2.1 Probability and Statistics Bases for Robotics

October 25, 2020

1 2.1 Probability and Statistics Bases for Robotics

The field of robotics has found great success using a probabilistic approach to handle uncertainty. In contrast to industrial robots, which reside in controlled environments, mobile robots (the focus of this book) have to adapt to additional detrimental factors such as: dynamic environments, sensor disturbances, or unreliable movement systems.

The core principle of this **probabilistic robotics** is to represent this uncertainty as probability distribution. In most cases we will use the observations from the environment (usually denoted as z_n), to estimate the most probable state (x_n) and how certain this prediction is (Σ_{x_n}).

In the series of notebooks in this chapter we will overview the **gaussian distribution**, one of the most used probability distributions!

[1]:
```
# IMPORTED LIBS

import numpy as np
from numpy import random
import matplotlib.pyplot as plt

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

1.1 2.1.1 The gaussian distribution

The gaussian distribution (also known as Normal distribution) is caracterized by two parameters:

- The **mean** (μ) is the expected value of the distribution.
- The **standard deviation** (σ) represents how dispersed are the possible values.

The probab1ility distribition function (**pdf**) of a given Gaussian distribution is defined as:

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

1.1.1 ASSIGNMENT 1: Computing and plotting gaussians

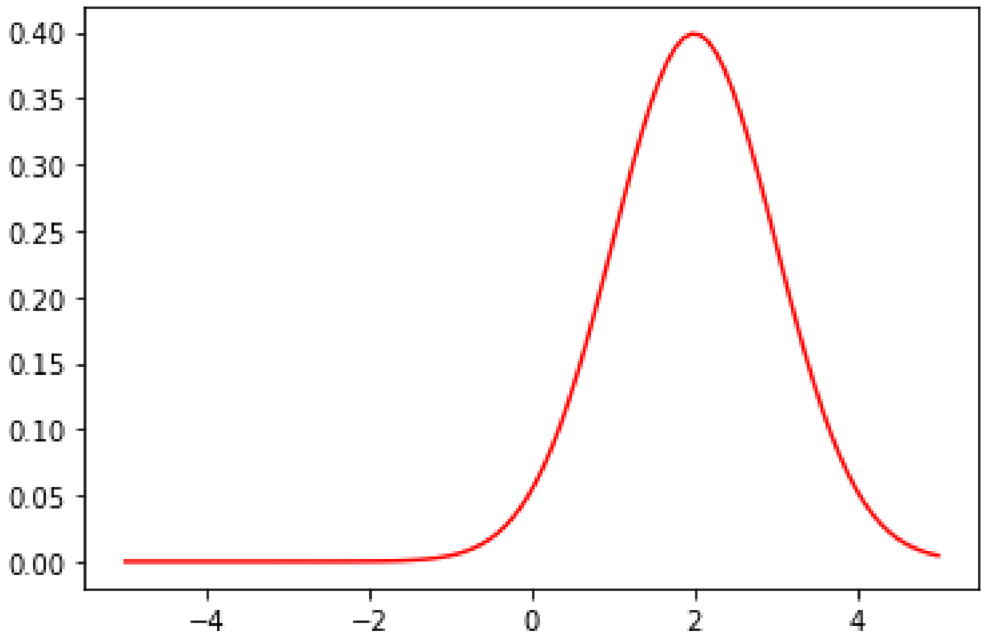
Complete the following function in order to plot a gaussian pdf with $\mu = 2$ and $\sigma = 1$.

Evaluate this gaussian pdf in the interval $x \in [-5,5]$, generating 100 samples between those values.

Hint: use the `np.linspace()` function, which returns evenly spaced numbers over a specified interval. The constant `np.pi` can be also useful.

[2]:
```
def evaluate_gaussian(mu, sigma, X):
    """Evaluates a gaussian distribution between in the given points

    Args:
        mu: mean of the distribution
        sigma: standard deviation of the distribution
        X: points where the function is going to be evaluated
    """
    variance = sigma ** 2 # Get the variance from the given standar deviation
    res = (1 / np.sqrt(2*np.pi*variance)) * np.exp(-np.power(X-mu,2) / (2*np.
 ↪power(sigma,2))) # Implement the gaussian distribution computation

    return res
```

[3]:
```
# RUN

# Gaussian parameters (mean and standard deviation)
mu = 2
sigma = 1

# Create the array of values where the gaussian distribution is going to be␣
 ↪evaluated
min_interval=-5
max_interval=5
n_samples=100
X = np.linspace(min_interval,max_interval,n_samples)

# Call the function and plot the results
res = evaluate_gaussian(mu, sigma, X)
plt.plot(X, res, 'r') # Show the results
plt.show() # Try what happens if you remove this line ;)
```



1.1.2 Sampling from a distribution

Sampling from a random distribution consists of generating a set of values that follows that random probability distribution to a given extent.

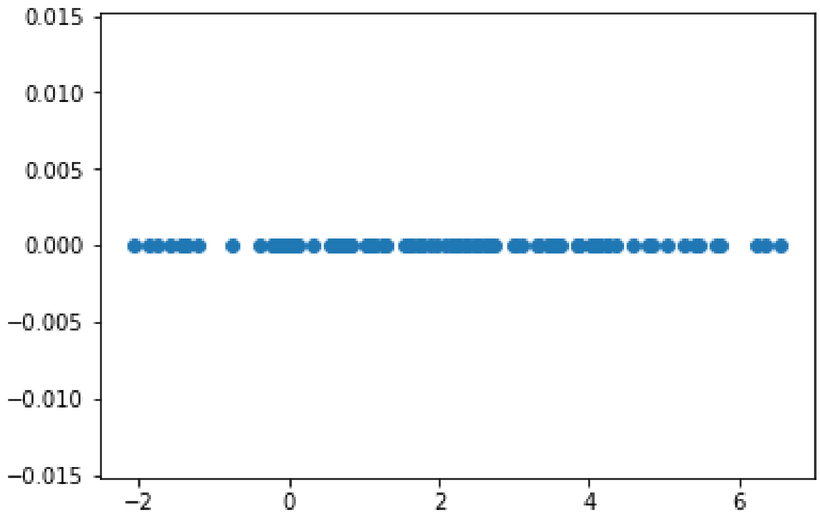
This is of special interest because use of sampling in *particle filters*.

1.1.3 ASSIGNMENT 2: Sampling from gaussians

Use the function `randn()` in the `random` module of numpy.

This module contains functions to do sampling for a variety of random distributions. You can find additional documentation here: Link

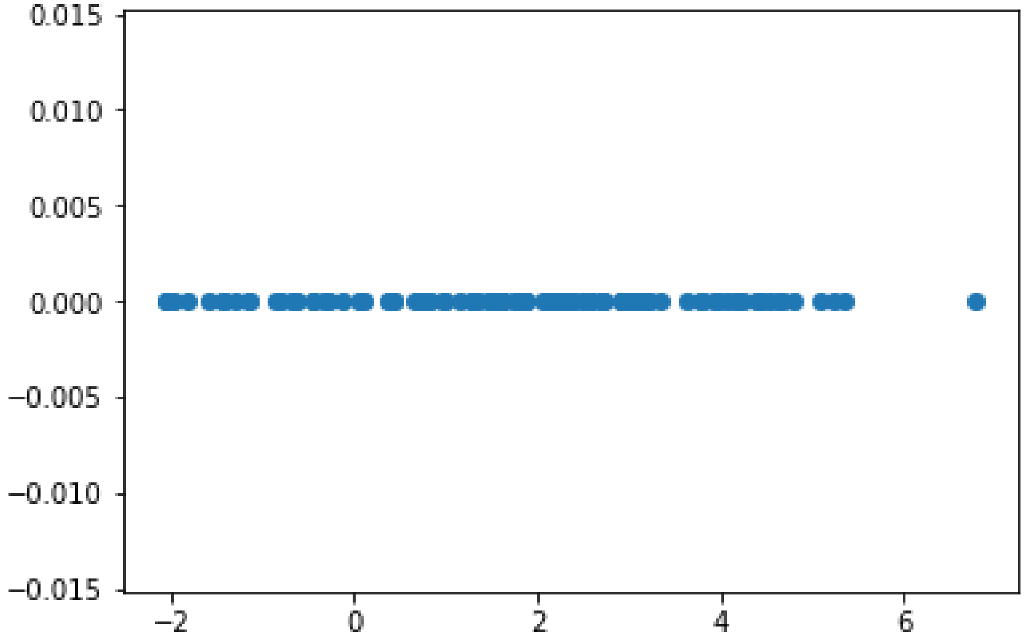
Sample a gaussian distribution with $\mu = 2$ and $\sigma = 2$. Then plot the resulting values along the x axis.



Example of a possible result

[4]:
```
def gen_samples(n, mu, sigma):
    """Generate n samples of a gaussian distribution

    Args:
        n: Number of samples
        mu: mean of the distribution
        sigma: standard deviation of the distribution

    Returns:
        array of samples
    """

    samples = sigma * np.random.randn(n) + mu
    return samples
```

[5]:
```
# RUN
# RUN
num = 100
mu = 2
sigma = 2
plt.scatter(gen_samples(num, mu, sigma), np.zeros(num))
plt.show()
```



1.1.4 Thinking about it (1)

Having completed the code above, you will be able to **answer the following questions**:

- Which value do the samples concentrate around? Why?

  They concentrate around the mean of the distribution, because the closer the sample is to the mean, the higher the value of the probability density function, which means that it is more likely that it will get sampled.

- Why we observe less samples the further they are from that value??

  The value of the probability density function decreases as the samples get further away from the mean, which means the further they are from the mean, the less likely it is that they will be sampled.

Indeed, if we keep sampling the distribution and build an histogram of the obtained samples, the resulting histogram will be similar to its respective gaussian given a large enough number of samples.

1.1.5 ASSIGNMENT 3: Building an histogram of samples

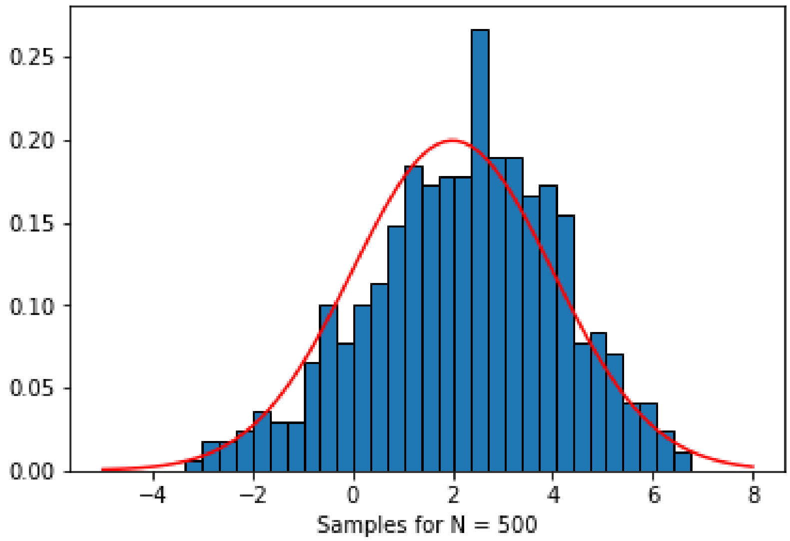
For checking this, we ask you to:

1. Create a large sample vector, i.e. size 1000.
2. Then, complete the function `hist_slice()`, which takes an array of samples and an integer `n`. This function plots the first `n` values of the array as a **histogram**.

3. To show the results of the exercise we will employ the use of Jupyter widgets. You can find more info about them here [link], but for the time being use the commented call to `interact`.

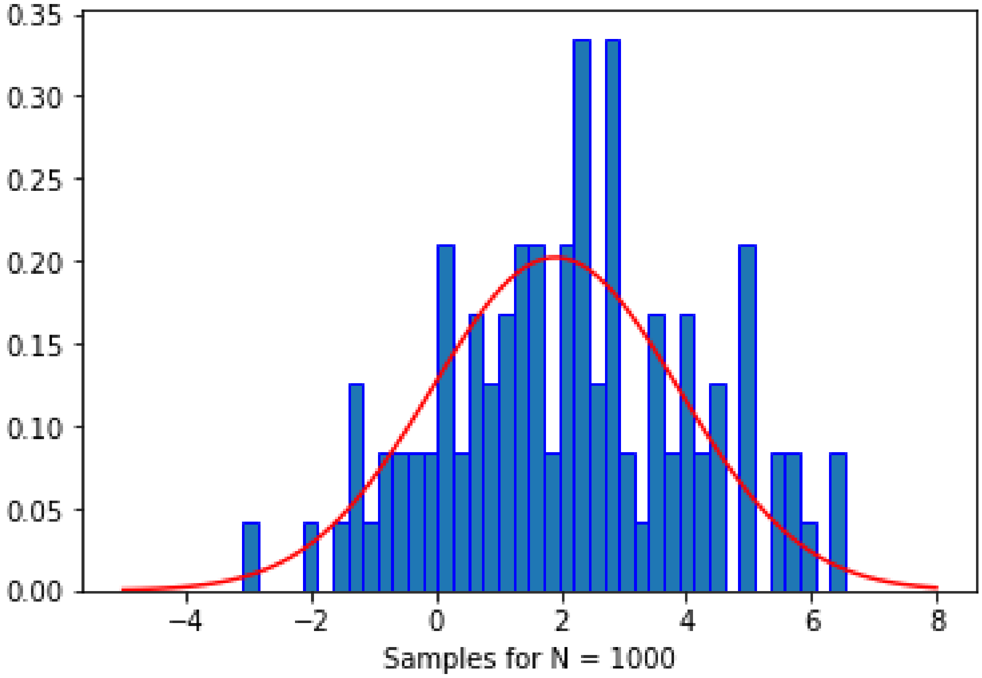
Play around with different parameters of the `plt.hist()` function from matplotlib.

The bars of the histogram should be normalized by the total area. (HINT: Set the optional `density` and `stacked` parameters of `hist()` to True)



[6]:
```
def hist_slice(samples, n):
    """Plot histogram for the first n values in samples"""
    X = np.linspace(-5., 8., 100)
    plt.plot(X, evaluate_gaussian(np.mean(samples), np.std(samples), X), 'r')
    plt.hist(samples[0:n-1], bins=40, edgecolor='b', density=True, stacked=True)
    plt.xlabel("Samples for N = %d" % (len(samples)))
    plt.show()
```

[7]:
```
# RUN
random.seed(0)
samples = gen_samples(1000, 2, 2)
n = 100
hist_slice(samples, n)
```



[8]:
```
# RUN
interact(hist_slice, samples=fixed(samples), n=(100, 1000, 100));
```

```
interactive(children=(IntSlider(value=500, description='n', max=1000, min=100, step=100), Outpu
```

1.2 2.1.2 Properties of the Gaussian distribution

Once we have acquired a certain amount of familiarity with the gaussian distribution, we can go along some of its principal properties, which are the main reason of this distribution's wide usage in robotics.

[9]:
```
# Imports

from scipy import stats
from scipy import signal
```

1.2.1 Central limit theorem

Property. The sum of N independent and identically distributed (i.i.d.) random variables, i.e. that belong to the same distribution and are independant to each other, becomes increasingly Gaussian the larger is N.

This property holds true regardless of the probability distribution was used to create the samples. It is one of the key concepts in probability, as it allows the generalization of many problems.

You can see a video demonstration of this by running the cell bellow:

[10]:

```
%%HTML
<center>
<iframe width="560" height="315" src="https://www.youtube.com/embed/dlbkaurTAUg?
↪autoplay=0&mute=1" frameborder="0" allow="accelerometer; autoplay;␣
↪encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
</center>
```

```
<IPython.core.display.HTML object>
```

1.2.2 ASSIGNMENT 4: Verifying the central limit theorem

We ask you to create a similar demonstration as the example above.

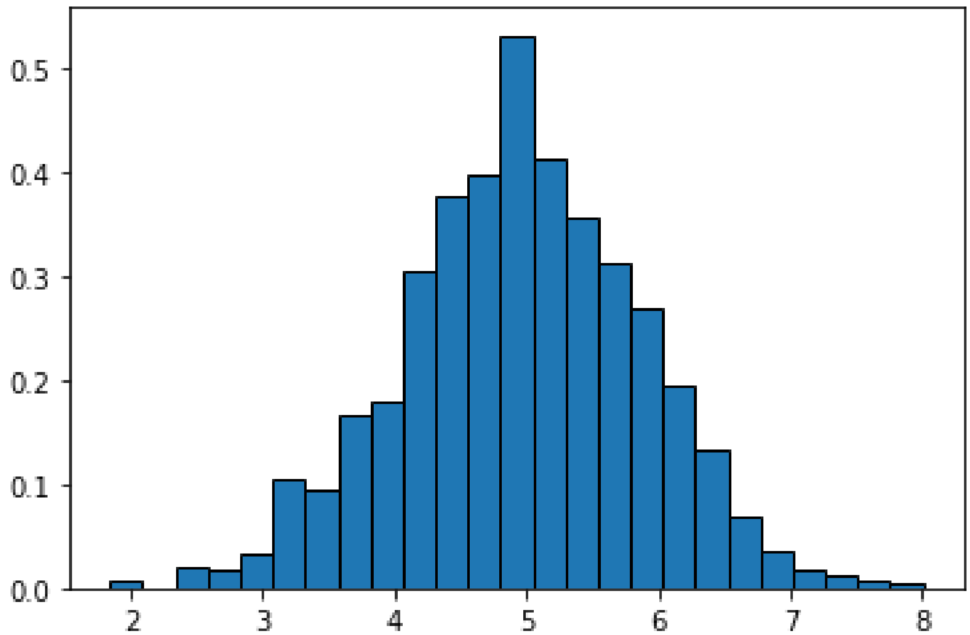
- Complete the following `plot_sum_demo` function. This function returns a vector of length `v_length`, which results from the sum of `N` randomly generated vectors using an uniform distribution $[0, 1)$. Each random vector should have the same length (for example `v_lenght=100`).
- Inside the function, plot the corresponding histogram.
- Finally, check that the resulting figure has the shape of a gaussian.

[11]:

```
def plot_sum(v_length, N):

    #create the vector for storing the sums
    sum_samples = np.zeros(v_length)

    # Generate N vectors of samples and sum them within sum_samples
    for _ in range(0, N):
        sum_samples += random.rand(v_length)

    # Plot the resultant histogram
    plt.hist(sum_samples,
             bins=25, density=True,
             stacked=True, edgecolor='black')
```

[12]:

```
# RUN
v_length = 1000
N = 10
plot_sum(v_length, N)
```



Now play a bit with the number of randomly generated vectors

[13]:
```
interact(plot_sum, v_length=fixed(v_length), N=(0, 25, 1));
```

```
interactive(children=(IntSlider(value=12, description='N', max=25), Output()), _dom_classes=('w
```

1.2.3 Product of gaussians

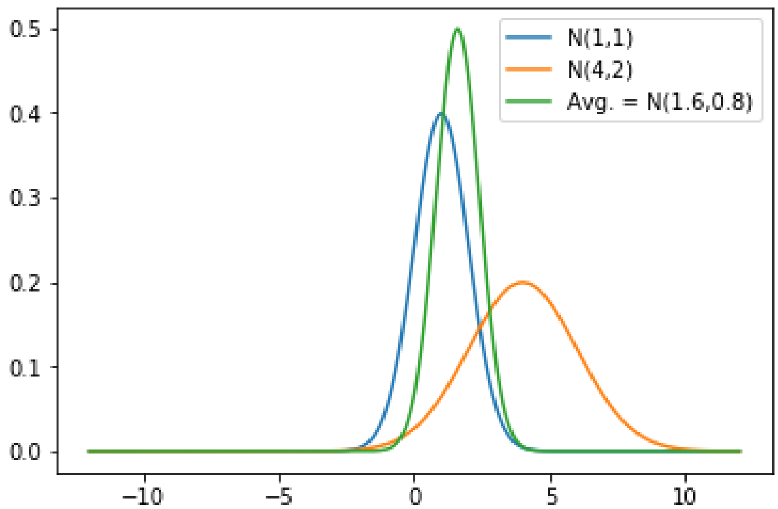
The weighted sum of two gaussians, results in a random variable which its the product of both. This product of 2 gaussians is defined as:

$$N\left(\frac{\sigma_2^2\mu_1 + \sigma_1^2\mu_2}{\sigma_1^2 + \sigma_2^2}, \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\right)$$

1.2.4 ASSIGNMENT 5: Multiplying gaussians

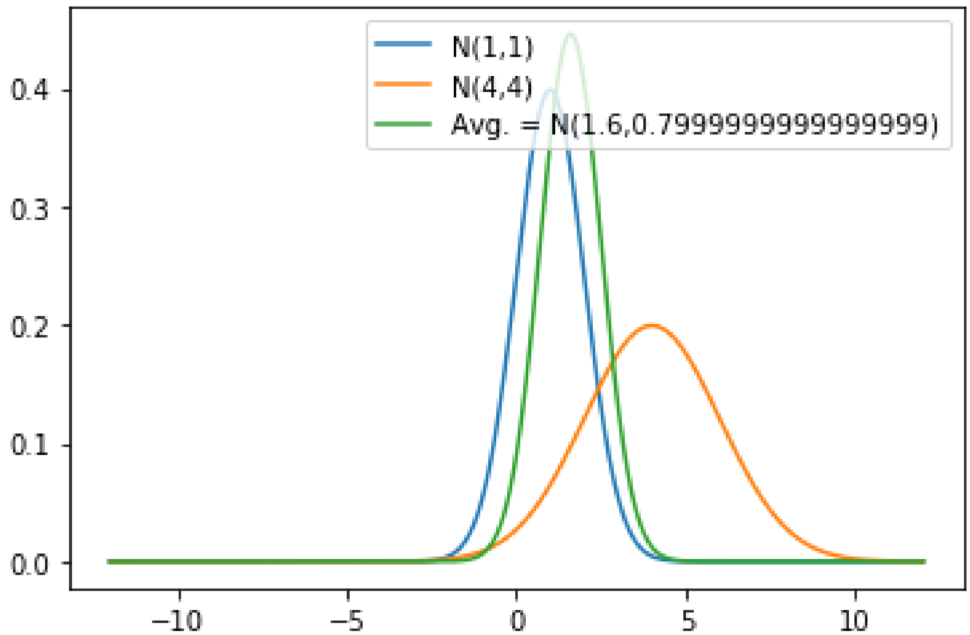
Complete the following function to compute the product of two gaussians distributions.

Draw the result and check that corresponds to the formula above playing with different distributions.



[14]:
```
def gaussians_product(mu1,mu2,sig1,sig2,x):

    var1, var2 = np.power(sig1,2),np.power(sig2,2) # Get the variances from the
 ↪standar deviations

    X = np.arange(-12, 12, 1/x)
    pdf1 = stats.norm(loc=mu1, scale=sig1).pdf(X)
    pdf2 = stats.norm(loc=mu2, scale=sig2).pdf(X)

    plt.plot(X, pdf1, label='N({},{})'.format(mu1, sig1 ** 2))
    plt.plot(X, pdf2, label='N({},{})'.format(mu2, sig2 ** 2))

    # Get the parameters defining the gaussian distribution resulting from
 ↪their product
    mu3 = (var2*mu1 + var1*mu2) / (var1+var2)
    sig3 = np.sqrt((var1*var2)/(var1+var2))
    c = stats.norm(loc=mu3, scale=sig3).pdf(X)

    plt.plot(X, c, label='Avg. = N({},{})'.format(mu3, sig3 ** 2))
    plt.legend()
```

[15]:
```
mu1, sig1 = 1, 1
mu2, sig2 = 4, 2
x = 1000

gaussians_product(mu1,mu2,sig1,sig2,x)
```



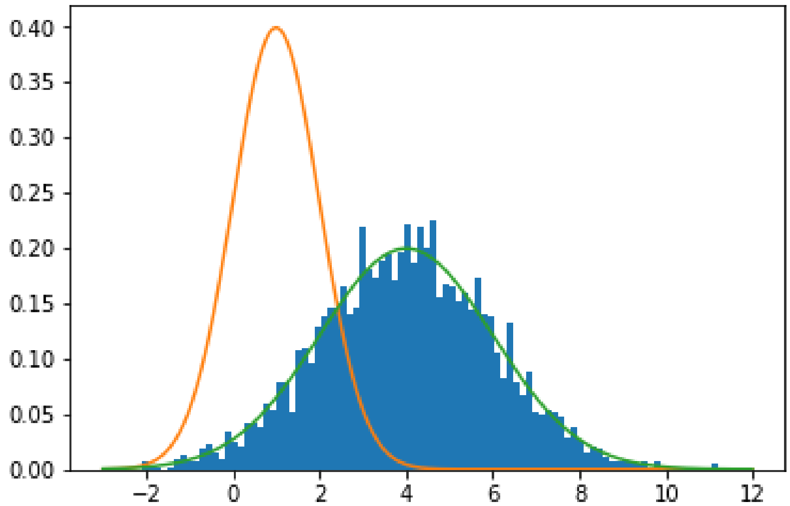
1.2.5 Linear transformation of gaussian random variables.

Property. The gaussian distributions are closed under linear transformations, i.e. when we apply a sum or product to normal random variables, the result is also a normal random variable.

This is also a remarkable property, for example in the field of robotics we can *operate normally over random distributions* as long as we only use linear functions. Otherwise, if we are in need to apply a *non-linear transformation* (e.g. sine, cosine, ...), the resulting probability distribution *will not correspond to any Gaussian pdf*, causing additional complications in the process.

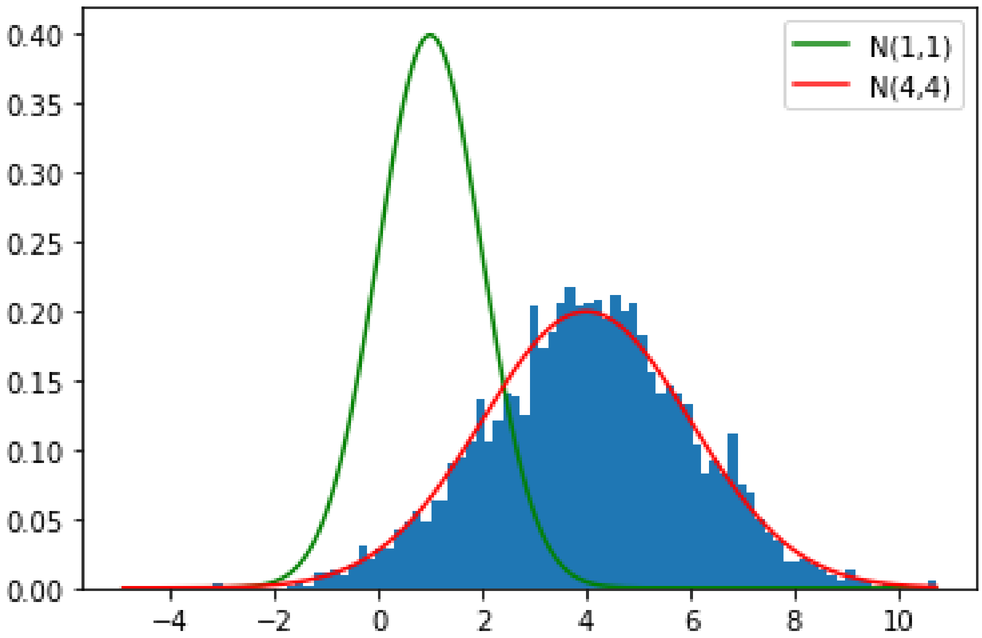
1.2.6 ASSIGNMENT 6: Applying linear transformations

- Generate a number `n_samples` of random samples from the dist. $N(1,1)$.
- Then transform it following the expression $y = a * x + b$ and plot the result for $a = b = 2$.
- Finally, draw on top the pdf of $N(4,4)$ and check that both are the same.



[16]:
```
def linear_transformation(n_samples, a, b):
    """Apply lineal transform. Generating n_samples samples from N(1,1)"""

    # Generates n_samples from N(1,1)
    mu = 1
    stdv = 1
    samples = stats.norm(loc=mu, scale=stdv).rvs(n_samples)

    samples_2 = a*samples+b # Apply the linear transformation to the samples

    # Plot histogram (blue bars)
    n, bins, patches = plt.hist(samples_2, bins=90, density=True, stacked=True)

    delta = 1/samples.size
    X = np.arange(bins[0], bins[-1], delta)
    A = stats.norm(loc=1, scale=1).pdf(X) # Evaluate N(1,1) in X
    B = stats.norm(loc=a*mu+b, scale=stdv*a).pdf(X) # Evaluate the resultant␣
 ↪distribution in X

    # Show results
    plt.plot(X, A, color='green', label='N({},{})'.format(mu, stdv ** 2))
    plt.plot(X, B, color='red', label='N({},{})'.format(a*mu+b, (stdv*a) ** 2))
    plt.legend()
```

[17]:
```
# RUN
n_samples = 3000
a = 2
b = 2
linear_transformation(n_samples, a, b)
```



Now play a bit with different values for a and b.

[18]:
```
interact(linear_transformation, n_samples=fixed(n_samples), b=(-5, 5, 1), a=(1,␣
↪10, 1))
```

```
interactive(children=(IntSlider(value=5, description='a', max=10, min=1), IntSlider(value=0, de
```

[18]:
```
<function __main__.linear_transformation(n_samples, a, b)>
```

1.3 2.1.3 Bidimensional normal distribution

Most useful applications of gaussian distributions does not only look at individual distributions or variables, but an assortment of random distributions which can be dependant to each other. Some examples of these *multidimensional distributions* we will use in following exercises are: the pose of a robot (x, y, θ), an observation from a series of range sensors $([z_0, z_1, \ldots, z_n])$, among others.

In the specific case of Gaussian distributions they present certain key differences:

- The *mean* (μ) now it contains a vector of n values $([\mu_1, \mu_2, \ldots, \mu_n]')$. Its dimensionality/shape is $(n \times 1)$, i.e. is a vertical vector.
- The *covariance* (now referred as Σ) is a full-blown matrix of shape $(n \times n)$. The case being, now we need to express the relations (i.e. dependence) of each variable to the rest.

[19]:
```
# Imports
from numpy import linalg
import sys
sys.path.append("..")
```

```python
from utils.PlotEllipse import PlotEllipse
```

1.4 Sum of bidimensional random variables

In this exercise, we will take a look at how gaussians beheave when we sum 2 multidimensional random variables (*RV*).

Given the sum of 2 multidimensional gaussian RVs (X_1, X_2), the resulting RV (X_3) also follows a gaussian distribution defined as:

$$\left.\begin{array}{l} X_1 \sim N(\mu_1, \Sigma_1) \\ X_2 \sim N(\mu_2, \Sigma_2) \\ X_3 = X_1 + X_2 \end{array}\right\} X_3 \sim N(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2)$$

1.4.1 ASSIGNMENT 7: Summing linear transformations

1. Generate and draw `n_samples` random samples from 2 different bidimensional dists. $N_1 = N(\mu_1, \Sigma_1)$ y $N_2 = N(\mu_2, \Sigma_2)$. The *mean* (μ_n) is a vector of dimension (2×1) and the *covariance* (σ_n) a matrix (2×2). They represent the mean and covariance of each dist. respectively. Use the function `multivariate_normal` from the module **scipy.stats**.
2. Draw both ellipses associated with each distribution. Use `PlotEllipse()` from the utils library that comes with these notebooks.
3. Sum both samples and draw the ellipse $x_3 \sim N(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2)$

WARN: When passing the mean to the `PlotEllipse()` function, it takes a vector (2×1), whereas `multivariate_normal()` takes a flat array (1×2).

Example

Results for an example:

```python
n_samples = 500

mean1 = np.vstack([1, 0])
sigma1 = np.array([[3, 2], [2, 3]])
mean2 = np.vstack([2, 3])
sigma2 = np.array([[2, 0], [0, 1]])
```

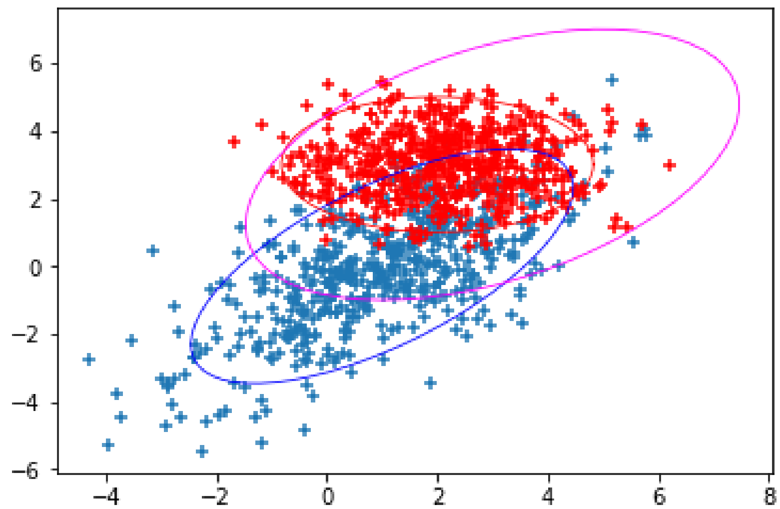
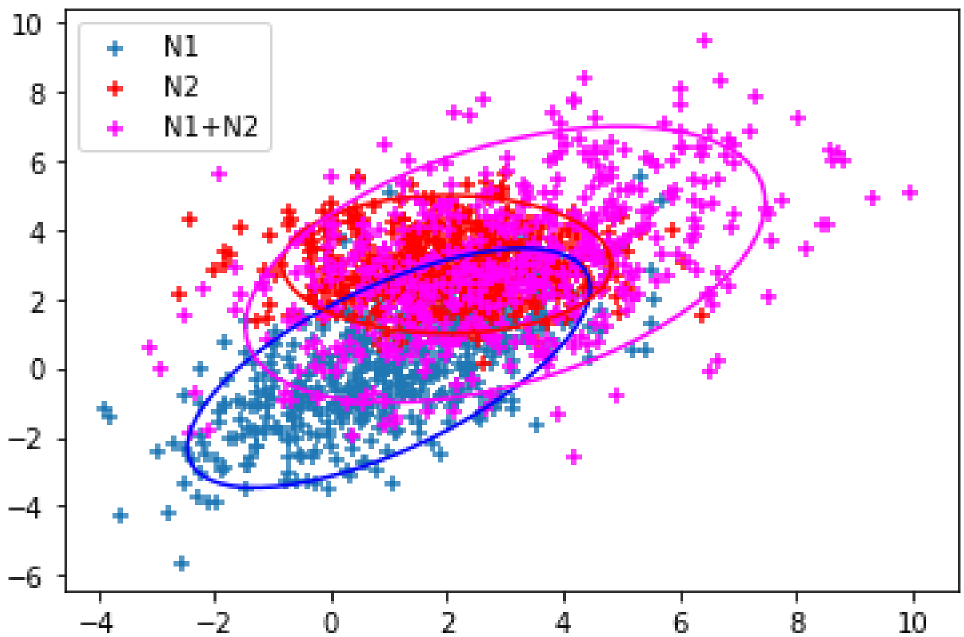


Fig. 1: Distribution of the sum of two RVs (in blue and red)

[20]:
```
def sum_of_rvs(mean1,sigma1,mean2,sigma2,n_samples):

    fig, ax = plt.subplots()

    # Build the normal distributions
    pdf1 = stats.multivariate_normal(mean=mean1.flatten(), cov=sigma1) # Hint:␣
↪you have to use .flatten()
    pdf2 = stats.multivariate_normal(mean=mean2.flatten(), cov=sigma2)

    # Generate n_samples from them
    rvs1 = pdf1.rvs(size=n_samples)
    rvs2 = pdf2.rvs(size=n_samples)

    # Draw samples as crosses
    plt.scatter([i[0] for i in rvs1], [i[1] for i in rvs1], marker='+',␣
↪label="N1")
    plt.scatter([i[0] for i in rvs2], [i[1] for i in rvs2], marker='+',␣
↪color='red', label="N2")

    # Draw ellipses
    mult = 2
    PlotEllipse(fig, ax, mean1, sigma1, mult, color='blue')
    PlotEllipse(fig, ax, mean2, sigma2, mult, color='red')

    # Compute and draw N1 + N2
    rvs3 = stats.multivariate_normal(mean=mean1.flatten()+mean2.flatten(),␣
↪cov=sigma1+sigma2).rvs(size=n_samples)
    plt.scatter(rvs3[:,0],rvs3[:,1], marker='+',color='magenta', label="N1+N2")
    PlotEllipse(fig, ax, mean1+mean2, sigma1+sigma2, mult, color='magenta')
```

```
    plt.legend()
```

[21]:
```
n_samples = 500
mean1 = np.vstack([1, 0])
sigma1 = np.array([[3, 2], [2, 3]])
mean2 = np.vstack([2, 3])
sigma2 = np.array([[2, 0], [0, 1]])

sum_of_rvs(mean1,sigma1,mean2,sigma2,n_samples)
```



1.5 Product of gaussian pdfs

The product of two gaussian distributions (*pdfs*) is also a gaussian distribution. This distribution corresponds to the weighted mean of samples from that same *pdfs*.

Given two gaussian distributions $N_1 \sim N(\mu_1, \Sigma_1)$ and $N_2 \sim N(\mu_2, \Sigma_2)$, the resulting gaussian N_3 is defined as:

$$\Sigma_3 = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1} \mu_3 = \Sigma_3 \left(\Sigma_2^{-1} \mu_1 + \Sigma_1^{-1} \mu_2 \right) N_3 = (\mu_3, \Sigma_3) \tag{1}$$

1.5.1 ASSIGNMENT 8: Multiplying bidimensional distributions

Given the two samples from the previous exercise, draw the ellipse (corresponding gaussian) that represents their weighted mean.

Example

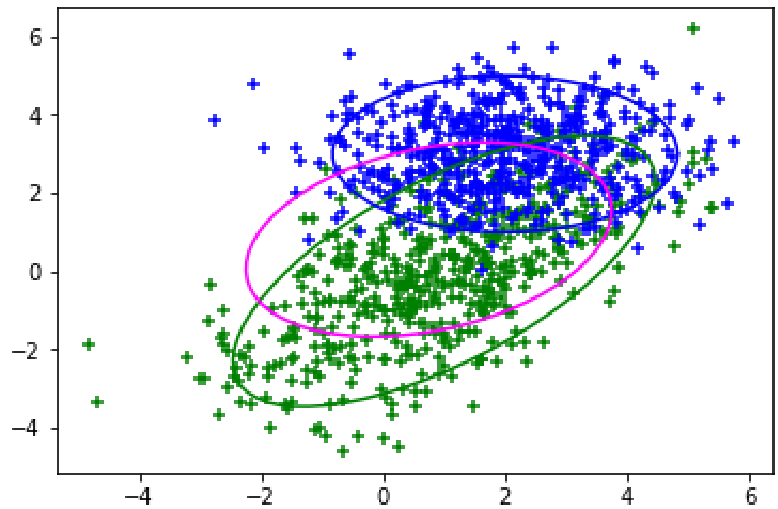
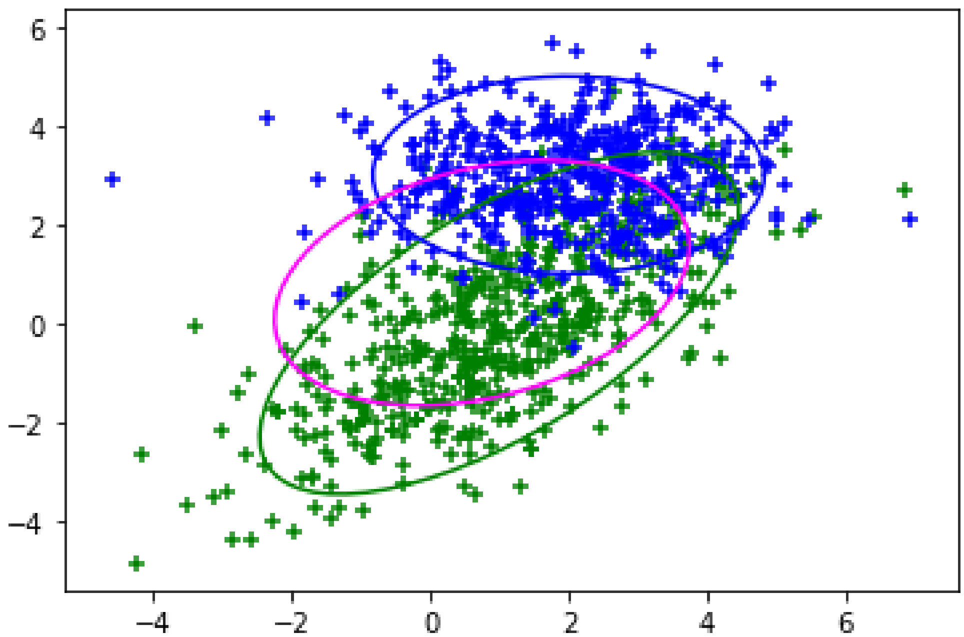


Fig. 2: Product of two pdfs (in blue and green)

[22]:
```
def bidimensional_gaussians_product(mean1,sigma1,mean2,sigma2,n_samples):

    fig, ax = plt.subplots()

    # Build the normal distributions
    pdf1 = stats.multivariate_normal(mean=mean1.flatten(), cov=sigma1)
    pdf2 = stats.multivariate_normal(mean=mean2.flatten(), cov=sigma2)

    # Generate n_samples
    rvs1 = pdf1.rvs(size=n_samples)
    rvs2 = pdf2.rvs(size=n_samples)

    # Draw the samples
    plt.scatter([i[0] for i in rvs1], [i[1] for i in rvs1], marker='+',␣
 ↪color='green')
    plt.scatter([i[0] for i in rvs2], [i[1] for i in rvs2], marker='+',␣
 ↪color='blue')

    # Calculate average of distributions
    invs1 = linalg.inv(sigma1) # Hint use linalg.inv
    invs2 = linalg.inv(sigma2)

    sigma3 = linalg.inv(invs1+invs2)
    mean3 = sigma3 @ (invs2 @ mean1 + invs1 @ mean2) # Hint: use the @ operator

    # Plot the ellipses
    mult = 2
    PlotEllipse(fig, ax, mean1, sigma1, mult, color='green')
    PlotEllipse(fig, ax, mean2, sigma2, mult, color='blue')
```

```
    PlotEllipse(fig, ax, mean3, sigma3, mult*1.5, color='magenta')
```

[23]:
```
n_samples = 500
mean1 = np.vstack([1, 0])
sigma1 = np.array([[3, 2], [2, 3]])
mean2 = np.vstack([2, 3])
sigma2 = np.array([[2, 0], [0, 1]])

bidimensional_gaussians_product(mean1,sigma1,mean2,sigma2,n_samples)
```



1.5.2 Linear transformation of normal RVs

As we mentioned at the start of this unit, when we linearly transform a gaussian random variable, the result is still a gaussian. This is a very desirable property to have, as it allows us to operate normally, as long as the functions are linear.

1.5.3 ASSIGNMENT 9: Applying linear transformation to bidimensional distributions

Using the previous samples x_1, check that the transformation $x_5 = A * x_1 + b$ results in a normal dist. $N(A\mu_1 + b, A\Sigma_1 A^T)$. Given the matrices `A` and `b` in the code below.

Example

Example of the result at scale=2.5 and the values given below:

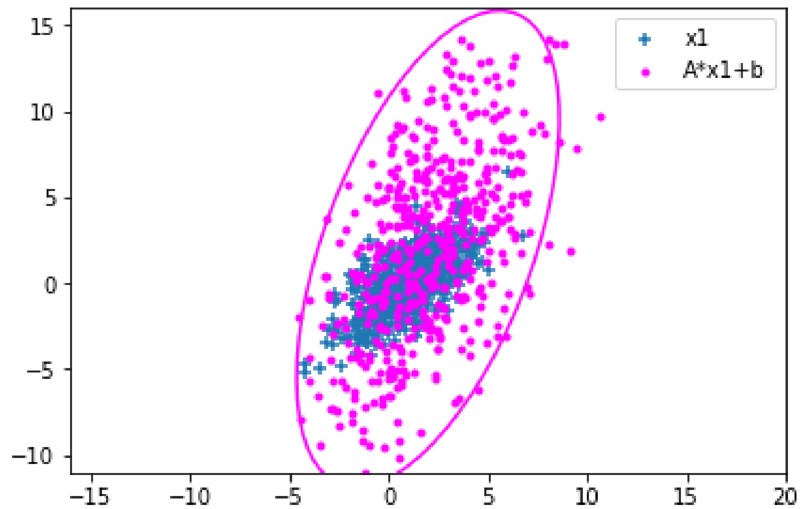
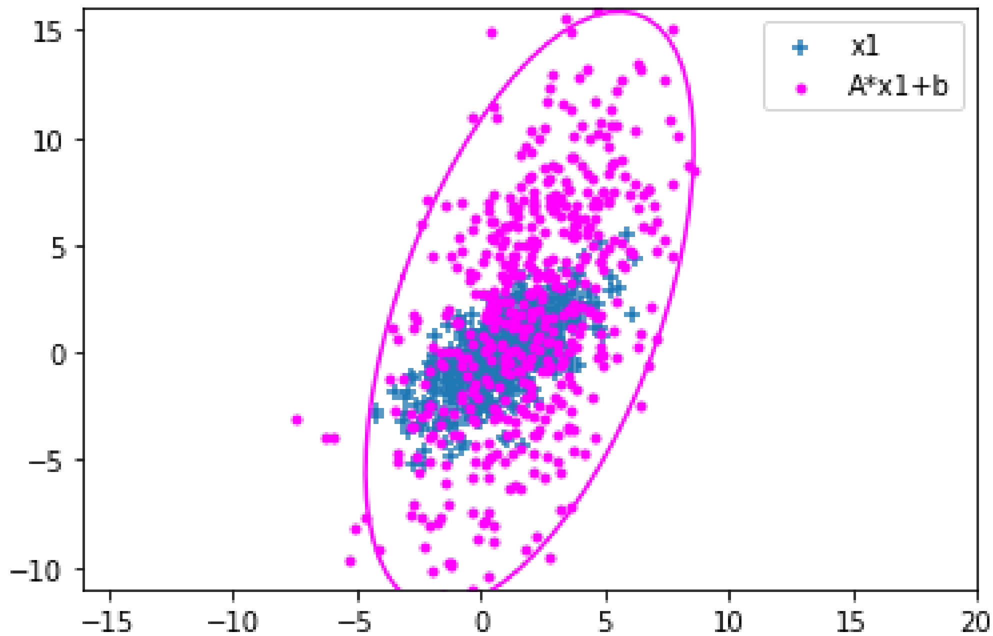


Fig. 3: Linear transformation of RVs. Original samples (in blue) and results (in magenta)

[24]:
```
def bidimensional_linear_transform(mean1,sigma1,mean2,sigma2,n_samples):

    fig, ax = plt.subplots()

    # Define the linear transformation
    A = np.array([[-1, 2], [2, 1.5]])
    b = np.vstack([3, 0])

    # Build distribution
    pdf1 = stats.multivariate_normal(mean1.flatten(), sigma1)

    # Draw samples from it
    rvs1 = pdf1.rvs(n_samples).T

    # Show the samples
    ax.set_xlim((-16, 20))
    ax.set_ylim((-11, 16))
    ax.scatter(rvs1[0], rvs1[1], marker='+', label="x1")

    # Apply linear transformation transformacion lineal
    x5 = A@rvs1+b # Hint: use the @ operator

    # Show the new samples and its ellipse
    ax.scatter(x5[0], x5[1], marker='.', color='magenta', label='A*x1+b')
    PlotEllipse(fig, ax, A@mean1+b, A@sigma1@(A.T), 2.5, color='magenta')
    ax.legend()
```

[25]:
```
n_samples = 500
mean1 = np.vstack([1, 0])
sigma1 = np.array([[3, 2], [2, 3]])
```

```
mean2 = np.vstack([2, 3])
sigma2 = np.array([[2, 0], [0, 1]])

bidimensional_linear_transform(mean1,sigma1,mean2,sigma2,n_samples)
```



1.6 Student discussion

In the cell below, discuss what has been done in the notebook, what you have found interesting, or any other relevant thought.

Since we are going to study a probabilistic approach to robotics, this introduction to some of the properties of Gaussian distributions is essential to better understand our next topics. The Central Limit Theorem justifies why these distributions are important: many variables of our robot will follow a normal distribution such as its position, the positions detected by the sensors, etc. The results that we get after summing, multiplying, and linearly transforming them help us deal with their PDFs better.