

$$\begin{array}{c|c|c|c}
 5 & 6 & 7 & \\
 \hline
 \cancel{5/5} & \cancel{4,75/5} & \cancel{9/10} & \cancel{12,75/20}
 \end{array}$$

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 $[x_{min}, x_{max}]$:

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

Bilde Verteilungsfunktion $F(x)$:

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

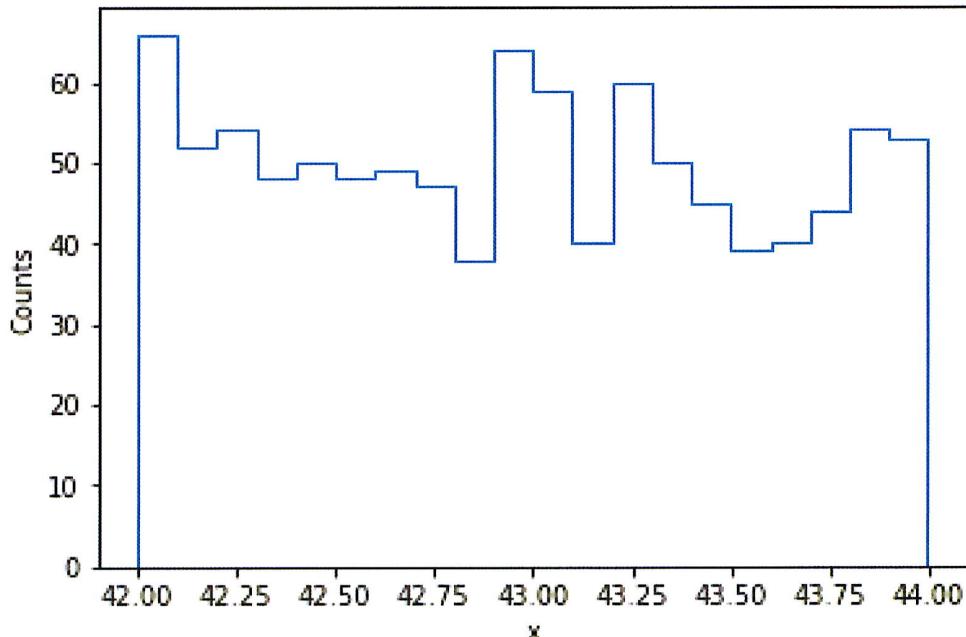
Bilde Umkehrfunktion $F^{-1}(y)$:

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

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()
```



1 P.

b) Exponentialgesetz:

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

Verteilungsfunktion:

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

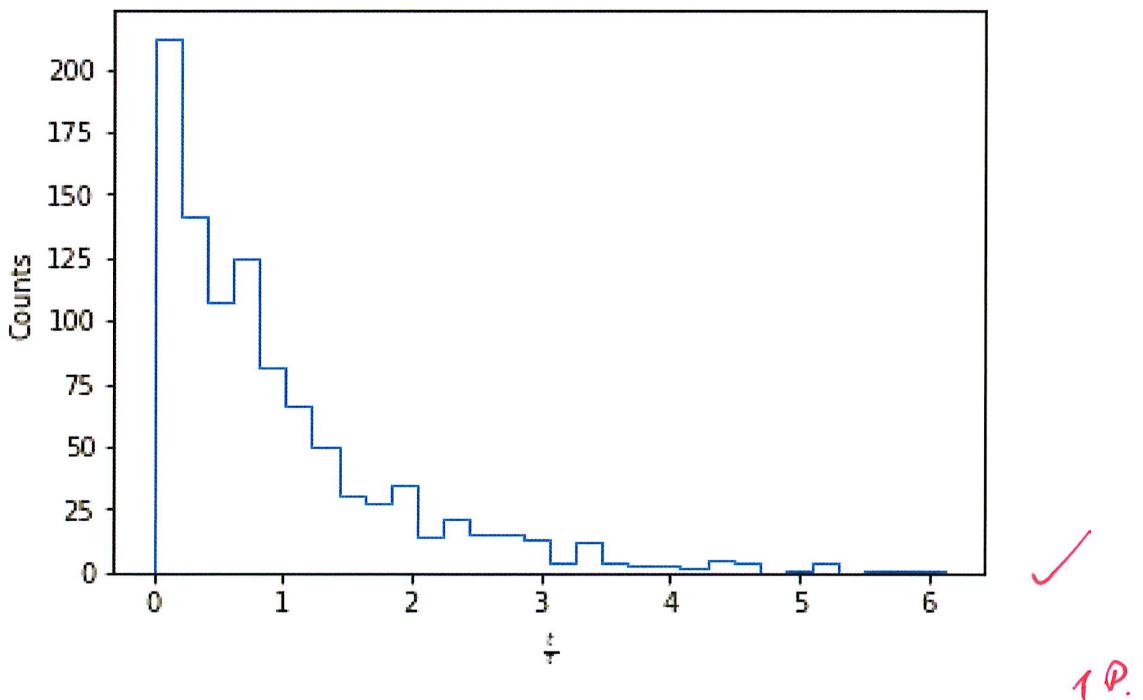
Umkehrfunktion:

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

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:

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

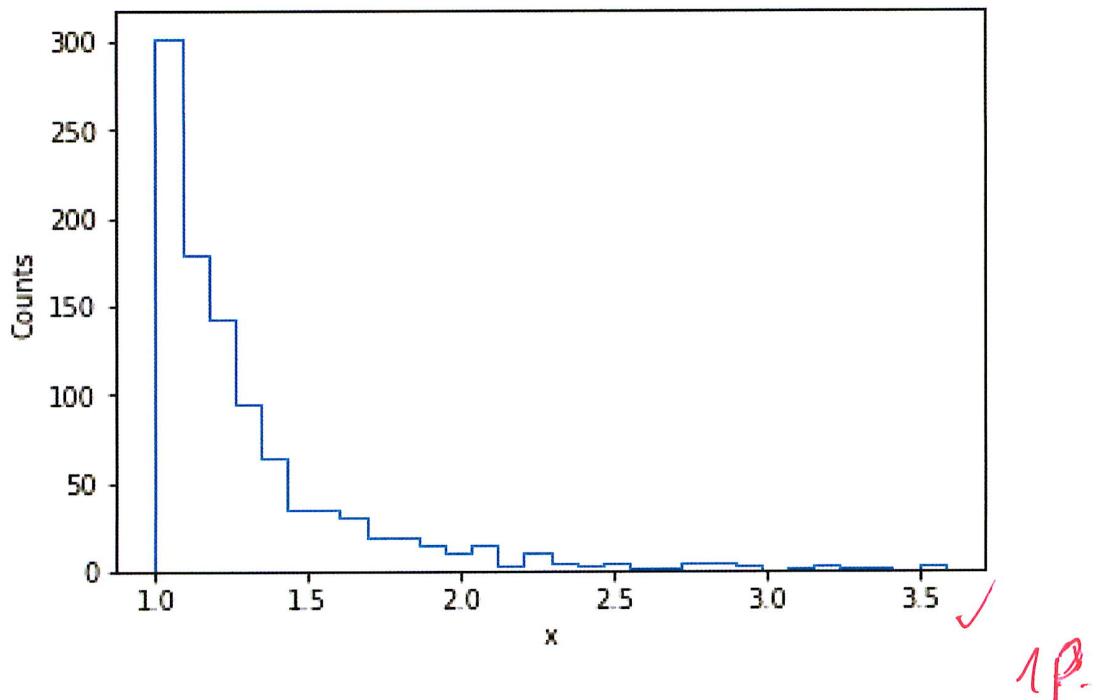
Umkehrfunktion:

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

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:

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

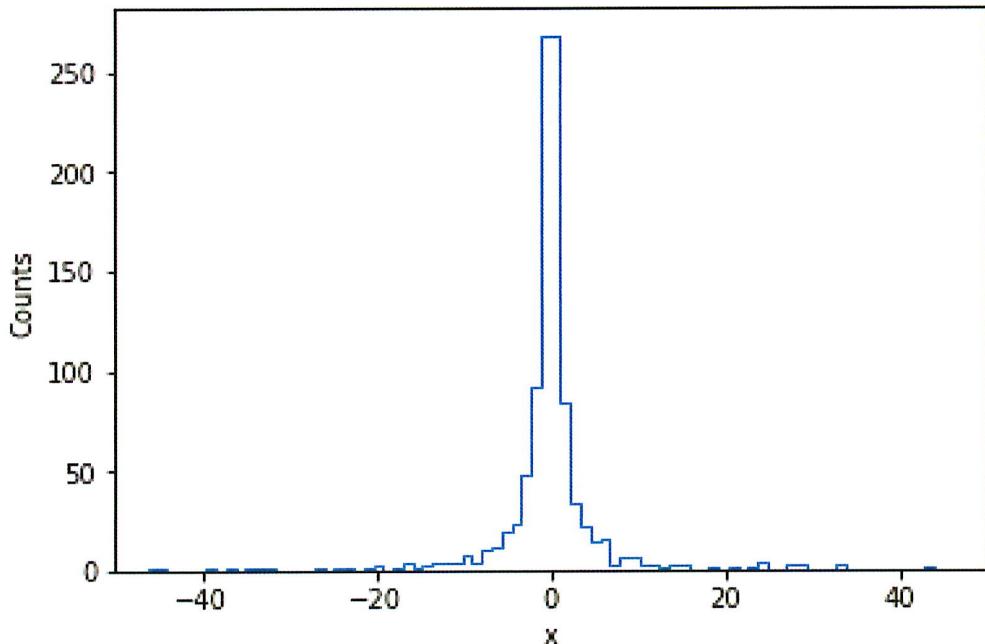
Umkehrfunktion:

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

Implementierung:

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

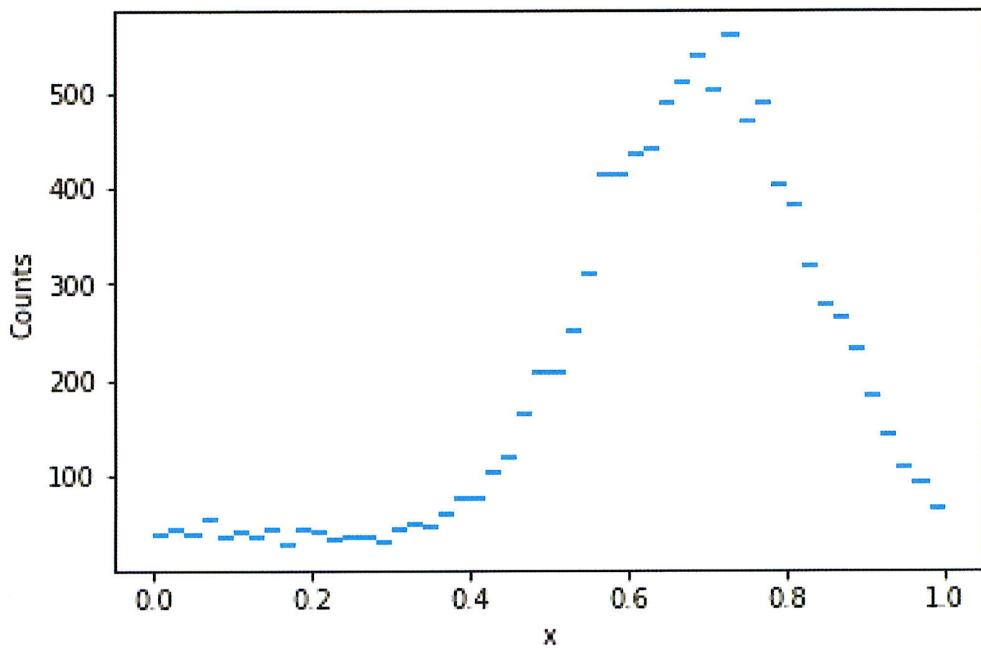
```
In [9]: fig, ax = plt.subplots(1, 1)
ax.hist(cauchy(uniform), bins = 500, histtype='step')
ax.set_xlim(-50, 50)
ax.set_xlabel('x')
ax.set_ylabel('Counts')
plt.show()
```



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()
```

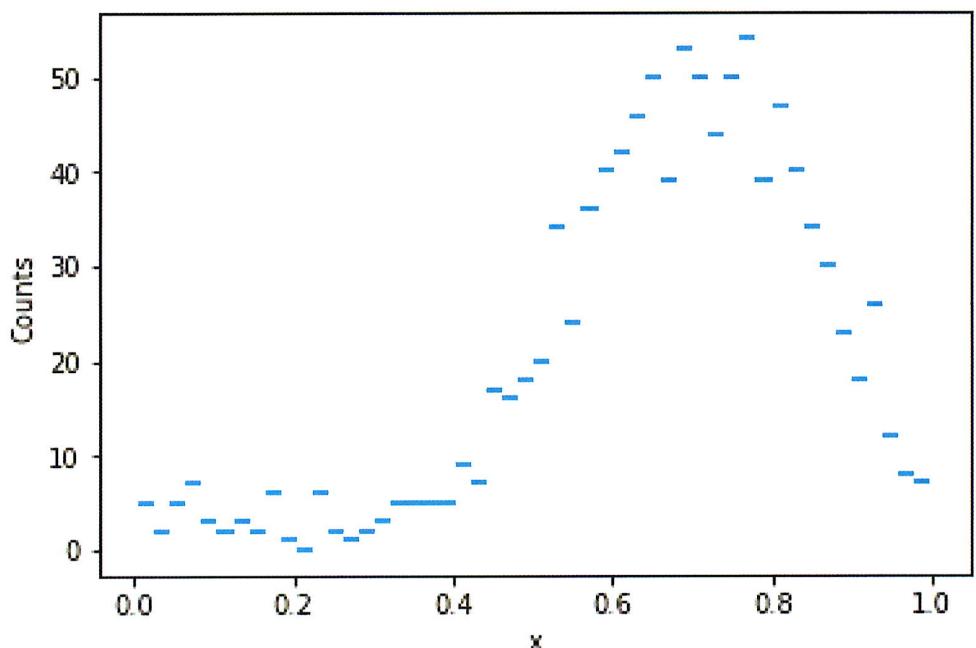


```
In [11]: norm = np.sum(counts * 0.2) #bin width is 0.2
norm_counts = counts / norm #normalize data
u1 = random.uniform(size = 10000) #
u2 = random.uniform(size = 10000) #two uniform samples

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(b
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.m
```

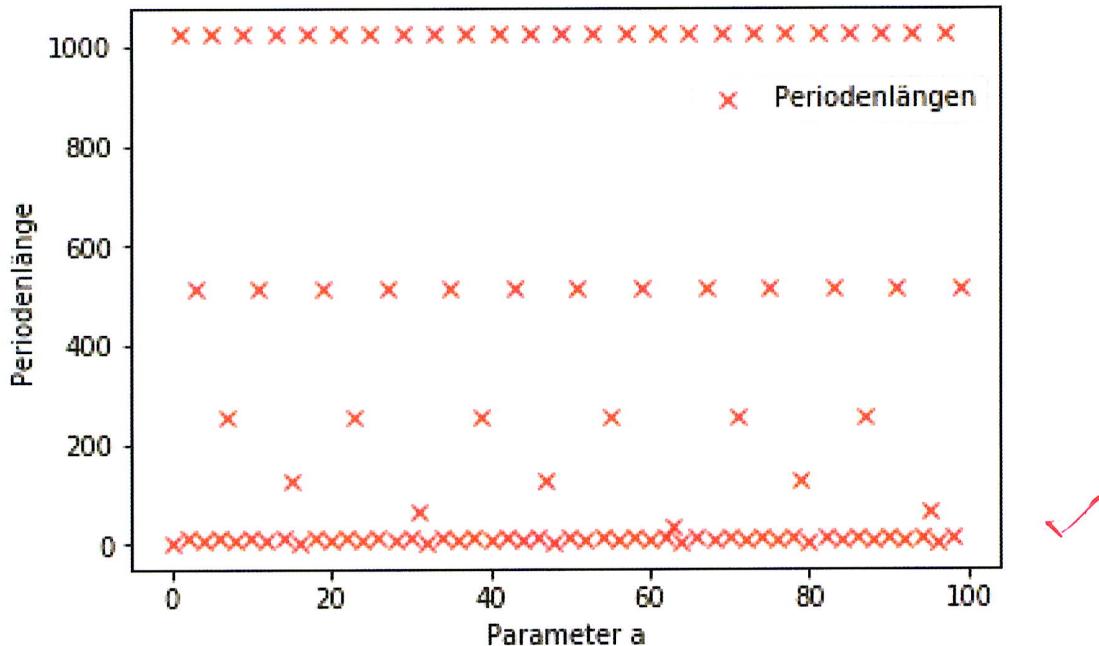
Die maximale Periodenlänge beträgt: 1024 ✓

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'Periodenlnge');
plt.xlabel('Parameter a')
plt.ylabel('Periodenlnge')
plt.legend(loc='upper right', bbox_to_anchor = (1,0.9))
```

Out[16]: <matplotlib.legend.Legend at 0x7fa275aff550>



Regeln fr a fehlen -0,25 P.

0,75 P.

b)

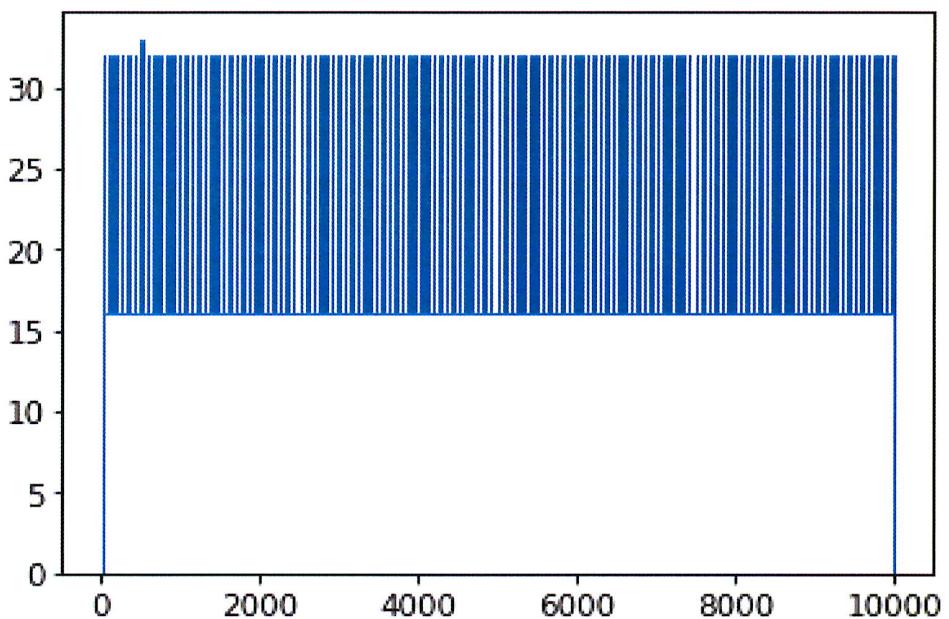
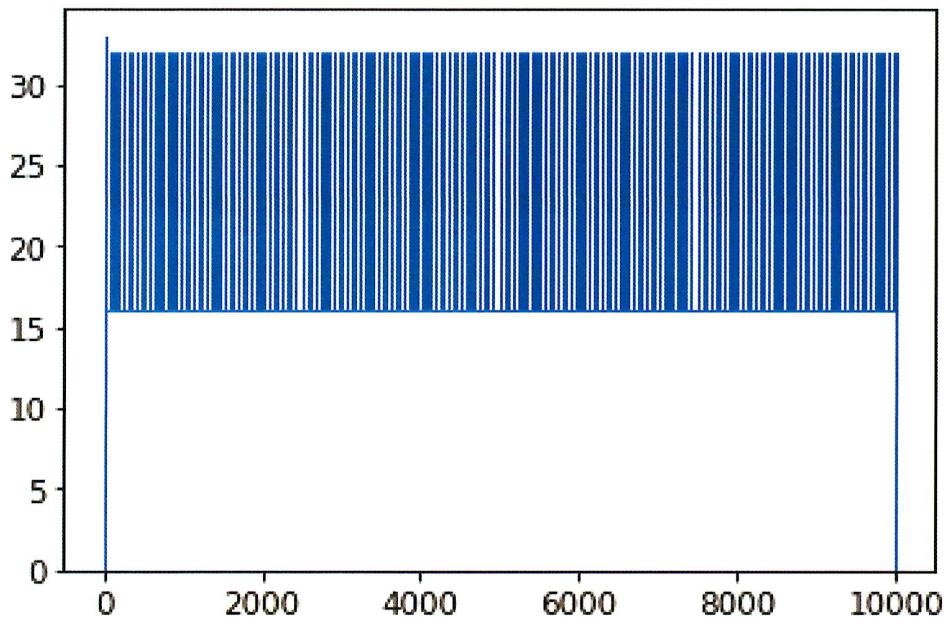
```
In [17]: def linGen2(a, b, m, x_0, length):
    random_numbers = [x_0]
    for i in range(length):
        random_numbers.append((a * random_numbers[i] + b) % m)
    return random_numbers
```

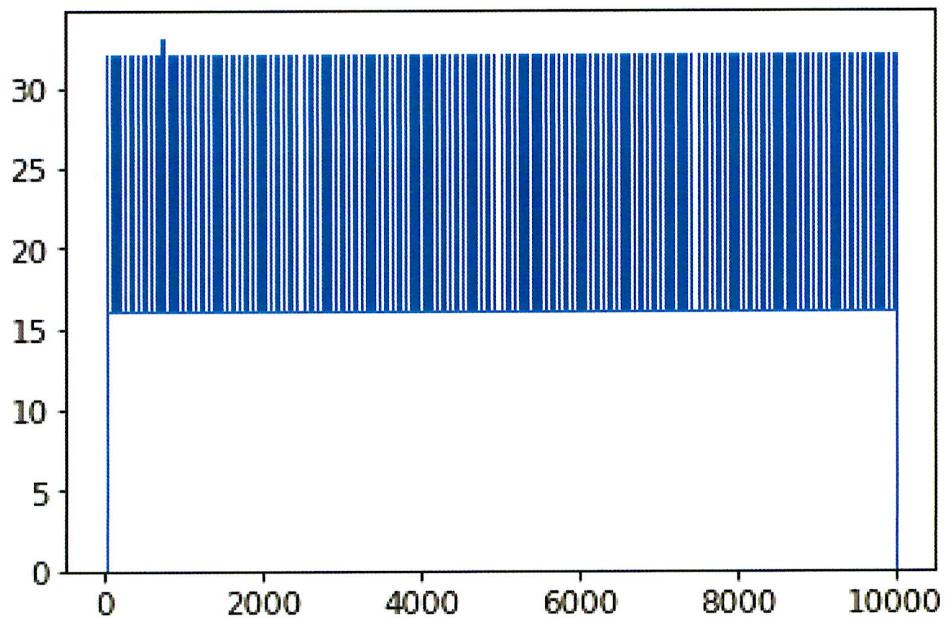
```
In [18]: Liste1 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 0, length = 10000)
Liste2 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 500, length = 10000)
Liste3 = linGen2(a = 1601, b = 3456, m = 10000, x_0 = 750, length = 10000)
```

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 =
plt.show()
plt.hist(Liste2, bins = 500, range = [np.min(Liste2), np.max(Liste2)], histtype =
plt.show()
```

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





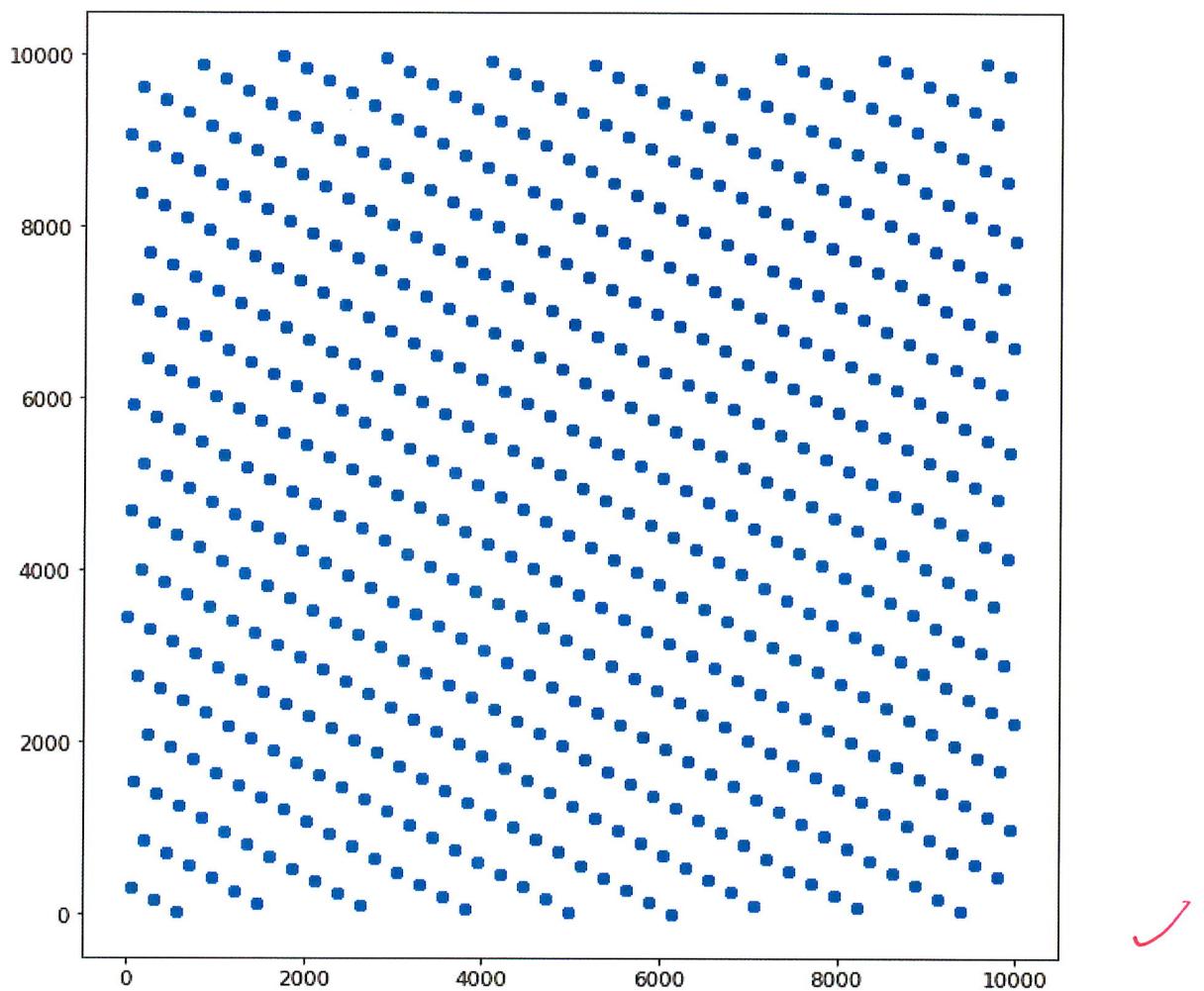
1P.

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

c)

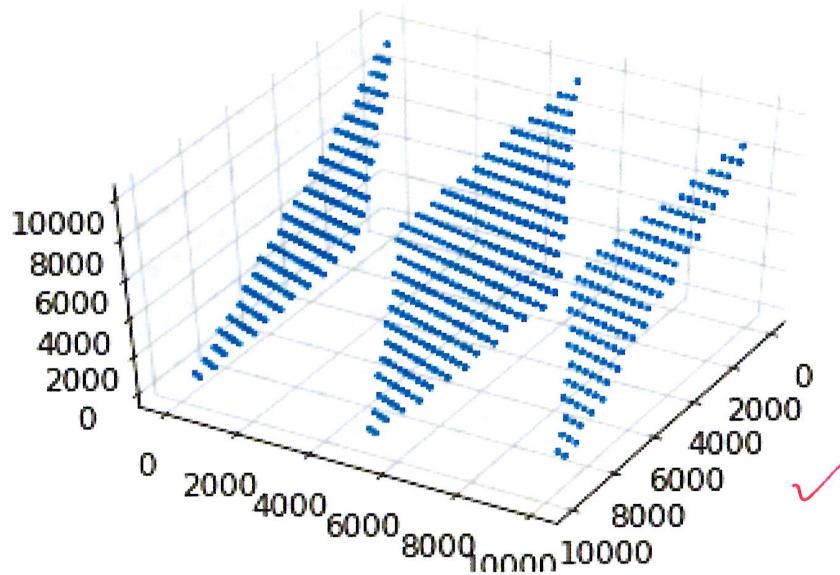
```
In [20]: plt.figure(figsize = [10,10])
plt.scatter(Liste1[0:-2:2], Liste1[1::2])
```

```
Out[20]: <matplotlib.collections.PathCollection at 0x7fa2738da3c8>
```



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()
```

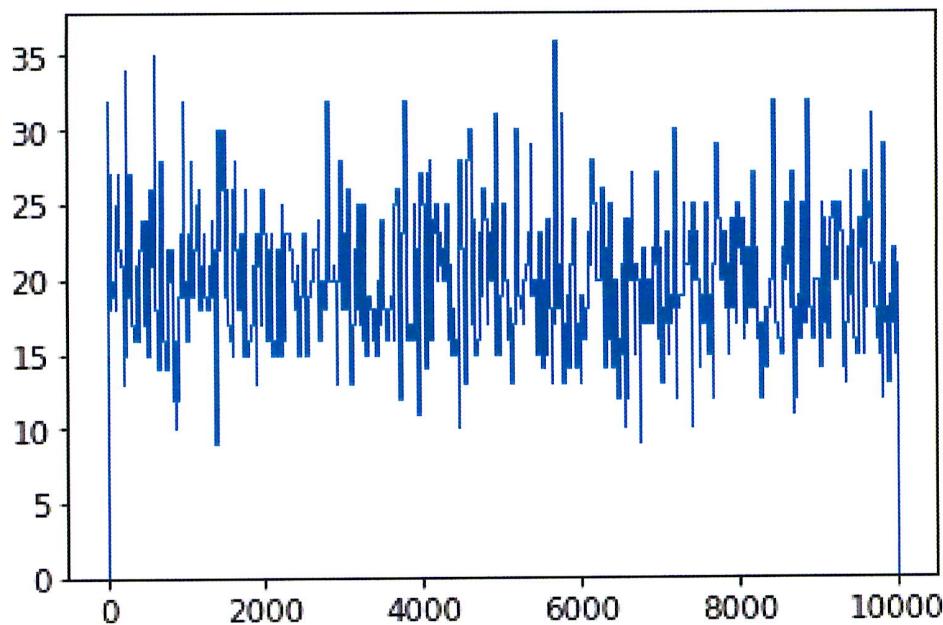


1P.

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.

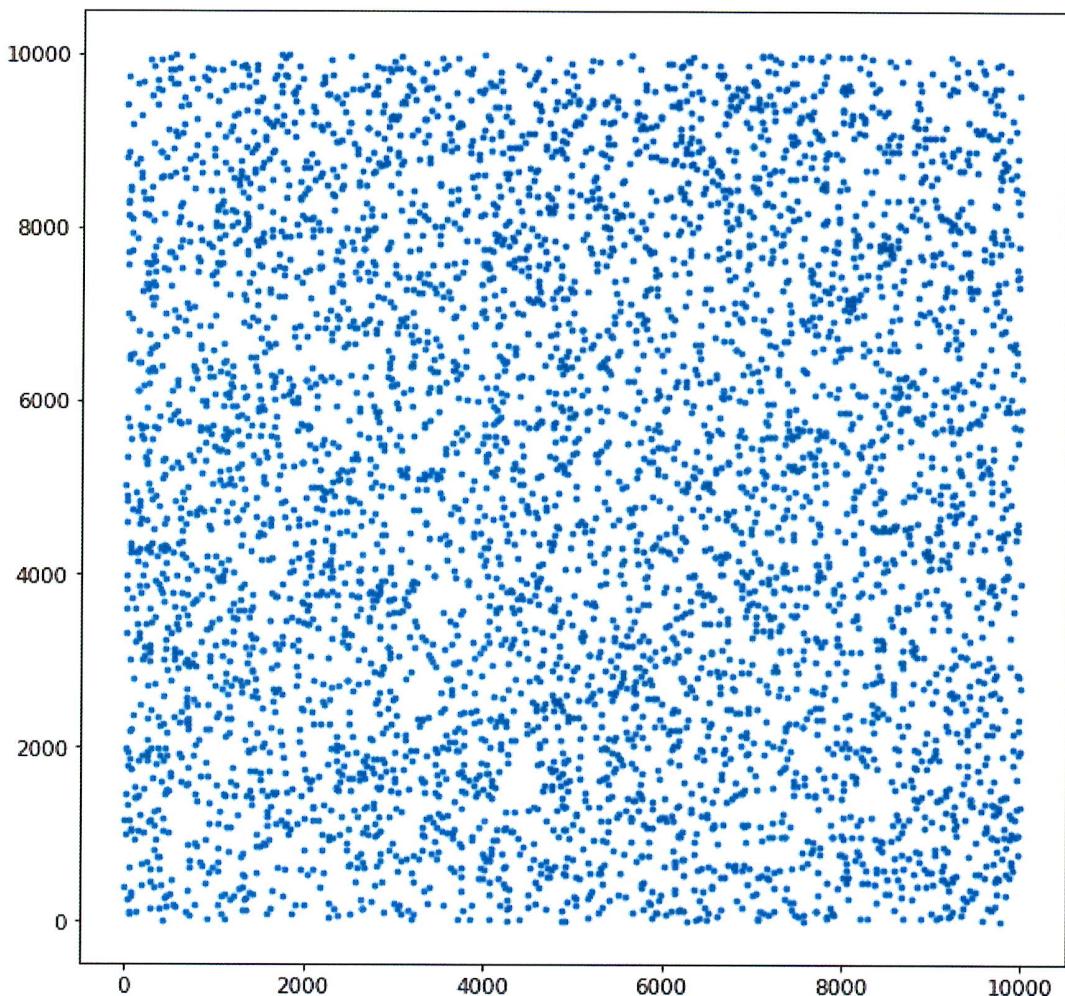
d)

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

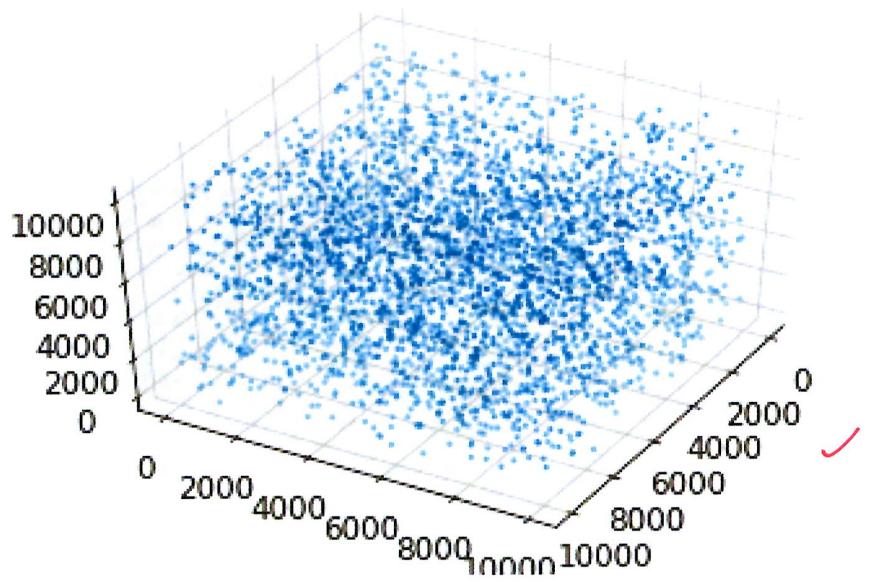


```
In [23]: plt.figure(figsize = [10,10])
plt.scatter(Random[0::2], Random[1::2], marker = 'o')

Out[23]: <matplotlib.collections.PathCollection at 0x7fa2737ae978>
```



```
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()
```



1 P.

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 Periodenlänge')
print('Der Wert 0.5 kann dabei für folgende Startwerte erhalten werden: ', Liste05)
```

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

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

1.3 Aufgabe 7

a)

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

in der Musterlösung
sind andere
Werte

der kann ich
leider nicht
lesen

okay, die Musterlösung wollte, dass
man die Werte $a = 1601$, $b = 3456$
and $m = 10000$ verwendet

1 P.

In [27]: $\text{Korr} = \text{Cov} / (\sigma_x * \sigma_y)$
 print("Der Korrelationskoeffizient beträgt:", Korr, '\n')

Der Korrelationskoeffizient beträgt: 0.8 ✓

0,5 P.

In [28]: $\text{COV} = [[\sigma_x^2, \text{Cov}], [\text{Cov}, \sigma_y^2]]$
 $\text{mean} = [\mu_x, \mu_y]$
 $\text{gauss_2d} = \text{np.random.multivariate_normal}(\text{mean}, \text{COV}, 15000)$

b)

Verwende die Schreibweise aus Aufgabenteil e):

$$f(u_x, u_y) = \lambda \exp(-\gamma [u_x^2 - 2\rho u_x u_y + u_y^2])$$

$f(u_x, u_y)$ soll konstant sein, somit muss lediglich der Exponent betrachtet werden:

$$u_x^2 - 2\rho u_x u_y + u_y^2 = -\frac{1}{\gamma} \text{konst.}$$

Es handelt sich dabei um eine Ellipsengleichung, somit liegen Punkte gleicher Wahrscheinlichkeit immer auf einer Ellipse.

1 P.

c)

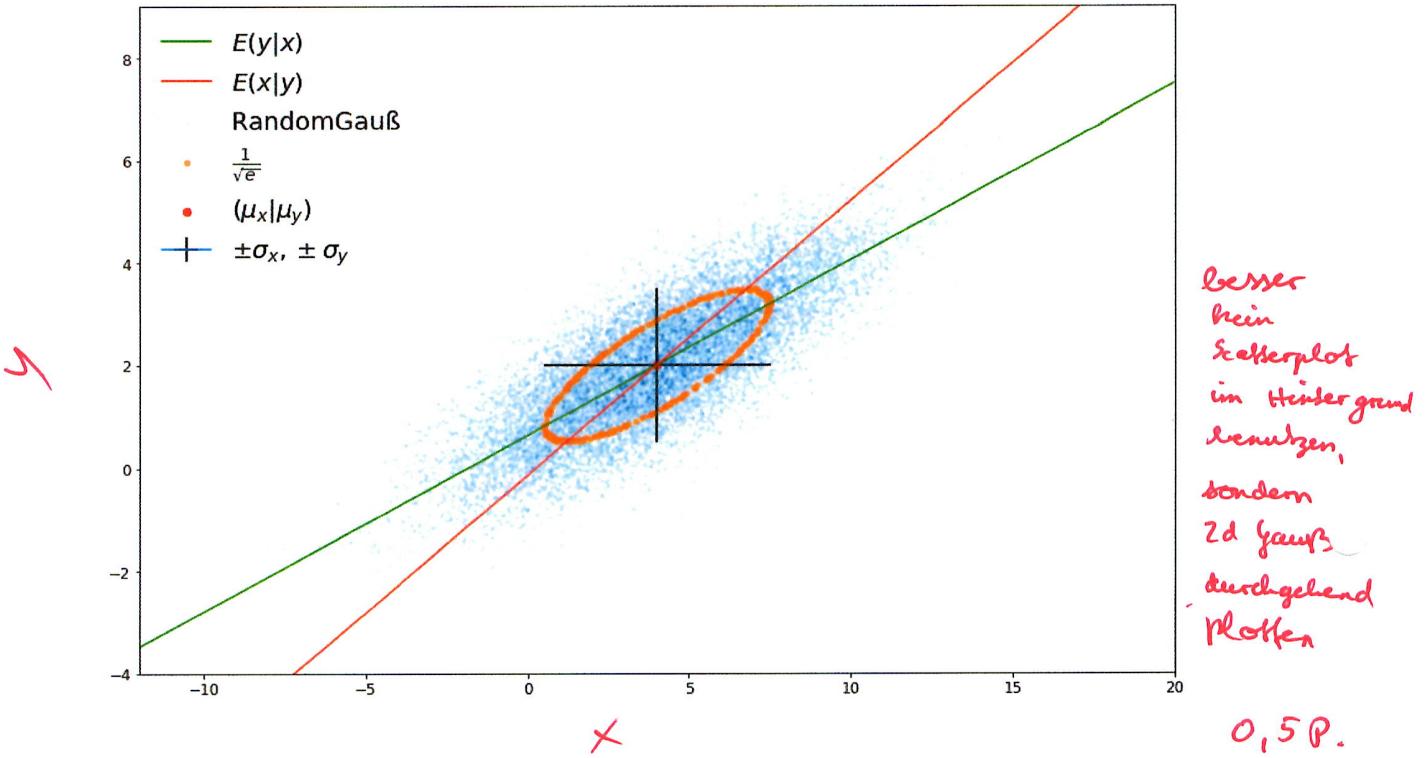
```
In [29]: def gauss2d(x,y):
    return 1 / (2 * np.pi * sigma_x * sigma_y * np.sqrt(1-Korr**2)) * np.exp(-1 /
```

TestGaus = gauss2d(gauss_2d[:,0], gauss_2d[:,1]) # zugehörige Wahrscheinlichkeitsw
e = gauss_2d[np.round(TestGaus,3) == np.round((1/np.sqrt(np.e))*np.max(TestGaus),3

```
plt.figure(figsize = [15,10])
plt.scatter(gauss_2d[:, 0], gauss_2d[:, 1], s=5, alpha=0.1, label='RandomGau')
plt.scatter(e[:,0], e[:,1], s=20, alpha = 0.7, label = r'$\frac{1}{\sqrt{e}}$')
plt.errorbar(mean[0], mean[1], xerr = (sigma_x), yerr=(sigma_y),
            ecolor='black', label = r'$\pm \sigma_x, \pm \sigma_y$')
plt.scatter(mean[0],mean[1], s = 40, alpha = 1,
            label = r'$(\mu_x | \mu_y)$', color = 'red')
x = np.linspace(-15,25)
plt.plot(x, Korr*sigma_y/sigma_x*(x-mu_x) + mu_y, 'g-', label = r'$E(y|x)$')
Ey = [Korr*sigma_x/sigma_y*(n-mu_y) + mu_x for n in x]
plt.plot(Ey, x, 'r-', label=r'$E(x|y)$')
plt.xlim(-12,20)
plt.ylim(-4,9)

plt.legend(loc='best', fontsize = 20)
```

Out[29]: <matplotlib.legend.Legend at 0x7fa273713710>



d) Die Matrix M ist hier die transponierte derjenigen Rotationsmatrix, die die Inverse der Kovarianzmatrix B diagonalisiert. B ist gegeben durch:

$$B = Cov^{-1} = \frac{1}{\sigma_x^2 \sigma_y^2 - Cov(x,y)} \begin{pmatrix} \sigma_y^2 & -Cov(x,y) \\ -Cov(x,y) & \sigma_x^2 \end{pmatrix}.$$

Das ist die Matrix, die in der quadratischen Form im Exponenten der mehrdim. Gaußverteilung steht. Eine Diagonalisierung entkoppelt also das Problem. Die Diagonalisierte Matrix hat die Eigenwerte auf der Diagonalen stehen. Hierbei handelt es sich um die gesuchten $\frac{1}{\sigma_x^2}$ und $\frac{1}{\sigma_y^2}$. Die Eigenwerte ergeben sich aus

$$\det \left(B - \frac{1}{\sigma_i'^2} \cdot 1 \right) \stackrel{!}{=} 0$$

Ausdrach durch unspektakuläre Umformungen zu

$$\frac{1}{\sigma_{x,y}'^2} = \frac{1}{2[\sigma_x^2 \sigma_y^2 - Cov^2(x,y)]} \left\{ \sigma_x^2 + \sigma_y^2 \pm \left[(\sigma_x^2 - \sigma_y^2)^2 + 4Cov^2(x,y) \right]^{\frac{1}{2}} \right\}.$$

Für die hier gegeben Werte kann man das dann ausrechnen:

```
In [30]: def newsig(sigx2, sigy2, cov2):
    return ((0.5 / (sigx2 * sigy2 - cov2)) * (sigx2 + sigy2
        + np.sqrt(
            (sigx2 - sigy2)**2
            + 4 * cov2
```

```

        )))**(-0.5),
(0.5 / (sigx2*sigy2 - cov2) * ( sigx2 + sigy2
- np.sqrt(
    (sigx2 - sigy2)**2
+ 4 * cov2
)))**(-0.5))

```

```
In [31]: mu_x = 4
mu_y = 2
sigx = 3.5
sigy = 1.5
cov = 4.2
```

```

gauss_2d = np.random.multivariate_normal([mu_x, mu_y],
[[sigx**2, cov], [cov, sigy**2]], 10000)

newsigma = newsig(sigx**2, sigy**2, cov**2)
print(f'sig_y, six_x = {newsigma}')

sig_y, six_x = (0.8485687438706406, 3.712132956525911)
```

18.

Nun soll die Rotaionsmatrix M bestimmt werden. Eine Rotationsmatrix in 2D ist allgemein gegeben durch

$$R = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

Die Transformation von Matrizen und Vektoren sähe dann so aus

$$S' = R^T S R, \quad \vec{x}' = R^T \vec{x}.$$

Die Matrix M ist in der Aufgabenstellung andersrum gewählt, sodass angesetzt werden kann

$$M = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

Von M kann man jetzt fordern, dass sie B diagonalisiert

$$\begin{pmatrix} \frac{1}{\sigma_y^2} & 0 \\ 0 & \frac{1}{\sigma_x^2} \end{pmatrix} \stackrel{!}{=} M B M^T.$$

Das gibt eins nices überbestimmtes Gleichungssystem für den Winkel α . Rechnen nur mit dem Eintrag unten links. Das ergibt die Gleichung

$$0 \stackrel{!}{=} \sigma_y^2 \sin(\alpha) \cos(\alpha) + \text{Cov}(x, y) (\cos^2(\alpha) - \sin^2(\alpha)) - \sigma_x^2 \sin(\alpha) \cos(\alpha)$$

Da kann man jetzt krasse Trigonometrie Skills rein stecken und erhält

$$0 = -\frac{1}{2} \sin(2\alpha) (\sigma_x^2 - \sigma_y^2) - \text{Cov}(x, y) \cos(2\alpha).$$

Daraus dann

$$\alpha = \frac{1}{2} \arctan \left(-\frac{2\text{Cov}(x, y)}{\sigma_x^2 - \sigma_y^2} \right).$$

✓

1,5 P.

In [32]: `def alpha(cov, sigx2, sigy2):
 return 0.5 * np.arctan(- 2 * cov / (sigx2 - sigy2))`

In [33]: `print(f'alpha ist ca. {alpha(cov, sigx**2, sigy**2) * 180 / np.pi:.3f} Grad')
alpha ist ca. -20.015 Grad` ✓

In [34]: `def rotate(x, alpha):
 #function that rotates a given vector x by an angle alpha.
 matrix = np.array([[np.cos(alpha), np.sin(alpha)],
 [-np.sin(alpha), np.cos(alpha)]])
 return matrix.dot(x)

old_x_axis = np.array([1, 0])
old_y_axis = np.array([0, 1])

new_x_axis = rotate(old_x_axis, alpha(cov, sigx**2, sigy**2))
new_y_axis = rotate(old_y_axis, alpha(cov, sigx**2, sigy**2))

def linear_function_from_vector(x, vec, center):
 # linear function along a vector x
 return vec[1] / vec[0] * (x - center[0]) + center[1]`

In [35]: `fig, ax = plt.subplots(1, 1, figsize = (10, 10))
ax.set_aspect('equal')

ax.scatter(gauss_2d[:, 0], gauss_2d[:, 1], s=5, alpha = 0.3, label = 'Data')

xlim = ax.get_xlim()
ylim = ax.get_ylim()
ax.set_xlim(xlim)
ax.set_ylim(ylim)
#plt.axes('equal')
xplot = np.linspace(xlim[0], xlim[1], 1000)

new_y_axis_values = linear_function_from_vector(xplot, new_y_axis, (mu_x, mu_y))
new_x_axis_values = linear_function_from_vector(xplot, new_x_axis, (mu_x, mu_y))

ax.plot(xplot,
 linear_function_from_vector(xplot, new_x_axis, (mu_x, mu_y)),
 color = 'r',
 linestyle = '--',
 label = 'Hauptachsen')
ax.plot(xplot,`

```

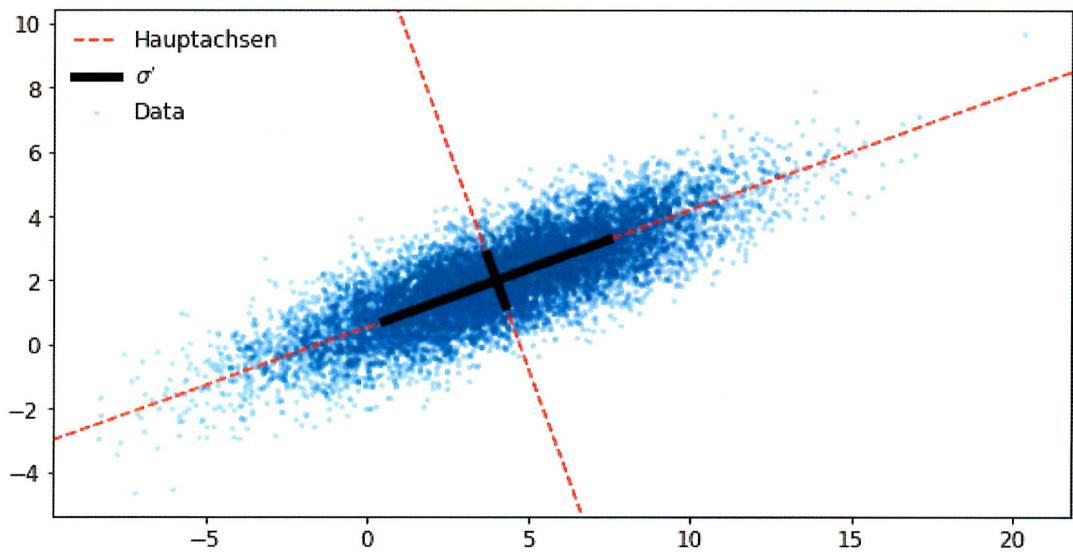
linear_function_from_vector(xplot, new_y_axis, (mu_x, mu_y)),
color = 'r',
linestyle = '--')

y = linear_function_from_vector(xplot, new_x_axis, (mu_x, mu_y))
mask = np.sqrt((xplot - mu_x)**2 + (y - mu_y)**2) <= newsigma[1]
ax.plot(xplot[mask], y[mask], linewidth = 5, color = 'k', label = '$\sigma$')

y = linear_function_from_vector(xplot, new_y_axis, (mu_x, mu_y))
mask = np.sqrt((xplot - mu_x)**2 + (y - mu_y)**2) <= newsigma[0]
ax.plot(xplot[mask], y[mask], linewidth = 5, color = 'k')

ax.legend()
plt.show()

```



e)

$$\begin{aligned}
f(y | x) &= \frac{f(x, y)}{g(x)} \text{ mit } g(x) = \int_{-\infty}^{\infty} f(x, y) dy \\
f(x, y) &= \frac{1}{2\pi\sigma_y\sigma_x\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)} \left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho \left(\frac{x-\mu_x}{\sigma_x}\right) \left(\frac{y-\mu_y}{\sigma_y}\right) + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 \right]\right) \\
\text{nutze } u_i &= \left(\frac{i-\mu_i}{\sigma_i}\right), \lambda = \frac{1}{2\pi\sigma_y\sigma_x\sqrt{1-\rho^2}} \text{ und } \gamma = \frac{1}{2(1-\rho^2)} : \\
f(u_x, u_y) &= \lambda \exp(-\gamma [u_x^2 - 2\rho u_x u_y + u_y^2])
\end{aligned}$$

$$g(x) = \lambda \sigma_y \exp(-\gamma u_x^2) \int_{-\infty}^{\infty} \exp(-\gamma([u_y - \rho u_x]^2 - \rho^2 u_x^2)) du_y \quad (1)$$

$$= \lambda \sigma_y \exp(-\gamma u_x^2) \exp(\gamma \rho^2 u_x^2) \int_{-\infty}^{\infty} \exp(-\gamma [u_y - \rho u_x]^2) du_y \quad (2)$$

$$= \frac{\lambda}{\sqrt{\gamma}} \sigma_y \exp(-u_x^2 [\gamma - \gamma \rho^2]) \int_{-\infty}^{\infty} \exp(-\theta^2) d\theta \quad (3)$$

$$= \frac{\lambda}{\sqrt{\gamma}} \sigma_y \exp(-u_x^2 [\gamma - \gamma \rho^2]) \sqrt{\pi} \quad (4)$$

$$f(y|x) = \frac{\sqrt{\gamma}}{\sigma_y \sqrt{\pi}} \exp(-\gamma[u_y^2 - 2\rho u_x u_y + \rho^2 u_x^2]) \quad \checkmark$$

Analog ergeben sich auch $h(y)$ und $f(x|y)$

sieht

$$h(y) = \frac{\lambda}{\sqrt{\gamma}} \sigma_x \exp(-u_y^2 [\gamma - \gamma \rho^2]) \sqrt{\pi} \quad \text{gut} \quad (5)$$

$$f(x|y) = \frac{\sqrt{\gamma}}{\sigma_x \sqrt{\pi}} \exp(-\gamma[u_x^2 - 2\rho u_x u_y + \rho^2 u_y^2]) \quad \checkmark \quad \text{aus} \quad (6)$$

f)

2 P.

$$\mathbb{E}(x|y) = \int_{-\infty}^{\infty} x f(x|y) dx$$

$$u_x = \frac{x - \mu_x}{\sigma_x} \Leftrightarrow x = u_x \sigma_x + \mu_x$$

$$\mathbb{E}(x|y) = \int_{-\infty}^{\infty} (u_x \sigma_x + \mu_x) \frac{\sqrt{\gamma}}{\sigma_x \sqrt{\pi}} \exp(-\gamma [u_x - \rho u_y]^2) \sigma_x du_x \quad (7)$$

$$= \int_{-\infty}^{\infty} u_x \sigma_x \sqrt{\frac{\gamma}{\pi}} \exp(-\gamma [u_x - \rho u_y]^2) du_x + \int_{-\infty}^{\infty} \mu_x \sqrt{\frac{\gamma}{\pi}} \exp(-\gamma [u_x - \rho u_y]^2) du_x \quad (8)$$

$$\text{substituiere } \sqrt{\gamma}(u_x - \rho u_y) = z \quad (9)$$

$$= \frac{\sigma_x \sqrt{\gamma}}{\sqrt{\pi}} \int_{-\infty}^{\infty} \left(\frac{1}{\sqrt{\gamma}} z + \rho u_y \right) \exp(-z^2) \frac{1}{\sqrt{\gamma}} dz + \frac{\mu_x}{\sqrt{\pi}} \int_{-\infty}^{\infty} \exp(-z^2) dz \quad (10)$$

$$= \sigma_x \rho u_y + \mu_x = \rho \frac{\sigma_x}{\sigma_y} (y - \mu_y) + \mu_x \quad (11)$$

Analog wieder für $\mathbb{E}(y|x)$

$$\mathbb{E}(y|x) = \rho \frac{\sigma_y}{\sigma_x} x - \mu_x + \mu_y \quad \checkmark$$

Beide Geraden sind in Aufgabenteil c) bereits eingezeichnet

2 P.