blatt02_nitschke_grisard

November 1, 2018

1 Blatt 2

1.1 Aufgabe 5

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

        uniform = random.uniform(size = 1000)
```

a) Gleichverteilung auf dem Gebiet $[x_{min}, x_{max}]$:

$$f(x) = \frac{1}{x_{max} - x_{min}}$$

Bilde Verteilungsfunktion $F(x)$:

$$F(x) = \int_{x_{min}}^{x} f(x')\,dx' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

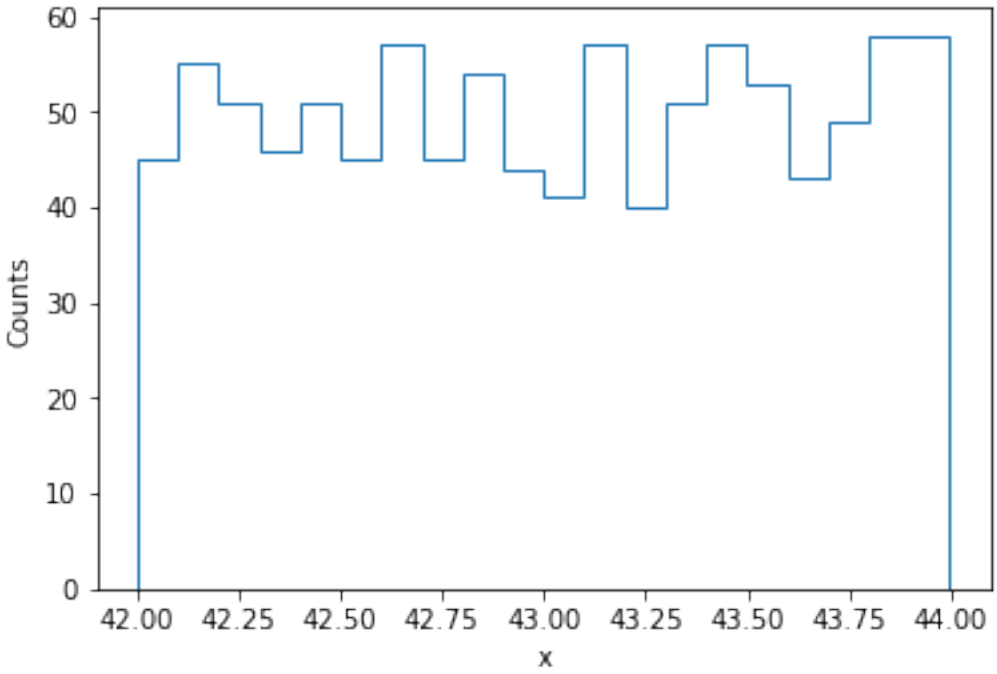
Bilde Umkehrfunktion $F^{-1}(y)$:

$$F^{-1}(y) = (x_{max} - x_{min})y + x_{min}.$$

Implementierung und Darstellung:

```
In [2]: def uniform_intervall(y, xmin, xmax):
            assert xmax >= xmin
            return xmin + y * (xmax - xmin)
```

```
In [3]: fig, ax = plt.subplots(1, 1)
        ax.hist(uniform_intervall(uniform, 42, 44), bins = 20, histtype='step')
        ax.set_xlabel('x')
        ax.set_ylabel('Counts')
        plt.show()
```



b) Exponentialgesetz:

$$f(t) = Ne^{-\frac{t}{\tau}} = \frac{1}{\tau}e^{-\frac{t}{\tau}}, \quad t \in [0, \infty)$$

Verteilungsfunkion:

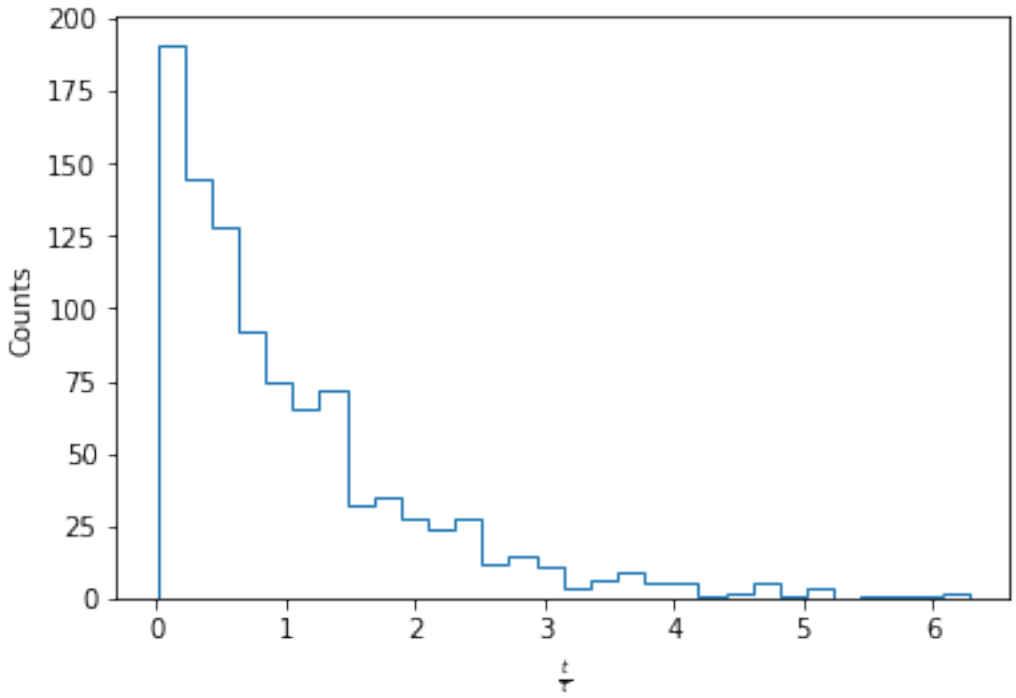
$$F(t) = \int_0^t f(t')\,dt' = (1 - e^{-\frac{t}{\tau}})$$

Umkehrfunktion:

$$F^{-1}(y) = \tau \ln\left(\frac{1}{1-y}\right)$$

Implementierung:

```
In [4]: def F(y, tau = 1):
            return tau * np.log(1 / (1 - y))

In [5]: fig, ax = plt.subplots(1, 1)
        ax.hist(F(uniform), bins = 30, histtype='step')
        ax.set_xlabel(r'$\frac{t}{\tau}$')
        ax.set_ylabel('Counts')
        plt.show()
```



c) Potenzgesetz:

$$f(x) = Nx^{-n} = \frac{1-n}{x_{max}^{1-n} - x_{min}^{1-n}} x^{-n}, \quad x \in [x_{min}, x_{max}], n \geq 2$$

Verteilungsfunktion:

$$F(x) = \int_{x_{min}}^{x} f(x')\, dx' = \frac{x^{1-n} - x_{min}^{1-n}}{x_{max}^{1-n} - x_{min}^{1-n}}$$

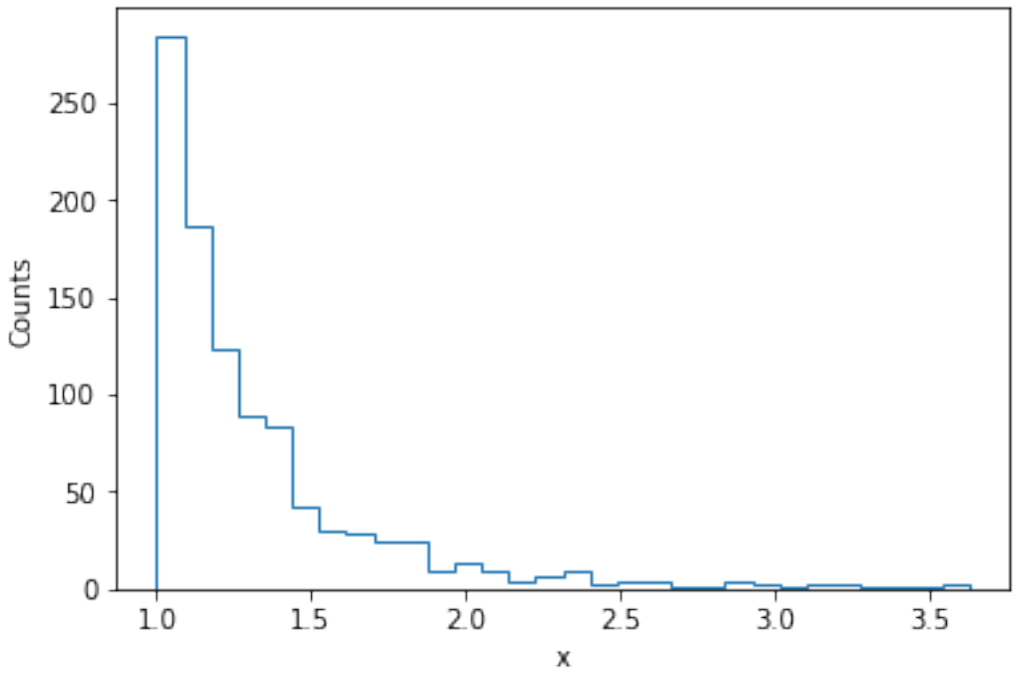
Umkehrfunktion:

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

Implementierung:

```
In [6]: def power(y, xmin, xmax, n):
            assert n >= 2
            return ( (xmax**(1-n) - xmin**(1-n)) * y +  xmin**(1-n) )**(1 / (1-n))
```

```
In [7]: fig, ax = plt.subplots(1, 1)
        ax.hist(power(uniform, xmin = 1, xmax = 4, n = 5), bins = 30, histtype='step')
        ax.set_xlabel('x')
        ax.set_ylabel('Counts')
        plt.show()
```



d) Cauchy-Verteilung:

$$f(x) = \frac{1}{\pi}\frac{1}{1+x^2}, \quad x \in (-\infty, \infty)$$

Verteilungsfunktion:

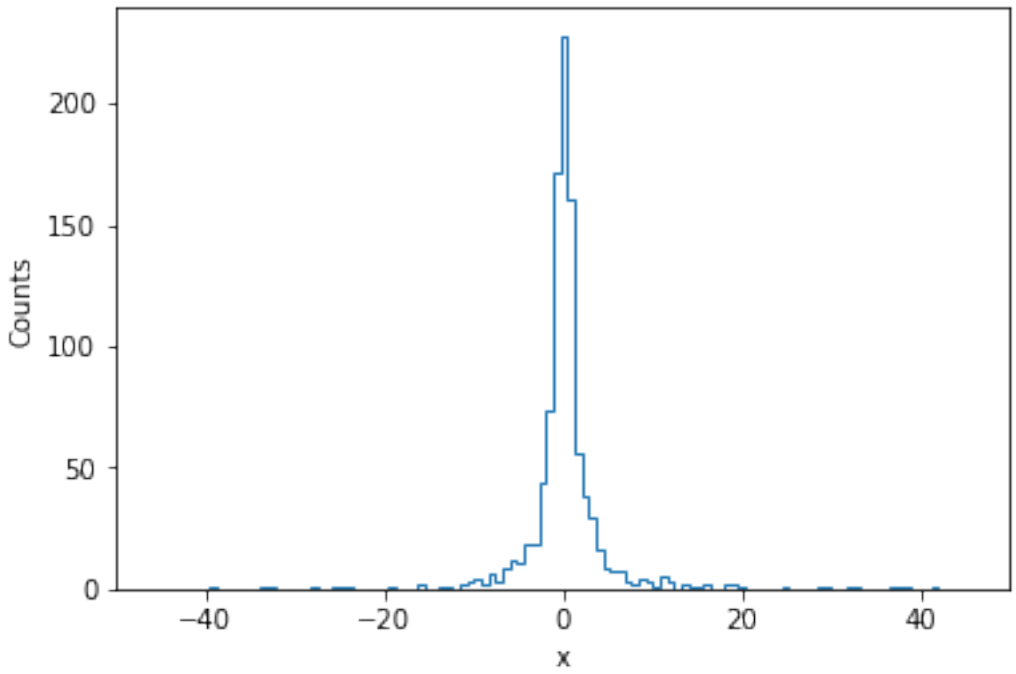
$$F(x) = \int_{-\infty}^{x} f(x')\,dx' = \frac{1}{\pi}\left[\arctan(x) + \frac{\pi}{2}\right]$$

Umkehrfunktion:

$$F^{-1}(y) = \tan\left(\pi y - \frac{\pi}{2}\right)$$

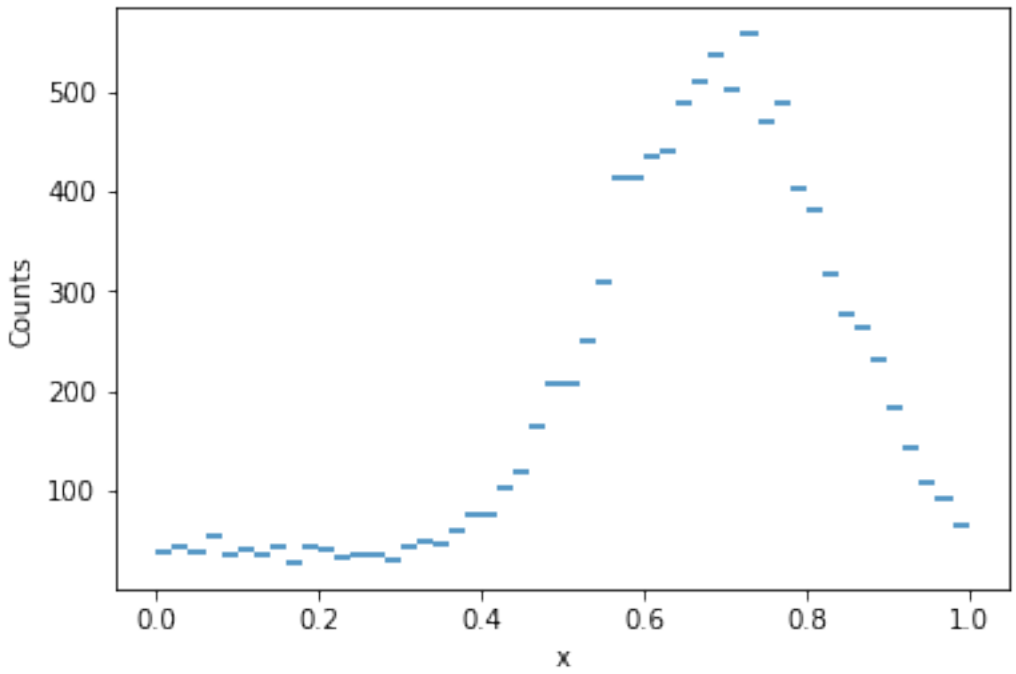
Implementierung:

```
In [8]: def cauchy(y):
            return np.tan(np.pi * (y - 1/2))

In [9]: fig, ax = plt.subplots(1, 1)
        ax.hist(cauchy(uniform), bins = 500, histtype='step')
        ax.set_xlim(-50, 50)
        ax.set_xlabel('x')
        ax.set_ylabel('Counts')
        plt.show()
```



e) Verteilung aus empirischem Histogramm mit der ’rejection sampling’ Methode

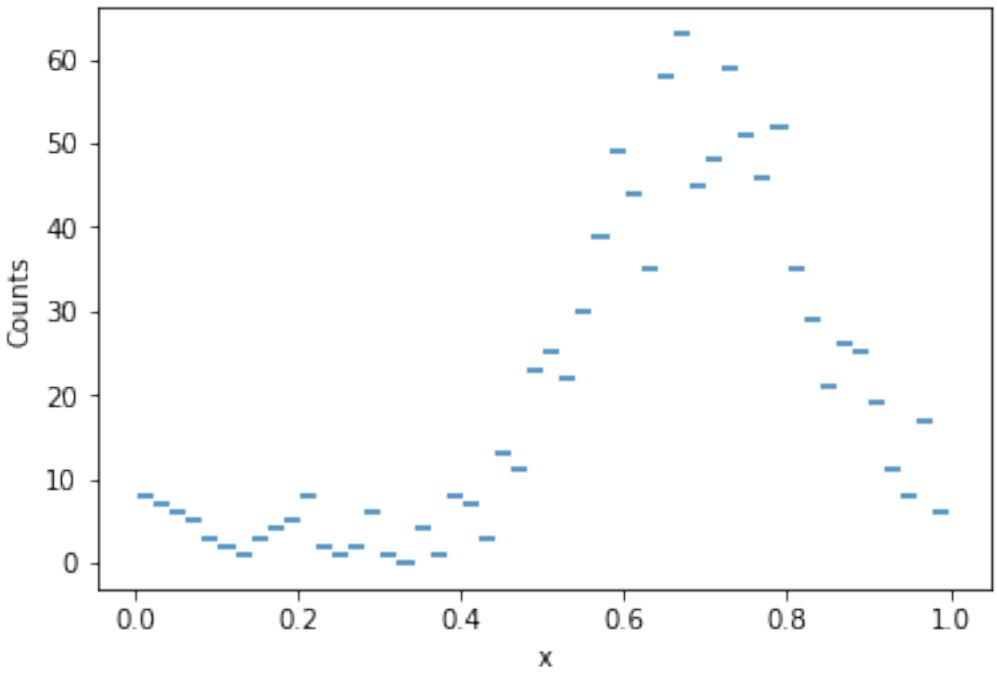
```
In [10]: import pandas as pd
         hist = pd.read_csv('empirisches_histogramm.csv')
         binmids = hist['binmid']
         counts = hist['counts']

         fig, ax = plt.subplots(1, 1)
         ax.errorbar(x = binmids, y = counts, xerr = 0.01, linestyle = '')
         ax.set_xlabel('x')
         ax.set_ylabel('Counts')
         plt.show()
```




```
In [11]: norm = np.sum(counts * 0.2) #bin width is 0.2
         norm_counts = counts / norm #normalize data
         u1 = random.uniform(size = 10000) #
         u2 = random.uniform(size = 10000) #two uniform samples

In [12]: def empirical(counts, binmids, u1, u2):
             binindex = u1 // 0.02 #get index of bin that includes u1 values
             return u1[ [ u2[i] <= counts[binindex[i]]
                         for i in range(len(u1)) ] ]
         y = empirical(norm_counts, binmids, u1, u2)
```

Von den erzeugten 20000 Zufallszahlen bleiben 4.905% übrig.

```
In [13]: fig, ax = plt.subplots(1, 1)
         counts, binedges = np.histogram(y, bins = 50)
         ax.errorbar(x = (binedges[:-1] + binedges[1:]) * 0.5, y = counts, xerr = np.diff(binec
         ax.set_xlabel('x')
         ax.set_ylabel('Counts')
         plt.show()
```

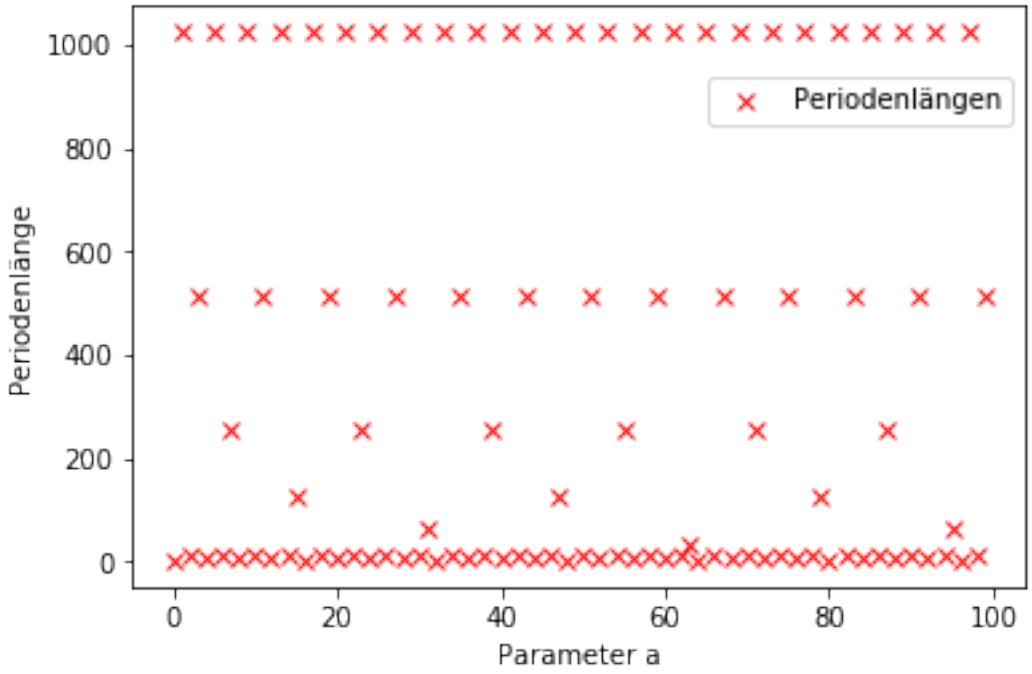


1.2 Aufgabe 6

```
In [14]: np.random.seed(0)
         def linGen(x_0, a, b, m):
             random_numbers = [x_0]
             for i in range(m):
                 next_number = (a * random_numbers[i] + b) % m
                 if (next_number in random_numbers):
                     return len(random_numbers), np.array(random_numbers)
                 else:
                     random_numbers.append(next_number)

In [15]: a = np.linspace(0,100,101)
         D = np.array([(linGen(x_0 = 4, a = n, b = 3, m = 1024))[0] for n in a])
         #d = np.array([linGen(n,3,1024,4)[1] for n in a])
         print("Die maximale Periodenlänge beträgt: ", np.max(D), '\n')
         print("Für folgende Werte von a ist die Periodenlänge maximal: ",'\n', a[D == np.max(D
```

```
Die maximale Periodenlänge beträgt:  1024

Für folgende Werte von a ist die Periodenlänge maximal:
 [ 1.  5.  9. 13. 17. 21. 25. 29. 33. 37. 41. 45. 49. 53. 57. 61. 65. 69.
 73. 77. 81. 85. 89. 93. 97.]
```

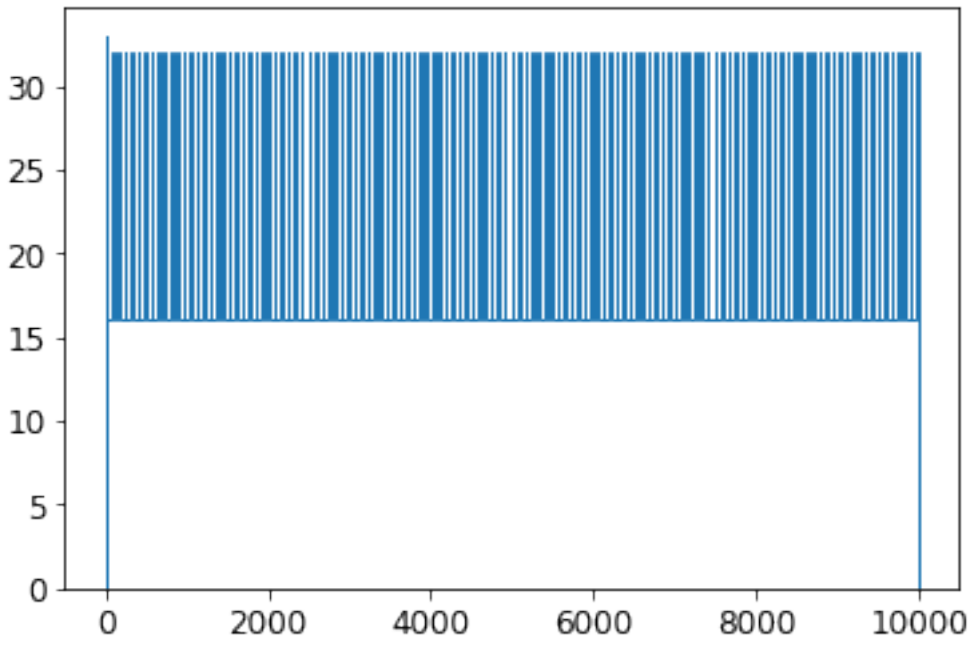
```
In [16]: plt.plot(a[:100], D[:100], 'rx', label = r'Periodenlängen');
         plt.xlabel('Parameter a')
         plt.ylabel('Periodenlänge')
         plt.legend(loc='upper right', bbox_to_anchor = (1,0.9))

Out[16]: <matplotlib.legend.Legend at 0x1c5d499ec18>
```

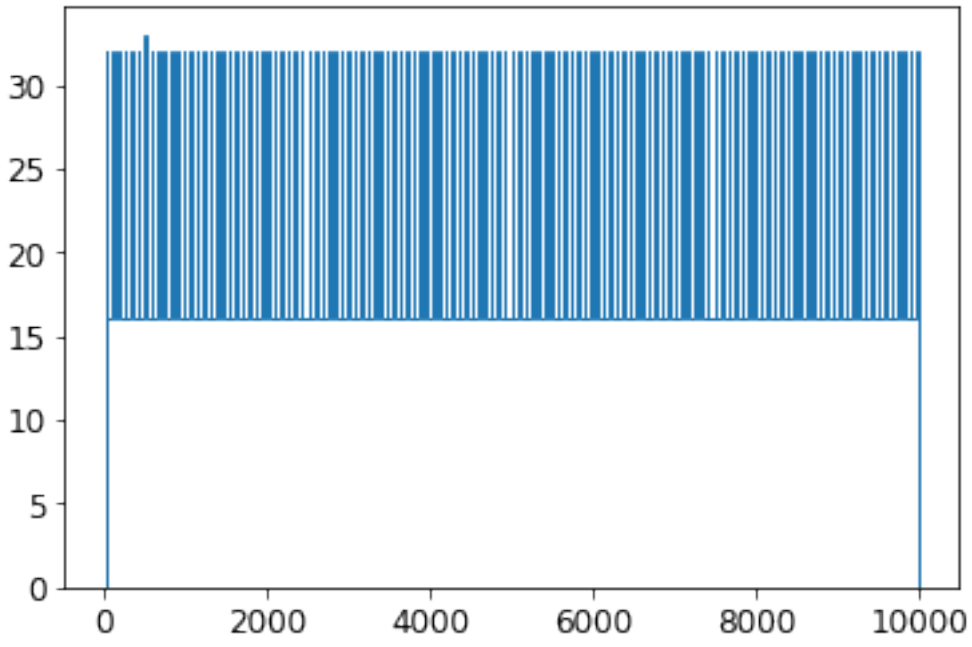


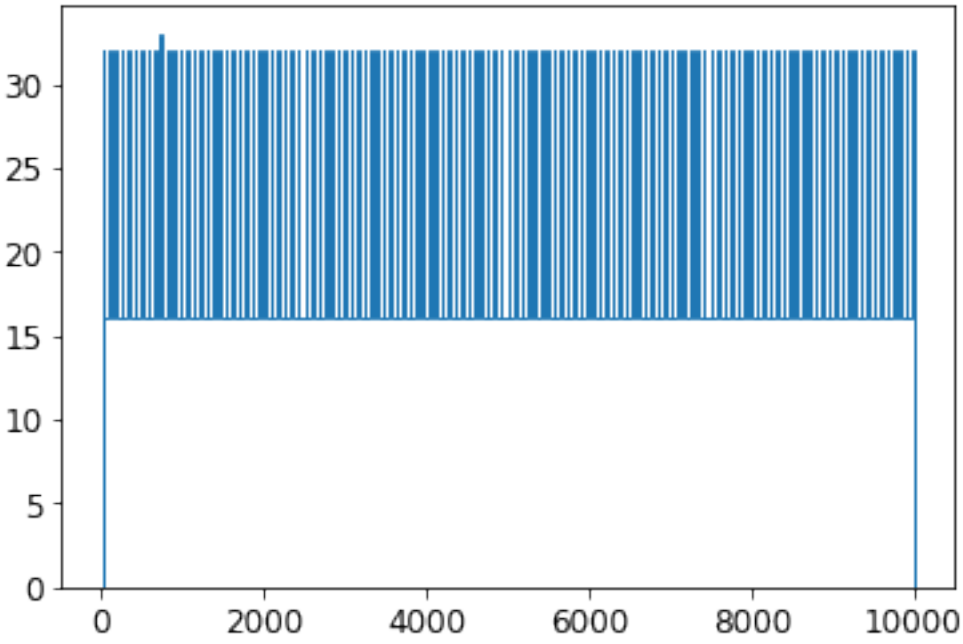
b)

```
In [17]: def linGen2(a, b, m, x_0, length):
             random_numbers = [x_0]
             for i in range(length):
                 random_numbers.append((a * random_numbers[i] + b) % m)
             return random_numbers

In [18]: Liste1 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 0,   length = 10000)
         Liste2 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 500, length = 10000)
         Liste3 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 750, length = 10000)

In [19]: %matplotlib inline

         plt.rcParams.update({'figure.figsize': (6, 4), 'font.size': 12})
         plt.hist(Liste1, bins = 500, range = [np.min(Liste1), np.max(Liste1)],  histtype = 'st
         plt.show()
         plt.hist(Liste2, bins = 500, range = [np.min(Liste2), np.max(Liste2)], histtype = 'ste
         plt.show()
```

```
plt.hist(Liste3, bins = 500, range = [np.min(Liste3), np.max(Liste3)], histtype = 'st
plt.show()
```



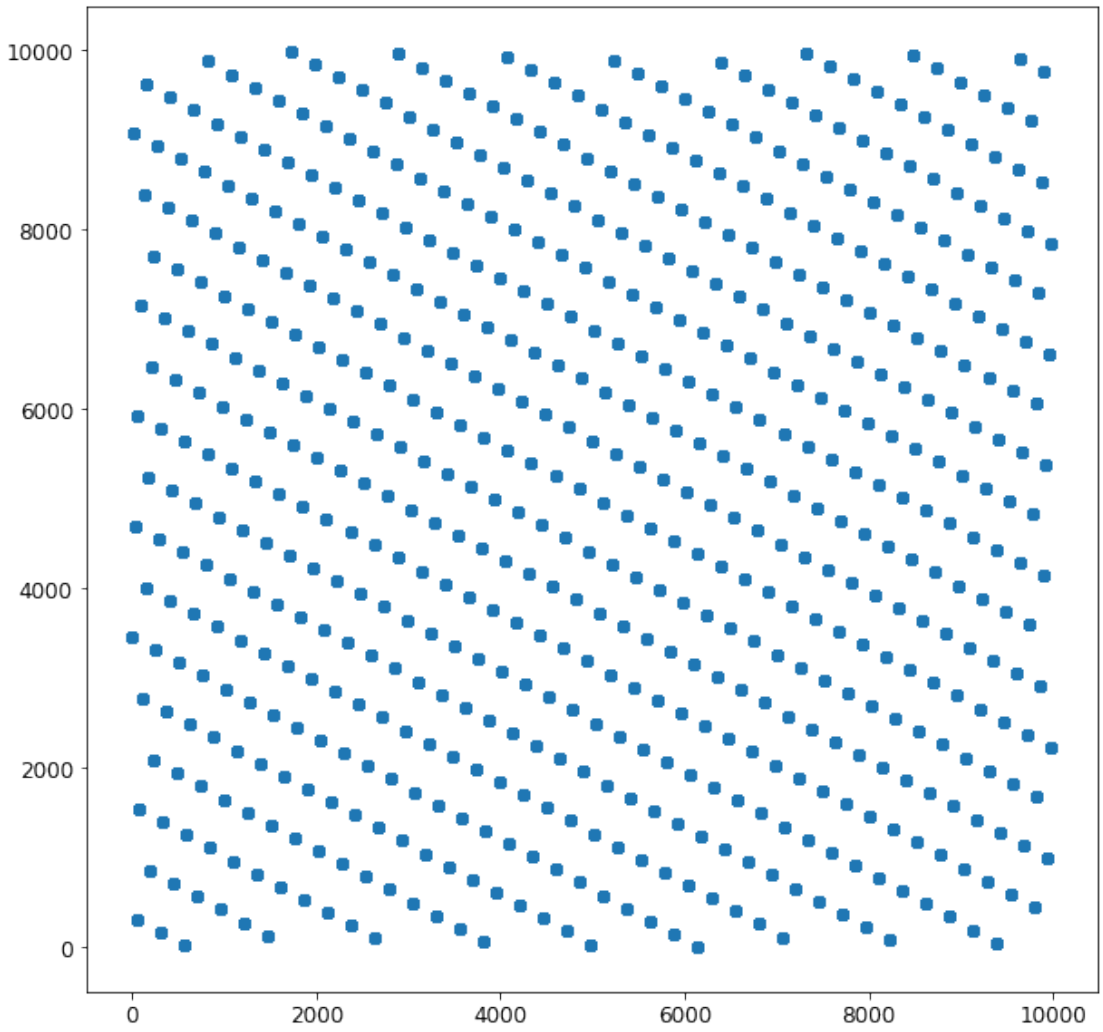




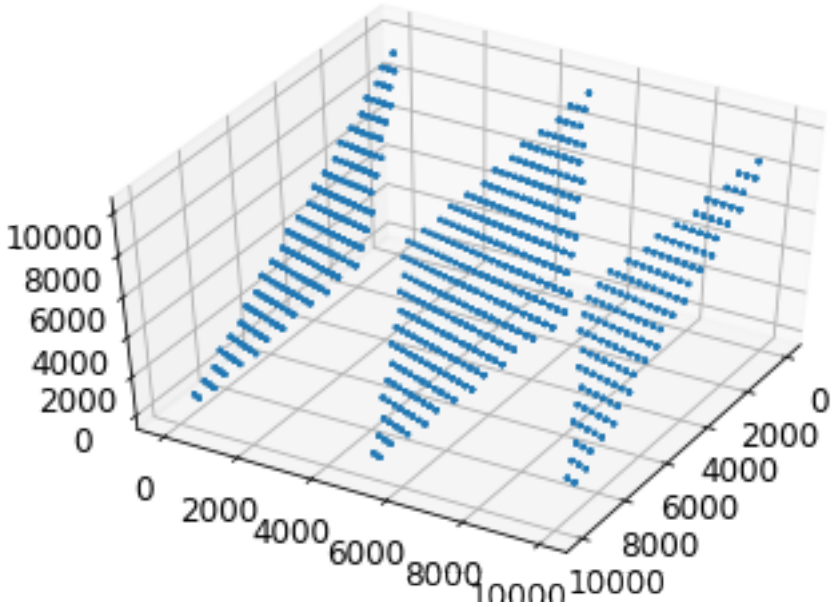
Die erhaltenen Zufallszahlen entsprechen nicht den Ansprüchen an einen guten Zufallsgenerator, da keine Gleichverteilung der Zahlen vorhanden ist. In dem Histogram sieht man jeweils einen Peak bei dem gewählten Startwert. Die Güte des Zufallsgenerators hängt jedoch nicht vom Startwert ab.

c)

```
In [20]: plt.figure(figsize = [10,10])
         plt.scatter(Liste1[0:-2:2], Liste1[1::2])

Out[20]: <matplotlib.collections.PathCollection at 0x1c5d69978d0>
```



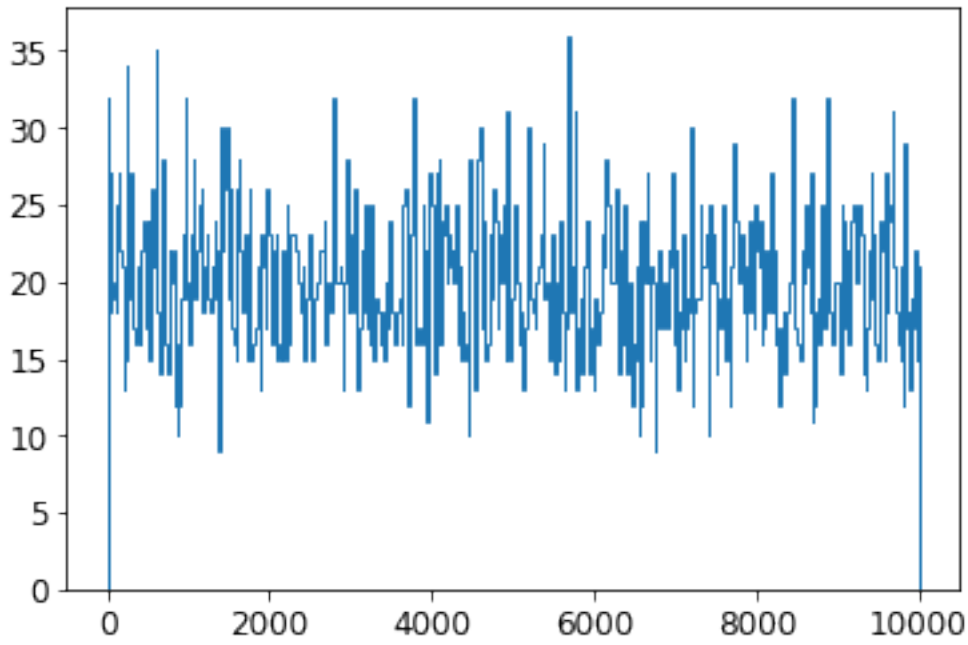
In [21]:
```
from mpl_toolkits .mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111 , projection ='3d')
ax.view_init(45, 30)
ax.scatter(
Liste1[0:-2:2], Liste1[1::2], Liste1[2::2],
lw=0,
s=5,
)
plt.show()
```



Der Spektraltest zeigt deutlich, dass es sich nicht um einen guten Zufallsgenerator handelt, da sowohl in 2D als auch in 3D ein starkes Muster zu erkennen ist.

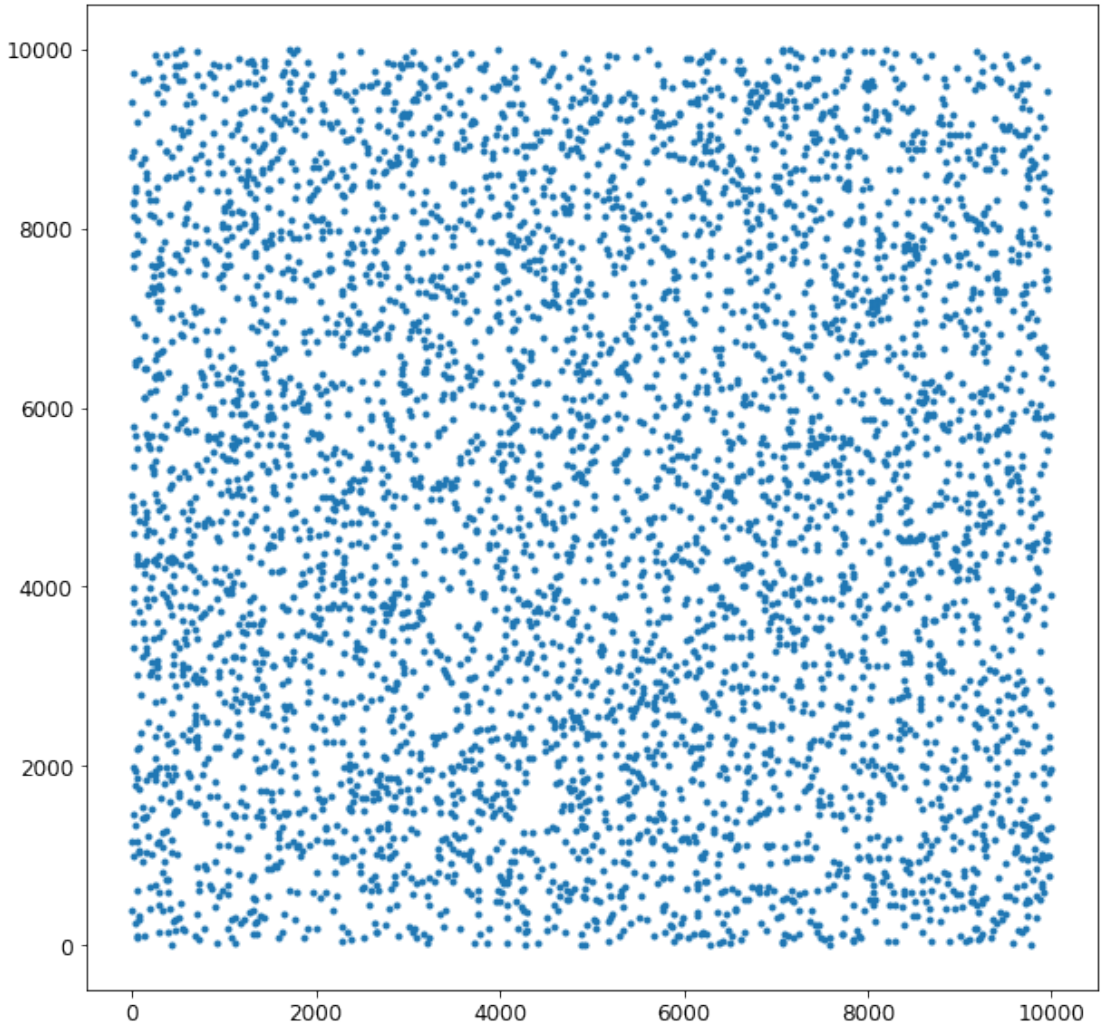
d)

```
In [22]: Random = np.random.uniform(0,10000,10000)
         plt.hist(Random, 500, range = [np.min(Random), np.max(Random)], histtype = 'step');
```

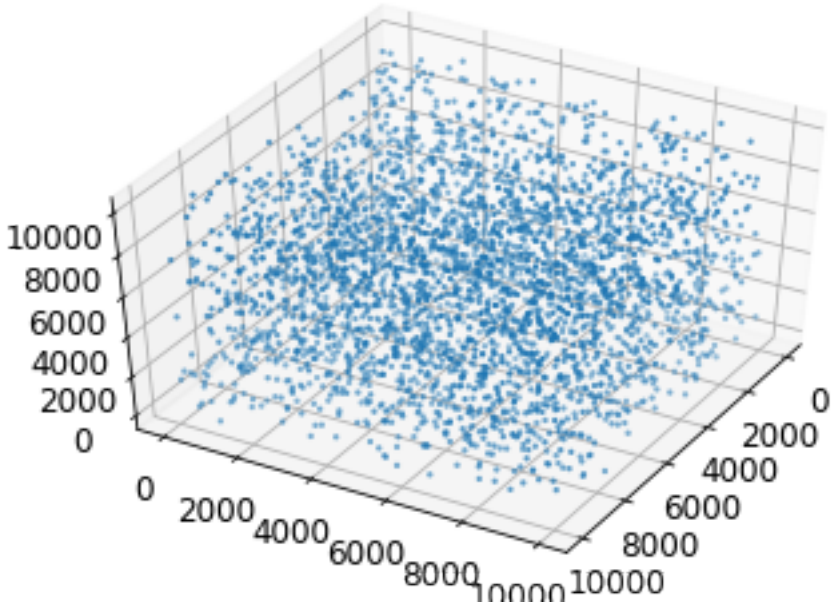


```
In [23]: plt.figure(figsize = [10,10])
         plt.scatter(Random[0::2], Random[1::2], marker = '.')
```

```
Out[23]: <matplotlib.collections.PathCollection at 0x1c5d6b83390>
```




```
In [24]: fig = plt.figure()
         ax = fig.add_subplot(111 , projection ='3d')
         ax.view_init(45, 30)
         ax.scatter(
         Random[0:-2:3], Random[1::3], Random[2::3],
         lw=0,
         s=5,
         )
         plt.show ()
```



e)

```
In [25]: x0 = np.linspace(0, 100, 201) # 0.5 steps
         De = [(linGen(x_0 = n, a = 5, b = 3, m = 1024))[1].tolist() for n in x0]
         # a = 5 is one of the values with max period
         Liste05 = []
         for n in De:
             if (0.5 in n) == True:
                 #print(n)
                 Liste05.append(x0[De.index(n)])
         print('In jeder Liste kommt der Wert 0.5 maximal einmal vor, da bei einer Periodenläng
         print('Der Wert 0.5 kann dabei für folgende Startwerte erhalten werden: ', Liste05[1:]
```

```
In jeder Liste kommt der Wert 0.5 maximal einmal vor, da bei einer Periodenlänge jeder Wert nur
```

```
Der Wert 0.5 kann dabei für folgende Startwerte erhalten werden:  [1.5, 2.5, 3.5, 4.5, 5.5, 6.5
```

1.3 Aufgabe 7

a)

```
In [26]: mu_x = 4
         mu_y = 2
         sigma_x = 3.5
         sigma_y = 1.5
         Cov = 4.2
```