9/20P.

SMD-Übungsblatt 2

Abgabe: 01.11.18

Yvonne Kasper yvonne.kasper@udo.edu ,
Robert Appel robert.appel@udo.edu ,
Julian Schröer julian.schroeer@udo.edu

1 Aufgabe1

Wiederholung der Methoden aus der Vorlesung Gegeben:

$$f(u) = U(0,1) = \begin{cases} 1, & 0 \leq x < 1 \\ 0, & \text{sonst} \end{cases} \tag{1}$$

Hier bezeichnet u gleichverteilte Zufallsvariablen mit der Wahrscheinlichkeitsdichte $f(u)$. Gesucht:

$$g(y) \quad \text{mit} \quad y \in [y_{min}, y_{max}] \tag{2}$$

Dabei bezeichnet y eine Zufallsvariable mit der Wahrscheinlichkeitsdichte $g(y)$. Die Transformation einer Gleichverteilung, um Zufallszahlen aus einer beliebigen Verteilung zu generieren ist gegeben durch

$$g(y)\mathrm{d} = U(0,1)\mathrm{d}u \tag{3}$$

$$\implies u = \int_{u_{min}}^{u} U(0,1)\mathrm{d}u = G(y) = \int_{ymin}^{y} g(y')\mathrm{d}y' \tag{4}$$

$$\implies y = G^{-1}(u)\ . \tag{5}$$

Für den Programmierteil dieser Aufgabe wurden die zufälligen Werte wie folgt erstellt.

```
TheAnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything = 42
np.random.seed(TheAnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything)
u = np.random.uniform(0,1,100)
```

plots/Aufgabe1.py

1.1 a)

Mit den zuvor dargestellten Methoden ergibt sich die Funktion

$$y = u \cdot (b - a) + a\ . \quad (\checkmark) \tag{6}$$

Die dazugehörigen Rechnungen sind in der Abbildung 1a dargestellt. In der Abbildung 2 in der oberen linken Ecke findet sich ein Histogramm der Funktion (grün). Im selben Histogramm ist die gegebene Gleichverteilung die Werte in den Grenzen [0,1] ausgibt gezeigt. In Python wurde die Funktion wie folgt implementiert.

```
def gleichverteili(uniforms, xmin,xmax):
  return (u*(xmax - xmin) + xmin)
```

plots/Aufgabe1.py

Was ist mit euch?

1P.

1.2 b)

Mit den zuvor dargestellten Methoden ergibt sich die Funktion

$$y = -\tau \ln(1-u)\ . \tag{7}$$

Die dazugehörigen Rechnungen sind in der Abbildung 1a dargestellt. In der Abbildung 2 in der oberen rechten Ecke findet sich ein Histogramm der Funktion (blau).In Python wurde die Funktion wie folgt implementiert.

```
def expotentialverteili(uniform, tau):
    return (- tau * np.log(1-u))
```

plots/Aufgabe1.py

1P.

1.3 c)

Mit den zuvor dargestellten Methoden ergibt sich die Funktion

$$y = \left(u \cdot \left(b^{1-n} - a^{1-n}\right) + a^{1-n}\right)^{\frac{1}{1-n}}\ . \quad (\checkmark) \tag{8}$$

Die dazugehörigen Rechnungen sind in der Abbildung 1b dargestellt. In der Abbildung 2 in der unteren linken Ecke findet sich ein Histogramm der Funktion (blau).In Python wurde die Funktion wie folgt implementiert.

```
def potenzverteili(uniform,n,xmin,xmax):
    return ((uniform*(xmax**(1-n) - xmin**(1-n)) + xmin**(1-n))**(1/(1-n)))
```

plots/Aufgabe1.py

1P.

1.4 d)

Mit den zuvor dargestellten Methoden ergibt sich die Funktion

$$y = \tan(\pi(u+1)) \tag{9}$$

Die dazugehörigen Rechnungen sind in der Abbildung 1b dargestellt. In der Abbildung 2 in der unteren rechten Ecke findet sich ein Histogramm der Funktion (blau).In Python wurde die Funktion wie folgt implementiert.

```
def cauchyverteili(uniforms):
    return np.tan(np.pi*(uniforms + 1))
```

plots/Aufgabe1.py

0.5P.

1.5 e)

Eine Methode um aus diskreten Werten Zufallsvariablen zu generieren ist, zuerst eine kummulative Wahrscheinlichkeit für alle x_k mit $k = 1, ..., n$ zu berechnen, mit

$$P_{k+1} = \sum_{i=1}^{k} P(x_i) \quad \text{mit} \quad P_1 = 0,\ P_{n+1} = 1\ . \tag{10}$$

Das wurde von uns in Python wie folgt implimentiert.

a)

$g(y)$ sei eine Gleichverteilung in d. Grenzen $[y_{min}, y_{max}]$ hier der Einfachheit halber $[a,b]$

$$\Rightarrow \quad g(y) = \begin{cases} \frac{1}{b-a}, & a \le y < b \\ 0, & \text{sonst} \end{cases}$$

$$\Rightarrow \quad G(y) = \int_a^y \frac{1}{b-a}\, dy' = \frac{y-a}{b-a} \stackrel{!}{=} u$$

$$\Rightarrow \quad \frac{y-a}{b-a} = u \Leftrightarrow \boxed{y = u(b-a) + a} \quad ✓$$

b)

$$g(y) = \mathcal{N} e^{-y/\tau} \qquad y \in [0, \infty)$$

Zuerst Normierung:

$$\int_0^\infty \mathcal{N} e^{-y/\tau} = \mathcal{N}(-\tau)\, e^{-y/\tau}\Big|_0^\infty = -\mathcal{N}\tau[0-1]$$

$$= \mathcal{N}\tau \stackrel{!}{=} 1 \Leftrightarrow \mathcal{N} = \frac{1}{\tau} \quad ✓$$

Transformation nach d. Methoden:

$$u = G(y) = \int_0^y \mathcal{N} \cdot e^{-y'/\tau}\, dy' = -\mathcal{N}\tau\left(e^{-y/\tau} - 1\right) \quad ✓$$

Normierung einsetzen $\rightarrow \quad u = 1 - e^{-y/\tau}$

$$\Rightarrow \quad 1 - u = e^{-y/\tau} \Leftrightarrow \ln(1-u) = -y/\tau$$

$$\boxed{\Leftrightarrow -\tau \ln(1-u) = y} \quad ✓$$

(a) a) und b).

c)

$g(y) = \mathcal{N} y^{-n}$ mit $y \in [y_{min}, y_{max}]$, $n \ge 2$

Zuerst Normierung mit $y \in [y_{min}, y_{max}] = [a,b]$:

$$\int_a^b \mathcal{N} y^{-n}\, dy \stackrel{n \ge 2}{=} \mathcal{N}\left[\frac{y^{-n+1}}{1-n}\right]_a^b$$

$$= \mathcal{N}\left[\frac{b^{1-n}}{1-n} - \frac{a^{1-n}}{1-n}\right] \stackrel{!}{=} 1$$

$$\Leftrightarrow \mathcal{N} = \frac{1-n}{b^{1-n} - a^{1-n}} \quad ✓$$

Transformation nach d. Methoden:

$$G(y) = \int_a^y \mathcal{N} y'^{-n}\, dy' = \mathcal{N}\left[\frac{y'^{1-n}}{1-n}\right]_a^y$$

$$= \mathcal{N}\left(\frac{y^{1-n}}{1-n} - \frac{a^{1-n}}{1-n}\right) = \mathcal{N}\, \frac{y^{1-n} - a^{1-n}}{1-n} \quad \Big| \text{ Normierung einsetzen}$$

$$= \frac{y^{1-n} - a^{1-n}}{b^{1-n} - a^{-n+1}} = u \quad ✓$$

$$\Leftrightarrow \boxed{y = \left(u\left(b^{1-n} - a^{1-n}\right) + a^{1-n}\right)^{\frac{1}{1-n}}} \quad ✓$$

d)

$$g(y) = \frac{1}{\pi}\, \frac{1}{1+y^2} \qquad y \in (-\infty, \infty)$$

$$\Rightarrow G(y) = \int_{-\infty}^y \frac{1}{\pi}\, \frac{1}{1+y'^2}\, dy' = \frac{1}{\pi} \arctan y \Big|_{-\infty}^y$$

$$= \frac{1}{\pi}\left[\arctan y + \frac{\pi}{2}\right] = \frac{1}{\pi} \arctan y + \frac{1}{2} = u$$

$$\Leftrightarrow \boxed{y = \tan(\pi(u+1))}$$

$y = \tan(\pi \cdot (u - 1/2))$ f.

(b) c) und d).

Abbildung 1: Rechnungen zu Aufgabe1

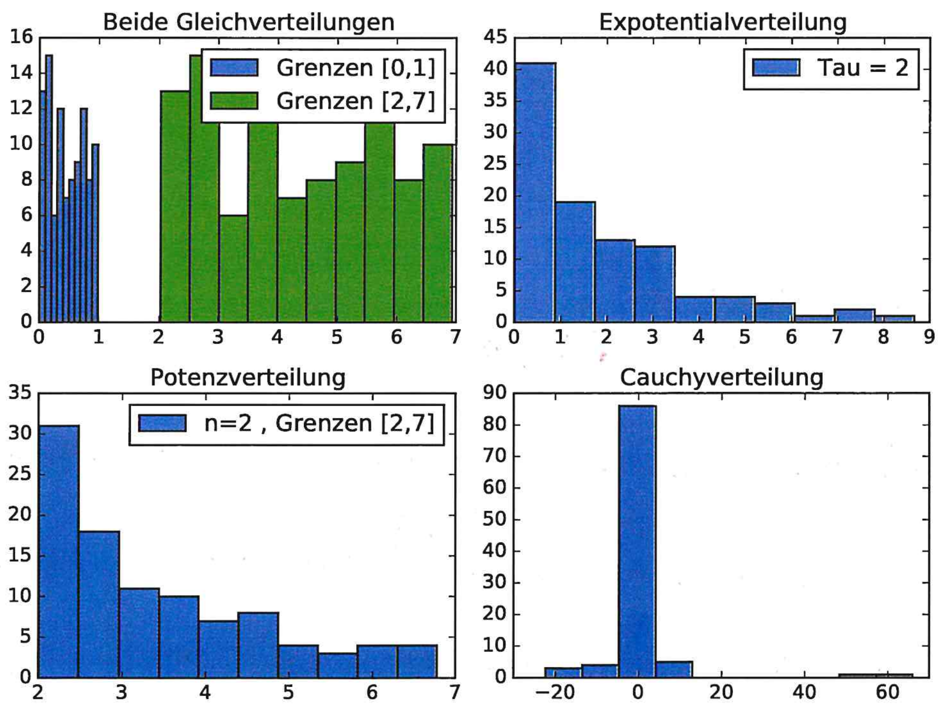

Abbildung 2: Diagramme zu den einzelnen Verteilungen.

```python
binmid, counts = np.genfromtxt('empirisches_histogramm.csv',delimiter=',' ,skip_header
    =1, unpack = True)
wahrschs = counts/sum(counts)
kumuwahr = np.array([])
for w in wahrschs:
  if w == wahrschs[0]:
    kumuwahr = np.append(kumuwahr,w)
  else:
    kumuwahr = np.append(kumuwahr,w + kumuwahr[-1])
```

ihr meint: beim ersten ~~Wert~~ Eintrag ...
ihr sagt: bei allen ~~Werten~~, Einträgen, die den Wert w[0] haben
(trifft auf 2 Bins zu!)

plots/Aufgabe1.py

Hier stellt *wahrsch* die Wahrscheinlichkeit berechnet aus den Counts da und *kumuwahr* die kummulative Wahrscheinlichkeit. Die for-Schleife führt eigentlich nur die Summe aus. Um nun die diskreten Zufallsvariablen zu bekommen nimmt man die gleichverteilten Zufallswerte u und vergleicht diese mit den Elementen $P_{k-1} < u < P_k$ so erhält man den Index k für den die Ungleichung erfüllt ist. Der Index ist dann auch der Index der diskreten Zufallsvariable anhand dem man die Zufallsvarable auslesen kann. In dem Fall hier ist es der Index eines *binmid* Wertes. Die Inplementierung in Python folgt.

```python
def diskretverteili(uniforms):
  vals = np.array([])
  for uni in uniforms:
    if uni > max(kumuwahr):
      continue
    else:
      vals = np.append(vals, binmid[kumuwahr == kumuwahr[kumuwahr >= uni][0]])
  return vals
```

notwendig weil max(kumuwahr) ≠ 1 ist, weil euch eine Addition wegen oben flöten geht (Im 3. Bin resetet er quasi)

plots/Aufgabe1.py

Das Historgamm dazu ist in der Abbildung 3 dargestellt.

0.5 P.

4P.

~~k = [ ]~~
~~k = wahrschs[0]~~
~~for w in wahrschs[1:]:~~

```
kumuwahr = np.array([ ])
kumuwahr = np.append(kumuwahr, wahrschs[0])
for w in wahrschs[1:]:
   kumuwahr = np.append(kumuwahr, w+kumuwahr[-1])
```

und statt x = np.append(x,y) geht auch
x.append(y)

PROBLEM:
Ihr zieht so nur diskrete Werte (entsprechend den Binmids), wir wollen aber kontinuierliche Zufallszahlen haben.

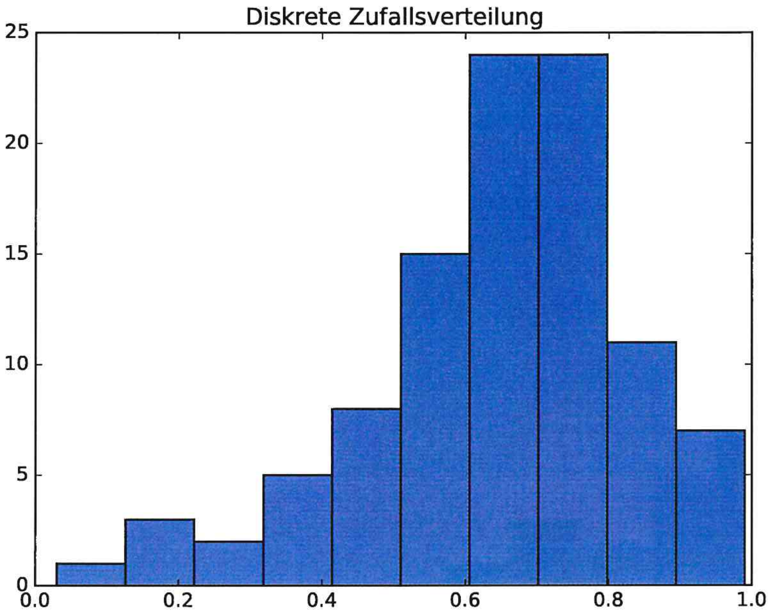

Abbildung 3: Histogramm zu den diskreten Zufallsvariablen.

2 Aufgabe 2

2.1 Teil a)

Zufallszahlgeneratoren wiederholen sich ab einer gewissen Anzahl an Rechenoperationen. Die Periodenlänge nennen wir P.
Für diesen Aufgabenteil wird ein linear-kongruenter Zufallszahlgenerator mit der Vorschrift

$$x_n = (a \cdot x_{n-1} + 3)\%1024$$

verwendet. Der Seed x_0 wurde für Aufgabenteil a) gleich Null gewählt. Dabei wurde der Multiplikator a auf einem Bereich von $0 - 80$ untersucht. In der Abbildung 4 ist die Periodenlänge in Abhängigkeit des Multiplikators a dargestellt.

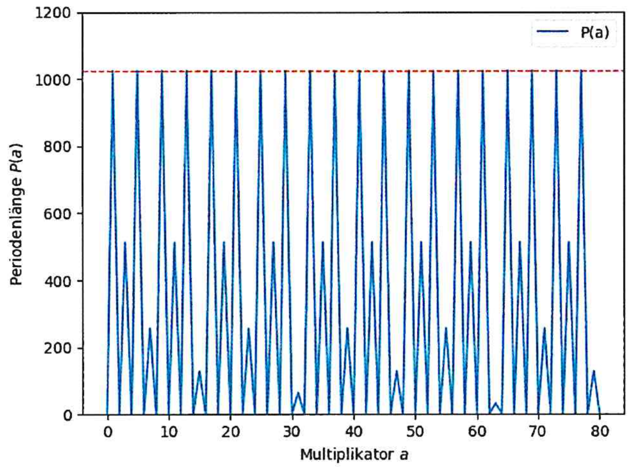

Abbildung 4: Darstellung der Periodenlänge in Abhängigkeit des Multiplikators.

Die theoretisch maximale Periodenlänge entspricht dem Modulus, hier $m = 1024$. Diese Periodenlänge wird auch erreicht.
In Abbildung 4 ist die maximale Periodenlänge eingezeichnet. Die Werte für a bei denen P maximal ist, sind:

$$1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77$$

Um eine maximale Periodenlänge zu erreichen müssen folgende Punkte gelten:

- $b \neq 0$
- b und m sind teilerfremd
- jeder Primfaktor von m teilt $(a - 1)$
- wenn m durch 4 teilbar ist, ist es auch $(a - 1)$

Bei der hier verwendeten Wahl von b und m sind die ersten beiden Punkte erfüllt. Um den nächsen Punkt zu überprüfen, wird eine Primfaktorzerlegung von m gesucht. Da

$$m = 1024 = 2^{10}$$

eine Zweierpotenz ist und nach dem Fundamentalsatz der Arithmetik die Eindeutigkeit der Primfaktorzerlegung sichergestellt ist, ist 2 der einzige Primfaktor. Alle Werte für a, bei den die Periodenlänge maximal ist, sind ungerade, das heißt, dass $(a - 1)$ durch 2 teilbar ist. Damit ist auch der dritte Punkt erfüllt. Für den vierten Punkt: $1024/4 = 256$ und für $(a - 1)$ gilt:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19$$

Null ist durch 4 teilbar aber wenn man sich die „Zufallszahlen" von a=1 anschaut, scheint es kein guter Generator zu sein

und damit bis auf $(a - 1) = 0$ auch durch 4 teilbar.
Also können die Ergebnisse mit den Regeln für *gute* linear-kongruente Generatoren erklärt werden.

1P.

2.2 Teil b)

Es wurden 10000 Zufallszahlen nach der Vorschrift

$$x_n = (1601 \cdot x_{n-1} + 3456)\%10000$$

erzeugt.
Die Werte sind in Abbildung 5 in einem Histogramm dargestellt.

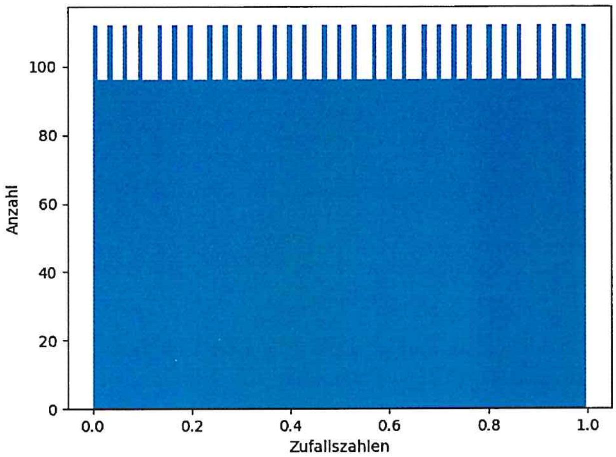

Abbildung 5: Histogramm der gezogenen Zufallszahlen.

Das Ergebnis zeigt, dass der linear-kongruente Generator keine wirkliche Gleichverteilung erzeugt, was ein *guter* Zufallszahlengenerator tun sollte. Die Höhe der einzelnen Spitzen ist dabei (minimal) vom Seed abhängig, allerdings wird nie eine echte Gleichverteilung entstehen. ✓ 1P.

2.3 Teil c)

In den folgenden Abbildungen sind Paare und Tripletts nachfolgender Zahlen dargestellt.

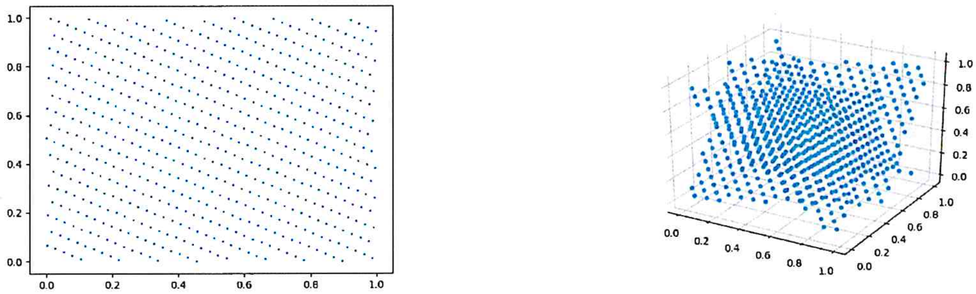

Abbildung 6: Scatterplots der jeweils aufeinander folgenden Paaren und Tripletts für den linear-kongruenten Generator.

Es ist zu erkennen, dass die Punkte in beiden Scatterplots eindeutige Muster erzeugen. In dem 2D-Plot zeigen sich Geraden, was auch nicht verwundet, da die Vorschrift im Endeffekt eine Geradengleichung beschreibt, mit einem Versatz durch die Modulo Operation. Dies wird Marsaglia Effekt genannt. Dies sollte ein *guter* Zufallszahlengenerator nicht zeigen.

1P.

2.4 Teil d)

Analog zum Teil c) wurden die Plots für den **numpy.random.unform()**-Generator gemacht.

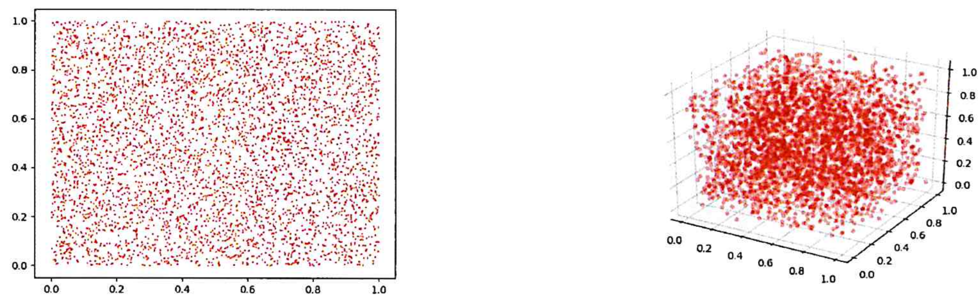

Abbildung 7: Scatterplots der jeweils aufeinander folgenden Paaren und Tripletts für den **numpy.random.unform()**-Generator.

Die Plots zeigen keine eindeutig erkennbare Muster zu erkennen, was darauf schließen lässt, dieser Zufallszahlgenerator nicht auf dem Prinzp des linear kongruenten Generator beruht. ✓ 1P.

auf jeden Fall nicht direkt

2.5 Teil e)

Ob der Generator aus Aufgabenteil a) den Wert $\frac{1}{2}$ erzeugen kann hängt sowohl vom Wert des Seeds x_0 als auch von dem es Multiplikators a ab. Um eine Kombination zu finden, welche den Wert $\frac{1}{2}$ erzeugen kann, wird die Ausgangsgleichung umgeformt. Wenn $x_1 = \frac{1}{2}$ sein soll, gilt:

$$\begin{aligned}
\frac{1}{2} &= (a \cdot x_0 + 3)\%1024 \\
512 &= (a \cdot x_0 + 3) \\
509 &= a \cdot x_0 \\
\frac{509}{a} &= x_0
\end{aligned}$$

Ist nun der Wert von $a = 1$ ist ein Startwert, der den Wert $\frac{1}{2}$ erzeugen kann, $x_0 = 509$. Dies ist eine Moglichkeit den Wert $\frac{1}{2}$ hat. ✓

1P.

5P.

Code fuer Blatt02

Kasper, Appel, Schroeer

2. November 2018

../B/1/Blatt02_Kasper_Appel_Schroeer/Blatt2_Kasper_Appel_Schroeer.py

```
def Aufgabe1():
    ### Some Sweet Random Data ###
    TheAnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything = 42
    np.random.seed(TheAnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything)
    u = np.random.uniform(0,1,100)
    ### Aufgaben a bis d - Fkt implimentierungen ###
    def gleichverteili(uniforms, xmin,xmax):
        return (u*(xmax - xmin) + xmin)

    def expotentialverteili(uniform, tau):
        return (- tau * np.log(1-u))

    def potenzverteili(uniform,n,xmin,xmax):
        return ((uniform*(xmax**(1-n) - xmin**(1-n)) + xmin**(1-n))**(1/(1-n)))

    def cauchyverteili(uniforms):
        return np.tan(np.pi*(uniforms + 1))
    ### Aufgabenteil e ###
    binmid, counts = np.genfromtxt('empirisches_histogramm.csv',delimiter=',' ,skip_header =1,
    unpack = True)
    wahrschs = counts/sum(counts)
    kumuwahr = np.array([])
    for w in wahrschs:
        if w == wahrschs[0]:
            kumuwahr = np.append(kumuwahr,w)
        else:
            kumuwahr = np.append(kumuwahr,w + kumuwahr[-1])
    def diskretverteili(uniforms):
        vals = np.array([])
        for uni in uniforms:
            if uni > max(kumuwahr):
                continue
            else:
                vals = np.append(vals,binmid[kumuwahr == kumuwahr[kumuwahr >= uni][0]])
        return vals
    plt.title('Diskrete Zufallsverteilung')
    plt.hist(diskretverteili(u))
    #plt.show()
    plt.savefig('A1eplot.pdf')
    plt.clf()


    ### Plots a bis d ###
    plt.subplot(2, 2, 1)
    plt.title('Beide Gleichverteilungen')
    plt.hist(u, label="Grenzen [0,1]")
    plt.hist(gleichverteili(u,2,7), label= "Grenzen [2,7]")
    plt.legend(loc='best')

    plt.subplot(2, 2, 2)
    plt.title('Expotentialverteilung')
    plt.hist(expotentialverteili(u, 2),label="Tau = 2")
    plt.legend(loc='best')

    plt.subplot(2, 2, 3)
    plt.title('Potenzverteilung')
    plt.hist(potenzverteili(u, 2, 2,7),label="n=2 , Grenzen [2,7]")
```

```python
plt.legend(loc='best')

plt.subplot(2, 2, 4)
plt.title('Cauchyverteilung')
plt.hist(cauchyverteili(u))

plt.tight_layout(pad=0, h_pad=1.08, w_pad=1.08)
#plt.show()
plt.savefig('A1abcd.pdf')
plt.clf()

def Aufgabe2():

    # Für a) und e)
    # Maximale Periodenlaenge ist m, durch den Modulo Operator,
    # suche nach dem wiederauftauchen des seeds x, da dann die Folge von vorne
    # beginnt. Da nur die Periodenlänge interessiert, werden die erzeugten
    # Zufallszahlen nicht durch m geteilt um eine Verteilung zwischen 0 und 1 zu
    # erhalten.
    # n ist der Bereich auf dem P untersucht werden soll und x der Seed
    def Periodenlaenge(n, x):
        Periodenlaenge = np.empty(n)
        for a in np.arange(0, n):
            Werte = np.empty(1025)
            Werte[0] = x  # Seed festlegen
            flag = False  # bool zur überprüfung ob Seed schon wieder da war
            for i in np.arange(1, 1025):  # über 1024 weil das die max. PL ist
                Werte[i] = ((a*(Werte[i-1])+3) % 1024)
                if((Werte[i] == x) and (flag is False)):
                    Periodenlaenge[a] = i  # Index d ersten Wiederholung des Seeds
                    flag = True
        return(Periodenlaenge)


    # Erzeugt n Zufallszahlen mit einem linear kongruenten Zufallszahlengenerator,
    # gibt ein n-komponentiges Array zurück. Dabei ist der Seed x und a,b,m fest
    def LinKongruent(x, n):
        Werte = np.empty(n)
        Werte[0] = x
        for i in np.arange(1, n):  # geht die Indices von 1 bis n-1 als integer ab
            Werte[i] = ((1601*(Werte[i-1])+3456) % 10000)
        return(Werte/10000)


    # Aufgabenteil a)

    n = 81
    Wertebereich = np.arange(0, n)
    Periodenlaenge = Periodenlaenge(n, 0)
    print('Die Periodenlänge ist maximal (also P(a)=m=1024) bei a=')
    print(Wertebereich[Periodenlaenge == 1024])
    plt.plot(Wertebereich, Periodenlaenge, label='P(a)')
    plt.ylim(0, 1200)
    plt.xlabel(r'Multiplikator $a$')
    plt.ylabel(r'Periodenlänge $P(a)$')
    plt.axhline(y=1024, linewidth=1, linestyle='--', color='r')
    plt.legend(loc='best')
    plt.savefig('Periodenlaenge.png')
    plt.clf()

    # Aufgabenteil b)

    n = 10000
    Zufallszahlen = LinKongruent(1, 10000)
    plt.hist(Zufallszahlen, bins=99)
    plt.xlabel(r'Zufallszahlen')
    plt.ylabel(r'Anzahl')
    plt.savefig('Zufallszahlen2b.png')
    plt.clf()

    # Aufgabenteil c)
```

```
    Zufallszahlen2 = Zufallszahlen.reshape(5000, 2)
    x = Zufallszahlen2[:, 0]
    y = Zufallszahlen2[:, 1]
    plt.scatter(x, y, s=0.3)
    plt.savefig('Paare2c.png')
    plt.clf()
    Zufallszahlen3 = LinKongruent(1, 9999)
    Zufallszahlen3 = Zufallszahlen3.reshape(3333, 3)
    x = Zufallszahlen3[:, 0]
    y = Zufallszahlen3[:, 1]
    z = Zufallszahlen3[:, 2]

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # ax.init_view(45, 30) # funktioniert leider nicht
    ax.scatter(x, y, z, lw=0, alpha=0.3)
    plt.savefig('Triplets2c.png')
    plt.clf()

    # Aufgabenteil d)
    ZZ = np.random.uniform(0, 1, 10000)
    ZZ2 = ZZ.reshape(5000, 2)
    x = ZZ2[:, 0]
    y = ZZ2[:, 1]
    plt.scatter(x, y, s=0.3, c='r')
    plt.savefig('Paare2d.png')
    plt.clf()

    ZZ3 = np.random.uniform(0, 1, 9999)
    ZZ3 = ZZ3.reshape(3333, 3)
    x = ZZ3[:, 0]
    y = ZZ3[:, 1]
    z = ZZ3[:, 2]

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(x, y, z, lw=0, alpha=0.3, c='r')
    plt.savefig('Triplets2d.png')
    plt.clf()


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import numpy as np
    from mpl_toolkits.mplot3d import Axes3D
    import scipy.constants as const
    from uncertainties import ufloat

    Aufgabe1()
    Aufgabe2()
```