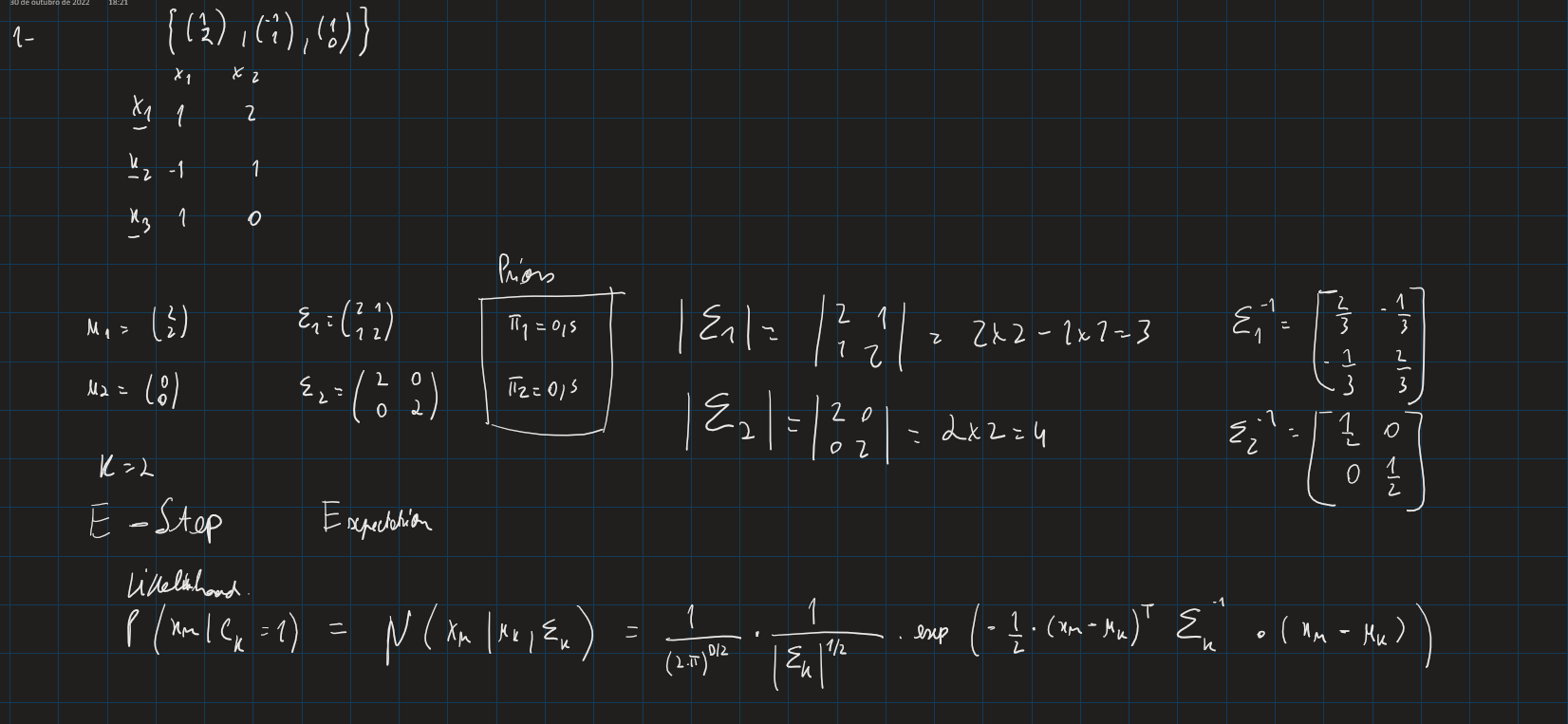
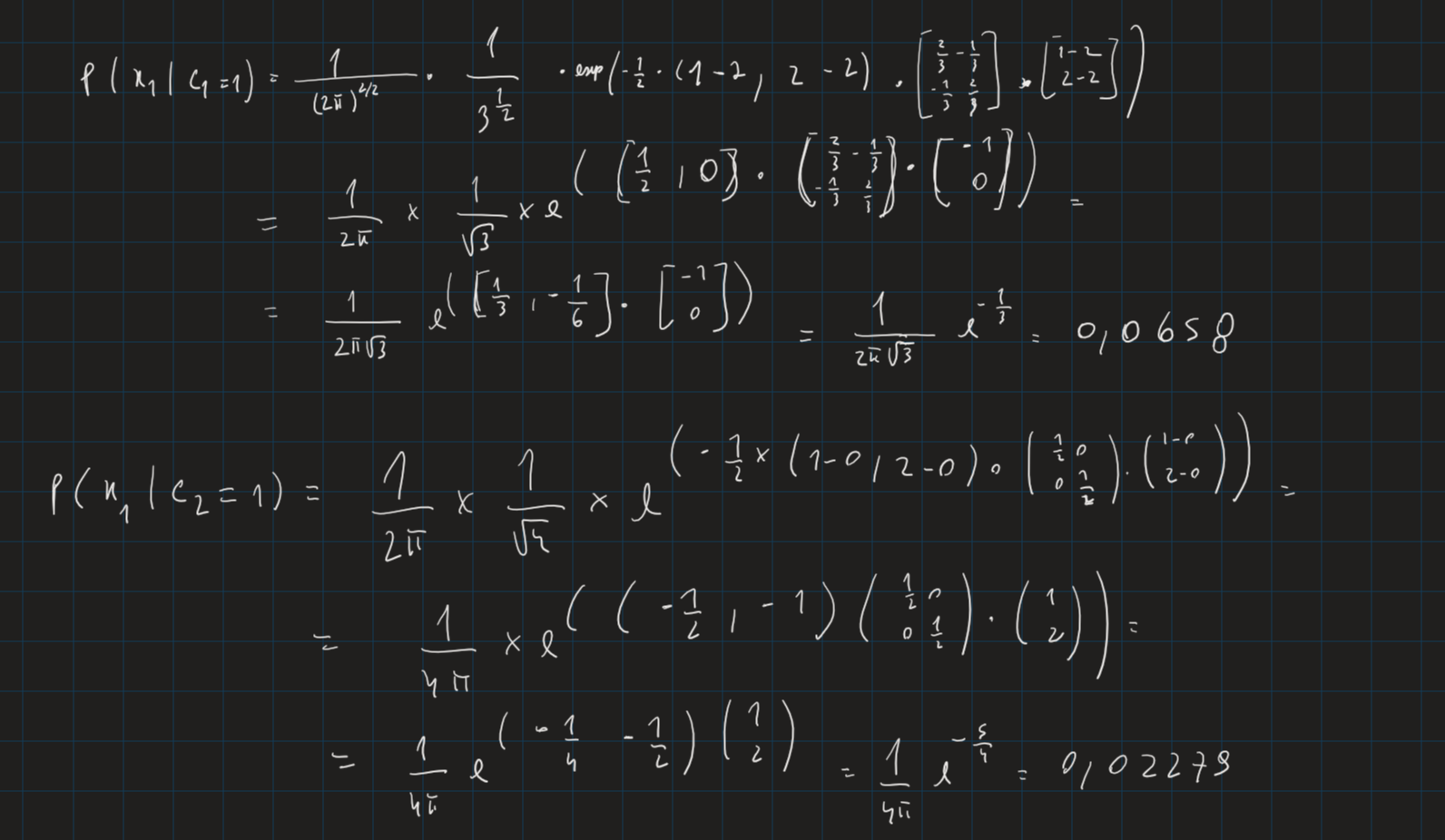
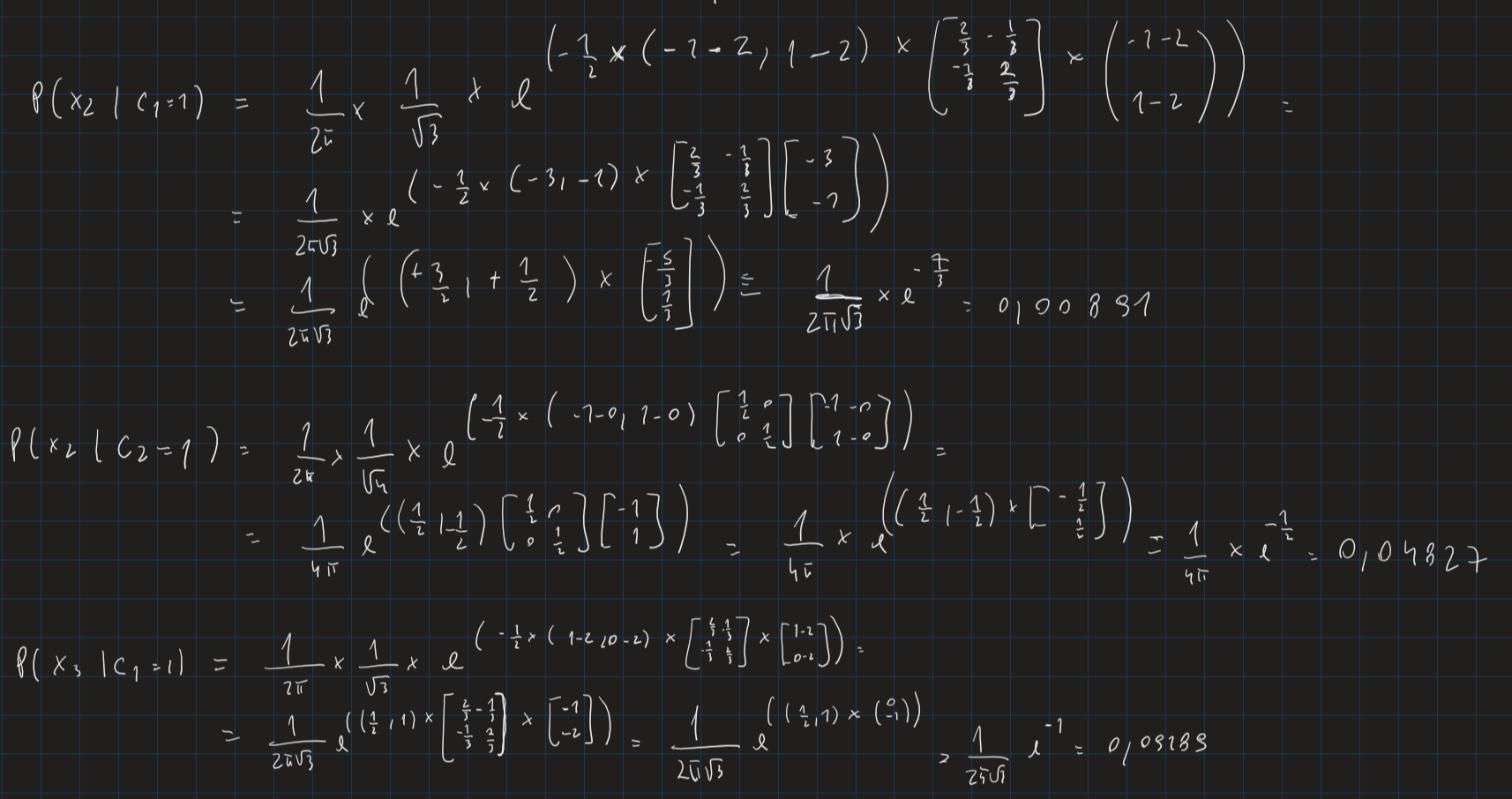
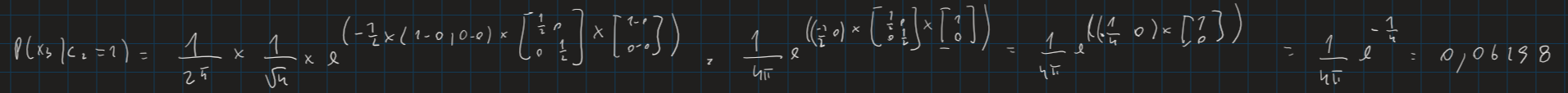
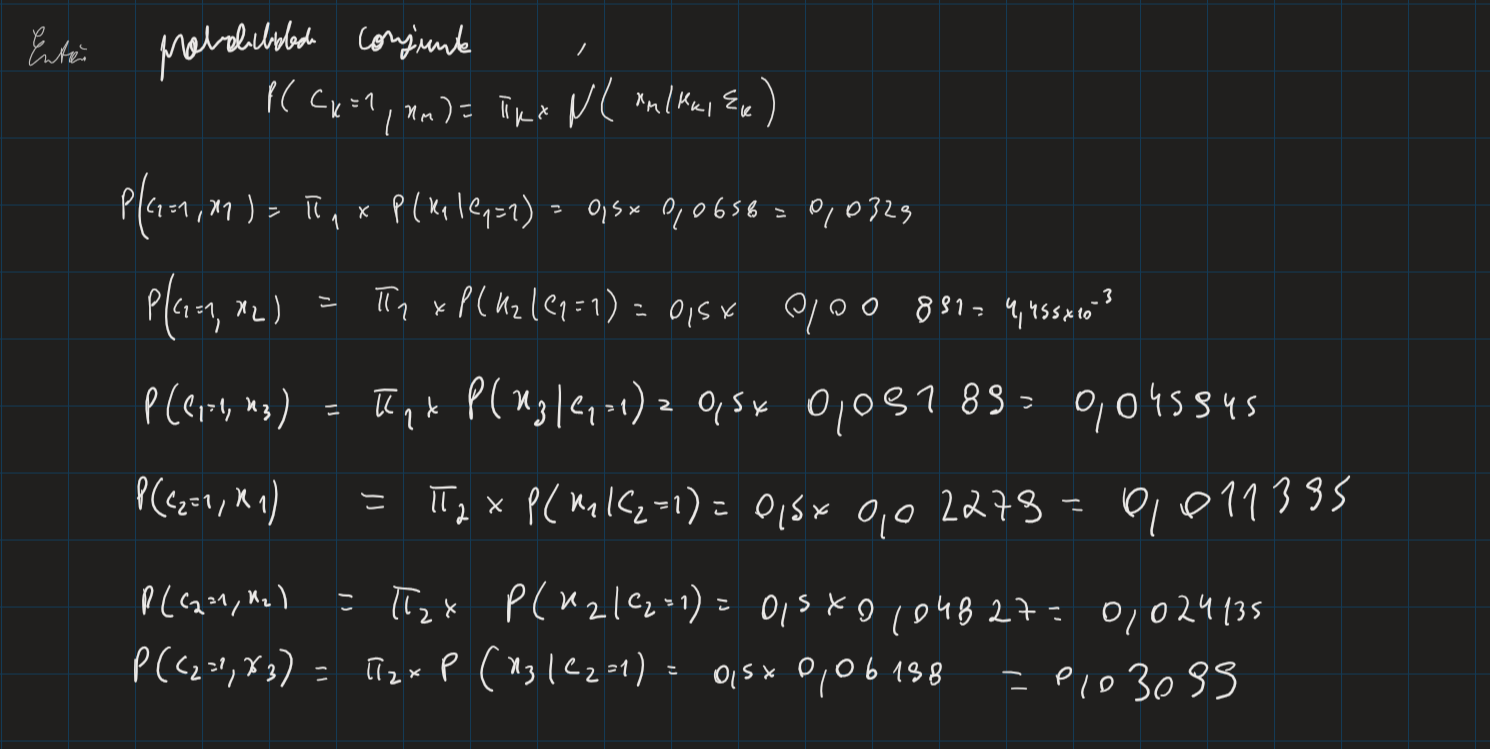
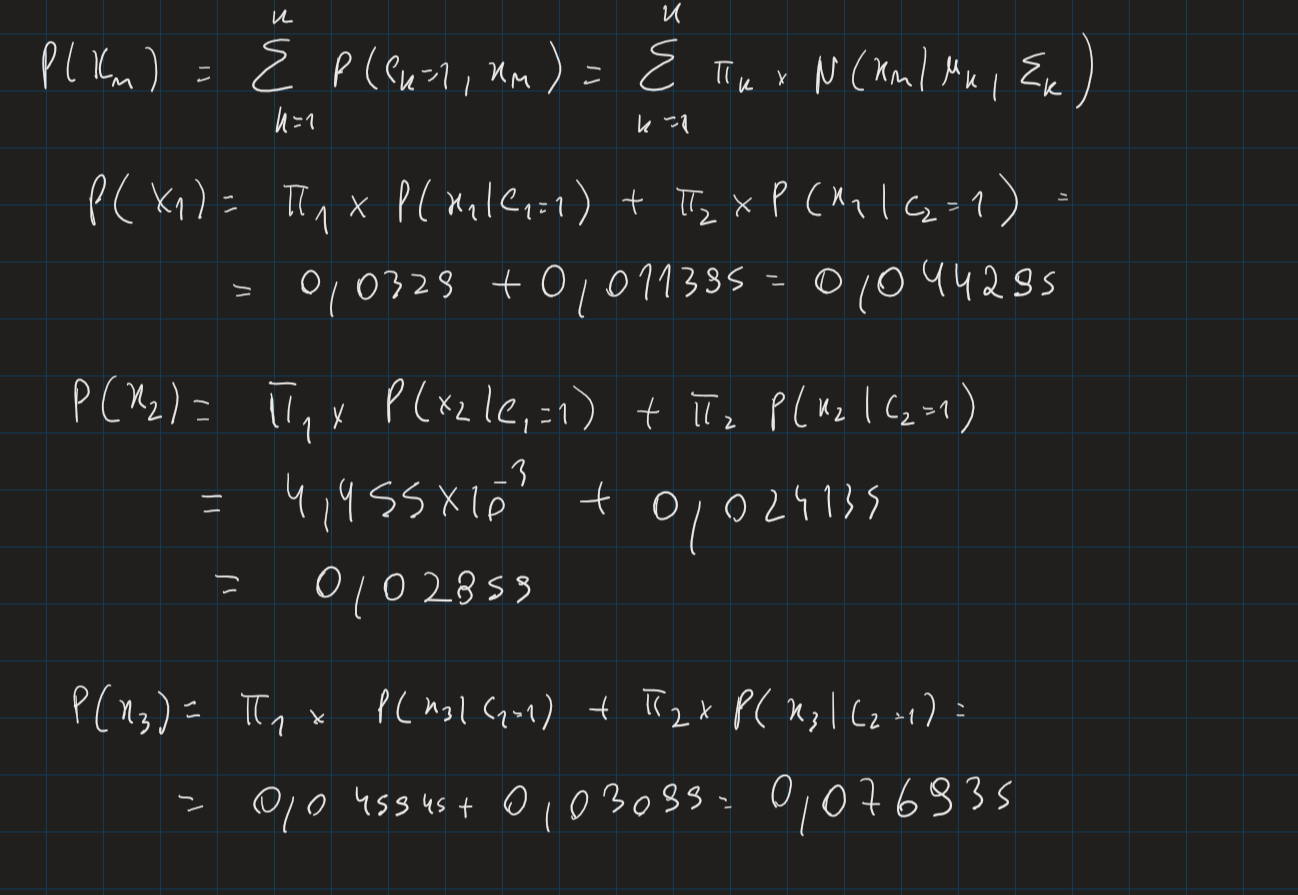
**I. Pen-and-paper**

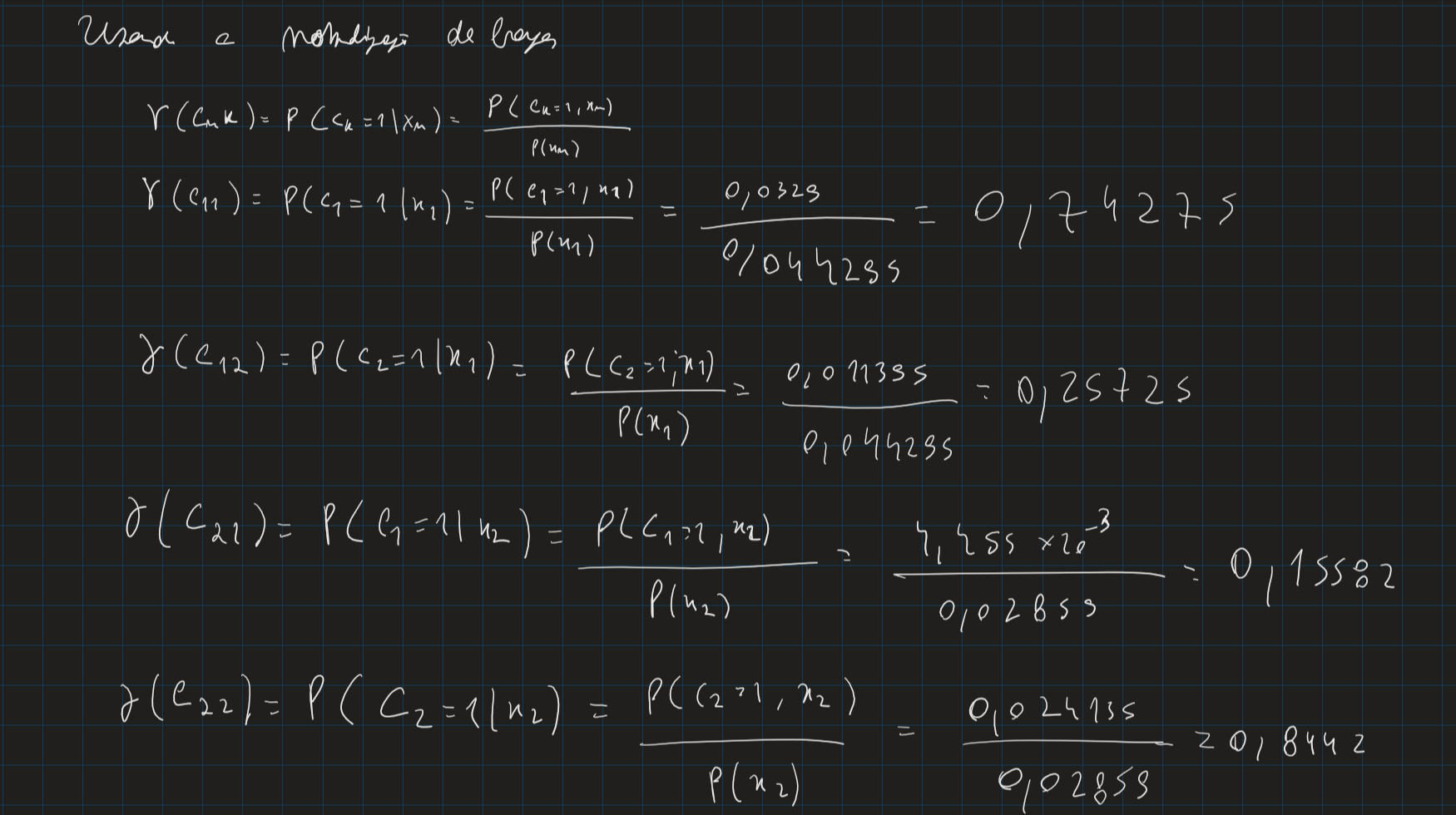


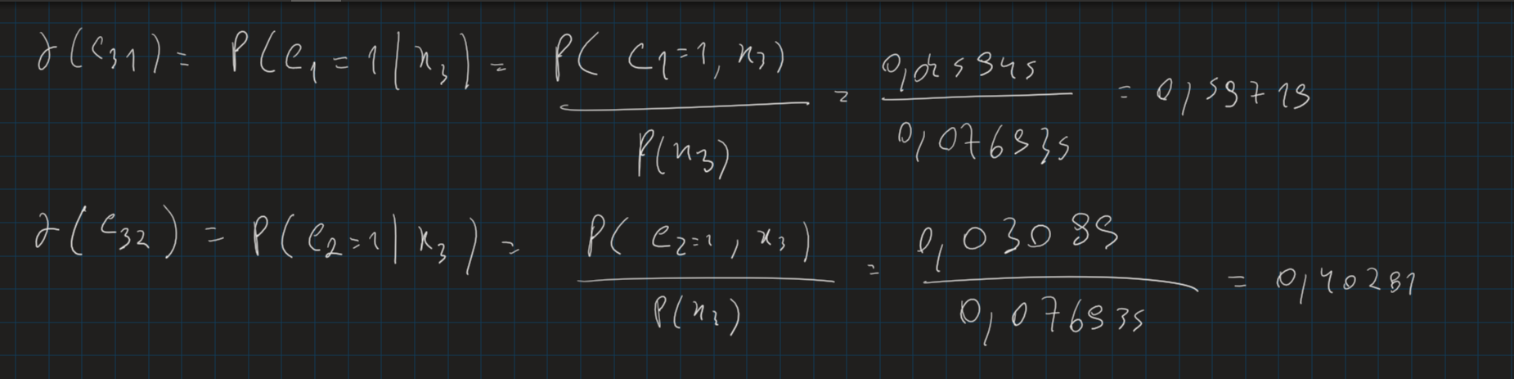


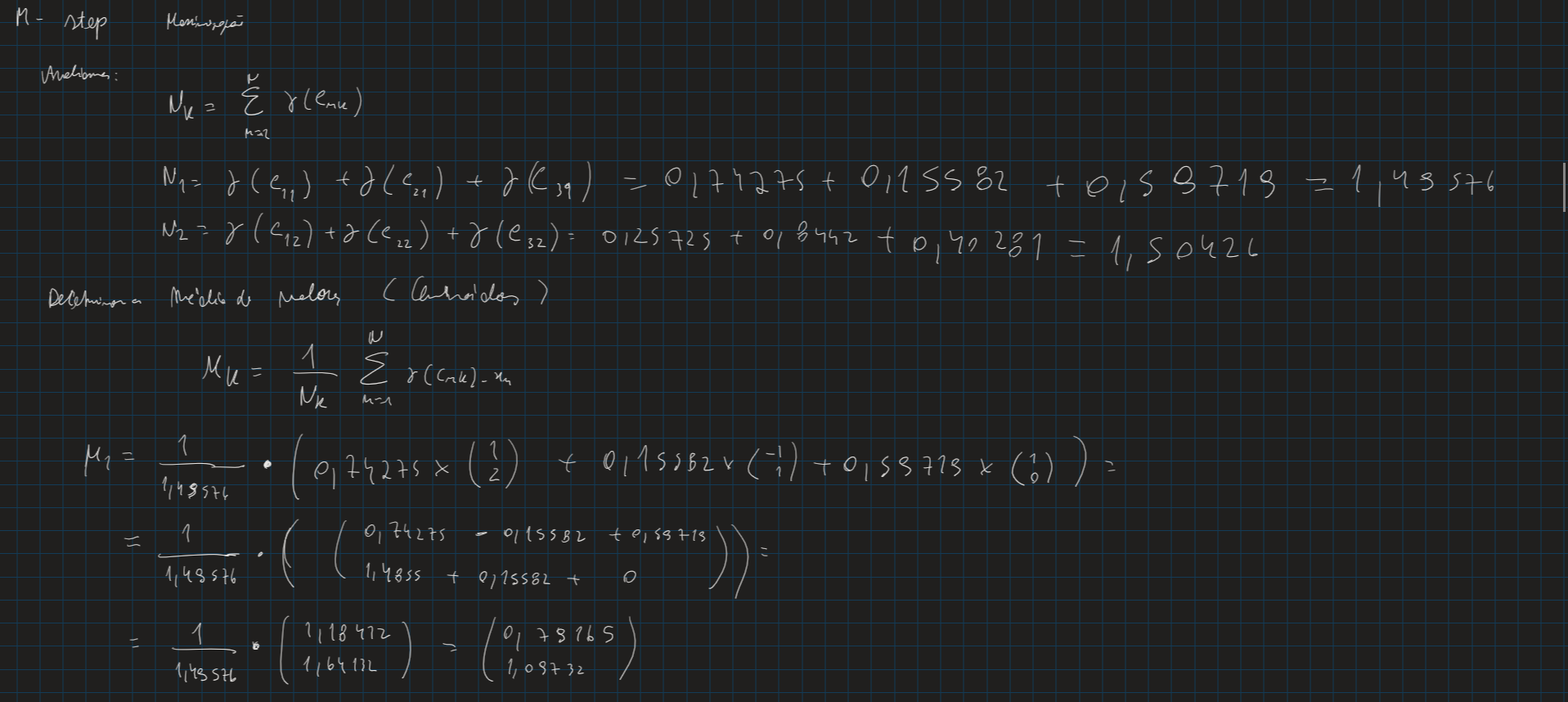
­­­­­­

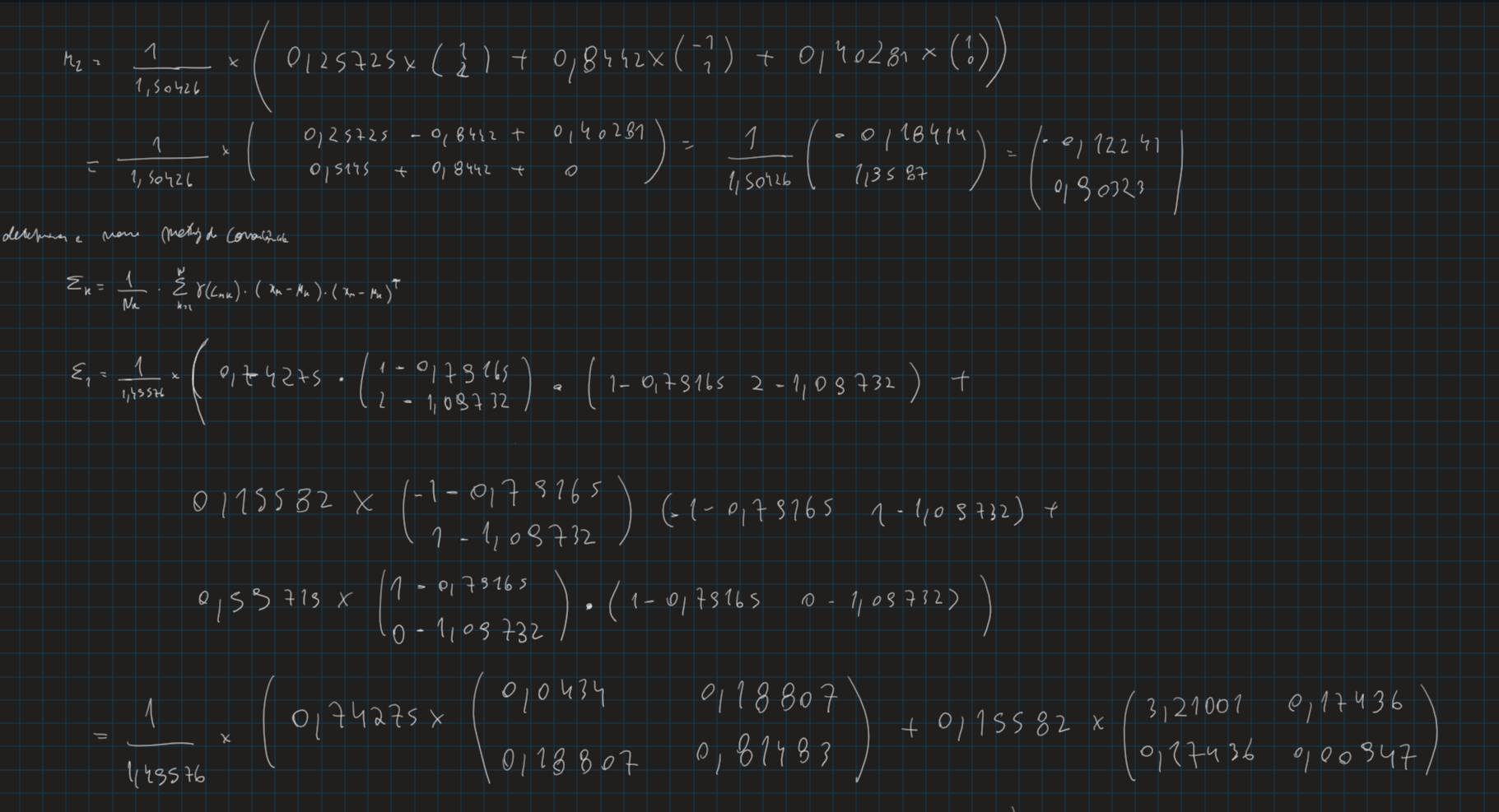
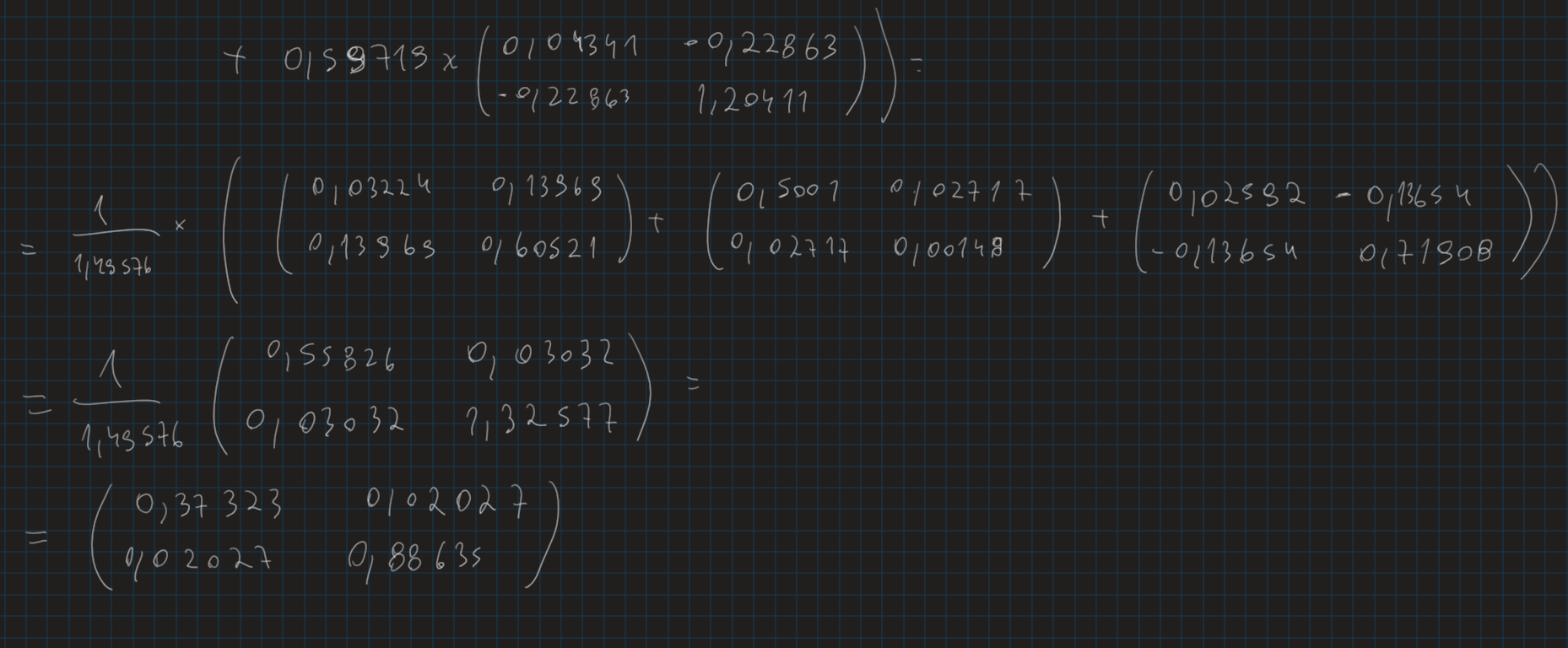


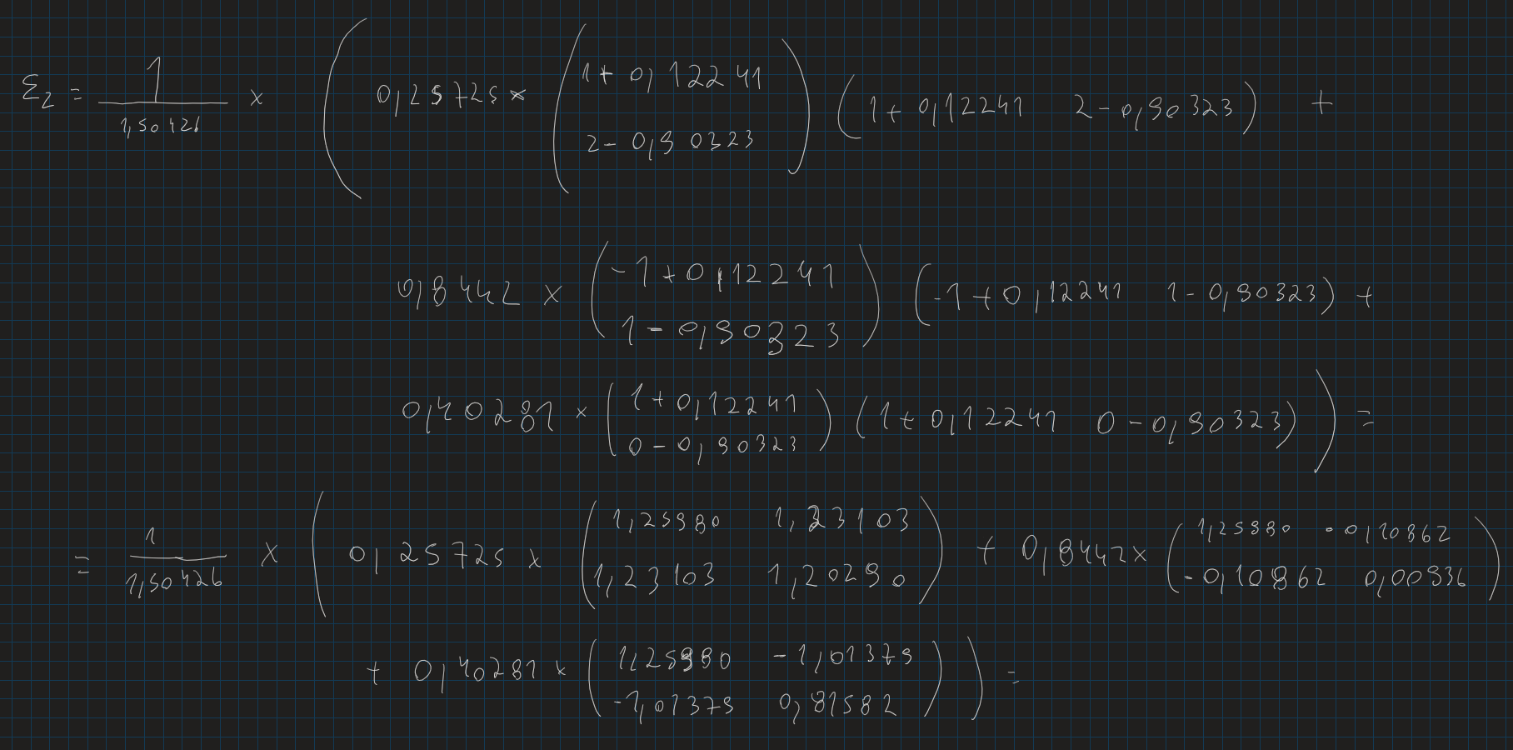
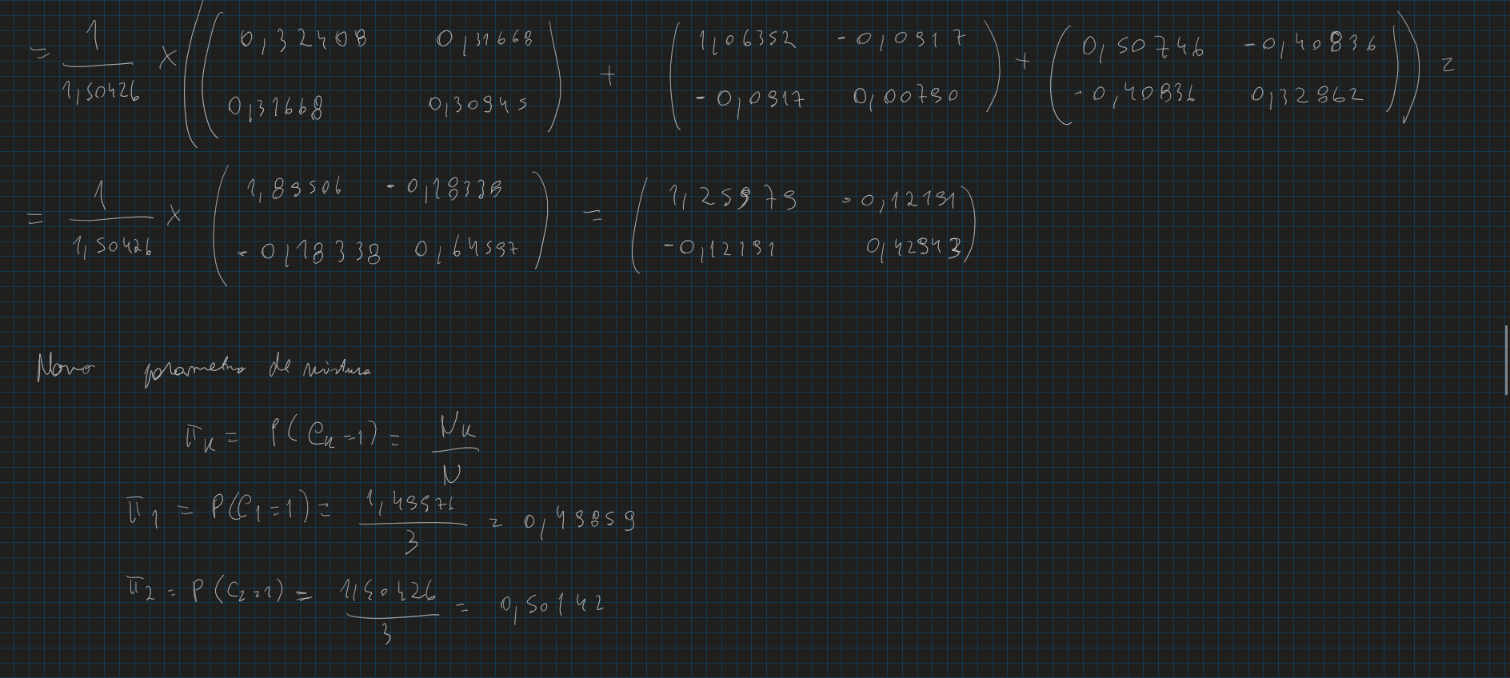


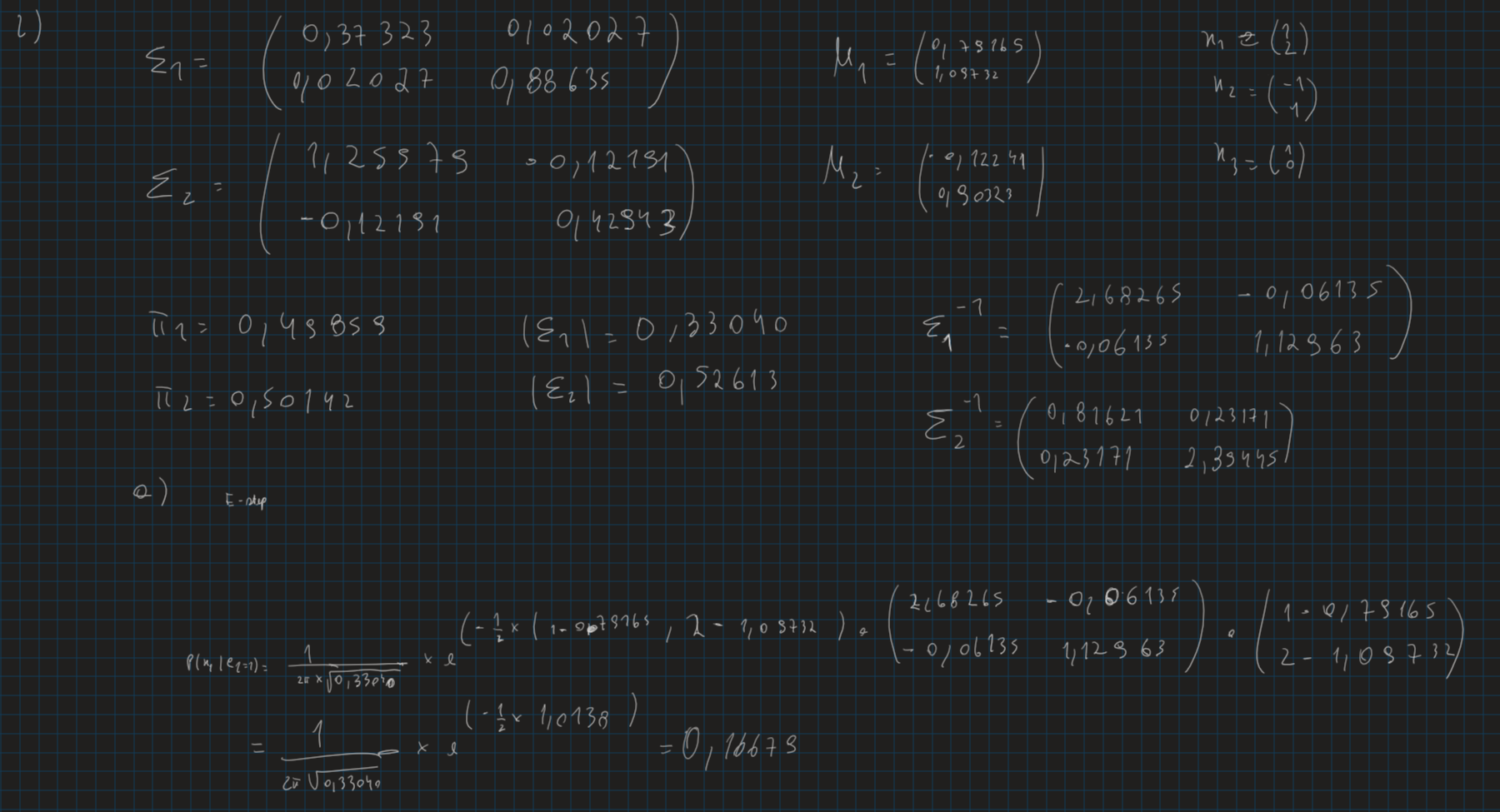


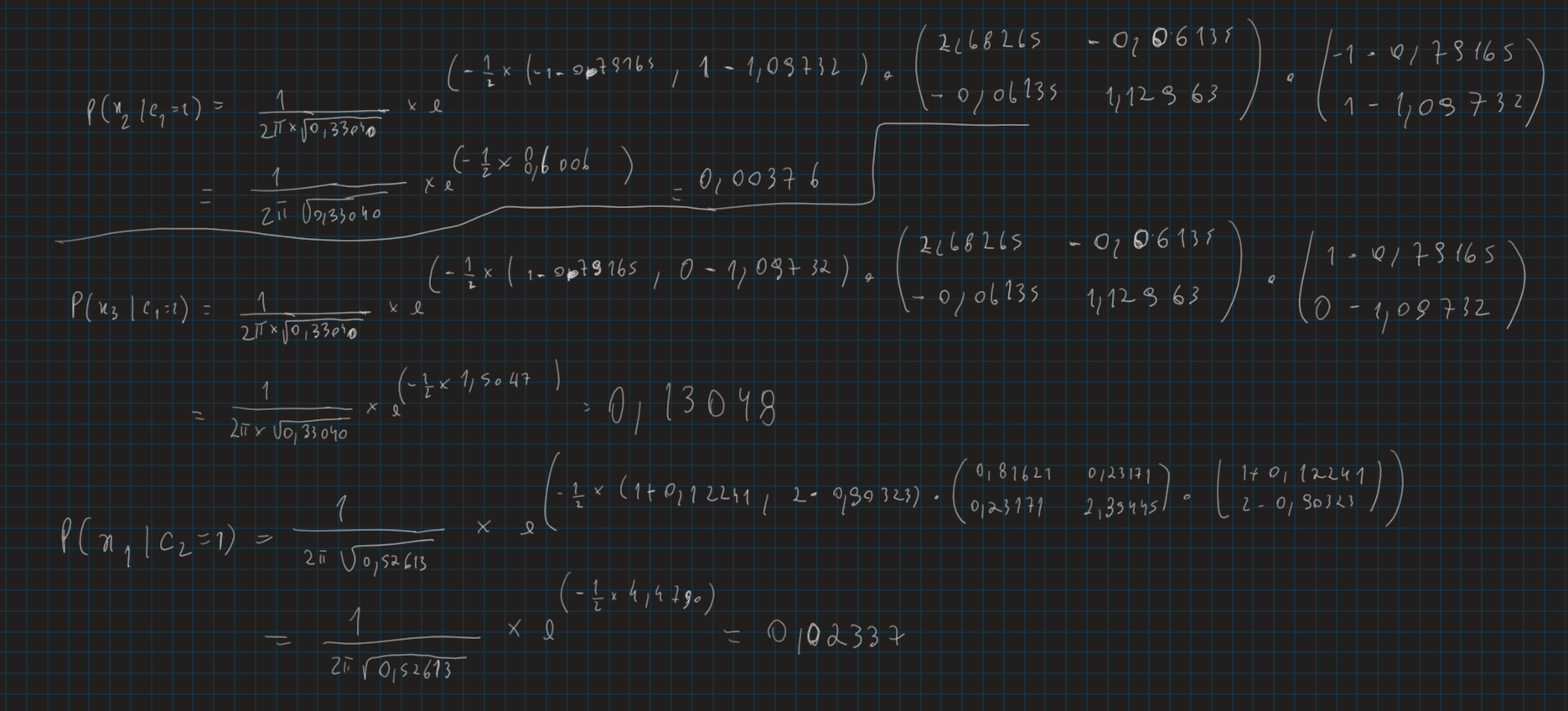


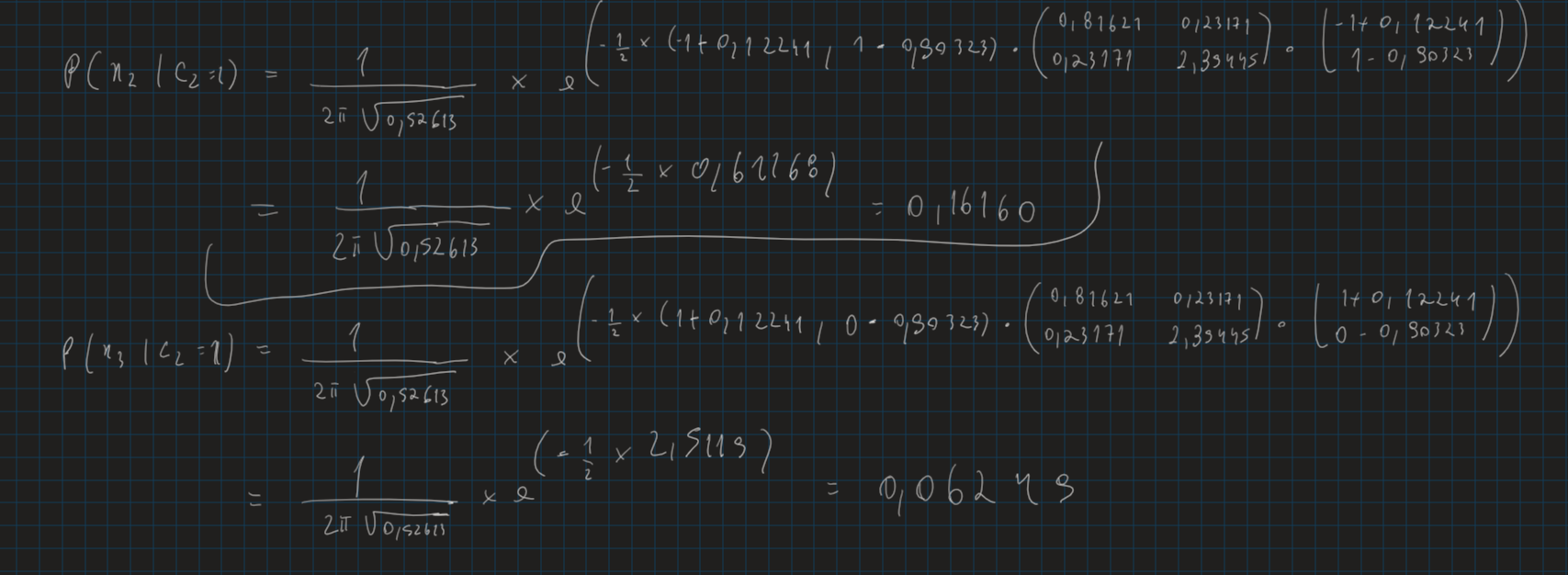


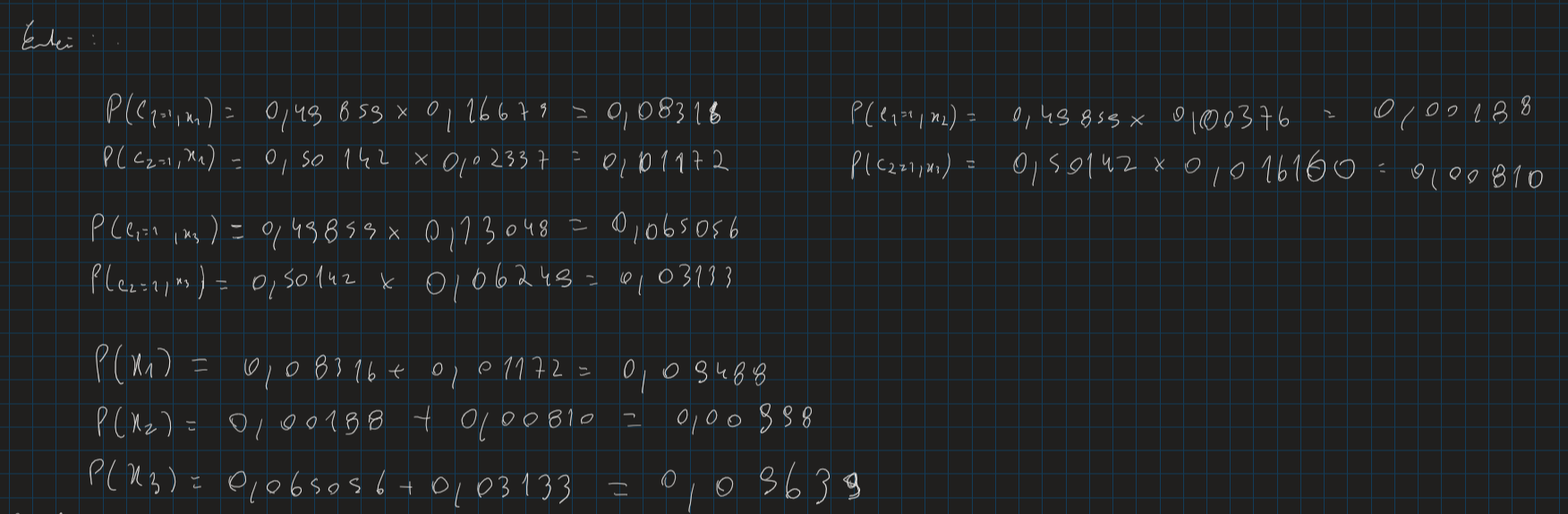


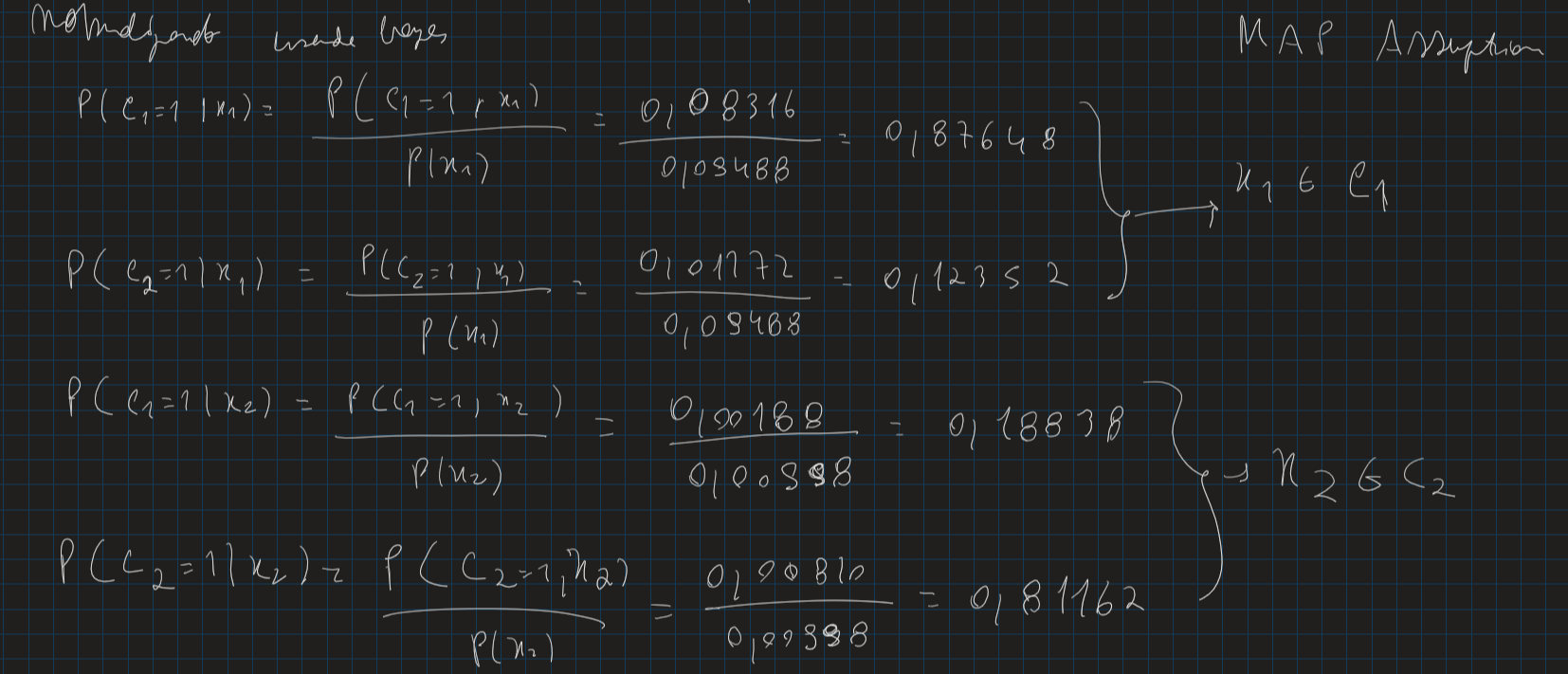


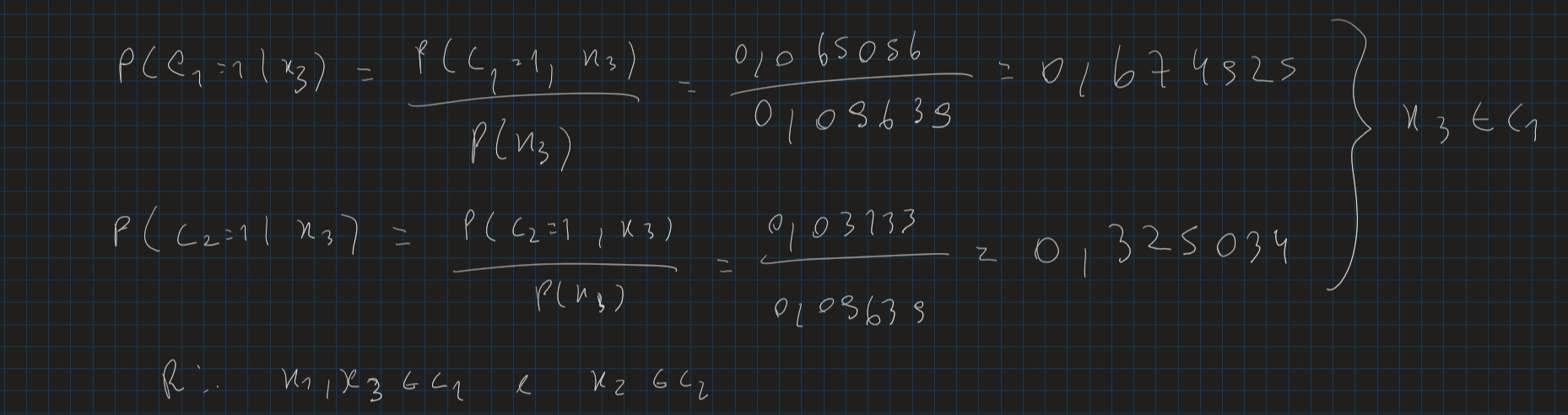
1. A)



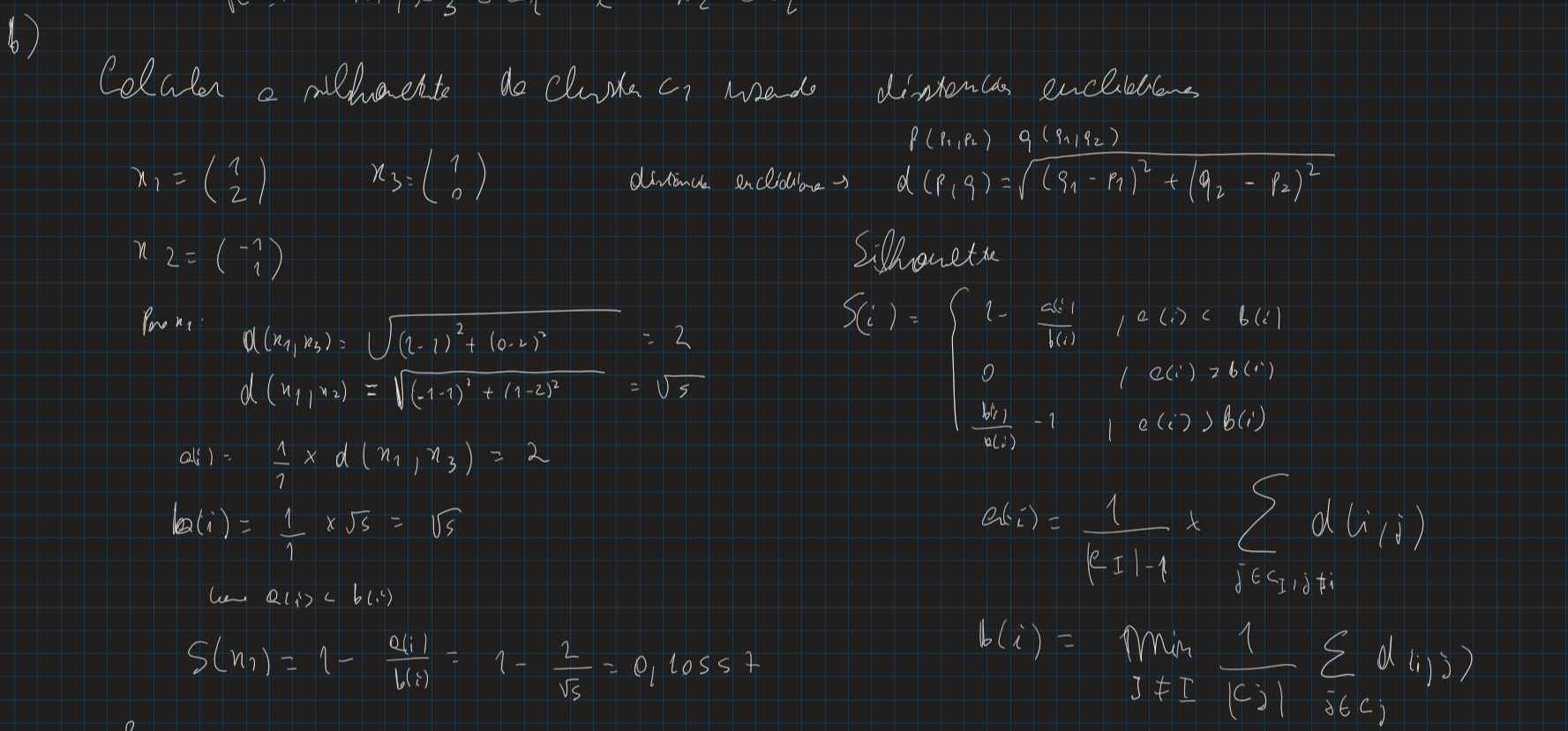


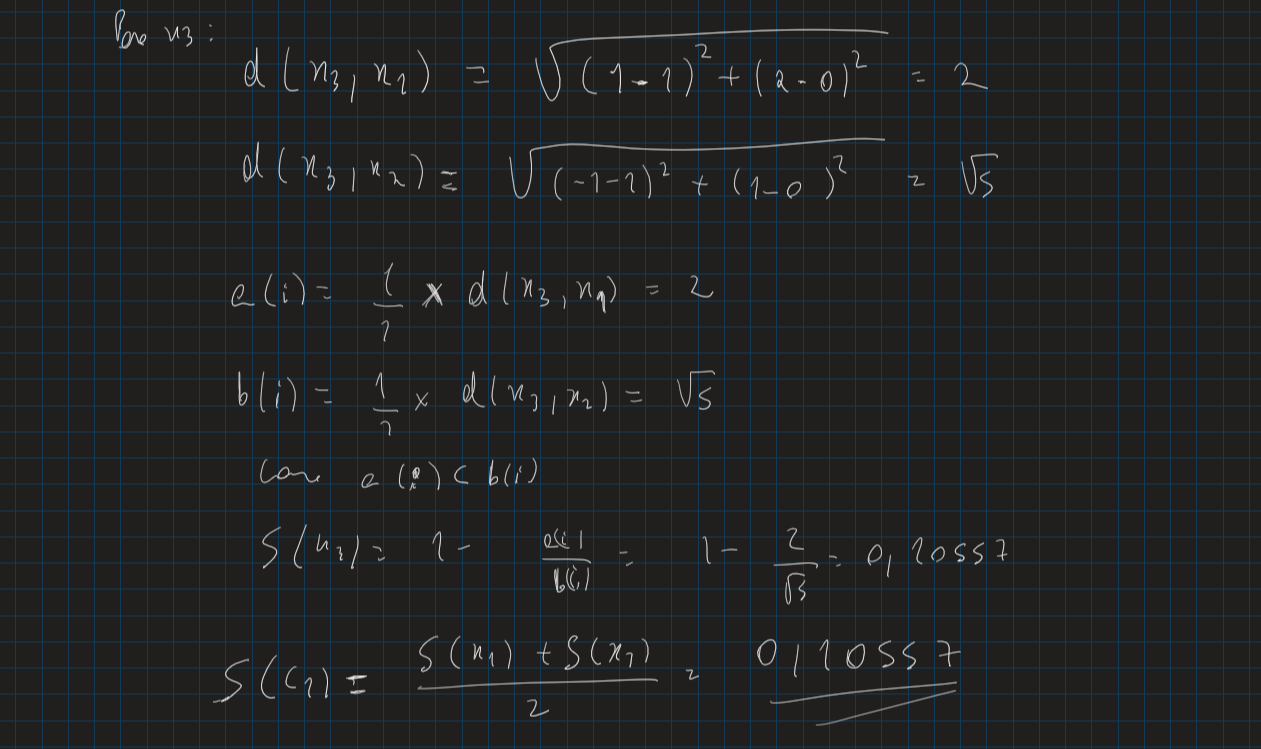






b)







**II. Programming and critical analysis**

**1)**

[0] Silhouette (euclidian): 0.1136202757517943

[1] Silhouette (euclidian): 0.11403554201377072

[2] Silhouette (euclidian): 0.1136202757517943

[0] Purity: 0.7671957671957672

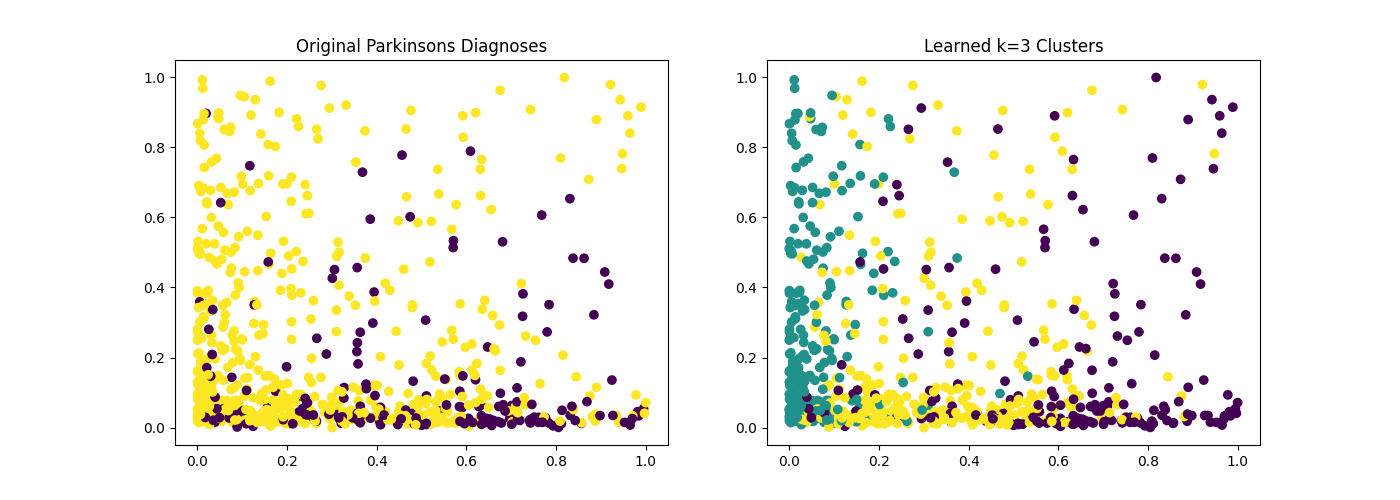
[1] Purity: 0.7632275132275133

[2] Purity: 0.7671957671957672

**2)**

O que está a causar o não determinismo é facto de estarmos inicialmente a considerar que os centroides das respetivas três clusters são completamente aleatórios e usando a distancia da euclidiana dos pontos a esses centroides associamos os pontos aos clusters dos respetivos centroides e caso o centroide altere quando os pontos estão associados aos seus respetivos clusters,  então voltamos a recalcular os centroides até que o calculo do novo centroide resulte nos centroides usados no calculo dos mesmos.

**3)**

****

**4)**

**É necessário 31 componentes principais para explicar mais de 80% de variabilidade**

**III. APPENDIX**

import fractions

import pandas as pd

import numpy as np

#\*####################################################################################

#\* 1)

#\*####################################################################################

#\* import data

from scipy.io.arff import loadarff

data = loadarff("pd\_speech.arff")

df = pd.DataFrame(data[0])

df['class'] = df['class'].str.decode('utf-8')

df['class'] = pd.to\_numeric(df["class"])

#\* aux variable

num\_columns = df.shape[1]

#\* pre-process data

from sklearn.preprocessing import MinMaxScaler

df\_scaled = df.copy()

df\_scaled.iloc[:, 0:num\_columns-1] = MinMaxScaler().fit\_transform(df.iloc[:, 0:num\_columns-1])

#\* partition data

X, y = df\_scaled.iloc[:, 0:num\_columns-1], df\_scaled["class"]

#\* parameterize clustering

from sklearn import cluster

kmeans\_algo\_0 = cluster.KMeans(n\_clusters=3, random\_state=0)

kmeans\_algo\_1 = cluster.KMeans(n\_clusters=3, random\_state=1)

kmeans\_algo\_2 = cluster.KMeans(n\_clusters=3, random\_state=2)

#\* learn the model

kmeans\_model\_0 = kmeans\_algo\_0.fit(X)

kmeans\_model\_1 = kmeans\_algo\_1.fit(X)

kmeans\_model\_2 = kmeans\_algo\_2.fit(X)

#\* produced clusters

y\_pred\_0 = kmeans\_model\_0.labels\_

y\_pred\_1 = kmeans\_model\_1.labels\_

y\_pred\_2 = kmeans\_model\_2.labels\_

#\* compute Silhouette

from sklearn import metrics

print("[0] Silhouette (euclidian):", metrics.silhouette\_score(X, y\_pred\_0, metric='euclidean'))

print("[1] Silhouette (euclidian):", metrics.silhouette\_score(X, y\_pred\_1, metric='euclidean'))

print("[2] Silhouette (euclidian):", metrics.silhouette\_score(X, y\_pred\_2, metric='euclidean'))

#/ compute Purity

import numpy as np

def purity\_score(y\_true, y\_pred):

#/compute contingency/confusion matrix

    confusion\_matrix = metrics.cluster.contingency\_matrix(y\_true, y\_pred)

    return np.sum(np.amax(confusion\_matrix, axis=0)) / np.sum(confusion\_matrix)

y\_true = y

print("[0] Purity:", purity\_score(y\_true, y\_pred\_0))

print("[1] Purity:", purity\_score(y\_true, y\_pred\_1))

print("[2] Purity:", purity\_score(y\_true, y\_pred\_2),"\n")

#\*###################################################################################

#\* 3)

#\*####################################################################################

#\* compute features' variances

from sklearn.feature\_selection import VarianceThreshold

selection = VarianceThreshold().fit(X)

#\* get second max variance

import heapq

max\_three\_variances = heapq.nlargest(3, selection.variances\_)

third\_max\_variance = max\_three\_variances[2]

#\* feature selection

X\_new = VarianceThreshold(threshold=third\_max\_variance).fit\_transform(X)

#\* plot

import matplotlib.pyplot as plt

plt.figure(figsize=(14, 5))

plt.subplot(121)

plt.title(label="Original Parkinsons Diagnoses")

plt.scatter(X\_new[:,0], X\_new[:,1], c=y)

plt.subplot(122)

plt.title(label="Learned k=3 Clusters")

plt.scatter(X\_new[:,0], X\_new[:,1], c=y\_pred\_0)

plt.savefig("figures/plot.png")

plt.show()

#\*####################################################################################

#\* 4)

#\*####################################################################################

#\* learn the transformation (components as linear combination of features)

from sklearn.decomposition import PCA

pca = PCA(n\_components=X.shape[1])

pca.fit(X)

variability = 0

number\_of\_principal\_components = 1

for f in pca.explained\_variance\_ratio\_:

    variability += f

    if variability > 0.80:

        break

    number\_of\_principal\_components += 1

print(number\_of\_principal\_components)

**END**