aufgabe1

November 1, 2018

1 Aufgabe 1)

```
In [1]: import numpy as np
        from numpy import random
        import matplotlib.pyplot as plt
        from scipy import integrate
```

1.1 Teilaufgabe a)

Zu erzeugen sind gleichverteilte Zufallszahlen im Intervall von $x_{\min}$ bis $x_{\max}$. Dazu darf ein Zufallszahlengenerator verwendet werden, der gleichverteilte Zufallszahlen zwischen 0 und 1 erzeugt. Es wird eine lineare Transformation durchgeführt, bei der die 0 auf $x_{\min}$ und 1 auf $x_{\max}$ abgebildet wird. Es ist ersichtlich, dass die Gleichverteilungseigenschaft erhalten bleibt. Es ergibt sich die Transformation

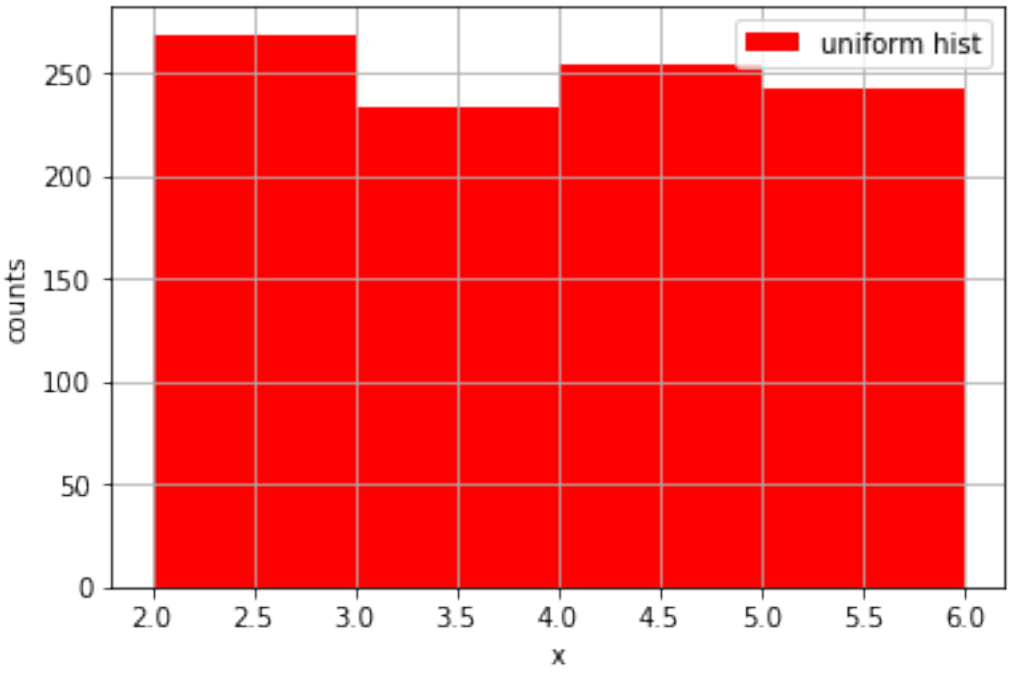
$$f(x) = (x_{\max} - x_{\min})x + x_{\min}\,, \tag{1}$$

wobei x aus den ursprünglichen Zahlen stammt und $f(x)$ im neuen Intervall gleichverteilt ist. Der Generator ist im Folgenden implementiert. Das Histogrammieren zeigt, da sich in Näherung der Anzahl an transformierten Samples eine Gleichverteilung ergibt.

```
In [2]: # Definiere einige Funktionen auch schon für spaeter
        def uniformTransform(x, x_min, x_max):
            return (x_max - x_min) * x + x_min

        func = lambda x: x**(-n)
        def powerLawPdf(x, x_min, x_max):
            return  1/integrate.quad(func, x_min, x_max)[0]* x**(-n)


        def powerLawCdf(x, x_min, x_max):
            return (x**(-n+1)-x_min**(-n+1))/(x_max**(-n+1)-x_min**(-n+1))


        def cauchyPdf(x):
            return 1/(1+x**2)*1/(np.pi)
```

```
In [3]: random.seed(42) # Setze einen Seed, um wiederholbare Ergebnisse zu erhalten
        uniformArrayBasic = random.uniform(size = 1000)
```

```
# Das ist die ursprüngliche gleichverteilte Sample, aus der wir alles generieren
# Die generierten Zahlen sind reell, gleichverteilt und zwischen 0 und 1
# Setze die Grenzen des neuen Intervalls
x_min = 2
x_max = 6
uniformArbitraryInterval = uniformTransform(uniformArrayBasic, x_min, x_max)
# Transformiere das Sample auf das neue Intervall
plt.hist(uniformArbitraryInterval,
         bins=np.arange(start=x_min, stop=x_max + 1, step=1),
         histtype='bar', label='uniform hist', color='r')
plt.grid()
plt.legend()
plt.xlabel('x')
plt.ylabel('counts')
plt.show()
```



1.2 Teilaufgabe b)

Wieder sind gleichverteilte Zufallszahlen im Intervall von 0 bis 1 gegeben. Diesmal soll daraus eine Verteilung mit der Wahrscheinlichkeitsdichtefunktion (PDF) $f(t) = N \exp(-t/\tau)$ im Bereich von 0 bis unendlich generiert werden. Die Normierungskonstante N berechnet sich hier zu $1/\tau$, da das besimmte Integral von f über den Definitionsbereich τ ergibt. Die kumulative Veteilungs-

funktion (CDF) berechnet sich zu

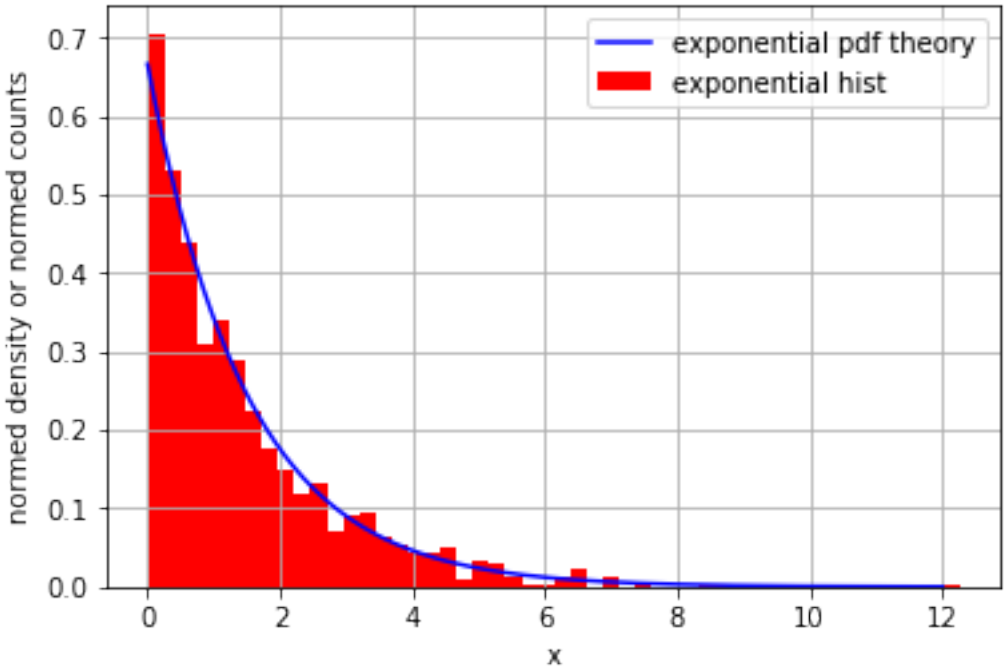
$$\begin{aligned} F(x) &= \int_{t_{\min}}^{t} \mathrm{d}t'\, f(t') \\ &= \frac{1}{\tau}\int_{0}^{t} \mathrm{d}t'\, \exp(-t/\tau) \\ &= \frac{1}{\tau}[-\tau \exp(-t/\tau)]_0^t = 1 - \exp(-t/\tau)\,. \end{aligned}$$

Wir wollen die exponentialverteilten Zufallszahlen mithilfe der Inversionsmethode berechnen, da diese effizient ist, weil keine Zufallszahlen verworfen werden müssen. Dazu muss die Inverse der CDF berechnet werden. Dafür wird $F(x) = y$ gesetzt und nach x umgestellt. Dann wird x umbenannt und mit der inversen CDF identifiziert. Für sie folgt

$$F^{-1}(t) = -\tau \ln|1 - t|\,. \tag{2}$$

Es folgt nun, dass die Anwendung von F^{-1} auf ein Sample gleichverteilter Zufallszahlen im Intervall von 0 bis 1 exponentialverteilte Zufallszahlen in den positiven reellen Zahlen liefert, da das Intervall mittransformiert wird. Die Implementation folgt. In das Histogramm ist auch die PDF eingefügt, sodass ersichtlicherweise exponentiell verteilte Zahlen erzeugt wurden.

```
In [4]: tau = 1.5
        x_lin = np.linspace(0, 8*tau, 1000)
        exponentialDistribution = - tau * np.log(np.abs(1 - uniformArrayBasic))
        plt.hist(exponentialDistribution, bins=50, normed=True,
                 histtype='bar', label='exponential hist', color='r')
        plt.plot(x_lin, np.exp(-x_lin/tau)/tau, 'b-', label='exponential pdf theory')

        #plt.plot(x_lin, 1-np.exp(-x_lin/tau), 'g-', label='exponential cdf theory')
        plt.grid()
        plt.legend()
        plt.xlabel('x')
        plt.ylabel('normed density or normed counts')
        plt.show()
```



1.3 Teilaufgabe c)

Nun wollen wir potenzverteilte Zufallszahlen im Bereich von $x_{\min}$ bis $x_{\max}$ generieren. Die Verteilung soll der PDF

$$f(x) = Nx^{-n} \tag{3}$$

folgen. Die Potenz n soll gröSSer als zwei sein. Die Normierungskonstante der Verteilungsfunktion ergibt sich zu

$$N = -\frac{1}{n+1}\frac{1}{x_{\max}^{1-n} - x_{\min}^{1-n}}\,. \tag{4}$$

Die CDF ist nach Integration von $x_{\min}$ bis x

$$F(x) = \frac{x^{1-n} - x_{\min}^{1-n}}{x_{\max}^{1-n} - x_{\min}^{1-n}}\,. \tag{5}$$

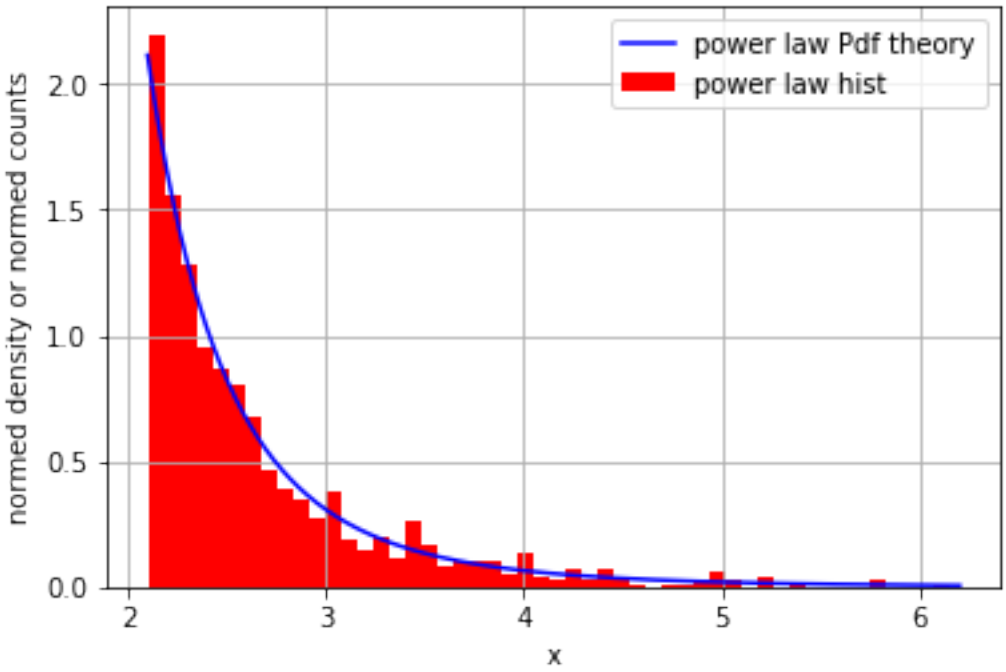
Diese Funktion kann invertiert werden, ihre Umkehrfunktion ist durch

$$F^{-1}(x) = \left((x_{\max}^{1-n} - x_{\min}^{1-n})x + x_{\min}^{1-n}\right)^{\frac{1}{1-n}} \tag{6}$$

gegeben. Das Anwenden dieser Funktion auf gleichverteilte Zufallszahlen aus dem Intervall von 0 bis 1 ergibt dann nach der Inversionsmethode potenzverteilte Zufallszahlen. Im generierten Histogramm ist ersichtlich, dass unsere Rechnung plausibel ist.

```
In [5]: plt.clf()
        n = 5.4
```

```
x_min = 2.1
x_max = 6.2
x_lin = np.linspace(x_min, x_max, 1000)
N = 1/integrate.quad(func, x_min, x_max)[0]
# We use numerical normalization to make our code easier to read
powerLawDistribution = ((1-n)/N*uniformArrayBasic + x_min**(-n+1))**(1/(-n+1))
plt.hist(powerLawDistribution, bins=50, normed=True,
         histtype='bar', label='power law hist', color='r')
plt.plot(x_lin, powerLawPdf(x_lin,x_min,x_max), 'b-',
         label='power law Pdf theory')
#plt.plot(x_lin, powerLawCdf(x_lin,x_min,x_max) ,
# 'g-', label='power law Cdf theory')
plt.grid()
plt.legend()
plt.xlabel('x')
plt.ylabel('normed density or normed counts')
plt.show()
```



1.4 Teilaufgabe d)

Wir wollen nun aus den gleichverteilten Zufallszahlen zwischen 0 und 1 cauchy-verteilte Zufallszahlen mit der Verteilungsfunktion

$$f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$$

generieren. Wieder nutzen wir die Inversionsmethode. Die CDF der Cauchy-Verteilung ist

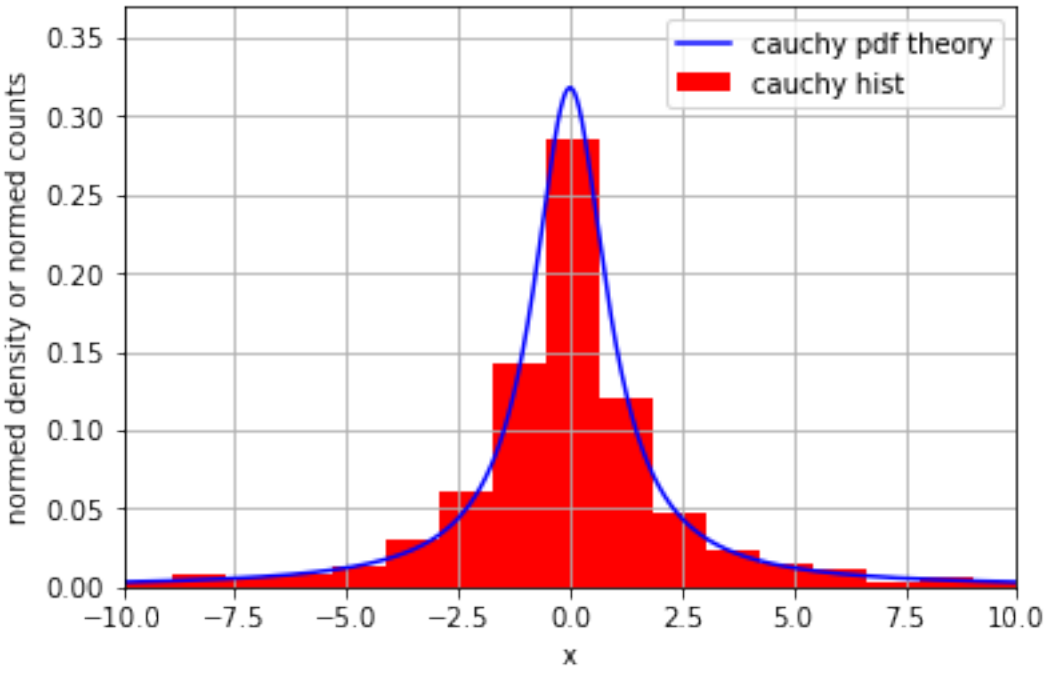
$$F(x) = \frac{1}{\pi}\arctan x + \frac{1}{2}.$$

Die Inverse der CDF berechnet sich zu

$$F^{-1}(x) = \tan(\pi(x - 1/2)).$$

Anwenden dieser Funktion auf die gegebenen Zufallszahlen liefert cauchy-verteilte Zufallszahlen. Dies wird im folgenden Code verdeutlicht.

```
In [6]: plt.clf()
        x_lin = np.linspace(-100, 100, 10000)
        cauchyDistribution = np.tan(np.pi*(uniformArrayBasic-1/2))
        plt.hist(cauchyDistribution, bins=1000, normed=True,
                 histtype='bar', label='cauchy hist', color='r')
        plt.plot(x_lin, cauchyPdf(x_lin), 'b-', label='cauchy pdf theory')
        #plt.plot(x_lin, powerLawCdf(x_lin,x_min,x_max) , 'g-',
        # label='power law Cdf theory')
        plt.grid()
        plt.legend()
        plt.xlabel('x')
        plt.ylabel('normed density or normed counts')
        plt.axis((-10,10,0,0.37))
        plt.show()
```



1.5 Teilaufgabe e)

Dieses Mal ist eine Stichprobe gegeben, die durch ein Histogramm parametrisiert ist. Es soll ein Zufallszahlengenerator geschrieben werden, der Zufallszahlen erzeugt, die so verteilt sind wie die der Stichprobe. Zuallererst ist der Generator als Funktion geschrieben. Die Inversionsmethode kann hier wenn überhaupt nur schlecht verwendet werden, da das Histogramm nur eine Annäherung an eine diskrete PDF ist und selbst wenn man aus dem Histogramm diese genäherte PDF ermitteln könnte, bleibt es unklar, ob diese integrierbar und ob die CDF invertierbar wäre. Deswegen basiert der Generator auf dem Rückweisungs- bzw. Verwerfungsverfahren von von Neumann. Dazu wird ein Zufallszahlenpaar (x,y) gezogen, wobei x auf $[0,1[$ stammt und y von 0 bis zur maximalen Anzahl an Counts im Histogramm reicht. Dann wird ermittelt, ob der Punkt in der ausgefüllten roten Fläche des Histogramms (also quasi unter der PDF) liegt oder nicht. Falls dies der Fall ist wird die Zufallszahl x akzeptiert. Die Implementierung in der Funktion ermittelt dies, indem sie sucht, wo x in den Bins einzuordnen ist und dann die Counts in diesem Bin mit y vergleicht.

In der Zelle darunter wird die Datei ausgelesen und gestückelt, um besser mit den Daten umgehen zu können. Danach wird ein Histogramm der gegebenen Daten gezeichnet und ausreichend viele Zufallszahlen generiert und ein Histogramm dieser gezeichnet, um einen Vergleich anstellen zu können. Wie zu erkennen ist, ähneln sich beide Verteilungen sehr, was auf eine funktionierende Generierung schlieSSen lässt. Zur Effizienz lässt sich sagen, dass hier auch viele Zufallszahlen verworfen werden müssen. Man hätte sich eventuell auch eine Hilfsdichte herleiten können, die das Histogramm eng einhüllt. Wir haben hier jedoch darauf verzichtet, da die Generation eines fast deckungsgleichen Histogramms nur wenige Sekunden dauert. Die Möglichkeit der Verbesserung unseres Generators durch so eine Hilfsdichte sei jedoch erwähnt.

```
In [7]: def empiricalRandomNumberGenerator(maxValue, ourBins, counts):
            x = random.uniform()
            y = random.uniform() * maxValue
            Bin = np.digitize(x, ourBins) # return the bin number of the x-value.
            #Starts at 1, so keep in mind we need to substract 1 later to index correctly
            countsAtBin = counts[Bin - 1]
            if y < countsAtBin:
                return x
            else:
                return None
```

```
In [8]: plt.clf()
        givenData = np.genfromtxt('empirisches_histogramm.csv', delimiter=',')
        givenData = np.delete(givenData,0,axis=0) # Delete the nans
        binmids, counts = zip(*givenData) # now we got 2 arrays with binminds and counts!
        ourBins = np.arange(start=0, stop=1.02, step=0.02)
        maxValue = np.amax(counts)
        generatedData = []
        for i in range(0,100000):
            generatedEmpiricalRandomNumber = empiricalRandomNumberGenerator(maxValue,
                                                                            ourBins,
                                                                            counts)
            if generatedEmpiricalRandomNumber != None:
                generatedData.append(generatedEmpiricalRandomNumber)
```

```
plt.subplot(1,2,1)
plt.hist(binmids, bins=ourBins, normed=True, weights = counts,
         histtype='bar', label='empirical hist', color='r')
plt.grid()
plt.legend()
plt.xlabel('x')
plt.ylabel('normed counts')
plt.subplot(1,2,2)
plt.hist(generatedData, bins=ourBins, normed=True,
         histtype='bar', label='generated hist', color='b')
plt.grid()
plt.legend()
plt.xlabel('x')
plt.ylabel('normed counts')
plt.tight_layout()
plt.show()
```