Σ	A1	A2
8	3	5

Blatt03_Abgabe

May 21, 2024

1 Blatt 03

Code für a), c) sieht gut aus.

1.1 Aufgabe 5

In Teilaufgabe b) soll für beliebige a bei festen Werten von m und c bestimmt werden, wie groß die Periodenlänge ist. Dies wird für a=[1,7] ausgetestet und man erhält für a=1 die höchste Periodenlänge bei 1023. Es wurde versucht Werte größer als 10 einzusetzen oder mit kleineren Werten die Liste zu vergrößern, jedoch beansprucht der Kernel viel Zeit, sodass es bei a=1,7 belassen wurde. Dementsprechend scheint es für kleine Werte von a größere PEriodenlängen zu geben.

[1]:
```
import project_c1
from project_c1.random import LCG
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from IPython.display import Code
from IPython.display import IFrame
```

[2]:
```
a = [1, 7]
for i in range(0, len(a)):
    Test = LCG(0, a=a[i], c=3, m=1024)
    seed = Test.state
    Test.advance()
    c = 0
    while seed != Test.state:
        Test.advance()
        c += 1
    print(f"Die Periodenlänge für {a[i]} beträgt {c}")
```

```
Die Periodenlänge für 1 beträgt 1023
Die Periodenlänge für 7 beträgt 255
```

Es gibt auch Werte für a, die m=1024 haben. a=1 ist aber trotzdem keine gute Wahl, weil die "Zufallszahlen" vorhersehbar sind → 3, 6, 9, ...

[3]:
```
a_w=10
Test2=LCG(0, a_w, c=3, m=1024)
seed1 = Test.state
Test2.advance()
v = 0
#Es folgt eine Endlosschleife, um zu demonstrieren, dass man a für hohe Werte␣
 ↪nicht austesten kann.
```

-1p.

1. 10 ist kein „hoher Wert“. Für bspw. 11 erhaltet ihr eine Periodenlänge von 511.
2. Debuggen: Wenn ihr euch die Schleife mal printen lasst, seht ihr, dass alle Zufallszahlen für manche Werte von a identisch sind. Für a=10 liegen z.B. alle „Zufallszahlen“ bei 341.

Es gibt also Startwerte, die in einen stationären Zustand fallen.

```
#while seed1 != Test2.state:
#    Test2.advance()
#    v += 1
#print(f"Die Periodenlänge für {a_w} beträgt {v}")
```

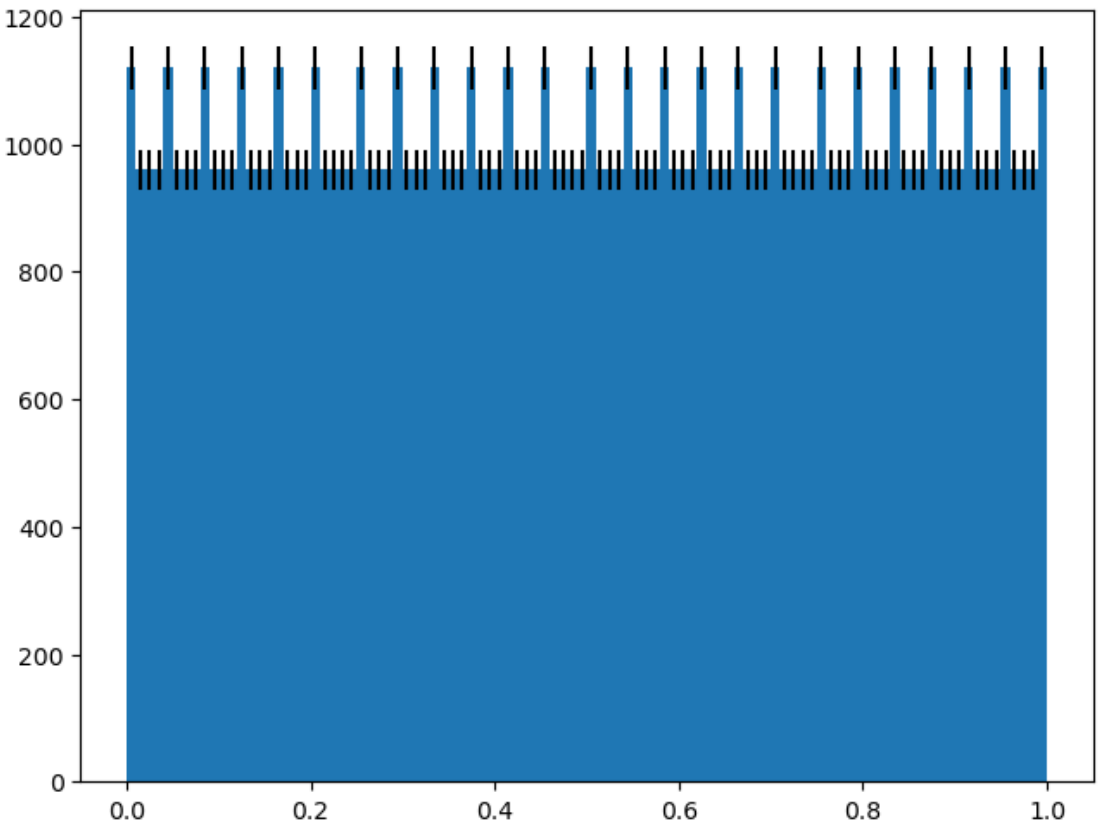
Für die Teilaufgabe d) wird ein Histogramm für die uniform- Verteilung der LCG- genererierten Zahlen mit 100 Bins erstellt. Es wird ersichtlich, dass bei konstanten Abständen bestimmte Floats mit derselben Häufigkeit öfters auftreten als andere Werte. Die konstanten Abstände implizieren eine feste Anordnung der Zahlen bzw. häufiges Durchlaufen von Perioden, die jedoch nicht im 1D- Spectral test explizit betrachtet wird. Somit scheint mit dem häufigen Auftreten bestimmter Reihenfolge kein guter Generator zu sein.

Ändert man den Seed, so erhält man einen anderen Wert für die Periodenlänge (siehe b)). Wenn ein Seed gewählt wird, bei der die Periodenlänge sich erhöht, so sinkt die Häufigkeit der Peaks (und die restlichen Häufigkeiten erhöhen sich), sodass eine Gleichverteilung höheren Ausmaßes vorliegt. Wenn im umgkehrten Fall ein Seed gewählt wird, bei der die Periodenlänge minnimiert wird, so steigt die Häufigkeit der Peaks und eine Gleichverteilung liegt "weniger" vor.

[4]:
```
Testd = LCG(0, a=1601, c=3456, m=10000)
c = Testd.uniform(0, 1, 100000)
print(c.dtype)
fig1, ax1 = plt.subplots(constrained_layout=True)
# limit = [0, 1]
bins = 100
hist_options = dict(
    bins=bins,
    # range=limit,
    histtype="step",
)
# plt.hist(c, label="Uniform-Distribution", **hist_options)
# plt.show()
# Oder
y, binEdges = np.histogram(c, bins=bins)
bin_centers = 0.5 * (binEdges[:-1] + binEdges[1:])
yerr = np.sqrt(y)
width = np.diff(binEdges)
ax1.bar(bin_centers, y, width=width, yerr=yerr)
```

```
float64
```

[4]:
```
<BarContainer object of 100 artists>
```



Für die Teilaufgabe e) wird ein 2D- und ein 3D- Scatterplot zu den Zufallszahlen erstellt. Es wird bei den 2D PLot ersichtlich, dass das Gitter gleichmäßig besetzt wird und ein konstanter Abstand zwischen den parallelen Linien vorliegt, sodass es sich unter Berücksichtigung der Tupel sich um einen guten Generator handeln kann. Bei dem 3D- Scatterplot sind 3 Ebenen erkennbar, zwischen denen ebenfalls ein konstanter Abstand vorliegt, sodass unter Betrachtung von Tripel der LCG "gute" Zufallszahlen generieren kann. Der LCG könnte jedoch besser sein, wenn sich der konstante Abstand minimiert und die Anzahl der parallelen Ebenen bzw. parallelen Linien erhöht, sodass eine "stärkere" Gleichverteilung vorliegt wie es bei dem von Numpy generierten Zufallszahlen zu sehen ist (siehe f)).

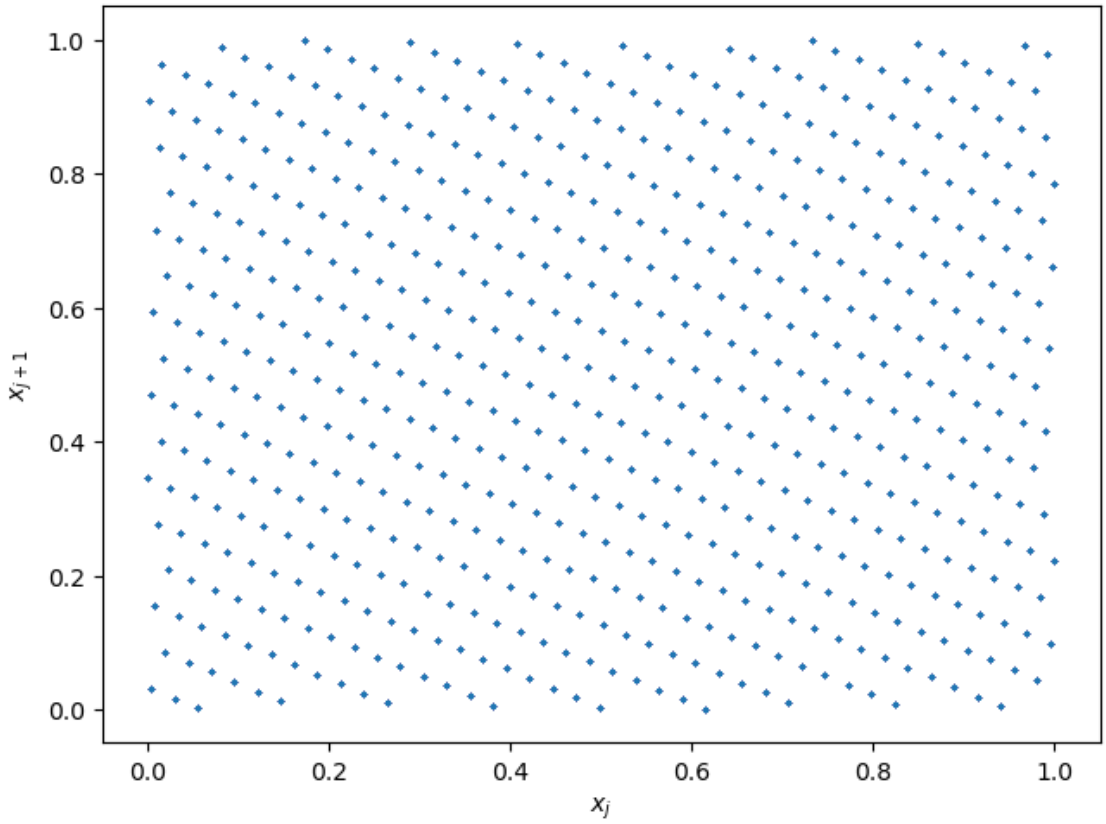
Inwiefern ist es gut, wenn Zufallszahlen an vorhersehbaren "Orten" sitzen? (bzw. wenn floats eine abzählbare Menge an Zahlen generiert?) -1p.

[5]:
```
d = c[1:]
c_2d = c[:-1]
fig2, ax2 = plt.subplots(constrained_layout=True)
ax2.set_xlabel(r"$x_{j}$")
ax2.set_ylabel(r"$x_{j+1}$")

ax2.scatter(
    c_2d,
    d,
    s=5,  # smaller dots,
    alpha=0.2,  # some transparency for this many points
```

→ Abstand ist egal. Es darf kein "Muster" geben.

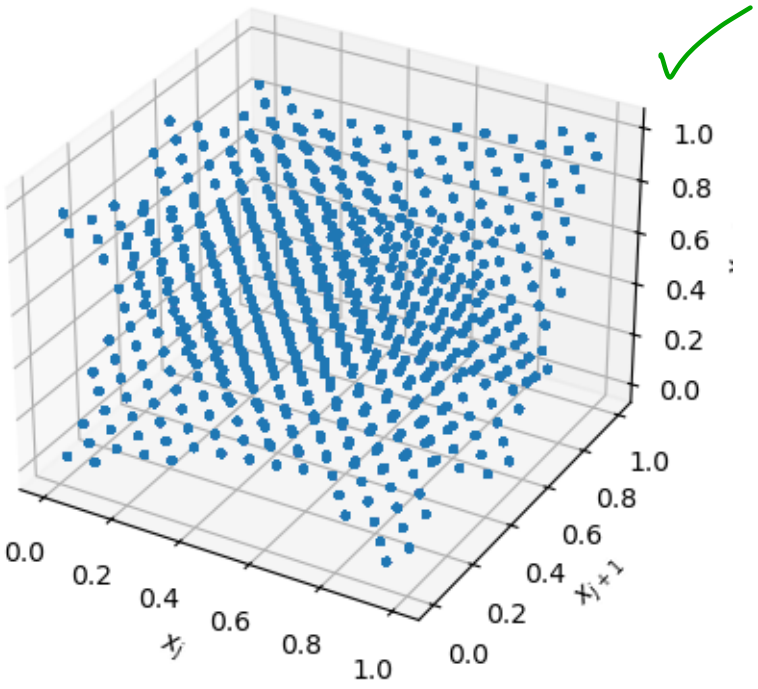
```
    linewidth=0,  # no border around points
)
```

[5]: <matplotlib.collections.PathCollection at 0x7f1ca67f57f0>



[6]:
```
fig3 = plt.figure()
ax3 = fig3.add_subplot(1, 1, 1, projection="3d")
c_3d = c[:-2]
d_3d = d[:-1]
e = c[2:]
ax3.set_xlabel(r"$x_{j}$")
ax3.set_ylabel(r"$x_{j+1}$")
ax3.set_zlabel(r"$x_{j+2}$")
ax3.scatter(c_3d, d_3d, e, s=5, alpha=0.3)
```

[6]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f1caea17830>



Bei Teilaufgabe f) werden Zufallszahlen vdurch numpy.random generiert und jeweils die Plots aus d) und e) erstellt. Bei dem Histogramm werden keine hohen, stark ausgeprägten Peaks ersichtlich wie bei LCG. Zudem liegt eine stärkere Verteilung der Daten bzw. eine starke Gleichverteilung in den Scatterplots vor vor, sodass der Numpy.random Generator deutlich besser ist als der LCG.

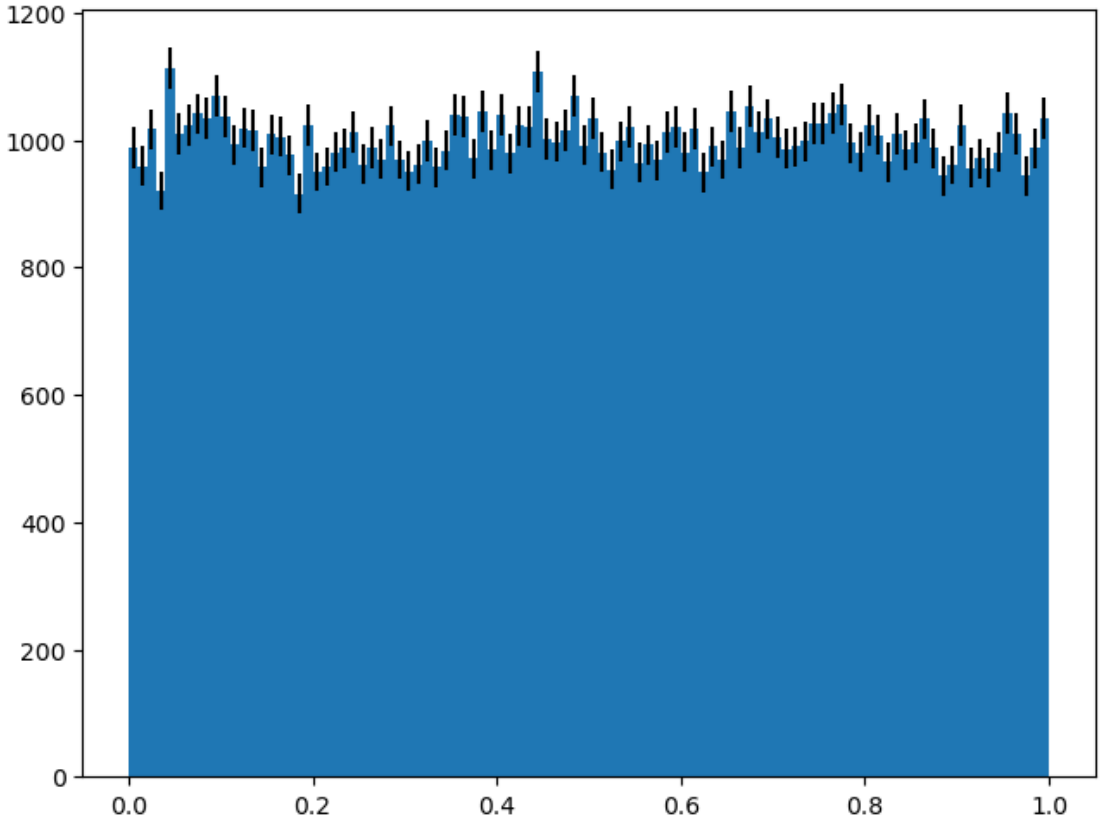
[7]:
```
#Kreiere random numbers mit np.random.default_rng()
rng = np.random.default_rng(0)
rints = rng.random((100000, 1)) #generates floats between 0 and 1
print(rints)
print(rints[10])
```

```
[[0.63696169]
 [0.26978671]
 [0.04097352]
 ...
 [0.77706236]
 [0.57903921]
 [0.50626261]]
[0.81585355]
```

[8]:
```
#Aufgabe d) random Generator mit histogramm
fig1, ax1 = plt.subplots(constrained_layout=True)
# limit = [0, 1]
bins = 100
```

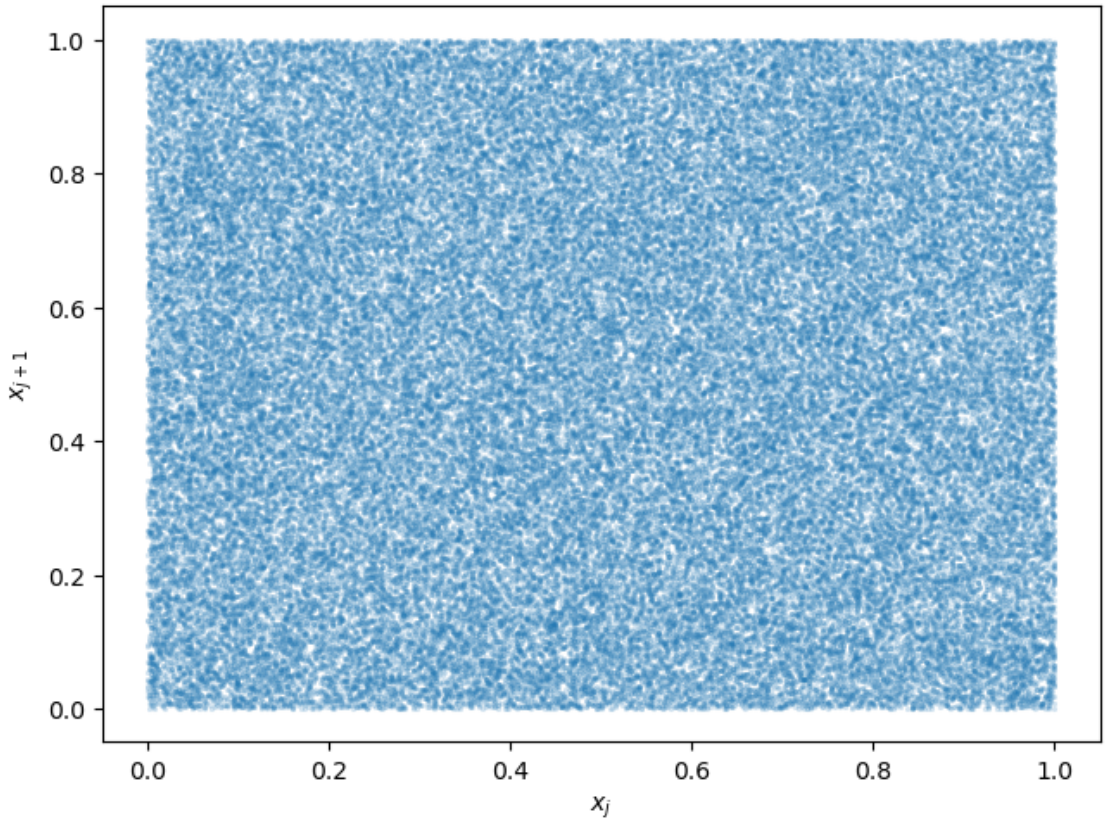
```
y, binEdges = np.histogram(rints, bins=bins)
bin_centers = 0.5 * (binEdges[:-1] + binEdges[1:])
yerr = np.sqrt(y)
width = np.diff(binEdges)
ax1.bar(bin_centers, y, width=width, yerr=yerr)
```

[8]: <BarContainer object of 100 artists>



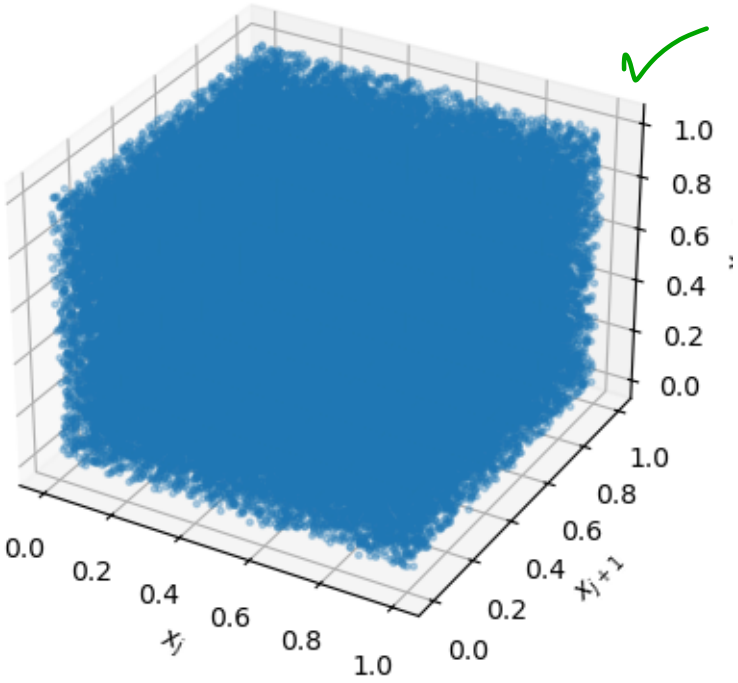
[9]:
```
rints_2d = rints[:-1]
d = rints[1:]
fig2, ax2 = plt.subplots(constrained_layout=True)
ax2.set_xlabel(r"$x_{j}$")
ax2.set_ylabel(r"$x_{j+1}$")
ax2.scatter(
    rints_2d,
    d,
    s=5,  # smaller dots,
    alpha=0.2,  # some transparency for this many points
    linewidth=0,  # no border around points
)
```

[9]: <matplotlib.collections.PathCollection at 0x7f1ca61d0920>



[10]:
```
fig3 = plt.figure()
ax3 = fig3.add_subplot(1, 1, 1, projection="3d")
ax3.set_xlabel(r"$x_{j}$")
ax3.set_ylabel(r"$x_{j+1}$")
ax3.set_zlabel(r"$x_{j+2}$")
rints_3d = rints[:-2]
d_3d = d[:-1]
e = rints[2:]
ax3.scatter(rints_3d, d_3d, e, s=5, alpha=0.3)
```

[10]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f1ca48ec1a0>



3/5

1.2 Aufgabe 6

a) Berechnung von N:

$$1 = \int_0^\infty f(x)\,\mathrm{d}x$$

$$\Longrightarrow \int_0^\infty N e^{-x/\tau}\,\mathrm{d}x = 1$$

$$= N\left[-\tau e^{-x/\tau}\right]_0^\infty = N\tau$$

$$\Longrightarrow N = \frac{1}{\tau}$$ ✓

Dann zur Berechnung der Funktion, die implimentiert wird:

$$\int_0^y \frac{1}{\tau} e^{-x/\tau}\,\mathrm{d}x = \frac{1}{\tau}\left[-\tau \cdot e^{-x/\tau}\right]_0^y = -e^{-y/\tau} + 1 = g(y)$$ ✓

$$\Longrightarrow \ln(-g+1) = -\frac{y}{\tau} \iff y = -\tau \cdot \ln(1-g)$$ ✓

b) Berechnung von N:

$$\int_{x_{\min}}^{x_{\max}} N \cdot x^{-n}\,\mathrm{d}x = \left[\frac{1}{1-n} \cdot x^{-(n-1)}\right]_{x_{\min}}^{x_{\max}}$$

$$= \frac{N}{1-n}\left(x_{\max}^{1-n} - x_{\min}^{1-n}\right) = 1$$ ✓

Dann zur Berechnung der Funktion, die implimentiert wird:

$$\begin{aligned}
\int_{x_{\min}}^{y} N x^{-n}\,\mathrm{d}x &= N\left[\frac{1}{1-n}x^{-(n-1)}\right]_{x_{\min}}^{y} \\
&= \frac{N}{1-n}\left(y^{1-n} - x_{\min}^{1-n}\right) \\
&= \frac{1}{x_{\max}^{1-n} - x_{\min}^{1-n}}\left(y^{1-n} - a^{1-n}\right) = g(y) \\
\implies y^{1-n} &= \left(x_{\max}^{1-n} - x_{\min}^{1-n}\right)\cdot g(y) + x_{\min}^{1-n} \\
\implies y &= \left(\left(x_{\max}^{1-n} - x_{\min}^{1-n}\right)\cdot g(y) + x_{\min}^{1-n}\right)^{\frac{1}{1-n}}
\end{aligned}$$

c) Hier ist kein N zu berechnen, daher wird direkt die Funktion, die importiert wird, bestimmt:

$$\begin{aligned}
\int_{-\infty}^{y} \frac{1}{\pi}\frac{1}{1+x^2}\,\mathrm{d} &= \frac{1}{\pi}\left[\tan^{-1}(x)\right]_{-\infty}^{y} \\
&= \frac{1}{\pi}\left(\tan^{-1}(y) + \frac{\pi}{2}\right) = g(y) \\
\implies \pi g - \frac{\pi}{2} &= \tan^{-1}(y) \\
\implies y &= \tan\left(\pi g - \frac{\pi}{2}\right)
\end{aligned}$$

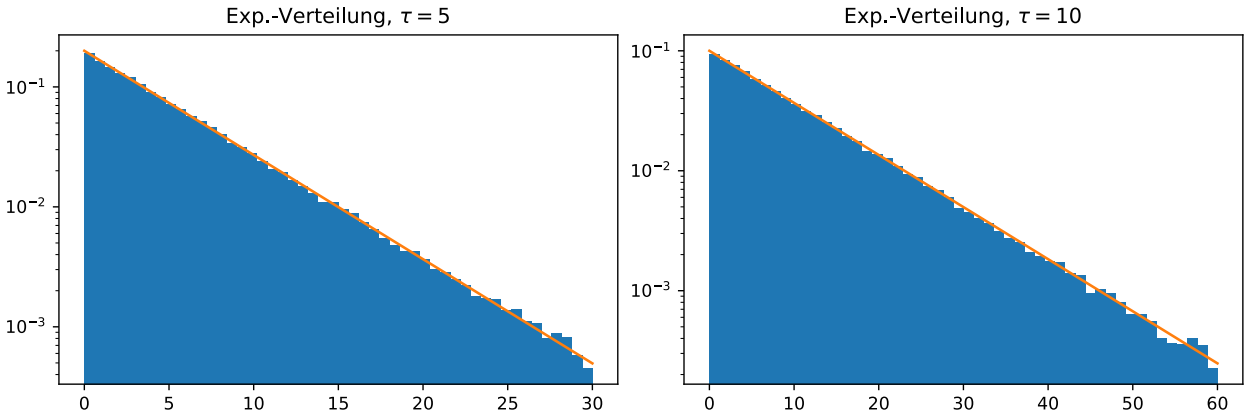
Die bestimmten Funktionen sind dann im Code implimentiert worden unter den entsprechenden Methoden.

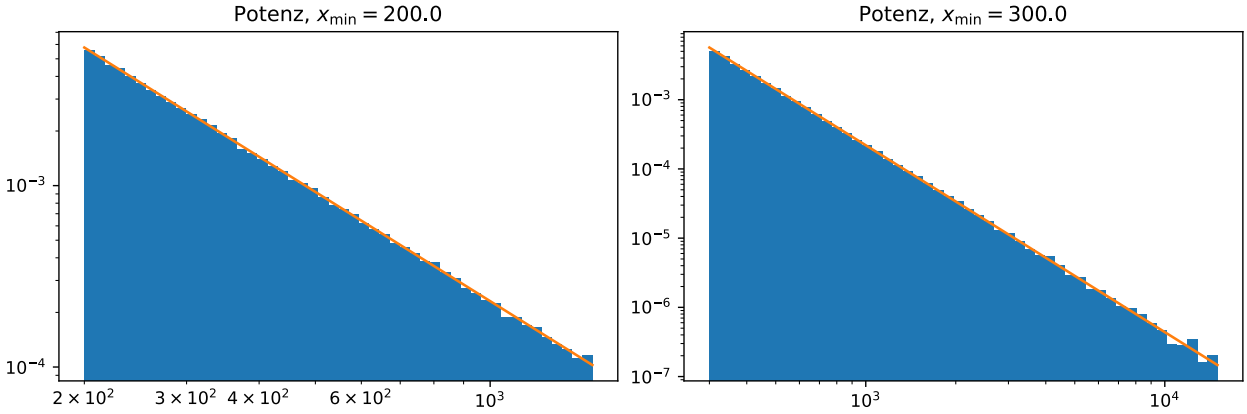
[11]:
```
path_to_file = "/home/amelie/Documents/SMD_A/project_c1/project_c1/random.py"

lines = Path(path_to_file).read_text().splitlines()
Code("\n".join(lines[72:127]), language="python")
```

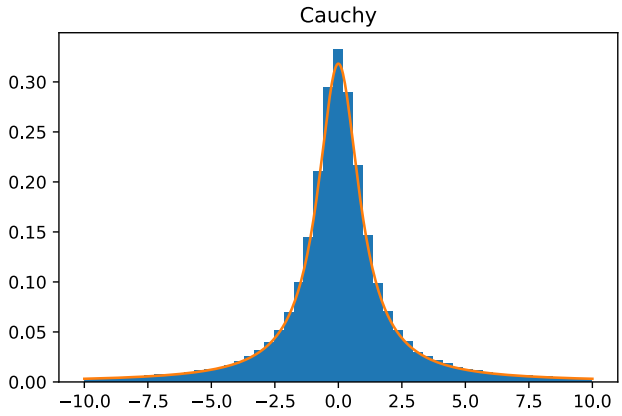
[11]:
```
class Generator(BaseGenerator):
    """
    Generate random numbers from different distribtions

    self.uniform(size=size) creates standard uniform random numbers.
    """

    def exponential(self, tau, size=None):
        """
        Draw exponentially distributed random numbers.

        Blatt 3, Aufgabe 2a)
        """
        # So können Sie ein array mit shape=size
        # mit standard gleichverteilten Zufallszahlen erzeugen
        u = self.uniform(size=size)

        # Fügen Sie hier den Code ein um Zufallszahlen aus der
```

```
        # angegebenen Verteilung zu erzeugen

        # dummy, so the code works. Can be removed / replaced
        values = -tau * np.log(1-u)

        return values

    def power(self, n, x_min, x_max, size=None):
        """
        Draw random numbers from a power law distribution
        with index n between x_min and x_max

        Blatt 3, Aufgabe 2b)
        """
        # Fügen Sie hier den Code ein um Zufallszahlen aus der
        # angegebenen Verteilung zu erzeugen
        u = self.uniform(size=size)
        N = np.power(x_max, 1-n) - np.power(x_min, 1-n)
        # dummy, so the code works. Can be removed / replaced
        values = (N * u + np.power(x_min, 1-n))**(1/(1-n))

        return values

    def cauchy(self, size=None):
        """
        Draw random numbers from a power law distribution
        with index n between x_min and x_max

        Blatt 3, Aufgabe 2c)
        """
        # Fügen Sie hier den Code ein um Zufallszahlen aus der
        # angegebenen Verteilung zu erzeugen
        u = self.uniform(size=size)
        # dummy, so the code works. Can be removed / replaced
        values = np.tan(np.pi * u - np.pi/2)

        return values
```

Unten ist dann noch die pdf zu sehen, die wir mithilfe des Testbefehls erzeugen sollten. Die Interpretation: Die Ergebnisse sehen soweit gut aus im Vergleich zu den vorgegebenen Linien, allerdings bilden sich größere Abweichungen bei hohen Potenzen (bzw. bei Cauchy in der 0).








```
Information:
  Exercise: Verteilungen
  Group name: project_c1
Runtime:
  exp:    4.2 ms (ref: 3.2 ms)
  power:  3.8 ms (ref: 2.6 ms)
  cauchy: 2.8 ms (ref: 3.1 ms)
Tests:
  exp mean correct: True
  power n correct:  True
```

top :)

5/5