Study of a mixture of two probability laws

EM Algorithm

SAUCE Justine

from the teachings of Y. Offret

In [ ]:
```
import numpy as np                                  # import numpy
import numpy.random as npr                          # import random for generating random numbers
import matplotlib.pyplot as plt                     # import matplotlib.pyplot for plotting
import matplotlib.patches as mpatches               # import matplotlib.patches for more plotting options
import seaborn as sns                               # import of seaborn for a more sophisticated graphic representation
from scipy.stats import norm                        # for generating pdf : norm.pdf(x, mean, cov)
from scipy.stats import multivariate_normal         # for generating pdf : multivariate_normal.pdf(x, mean, cov)
```

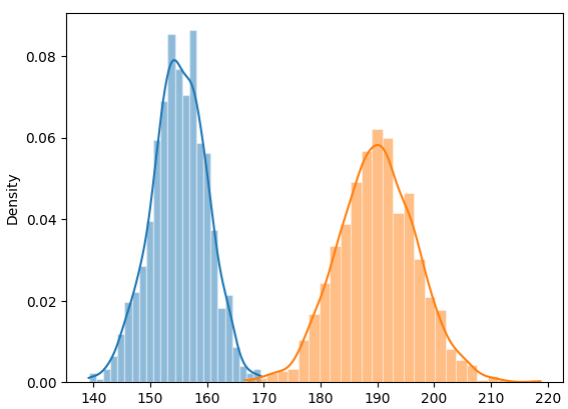
Mixture of Gaussian distributions

The objective is to find the mean and variance parameters of two samples from Gaussian distributions, mixed in a data set. To illustrate this example, let us simulate a sample of male size of law $N(190, 49)$ and another of female size of law $N(155, 25)$ and plot the distribution of these two samples.

In [ ]:
```
males = np.random.normal(190, 7, 1000)
# males = [190.57287445, 190.78204877, 192.92937929, 173.1494335, ...]
females = np.random.normal(155, 5, 1000)
# females = [164.52823261, 152.03498728, 157.57573209, 155.22649006, ...]

sns.histplot(females, label="Females", kde=True, stat="density", edgecolor=(1, 1, 1, .4))
sns.histplot(males, label="Males", kde=True, stat="density", edgecolor=(1, 1, 1, .4))
```

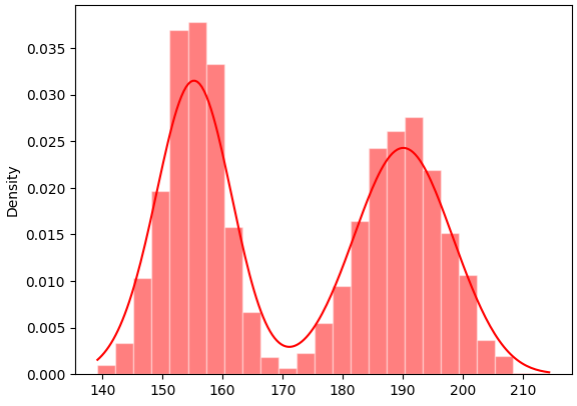
Out[ ]:
```
<AxesSubplot: ylabel='Density'>
```



Now imagine that the two are mixed, here, in red, is what we will obtain as density. The goal is to find the parameters defining the two groups.

In [ ]:
```
X  = np.concatenate((females,males))
sns.histplot(X, label="mix", bins=25, kde=True, stat="density", color="red", edgecolor=(1, 1, 1, .4))
```

Out[ ]:
```
<AxesSubplot: ylabel='Density'>
```



Definition of the problem

We model the height distribution of male and female students respectively by normal distributions $N(\mu_h, \sigma_h^2)$ and $N(\mu_f, \sigma_f^2)$. We will posit $v_h = \sigma_h^2$ and $v_f = \sigma_f^2$. Moreover, we fix p the proportion of female students so that the sex of an individual chosen at random and uniformly is modeled by a random variable S of Bernoulli distribution of parameter p. The height is modeled by a random variable X which is a mixture of the two previous normal laws in the proportions $1-p$ and p.

We will denote g_f the density for female students and g_m that for male students,

$$g_f(x) = \frac{1}{\sqrt{2\pi\sigma_f^2}} \exp^{-\frac{(x-\mu_f)^2}{2\sigma_f^2}}$$

$$g_m(x) = \frac{1}{\sqrt{2\pi\sigma_m^2}} \exp^{-\frac{(x-\mu_m)^2}{2\sigma_m^2}}$$

The density of X is therefore given by,

$$h_X(x) = pg_f(x) + (1-p)g_m(x)$$

Given that an individual has size x, the probability, as a function of Θ, that the individual is female is given by,

$$P_\Theta(S=1|X=x) = \frac{pg_f(x)}{pg_f(x) + (1-p)g_m(x)}$$

Estimation of parameters: EM algorithm

We denote $\log L_\Theta((x_1, \dots, x_n; s_1, \dots, s_n)$ the log-likelihood of an n-sample of sizes and sexes corresponding with the parameters Θ. It is given by,

$$L_\Theta(X;S) = \prod_{i=1}^n (pg_f(x_i)1_{S_i=1} + (1-p)g_m(x_i)1_{S_i=0})$$

$$\begin{aligned}\log L_\Theta(X;S) = &\log(p)\sum_{i=1}^n 1_{S_i=1} + \log(1-p)\sum_{i=1}^n 1_{S_i=0} \\ &-\frac{1}{2}\log(2\pi\sigma_f^2)\sum_{i=1}^n 1_{S_i=1} - \sum_{i=1}^n \left(-\frac{(x_i-\mu_f)^2}{2\sigma_f^2}\right)1_{S_i=1} \\ &-\frac{1}{2}\log(2\pi\sigma_m^2)\sum_{i=1}^n 1_{S_i=0} - \sum_{i=1}^n \left(-\frac{(x_i-\mu_m)^2}{2\sigma_m^2}\right)1_{S_i=0}\end{aligned}$$

Expectation Phase

$$
\begin{aligned}
\mathbb{E}_{\Theta'}\left[\log L_\Theta(X;S)|X=x\right] = \log(p)\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) + \log(1-p)\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i) \\
-\frac{1}{2}\log(2\pi\sigma_f^2)\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) - \sum_{i=1}^n\left(-\frac{(x_i-\mu_f)^2}{2\sigma_f^2}\right)\mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) \\
-\frac{1}{2}\log(2\pi\sigma_m^2)\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i) - \sum_{i=1}^n\left(-\frac{(x_i-\mu_m)^2}{2\sigma_m^2}\right)\mathbb{P}_{\Theta'}(S_i=0|X_i=x_i)
\end{aligned}
$$

Maximisation Phase

▷ Estimation of the parameter p :

$$
\begin{aligned}
& \frac{\partial \mathbb{E}_{\Theta'}\left[\log L_\Theta(X;S)|X=x\right]}{\partial p} = 0 \\
\iff\ & \frac{1}{p}\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) = \frac{1}{(1-p)}\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i) \\
\iff\ & \frac{1}{p} = \frac{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i) + \sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)}{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)} \\
\iff\ & p = \frac{1}{N}\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)
\end{aligned}
$$

▷ Estimation of the parameters μ_f and μ_m *(using the symmetry of roles)*:

$$
\begin{aligned}
& \frac{\partial \mathbb{E}_{\Theta'}\left[\log L_\Theta(X;S)|X=x\right]}{\partial \mu_f} = 0 \\
\iff\ & \sum_{i=1}^n x_i\mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) = \mu_f\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) \\
\iff\ & \mu_f = \frac{\sum_{i=1}^n x_i\mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)}{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)} \\
& \mu_m = \frac{\sum_{i=1}^n x_i\mathbb{P}_{\Theta'}(S_i=0|X_i=x_i)}{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i)}
\end{aligned}
$$

▷ Estimation of the parameters σ_f^2 and σ_m^2 *(using the symmetry of roles)*:

$$
\begin{aligned}
& \frac{\partial \mathbb{E}_{\Theta'}\left[\log L_\Theta(X;S)|X=x\right]}{\partial \sigma_f^2} = 0 \\
\iff\ & \sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) = \sum_{i=1}^n \frac{(x_i-\mu_f)^2}{\sigma_f^2}\mathbb{P}_{\Theta'}(S_i=1|X_i=x_i) \\
\iff\ & \sigma_f^2 = \frac{\sum_{i=1}^n (x_i-\mu_f)^2\mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)}{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=1|X_i=x_i)} \\
& \sigma_m^2 = \frac{\sum_{i=1}^n (x_i-\mu_m)^2\mathbb{P}_{\Theta'}(S_i=0|X_i=x_i)}{\sum_{i=1}^n \mathbb{P}_{\Theta'}(S_i=0|X_i=x_i)}
\end{aligned}
$$

Implementation of EM algorithm

In [ ]:
```
def EM_Gauss(X, p, mu1, std1, mu2, std2, N):

    for i in range (N):

        # Normal density
        p0 = (1-p) * norm.pdf(X, loc=mu2, scale=std2)
        p1 = p * norm.pdf(X, loc=mu1, scale=std1)

        # Probabilities
        P0=p0/(p1+p0)
        P1=p1/(p1+p0)

        P0_Z = []; P0_Z2 = []
        P1_Z = []; P1_Z2 = []

        for j in range(len(X)):

            # udpates for mu1
            P0_Z.append(X[j]*P0[j])

            # updates for mu2
            P1_Z.append(X[j]*P1[j])

        # Mean updates
        mu1 = np.sum(P1_Z)/np.sum(P1)
        mu2 = np.sum(P0_Z)/np.sum(P0)

        for k in range(len(X)):

            # updates for var1
            P1_Z2.append(((X[k]-mu1)**2)*P1[k])

            # updates for var2
            P0_Z2.append(((X[k]-mu2)**2)*P0[k])


        # Variance updates
        var1 = np.sum(P1_Z2)/np.sum(P1)
        var2 = np.sum(P0_Z2)/np.sum(P0)

        # Standard deviation updates
        std1 = np.sqrt(var1)
        std2 = np.sqrt(var2)

        # p udpate
        p=np.mean(P1)

    return p, mu1, std1, mu2, std2
```

We can try it on the previously simulated data.

In [ ]:
```
# Estimated parameters
F_mean = 158; F_std = 6
M_mean = 192; M_std = 8

# Number of simulations
N = 100

# Allocation probability
p = 0.5

# Call
p_EM, F_mean_EM, F_std_EM, M_mean_EM, M_std_EM = EM_Gauss(X, p, F_mean, F_std, M_mean, M_std, N)

# Results
print(f'The parameter p is equal to {round(p_EM,2)}.')
print(f'The average of the female group is {round(F_mean_EM,2)}.')
print(f'The standard deviation of the female group is {round(F_std_EM,2)}.')
print(f'The average of the male group is {round(M_mean_EM,2)}.')
print(f'The standard deviation of the male group is {round(M_std_EM,2)}.')
```

```
The parameter p is equal to 0.5.
The average of the female group is 155.07.
The standard deviation of the female group is 4.92.
The average of the male group is 190.0.
The standard deviation of the male group is 7.08.
```

Data Import

We will now try it on a dataset containing real data about sizes (cm). We want to try to differentiate two groups based on gender.

In [ ]:
```
size=[]
with open("Data.txt","r") as file:
    for l in file:
        T=l.split("\n")
        size.append(float(T[0]))

# Overview
print(f'Overview of the first observations : {size[:5]}')
```

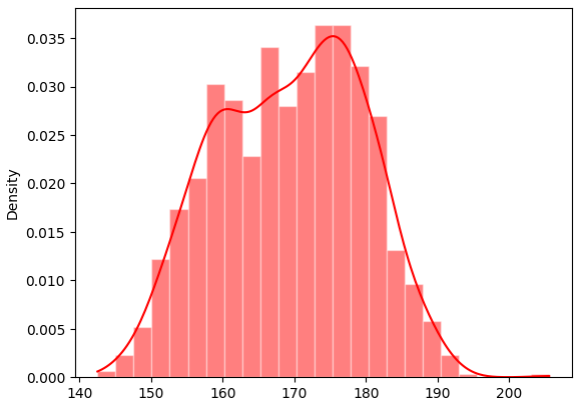
```
Overview of the first observations : [149.70338059739308, 147.9459325577882, 184.97576608695815, 150.421849064984, 177.3682984471994]
```

Data distribution

In [ ]:
```python
sns.histplot(size, label="mix", bins=25, kde=True, stat="density", color="red", edgecolor=(1, 1, 1, .4))

print(f'Mean : {np.mean(size)}')
print(f'Standard deviation : {np.std(size)}')
```

```
Mean : 169.3545332329132
Standard deviation : 10.1553787156645
```



In [ ]:
```python
# Estimated parameters
F_mean = 158; F_std = 4
M_mean = 175; M_std = 5

# Number of simulations
N = 100

# Allocation probability
p = 0.5

# Call
p_EM, F_mean_EM, F_std_EM, M_mean_EM, M_std_EM = EM_Gauss(size, p, F_mean, F_std, M_mean, M_std, N)

# Results
print(f'The parameter p is equal to {round(p_EM,2)}.')
print(f'The average of the female group is {round(F_mean_EM,2)}.')
print(f'The standard deviation of the female group is {round(F_std_EM,2)}.')
print(f'The average of the male group is {round(M_mean_EM,2)}.')
print(f'The standard deviation of the male group is {round(M_std_EM,2)}.')
```

```
The parameter p is equal to 0.38.
The average of the female group is 159.27.
The standard deviation of the female group is 5.73.
The average of the male group is 175.53.
The standard deviation of the male group is 6.76.
```

Mixture of 2D Gaussian distributions

We will note p the mixing parameter of two Gaussian vectors whose respective laws are $N(m_1, K_1)$ and $N(m_2, K_2)$ with, $m_1 = (x_1, y_1)$ and $m_2 = (x_2, y_2)$ the two means, and

$$K_1 = \begin{pmatrix} \sigma_1^2 & c_1 \\ c_1 & \tau_1^2 \end{pmatrix}, \quad K_2 = \begin{pmatrix} \sigma_2^2 & c_2 \\ c_2 & \tau_2^2 \end{pmatrix}$$

their covariance matrix.

Parameters will be estimated as follows :

▷ Estimation of the parameter p :

$$p = \frac{1}{N} \sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))$$

▷ Estimation of the parameters $m_1 = (x_1, y_1)$ and $m_2 = (x_2, y_2)$ *(using the symmetry of roles)*:

$$x_1 = \frac{\sum_{i=1}^{n} u_i \mathbb{P}_{\Theta'}(S_i = 1 | X_i = x_i)}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}$$

$$y_1 = \frac{\sum_{i=1}^{n} v_i \mathbb{P}_{\Theta'}(S_i = 1 | X_i = x_i)}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}$$

$$x_2 = \frac{\sum_{i=1}^{n} u_i \mathbb{P}_{\Theta'}(S_i = 0 | X_i = x_i)}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

$$y_2 = \frac{\sum_{i=1}^{n} v_i \mathbb{P}_{\Theta'}(S_i = 0 | X_i = x_i)}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

▷ Estimation of the parameters K_1 and K_2 *(using the symmetry of roles)*:

$$\sigma_1^2 = \frac{\sum_{i=1}^{n} (u_i - x_1)^2 \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}$$

$$c_1 = \frac{\sum_{i=1}^{n} (u_i - x_1)(v_i - y_1) \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

$$\tau_1^2 = \frac{\sum_{i=1}^{n} (u_i - y_1)^2 \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

$$\sigma_2^2 = \frac{\sum_{i=1}^{n} (v_i - x_2)^2 \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 1 | Z_i = (u_i, v_i))}$$

$$c_2 = \frac{\sum_{i=1}^{n} (u_i - x_2)(v_i - y_2) \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

$$\tau_2^2 = \frac{\sum_{i=1}^{n} (v_i - y_2)^2 \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}{\sum_{i=1}^{n} \mathbb{P}_{\Theta'}(S_i = 0 | Z_i = (u_i, v_i))}$$

Implementation of EM algorithm 2D

In [ ]:
```python
def EM_Gauss_2D(Z, p, m1, K1, m2, K2, n):

    for i in range (n):

        p0 = (1-p)* multivariate_normal.pdf(Z, mean=m2, cov=K2)
        p1 = p * multivariate_normal.pdf(Z, mean=m1, cov=K1)

        # Probabilities
        P0=p0/(p1+p0)
        P1=p1/(p1+p0)

        P0_Z1 = []; P0_Z2 = []; P0_ZX = []; P0_ZY = []; Cov2 = []
        P1_Z1 = []; P1_Z2 = []; P1_ZX = []; P1_ZY = []; Cov1 = []

        for j in range(len(Z)):

            # udpates for m1
            P0_Z1.append(Z[j][0]*P0[j]) # ui
            P0_Z2.append(Z[j][1]*P0[j]) # vi

            # updates for m2
            P1_Z1.append(Z[j][0]*P1[j])
            P1_Z2.append(Z[j][1]*P1[j])

        # Mean updates
        x1=np.sum(P1_Z1)/np.sum(P1)
        y1=np.sum(P1_Z2)/np.sum(P1)
```

```
        x2=np.sum(P0_Z1)/np.sum(P0) #x_2
        y2=np.sum(P0_Z2)/np.sum(P0) #y_2

        m1 = np.array([x1,y1])
        m2 = np.array([x2,y2])

        for k in range(len(Z)):

            # updates for K1
            P1_ZX.append(((Z[k][0]-x1)**2)*P1[k])
            P1_ZY.append(((Z[k][1]-y1)**2)*P1[k])
            Cov1.append(((Z[k][0]-x1)*(Z[k][1]-y1))*P1[k])

            # updates for K2
            P0_ZX.append(((Z[k][0]-x2)**2)*P0[k])
            P0_ZY.append(((Z[k][1]-y2)**2)*P0[k])
            Cov2.append(((Z[k][0]-x2)*(Z[k][1]-y2))*P0[k])

        # Cov Matrix updates (K1)
        sigma1 = np.sum(P1_ZX)/np.sum(P1)
        c1 = np.sum(Cov1)/np.sum(P1)
        tau1 = np.sum(P1_ZY)/np.sum(P1)
        K1 = np.array([[sigma1,c1],[c1,tau1]])

        # Cov Matrix updates (K2)
        sigma2 = np.sum(P0_ZX)/np.sum(P0)
        c2 = np.sum(Cov2)/np.sum(P0)
        tau2 = np.sum(P0_ZY)/np.sum(P0)
        K2 = np.array([[sigma2,c2],[c2,tau2]])

        # p udpate
        p=np.mean(P1)

    return p, m1, K1, m2, K2
```

Data Import & Visualization

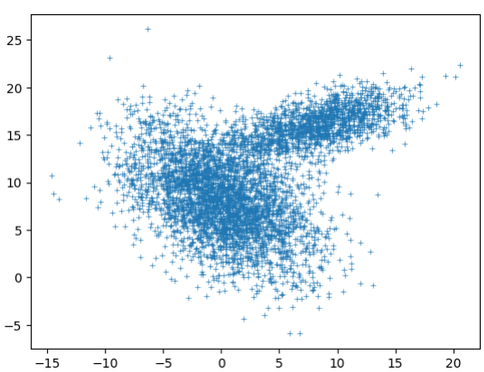
We test it on a sample of size 5000 of a mixture of two Gaussian vectors.

In [ ]:
```
# Data Import
brute=[]
with open("Brute.txt","r") as file:
    for l in file:
        T=[str(d) for d in l.split(",")]
        brute.append([float(T[0]),float(T[1])])

x = []
y = []
for i in range (len(brute)):
    x.append(brute[i][0])
    y.append(brute[i][1])

# Scatter plot
plt.scatter(x, y, marker = '+', s = 20, linewidth = 0.5)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x15be85b7e10>
```



The parameters of each group are estimated " visually " by using the scatter plot.

Note : The variance on the x-axis is represented by σ, the variance on the y-axis by τ. The covariance is given by $c = \frac{\tau}{\sigma}\sigma^2$.

In [ ]:
```
# Estimated parameters
m1_0 = np.array([0,10]); m2_0 = np.array([10,15])
K1_0 = np.array([[16,-16],[-16,25]]); K2_0 = np.array([[6.25,7],[7,12.25]])

# Number of simulations
N = 100

# Allocation probability
p = 0.5

p_EM, m1_EM, K1_EM, m2_EM, K2_EM = EM_Gauss_2D(brute, p, m1_0, K1_0, m2_0, K2_0, N)
```

In [ ]:
```
print(f'The parameter, p, is equal to {round(p_EM,2)}.')

print(f'The average, m1, is equal to {m1_EM}.')
print(f'The variance matrix, K1, equals {K1_EM}.')

print(f'The average, m2, is equal to {m2_EM}.')
print(f'The variance matrix, K1, equals {K2_EM}.')
```

```
The parameter, p, is equal to 0.71.
The average, m1, is equal to [-0.04900987  8.17062798].
The variance matrix, K1, equals [[15.66939399 -7.38106574]
 [-7.38106574 15.88545662]].
The average, m2, is equal to [ 8.199892   16.03891598].
The variance matrix, K1, equals [[14.01407641  4.23533613]
 [ 4.23533613  3.62163294]].
```

We represent the data by distinguishing the two groups based on the parameters obtained with the EM algorithm.

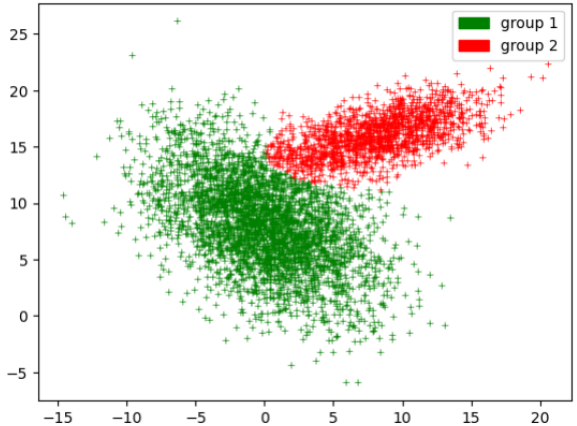
In [ ]:
```
# Distributions
p1 = p * multivariate_normal.pdf(brute, mean=m1_EM, cov=K1_EM)
p0 = (1-p)* multivariate_normal.pdf(brute, mean=m2_EM, cov=K2_EM)

P0 = p0/(p1+p0)
P1 = p1/(p1+p0)

for i in range(len(brute)):
    # From which distribution comes the i th observation?
    if(P1[i]>P0[i]):
        color = 'g'
    else:
        color = 'r'
    # Plotting
    plt.scatter(x[i], y[i], c=color, marker = '+', s = 20, linewidth = 0.5)

# Legend
green_patch = mpatches.Patch(color='green', label='group 1')
red_patch = mpatches.Patch(color='red', label='group 2')
plt.legend(handles=[green_patch, red_patch])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x15b88224c50>
```



Adequacy of classification

We have a "Classification" file corresponding to the labels of the groups from which the data comes. We can therefore use it to count the number of data that have been well classified by the EM algorithm.

In [ ]:
```
# Import
Classification=[]
with open("Classification.txt","r") as file:
    Classification = [int(l) for l in file]

# Overview
Classification[0:10]
```

Out[ ]:
```
[2, 2, 1, 2, 2, 2, 2, 2, 2, 2]
```

In [ ]:
```
def Classification_Gauss(p, m1, K1, m2, K2, Z, C):

    p1 = p * multivariate_normal.pdf(Z, mean=m1, cov=K1)
    p2 = (1-p)* multivariate_normal.pdf(brute, mean=m2, cov=K2)

    P0 = p2/(p1+p2)
    P1 = p1/(p1+p2)

    compt = 0
    clusters = []

    for i in range(len(Z)):

        if(P1[i]>P0[i]):
            clusters.append(2)
        else:
            clusters.append(1)

        if C[i]==clusters[i] : compt=compt+1

    print(f'Clusters contains {compt/len(Z)*100}% of same values as Classification.')

    return
```

In [ ]:
```
Classification_Gauss(p_EM, m1_EM, K1_EM, m2_EM, K2_EM , brute, Classification)
```

```
Clusters contains 96.94% of same values as Classification.
```