Contents

Assignment 1

Computational Intelligence, SS2020

Team Members		
Last name	First name	Matriculation Number
Reindl	Hannes	01532129
Samer	Philip	01634718
Rösel	David	01634719

1 Introduction

The aim of this assignment is to estimate the position of an agent based on (noisy) distance measurements from $N_A = 4$ anchors.

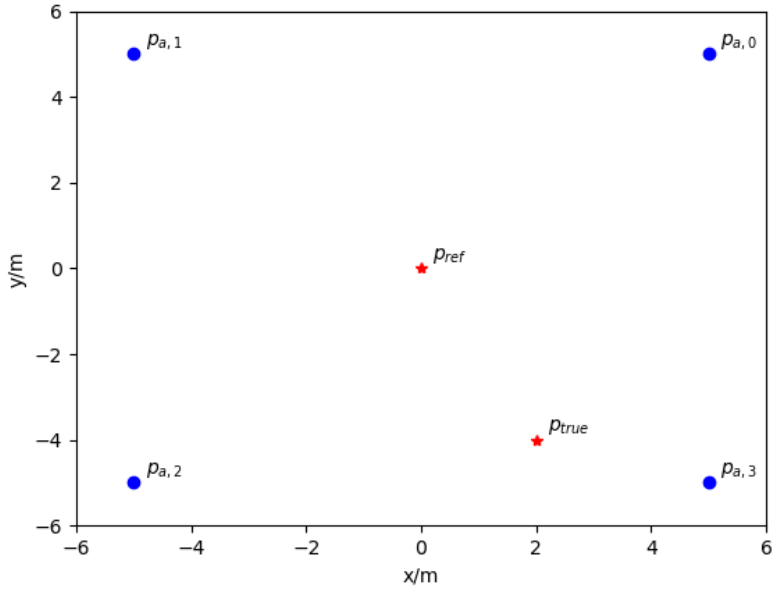


Figure 1: Sketch of anchors

Figure 1 shows the 4 agents shown by blue dots, a reference point p_{ref} to estimate the model parameters and an 'unknown' point p_{true}. The coordintes of the 4 agents are

$$\mathbf{p}_{a_0} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \qquad \mathbf{p}_{a_1} = \begin{bmatrix} -5 \\ 5 \end{bmatrix} \qquad \mathbf{p}_{a_2} = \begin{bmatrix} -5 \\ -5 \end{bmatrix} \qquad \mathbf{p}_{a_3} = \begin{bmatrix} 5 \\ -5 \end{bmatrix} \tag{1}$$

2 Maximum Likelihood Estimation of Model Parameters

2.1 Find the exponential distribution in scenario 2

To derive the exponential distributed anchor, we displayed a histogram of the measurements values for all four anchors.

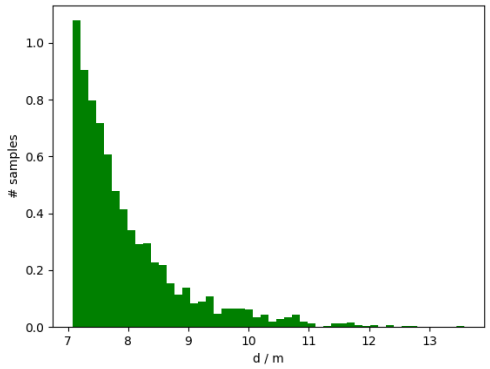


Figure 2: Histogram 0-th anchor

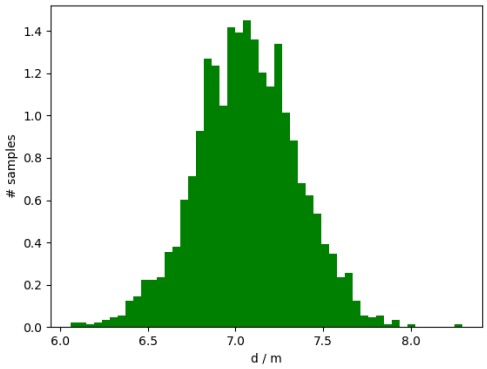


Figure 3: Histogram 1-st anchor

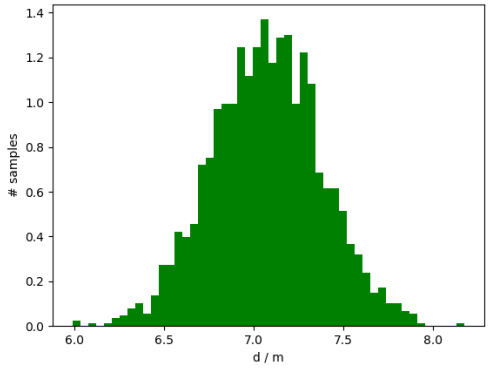


Figure 4: Histogram 2-nd anchor

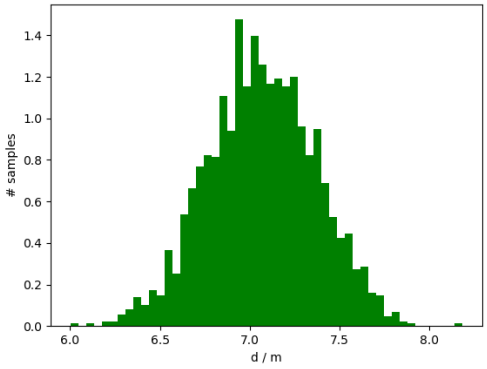


Figure 5: Histogram 3-rd anchor

From these plots we can see, that the 0-th anchor has an exponential distribution which starts at the distance $d = 7$ m following the given formula.

$$p(\tilde{d_n}(a_i, \mathbf{p}|\mathbf{p}) = \begin{cases} \lambda_i \cdot e^{\frac{(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p}))^2}{2\sigma_i^2}} & \text{for } \tilde{d_n}(a_i, \mathbf{p}) \geq d(a_i, \mathbf{p} \\ 0 & \text{else} \end{cases} . \tag{2}$$

Further to fully automatize this process, the function `stats.mstats.normaltest()` delivers two parameters depending on the distribution. Since we only need to distinguish exponential and Gaussian distribution a simple if statement comparing the calculated p-value to a given α does the job. The function suggested to use $\alpha = 0.001$ which worked for our measurements.

2.2 Analytically derivation for the maximum likelihood solution for the exponential distribution

To get the value of lambda you have to take the log likelihood function and get the maximum of it. This is done by differentiating with lambda and setting it to zero. This is shown in the following part:

$$p(\tilde{d}_n(a_i, \mathbf{p})|\mathbf{p}) = \lambda_i e^{-\lambda_i(\tilde{d}_n(a_i,\mathbf{p}) - d(a_i,\mathbf{p}))}$$

$$\begin{aligned} p(x_1, ..., x_N|\underline{\Theta}) &= p_1(x_1|\underline{\Theta}_1) \ \cdot \ \ ... \ \cdot \ \ p_N(x_N|\underline{\Theta}_N) \\ &= p(x_1|\underline{\Theta}) \ \cdot \ \ ... \ \cdot \ \ p(x_N|\underline{\Theta}) \\ &= \prod_{i=1}^{N} \ p(x_i \mid \underline{\Theta}) \end{aligned}$$

$$\begin{aligned} p(x_1, ..., x_N|\underline{\Theta}) &= \prod_{i=1}^{N} \ \lambda_i e^{-\lambda_i(\tilde{d}_n(a_i,\mathbf{p}) - d(a_i,\mathbf{p}))} \\ &= \lambda^N \cdot e^{(-\lambda \sum_{i=1}^{N}(x_i - \mu))} \quad \text{where } x_i = \tilde{d}_n(a_i, \mathbf{p}) \text{ and } \mu = d(a_i, \mathbf{p})) \\ ln(p(x_1, ..., x_N|\underline{\Theta})) &= N \cdot ln(\lambda) - \lambda \sum_{i=1}^{N}(x_i - \mu) \\ \frac{dln(p(x_1, ..., x_N|\underline{\Theta}))}{d\lambda} &= \frac{N}{\lambda} - \sum_{i=1}^{N}(x_i - \mu) \stackrel{!}{=} 0 \\ \lambda &= \frac{N}{\sum_{i=1}^{N}(x_i - \mu)} \end{aligned}$$

For sigma squared we took the formula as it was calculated in the lecture.

$$\sigma^2 = \frac{1}{N} \cdot \sum_{i=1}^{N}(x_i - \mu)^2$$

2.3 Parameter estimation for all anchors

In the following table there are the values of the variance for all anchors and all scenarios listed. In order to properly display the table we had to cut some decimals.

Anchor Nr.	Scenario1	Scenario2	Scenario2 w.o. Anc.Nr.1	Scenario3
1	$\sigma^2 = 0.0909$	$\lambda = 1.1210$	$\lambda = 1.1210$	$\lambda = 1.0797$
2	$\sigma^2 = 0.0884$	$\sigma^2 = 0.0837$	$\sigma^2 = 0.0837$	$\lambda = 1.0888$
3	σ^2 =0.0871	$\sigma^2 = 0.0933$	$\sigma^2 = 0.0933$	$\lambda = 1.1109$
4	$\sigma^2 = 0.0925$	$\sigma^2 = 0.0899$	$\sigma^2 = 0.0899$	$\lambda = 1.1372$

In the following table the mean error an error variance are displayed for each scenario.

	Scenario1	Scenario2	Scenario2 w.o. Anc.Nr1	Scenario 3
Error Mean:	0.2779	0.6402	0.3988	1.2658
Error Variance:	0.0216	0.2745	0.0541	0.9438

3 Estimation of the Position

3.1 Least-Squares Estimation of the Position

Show that the Least-Square estimator is equivalent to the maximum likelihood estimator.

$$\tilde{\mathbf{p}}_{LS}(n) = \underset{\mathbf{p}}{\operatorname{argmin}} \sum_{i=0}^{N-1} \left(\tilde{d_n}(a_i, \mathbf{p}) - d(a_i, \mathbf{p}) \right)^2 \tag{3}$$

The maximum likelihood estimator is given by (with iid assumption):

$$\tilde{\mathbf{p}}_{ML}(n) = \underset{\mathbf{p}}{\operatorname{argmax}} \prod_{i=0}^{N_A-1} p(\tilde{d_n}(a_i, \mathbf{p})|\mathbf{p}) \tag{4}$$

Since we are only interested in the argument where the function has its maximum and not the maximum itself, a strict monotonic increasing function does not change the position of the argument. The logarithm is a strict monotonic increasing function for all values of $p(\tilde{d_n}(a_i, \mathbf{p})|\mathbf{p})$

$$\tilde{\mathbf{p}}_{ML}(n) = \underset{\mathbf{p}}{\operatorname{argmax}} \ln \left(\prod_{i=0}^{N_A-1} p(\tilde{d_n}(a_i, \mathbf{p})|\mathbf{p}) \right) \tag{5}$$

$$= \underset{\mathbf{p}}{\operatorname{argmax}} \sum_{i=0}^{N_A-1} \ln \left(p(\tilde{d_n}(a_i, \mathbf{p})|\mathbf{p}) \right) \tag{6}$$

$$\tag{7}$$

Using the rules for the logarithm the product can be simplified to a sum. Now we input the function of a Gaussian distribution

$$\tilde{\mathbf{p}}_{ML}(n) = \underset{\mathbf{p}}{\operatorname{argmax}} \sum_{i=0}^{N_A-1} \ln\left(\sqrt{\frac{1}{2\pi\sigma_i^2}} \cdot e^{-\frac{\left(\tilde{d_n}(a_i,\mathbf{p})-d(a_i,\mathbf{p})\right)^2}{2\sigma_i^2}}\right) \tag{8}$$

$$= \underset{\mathbf{p}}{\operatorname{argmax}} \sum_{i=0}^{N_A-1} \ln\left(\sqrt{\frac{1}{2\pi\sigma_i^2}}\right) - \frac{\left(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)^2}{2\,\sigma_i^2} \cdot \ln(e) \tag{9}$$

Since the term in the square root does not depend on $\mathbf{p}$, it does not change the position of argmax and can be elided.

$$\tilde{\mathbf{p}}_{ML}(n) = \underset{\mathbf{p}}{\operatorname{argmax}} \sum_{i=0}^{N_A-1} -\frac{\left(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)^2}{2\,\sigma_i^2} \tag{10}$$

Also the denominator $2\,\sigma_i^2$ is just a constant scaling with respect to $\mathbf{p}$ and can be elided too. Additionally the argmax can be rewritten as $-$argmin:

$$\tilde{\mathbf{p}}_{ML}(n) = -\underset{\mathbf{p}}{\operatorname{argmin}} \sum_{i=0}^{N_A-1} -\left(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)^2 \tag{11}$$

$$= \underset{\mathbf{p}}{\operatorname{argmin}} \sum_{i=0}^{N_A-1} \left(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)^2 = \tilde{\mathbf{p}}_{LS}(n) \tag{12}$$

Under the condition that the data is iid and Gaussian distributed the Maximum likelihood estimator(MLE) is the same as the Least-Square estimator(LSE)

3.2 Gauss-Newton Algorithm for Position Estimation

To use the Gauss-Newton Algorithm you have to calculate the Jacobi Matrix. In the following equations there are the elements calculated for the Jacobi Matrix. Here it is important, that $\tilde{d_n}$ is just a measurement and therefore a constant, which can be cut by differentiating. x_i and y_i are the known values of the anchor coordinates. The values of x and y are calculated with every iteration of the calculation which follows. The initial x and y values are chosen randomly.

$$
\begin{aligned}
[\mathbf{J}(\mathbf{p})]_{i,1} &= \frac{\partial(\tilde{dn}(a_i, \mathbf{p} - d(a_i, \mathbf{p})}{\partial x} \\
[\mathbf{J}(\mathbf{p})]_{i,2} &= \frac{\partial(\tilde{dn}(a_i, \mathbf{p} - d(a_i, \mathbf{p})}{\partial y} \\
d(a_i, \mathbf{p}) &= \sqrt{(x_i - x)^2 + (y_i - y)^2} \\
[\mathbf{J}(\mathbf{p})]_{i,1} &= \frac{x_i - x}{\sqrt{(x_i - x)^2 + (y_i - y)^2}} \\
[\mathbf{J}(\mathbf{p})]_{i,1} &= \frac{x_i - x}{\sqrt{(x_i - x)^2 + (y_i - y)^2}}
\end{aligned}
$$

With every iteration the value gets closer to the original. This will continue up until a maximum number of iterations or until the value change between the iterations falls under a minimum.

$$
\hat{\mathbf{p}}^{(t+1)} = \hat{\mathbf{p}}^{(t)} - \left(\mathbf{J}^T(\hat{\mathbf{p}}^{(t)}) \cdot \mathbf{J}(\hat{\mathbf{p}}^{(t)})\right)^{-1} \cdot \mathbf{J}^T(\hat{\mathbf{p}}^{(t)}) \cdot (d_n(\mathbf{p}) - d(\hat{\mathbf{p}}^{(t)})
$$

Scenario 1: Figure 6 shows the CDF and the estimated points for the "unknown" position.

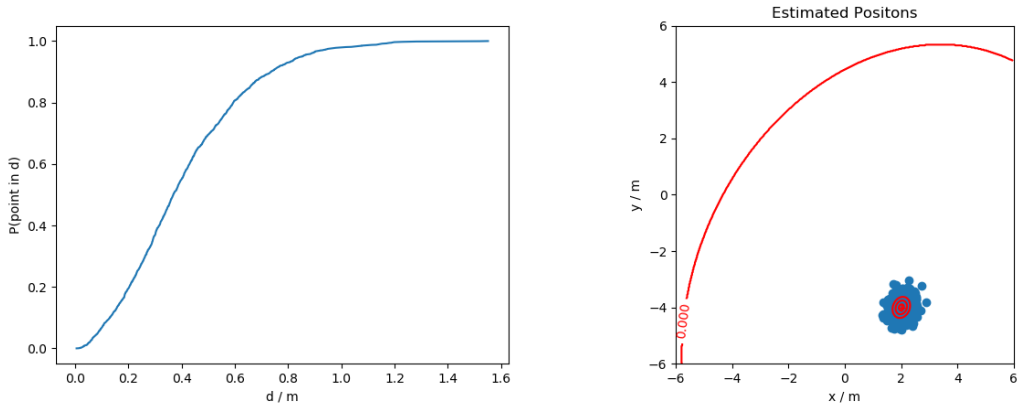


Figure 6: CDF and position Plot for scenario 1

In the position plot *(right plot)*, the estimated coordinates spread in a gaussian way near the true point of

$$
\mathbf{p}_{true} = \begin{bmatrix} 2 \\ -4 \end{bmatrix} \tag{13}
$$

The estimated points are close to the true value and the Least-Square Aproximation using the iterative Gauss-Newton algorithm results in a trustworthy outcome. The CDF

plot allows us to limit how much the result can scatter. For example, 90% of the time we can be sure the result will be within 0.8 m of the true value and close to 100% for 1 m distance.

Scenario 2: In this scenario one of the anchors (the top right anchor) does not cause errors in a Gaussian distributed manner, but in an exponential one.

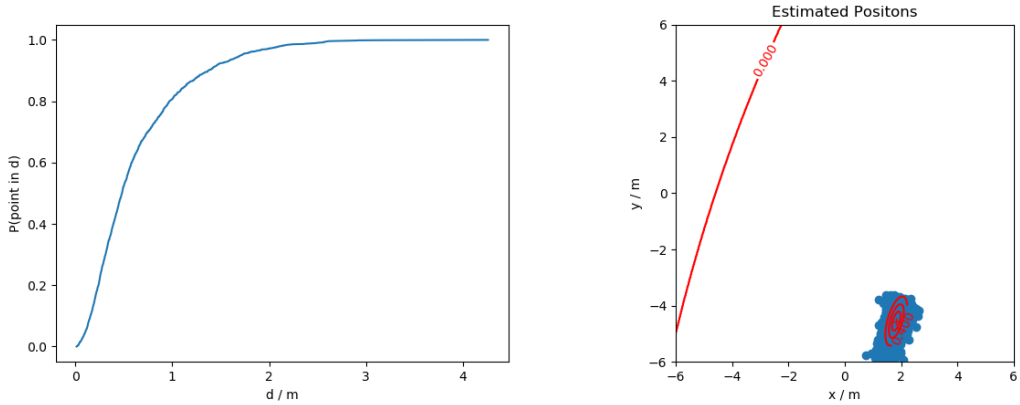


Figure 7: CDF and position Plot for scenario 2

This can be seen in the position plot - the estimated points are not scattered in an elliptic or circular manner anymore. Looking at the scattered points from the top right, it seems the points start at one line, which resembles the harsh start of the exponential distribution (see Figure 8)

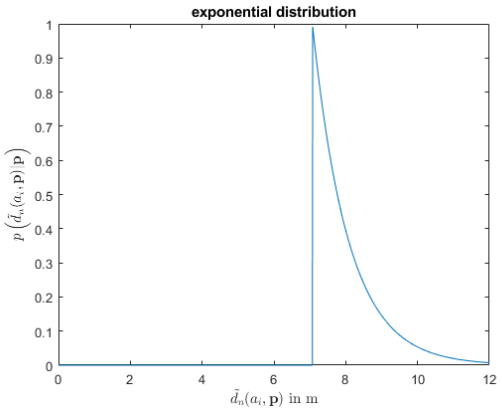


Figure 8: shifted exponential distribution

This causes the estimation to be much worse, which is also reflected in the CDF. It takes a distance of 2 m now to be 90% sure the point is within this range. The Least Squares approximation assumes the anchors to measure in a Gaussian distributed

manner. Since the top-right anchor does not behave like this, the results is much worse than in scenario 1.

Scenario 2, exponential distributed anchor left out Using only the data of the 3 remaining Gaussian distributed anchors the estimation via Least Squares causes better results, although less data is used to estimate the point. The impact of the non-Gaussian distributed anchor causes a lot of error due to the assumption of LS, that the samples are iid and Gaussian distributed. LS was not designed for exponential distributions.

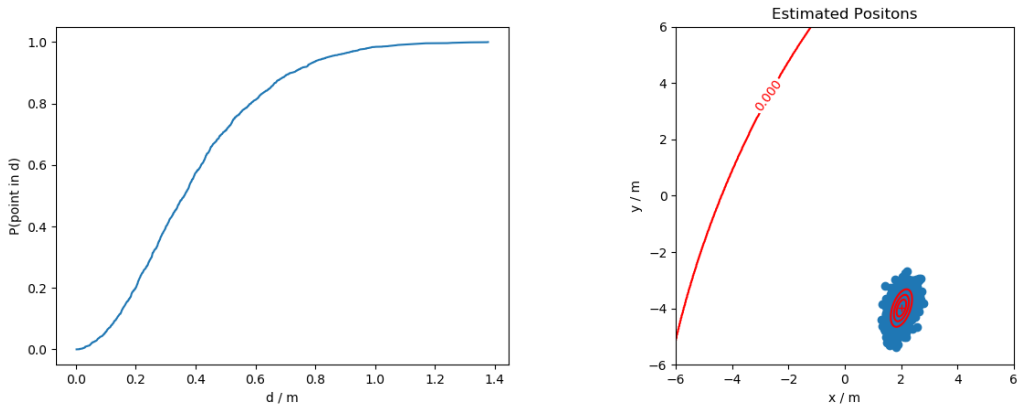


Figure 9: CDF and position Plot for scenario 2 with only 3 Anchors

The estimated points in the position plot of figure 9 now follow a Gaussian distribtuion again and the CDF also delivers a higher certainty with lower values of d.

Scenario 3: In this scenario all of the anchors measurement error are distributed in an exponential manner.

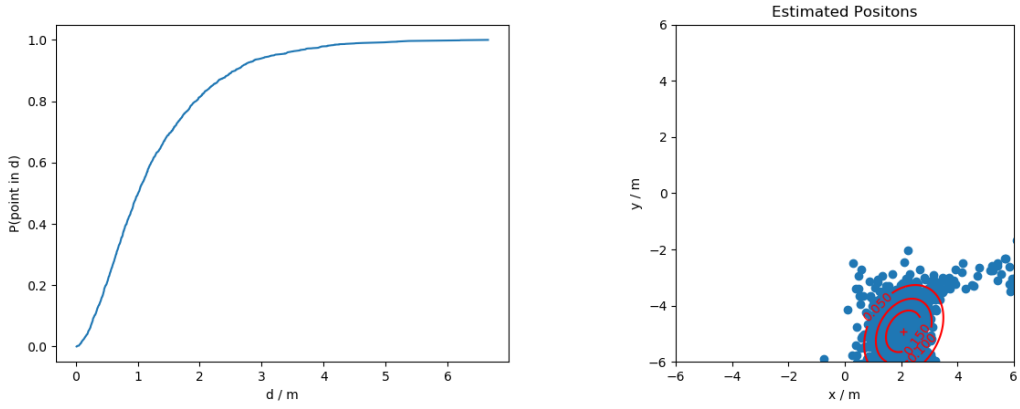


Figure 10: CDF and position Plot for scenario 3

In the position plot of figure 10 the estimated points are much more scattered and then contour lines are not as close together as before. The CDF plot also shows this. With a 4 m distance from the true location it only has about 90% certainty - by far the worst result. Again, the LS algorithm assumes Gaussian distribution and does not adapt to other distributions.
In the following table the mean error an error variance are displayed for each scenario.

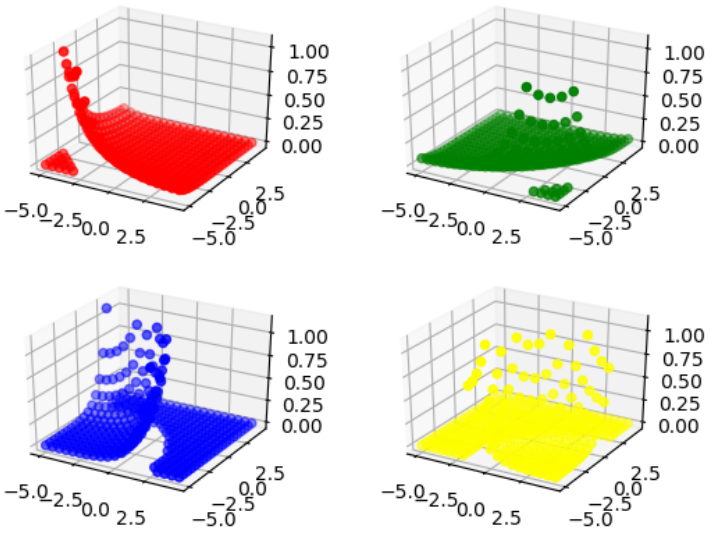
	Scenario1	Scenario2	Scenario2 w.o. Anc.Nr1	Scenario 3
Error Mean:	0.2779	0.6402	0.3988	1.2658
Error Variance:	0.0216	0.2745	0.0541	0.9438

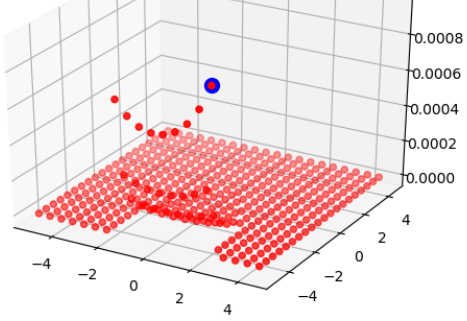
As one can see the mean error between Scenario 2 with or without the exponential anchor is getting lower without the anchor as well as the variance. By far the worst values are in scenario 3 where all the anchors were exponentially distributed. Since the method is meant for Gaussian it is pretty obvious this should happen since as we've shown in 3.1 the Least Squares Estimate is the same as the maximum likelihood for Gaussian.
The function $\text{least}_s quares_G N(p_a nchor, p_s tart, measurements_n, max_i ter, tol)$,

3.3 Numerical Maximum Likelihood Estimation of the Position

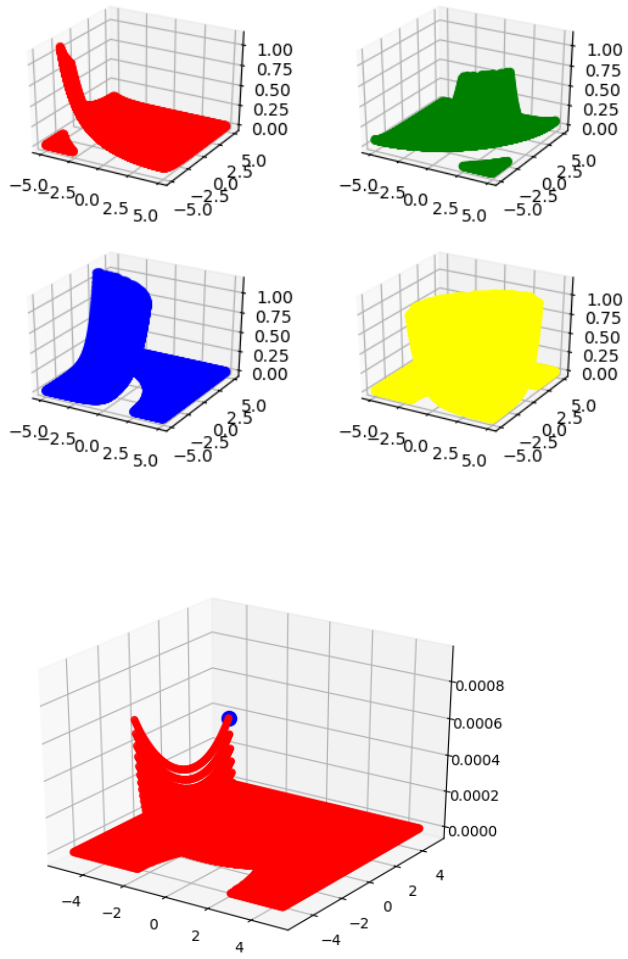
3.3.1 Single Measurement

Here the task was to build the joint likelihood for the first measurement n=0 in accordance of the value of x and y over a 2 dimensional grid.
In the following graph we can see the likelihood from the individual anchors as well as the joint likelihood over the 2 dimensional grid. The resolution was chosen to be 0.5 so that the individual plot points stand out more. On the following page the given resolution of 0.05 is implemented.





In the following graph we can see the likelihood from the individual anchors as well as the joint likelihood over the 2 dimensional grid. As one can see in the region close to



each individual anchor the plot suddenly is equal to zero. The reason for this is, that the likelihood of the exponential is given by the following formula and in those cases dn is no longer greater than d:

$$p(\tilde{d_n}(a_i, \mathbf{p}|\mathbf{p}) = \begin{cases} \lambda_i \cdot e^{\frac{(\tilde{d_n}(a_i,\mathbf{p}) - d(a_i,\mathbf{p}))^2}{2\sigma_i^2}} & \text{for } \tilde{d_n}(a_i, \mathbf{p}) \geq d(a_i, \mathbf{p} \\ 0 & \text{else} \end{cases} .$$

Therefor the likelihood in this region is equal to zero.

It is not very likely that one can find the maximum of this function by using a gradient ascent algorithm. The reason for this is there are two problems. Either it gets stuck in a region were the likelihood is constantly zero for a broad array or it gets the local maximum but not the global. As we did see on the graphs on the last we both have a global and local maximum, as well as wide areas with no change in value of the likelihood. According to the calculation the maximum sits at x=2.5 and y=-5, it is marked blue in the graphs above. The point should sit at x=2 and y =-4 according to the assignment. So it is not at the true position but it currently only depends on one measurement, so the solution is not as precise as one might wish.

3.3.2 Multiple Measurements

Now we want to estimate the position for all N measurements. The computation is the same as in the previous task. To improve the computation-time we switch from loops to vectors, now only the plotting of the 3D graphics take same time. If we compared the least-squares algorithm with the numerical maximum likelihood estimation, the numerical maximum likelihood estimation has a much lower mean error and variance (table). The problem with the numerical maximum likelihood estimation is dependence on the grid resolution. The result changes with the step size of the grid, so a direct comparison is not really fair. Because we only compute the likelihood for a finite amount of points and not continous.

For each measurement cycle we computed the most likely point the agent will be. This was repeated for all 2000 cycles. A counter kept track how often the maximum point was on each grid point and is displayed on the z-axis in figure 11.

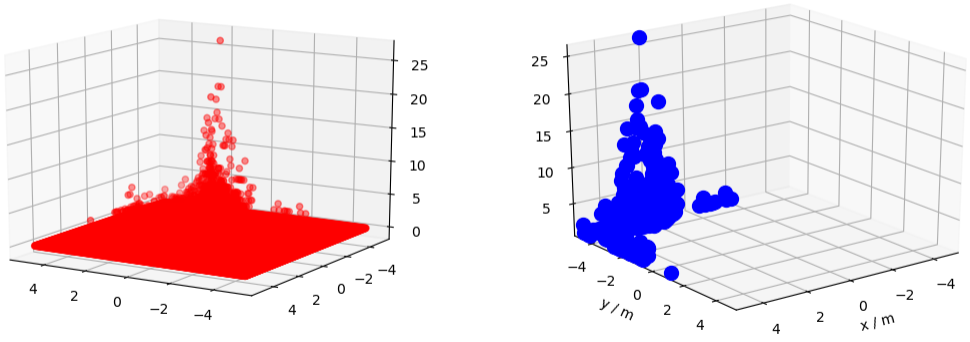


Figure 11: numerical maximum likelihood plot **with** points of 0 occurence *(left)* and **without** *(right)*

The last part was to compute the Bayesian estimation for every position in the grid. The formula for the computation can be found under (14).

$$\hat{\mathbf{P}} = \underset{\mathbf{p}}{\operatorname{argmax}} \;\; p(\tilde{d_n}(a_i, \mathbf{p}|\mathbf{p}) \; p(\mathbf{p}) \tag{14}$$

Like before we visualize the outcome for all measurements with the same grid resolu-

	least-squares	numerical maximum likelihood	Bayesian
Error Mean:	1.2607	0.9085	0.3080
Error Variance:	0.9195	0.4660	0.0134

tion(= 0.05 m). Compared to 11 the summed up most-likelihood-points scatter much less and the mean error is the smallest of all 3 methods. For a better view of the plots, it is recommended to execute the python file.

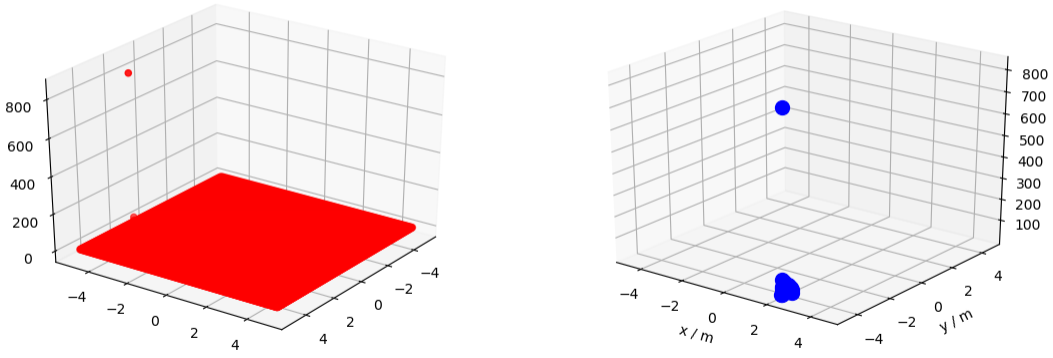


Figure 12: bayesian plot **with** points of 0 occurence *(left)* and **without** *(right)*

In figure 13 the CDF all three methods are displayed. As expected the least-squares method has the worst result. Bayesian on the other hand allows to locate the agent with a 100% certainty within a range of 0.5 meters - even better than least squares when all anchor errors were Gaussian distributed.

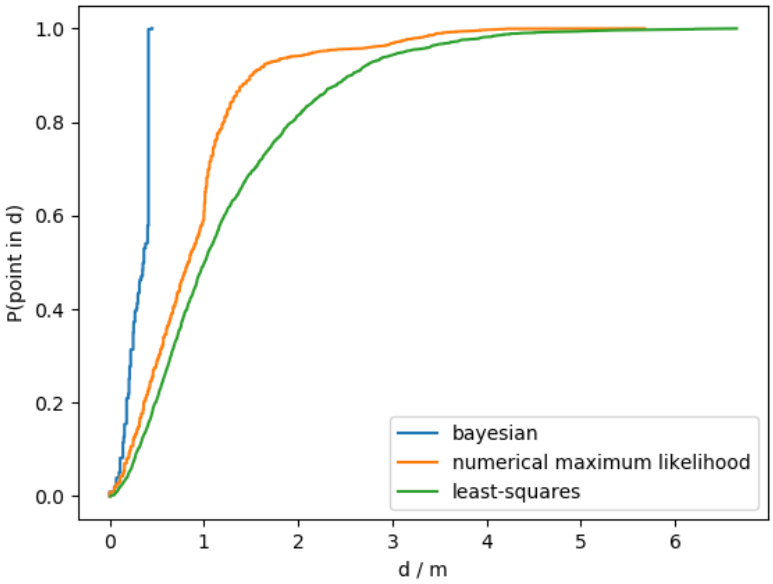


Figure 13: CDF for each estimator

4 Attachment

The pdf file of the given assignment sheet as well as the code can be found on the next pages.

Computational Intelligence SS20
Homework 1
Maximum Likelihood Estimation

Harald Leisenberger

Tutor:	Ismail Geles, ismail.geles@student.tugraz.at
Points to achieve:	20 pts
Extra points:	5* pts
Info hour:	Yet to be announced!
Deadline:	21.04.2020 23:59
Hand-in mode:	Use the **cover sheet** from the website. Submit (i) all **python files (as *.zip)** and (ii) a colored version of **your report (as *.pdf)** at the teachcenter `https://tc.tugraz.at`. (Please name your files *hw1-Familyname1Familyname2Familyname3.pdf* and *hw1-Familyname1Familyname2Familyname3.zip*)

General remarks

Your report must be self-contained and must therefore include all relevant plots, results, and discussions. Your submission will be graded based on:

- The correctness of your results (Is your code doing what it should be doing? Are your plots consistent with what algorithm XY should produce for the given task? Is your derivation of formula XY correct?)
- The depth and correctness of your interpretations (Keep your interpretations as short as possible, but as long as necessary to convey your ideas)
- The quality of your plots (Is everything important clearly visible, are axes labeled, . . . ?)

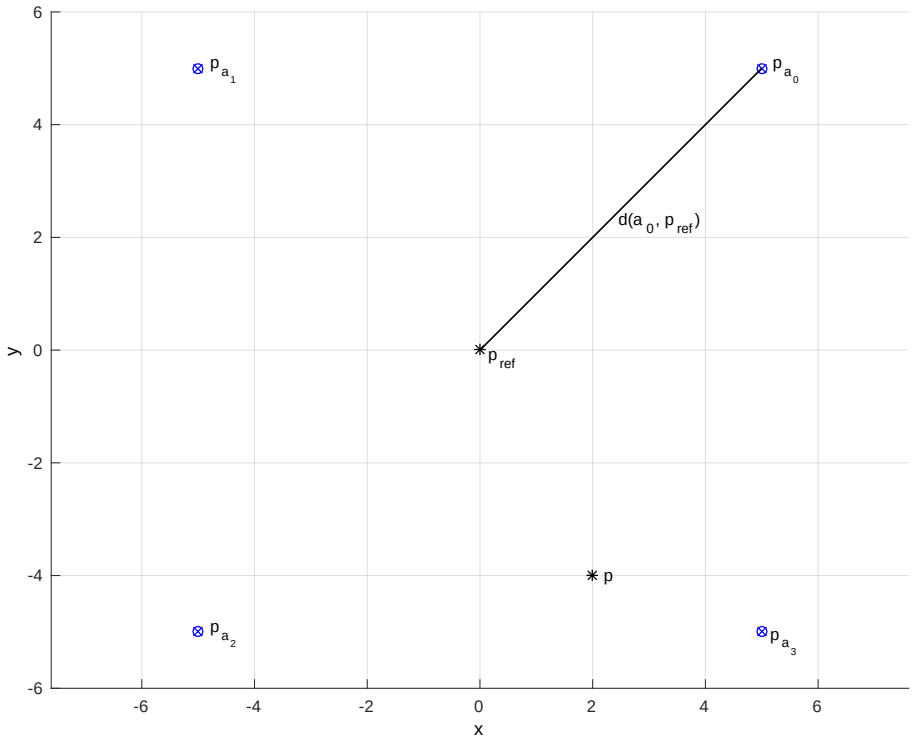


Figure 1: Four anchors (at known positions p_{a_i}, like the satellites in GPS) should be used to estimate the position of an agent, indicated as $\mathbf{p}$. For calibrating the anchors we use $\mathbf{p}_{\text{ref}}$.

1 Introduction

Download `ML.zip` from the course website and unzip. `HW1_skeleton.py` contains a skeleton of the main script that you should implement.

The aim of this assignment is to estimate the position $\mathbf{p} = [x, y]^T$ of an *agent* based on (noisy) distance measurements from $N_A = 4$ *anchors* that are denoted by a_i where $i = 0, \ldots, N_A - 1$. These anchors are shown in Fig 1, and are at the known positions

$$\mathbf{p}_{\text{a}_0} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad \mathbf{p}_{\text{a}_1} = \begin{bmatrix} -5 \\ 5 \end{bmatrix}, \quad \mathbf{p}_{\text{a}_2} = \begin{bmatrix} -5 \\ -5 \end{bmatrix}, \quad \mathbf{p}_{\text{a}_3} = \begin{bmatrix} 5 \\ -5 \end{bmatrix}. \tag{1}$$

2 Maximum Likelihood Estimation of Model Parameters [6 Points]

Because of measurement errors we do not know the *true* distance $d(a_i, \mathbf{p})$ between the anchor a_i and the agent but only have access to N noisy distance measurements $\tilde{d}_n(a_i, \mathbf{p})$. In order to correct for these errors, we need a statistical model that describes the error in the measurements and relates it to the unknown parameter $\mathbf{p}$ that we intent to estimate.

First, we must specify the statistical model for each anchor and estimate the according model parameters. In order to do so we performed $N = 2000$ reference measurements to an agent at known position $\mathbf{p}_{\text{ref}}$ for each anchor; all N measurements form the vector $\tilde{\mathbf{d}}(a_i, \mathbf{p})$. These measurements are contained in the file `reference_i.data` as an $(N \times N_{\text{A}})$ matrix. Note that the dependence between distance and the position is given as the Euclidean distance between an anchor and the agent, i.e.,

$$d(a_i, \mathbf{p}) = \sqrt{(x_i - x)^2 + (y_i - y)^2}. \tag{2}$$

The distance measurements to the i-th anchor are distributed according to one of the following two distributions:

$$\text{Case I (Gaussian):} \quad p\big(\tilde{d}_n(a_i,\mathbf{p})|\mathbf{p}\big) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{\left(\tilde{d}_n(a_i,\mathbf{p})-d(a_i,\mathbf{p})\right)^2}{2\sigma_i^2}} \tag{3}$$

$$\text{Case II (Exponential):} \quad p\big(\tilde{d}_n(a_i,\mathbf{p})|\mathbf{p}\big) = \begin{cases} \lambda_i e^{-\lambda_i\left(\tilde{d}_n(a_i,\mathbf{p})-d(a_i,\mathbf{p})\right)}, & \tilde{d}_n(a_i,\mathbf{p}) \geq d(a_i,\mathbf{p})) \\ 0 & \text{else} \end{cases} \tag{4}$$

We consider three different scenarios where each scenario considers a different assignment of the measurement models to the anchors:

Scenario 1: Measurements of all anchors follow the Gaussian model.

Scenario 2: Measurements of one anchor follows the Exponential model, the other ones follow the Gaussian model.

Scenario 3: Measurements of all anchors follow the Exponential model.

For all scenarios we will estimate the model parameters θ (λ_i or σ_i depending on the model) using the maximum likelihood method, i.e.,

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, p\big(\tilde{\mathbf{d}}(a_i,\mathbf{p})|\mathbf{p}\big) \tag{5}$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{n=0}^{N-1} p\big(\tilde{d}_n(a_i,\mathbf{p})|\mathbf{p}\big). \tag{6}$$

Therefore, you have to perform the following three steps:

- For scenario 2, find out which anchor has exponentially distributed measurements.
- Analytically derive the maximum likelihood solution for the exponential distribution.
- Estimate the parameters for all anchors, i.e., estimate σ_i^2 and λ_i in all 3 scenarios using the maximum likelihood method according to (6).

3 Estimation of the Position

After specification of the statistical model we proceed to the position estimation of the agent at an unknown position $\mathbf{p}$. Note that the estimation of the position must not take into account the true position $\mathbf{p}_{\text{true}}$ but only the measurement data provided in `measurements_i.data` . The true position $\mathbf{p}_{\text{true}}$ is only provided for error evaluation.

Ideally we would like to obtain the ML estimation of the position $\mathbf{p}$ for the n-th measurement of all anchors, i.e.,

$$\hat{\mathbf{p}}_{ML}(n) = \underset{\mathbf{p}}{\operatorname{argmax}} \prod_{i=0}^{N_\text{A}-1} p\big(\tilde{d}_n(a_i,\mathbf{p})|\mathbf{p}\big), \tag{7}$$

$$= \underset{\mathbf{p}}{\operatorname{argmax}}\, p(\tilde{\mathbf{d}}_n(\mathbf{p})|\mathbf{p}) \tag{8}$$

where the vector $\tilde{\mathbf{d}}_n(\mathbf{p})$ contains all $N_\text{A} = 4$ distance measurements.

3.1 Least-Squares Estimation of the Position [3 Points]

The nonlinear dependence in (2), however, prevents a closed form solution of an ML estimator for $\mathbf{p}$. A popular approximation to the maximum likelihood estimator is the least-squares estimator that minimizes the sum of squares of the measurement errors:

$$\begin{aligned}\hat{\mathbf{p}}_{ML}(n) \approx \hat{\mathbf{p}}_{LS}(n) &= \underset{\mathbf{p}}{\operatorname{argmin}} \sum_{i=0}^{N_\mathrm{A}-1} \left(\tilde{d}_n(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)^2 \\ &= \underset{\mathbf{p}}{\operatorname{argmin}} \|\tilde{\mathbf{d}}_n(\mathbf{p}) - \mathbf{d}(\mathbf{p})\|^2, \end{aligned} \tag{9}$$

where $\mathbf{d}(\mathbf{p})$ is a vector that contains the true distances to all anchors (according to (2)). Therefore, you have to perform the following task:

- Show analytically that for scenario 1 (all anchors are Gaussian), the least-squares estimator of the position is equivalent to the maximum likelihood estimator, i.e., you have to prove that (8) equals (9).

3.2 Gauss-Newton Algorithm for Position Estimation [9 Points]

Because of the nonlinear dependence between distance and position in (2), an iterative algorithm is required to obtain the least-squares approximation. We will use the Gauss-Newton algorithm: it requires linearization, i.e., the first-order derivatives, of the measurement error. The collection of all derivatives is the Jacobian matrix $\mathbf{J}(\mathbf{p})$ that has dimension $N_\mathrm{A} \times 2$, with both columns defined according to

$$[\mathbf{J}(\mathbf{p})]_{i,1} = \frac{\partial\left(\tilde{d}_n(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)}{\partial x}, \quad [\mathbf{J}(\mathbf{p})]_{i,2} = \frac{\partial\left(\tilde{d}_n(a_i,\mathbf{p}) - d(a_i,\mathbf{p})\right)}{\partial y}. \tag{10}$$

After an initial guess $\hat{\mathbf{p}}^{(0)}$, the algorithm refines the position-estimate in the t-th iteration as

$$\hat{\mathbf{p}}^{(t+1)} = \hat{\mathbf{p}}^{(t)} - \left(\mathbf{J}^T(\hat{\mathbf{p}}^{(t)})\mathbf{J}(\hat{\mathbf{p}}^{(t)})\right)^{-1} \mathbf{J}^T(\hat{\mathbf{p}}^{(t)}) \left(\tilde{\mathbf{d}}_n(\mathbf{p}) - \mathbf{d}(\hat{\mathbf{p}}^{(t)})\right) \tag{11}$$

The algorithm stops after a previously defined maximum number of iterations or if the change in the estimated position is smaller than a chosen tolerance value γ, i.e., if $\|\hat{\mathbf{p}}^{(t)} - \hat{\mathbf{p}}^{(t-1)}\| < \gamma$.

Implement and evaluate the Gauss-Newton algorithm according the following tasks:

- Implement the Gauss-Newton algorithm to find the least-squares estimate for the position. Therefore, write a function `least_squares_GN(p_anchor,p_start,measurements_n, max_iter, tol)`, which takes the $(2 \times N_\mathrm{A})$ anchor positions, the (2×1) initial position, the $(N_\mathrm{A} \times 1)$ distance measurements, the maximum number of iterations, and the chosen tolerance as input. You may choose suitable values for the tolerance and the maximum number of iterations on your own. The output is the estimated position. [4 Points]
- Run your algorithm for each of the $N = 2000$ independent measurements; choose the initial guess $\hat{\mathbf{p}}^{(0)}$ randomly according to a uniform distribution within the square spanned by the anchor points. Take a closer look at:
  - The mean and variance of the position estimation error $\|\hat{\mathbf{p}}_\mathrm{LS} - \mathbf{p}\|$.
  - Scatter plots of the estimated positions. Fit a two-dimensional Gaussian distribution to the point cloud of estimated positions and draw its contour lines with the provided function `plot_gauss_contour(mu,cov,xmin,xmax,ymin,ymax,title)`. Do the estimated positions look Gaussian?

  - Compare the different scenarios by looking at the cumulative distribution function (CDF) of the position estimation error, i.e. the probability that the error is smaller than a given value. Use the provided function `Fx,x = ecdf(realizations)` for the estimation of the CDF and `plt.plot(x,Fx)` for plotting. What can you say about the probability of large estimation errors, and how does this depend on the scenario?

  Discuss the performance of your estimation based on the above analysis! [4 Points]

- Consider scenario 2: the least-squares estimator is equivalent to the maximum likelihood estimator for Gaussian error distributions. Therefore, neglect the anchor with the exponentially distributed measurements! Compare your results to before! What can you observe (Gaussianity of the estimated positions, probability of large errors, ...)? [1 Point]

3.3 Numerical Maximum Likelihood Estimation of the Position

For non-Gaussian distributed data, the maximum likelihood-estimator is in general not equivalent to the least-squares estimator. In this example, we consider scenario 3 (all anchors have exponentially distributed measurements) and compare the least-squares estimator to a maximum likelihood estimator. As a closed-form solution is not available because of the non-linearity we will evaluate the likelihood numerically and select the point that maximizes it.

3.3.1 Single Measurement [2 Points]

Let us consider the first measurement $n = 0$ for all all N_A anchors now. For this single measurement $\tilde{\mathbf{d}}_0(\mathbf{p})$ you have to perform the following tasks:

- Evaluate the joint likelihood $p\big(\tilde{\mathbf{d}}_0(\mathbf{p})|\mathbf{p}\big)$ according to (4) over a 2-dimensional grid with a resolution of 0.05. Confine the evaluation to the square region enclosed by the anchors.
- Why might it be hard to find the maximum of this function with a gradient ascent algorithm using an arbitrary starting point within the evaluation region?
- Is the maximum at the true position? Explain your answer!

3.3.2 Multiple Measurements [5* Points]

Now, estimate the position for all N measurements. For all individual measurement, compute a numerical maximum likelihood estimate based on the joint likelihood evaluated over the grid, i.e., take the maximum of $p\big(\tilde{\mathbf{d}}_n(\mathbf{p})|\mathbf{p}\big)$ as an estimate. Then perform the following tasks:

- Compare the performance of this estimator (using the same considerations as above) to the least-squares algorithm for the data from scenario 3.
- Is this comparison fair? Is this truly a maximum likelihood estimator?
- Using a Gaussian *prior* pdf $p(\mathbf{p})$ centered at $\mathbf{p}$ with a diagonal covariance matrix $\Sigma = \mathrm{diag}\{\sigma_p^2\}$ with $\sigma_p = 1$, compute a Bayesian estimator for the position

$$\hat{\mathbf{p}}_\mathrm{Bayes} = \underset{\mathbf{p}}{\mathrm{argmax}}\, p\big(\tilde{\mathbf{d}}_0(\mathbf{p})|\mathbf{p}\big)p(\mathbf{p}). \tag{12}$$

  Again, use the evaluation over the grid. Describe the results! Does the prior knowledge enhance the accuracy?

Listing 1: Insert code directly in your document

```
#Filename: HW1_skeleton.py

import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import scipy.stats as stats
import copy
from mpl_toolkits.mplot3d import Axes3D

showZeros = True
# True: shows all points int he plot;
# False: Shows only points hight than 0;
stepsize= 0.05


#Used for the CDU in one plot
global xForPrint
global fxForPrint
xForPrint = [[],[]]
fxForPrint  = [[],[]]

#-------------------------------------------------------------------------
# Assignment 1
def main():

    # choose the scenario
    scenario = 3    # all anchors are Gaussian
    #scenario = 2    # 1 anchor is exponential, 3 are Gaussian
    #scenario = 3    # all anchors are exponential

    # specify position of anchors
    p_anchor = np.array([[5,5],[-5,5],[-5,-5],[5,-5]])
    nr_anchors = np.size(p_anchor,0)

    # position of the agent for the reference mearsurement
    p_ref = np.array([[0,0]])
    # true position of the agent (has to be estimated)
    p_true = np.array([[2,-4]])


    plot_anchors_and_agent(nr_anchors, p_anchor, p_true, p_ref)

    # load measured data and reference measurements for the chosen scenario
    data,reference_measurement = load_data(scenario)

    # get the number of measurements
    assert(np.size(data,0) == np.size(reference_measurement,0))
    nr_samples = np.size(data,0)

    #1) ML estimation of model parameters
    #TODO
```

```
params = parameter_estimation(reference_measurement,nr_anchors,p_anchor,p_ref)

#2) Position estimation using least squares
#TODO
position_estimation_least_squares(data,nr_anchors,p_anchor, p_true, True)

if(scenario == 3):

    print("\n3.3.1 Single Measurement")
    p_d0 = np.zeros(nr_anchors)
    help = np.zeros(nr_anchors)
    p_anchor_x  = p_anchor.T[0]
    p_anchor_y  = p_anchor.T[1]

    gridSize = int(10/stepsize)
    counter = np.zeros((gridSize, gridSize))

    Xplot, Yplot = np.mgrid[-5:5:stepsize, -5:5:stepsize]
    X = Xplot
    Y = Yplot

    help0 = np.zeros(Xplot.shape)
    p_Anchor0 = np.zeros(Xplot.shape)
    help1 = np.zeros(Xplot.shape)
    p_Anchor1 = np.zeros(Xplot.shape)
    help2 = np.zeros(Xplot.shape)
    p_Anchor2 = np.zeros(Xplot.shape)
    help3 = np.zeros(Xplot.shape)
    p_Anchor3 = np.zeros(Xplot.shape)

    help0 = data[0][0] - ((p_anchor_x[0]- X)**2 + (p_anchor_y[0]-Y)**2 )**0.5
    p_Anchor0 = params[0][0] * np.exp(- params[0][0] *  help0)
    p_Anchor0[np.where(help0 < 0)] = 0
    help1 = data[0][1] - ((p_anchor_x[1]- X)**2 + (p_anchor_y[1]-Y)**2 )**0.5
    p_Anchor1 = params[0][1] * np.exp(- params[0][1] *  help1)
    p_Anchor1[np.where(help1 < 0)] = 0
    help2 = data[0][2] - ((p_anchor_x[2]- X)**2 + (p_anchor_y[2]-Y)**2 )**0.5
    p_Anchor2 = params[0][2] * np.exp(- params[0][2] *  help2)
    p_Anchor2[np.where(help2 < 0)] = 0
    help3 = data[0][3] - ((p_anchor_x[3]- X)**2 + (p_anchor_y[3]-Y)**2 )**0.5
    p_Anchor3 = params[0][3] * np.exp(- params[0][3] *  help3)
    p_Anchor3[np.where(help3 < 0)] = 0

    p_d = p_Anchor0 * p_Anchor1 * p_Anchor2 * p_Anchor3

    arrayX, arrayY = np.where(p_d==p_d.max())
    counter[arrayX, arrayY] = counter[arrayX, arrayY] + 1

    print("Joint likelihood for the first Measurement")
    print("Maximum :",arrayX*stepsize-5,arrayY    *stepsize-5)

    fig = plt.figure()
    ax1 = fig.add_subplot(2,2,1, projection='3d')
    ax2 = fig.add_subplot(2,2,2, projection='3d')
```

```
ax3 = fig.add_subplot(2,2,3, projection='3d')
ax4 = fig.add_subplot(2,2,4, projection='3d')
fig2 = plt.figure()
ax = fig2.add_subplot(111, projection='3d')

if showZeros:

    ax1.scatter(Xplot,Yplot,p_Anchor0,alpha=0.6 , c="red")
    ax2.scatter(Xplot,Yplot,p_Anchor1,alpha=0.6 , c="green")
    ax3.scatter(Xplot,Yplot,p_Anchor2,alpha=0.6 , c="blue")
    ax4.scatter(Xplot,Yplot,p_Anchor3,alpha=0.6 , c="yellow")
    ax.scatter(Xplot,Yplot,p_d,  c="red")
    ax.plot(Xplot[arrayX,arrayY],Yplot[arrayX,arrayY], p_d.max(), 'bo', markersize=

else:
    Xplot = Xplot.flatten()
    Xplot = Xplot.tolist()
    Yplot = Yplot.flatten()
    Yplot = Yplot.tolist()
    counter = counter.flatten()
    counter = counter.tolist()

    p_Anchor0 = p_Anchor0.flatten()
    p_Anchor1 = p_Anchor1.flatten()
    p_Anchor2 = p_Anchor2.flatten()
    p_Anchor3 = p_Anchor3.flatten()
    p_d = p_d.flatten()
    p_d = p_d.tolist()
    p_Anchor0 = p_Anchor0.tolist()
    p_Anchor1 = p_Anchor1.tolist()
    p_Anchor2 = p_Anchor2.tolist()
    p_Anchor3 = p_Anchor3.tolist()

    for i in range(len(p_Anchor0)):
        if p_Anchor0[i] != 0:
            ax1.plot([Xplot[i]],[Yplot[i]], p_Anchor0[i], 'ro', markersize= 10)
        if p_Anchor1[i] != 0:
            ax2.plot([Xplot[i]],[Yplot[i]], p_Anchor1[i], 'bo', markersize= 10)
        if p_Anchor2[i] != 0:
            ax3.plot([Xplot[i]],[Yplot[i]], p_Anchor2[i], 'go', markersize= 10)
        if p_Anchor3[i] != 0:
            ax4.plot([Xplot[i]],[Yplot[i]], p_Anchor3[i], 'yo', markersize= 10)

    for i in range(len(counter)):
        if p_d[i] != 0:
            ax.plot([Xplot[i]],[Yplot[i]], p_d[i], 'bo', markersize= 10)
plt.show()

#3) Postion estimation using numerical maximum likelihood
#TODO
position_estimation_numerical_ml(data,nr_anchors,p_anchor, params, p_true)

#4) Position estimation with prior knowledge (we roughly know where to expect the a
#TODO
```

```
        # specify the prior distribution
        prior_mean = p_true
        prior_cov = np.eye(2)
        position_estimation_bayes(data,nr_anchors,p_anchor,prior_mean,prior_cov, params, p_

    pass
#---------------------------------------------------------------------------
#---------------------------------------------------------------------------
def parameter_estimation(reference_measurement,nr_anchors,p_anchor,p_ref):
    """ estimate the model parameters for all 4 anchors based on the reference measurements,
    Input:
        reference_measurement... nr_measurements x nr_anchors
        nr_anchors... scalar
        p_anchor... position of anchors, nr_anchors x 2
        p_ref... reference point, 2x2 """


    print("\n2. Maximum likelihood Estimation of Model Parameters")
    params = np.zeros([1, nr_anchors])

    anchor_0 = reference_measurement.T[0]
    anchor_1 = reference_measurement.T[1]
    anchor_2 = reference_measurement.T[2]
    anchor_3 = reference_measurement.T[3]


    # plt.hist(anchor_0, 50, density=True, facecolor='g')
    # plt.ylabel('# samples')
    # plt.xlabel('d / m')
    # plt.show()
    # plt.hist(anchor_1, 50, density=True, facecolor='g')
    # plt.ylabel('# samples')
    # plt.xlabel('d / m')
    # plt.show()
    # plt.hist(anchor_2, 50, density=True, facecolor='g')
    # plt.ylabel('# samples')
    # plt.xlabel('d / m')
    # plt.show()
    # plt.hist(anchor_3, 50, density=True, facecolor='g')
    # plt.ylabel('# samples')
    # plt.xlabel('d / m')
    # plt.show()

    # fig = plt.figure()
    # ax1 = fig.add_subplot(2,2,1)
    # ax2 = fig.add_subplot(2,2,2)
    # ax3 = fig.add_subplot(2,2,3)
    # ax4 = fig.add_subplot(2,2,4)
    # ax1.hist(anchor_0, 50, density=True, facecolor='g')
    # ax2.hist(anchor_1, 50, density=True, facecolor='g')
    # ax3.hist(anchor_2, 50, density=True, facecolor='g')
    # ax4.hist(anchor_3, 50, density=True, facecolor='g')
    # plt.show()
```

```
    mu_0 = ((0-5)**2 + (0-5)**2)**0.5
    mu_1 = ((0+5)**2 + (0-5)**2)**0.5
    mu_2 = ((0+5)**2 + (0+5)**2)**0.5
    mu_3 = ((0-5)**2 + (0+5)**2)**0.5

    ####################################Gaussian
    # mu_0 = np.sum(anchor_0)/ anchor_0.shape[0]
    # mu_1 = np.sum(anchor_1)/ anchor_1.shape[0]
    # mu_2 = np.sum(anchor_2)/ anchor_2.shape[0]
    # mu_3 = np.sum(anchor_3)/ anchor_3.shape[0]


    alpha = 1e-3
    k2_1, p_1 = stats.mstats.normaltest(anchor_0)
    k2_2, p_2 = stats.mstats.normaltest(anchor_1)
    k2_3, p_3 = stats.mstats.normaltest(anchor_2)
    k2_4, p_4 = stats.mstats.normaltest(anchor_3)

    if p_1 >=  alpha:
        var_0 = np.sum((anchor_0 - mu_0)**2) / anchor_0.shape[0]
        print("Anchor 0: Gaussian with sigma: ", var_0)
    else:
        var_0 = anchor_0.shape[0] / np.sum((anchor_0 - mu_0))
        print("Anchor 0: Expotnential with lambda: ", var_0)
    if p_2 >=  alpha:
        var_1 = np.sum((anchor_1 - mu_1)**2) / anchor_1.shape[0]
        print("Anchor 1: Gaussian with sigma: ", var_1)
    else:
        var_1 = anchor_1.shape[0] / np.sum((anchor_1 - mu_1))
        print("Anchor 1: Expotnential with lambda: ", var_1)
    if p_3 >=  alpha:
        var_2 = np.sum((anchor_2 - mu_2)**2) / anchor_2.shape[0]
        print("Anchor 2: Gaussian with sigma: ", var_2)
    else:
        var_2 = anchor_2.shape[0] / np.sum((anchor_2 - mu_2))
        print("Anchor 2: Expotnential with lambda: ", var_2)
    if p_4 >=  alpha:
        var_3 = np.sum((anchor_3 - mu_3)**2) / anchor_3.shape[0]
        print("Anchor 3: Gaussian with sigma: ", var_3)
    else:
        var_3 = anchor_3.shape[0] / np.sum((anchor_3 - mu_3))
        print("Anchor 3: Expotnential with lambda: ", var_3)

    params[0][0] = var_0
    params[0][1] = var_1
    params[0][2] = var_2
    params[0][3] = var_3

    return params
#-------------------------------------------------------------------------------
def position_estimation_least_squares(data,nr_anchors,p_anchor, p_true, use_exponential):
    """estimate the position by using the least squares approximation.
    Input:
        data... distance measurements to unkown agent, nr_measurements x nr_anchors
```

```
    nr_anchors... scalar
    p_anchor... position of anchors, nr_anchors x 2
    p_true... true position (needed to calculate error) 2x2
    use_exponential... determines if the exponential anchor in scenario 2 is used, bool

print("\n3.2 Gauss-Newton Algorithm for Position Estimation")
nr_samples = np.size(data,0)
#TODO set parameters
tol = 0.0001  # tolerance
max_iter = 1000  # maximum iterations for GN

xmin=-6
ymin=-6
xmax=6
ymax=6

point = np.zeros((data.shape[0],2))
d_error = np.zeros(data.shape[0])

if not use_exponential:
    p_anchor = np.delete(p_anchor, 0,0)
    data =  np.delete(data.T, 0,0)
    data = data.T


for i in range(0, nr_samples):

    p_start = [np.random.uniform(low=-5.0, high=5.0),np.random.uniform(low=-5.0, high=5
    measurements_n = data[i]
    point[i] = least_squares_GN(p_anchor,p_start, measurements_n, max_iter, tol)

# TODO calculate error measures and create plots----------------

d_error[:] =  ((point.T[0][:] - p_true[0][0])**2 +  (point.T[1][:] - p_true[0][1])**2)**


mu_error  = np.sum(d_error) / point.shape[0]
var_error = np.sum((d_error - mu_error)**2) / point.shape[0]
print("Error Mean: ",mu_error,"\nError Variance", var_error)

mu_x = np.sum(point.T[0]) / point.shape[0]
mu_y = np.sum(point.T[1]) / point.shape[0]
cov = np.cov(point.T)
plt.scatter(point.T[0], point.T[1])
plt.xlim([xmin,xmax])
plt.xlabel('x / m')
plt.ylabel('y / m')
plt.ylim([ymin,ymax])
plot_gauss_contour([mu_x,mu_y], cov,xmin,xmax,ymin,ymax, title="Estimated Positons")
Fx, x = ecdf(d_error)
plt.plot(x,Fx)
xForPrint[1] = x
fxForPrint[1] = Fx
plt.ylabel('P(point in d)')
```

```
    plt.xlabel('d / m')
    plt.show()
#-------------------------------------------------------------------------
def position_estimation_numerical_ml(data,nr_anchors,p_anchor, lambdas, p_true):
    """ estimate the position by using a numerical maximum likelihood estimator
    Input:
        data... distance measurements to unkown agent, nr_measurements x nr_anchors
        nr_anchors... scalar
        p_anchor... position of anchors, nr_anchors x 2
        lambdas... estimated parameters (scenario 3), nr_anchors x 1
        p_true... true position (needed to calculate error), 2x2 """
    global xForPrint, fxForPrint
    print("\n3.3.2 Multiple Measurements")

    nr_samples = np.size(data,0)
    p_d0 = np.zeros(nr_anchors)
    help = np.zeros(nr_anchors)
    p_anchor_x  = p_anchor.T[0]
    p_anchor_y  = p_anchor.T[1]

    gridSize = int(10/stepsize)
    counter = np.zeros((gridSize, gridSize))
    Xplot, Yplot = np.mgrid[-5:5:stepsize, -5:5:stepsize]
    X = Xplot
    Y = Yplot

    help0 = np.zeros(Xplot.shape)
    p_Anchor0 = np.zeros(Xplot.shape)
    help1 = np.zeros(Xplot.shape)
    p_Anchor1 = np.zeros(Xplot.shape)
    help2 = np.zeros(Xplot.shape)
    p_Anchor2 = np.zeros(Xplot.shape)
    help3 = np.zeros(Xplot.shape)
    p_Anchor3 = np.zeros(Xplot.shape)
    d_error = []

    for k in range(nr_samples):

        help0 = data[k][0] - ((p_anchor_x[0]- X)**2 + (p_anchor_y[0]-Y)**2 )**0.5
        p_Anchor0 = lambdas[0][0] * np.exp(- lambdas[0][0] *  help0)
        p_Anchor0[np.where(help0 < 0)] = 0

        help1 = data[k][1] - ((p_anchor_x[1]- X)**2 + (p_anchor_y[1]-Y)**2 )**0.5
        p_Anchor1 = lambdas[0][1] * np.exp(- lambdas[0][1] *  help1)
        p_Anchor1[np.where(help1 < 0)] = 0

        help2 = data[k][2] - ((p_anchor_x[2]- X)**2 + (p_anchor_y[2]-Y)**2 )**0.5
        p_Anchor2 = lambdas[0][2] * np.exp(- lambdas[0][2] *  help2)
        p_Anchor2[np.where(help2 < 0)] = 0

        help3 = data[k][3] - ((p_anchor_x[3]- X)**2 + (p_anchor_y[3]-Y)**2 )**0.5
        p_Anchor3 = lambdas[0][3] * np.exp(- lambdas[0][3] *  help3)
        p_Anchor3[np.where(help3 < 0)] = 0
```

```
            p_d = p_Anchor0 * p_Anchor1 * p_Anchor2 * p_Anchor3

            arrayX, arrayY = np.where(p_d==p_d.max())

            counter[arrayX, arrayY] = counter[arrayX, arrayY] + 1
            d_error.append((((arrayX*stepsize-5) - p_true[0][0])**2 +  ((arrayY*stepsize-5) - p_

        bigX , bigY =     np.where(counter == counter.max())

        print("\nNumerical maximum likelihood based on the joint likelihood")
        print("Maximum :",bigX*stepsize-5,bigY*stepsize-5)

        d_error = np.asarray(d_error)
        d_error = d_error.flatten()
        mu_error  = np.sum(d_error) / d_error.shape[0]
        var_error = np.sum((d_error - mu_error)**2) / d_error.shape[0]
        print("Error Mean: ",mu_error,"\nError Variance", var_error)

        fig2 = plt.figure()
        ax = fig2.add_subplot(111, projection='3d')

        if showZeros:
            ax.scatter(Xplot,Yplot,counter, c="red")
        else:
            Xplot = Xplot.flatten()
            Xplot = Xplot.tolist()
            Yplot = Yplot.flatten()
            Yplot = Yplot.tolist()

            counter = counter.flatten()
            counter = counter.tolist()

            for i in range(len(counter)):
                if counter[i] != 0:
                    ax.plot([Xplot[i]],[Yplot[i]], counter[i], 'bo', markersize= 10)

        plt.xlim([-5,5])
        plt.xlabel('x / m')
        plt.ylabel('y / m')
        plt.ylim([-5,5])
        plt.show()

        Fx, x = ecdf(d_error)
        xForPrint[0] = x
        fxForPrint[0] = Fx
        pass
#-------------------------------------------------------------------------------
def position_estimation_bayes(data,nr_anchors,p_anchor,prior_mean,prior_cov,lambdas, p_true)
    """ estimate the position by accounting for prior knowledge that is specified by a bivar
    Input:
        data... distance measurements to unkown agent, nr_measurements x nr_anchors
        nr_anchors... scalar
        p_anchor... position of anchors, nr_anchors x 2
        prior_mean... mean of the prior-distribution, 2x1
```

```
        prior_cov... covariance of the prior-dist, 2x2
        lambdas... estimated parameters (scenario 3), nr_anchors x 1
        p_true... true position (needed to calculate error), 2x2 """
global xForPrint, fxForPrint

nr_samples = np.size(data,0)
p_d0 = np.zeros(nr_anchors)
help = np.zeros(nr_anchors)
p_anchor_x  = p_anchor.T[0]
p_anchor_y  = p_anchor.T[1]

gridSize = int(10/stepsize)
counter = np.zeros((gridSize, gridSize))
Xplot, Yplot = np.mgrid[-5:5:stepsize, -5:5:stepsize]
X = Xplot
Y = Yplot

help0 = np.zeros(Xplot.shape)
p_Anchor0 = np.zeros(Xplot.shape)
help1 = np.zeros(Xplot.shape)
p_Anchor1 = np.zeros(Xplot.shape)
help2 = np.zeros(Xplot.shape)
p_Anchor2 = np.zeros(Xplot.shape)
help3 = np.zeros(Xplot.shape)
p_Anchor3 = np.zeros(Xplot.shape)
d_error = []

prior = (1/(2 * np.pi)) * np.exp(-(0.5 * ((X-2)**2 + (Y+4)**2)))

for k in range(nr_samples):


    help0 = data[k][0] - ((p_anchor_x[0]- X)**2 + (p_anchor_y[0]-Y)**2 )**0.5
    p_Anchor0 = lambdas[0][0] * np.exp(- lambdas[0][0] *  help0) * prior
    p_Anchor0[np.where(help0 < 0)] = 0
    help1 = data[k][1] - ((p_anchor_x[1]- X)**2 + (p_anchor_y[1]-Y)**2 )**0.5
    p_Anchor1 = lambdas[0][1] * np.exp(- lambdas[0][1] *  help1) * prior
    p_Anchor1[np.where(help1 < 0)] = 0
    help2 = data[k][2] - ((p_anchor_x[2]- X)**2 + (p_anchor_y[2]-Y)**2 )**0.5
    p_Anchor2 = lambdas[0][2] * np.exp(- lambdas[0][2] *  help2) * prior
    p_Anchor2[np.where(help2 < 0)] = 0
    help3 = data[k][3] - ((p_anchor_x[3]- X)**2 + (p_anchor_y[3]-Y)**2 )**0.5
    p_Anchor3 = lambdas[0][3] * np.exp(- lambdas[0][3] *  help3) * prior
    p_Anchor3[np.where(help3 < 0)] = 0
    p_d = p_Anchor0 * p_Anchor1 * p_Anchor2 * p_Anchor3
    arrayX, arrayY = np.where(p_d==p_d.max())
    counter[arrayX, arrayY] = counter[arrayX, arrayY] + 1
    d_error.append((((arrayX*stepsize-5) - p_true[0][0])**2 +  ((arrayY*stepsize-5) - p_


bigX , bigY =     np.where(counter == counter.max())

print("\nBayesian estimator")
print("Maximum: ",bigX*stepsize-5,bigY*stepsize-5)
```

```
    d_error = np.asarray(d_error)
    d_error = d_error.flatten()

    mu_error  = np.sum(d_error) / d_error.shape[0]
    var_error = np.sum((d_error - mu_error)**2) / d_error.shape[0]
    print("Error Mean: ",mu_error,"\nError Variance", var_error)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')


    if showZeros:
        ax.scatter(Xplot,Yplot,counter, c="red")
    else:
        Xplot = np.delete(Xplot, np.where(counter==0))
        Yplot = np.delete(Yplot, np.where(counter==0))

        Xplot = Xplot.flatten()
        Xplot = Xplot.tolist()
        Yplot = Yplot.flatten()
        Yplot = Yplot.tolist()

        counter = counter.flatten()
        counter = counter.tolist()

        for i in range(len(counter)):
            if counter[i] != 0:
                ax.plot([Xplot[i]],[Yplot[i]], counter[i], 'bo', markersize= 10)

    plt.xlim([-5,5])
    plt.xlabel('x / m')
    plt.ylabel('y / m')
    plt.ylim([-5,5])
    plt.show()

    Fx, x = ecdf(d_error)
    plt.plot(x,Fx)
    plt.plot(xForPrint[0],fxForPrint[0])
    plt.plot(xForPrint[1],fxForPrint[1])
    plt.legend(["bayesian","numerical maximum likelihood","least-squares"])
    plt.ylabel('P(point in d)')
    plt.xlabel('d / m')
    plt.show()
    pass
#-------------------------------------------------------------------------------
def least_squares_GN(p_anchor,p_start, measurements_n, max_iter, tol):
    """ apply Gauss Newton to find the least squares solution
    Input:
        p_anchor... position of anchors, nr_anchors x 2
        p_start... initial position, 2x1
        measurements_n... distance_estimate, nr_anchors x 1
        max_iter... maximum number of iterations, scalar
        tol... tolerance value to terminate, scalar"""
```

```
    nr_anchors = p_anchor.shape[0]
    jacobi = np.zeros((nr_anchors,2))

    p_anchor_x  = p_anchor.T[0]
    p_anchor_y  = p_anchor.T[1]

    d = np.zeros(nr_anchors)
    d_n = measurements_n
    p_now = p_start

    for i in range(max_iter):

        for j in range(nr_anchors):
            d[j] = (((p_anchor_x[j]-p_now[0])**2 + (p_anchor_y[j]-p_now[1])**2 )**0.5)
            jacobi[j][0] = (p_anchor_x[j]-p_now[0]) / (((p_anchor_x[j]-p_now[0])**2 + (p_anc
            jacobi[j][1] = (p_anchor_y[j]-p_now[1]) / (((p_anchor_x[j]-p_now[0])**2 + (p_anc

        help = np.matmul(np.linalg.inv(np.matmul(jacobi.T,jacobi)), np.matmul(jacobi.T, (d_n
        p_t1 = p_now - help

        d_now = ((p_t1[0] -p_now[0])**2 + (p_t1[1]-p_now[1])**2)**0.5

        if d_now < tol:
            return p_t1

        p_now = p_t1

    return None

#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
# Helper Functions
#-----------------------------------------------------------------------------
def plot_gauss_contour(mu,cov,xmin,xmax,ymin,ymax,title="Title"):

    """ creates a contour plot for a bivariate gaussian distribution with specified paramet

    Input:
      mu... mean vector, 2x1
      cov...covariance matrix, 2x2
      xmin,xmax... minimum and maximum value for width of plot-area, scalar
      ymin,ymax....minimum and maximum value for height of plot-area, scalar
      title... title of the plot (optional), string"""

        #npts = 100
    delta = 0.025
    X, Y = np.mgrid[xmin:xmax:delta, ymin:ymax:delta]
    pos = np.dstack((X, Y))

    Z = stats.multivariate_normal(mu, cov)
    plt.plot([mu[0]],[mu[1]],'r+') # plot the mean as a single point
    plt.gca().set_aspect("equal")
    CS = plt.contour(X, Y, Z.pdf(pos),3,colors='r')
    plt.clabel(CS, inline=1, fontsize=10)
```

```
    plt.title(title)
    plt.show()
    return

#-------------------------------------------------------------------------
def ecdf(realizations):
    """ computes the empirical cumulative distribution function for a given set of realizati
    The output can be plotted by plt.plot(x,Fx)

    Input:
      realizations... vector with realizations, Nx1
    Output:
      x... x-axis, Nx1
      Fx...cumulative distribution for x, Nx1"""
    x = np.sort(realizations)
    Fx = np.linspace(0,1,len(realizations))
    return Fx,x

#-------------------------------------------------------------------------
def load_data(scenario):
    """ loads the provided data for the specified scenario
    Input:
        scenario... scalar
    Output:
        data... contains the actual measurements, nr_measurements x nr_anchors
        reference.... contains the reference measurements, nr_measurements x nr_anchors"""
    data_file = 'measurements_' + str(scenario) + '.data'
    ref_file =  'reference_' + str(scenario) + '.data'

    data = np.loadtxt(data_file,skiprows = 0)
    reference = np.loadtxt(ref_file,skiprows = 0)

    return (data,reference)
#-------------------------------------------------------------------------
def plot_anchors_and_agent(nr_anchors, p_anchor, p_true, p_ref=None):
    """ plots all anchors and agents
    Input:
        nr_anchors...scalar
        p_anchor...positions of anchors, nr_anchors x 2
        p_true... true position of the agent, 2x1
        p_ref(optional)... position for reference_measurements, 2x1"""
    # plot anchors and true position
    plt.axis([-6, 6, -6, 6])
    for i in range(0, nr_anchors):
        plt.plot(p_anchor[i, 0], p_anchor[i, 1], 'bo')
        plt.text(p_anchor[i, 0] + 0.2, p_anchor[i, 1] + 0.2, r'$p_{a,' + str(i) + '}$')
    plt.plot(p_true[0, 0], p_true[0, 1], 'r*')
    plt.text(p_true[0, 0] + 0.2, p_true[0, 1] + 0.2, r'$p_{true}$')
    if p_ref is not None:
        plt.plot(p_ref[0, 0], p_ref[0, 1], 'r*')
        plt.text(p_ref[0, 0] + 0.2, p_ref[0, 1] + 0.2, '$p_{ref}$')
    plt.xlabel("x/m")
    plt.ylabel("y/m")
    plt.show()
```

```
        pass

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
if __name__ == '__main__':
    main()
```