Laboratory 4

In this laboratory we will focus on computing probability densities and ML estimates.

Gaussian density estimation

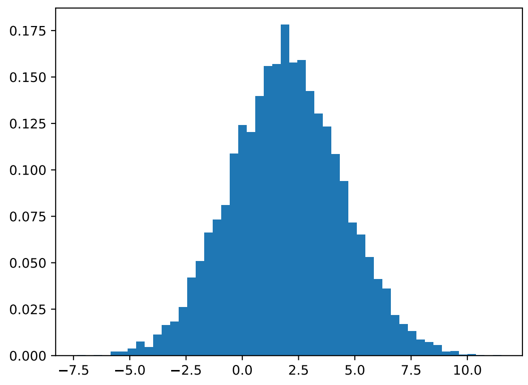
The file `Data/XGau.npy` contains a synthetic dataset of 1-dimensional samples that we will use for the first part of the laboratory. You can load the data with

```
XGAU = numpy.load('Data/XGau.npy')
```

`XGAU` is a 1-D `ndarray` of shape `(10000,)`. Since for this part we work with 1-dimensional data, we use 1-D arrays rather than row vectors (this will simplify visualization)

Plot the normalized histogram of the dataset:

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(XGAU, bins=50, density=True)
plt.show()
```



Gaussian density

Write a function `GAU_pdf(x, mu, var)` that computes the normal density

$$\mathcal{N}(x|\mu, v) = \frac{1}{\sqrt{2\pi v}} e^{-\frac{(x-\mu)^2}{2v}}$$

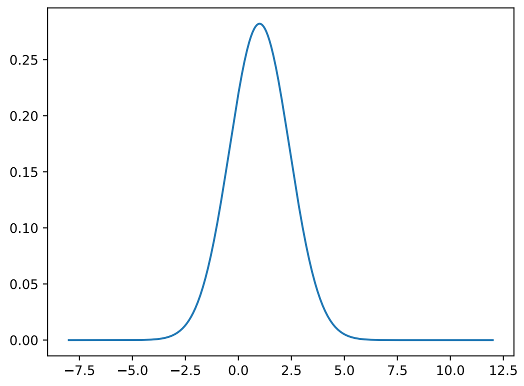
Suggestion 1: In the following we will need to compute the density for all samples of the dataset `XGAU`. You can already write the function so that it's able to compute the density for all samples in an array. The function should accept a 1-D array `x` and compute the density for all elements in `x`. The output should thus be a 1-D array `y`, with $y_i = \mathcal{N}(x_i|\mu, v)$.

Suggestion 2: You can write an additional function that computes the density for a single scalar value x, and then a function that loops over all elements of an array of samples to compute the corresponding densities. However, you can also exploit broadcasting: if you have a 1-D array `x` and a scalar `mu`, then computing `y = x-mu` gives a 1-D array where element `y[i]` corresponds to `y[i] = x[i] - mu`. You shall see that the expression for the pdf will be very similar to that for a scalar value `x`

Suggestion 3: π is already available as `numpy.pi`. Exponentials can be computed using `numpy.exp`, square roots using either `numpy.sqrt(a)` or `a**0.5`

Try visualizing your density:

```
import matplotlib.pyplot as plt
plt.figure()
XPlot = numpy.linspace(-8, 12, 1000)
plt.plot(XPlot, GAU_pdf(XPlot, 1.0, 2.0))
```



You can also check your density is correct by comparing your values with those contained in `Solution/CheckGAUPdf.npy`

```
pdfSol = numpy.load('Solution/CheckGAUPdf.npy')
pdfGau = GAU_pdf(XPlot, 1.0, 2.0)
print(numpy.abs(pdfSol - pdfGau).mean())
```

The result should be zero or very close to zero (it may not be exactly zero due to numerical errors, however it should be a very small number, e.g. $\approx 10^{-17}$)

We can now try to compute the likelihood $\mathcal{L}(\mu, v) = \prod_{i=1}^{n} \mathcal{N}(x_i|\mu, v)$ for our dataset `XGAU`. Let's compute the value of the likelihood for $\mu = 0, v = 1$.

```
ll_samples = GAU_pdf(XGAU, 1.0, 2.0)
likelihood = ll_samples.prod()
print(likelihood)
```

We get $\mathcal{L}(\mu, v) = 0$. Why?
Try with different values. What do you get?
Check the values of the density for each sample `ll_samples`. They are small. The likelihood is the product of all these values. Since floating points cannot represent arbitrary small values, the product saturates to zero.

Rather than using directly the density, it's better to work with its logarithm.

$$\log \mathcal{N}(x|\mu, v) = -\frac{1}{2}\log(2 * \pi) - \frac{1}{2}\log v - \frac{(x-\mu)^2}{2v}$$

Write a function `GAU_logpdf(x, mu, v)` which computes the log-density using the expression above (`numpy.log` allows computing logarithms).
Check again your density (compute the density from the log-density, visualize it, compare the values with those obtained using the previous function)

The log-likelihood is defined as

$$\ell(\mu, v) = \log \mathcal{L}(\mu, v) = \sum_{i=1}^{n} \log \mathcal{N}(x_i|\mu, v)$$

and corresponds to a sum of the log-density of the samples x_i. Compute the log-likelihood for different values of μ and v — ℓ does not saturate. We can thus utilize ℓ to compare likelihoods for different model parameters.

Gaussian ML estimate

We now want to estimate the parameters that better describe our dataset `XGAU`. We have seen that the Maximum Likelihood estimates are given by

$$\mu^*_{ML} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad v^*_{ML} = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu^*_{ML})^2$$

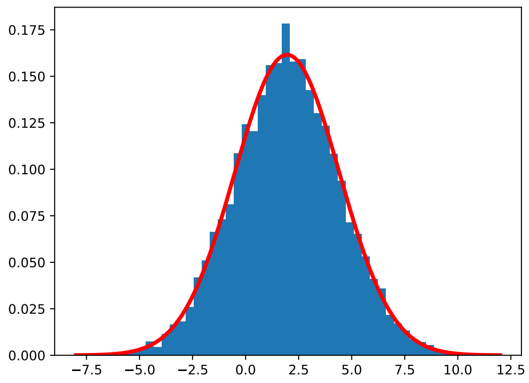
Compute the ML estimates for the data `XGAU`. You can compute the sum of elements of an array using the `.sum` method of the array. Alternatively, you can use the methods `.mean` and `.var` to compute the samples mean and variance.

Compute the log-likelihood using the ML estimates $\ell(\mu^*_{ML}, v^*_{ML})$. This requires computing the log-density for all samples, and summing the obtained values. You should get (up to numerical precision errors)

```
ll = loglikelihood(XGAU, m_ML, v_ML)
print(ll)
```

```
-23227.07765460272
```

Plot the density on top of the histogram (remember that you should plot the density, not the log-density, so you need to plot `numpy.exp(logpdf_GAU(XPlot, m_ML, v_ML))`



In this case we have a good fit of the histogram, and the density well represents the distribution of our data. Check what happens if we reduce the number of samples used to estimate the ML parameters.

Multivariate Gaussian

The Multivarite Gaussian (MVG) density is defined as

$$\log \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{M}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$$

where M is the size of the feature vector $\boldsymbol{x}$, and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.

Write a function to compute the log-density `logpdf_GAU_ND(x, mu, C)` for a single sample `x`. `x` and `mu` should both be a numpy `array` of shape `(M, 1)`, whereas `C` is a numpy `array` of shape `(M, M)` representing the covariance matrix $\boldsymbol{\Sigma}$.

Suggestion 1: As for the previous case, you can try writing the function so that it is able to compute the densities for all samples of matrix $\boldsymbol{X} = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_n]$, outputting an array (1-D, or 2-D row vector) $Y = [\log \mathcal{N}(x_1|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \ldots \log \mathcal{N}(x_n|\boldsymbol{\mu}, \boldsymbol{\Sigma})]$.

Suggestion 2: Broadcasting and re-arranging of the computations can also be used in this case to significantly speed up the computation of Y, and avoid explicitly looping over the elements of $\boldsymbol{X}$. However, getting it right is much more complex than in the 1-dimensional case. If you're interested in trying anyway, you can ask for more insights. Otherwise, simply compute the array of log-densities using a loop over the columns of the data matrix `X`.

Suggestion 3: The inverse of a 2-D array `C` can be computed using `numpy.linalg.inv`. The log-determinant $\log|C|$ of `C` can be computed using `numpy.linalg.slogdet` — pay attention that the function returns **two** values, the absolute value of the log-determinant is the second one (the first is the sign of the determinant, which, for covariance matrices, is positive). Again, directly computing the logarithm of the determinant can cause numerical issues. `numpy.linalg.slogdet` performs the computations in a way that is robust also for very small values of $|C|$.

You can check your pdf comparing with the results in the `Data` folder:

```
XND = numpy.load('Solution/XND.npy')
mu = numpy.load('Solution/muND.npy')
C = numpy.load('Solution/CND.npy')
pdfSol = numpy.load('Solution/llND.npy')
pdfGau = logpdf_GAU_ND(XND, mu, C)
print(numpy.abs(pdfSol - pdfGau).mean())
```