Stefan, Jonah

13	14	
13,5/15	5/5	18,5/20

Blatt 5

Aufgabe 13

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from pandas import DataFrame, Series
        from math import inf
        gamma = 2.7
        np.random.seed(42)
```

a)

```
In [2]: def PDF(E):
            return (gamma - 1) * E**(- gamma)

        def CDF(E):
            return 1 - E**(1 - gamma)

        def INV_CDF(y):
            return (1 - y)**(1 / (1 - gamma))
```

✓

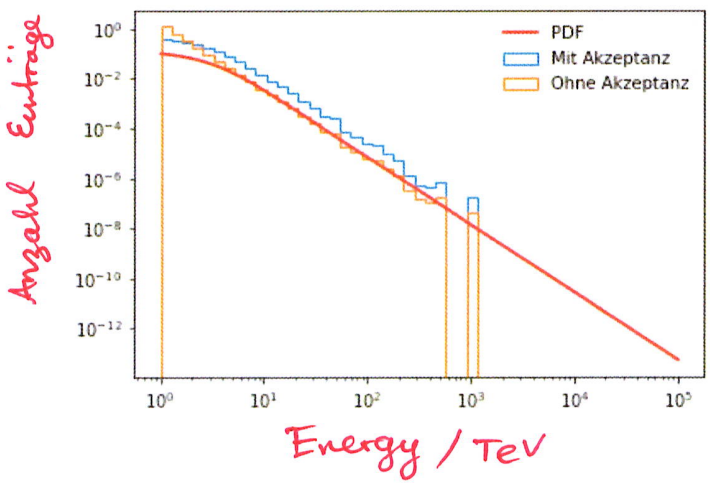
```
In [3]: y = np.random.uniform(0, 1, int(1e5))

In [4]: Energy = INV_CDF(y)
```

3P.

b)

```
In [5]: def P(E):
            return (1 - np.exp(-E / 2))**3

In [6]: uniform = np.random.uniform(size = len(y))
        AcceptanceMask = np.array([uniform < P(Energy) for uniform,
                                   Energy in zip(uniform, Energy)])
```

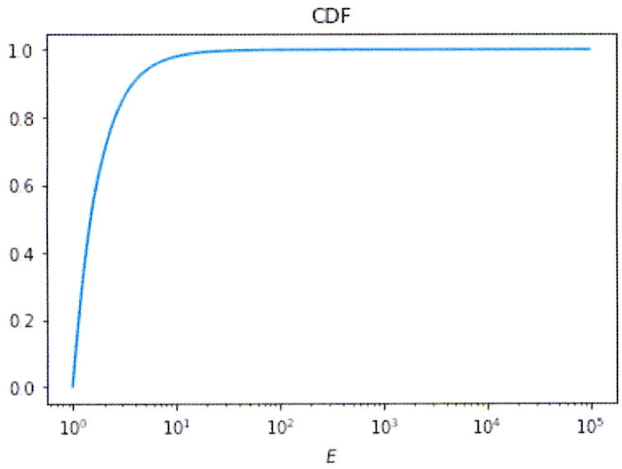
```
In [7]: plot_energy = np.logspace(0, 5, 1000)
        plt.plot(plot_energy, PDF(plot_energy) * P(plot_energy), 'r-', label = 'PDF')
        plt.hist(Energy[AcceptanceMask], bins = np.logspace(0, 5, 50),
                 density = True, histtype = 'step',
                 label = 'Mit Akzeptanz')
        plt.hist(Energy, bins = np.logspace(0, 5, 50), density = True,
                 histtype = 'step', label = 'Ohne Akzeptanz')
        plt.xscale('log')
        plt.yscale('log')
        plt.legend()
        plt.show()
```



Darstellung nicht optimal durch Normierung sind Bineinträge mit Akzeptanz höher als Bineinträge mit allen simulierten Events -1 P.

Kommentar: Man erkennt, dass ab einem Energiewert von ca. 1000 TeV keine Bins mehr befüllt werden. Dies liegt an der endlichen Länge des gleichverteilten Samples, das für die Rückweisungsmethode verwendet wird. Die unten abgebildete CDF zeigt, dass Hohe Werte für E aus Werten nahe 1 der Gleichverteilung hervorgehen. 2 P.

```
In [8]: plt.plot(plot_energy, CDF(plot_energy))
        plt.xscale('log')
        plt.xlabel('$E$')
        plt.title('CDF')
        plt.show()
```




```
In [9]: data = DataFrame()
        data['Energy'] = Series(Energy)
        data['AcceptanceMask'] = Series(AcceptanceMask)
```

Polarmethode: Erzeugt eine Standardnormalverteilung

In [10]:
```
def polar_method(size):
    v1 = 2 * np.random.uniform(0, 1, size) - 1
    v2 = 2 * np.random.uniform(0, 1, size) - 1
    s = v1**2 + v2**2
    while (True in (s >= 1)):
        v1[s >= 1] = 2 * np.random.uniform(0, 1, len(s[s >= 1])) - 1
        v2[s >= 1] = 2 * np.random.uniform(0, 1, len(s[s >= 1])) - 1
        s[s >= 1]  = v1[s >= 1]**2 + v2[s >= 1]**2

    x1 = v1 * np.sqrt(- 2 / s * np.log(s))
    x2 = v2 * np.sqrt(- 2 / s * np.log(s))
    return x1, x2
```

Die Funktion 'random_gaus' erzeugt eine 1- oder 2-dim Gaußverteilung, indem die Standardnormalverteilung aus der Polarmethode transformiert wird. Zusätzlich wird ermöglicht ausschließlich Werte aus einem gegebenen Bereich (z.B. Detektor) zu ziehen.

In [11]:
```
def random_gaus(mu, sig, size, rho = 0, two_dim = False, lim = (0, inf)):
    x_std, y_std = polar_method(size)
    x = np.sqrt(1 - rho**2) * sig * x_std + rho * sig * y_std + mu
    #formula for x transformation

    mask = ((x < lim[0]) | (x > lim[1]))  #generate new numbers, when out of lim
it
    while (True in mask):
        x_std[mask], y_std[mask] = polar_method(len(x[mask]))
        x[mask] = np.sqrt(1 - rho**2) * sig * x_std[mask] + rho * sig * y_std[ma
sk] + mu
        mask = ((x < lim[0]) | (x > lim[1]))

    if two_dim:
        y = sig * y_std + mu
        #formula for y transformation

        mask = ((y < lim[0]) | (y > lim[1]))
        while (True in mask):
            x_std[mask], y_std[mask] = polar_method(len(y[mask]))
            y[mask] = sig * y_std[mask] + mu
            mask = ((y < lim[0]) | (y > lim[1]))
        return x, y

    else:
        return x
```

c)

In [12]:
```
def hits(E):

    NumberOfHits = round(random_gaus(mu = 10*E, sig = 2*E,
                                     size = 1, lim = (0, inf))[0], 0)
    return NumberOfHits
```

In [13]:
```
NumberOfHits = [hits(E) for E in Energy]
```

In [14]:
```
data['NumberOfHits'] = Series(NumberOfHits)
```

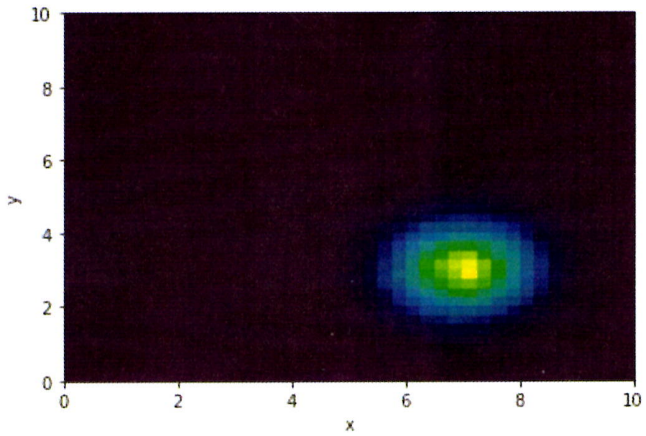
d)

In [15]:
```
def location(N, center):
    x = random_gaus(mu = center, sig = 1 / np.log10(N + 1),
                    rho = 0, size = 1, lim = (0, 10))[0]
    return x
```

In [16]:
```
x = [location(N, 7) for N in NumberOfHits]
y = [location(N, 3) for N in NumberOfHits]
```

In [17]:
```
data['x'] = Series(x)
data['y'] = Series(y)
```

In [18]:
```
plt.hist2d(x, y, bins = [40, 40], range = [[0, 10], [0, 10]])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



colorbar wäre noch schön

3P-

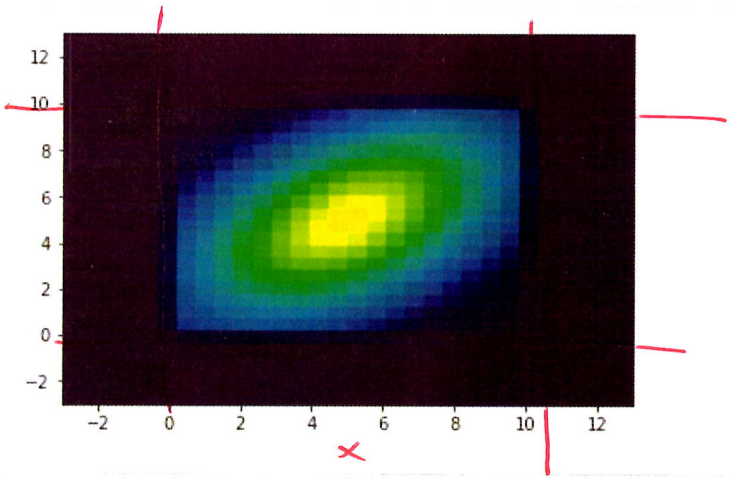
In [19]:
```
data.to_hdf('NeutrinoMC.hdf5', key = 'Signal')
```

e)

In [20]:
```
noise = DataFrame()
```

In [21]:
```
rho = 0.5
sig = 3
mu = 5

x, y = random_gaus(mu = 5, sig = 3, two_dim = True,
                   size = int(1e7), rho = 0.5,
                   lim = (0, 10))

noise['x'] = Series(x)
noise['y'] = Series(y)
```

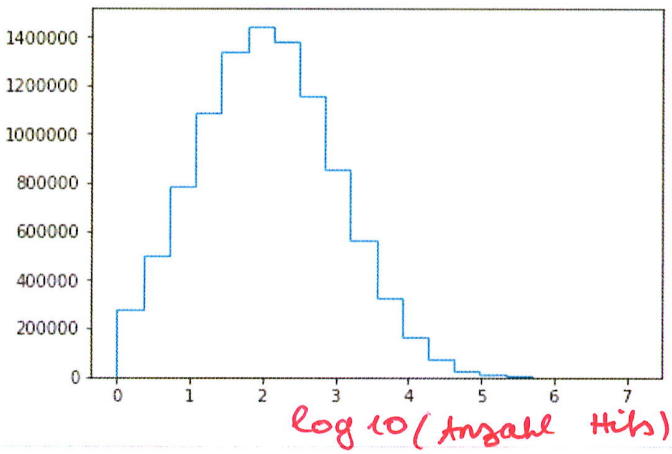
In [22]:
```
plt.hist2d(x, y, bins = [30, 30], range = [[-3, 13], [-3, 13]])
plt.show()
```



3P.

In [23]:
```
log_NumberOfHits = random_gaus(mu = 2, sig = 1, size = int(1e7))
NumberOfHits_noise = np.round(10**log_NumberOfHits, 0)
```

In [24]:
```
plt.hist(log_NumberOfHits, bins = 20, histtype='step')
plt.show()
```



Logscale für y-Achse

-0,5P

2,5P.

In [25]:
```
noise['NumberOfHits_noise'] = Series(NumberOfHits_noise)
noise.to_hdf('NeutrinoMC.hdf5', key = 'Background')
```

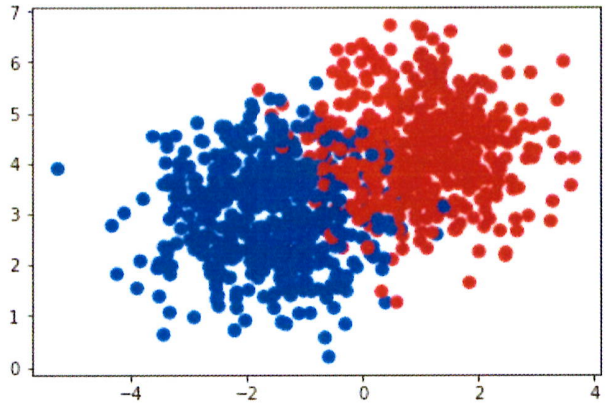
Aufgabe 14

a)

In [26]:
```
from sklearn.datasets import make_blobs
from matplotlib.colors import ListedColormap
discrete_cmap = ListedColormap([(0.8, 0.2, 0.3), (0.1, 0.8, 0.3), (0, 0.4, 0.8)]
)
```

Schreibt ruhig an die Achsen was zu sehen ist 😉

```
In [27]: X, y = make_blobs(n_samples=1000, centers=2, n_features = 4, random_state=0)
         plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=discrete_cmap)

Out[27]: <matplotlib.collections.PathCollection at 0x2598ee928d0>
```



ich würde die Punkte ruhig kleiner machen, damit man mehr sieht

x_1

x_0

1P.

b)

Bei der Hauptkomponentenanalyse geht es darum, eine Basis im Raum zu finden, entlang derer Eigenvektoren die Varianz maximiert wird. Die Vielzahl an Daten soll durch eine möglichst geringe Anzahl an aussgekräftigen Hauptkomponenten genähert werden, die Dimension der Datenpunkte wird also reduziert von d auf $k < d$.

Die Hauptkomponentenanalyse besteht dabei aus mehreren Schritten. Zuerst werden die Daten um den Mittelwert $\vec{\mu}$ zentriert: $x_i' = x_i - \mu$. Anschließend wird die Kovarianzmatrix bezüglich einer Zufallszahl mit beliebiger Dimension bestimmt. Aus der Kovarianzmarix ergeben sich entsprechend d Eigenwerte $\lambda_1, \ldots, \lambda_d$ und Eigenvektoren $\nu_1, \ldots, \nu_d$. Die Eigenvektoren werden gemäß ihrer Größe sortiert. Bei einer Reduzierung auf k Dimensionen werden nur die k höchsten Eigenwerte und Eigenvektoren benötigt, die anderen Werte können verworfen werden. Mithilfe dieser Eigenvektoren kann die Transformationsmatrix $\mathbf{W}$ bestimmt werden, welche die Eigenvektoren als Spalten enthält. Die einzelnen Vektoren können dann gemäß $\mathbf{X'} = \mathbf{WX}$ bestimmt werden.

aus Datenmatrix X

1P.

c)

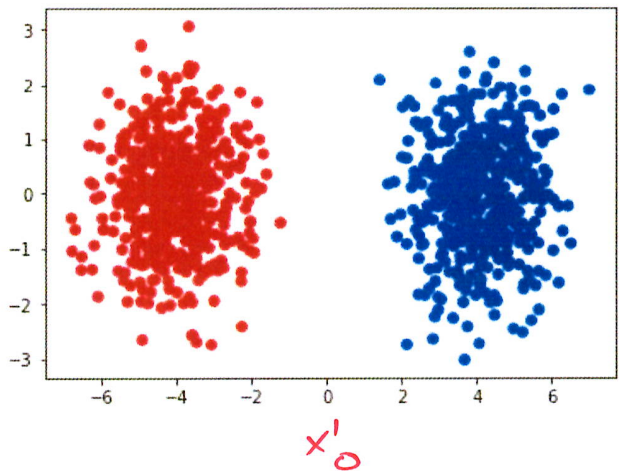
```
In [28]: # print(X.shape)
         c = np.cov(X, rowvar=False)

         l, W = np.linalg.eigh(c)

         # Reihenfolge umkehren. Größte Eigenwerte zuerst.
         l = l[::-1]
         W = W[:, ::-1]

         print(f'Die Eigenwerte lauten: {l}')

         Die Eigenwerte lauten: [17.51933024  0.99958442  0.98813673  0.89875061]
```

1P.

Die Eigenwerte definieren die Eigenräume, auf die der Datensatz projeziert werden kann. Die Eigenwerte geben die Varianz der durch sie definierten Hauptkomponenten an. Da die Varianz maximiert werden soll, werden die Achsen der größten Eigenwerte verwendet und der Rest verworfen. Hier ist es sinnvoll sich nur auf die Achse mit Eigenwert ~ 18 zu beschränken.

1P.

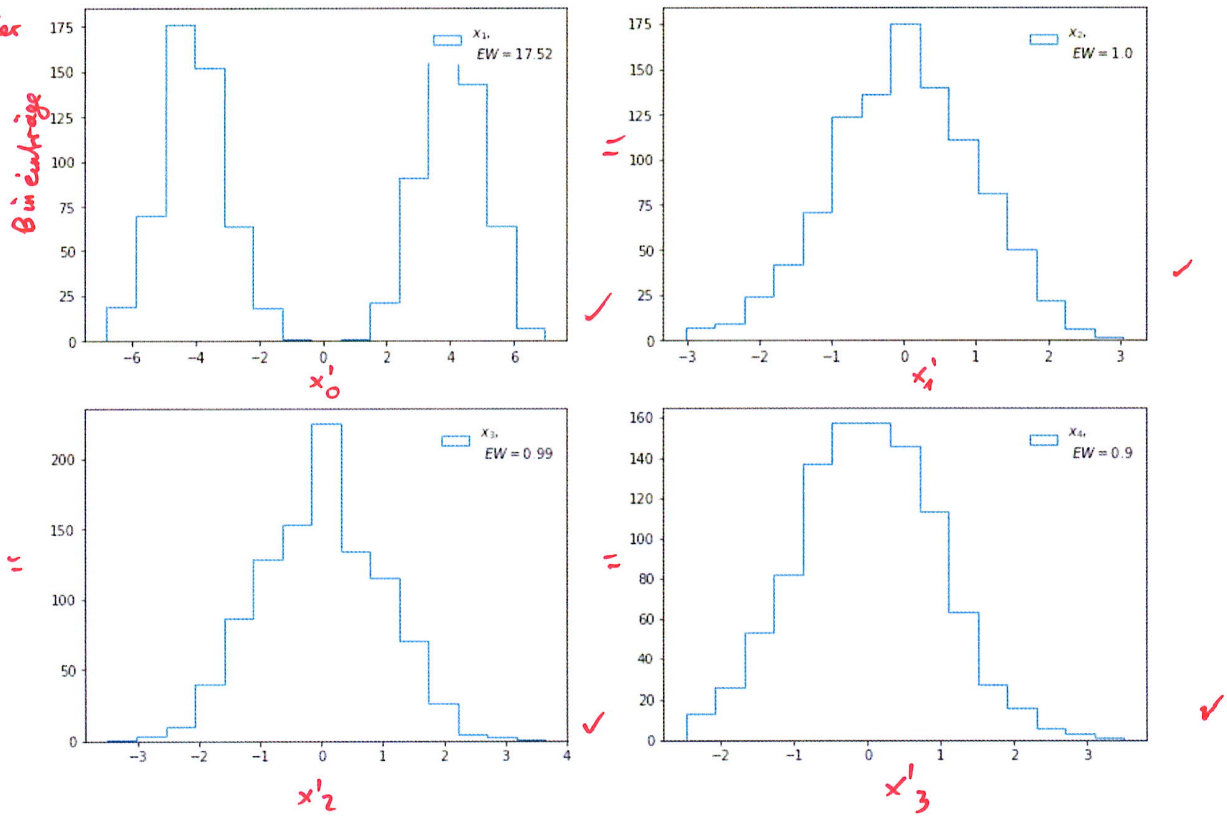
```
In [29]: from sklearn.decomposition import PCA

         pca = PCA(n_components = 4)
         transformed = pca.fit_transform(X)
         plt.scatter(transformed[:, 0], transformed[:, 1], c=y, cmap = discrete_cmap)
         plt.show()
```



d)

```
In [30]: fig = plt.figure(figsize = (14, 10))
         for i in range(4):
             ax = plt.subplot(221 + i)
             ax.hist(transformed[:, i], bins = 15,
                     histtype = 'step',
                     label = f'$x_{i + 1}$,\n $EW = {round(l[i], 2)}$')
             ax.legend()
         plt.show()
```

man hätte hier noch die 2 Populationen farblich kennzeichnen können



Kommentar: Es zeigt sich, dass die Hauptachse mit dem größten Eigenwert tatsächlich die beste zur Trennung der Daten ist.

1P.