Aufgabe10

November 15, 2018

Aufgabe 10 SMD

Eine korrelierte 2D GauSS Verteilung hat die Form:

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y(1-\rho^2)} e^{\frac{-1}{2(1-\rho^2)}\left(\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 - 2\rho\frac{x-\mu_x}{\sigma_x}\frac{y-\mu_y}{\sigma_y}\right)}$$

bzw.

$$f(u_x, u_y) = \lambda e^{-\gamma(u_x^2 - 2\rho u_x u_y + u_y^2)}$$

$$g(u_x) = \int_{-\infty}^{\infty} f(x,y)\mathrm{d}y = \frac{\lambda}{\sqrt{\gamma}}\sigma_y\sqrt{\pi}e^{-u_x^2(\gamma-\gamma\rho^2)}$$

Es gilt nach der Formel für bedingte Wahrscheinlichkeiten:

$$f(u_y|u_x) = \frac{f(u_x,u_y)}{g(u_x)} = \frac{1}{\sqrt{\gamma}}\sigma_y\sqrt{\pi}e^{-\gamma((\rho u_x)^2 - 2\rho u_x u_y + u_y^2)}$$

Für $u_x = \frac{\overline{x}}{\sigma_x} = \frac{x-\mu_x}{\sigma_x}$, $u_y = \frac{\overline{y}}{\sigma_y} = \frac{y-\mu_y}{\sigma_y}$ und $\gamma = \frac{1}{2(1-\rho^2)}$ stimmt dies mit der Formel aus der Aufgabenstellung überein. Für die Korrelation ρ' gilt dabei $\rho' = \sqrt{\frac{\alpha}{1+\alpha^2}}$ mit $\alpha = b\frac{\sigma_x}{\sigma_{y|x}}$. AuSSerdem gilt:

$$E(y|x) = bx + a = \frac{\rho'\sigma_y'}{\sigma_x'}(x - \mu_x') + \mu_y'$$

Durch Koeffizientenvergleich ergibt sich:

$$\sigma_y' = b\frac{\sigma_x'}{\rho'}$$

und

$$\mu_y' = a + \rho'\frac{\sigma_y'}{\sigma_x'}\mu_x'$$

Aufgabenteil b)

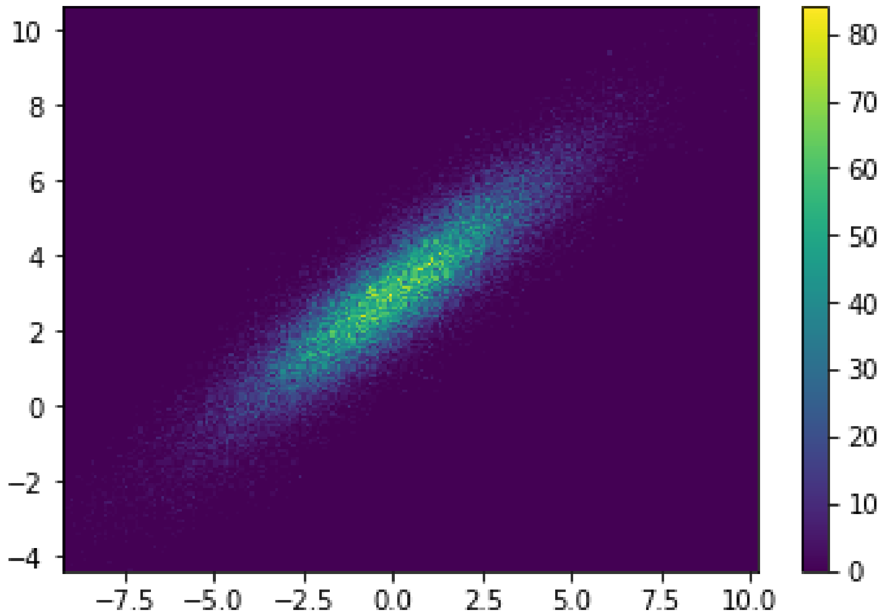
Anmerkung: Da ich nicht auf dem Schirm hatte, dass ich dazu vorgefertigte Funtionen nutzen kann, habe ich die Werte nach den jeweiligen Verteilungsfunktionen duch den Metropolis-Algorithmus mit einer gleichverteilten Schritt-PDF und einer Schrittweite von 1 erstellt. Dadurch konnte ich leider keinen Randomseed setzen.

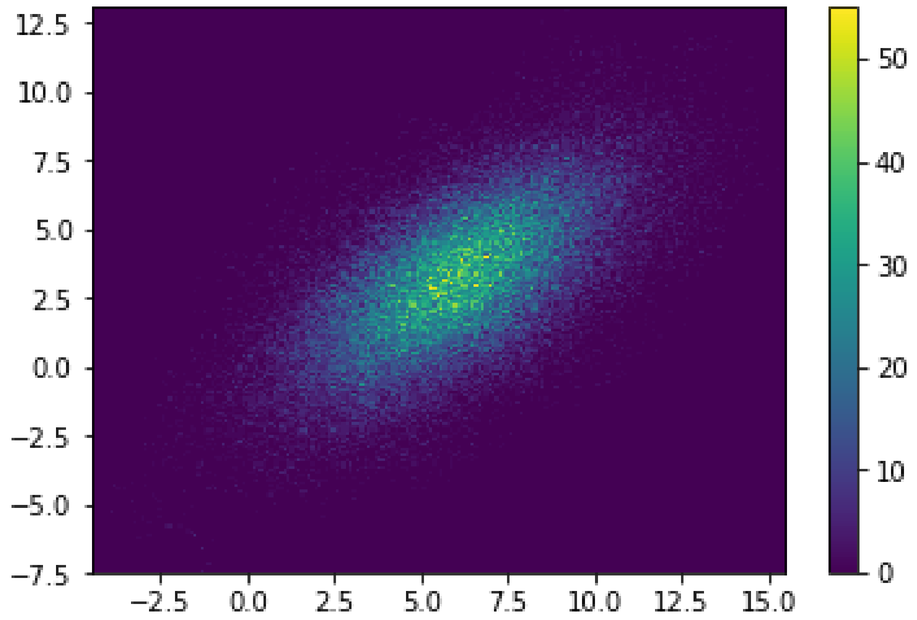
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import scipy.constants as const

In [2]: sx0=3.5
        sy0=2.6
        ux0=0
        uy0=3
        rho0=0.9

        a=-0.5
        b=0.6
        ux=6
        sx=3.5
        syx=1
        alpha=b*sx/syx
        rho=(alpha/(1+alpha**2))**0.5
        sy=b*sx/rho
        uy=a+rho*sy*ux/sx
        print(rho)
        print(sy)
        print(uy)

        def p0(x,y):
            return 1/(2*const.pi*sx0*sy0*
                      (1-rho0**2))*np.exp((-1/(1-rho0**2))*
                                          (((x-ux0)/sx0)**2+((y-uy0)/sy0)**2-
                                           2*rho0*((y-uy0)/sy0)*((x-ux0)/sx0)))
        def p1(x,y):
            return 1/(2*const.pi*sx*sy*
                      (1-rho**2))*np.exp((-1/(1-rho**2))*
                                         (((x-ux)/sx)**2+((y-uy)/sy)**2-
                                          2*rho*((y-uy)/sy)*((x-ux)/sx)))

        def Metro0(s,x0,number):
            w=[x0]
            time = [0]
            for i in range(1,number):
                y=w[i-1]+np.random.uniform(-s,s,2)
                if min(1,p0(y[0],y[1])/p0(w[i-1][0],w[i-1][1])) < np.random.uniform(0,1):
                    w.append(w[i-1])
                else:
                    w.append(y)
                time.append(i)
            return time,w

        time0, w0 = Metro0(1,(0,3),100000)
```

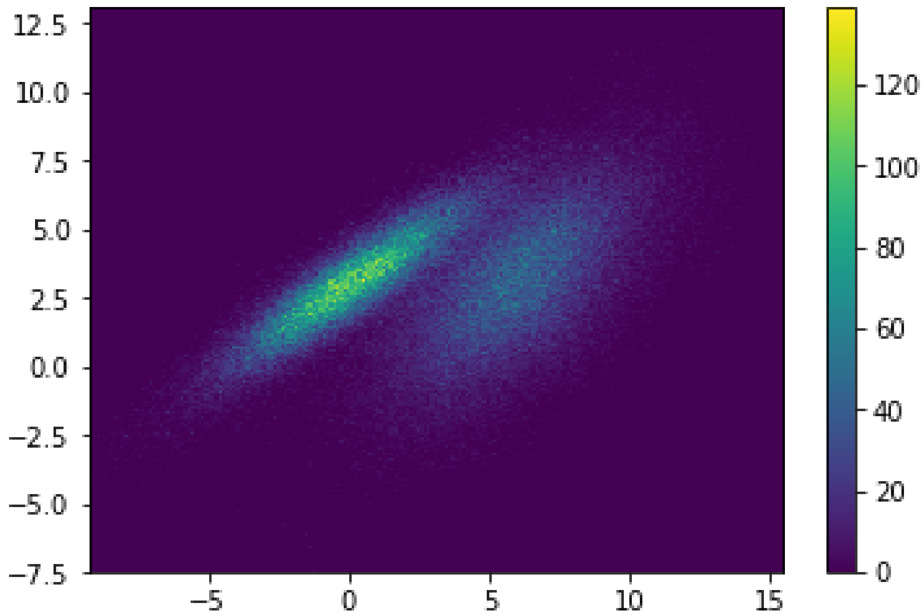
```
def Metro1(s,x0,number):
    w=[x0]
    time = [0]
    for i in range(1,number):
        y=w[i-1]+np.random.uniform(-s,s,2)
        if min(1,p1(y[0],y[1])/p1(w[i-1][0],w[i-1][1])) < np.random.uniform(0,1):
            w.append(w[i-1])
        else:
            w.append(y)
        time.append(i)
    return time,w

time1, w1 = Metro1(1,(ux,uy),100000)

x0=[]
y0=[]
x1=[]
y1=[]

for i in time0:
    x0.append(w0[i][0])
    y0.append(w0[i][1])


for i in time1:
    x1.append(w1[i][0])
    y1.append(w1[i][1])

#plt.scatter(x0,y0,s=5,alpha=0.002)
#plt.show()
plt.hist2d(
    x0,
    y0,
    bins=[200, 200]
)
plt.colorbar()
plt.show()
plt.hist2d(
    x1,
    y1,
    bins=[200, 200]
)
plt.colorbar()
plt.show()

yg = y0+y1
xg = x0+x1
plt.hist2d(
```

```
        xg,
        yg,
        bins=[200, 200]
    )
    plt.colorbar()
    plt.show()
```

```
0.6230329489303637
3.370608253713267
3.1000000000000005
```



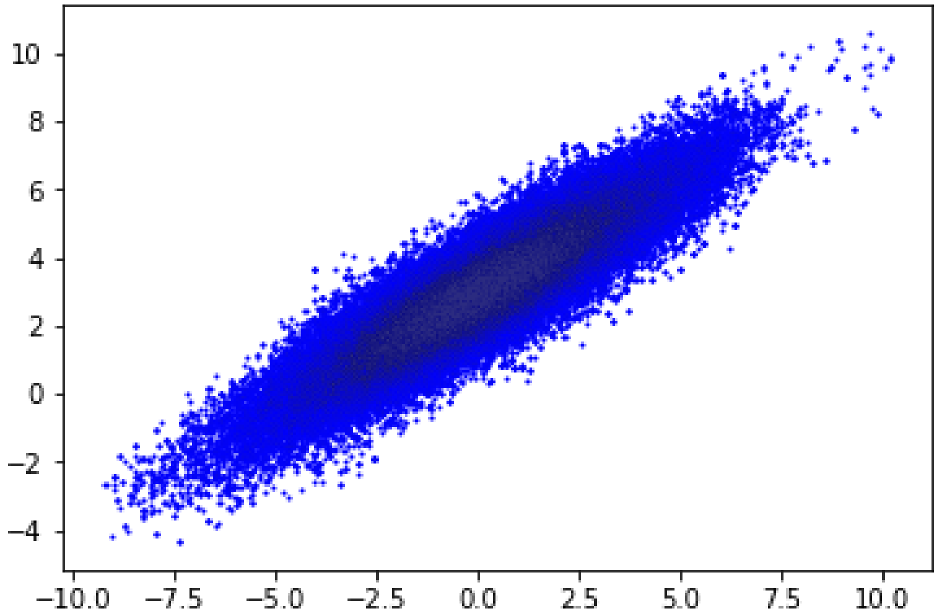


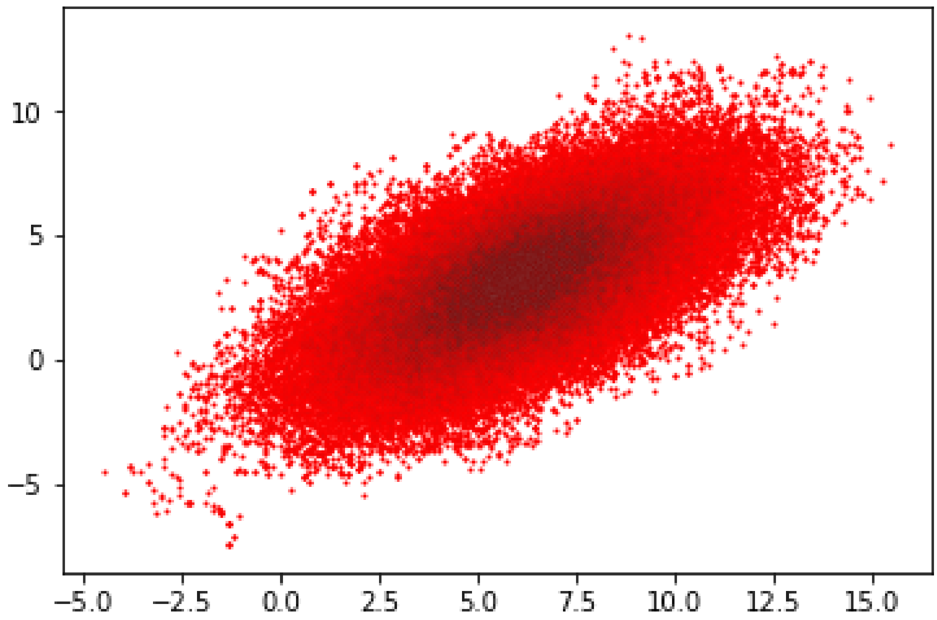


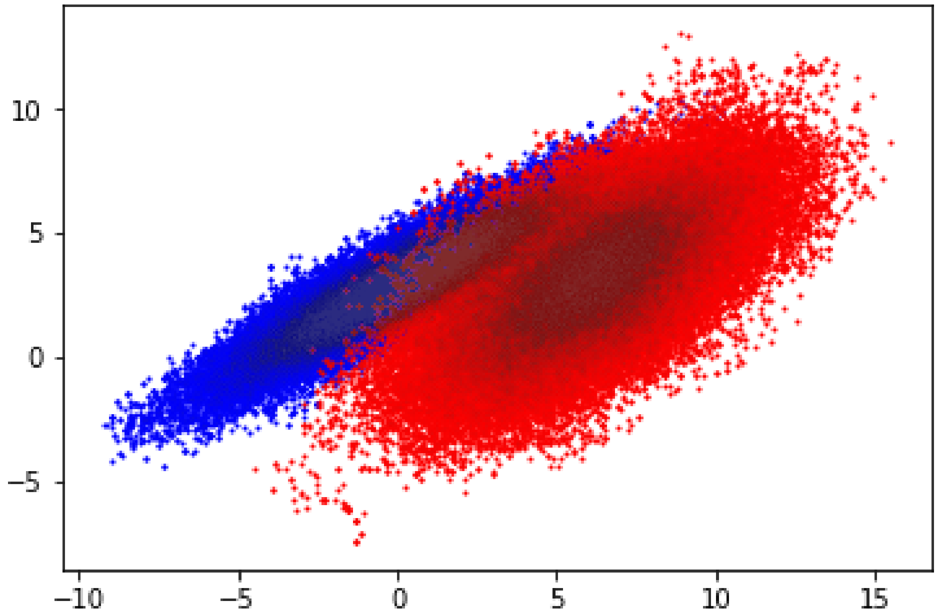
Jetzt nochmal die Scatterplots:


```
In [3]: plt.scatter(x0,y0,s=1,c='b')
        plt.scatter(x0,y0,s=1,c='grey',alpha=0.01)
```

```
plt.show()
plt.scatter(x1,y1,s=1,c='r')
plt.scatter(x1,y1,s=1,c='grey',alpha=0.01)
plt.show()
plt.scatter(x0,y0,s=1,c='b')
plt.scatter(x1,y1,s=1,c='r')
plt.scatter(x0,y0,s=1,c='grey',alpha=0.01)
plt.scatter(x1,y1,s=1,c='grey',alpha=0.01)
plt.show()
```







Aufgabenteil c)

Zunächst werden 100 Stichprobenwerte genommen und davon Mittelwerte und Standardabweichungen berechnet.

```
In [4]: stichx0=[]
        stichy0=[]
        for i in range(0,int(len(w0)/1000)):
            stichx0.append(x0[i*1000])
            stichy0.append(y0[i*1000])
        print('ux0 = ', np.mean(stichx0),'pm',np.std(stichx0))
        print('uy0 = ', np.mean(stichy0),'pm',np.std(stichy0))
        stichx1=[]
        stichy1=[]
        for i in range(0,int(len(w1)/1000)):
            stichx1.append(x1[i*1000])
            stichy1.append(y1[i*1000])
        print('ux1 = ', np.mean(stichx1),'pm',np.std(stichx1))
        print('uy1 = ', np.mean(stichy1),'pm',np.std(stichy1))
        print('uxg = ', np.mean(stichx1+stichx0),'pm',np.std(stichx1+stichx0))
        print('uyg = ', np.mean(stichy1+stichy0),'pm',np.std(stichy1+stichy0))
        ux0=np.mean(stichx0)
        fx0=np.std(stichx0)
        uy0=np.mean(stichy0)
        fy0=np.std(stichy0)
        ux1=np.mean(stichx1)
        fx1=np.std(stichx1)
        uy1=np.mean(stichy1)
        fy1=np.std(stichy1)
        uxg=np.mean(stichx0+stichx1)
        fxg=np.std(stichx0+stichx1)
        uyg=np.mean(stichy0+stichy1)
        fyg=np.std(stichy0+stichy1)
```

```
ux0 =  -0.071862882346 pm 2.1928913398
uy0 =  2.93455614082 pm 1.8274113598
ux1 =  5.88790721152 pm 2.5544379163
uy1 =  3.27233602098 pm 2.4905890783
uxg =  2.90802216459 pm 3.81401070281
uyg =  3.1034460809 pm 2.19083475628
```

Es ergeben sich (für einen der Durchläufe)

$$\mu_{x,P0} = -0.32 \qquad \sigma_{x,P0} = 2.31$$
$$\mu_{y,P0} = 2.80 \qquad \sigma_{y,P0} = 1.76$$
$$\mu_{x,P1} = 6.06 \qquad \sigma_{x,P1} = 2.45$$
$$\mu_{y,P1} = 3.53 \qquad \sigma_{y,P1} = 2.24$$
$$\mu_{x,Pges} = 2.87 \qquad \sigma_{x,Pges} = 3.98$$
$$\mu_{y,Pges} = 3.16 \qquad \sigma_{y,Pges} = 2.05$$

Anhand dieser Daten werden Covarianzmatrizen und Korrelationsfaktoren bestimmt

In [5]:
```
c00 = []
c10 = []
c11 = []
for i in range(0,int(len(w0)/1000)-1):
    c00.append((stichx0[i]-ux0)*(stichx0[i]-ux0))
    c10.append((stichx0[i]-ux0)*(stichy0[i]-uy0))
    c11.append((stichy0[i]-uy0)*(stichy0[i]-uy0))

c0=[[np.mean(c00),np.mean(c10)],[np.mean(c10),np.mean(c11)]]
print('c0=',c0)

c00 = []
c10 = []
c11 = []
for i in range(0,int(len(w1)/1000)-1):
    c00.append((stichx1[i]-ux)*(stichx1[i]-ux))
    c10.append((stichx1[i]-ux)*(stichy1[i]-uy))
    c11.append((stichy1[i]-uy)*(stichy1[i]-uy))

c1=[[np.mean(c00),np.mean(c10)],[np.mean(c10),np.mean(c11)]]
print('c1=',c1)

c00 = []
c10 = []
c11 = []
stichxg=stichx0+stichx1
stichyg=stichy0+stichy1
for i in range(0,int(len(w1)/1000)-1):
    c00.append((stichxg[i]-uxg)*(stichxg[i]-uxg))
    c10.append((stichxg[i]-uxg)*(stichyg[i]-uyg))
    c11.append((stichyg[i]-uyg)*(stichyg[i]-uyg))

cg=[[np.mean(c00),np.mean(c10)],[np.mean(c10),np.mean(c11)]]
print('cg=',cg)
print('rho0=', c0[0][1]/(fx0*fy0))
print('rho1=', c1[0][1]/(fx1*fy1))
print('rhog=', cg[0][1]/(fxg*fyg))
```

```
c0= [[4.7855524571097172, 3.4396172510399805], [3.4396172510399805, 3.2240371295273635]]
c1= [[6.5542692401490257, 3.8143783864070109], [3.8143783864070109, 6.112521538954689]]
cg= [[13.825759672039633, 4.0630916995607738], [4.0630916995607738, 3.2656706793563774]]
rho0= 0.85833473072
rho1= 0.599551289821
rhog= 0.486256135738
```

Für einen Durchlauf ergeben sich

$$cov_{P0} = \begin{pmatrix} 5.35 & 3.59 \\ 3.59 & 3.08 \end{pmatrix} \qquad \rho_{P0} = 0.89$$

$$cov_{P1} = \begin{pmatrix} 5.99 & 3.83 \\ 3.83 & 5.21 \end{pmatrix} \qquad \rho_{P1} = 0.7$$

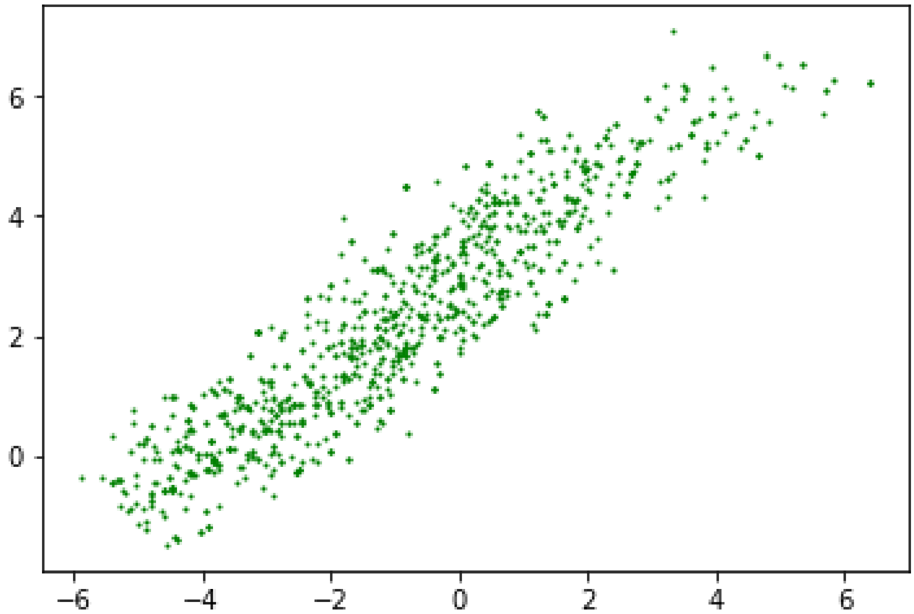
$$cov_{Pges} = \begin{pmatrix} 15.54 & 4.76 \\ 4.76 & 3.21 \end{pmatrix} \qquad \rho_{Pges} = 0.58$$

Es ist zu beachten, dass aufgrund der Nutzung des Metropolis Algorithmus zur Erstellung der Werte kein Random-Seed gesetzt werden konnte, sodass die Werte in den Markdowns lediglich ähnlich zu den direkten Python Ausgaben sind.

Aufgabenteil d)

Zunächst wir die letzte Popultion $P_{0,\text{klein}}$erzeugt:

```
In [6]: sx0=3.5
        sy0=2.6
        ux0=0
        uy0=3
        rho0=0.9

        time0k, w0k = Metro0(1,(0,3),1000)
        x0k=[]
        y0k=[]

        for i in time0k:
            x0k.append(w0k[i][0])
            y0k.append(w0k[i][1])
        plt.scatter(x0k,y0k,s=1,c='g')
        plt.show()
```

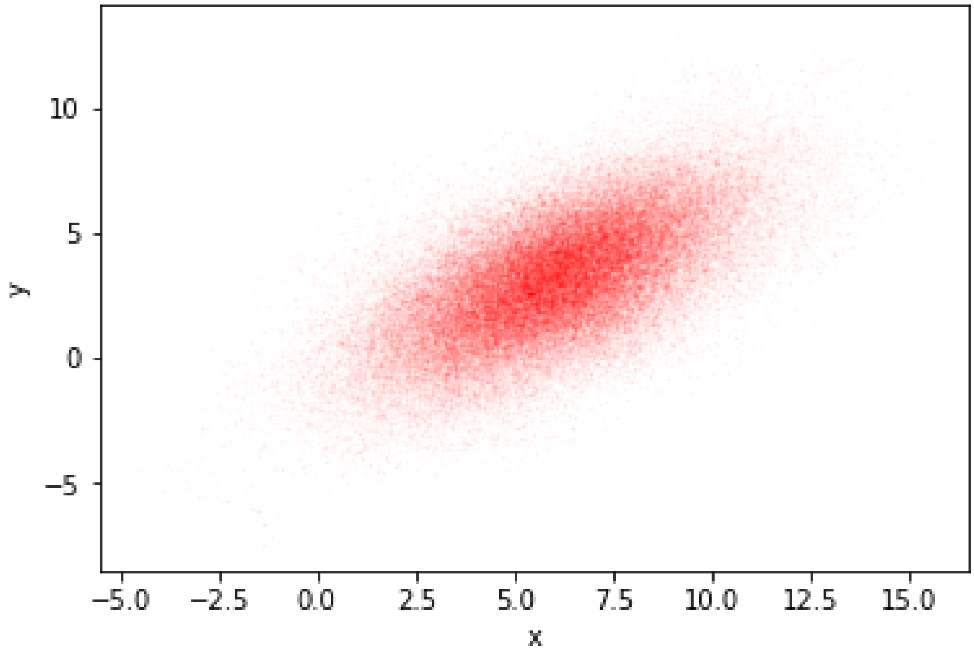


Dann werden die Verteilungen mithilfe von Pandas abgespeichert

```
In [7]: import pandas as pd
        signal0 = pd.DataFrame({
            'x': x0,
            'y': y0,
        })
        signal1 = pd.DataFrame({
            'x': x1,
            'y': y1,
        })
        signal0k = pd.DataFrame({
            'x': x0k,
            'y': y0k,
        })
        signal0.to_hdf('data.hdf5', key='p0')
        signal1.to_hdf('data.hdf5', key='p1')
        signal0k.to_hdf('data.hdf5', key='p0k')
```

Hier habe ich das auslesen probiert

```
In [8]: probe = pd.read_hdf('data.hdf5', key='p1')
        probe.plot.scatter('x','y',s=1,c='r',alpha=0.01)
        plt.show()
```


10
5
0
−5
y
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

A11

$x_0: \begin{pmatrix} x \\ y \end{pmatrix} \in \left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1,5 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix} \right\}$

$x_1: \begin{pmatrix} x \\ y \end{pmatrix} \in \left\{ \begin{pmatrix} 1,5 \\ 1 \end{pmatrix}, \begin{pmatrix} 2,5 \\ 1 \end{pmatrix}, \begin{pmatrix} 3,5 \\ 1 \end{pmatrix}, \begin{pmatrix} 2,5 \\ 2 \end{pmatrix}, \begin{pmatrix} 3,5 \\ 2 \end{pmatrix}, \begin{pmatrix} 4,5 \\ 2 \end{pmatrix} \right\}$

a) $\mu_0 = \frac{1}{6} \begin{pmatrix} \sum_{i=1}^{6} x_i \\ \sum_{i=1}^{6} y_i \end{pmatrix} = \frac{1}{6} \begin{pmatrix} 11,5 \\ 12 \end{pmatrix} = \begin{pmatrix} 23/12 \\ 2 \end{pmatrix} \approx \begin{pmatrix} 1,916 \\ 2 \end{pmatrix}$

$\mu_1 = \frac{1}{6} \begin{pmatrix} \sum_{i=1}^{6} x_i \\ \sum_{i=1}^{6} y_i \end{pmatrix} = \frac{1}{6} \begin{pmatrix} 18 \\ 9 \end{pmatrix} = \begin{pmatrix} 3 \\ 1,5 \end{pmatrix}$

$S_i = \sum_{i=1}^{n} (\vec{x}_i - \vec{\mu}_j)(\vec{x}_i - \vec{\mu}_j)^T$

$S_0 = \begin{pmatrix} -11/12 \\ -1 \end{pmatrix} \begin{pmatrix} -11/12 & -1 \end{pmatrix} + \begin{pmatrix} -5/12 \\ 0 \end{pmatrix} \begin{pmatrix} -5/12 & 0 \end{pmatrix} + \begin{pmatrix} 1/12 \\ -1 \end{pmatrix} \begin{pmatrix} 1/12 & -1 \end{pmatrix} + \begin{pmatrix} 1/12 \\ 0 \end{pmatrix} \begin{pmatrix} 1/12 & 0 \end{pmatrix}$

$+ \begin{pmatrix} 1/12 \\ 1 \end{pmatrix} \begin{pmatrix} 1/12 & 1 \end{pmatrix} + \begin{pmatrix} 13/12 \\ 1 \end{pmatrix} \begin{pmatrix} 13/12 & 1 \end{pmatrix}$

$= \begin{pmatrix} 53/24 & 2 \\ 2 & 4 \end{pmatrix}$

$S_1 = \begin{pmatrix} -1,5 \\ -0,5 \end{pmatrix} \begin{pmatrix} -1,5 & -0,5 \end{pmatrix} + \begin{pmatrix} -0,5 \\ -0,5 \end{pmatrix} \begin{pmatrix} -0,5 & -0,5 \end{pmatrix} + \begin{pmatrix} 0,5 \\ -0,5 \end{pmatrix} \begin{pmatrix} 0,5 & -0,5 \end{pmatrix}$

$+ \begin{pmatrix} -0,5 \\ 0,5 \end{pmatrix} \begin{pmatrix} -0,5 & 0,5 \end{pmatrix} + \begin{pmatrix} 0,5 \\ 0,5 \end{pmatrix} \begin{pmatrix} 0,5 & 0,5 \end{pmatrix} + \begin{pmatrix} 1,5 \\ 0,5 \end{pmatrix} \begin{pmatrix} 1,5 & 0,5 \end{pmatrix} = \begin{pmatrix} 5,5 & 1,5 \\ 1,5 & 1,5 \end{pmatrix}$

$S_w = S_0 + S_1 = \begin{pmatrix} 185/24 & 3,5 \\ 3,5 & 5,5 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

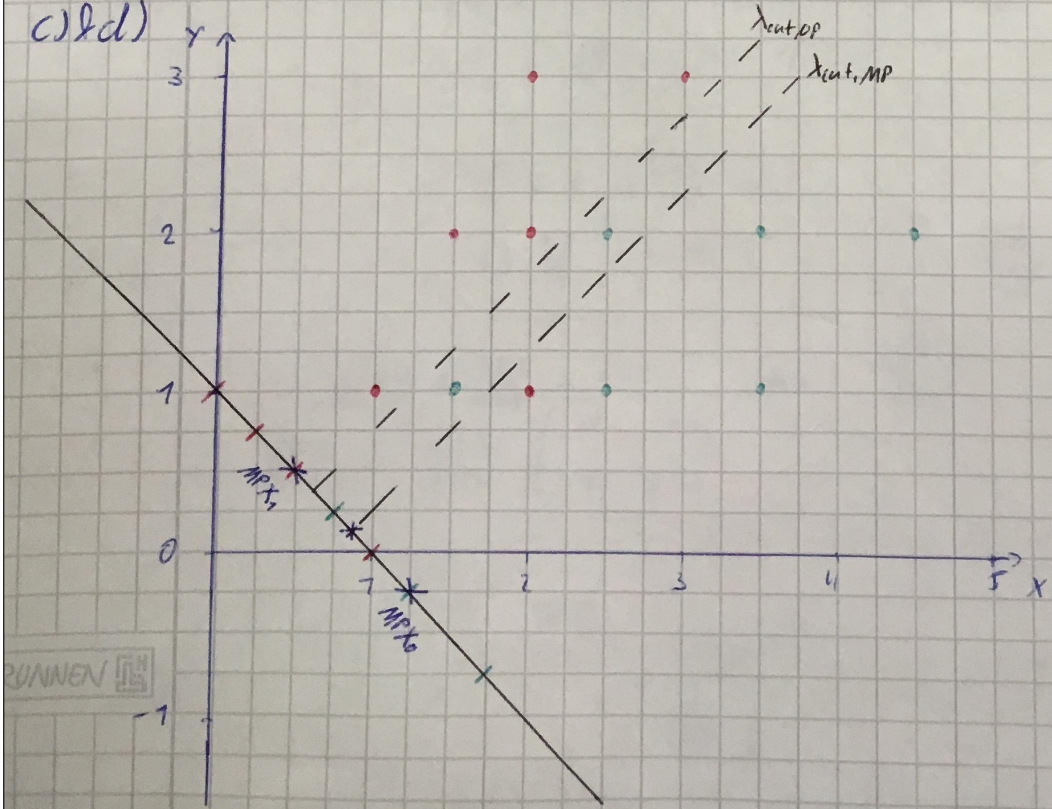
b) $\vec{\lambda}^* = S_w^{-1}(\mu_0 - \mu_1) \qquad (\mu_0 - \mu_1) = \begin{pmatrix} -13/12 \\ 1/2 \end{pmatrix}$

$S_w^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \frac{1}{\frac{2035}{48} - \frac{49}{4}} \begin{pmatrix} 5,5 & -3,5 \\ -3,5 & 185/24 \end{pmatrix}$

$= \frac{48}{1447} \begin{pmatrix} 5,5 & -3,5 \\ -3,5 & 185/24 \end{pmatrix}$

$\vec{\lambda}^* = \begin{pmatrix} -\frac{370}{1447} \\ \frac{367}{1447} \end{pmatrix}$

c) & d)



x_0 $\quad$ x_1

$\lambda = 1 + \frac{\lambda_x}{\lambda_y} x$

$= 1 - \frac{370/1447}{367/1447} x$

$= 1 - \frac{370}{367} x$

$\lambda = 1 - 1{,}008x$

Projektion x_0

Projektion x_1

MP: Mittelpunkt der Projektion

$\lambda_{cut,op}$: optisch gewählt für möglichst wenig falsche Punkte

$\lambda_{cut,MP}$: Der Mittelpunkt zwischen biden Mittelpunkten auf $\vec{\lambda}$

e) Reinheit: $R = \frac{tp}{tp+fp}$

tp: true positive
fp: false positive

Effizienz: $E = \frac{tp}{tp+fn}$

fn: false negative

Mit $\lambda_{cut,op}$:

Aus sicht von x_0: $R = \frac{6}{6+1} = \frac{6}{7}$ $\quad E = \frac{6}{6+0} = 1$

Aus Sicht von x_1: $R = \frac{5}{5+0} = 1$ $\quad E = \frac{5}{5+1} = \frac{5}{6}$

Mit $\lambda_{cut,MP}$:

Aus sicht von x_0: $R = \frac{4}{4+1} = \frac{4}{5}$ $\quad E = \frac{4}{4+2} = \frac{4}{6}$

Aus Sicht von x_1: $R = \frac{5}{5+2} = \frac{5}{7}$ $\quad E = \frac{5}{5+1} = \frac{5}{6}$

Aufgabe12

November 15, 2018

Aufgabe 12

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        from numpy.linalg import inv
```

```
In [2]: P0 = pd.read_hdf('zwei_populationen.h5', key='P_0_10000')
        P1 = pd.read_hdf('zwei_populationen.h5', key='P_1')
```

```
In [3]: P0_x = P0['x']
        P0_y = P0['y']
        P1_x = P1['x']
        P1_y = P1['y']
```

Teil a)

```
In [4]: mu_P0_x = np.mean(P0_x)
        mu_P0_y = np.mean(P0_y)
        mu_P1_x = np.mean(P1_x)
        mu_P1_y = np.mean(P1_y)
        print("mu_P0")
        print(mu_P0_x, mu_P0_y)
        print("mu_P1")
        print(mu_P1_x, mu_P1_y)
```

```
mu_P0
-0.02743075223963595 2.9799446468232453
mu_P1
5.986448205069931 3.085282896934817
```

Teil b)

```
In [5]: V_P0 = np.cov(P0_x - mu_P0_x, P0_y - mu_P0_y)
        V_P1 = np.cov(P1_x - mu_P1_x, P1_y - mu_P1_y)
        V_P0_P1 = V_P0 + V_P1
        mat_mu = np.mat(((mu_P0_x - mu_P1_x), (mu_P0_y - mu_P1_y))).T
```

```
        V_B = mat_mu*mat_mu.T
        print('V_P0', V_P0)
        print('V_P1', V_P1)
        print('V_P0_P1', V_P0_P1)
        print('V_B', V_B)
```

```
V_P0 [[ 12.20892862   8.15840984]
 [  8.15840984   6.72286327]]
V_P1 [[ 12.35218537   7.41075614]
 [  7.41075614   5.47731503]]
V_P0_P1 [[ 24.56111399  15.56916598]
 [ 15.56916598  12.20017829]]
V_B [[  3.61667401e+01   6.33491486e-01]
 [  6.33491486e-01   1.10961469e-02]]
```

Teil c

```
In [6]: lambda1 = inv(V_P0_P1)*mat_mu
        print(lambda1)
        # ay= bx <=> y = b/a x
        a = np.round(-float(lambda1[0]/lambda1[1]))
        print('Geradengleichung: y =', np.round(float(-lambda1[0]/lambda1[1]), 4), 'x')

        #S_W_S_B = inv(V_P0_P1)*V_B
        #print(S_W_S_B)
        #Det = (S_W_S_B[0,0]
        #print(Det)
```

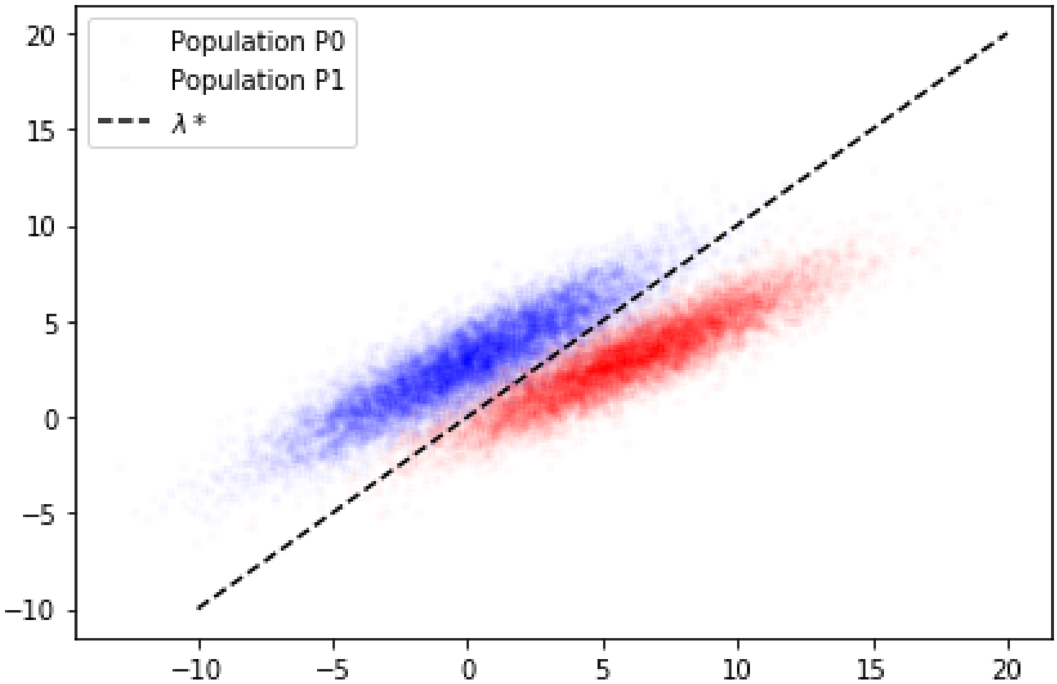
```
[[-1.2529094 ]
 [ 1.59025678]]
Geradengleichung: y = 0.7879 x
```

```
In [7]: #mat_mu = np.mat(((-5.4), (-4))).T
        #V_B = mat_mu*mat_mu.T
        #V_P0_P1 = np.mat(((13.2, -2.2), (-2.2, 26.4)))
        #lambda1 = inv(V_P0_P1)*mat_mu
        #print(lambda1)
        #a =[1,2,3]
        #b= [5,0,4]
        #print(min(min(a),min(b)))
```

```
In [8]: def lin(x,a):
            return x*a
```

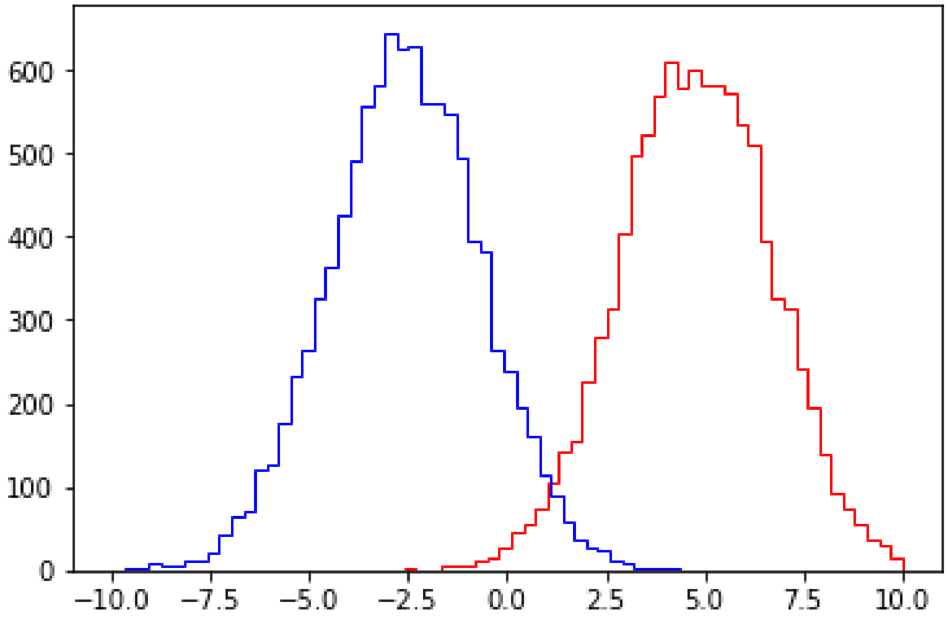
```
In [9]: x = np.linspace(-10,20)
        plt.plot(P0_x,P0_y,'b.',alpha=0.01,label='Population P0')
        plt.plot(P1_x,P1_y,'r.',alpha=0.01,label='Population P1')
```

```
plt.plot(x, lin(x,a), 'k--',label=r'$\lambda *$')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



Teil d)

```
In [10]: datahist_P0 = lambda1.T * np.mat(((P0_x),(P0_y)))
         datahist_P1 = lambda1.T * np.mat(((P1_x),(P1_y)))
         #print(datahist_P0)
         plt.hist(datahist_P0.T,bins=50,range=(-5,10),color='r',histtype='step',label='Signal')
         plt.hist(datahist_P1.T,bins=50,range=(-10,5),color='b',histtype='step', label='Untergru
         plt.show()
```



Teil e)-g)

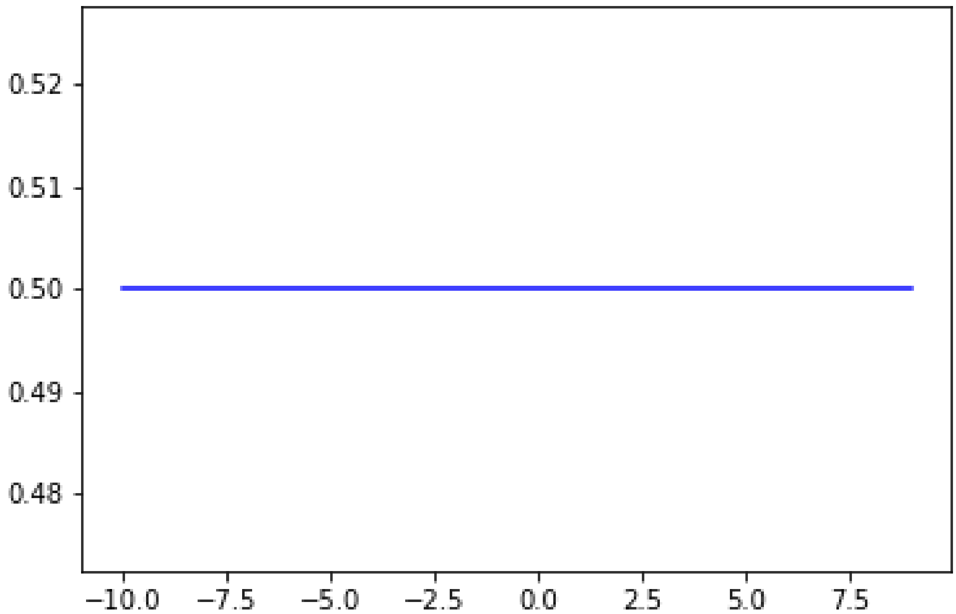
In [11]: Ich habe es nicht hinbekommen λ_{cut} als Fkt. von λ bzw. die jeweilig
Ich habe den Rest so geschrieben, dass wenn man diesen Fehler korrigiert direkt weiter

In [12]:
```
# Reinheit
reinh =[]
#Effizenz
effiz =[]
value = []
# Signal zu Hintergrund
S_B = []
# Signifikanz
sqrt_S_B =[]
tp = 0
eff = 0
fp = 0
sq_s_b = 0
fn = 0
i = 0
# Hier muss ein Fehler sein
for x_value in range(-10,10):
    while i < 10000:
        if x_value < datahist_P0.T[i]:
            tp += 1
        if x_value < datahist_P1.T[i]:
```

```
                fp += 1
            else:
                fn += 1
            i += 1
        rein = tp / (tp + fp)
        eff = tp / (tp + fn)
        s_b = tp / fp
        sq_s_b = tp / (np.sqrt(tp + fp))
        reinh.append(rein)
        effiz.append(eff)
        S_B.append(s_b)
        sqrt_S_B.append(sq_s_b)
        value.append(x_value)
```

```
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [13]: #x_value = np.linspace(4, 6)
         #x_value = np.linspace(min(min(datahist_P0),min(datahist_P1)),max(max(datahist_P0),max(
         plt.plot(value, reinh,'b-',label='Reinheit')
         plt.plot(value, effiz,'g-',label='Effizenz')
         plt.plot(value, S_B,'r-',label='Signal zu Hintergrund')
         plt.plot(value, sqrt_S_B,'g-',label='Signifikanz')
         #plt.legend(loc='best')
         #plt.tight_layout()
         plt.show()
```



Aufgabe12h

November 15, 2018

Aufgabe 12h

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        from numpy.linalg import inv
```

```
In [2]: P0 = pd.read_hdf('zwei_populationen.h5', key='P_0_1000')
        P1 = pd.read_hdf('zwei_populationen.h5', key='P_1')
```

```
In [3]: P0_x = P0['x']
        P0_y = P0['y']
        P1_x = P1['x']
        P1_y = P1['y']
```

```
In [4]: mu_P0_x = np.mean(P0_x)
        mu_P0_y = np.mean(P0_y)
        mu_P1_x = np.mean(P1_x)
        mu_P1_y = np.mean(P1_y)
        print("mu_P0")
        print(mu_P0_x, mu_P0_y)
        print("mu_P1")
        print(mu_P1_x, mu_P1_y)
```

```
mu_P0
-0.09576791285523094 2.878846798570514
mu_P1
5.986448205069931 3.085282896934817
```

```
In [5]: V_P0 = np.cov(P0_x - mu_P0_x, P0_y - mu_P0_y)
        V_P1 = np.cov(P1_x - mu_P1_x, P1_y - mu_P1_y)
        V_P0_P1 = V_P0 + V_P1
        mat_mu = np.mat(((mu_P0_x - mu_P1_x), (mu_P0_y - mu_P1_y))).T
        V_B = mat_mu*mat_mu.T
        print('V_P0', V_P0)
        print('V_P1', V_P1)
        print('V_P0_P1', V_P0_P1)
        print('V_B', V_B)
```

```
V_P0 [[ 12.23612255   8.16049883]
 [  8.16049883   6.75819008]]
V_P1 [[ 12.35218537   7.41075614]
 [  7.41075614   5.47731503]]
V_P0_P1 [[ 24.58830792  15.57125497]
 [ 15.57125497  12.2355051 ]]
V_B [[ 36.99335291   1.25558896]
 [  1.25558896   0.04261586]]
```

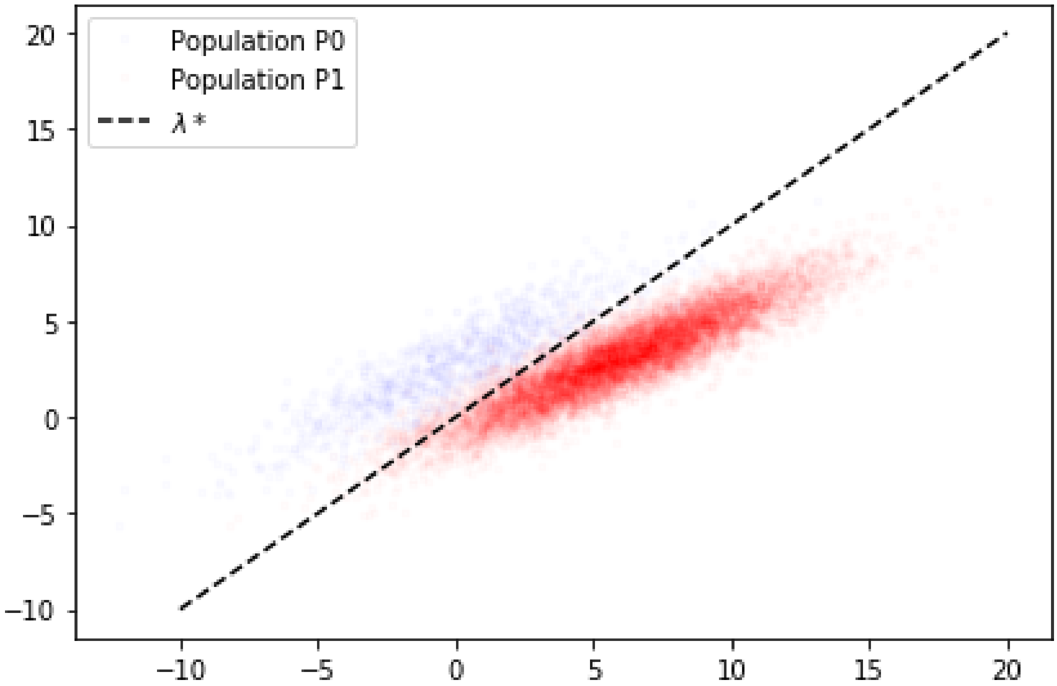
```
In [6]: lambda1 = inv(V_P0_P1)*mat_mu
        print(lambda1)
        # ay= bx <=> y = b/a x
        a = np.round(-float(lambda1[0]/lambda1[1]))
        print('Geradengleichung: y =', np.round(float(-lambda1[0]/lambda1[1]), 4), 'x')

        #S_W_S_B = inv(V_P0_P1)*V_B
        #print(S_W_S_B)
        #Det = (S_W_S_B[0,0]
        #print(Det)
```

```
[[-1.21953973]
 [ 1.53514938]]
Geradengleichung: y = 0.7944 x
```

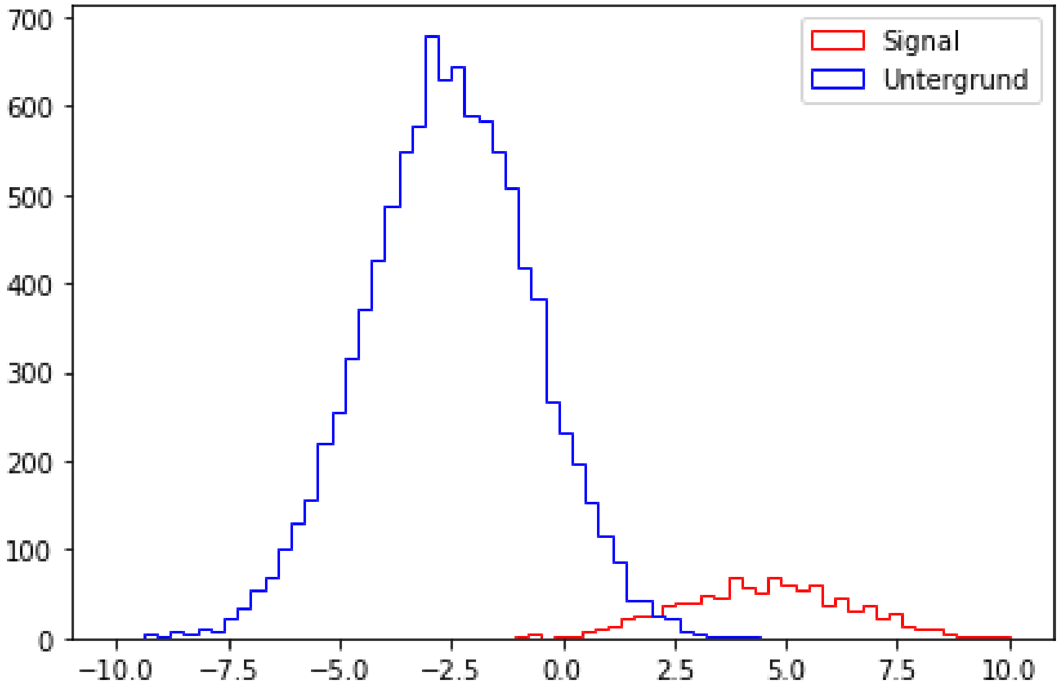
```
In [7]: #mat_mu = np.mat(((-5.4), (-4))).T
        #V_B = mat_mu*mat_mu.T
        #V_P0_P1 = np.mat(((13.2, -2.2), (-2.2, 26.4)))
        #lambda1 = inv(V_P0_P1)*mat_mu
        #print(lambda1)
        #a =[1,2,3]
        #b= [5,0,4]
        #print(min(min(a),min(b)))
```

```
In [8]: def lin(x,a):
            return x*a
```

```
In [9]: x = np.linspace(-10,20)
        plt.plot(P0_x,P0_y,'b.',alpha=0.01,label='Population P0')
        plt.plot(P1_x,P1_y,'r.',alpha=0.01,label='Population P1')
        plt.plot(x, lin(x,a), 'k--',label=r'$\lambda *$')
        plt.legend(loc='best')
        plt.tight_layout()
        plt.show()
```




```
In [10]: datahist_P0 = lambda1.T * np.mat(((P0_x),(P0_y)))
         datahist_P1 = lambda1.T * np.mat(((P1_x),(P1_y)))
         #print(datahist_P0)
         plt.hist(datahist_P0.T,bins=50,range=(-5,10),color='r',histtype='step',label='Signal')
         plt.hist(datahist_P1.T,bins=50,range=(-10,5),color='b',histtype='step',label='Untergrun
         plt.legend(loc='best')
         plt.tight_layout()
         plt.show()
```



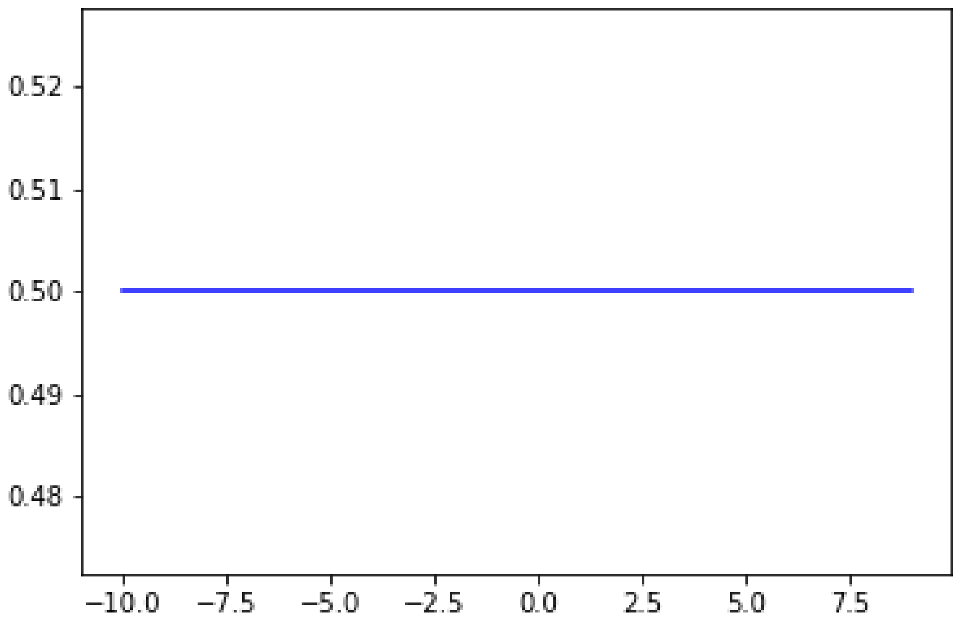
Wie schon bei den vorherigen Teil habe ich es nicht hinbekommen λ_{cut} als Fkt. von λ bzw. die jeweiligen Fkt. in Abhängigkeit von λ_{cut} darzustellen. Ich habe den Rest so geschrieben, dass wenn man diesen Fehler korrigiert direkt weiter machen kann.

```
In [12]: # Reinheit
         reinh =[]
         #Effizenz
         effiz =[]
         value = []
         # Signal zu Hintergrund
         S_B = []
         # Signifikanz
         sqrt_S_B =[]
         tp = 0
         eff = 0
         fp = 0
         sq_s_b = 0
         fn = 0
         i = 0
         # Hier muss ein Fehler sein
         for x_value in range(-10,10):
             while i < 1000:
                 if x_value < datahist_P0.T[i]:
                     tp += 1
                 if x_value < datahist_P1.T[i]:
```

```
                fp += 1
            else:
                fn += 1
            i += 1
        rein = tp / (tp + fp)
        eff = tp / (tp + fn)
        s_b = tp / fp
        sq_s_b = tp / (np.sqrt(tp + fp))
        reinh.append(rein)
        effiz.append(eff)
        S_B.append(s_b)
        sqrt_S_B.append(sq_s_b)
        value.append(x_value)
```

```
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [13]: #x_value = np.linspace(4, 6)
         #x_value = np.linspace(min(min(datahist_P0),min(datahist_P1)),max(max(datahist_P0),max(
         plt.plot(value, reinh,'b-',label='Reinheit')
         plt.plot(value, effiz,'g-',label='Effizenz')
         plt.plot(value, S_B,'r-',label='Signal zu Hintergrund')
         plt.plot(value, sqrt_S_B,'g-',label='Signifikanz')
         #plt.legend(loc='best')
         #plt.tight_layout()
         plt.show()
```



Trotz der 10 fach kleineren Zahl an "Signal"-Werten sind die Ergebnisse wie die Mittelwerte, Kovarianzmatrizen der beiden Verteilungen (P_0_10000 und P_0_1000) nahe zu gleich. Zum Rest kann ich leider nicht sagen :(