this, we add some noise (jitter) to the x and y values of the markes. Explain how randomness is used here.

4. Jitter plots, like this one, are meant to give some more insight into the data to check whether we use the right number of bins. Change the number of bins to 2, include the plot with the jitters, and explain how the marks help to see that two bins is much too small.

Python Code

```python
import numpy as np
from scipy.stats import bernoulli, uniform
import matplotlib.pyplot as plt


np.random.seed(3)

n = 10
n_experiments = 50

p = 0.4
S = bernoulli(p).rvs([n, n_experiments]) * 2 - 1  # this
# print(S)

x = np.zeros([n, n_experiments])
x[0, :] = 100  # this
f = 0.25

for i in range(1, n):
    x[i, :] = x[i - 1, :] * (1 + f * S[i, :])  # this

print(x[-1, :].mean(), x[-1, :].std())
print(x[-1, :])
```

```python

plt.hist(x[-1, :], bins=20, density=True)
plt.plot(x[-1, :], np.full_like(x[-1, :], -0.01), "xk", markeredgewidth=1)
plt.savefig("figures/Kelly-no-jitter.pdf")


x_jitter = x[-1, :] + uniform(loc=-3, scale=6).rvs(size=n_experiments)
y_jitter = np.full_like(x[-1, :], -0.01) + uniform(loc=-0.01, scale=0.02).rvs(
    size=n_experiments
)
plt.plot(x_jitter, y_jitter, "xk", markeredgewidth=1)
plt.savefig("figures/Kelly-jitter.pdf")
```

6.2 *BH.9.37*

With this exercise we illustrate why (and how) to use bootstrapping. At first it appears a bit strange: we have a set of observations (samples) $X = \{X_1, X_2, \dots, X_n\}$. For this we can compute the mean, std, and the empirical distribution with standard procedures, so why sample (bootstrap) from X?

One type of question that seems hard to answer by classical means is to characterize the median, in particular the std and distribution of the median, of the population (that is, the population from which we took the sample X).

Bootstrapping is nowadays used a lot in datascience.

Ex 6.3. Run the code below, and then explain what is `sample`, `bootstrap` and `medians`. Why do we use `axis=1`?

Ex 6.4. Change some numbers (such as the range of the random numbers from which we obtained the initial sample). Include your code, make the plot, and include it in your assignment.

Python Code

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(3)

sample = np.random.randint(200, size=100)
n_boot = 5000
bootstrap = np.random.choice(sample, replace=True, size=(n_boot, len(sample)))

medians = np.median(bootstrap, axis=1)
# print(medians.mean(), medians.std())
# print(np.percentile(medians, [2.5, 97.5]))

```

```python
plt.hist(medians, bins=100, density=True)
plt.axvline(np.percentile(medians, 2.5), c='r', lw=2)
plt.axvline(medians.mean(), c='r', lw=2)
plt.axvline(np.percentile(medians, 97.5), c='r', lw=2)
plt.savefig("figures/bootstrap.pdf")
```

Ex 6.5. Finally, let us compare the information we obtained from bootstrapped to the medians we would obtain when we would sample from the real population.

1. What is here the real population?
2. In a real life situation, what is less costly: real sampling or bootstrapping?
3. Explain the ideas of the code below.
4. Compare this figure to the one obtained by bootstrapping. Comment on major differences.

Python Code

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(3)

samples = np.random.randint(200, size=(500, 100))
medians = np.median(samples, axis=1)
plt.hist(medians, bins=100, density=True)
plt.savefig("figures/real_medians.pdf")
```

6.3 *BH.9.50*

This simulation exercise is based on BH.9.25. Please read the exercise first. Do the exercise while thinking about the code.

Ex 6.6. Here is some code to simulate N. Explain the non-trivial steps. (By now you should have an idea about boiler-plate code and what code is specific for the problem.)

Python Code

```python
import numpy as np
from scipy.stats import poisson, expon

np.random.seed(3)

num = 100
N = np.zeros(num)
```

```python
Labda = expon(scale=1)
labdas = Labda.rvs(num)
for i in range(num):
    labda = labdas[i]
    N[i] = poisson(labda).rvs()

print(N.mean(), N.var())
adam = Labda.mean()
eve = Labda.mean() + Labda.var()
print(adam, eve)
```

Ex 6.7. Run the above code and compare the result to the theoretical value. What do you observe? Rerun the code with `num = 10000`. Does this result in a better estimate? What do you learn from this?

Ex 6.8. The next step is to estimate the mean and variance of the total claim size S. Here is the code, run it and explain the relevant parts.

Python Code

```python
import numpy as np
from scipy.stats import poisson, expon, lognorm

np.random.seed(3)

num = 100
Labda = expon(scale=1)
labdas = Labda.rvs(num)
mu, sigma = 3, 1
X = lognorm(loc=mu, s=sigma)
S = np.zeros(num)
for i in range(num):
    N = poisson(labdas[i]).rvs()
    S[i] = X.rvs(N).sum()

print(f"{S.mean()=}")
adam = X.mean() * Labda.mean()
print("mean_theoretical: ", adam)
print(f"{S.var()=}")
eve = Labda.mean() * X.var() + X.mean() ** 2 * (Labda.mean() + Labda.var())
print("var_theoretical: ", eve)
```

Ex 6.9. Here is the code to plot the distribution. Change λ such that it is $\sim \text{Exp}(1/2)$ and include the figure.

Python Code

```python
import numpy as np
from scipy.stats import poisson, expon
```

```python
import matplotlib.pyplot as plt

np.random.seed(3)

num = 100
N = np.zeros(num)
Labda = expon(scale=1)
labdas = Labda.rvs(num)
for i in range(num):
    labda = labdas[i]
    N[i] = poisson(labda).rvs()

plt.hist(N, density=True)
plt.savefig("figures/bh-9.50-fig.pdf")
```

Ex 6.10. Finally, let's do the integration numerically to compute $\mathsf{P}\{N=3\}$. Explain the code. Why, in particular, do I compare the result against `geom(1/2).pmf(4)`? (Hint, read the documentation of `scipy.stats.geom` to understand why.)

Python Code

```python
import numpy as np
from scipy.integrate import quad
from scipy.stats import expon, geom

Labda = expon(scale=1)

def integrand(labda):
    return np.exp(-labda) * labda ** 3 / 6 * Labda.pdf(labda)

res = quad(lambda labda: integrand(labda), 0, np.infty)
print(res)
print(geom(1 / 2).pmf(3))
print(geom(1 / 2).pmf(4))
```

6.4 *BH.10.2.3*

Let us try to understand the weak law of large numbers by means of simulation. An easy example is to take $X_i \sim \text{Unif}(0,1)$, so that is what we do here.

Python Code

```python
import numpy as np
from numpy.random import uniform

np.random.seed(3)

```

```
n = 10
N = 50 # num samples

mu = 1 / 2
var = 1 / 12
eps = 0.1

X = uniform(size=[num_samples, n])

Y = X.mean(axis=1)

larger = np.abs(Y - mu) > eps
count = larger.sum()
P = count / N
RHS = var / (n * eps * eps)
print(P, RHS)
```

R Code

```
set.seed(3)

n = 10
N = 50

mu = 1 / 2
var = 1 / 12
eps = 0.1

X = matrix(runif(N * n), N, n)

Y = rowMeans(X)

larger = abs(Y - mu) > eps
count = sum(larger)
P = count / N
RHS = var / (n * eps * eps)
print(P)
print(RHS)
```

Ex 6.11. Explain mu and var.

,

Ex 6.12. What is Y? What is the symbol that BH use for this?

Ex 6.13. What are the meanings of larger and count?

Ex 6.14. What is `RHS` in the notation of BH?

Ex 6.15. What inequality of BH do we check by printing `RHS` and `P`?

Ex 6.16. Choose some different values for n and the sample size N. Is the inequality always true?

7 ASSIGNMENT 7

7.1 *BH.10.9*

Ex 7.1. In the code below we compute or simulate the various expressions of BH.10.9. Check the results and compare it to the analytical results. Are both results in line? If not, do you know why that might be the case? What can you do about that?

I don't know how to build $\mathsf{E}[\mathsf{V}[Y|X]]$ in any sensible way. Since Y is independent of X, the conditioning does not have any influence. Thus, this numerical exercise is skipped.

Python Code

```python
import numpy as np
from scipy.stats import expon

np.random.seed(3)

X = expon(2)
Y = expon(2)

print(np.exp(-X.mean()), X.expect(lambda x: np.exp(-x)))

n_sample = 1000
X_sample = X.rvs(n_sample)
Y_sample = Y.rvs(n_sample)
p1 = sum(X_sample > Y_sample + 3) / n_sample
p2 = sum(Y_sample > X_sample + 3) / n_sample
print(p1, p2)

p3 = sum(X_sample > Y_sample - 3) / n_sample
print(p1, p3)

EXY = X_sample @ Y_sample / n_sample
EX4 = X.expect(lambda x: x ** 4)
print(EX4, EXY * EXY)

p = sum(np.abs(X_sample + Y_sample) > 3) / n_sample
print(p, X.mean())
```

Ex 7.2. Now make the plot and explain what is wrong.

Ex 7.3. The line marked as 'this' should be replaced with

Python Code

```python
plt.plot(supp, pmf * np.sqrt(n))
```

Explain why. (You should memorize that comparing PMFs and PDFs is not straightforward.)

Ex 7.4. Why do I use simulation to estimate $\mathsf{E}[XY]$? Recall an earlier assignment in which we used numerical integration. How successful was that?

Ex 7.5. What would you do to increase the credibility of the claims. (Note these claims are based on case checking and simulation. There is no actual proof, however, they can act as counter-examples.)

7.2 *BH.10.28*

Ex 7.6. Let us first plot the PMF of the standardized version of X_n for $n = 10$. Explain the code below BEFORE making the plot.

Python Code

```python
import numpy as np
from scipy.stats import poisson, norm
import matplotlib.pyplot as plt

n = 10
X = poisson(n)
N = norm()

pmf = np.zeros(2 * n)
for i in range(len(pmf)):
    pmf[i] = X.pmf(i)
supp = (np.linspace(0, len(pmf), num=len(pmf)) - n) / np.sqrt(n)


plt.plot(supp, pmf)  # this
plt.plot(supp, norm.pdf(supp))
plt.savefig("figures/bh-10-28.pdf")
```

Ex 7.7. Now make the plot and explain what is wrong.

Ex 7.8. The line marked as ‘this’ should be replaced with

Python Code

```python
plt.plot(supp, pmf * np.sqrt(n))
```

Explain why. (You should memorize that comparing PMFs and PDFs is not straightforward.)

Ex 7.9. Make the plot of $n = 2$ and $n = 20$, include both plots and explain the results.

Ex 7.10. The code below computes $M_Y(s)$ where Y is the standardized version of X_n. Then it compares M_Y to the MGF of the standard normal distribution.

1. Explain how it works.

Python Code

```python
import numpy as np
from scipy.stats import poisson, norm
import matplotlib.pyplot as plt

n = 10
X = poisson(n)
num = 50
S = np.linspace(-1, 1, num)
M = np.zeros(num)
pmf = X.pmf(range(100)) # this

for i in range(num):
    for j in range(len(pmf)):
        M[i] += np.exp(S[i] * (j - n) / np.sqrt(n)) * pmf[j]


plt.plot(S, M, label="M X")
plt.plot(S, np.exp(S * S / 2), label="M N")
plt.xlabel("s")
plt.legend()
plt.savefig("figures/bh-10-28-mgf.pdf")
```

Ex 7.11. The line marked as 'this' computes the PMF upfront. Why is that a good idea? Why do we stop the computation of the PMF at 100?

Ex 7.12. In the code below, one of the for loop is removed.

1. What is the change?

2. Will this change have an effect on the speed of the computation?

3. Which of the two alternatives do you find more readible?

Python Code

```python
import numpy as np
from scipy.stats import poisson, norm
import matplotlib.pyplot as plt

n = 10
X = poisson(n)
num = 50
```

```python
S = np.linspace(-1, 1, num)
M = np.ones(num)

for i in range(num):
    M[i] = X.expect(lambda j: np.exp(S[i] * (j - n) / np.sqrt(n)))

plt.plot(S, M, label="M X")
plt.plot(S, np.exp(S * S / 2), label="M N")
plt.xlabel("s")
plt.legend()
plt.savefig("figures/bh-10-28-mgf.pdf")
```

Ex 7.13. Change n to 20 and to 50. Does the quality improve? Choose some value for n that you like, and include the graph.

7.3 *BH.10.30*

Ex 7.14. Explain how the code below implements the solution of parts a and b BH.10.30. (Of course we skip the proof of the convergence to $g(\alpha)$ here.)

Python Code

```python
import numpy as np
from scipy.stats import bernoulli
import matplotlib.pyplot as plt

np.random.seed(3)

num = 100
p, alpha = 0.5, 0.5
revenue = 0.5 + 1.2 * bernoulli(p).rvs(num)

Y = np.ones(num)
Y[0] = 1
for i in range(1, num):
    Y[i] = (1 - alpha + alpha * revenue[i]) * Y[i - 1]

print(np.log(Y[-1]) / num)

# plt.plot(Y) # this

def g(alpha):
    return 0.5 * np.log((1 + 0.7 * alpha) * (1 - 0.5 * alpha))


plt.ylim(-10 * g(alpha), 10 * g(alpha)) # this
plt.axhline(g(alpha), c='r', lw=2, label="g")
plt.plot(np.log(Y)[1:] / range(1, num), label="(log Yn)/n")
plt.legend()
plt.savefig("figures/bh-10-30.pdf")
```

Ex 7.15. Uncomment the relevant line to obtain a plot of Y_n as a function n. Make the plot and include it in your assigment.

Ex 7.16. Take `num` much larger, e.g., 100 000. What is $(\log Y_n)/n$ now?

Ex 7.17. Print $(\log Y_n)/n$ for various values of α. For instance, take $\alpha = 1/10, 2/10, 3/10, 4/10$.

Ex 7.18. If you like a simple challenge, include a plot of $(\log Y_n)/n$ as a function of α. However, skip this if you don't have time, or interest in this extension of the problem. (Perhaps, if you find programming hard, you should do it to improve your skills :-))

7.4 *BH.10.39*

BH.10.39.a asks about the first time such that some exponential rv exceeds a certain threshold. Part b is about when the sum of a number of r.v.s exceed a threshold. Such problems are called *hitting times*.

Python Code

```python
import numpy as np
from scipy.stats import expon

np.random.seed(3)

X = expon(scale=1)

# part a

N = 10
res = np.zeros(N)
for i in range(N):
    n = 0
    while X.rvs() < 1:
        n += 1
    res[i] = n

print(res.mean(), res.std())

# part b

N = 100
```

```python
res = np.zeros(N)
for i in range(N):
    M = X.rvs()
    n = 1
    while M < 10:
        M += X.rvs()
        n += 1
    res[i] = n

print(res.mean(), res.std())
```

R Code

```r
set.seed(3)

# part a

N = 10
res = rep(0, N)
for (i in 1:N) {
  n = 0
  while (rexp(1, rate = 1) < 1) {
    n = n + 1
    res[i] = n
  }
}

print(mean(res))
print(sd(res))

# part b

N = 100
res = rep(0, N)
for (i in 1:N) {
  M = rexp(1, rate = 1)
  n = 1
  while(M < 10) {
    M = M + rexp(1, rate = 1)
    n = n + 1
    res[i] = n
  }
}

print(mean(res))
print(sd(res))
```

By now you have so much experience with reading code that you must be able to explain all without intermediate steps to guide you.

Ex 7.19. Explain how the first part of the code simulates BH.10.39.a.

Ex 7.20. Explain how the second part of the code simulates BH.10.39.b.