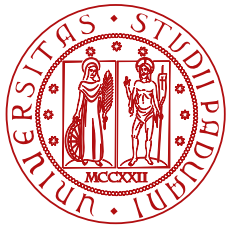

UNIVERSITÀ DEGLI STUDI DI PADOVA


DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

MASTER THESIS IN CONTROL SYSTEMS ENGINEERING

# Navigation and estimation of separative labelling curves

MASTER CANDIDATE

**Giacomo Sartori**

**Student ID 2053071**

SUPERVISOR

**Prof. Karl H. Johansson, Postdoc. Aneesh Raghavan**

**KTH Royal Institute of Technology**

CO-SUPERVISOR

**Prof. Alessandro Chiuso**

**University of Padova**

ACADEMIC YEAR
2022/2023
Graduation date: 10/10/2023

*To my family,*
*to my friends,*
*to my land,*
*to my efforts,*
*to all the dark and light days,*
*to a future of joy and love.*

**Abstract**

Motion planning algorithms have been developed in different fields by the control engineering community to do different tasks: move one or more agents from a starting state to an ending state, to track a reference trajectory, etc. To do so, both robotic, vision and learning techniques have been used. At the same time, a lot of effort in the computer engineering community has been done in the machine learning field to classify labeled data sets. The aim of this work is to use concepts and tools of both these fields to build up different motion planning algorithms for classification tasks, that make an agent with a specific dynamics to move in a space where two labeled regions are present, observing the labels just in the points where it discretely pass through, in order to estimate and navigate the curve separating the different regions. The ideal case where no errors on the measurements of the labels are present is firstly studied, where convergence to the separative curve is analyzed and reached under certain conditions. After that, two types of noises in the measurements are studied, namely uniform distributed and gaussian distributed errors. In this case, navigation is reached for both the error models, while convergence is just reached (in a probabilistic sense) in the uniform distributed error model. The case where the separative curve in not linear is also addressed, both in the ideal no error case and in the error model in the label's measurements, and the navigation and a good estimation of the curve is achieved, whereas no convergence is possible.

**Sommario**

Algoritmi di pianificazione del movimento sono stati sviluppati in differenti campi dalla comunità di ingegneria del controllo per svolgere diversi compiti: muovere uno o più agenti da una posizione iniziale a una posizione finale, per tracciare una traiettoria di riferimento, etc. Per fare ciò sono state utilizzate sia tecniche di robotica, che di visione e di apprendimento. Allo stesso tempo, molto sforzo è stato fatto dalla comunità di ingeneria informatica nel campo dell'apprendimento automatico per classificare degli insiemi di punti con delle etichette. Lo scopo di questo lavoro è unire assieme questi due approcci per realizzare degli algoritmi innovativi che facciano muovere un agente con una specifica dinamica e osservando le etichette dei punti in cui transita, nello spazio dove due regioni con specifiche etichette sono presenti, per stimare e navigare la curva che le separa.
Il caso ideale senza alcun errore nelle misurazioni delle etichette è esaminato per primo, dove la convergenza alla curva di separazione è analizzata e raggiunta con certe condizioni. Dopo di che, due tipi di rumori nelle misurazioni sono studiati, errori uniformemente distribuiti ed errori distribuiti in modo gaussiano. In questo caso, la navigazione è soddisfatta in entrambi i modelli, mentre la convergenza (in un senso probabilistico) è raggiunta solo nel modello con errori uniformemente distribuiti. Inoltre, il caso in cui la curva di separazione non è lineare è stato investigato, sia nel caso ideale senza errori che in quello con un modello di errrore nelle misurazioni delle etichette, realizzando la navigazione e una buona stima della curva, mentre la convergenza non è possibile.

# Acknowledgments

This work might be the final part of my studying path started when I was three years old. So I would like to thank all the people that helped me and leaded me to realize this work. First of all my whole family, that allowed me to come in Sweden spending a semester abroad while supporting from home. Even though I do not usually thank you explicitly, I am super happy to be part of you! All my friends from Italy and not only, I appreciated that some of them visited me during my exchange period, some others called me or just texted me. All the international people that I met abroad, making me know a lot of new things, and living different days compared to the ones I used to live in my home town. I am also glad to UE, to have created this possibility of studying abroad interconnecting many universities in a very reach network, giving to every student the chance to see and live in a different country; I had the possibility of living in Valencia during my bachelor and now in Stockholm, embracing two very different cultures and tasting the flavours (with the related pros and cons) of their alleys, their natural places, their inhabitants and their universities, at the point that I feel very moved while I am writing these lines. I am glad to all my professors that I have had in my studies, from the child shool in my small town, Arzerello, to the university in Padua, UNIPD. I am also glad at KTH university for the way I was treated, with an "office" in the same floor with many PHD students and professors, breathing research. I want to thank my examiner prof. Karl H. Johansson to have taken me in his group and to have met me once, giving me precious suggestions. Last, but not least, I want to thank my supervisor Postdoc. Aneesh Raghavan, that reviewed my thesis. Although our relationship has always been up-down, probably for our different nature, has showed me how research is made, maybe treating me as a PHD student. This has meant to me a big growth in terms of knowledge on scientific literature, as I have read during my work a lot of related papers, in terms of capability of writing a scientific doc-

ument, as I have been part of a paper developed by him, in terms of struggling to reach a specific goal, giving me the freedom (and the uncertainty) of develop my thesis as I preferred, and challenging me with the developing of proofs as rigorous as possible. I strongly believe that all of this will be very important for my future carer, everywhere it will take place.

I can conclude saying that this work has been part of my routine in these six months, that have somehow changed myself and making me realize a lot of new things about my identity and myself. In fact, I consider the time I spent working on this thesis as a long trip, that however does not end here, as it is part of a bigger trip that has been called life ;).

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Code Snippets

# List of Acronyms

**DTMC** Discrete-time Markov chain

**CTMC** Continuous-time Markov chain

**SVM** Support Vector Machine

**ML** Machine Learning

**MA** Moving Average

**BU** Butterwoth

**RL** Reinforcement Learning

**MPC** Model Predictive Control

**OC** Optimal Control

**RRT*** Rapid Random Tree Star

**ERM** Empirical Risk Minimization

**SVD** Singular Value Decomposition

**RR** Ridge Regression

**RMSE** Root Mean Square Error

# 1

# Introduction to the problem and contributions

The aim of this thesis is to study the possibility of estimating a curve (that in our setup is a **classifier**) that separates a region $\mathbb{S}$ into two labeled regions $\mathbb{S}_1$ and $\mathbb{S}_2$ in a $2D$ space, as in Figure 1.1. To get started, firstly we consider a pure sampling problem, and we solve it, assuming that any possible point can be sampled revealing its label. Then, an agent with its own dynamics is introduced. To keep the problem with the minimum number of assumptions, the only information that is provided is:

- the knowledge of two initial points in the considered space with opposite label;
- the capacity of the agent to observe, measure, only the label of the data points where it passes through in a discrete time sequence.

Moreover, the addition of some noise on the label's measurements adds more complexity to face and to solve the problem, but at the same time, makes it more close to the real world's scenario.

The solutions that will be provided in this work vary depending on the type of separative curve: if it is a linear curve, namely a line, both convergence of the trajectory of the agent to the line and the relative estimation will be sought, whereas if it is a more generic curve, only the navigation will be sought by the agent, leaving the estimation of the curve as an additional problem treated differently.

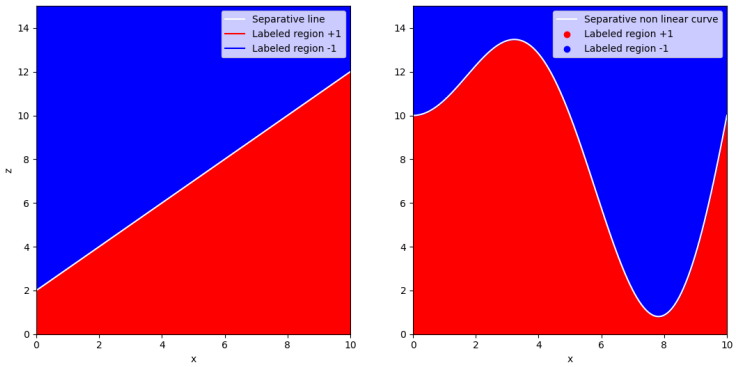


Figure 1.1: Division of the space in two labeled regions $\mathbb{S}_1$ (red) and $\mathbb{S}_2$ (blue), in the linear and non linear case.

## 1.1 Contributions and outline of the thesis

This work is divided in different Chapters in which the problem is analyzed by different perspectives. In the **Sampling problem** chapter (3), Algorithm 1 has been developed to estimate a point belonging to the separative line and from that to estimate the whole line or curve by iterating it as many times as needed. The proof of its convergence and considerations about its complexity are also provided. In the next chapter, **Identification of linear classifiers with noiseless data** (4), some dynamic constraints are introduced, by considering a single 2-dimensional discrete-time integrator with a third pseudo-state component given by the direction of the agent, that is useful to move it. The same agent will be used also in the other chapters. In this scenario, namely without errors on the measurements of the labels, a solution is provided by Algorithm 2 that makes the agent converge to the separative line and navigating around it. This achievement allows to obtain in a real-time fashion the estimate of the line, and does not require for further estimations tools.
Then, in chapter **Identification of linear classifiers with noisy data** (5), we introduce two error noise models in the measurements of the labels, the uniform error distributed and the gaussian error distributed ones. As for the uniform distributed error model, a procedure able to give a relation between the total number of data points needed to converge and the probability of the Algorithm 3 to end up with success is provided, by using Markov processes reasonings and analysis. As for the gaussian distributed error model, convergence cannot

be achieved but the agent can still navigate around the line and eventually estimate it in a proper well-enough way.
The next chapter, **Identification of special nonlinear classifiers with noisy/noiseless data** (6), extends the setup to the case with non linear separative curve, where we do not achieve any convergence, but the agent can still navigate different curves both in the ideal no measurement's error case and in the uniform and gaussian noise error model. Simulations with a sinusoidal elliptic curves are presented. An estimation part is needed to retrieve an estimate of the separative curve from the data points collected by the agent. Different estimation's approaches are presented, and eventually the one using a smoothing filter is preferred on the more computational demanding Machine Learning (ML) algorithms.
In the last chapter, **Conclusions and Future Work's chapter** (7), a final list and recap of all the results is presented with a final discussion. Besides, we also propose further possible research that can be done from this work.
All the chapters are based on more generic themes belonging to different areas, such as control theory, machine learning ML, probability, that are briefly explained in the Appendix 8, but that rely on the fact that the reader is already familiar with some of these topics. Moreover, all the chapters have the implementations and simulations of the algorithms with their relative comments.

# 2

# Motivation and literature review

The problem of estimating a line and or navigating it using a mobile agent has been attached in the robotic and control community quite intensively and many different algorithms and procedures have been developed in the last decades.
Some examples can be found in the Reinforcement Learning (RL) field: an agent that is in a certain state, that does not know the environment where it lays, takes a certain action based on the reward it got in order to maximize the cumulative reward (long-term reward), or, in a probabilistic setting, the expected cumulative reward. The agent often takes the actions following the so-called exploration-exploitation trade-off, in order to explore the unknown states with their relative rewards, and to exploit greedily the knowledge it made, up to that time moment, about the explored states.
However, RL requires plenty of data and involves a lot of computation before figuring a good strategy out. In this work instead, we try to use as less data (data points) as possible, thus significantly differentiating the two approaches.
In the sampling motion planning algorithms community, other important considerable works can be found in the Rapid Random Tree Star (RRT*) algorithm, where a randomic tree is built in a fast way ($O(n \log n)$) from an initial starting point to a final one, avoiding obstacles and with the shortest length. However, also this algorithm is not suited for the setup of this work as the latter does not provide an initial and a final point.
Another well known and studied solution might be the Model Predictive Control (MPC), a control technique that iteratively solves an optimization problem to find an input sequence at each iteration that minimizes a cost function and

takes into account for time varying constraints. However, MPC is used in applications where sensors provide information also about the future states, whereas in this work we consider the agent provided only of a simple measuring system that can observe the label of the point where it is, with the same sampling frequency of the discrete time iteration of the system. Hence, also MPC is not so useful in our scenario.
We may then think of optimal control Optimal Control (OC), that can be thought of as a simplification of the MPC in the case where the constraints are not varying in time, and in fact we tried to use this technique at some point during the development part to make the agent move from one side to the other one of the separative line. However, it turns out that the optimization problems we are going to solve, namely 4.25 and 4.26 do not need for an OC approach as they can be solved just minimizing the cost function iteration by iteration.
Instead, the way this problem is attached is by using the online information of the label of the current state data point, together with the previous ones, and adapting a control law as a consequence of such labels in a real-time fashion.

From the point of view of the utility of this work, although it can seem a purely theoretic problem, practical extensions can be studied as feature research (see Chapter 7). Examples of concrete applications where an agent can ”learn” how to move on a curve just using its previous labeled sampled states are: robotic agents, like car models that must identify a line separating an area where they can transit from another one that is not accessible for different reasons (security, privacy) and so they track it by just navigating it around the margin.
In the scientific literature, similar problems can be found in the works of [3] and of [4], where the applications where about the estimation of the Algal Bloom in the Baltic sea, and the agent an unmanned surface vehicle. However, in both the works, the initial assumptions and the setups are different from the ones used in ours.

# 3 Sampling Problem

## 3.1 Intro

In this chapter, the sampling problem to identify linear and non linear classifiers is addressed, namely we study how to pick data points sampling them in a strategical sequential way in order to estimate a line, an hyperplane or a non linear curve, that separates two labelled regions on the space. In particular Algorithm 1 is developed to estimate a point belonging to the separative classifier in exponential convergence rate, provided the knowledge of two initial given points with different labels. In the case of non linear classifiers, if the shape is given it is possible to use less estimates, otherwise the more complex the shape of the classifier is, the largest the number of points to be estimated by using multiple times Algorithm 1, estimating the final curve by using an interpolation.

## 3.2 Problem formulation

Starting from the simplest example of a linear separative curve in the $2D$ space, provided that there exists a separative line between $\mathbb{S}_1$ and $\mathbb{S}_2$ with labels $1$ and $-1$ with equation:

$$z = m^* x + q^*, \tag{3.1}$$

the goal is to try to strategically sample $j$ points to estimate $\hat{m}$ and $\hat{q}$ such that:

$$\lim_{j\to\infty} \|\hat{m}_j - m^*\| = 0 \text{ and } \lim_{j\to\infty} \|\hat{q}_j - q^*\| = 0. \tag{3.2}$$

Two assumptions are made:

- an oracle provides the true labels of the data points that are sampled;
- to have full knowledge of two initial points, namely $P_{0_1}$ (or $\mathbf{x}_{0_1}$) and $P_{0_2}$ (or $\mathbf{x}_{0_2}$), with different labels.

The problem can be also written, in the case we just care about achieving a specific threshold value of accuracy, as seeking for $\hat{m}$ and $\hat{q}$ such that:

$$\lim_{j \to J} \|\hat{m}_j - m^*\| < \epsilon_m \text{ and } \lim_{j \to J} \|\hat{q}_j - q^*\| < \epsilon_q, \tag{3.3}$$

for any initial pair $P_{0_1}$ and $P_{0_2}$ with different labels, where $P_J$ is the final sampled (or estimated) point at iteration $j = J$.
We also generalize the problem for high dimensional spaces in the following way. We consider the space $\mathbb{R}^d$. The space is divided into two regions by a hyperplane. Every other point in the space carries a label. The hyperplane is unknown and the objective is to find the same. The parameters of the true hyperplane is denoted by $(w^*, b^*)$. Initially, we are given two labeled data points, $\{x_{0_1}, 1\}$ and $\{x_{0_2}, -1\}$. With these two data points we estimate a point belonging to the separative hyperplane, close enough to it. With some more initial points we estimate in total $d$ points with the same procedure. Eventually, the goal is to estimate these points such that the estimated hyperplane parametrized by $(\hat{w}, \hat{b})$ is such that:

$$\|\hat{w} - w^*\| < \epsilon_w \text{ and } \|\hat{b} - b^*\| < \epsilon_b. \tag{3.4}$$

## 3.3 Proposed solution

In this section the solution we developed is presented.
The first chosen data point $\mathbf{x_1}$ is the average between the two initial given data points. Once checked its label, we take as new data point the average between $\mathbf{x_1}$ and the initial point with opposite label, and we save $\mathbf{x_1}$ as the new initial point on its side (as now the initial point with the same label is farther than $\mathbf{x_1}$ from the hyperplane and hence less informative); namely either $\mathbf{x_{0_1}}$ or $\mathbf{x_{0_2}}$ is updated as $\tilde{\mathbf{x}}_{0_i} = \mathbf{x_1}$. By repeat the procedure taking at iteration $j$ the point $\mathbf{x_j}$ that is the average between $\mathbf{x_{j-1}}$ and the point among $\tilde{\mathbf{x}}_{0_1}$ and $\tilde{\mathbf{x}}_{0_2}$ with different label from $\mathbf{x_{j-1}}$. The final estimate, can be retrieved as $\hat{\mathbf{x}}$, the average between the last point $\mathbf{x_J}$ and the last point $\mathbf{x_j}$ with opposite label from $\mathbf{x_J}$, where $J$ can be chosen at

the beginning or can be found running the algorithm and stopping it when, for instance, two consecutive data points are closer by than a certain threshold.
The convergence of the sequence of the data points to an estimated point $\hat{\mathbf{x}}$ is guaranteed to be equal to an actual point $\bar{\mathbf{x}}$ on the true hyperplane when the number of data points grows to infinity.

The following definition is needed to prove the convergence of the proposed solution to a point belonging to the separative curve.
**Definition(Cauchy sequence):** Let $(X, d)$ be a metric space. A sequence $\{\mathbf{x}_n \in X\}$ is said to be Cauchy if given $\epsilon > 0$ there exists a natural number $N_\epsilon$ such that $d(\mathbf{x}_m, \mathbf{x}_n) < \epsilon$ for all $m, n > N_\epsilon$.

**Proposition.** ***(proof of the convergence)*** *The sequence built up in the procedure 3.3 converges to a point of the classifier. Therefore, we can estimate a linear classifier as in the sense of Eq. 3.2 and Eq. 3.3 (the proposition also holds for $d-$dimensional spaces, as we only use euclidean distance reasonings).*

*Proof.* We consider the metric space $(\mathbb{R}^d, d)$, where the metric $d$ is the usual euclidean distance. Such metric space is complete. If we prove that the sequence $\{\mathbf{x}\}_{j=1,2,\ldots}$ is a Cauchy sequence then we prove in turn that the sequence is convergent to a fixed point, and such a point can only be a point on the true hyperplane because the sequence lies around the true hyperplane and gets closer and closer to it by construction.
Let's construct a generic sequence $\{\mathbf{x}_j\}_j$: $\mathbf{x}_1 = \frac{\mathbf{x}_{0_1}+\mathbf{x}_{0_2}}{2}$, $\mathbf{x}_2$ already depends on the labels of the previous points and on their positions (this is a non linear relation), and can be either $\mathbf{x}_2 = \frac{1}{2}\left[\frac{\mathbf{x}_{0_1}+\mathbf{x}_{0_2}}{2} + \mathbf{x}_{0_1}\right] = \frac{3\mathbf{x}_{0_1}+\mathbf{x}_{0_2}}{4}$ or $\mathbf{x}_2 = \frac{1}{2}\left[\frac{\mathbf{x}_{0_1}+\mathbf{x}_{0_2}}{2} + \mathbf{x}_{0_2}\right] = \frac{\mathbf{x}_{0_1}+3\mathbf{x}_{0_2}}{4}$, however is possible to write it as $\mathbf{x}_2 = \frac{\lambda_2\mathbf{x}_{0_1}+(2^2-\lambda_2)\mathbf{x}_{0_2}}{2^2}$, where $1 \leq \lambda_2 < 2^2$.
In general,

$$\mathbf{x}_j = \frac{\lambda_j\mathbf{x}_{0_1} + (2^j - \lambda_j)\mathbf{x}_{0_2}}{2^j}, \text{ where } 1 \leq \lambda_j < 2^j. \tag{3.5}$$

So:

$$
\begin{aligned}
&d(\mathbf{x}_m, \mathbf{x}_n) = ||\mathbf{x}_n - \mathbf{x}_m|| = \\
&= \sqrt{\left(\frac{\lambda_n x_{0_{11}} + (2^n - \lambda_n)x_{0_{21}}}{2^n} - \frac{\lambda_m x_{0_{11}} + (2^m - \lambda_m)x_{0_{21}}}{2^m}\right)^2 + \left(\frac{\lambda_n x_{0_{12}} + (2^n - \lambda_n)x_{0_{22}}}{2^n} - \frac{\lambda_m x_{0_{12}} + (2^m - \lambda_m)x_{0_{22}}}{2^m}\right)^2} \leq \\
&\leq \sqrt{\left(\frac{\lambda_n x_{0_{11}} + (2^n - \lambda_n)x_{0_{21}}}{2^{min(n,m)}} - \frac{\lambda_m x_{0_{11}} + (2^m - \lambda_m)x_{0_{21}}}{2^{min(n,m)}}\right)^2 + \left(\frac{\lambda_n x_{0_{12}} + (2^n - \lambda_n)x_{0_{22}}}{2^{min(m,n)}} - \frac{\lambda_m x_{0_{12}} + (2^m - \lambda_m)x_{0_{22}}}{2^{min(n,m)}}\right)^2} = \\
&= \sqrt{\left(\frac{a}{2^{min(n,m)}}\right)^2 + \left(\frac{b}{2^{min(n,m)}}\right)^2} = \sqrt{\frac{a^2 + b^2}{2^{2min(n,m)}}} = \frac{\sqrt{a^2 + b^2}}{2^{min(n,m)}} < \epsilon,
\end{aligned}
\tag{3.6}
$$

where

$$a = a(\lambda_n, \lambda_m, x_{0_{11}}, x_{0_{21}}) = \lambda_n x_{0_{11}} + (2^n - \lambda_n)x_{0_{21}} - (\lambda_m x_{0_{11}} + (2^m - \lambda_m)x_{0_{21}}),$$

$$b = b(\lambda_n, \lambda_m, x_{0_{12}}, x_{0_{22}}) = \lambda_n x_{0_{12}} + (2^n - \lambda_n)x_{0_{22}} - (\lambda_m x_{0_{12}} + (2^m - \lambda_m)x_{0_{22}}).$$

The latter inequality can be written as

$$\frac{2^{min(n,m)}}{\sqrt{a^2 + b^2}} > \frac{1}{\epsilon} \implies \frac{2min(n,m)}{\log_2(a^2 + b^2)} > \frac{1}{\log_2(\epsilon)} = -\log_2(\epsilon). \tag{3.7}$$

Therefore we can choose,

$$\frac{min(n,m)}{\log_2(a^2 + b^2)} \geq N > -\frac{1}{2}\log_2(\epsilon). \tag{3.8}$$

□

**Proof of the limit.** As for the limit, now that we know that the limit exists, we have to prove that $\lim_{j\to\infty} d(\mathbf{x}_j, \bar{\mathbf{x}}) = 0$.
Indeed, by contrapositive, let's suppose that $\lim_{j\to\infty} d(\mathbf{x}_j, \bar{\mathbf{x}}) = c \neq 0$. This would mean that there exists a new iteration in which $\mathbf{x}_\infty$, that is the position at time $t = +\infty$, would be such that $d(\frac{\mathbf{x}_\infty + \mathbf{x}_{opposite}}{2}, \bar{\mathbf{x}}) < d(\mathbf{x}_\infty, \bar{\mathbf{x}})$ and this is an absurd since the sequence is moving towards the objective hyperplane. In fact the sequence, keeps averaging between the closest points from the classifier with different labels, hence reducing the distance from it. □

By repeating the same procedure we find $d$ estimates of points belonging to a $d-1$ dimensional hyperplane separating two $d$ dimensional half-spaces.

For example, for a line, that is a 1-dimensional hyperplane, we need 2 estimates to find out the estimated hyperplane.

From the proposed solution, Algorithm 1 can be easily composed, with just some reminds:

- $J$ is fixed a priori and it has not necessarily the same meaning of the same symbol proposed in Section 3.3 as if such a value does not allow the algorithm to converge inside the $\epsilon$ interval, then it is different, however, as explained in the subsection 3.4.1, it is not difficult at all to make it convergent as it does that very fast;
- $\delta$ is chosen depending on the accuracy that one wants to achieve.

**Algorithm 1** Sampling algorithm for identification of a point belonging to a generic classifier

**Require:** $p_{0_1}, p_{0_2}, J, \delta$
  $p_1 \leftarrow avg(p_{0_1}, p_{0_2})$
  $\hat{p} \leftarrow p_1$
  $j \leftarrow 2$
  **while** $j < J \; and \; d(x_j, x_{j-1}) > \delta$ **do**
    **if** $y_{j-1} = y_{p_{0_1}}$ **then**
      $p_{0_1} \leftarrow p_{j-1}$
      $p_{opposite} \leftarrow p_{0_2}$
    **else**
      $p_{0_2} \leftarrow p_{j-1}$
      $p_{opposite} \leftarrow p_{0_1}$
    $p_j \leftarrow avg(p_{j-1}, p_{opposite})$
    $\hat{p} \leftarrow p_j$
    $j \leftarrow j + 1$
  **return** $\hat{p}$

## 3.4 Simulations and results

The next figure shows a sequence of data points sampled by Algorithm 1, in which it is possible to appreciate its convergent behaviour towards an estimate of a point belonging to the true line.

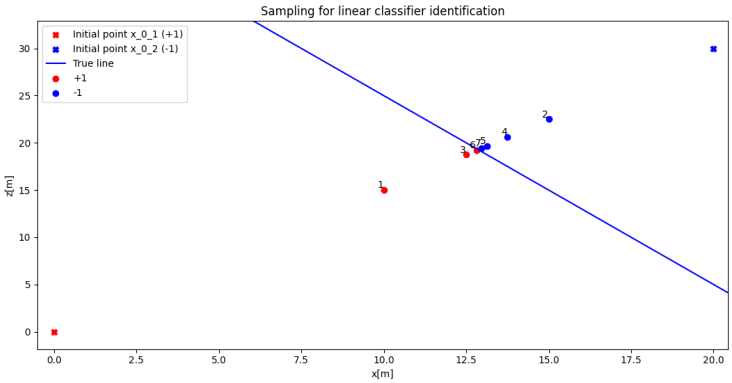


Figure 3.1: Example of the first 7 points sampled by Algorithm 1.

Indeed, the next figure shows furthermore how the error distance decreases to zero in a quite small number of data points.

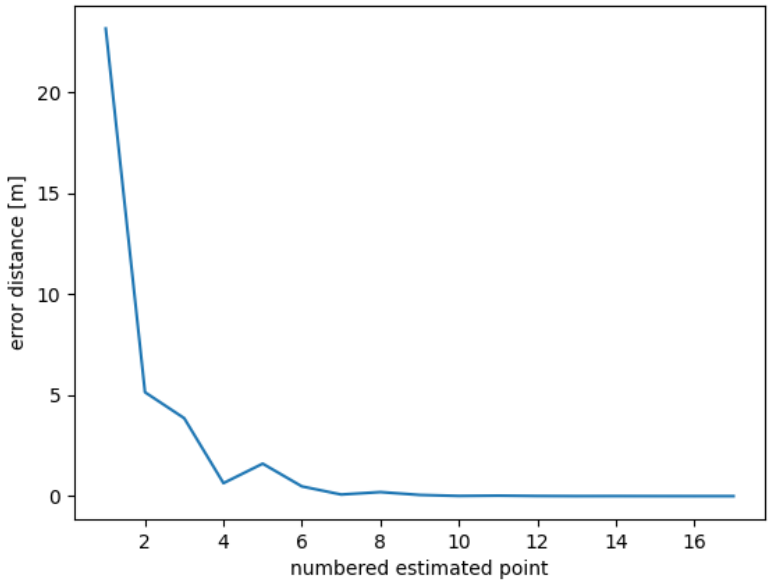


Figure 3.2: Error distance of the points sampled by Algorithm 1.

By iterating the same procedure (it is just necessary another known initial given point, but since is possible to sample any point this is not an issue), more points belonging to the true separative line can be estimated and from there we can obtain the equation of such a curve.

**Note**: in Figure 3.2 the error curve is not monotonically decreasing, as in the interval of points $\{4, 6\}$ the error increases a bit. This is because it can happen

that, sampling in an opposite side, the new data point has a distance larger than the previous one (that was on the other side), however, the important thing, is that between two consecutive points on the same region side (e.g. point 3 and 6) the function is monotonically decreasing, as in fact this is the case.

### 3.4.1 Complexity, lower bound rate of convergence of Algorithm 1

The complexity of Algorithm 1 is simply $O(n)$, as one while loop is necessary to obtain the final estimate and in such a loop we only use the the closest 2 points with opposite labels to obtain the new data point computing the average.
As far as the rate of convergence, we can simply consider the worst case, that is the one where the two initial points are such that one of them is very close, in particular $\epsilon$ distant, from the separative line $z = m^*x + q^*$, whereas the other is more far away. Therefore, let $d_{max} = max\{d(\mathbf{x}_{0_1}, z = m^*x + q^*), d(\mathbf{x}_{0_2}, z = m^*x + q^*)\}$, then it must hold:

$$\frac{d_{max}}{2^j} < \epsilon, \tag{3.9}$$

meaning that the minimum total number of points, in the worst case, to make Algorithm 1 converge, is:

$$J > \log_2 \frac{d_{max}}{\epsilon}. \tag{3.10}$$

Notice that this minimum number is quite easy to accomplish, as it scales with a logarithm function, and that the actual Algorithm 1 in a generic configuration is very close to this logarithm behaviour, hence reaching convergence (although this has not been proved) exponentially fast.

## 3.5 Non linear separative curves

We extend problem 3.2 to the non linear case, where Eq. 3.1 becomes:

$$z(x) = h^*(x), \tag{3.11}$$

where $h^*(\cdot)$ is a non linear function, or becomes:

$$C^* = \{(x(t), y(t)) : t \in \mathcal{I}\}, \tag{3.12}$$

where $\mathcal{I}$ is an interval and $t$ is the parameter, in the case the separative curve is a parametrized curve.
Instead, Eq. 3.2 becomes:

$$\lim_{n\to\infty} \|\hat{h}_n - h^*\| = 0 \text{ or } \lim_{n\to\infty} \|\hat{C}_n - C^*\| = 0, \tag{3.13}$$

where the norm operator, $\|\cdot\|$, indicates a loss function that can be properly chosen.

### 3.5.1 Solution attempt with Kernel methods

An idea to solve the problem presented in Section 3.2 is the one of using the properties of the kernel functions (see Section 8.2).
In fact, the property of mapping a low-dimensional space into a larger one in order to make the separative non linear curve in the low-dimensional space linear into the larger one may be the solution to the problem. In fact, after having mapped it into a linear hyperplane (of dimension $d$, let's say), we may just use Algorithm 1 $d$ times to estimate it.

**However, the proposed procedure is not achievable in practice for the following important reason:**
Let's consider the quadratic polynomial kernel $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^2$ where $\mathbf{x} \in \mathbb{R}^2$ and whose feature map is defined as

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 \end{bmatrix}' \in \mathbb{R}^6, \tag{3.14}$$

and suppose that the curve to be estimated is a linearly separable in this $6D$ augmented space. However, even though we had the at least $6$ points needed for retrieve the hyperplane that linearly separates the two label regions in the $6D$ state space, that could be computed by Algorithm 1 iterating it $6$ times, we would not be able to retrieve in turn the non linear curve that separates the two sets in the original state space because the feature map is non invertible (just noticing that the domain is two dimensional and the codomain is six dimensional) and hence its kernel (in algebraic sense here) is not the null space and infinitely many solutions would exist for the possible separative estimated curve.

### 3.5.2 The shape is given

Let's consider the case in which the shape of the separative curve is given. This is a strong assumption but also a good starting point.
As in the classification literature has been made often times, we consider as first non linear case where the separative curve is a circle, or its generalization shape, an **ellipse**, as in Eq. 3.15:

$$\frac{(x - x_{center})^2}{a^2} + \frac{(z - z_{center})^2}{b^2} = 1. \tag{3.15}$$

Such equation has $4$ parameters. However, the ellipse has reflectional and rotational symmetries and also for any set of $4$ points there are infinitely many possible ellipses so the minimal number of points to uniquely define an ellipse is $5$.
The idea is to find $5$ points on the separative curve and then using them to retrieve the correct equation. To find such points we can use the proposed solution of Section 3.3 and its relative algorithm, Algorithm 1. By iterating it $5$ times, all the needed points are found.
We simulate in this way a scenario with a separative curve given by the ellipse with parameters:

$$x_{center} = 10\,\mathrm{m},\ z_{center} = 15\,\mathrm{m},\ a = 9\,\mathrm{m},\ b = 4\,\mathrm{m}. \tag{3.16}$$

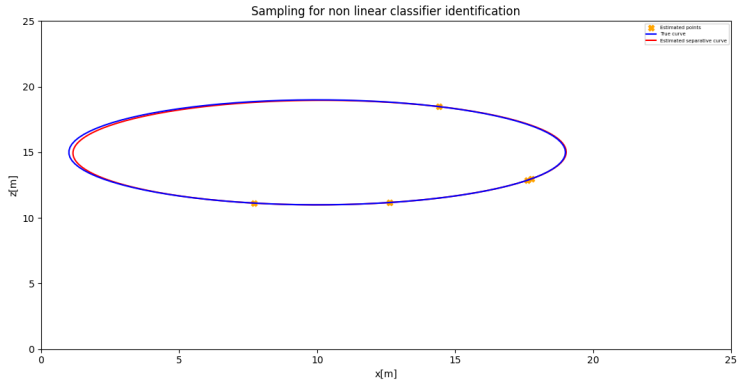


Figure 3.3: Estimated curve by using Algorithm 1 for $5$ points, with the knowledge the shape was an ellipse.

The obtained ellipse has been obtained by using the $5$ estimated points and then

solving the following algebraic system problem (see [10]):

$$\mathbf{E}\rho = \mathbb{1}, \tag{3.17}$$

where:

$$\mathbf{E} = \begin{bmatrix} x_1^2 & x_1 z_1 & z_1^2 & x_1 & z_1 \\ x_2^2 & x_2 z_2 & z_2^2 & x_2 & z_2 \\ x_3^2 & x_3 z_3 & z_3^2 & x_3 & z_3 \\ x_4^2 & x_4 z_4 & z_4^2 & x_4 & z_4 \\ x_5^2 & x_5 z_5 & z_5^2 & x_5 & z_5 \end{bmatrix}, \ \rho = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}, \ \mathbb{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \tag{3.18}$$

exploiting the fact that a generic ellipse can be written as:

$$ax^2 + bxz + cz^2 + dx + ez + f = 0, \tag{3.19}$$

and the last term $f$ can be normalized to $1$.

### 3.5.3 The shape is not given

We extend now our setup to the case in which the separative curve's shape is not known. Here, the best reasonable way is to use Algorithm 1 in order to estimate $m$ data points belonging to the separative curve, and then interpolate them to get an estimate of such a curve.
We try with an ellipse, as in 3.16, and we observe the results of two estimates, one with $m = 15$ data points and the second with $m = 40$ data points.

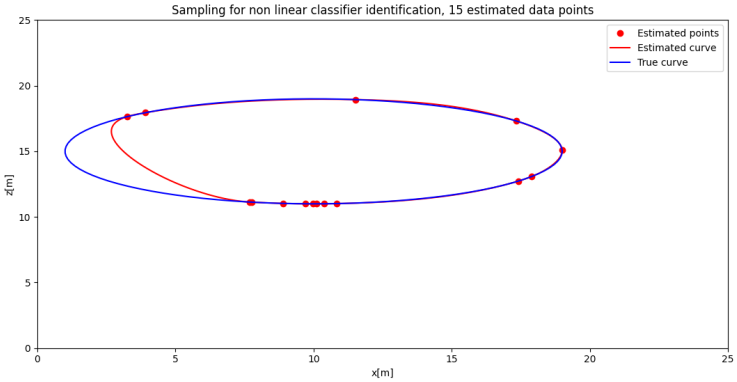


Figure 3.4: Estimated curve by using Algorithm 1 and interpolation of $m = 15$ data points.

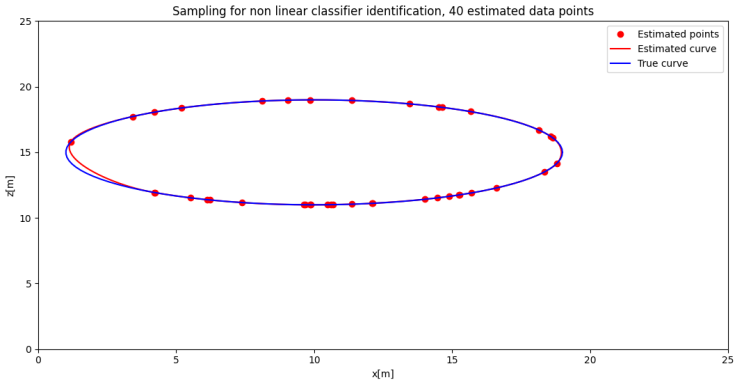


Figure 3.5: Estimated curve by using Algorithm 1 and interpolation of $m = 40$ data points.

As expected, the estimate with $m = 40$ data points is more accurate than the one with less data points, as in particular the left side of the ellipse has not been well sampled in the case with $m = 15$ data points, whereas with more estimated data points, the probability of cover the majority of the informative space is higher. For the interpolation, we have used a Spline kernel interpolation that uses low-degree polynomials in each of the intervals and chooses the polynomial pieces such that they fit smoothly together.

# 4

# Identification of linear classifiers with noiseless data

## 4.1 Intro

In this chapter, we consider linear classifiers with noiseless data. First of all, we define an agent with a state space dynamics and we develop Algorithm 2 to make it moving around the separative line changing its direction depending on the previously sampled data points. We study the rate of convergence of such algorithm and we present some simulated results.

## 4.2 Problem formulation in the linear case

The considered problem is the following one (see Figure 4.1).
Consider the $2D$ euclidean space, and in particular a region $\mathbb{S} \subseteq \mathbb{R}^2$, and we use as axes $x$ and $z$. Such region $\mathbb{S}$ is divided in two sub regions, $\mathbb{S}_1$ and $\mathbb{S}_2$, by the line described by the equation:

$$z = m^* x + q^*, \tag{4.1}$$

so that any point $P$ sampled on the region above the line, meaning that, $z_P - m^* x_P - q^* \geq 0$, has label $y_P = +1$ and every point $P$ sampled below the line, meaning that, $z_P - m^* x_P - q^* \leq 0$, has label $y_P = -1$.
A point is defined by the $x$-$z$ coordinates and its own label, i.e. by the triple

$P = (x_P, z_P, y_P)$, where sometimes we also use the notation $\mathbf{x}$ to denote the pair $\mathbf{x} = (x_P, z_P)$.
We are also given two initial points, $P_{0_1}$ and $P_{0_2}$, with opposite labels.
The agent is described in subsection 4.3 and starts moving from the point $x_{0_1}$. Our goal is to make it moving around the line, possibly reducing as much as possible his distance from the line and making it move with a direction that points towards the slope of the line, as depicted by Figure 4.1. Thus, strategically collecting data, we want to estimate $\hat{m}$ and $\hat{q}$ such that:

$$\lim_{j\to\infty} \|\hat{m}_j - m^*\| = 0 \text{ and } \lim_{j\to\infty} \|\hat{q}_j - q^*\| = 0, \tag{4.2}$$

or, at least, after $J$ data points:

$$\lim_{j\to J} \|\hat{m}_j - m^*\| < \epsilon \quad \text{and} \quad \lim_{j\to J} \|\hat{q}_j - q^*\| < \epsilon. \tag{4.3}$$

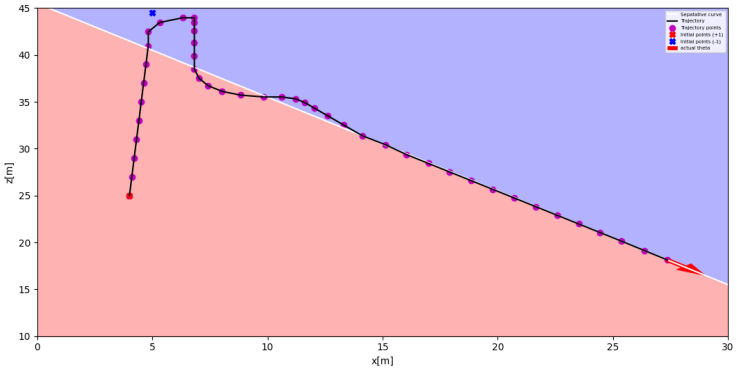


Figure 4.1: Example of navigation and of a convergent trajectory towards the separative line. $\mathbb{S}_1$ is marked in red while $\mathbb{S}_2$ in blue.

## 4.3 Agent's dynamics

The agent is described by a discrete-time first order integrator state space model, given by the following motion's equations:

$$\begin{aligned} x(k+1) &= x(k) + T \times v_1(k) \\ z(k+1) &= z(k) + T \times v_2(k). \end{aligned} \tag{4.4}$$

We also make use of another pseudo-state component, $\theta$, defined as:

$$\theta(k+1) = \arctan\left(\frac{z(k+1) - z(k)}{x(k+1) - x(k)}\right). \tag{4.5}$$

We can re-write the equations in a more compact way, that is in the state space model, call it $\Sigma$:

$$\zeta(k+1) = A\zeta(k) + B\mathbf{v}(k) + f(\mathbf{x}(k+1), \mathbf{x}(k)), \tag{4.6}$$

where

$$\zeta(k+1) = \begin{bmatrix} x(k+1) \\ z(k+1) \\ \theta(k+1) \end{bmatrix}, \ \mathbf{v}(k) = \begin{bmatrix} v_1(k) \\ v_2(k) \end{bmatrix} \tag{4.7}$$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \ B = \begin{bmatrix} T & 0 \\ 0 & T \\ 0 & 0 \end{bmatrix}, \tag{4.8}$$

and, finally,

$$f(\mathbf{x}(k+1), \mathbf{x}(k)) = \begin{bmatrix} 0 \\ 0 \\ \arctan(\frac{z(k+1)-z(k)}{x(k+1)-x(k)}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \arctan(\frac{v_2(k)}{v_1(k)}) \end{bmatrix}. \tag{4.9}$$

The third component of the state, $\theta(k)$, indicates the direction of the agent at instant $k$, and can vary between $[-\frac{pi}{2}, +\frac{pi}{2}]$. Its dynamics is clearly non linear, as is the composition of two non linear functions.
We spend some words about such function:

$$z = \arctan\left(\frac{y}{x}\right), \tag{4.10}$$

that is the function representing the direction of the agent.
Such a function has as domain

$$\mathcal{D} = \left\{(x, y) \in \mathcal{R}^2 | x \neq 0\right\}, \tag{4.11}$$

and as range,

$$C = \{(x, y) | -\frac{\pi}{2} \leq x \leq \frac{\pi}{2} \wedge -\frac{\pi}{2} \leq y \leq \frac{\pi}{2}\}. \tag{4.12}$$

Therefore we cannot impose null movement along the $x$ coordinate.
It can be also useful to visualize the $3D$ plot of this function, as we will want to manipulate the inputs, namely the velocities, to achieve a desired direction target:

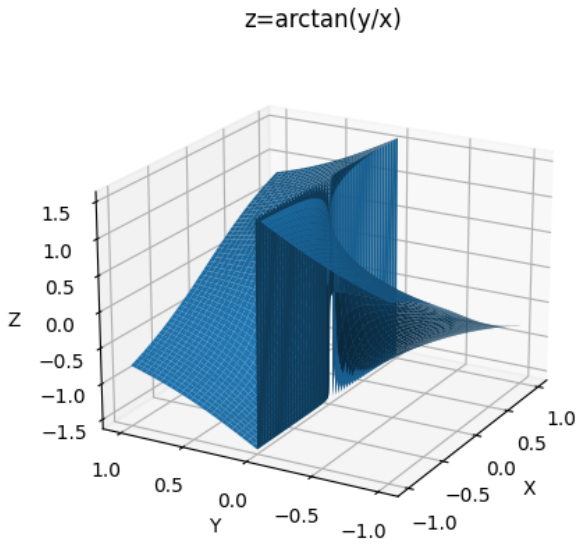


Figure 4.2: Arctangent function.

The function is non-convex, so non optimal local minimum could arise when a desired theta target will be searched.
However, also for limited velocities, the fact that is the ratio between them that is fed to the $\arctan(\cdot)$ function allows to easily span all the angles between $-\frac{pi}{2}$ and $+\frac{pi}{2}$.

In addition to that, the agent can measure at each iteration the label of the sample point where it is laying on in that instant, that is:

$$y(k) = \text{sign}\left(h(\mathbf{x}(\mathbf{k})\right), \tag{4.13}$$

where $h(\cdot)$ is a linear or non linear function (in this chapter is linear). Depending on the measured label, the agent is going to take a decision on where to move at the next iteration. The inputs are the velocities fed to the system and they have

a common constraint, caused by the finite power of the actuators, that is:

$$\|\mathbf{v}(k)\| \leq v_{max}. \tag{4.14}$$

we also suppose that they are not changing too abruptly, meaning that the accelerations are not too large, as this is in practice infeasible. To solve this issue, we could recur to a second order integrator or we can set some constraints between two consecutive velocities, imposing that their difference is not too large in norm. So, in order to keep the state space limited in size, we do not use a second integrator but we impose a constraint on the variation of the velocities, i.e.:

$$\begin{bmatrix} |v_1(k+1) - v_1(k)| \\ |v_2(k+1) - v_2(k)| \end{bmatrix} \leq \begin{bmatrix} \Delta v_{max} \\ \Delta v_{max} \end{bmatrix}. \tag{4.15}$$

This allows to approximate the acceleration as very small numbers,given the fact that:

$$a_1(k+1) = \frac{v_1(k+1) - v_1(k)}{T}. \tag{4.16}$$

The single integrator with 2 independent inputs is a very common reachable system, though, the addition of the third state component as in 4.4, makes arising a question:

**Is the system reachable?**

Despite the system being non linear, we notice that the dynamics of the third state component does not depend on the previous state (as the last row of $A$ is a null vector). So, as for the reachability of the last state component, it is enough to check if any state is not reachable by the control input just by inverting Eq. 4.9. Moreover, since the first two state components are independent from the last one, the intersection of the reachable states of the subsystem made by these two components and the reachable state of the last one is the overall reachable set.

Thus, by recalling the reachability theory in Appendix 8.5.1, it is possible to find out a counterexample where, given a specific target final state, we are not able to reach it from any initial state.

Let:

$$\zeta(\bar{k}) = \begin{bmatrix} x(\bar{k}) \\ z(\bar{k}) \\ \theta(\bar{k}) \end{bmatrix}, \tag{4.17}$$

this implies that:

$$v_2 = v_1 \tan(\theta(\bar{k})), \tag{4.18}$$

where, for simplicity, we call $\tan(\theta(\bar{k})) = c$.
Then, we can ignore the third state component, as does not depend on its previous state and we assume it to be reachable just by properly varying the velocities, and we can rewrite the state space model for the first two components in a different way, incorporating the constraint of Eq. 4.18, call it $\Sigma_c$, as follows:

$$\mathbf{x}(\bar{k}) = A_c \mathbf{x}(\bar{k}-1) + B_c v_1(\bar{k}-1), \tag{4.19}$$

where, assuming for simplicity $T = 1\,\mathrm{sec}$,

$$A_c = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; B_c = \begin{bmatrix} 1 \\ c \end{bmatrix}. \tag{4.20}$$

We compute now the reachability matrix in $k = 1$ steps:

$$\mathcal{R}_c(1) \triangleq [\mathbf{B_c}] = \begin{bmatrix} 1 \\ c \end{bmatrix} \tag{4.21}$$

So, it is evident that:

$$\mathrm{rank}\,(\mathcal{R}_c) \neq 2,$$

for any value of $c$.
Hence the system is not reachable in one step!
For more than one steps we have that the input matrix is $B_c$ just for the last step while for the others, it holds:

$$B_{c_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{4.22}$$

and so, ignoring the constraints on $v_{max}$ of Eq. 4.14 and on $\Delta v_{max}$ of Eq. 4.15,

the system is reachable in $2$ steps, as:

$$\mathcal{R}_c(2) \triangleq [B_c, A_c B_{c_1}] = \left[\begin{bmatrix} 1 \\ c \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right]. \tag{4.23}$$

However, we also remember that given a target $\theta(\bar{k})$, we can reach that direction if the other two state coordinates are such that $z(\bar{k}) = cx(\bar{k})$.
This will be important later on when the proposed solution will be introduced, where, in fact, only the third state component $\theta$ will be the target, without worrying of the other two.

## 4.4 Proposed solution

Here we introduce the proposed solution for the estimation of a separative line sampling the space with a agent whose dynamics has been described in Section 4.3. Here we state some assumptions, that make the problem more tractable:

- the agent starts from the point $P_{0_1}$, that is below the other initial point $P_{0_2}$, so it will starts moving in that direction;
- the measurement of the labels is provided by a sensor, that can be viewed as an oracle;
- no errors on the measurement of the labels occur.

The strategy we adopt is the following. The agent starts moving on the region $\mathbf{S}_1$ towards the point $P_{0_2}$, on the region $\mathbb{S}_2$.
It measures the label $y(k)$ of the position of the state where it is, with the following labelling function:

$$y(k) = \text{sign}\left(h(\mathbf{x}(k))\right) = \begin{cases} +1, & \text{if } z(k) - m^* x(k) - q^* \geq 0 \\ -1, & \text{if } z(k) - m^* x(k) - q^* < 0 \end{cases} \tag{4.24}$$

To move towards $P_{0_2}$, also called in Algorithm 2 $P_{opposite}$, it solves the following optimization problem, called $opt\_problem\_1$:

$$\begin{aligned} \min_{\mathbf{u}(k)} \quad & \|\mathbf{x}_{k+1} - \mathbf{x}_{P_{opposite}}\| \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = A_r \mathbf{x}_k + B_r \mathbf{u_k} \\ & \|\mathbf{u}(k)\| \leq v_{max} \end{aligned} \tag{4.25}$$

where $A_r = A_c$ and $B_r = B_{c_1}$.
**Note 1**: In this initial trajectory we do not consider the additional constraint on the variation on the velocities, as we expect that the agent will move to the target point with maximum allowed velocity (therefore going straight and with constant velocities) and we can also suppose that the initial velocities are aligned with the target ones.
**Note 2**: to solve numerically the optimization problem we use the optimization routines provided by Scipy, in Python, and in particular its $minimize(\cdot)$ function.

As soon as the agent measures a position in which the label is on the region $\mathbb{S}_2$, it turns to the right, namely it imposes a $\theta^*(0) = -pi/2$, since the two initial points are on the left part of the considered space environment, and decreases the direction at each time steps of some degrees, as max as the constraints on the velocities allow so, with the objective of crossing again the line by sampling a new point on the region $\mathbb{S}_1$.
To find the next states, it solves the following optimization problem, where $\theta^*$ is the true theta, called $opt\_problem\_2$:

$$\begin{aligned} \min_{\mathbf{u}(k)} \quad & \|\theta_{k+1} - \theta^*(g)\| \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = A_r \mathbf{x}_k + B_r \mathbf{u_k} \\ & \|\mathbf{u}(k)\| \leq v_{max} \\ & |u_1(k) - u_1(k-1)| \leq \Delta v_{max} \\ & |u_2(k) - u_2(k-1)| \leq \Delta v_{max} \end{aligned} \tag{4.26}$$

Once the agent realizes that it has crossed the line, it makes a first rough estimate of the range of directions where the line can be. We take the four points that have been sampled immediately after and later the crossing of the line, and we call them respectively, $n_1$, $n_2$, $n_3$ and $n_4$, as depicted in Figure 4.3. Therefore we know that the true angular coefficient $m^*$ is such that $m^* \in [m_{min}, m_{max}]$, where:

$$m_{min} = \frac{z_{n_2} - z_{n_4}}{x_{n_2} - x_{n_4}} \quad \text{and} \quad m_{max} = \frac{z_{n_1} - z_{n_3}}{x_{n_1} - x_{n_3}}, \tag{4.27}$$

or in terms of $\theta^*$, such that $\theta^* \in [\theta_{min}, \theta_{max}]$, where:

$$\theta_{min} = \arctan(m_{min}) \quad \text{and} \quad \theta_{max} = \arctan(m_{max}). \tag{4.28}$$

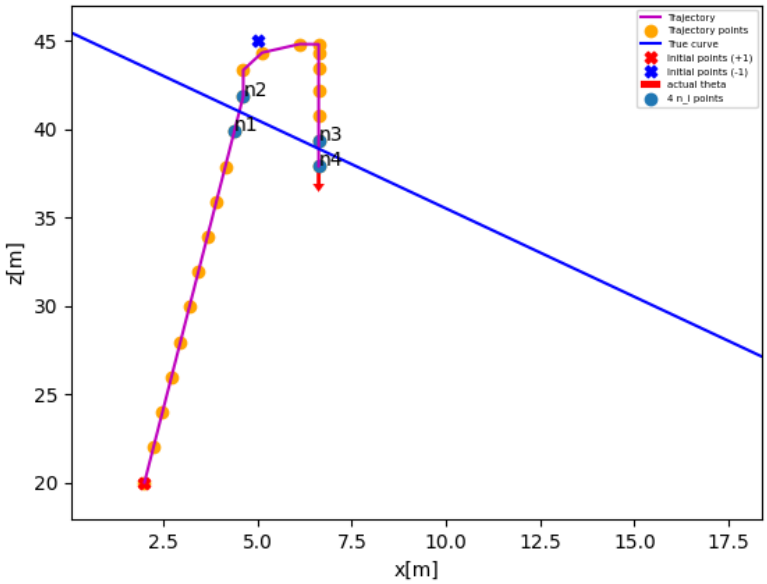


Figure 4.3: The $4$ $n_i$ trajectory points used to estimated the $\theta$ bounds.

The next step is to move the agent through some more states until it crosses again the line and comes back on the region $\mathbf{S}_2$. To do so, we just impose that it turns its $\theta$ direction towards the maximum amount of slope where the line can lay, but since we want to reduce this range of uncertainty of possible directions, we sum or subtract the bound by a new variable $\Delta\theta$. The target $\theta$, call it $\theta_{target}$, is defined by the following updating rule:

$$\theta_{target}(g) = \begin{cases} \theta_{max}(g) - \Delta\theta, & \text{if } g > 0 \, is \, odd \\ \theta_{min}(g) + \Delta\theta, & \text{if } g > 0 \, is \, even \end{cases} \tag{4.29}$$

where $g = 0, 1, \ldots, G$ is the numbered period that goes over the consecutive points of the trajectory of the agent on the same region, and $g = 0$ for the very first initial arc, meaning the one going from $n_2$ to $n_3$ on $\mathbb{S}_2$. $G$ is instead the last period, meaning that we considered the agent to have reached convergence in such a time slot.

Notice that, as pointed out in Section 4.3, requiring that the agent reaches the inclination given by $\theta_{target}$ is possible if we do not care about the other two states components, as in fact we do.

To guarantee that the agent crosses the line, since it might reach the target $\theta$ direction before of crossing the line, we then keep feeding the agent with the same

velocities, so that it keeps moving, along the same direction, until it measures a different label. After this step, two things can happen:

- either the agent observes a different label, meaning that it has crossed the margin. In this case, we iterate the previous procedure, by updating $\theta_{min}(g)$ and $\theta_{max}(g)$ and afterwards $\theta_{target}$;
- or the agent does not sample a different label anymore, meaning that the true direction, $\theta^*$, was between the current $\theta_{target}$ and $\theta_{target} \pm \Delta\theta$, where we pick either + if we are in a period where $g$ is odd (i.e. we are below the line and we had decreased too much the upper bound $\theta_{max}$), or − if instead $g$ is even (i.e. we are above the line and we had decreased too much the lower bound $\theta_{min}$). This is also the terminal criterion, meaning that we have found $G$.

Therefore, after the agent has turned right in the very first turn and we have reached again the region $\mathbf{S}_1$, we adopt, for $g \geq 2$ (since for $g = 1$ we just initialize the bounds $\theta_{min}$ and $\theta_{max}$ as in Equation (4.28) and we compute $\theta_{target}$ as in Equation (4.29), the following update rules for the bounds of $\theta$ and for $\theta_{target}$ any time the agent changes label:

1. we update $n_1$, $n_2$, $n_3$ and $n_4$ so that $n_1$ is the point on the lower left part, $n_2$ is the point on the upper left part, $n_3$ is the point on the upper right part and $n_4$ the one on the lower right part, as showed in Figure 4.3;
2. we compute new tentative for the bounds of the slope of the line as in Equation (4.27), and we accept them, meaning that we update the old one with these, if they are more restrictive.
3. we add or subtract $\Delta\theta$, depending on $g$;
4. we finally update $\theta_{target}$, as in Equation (4.29).

In order to better describe an algorithm we need the following functions, whose implementation in Python can be found in Section 8.11 of Appendix 8:

- the function $find\_bounds()$ (see 8.11, Algorithm 2), that computes the bounds $m_{min}$ and $m_{max}$;
- the function $find\_theta\_target()$ (see 8.11, Algorithm 2), that given the bounds on the direction and the current label, returns the target theta to be reached by the agent, with the additional improvement given by $\Delta\theta$.

In other words, we have described the following algorithm, namely Algorithm 2:

**Algorithm 2** Line navigation with no errors

**Require:** $X_{max} = 100, \Delta\theta = 1, \Delta v_{max} = 0.1, T = 1, v_{max} = 2$ {Can be set differently}
  $g \leftarrow 0,\ k \leftarrow 0,\ P(k) \leftarrow P_{0_1},\ P_{opposite} \leftarrow P_{0_2},\ \theta(k) \leftarrow None$ {$\theta$ is not defined for k=0}
  $\zeta(k) = (\mathbf{x}(P(k)), \theta(k))$
  $update\ y(\mathbf{x}(k))$ {see Eq. 4.24}
  $flag_+ \leftarrow y(k) == +1$
  **while** $True$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_1()$ {see Eq. 4.25}
    $k \leftarrow k + 1$
    $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
    $update\ y(\mathbf{x}(k))$
    $flag_+ \leftarrow y(k) == +1$
    **if** $y(k) \times \text{sign}\left(h(\mathbf{x}_{P_{opposite}})\right) > 0$ **then**
      $break$ {We have crossed the margin}
  $n_1 \leftarrow \mathbf{x}(k-1),\ n_2 \leftarrow \mathbf{x}(k)$
  **while** $x(k) < X_{max}$ **do**
    **if** $g \geq 1$ **then**
      $find\_bounds()$ {See Eq. 8.11}
    $find\_theta\_target()$ {See Eq. 8.11}
    **while** $x(k) < X_{max}$ and $flag_+ == False$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) = A\zeta(k-1) + \mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow y(k) == +1$
    **if** $g \geq 1$ **then**
      $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3$
    $n_3 \leftarrow \mathbf{x}(k-1),\ n_4 \leftarrow \mathbf{x}(k)$
    $g \leftarrow g + 1$
    $find\_bounds(),\ find\_theta\_target()$
    **while** $x(k) < X_{max}$ and $flag_+ == True$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow y(k) == +1$
    $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3,$
    $n_3 \leftarrow \mathbf{x}(k),\ n_4 \leftarrow \mathbf{x}(k-1)$
    $g \leftarrow + = 1$

**Note 1**: set $X_{max}$ large enough for the algorithm to converge, meaning to reach $G$.

**Note 2**: in the following we call, improperly but for sake of compactness, $\theta_{fin} = \theta(k)$, where $k$ is the total number of data points, as $\theta(G)$.

The final estimates of $\hat{\theta}$ and $\hat{q}$ can be retrieved as follows:

$$\hat{\theta} = \begin{cases} \frac{2\theta_G + \Delta\theta}{2}, & \text{if } G > 0 \; is \; odd \\ \frac{2\theta_G - \Delta\theta}{2}, & \text{if } G > 0 \; is \; even \end{cases}, \tag{4.30}$$
$$\hat{q} = z_{3,4_G} - \tan(\hat{\theta}) x_{3,4_G},$$

where $P_{3,4_G} = (x_{3,4_G}, z_{3,4_G}) = \frac{n_{3_G} + n_{4_G}}{2}$, $\theta_G$ is the last estimated direction, and $n_{3_G}$ and $n_{4_G}$ are the last computations of $n_3$ and $n_4$.

### 4.4.1 Guarantees on the results/Convergence proof

We prove that Algorithm 2 make the agent to converge in $\theta$ (and this implies its estimation) to a specific interval depending on $\Delta\theta$, and to estimate in a real-time fashion the parameter $q$, meaning that, although the agent cannot converge in $q$ as it has a residual error in $\theta$, the estimate of $q$ is still good and depends on the distance between $n_{3_G}$ and $n_{4_G}$.

**Proposition.** ***(Proof of the convergence)*** *Algorithm 2 makes $\theta(g)$ converge to a value such that:*

$$|\theta(G) - \theta^*| \leq 2\Delta\theta. \tag{4.31}$$

*Proof.* The Algorithm 2 decreases the range of variation of the variable $\theta$, namely $\{\theta_{min}(g)\}_{g=1,2,\dots,G}$ is an increasing sequence whereas $\{\theta_{max}(g)\}_{g=1,2,\dots,G}$ is decreasing. Thus, by the "Sandwich theorem" (or, in italian, "Carabinieri theorem"), the range of $\theta(G)$ is such that $\theta(G) \in [\theta_{min}(G), \theta_{min}(G)]$. In fact, the limit of the two sequences are respectively $\theta_{max}(G)$ and $\theta_{min}(G)$ and hence $\theta(G)$ is shrunk between them.
Moreover, by construction, also $\theta^*$ belongs to such an interval, whose width is $2\Delta\theta$, and so $|\theta(G) - \theta^*| \leq 2\Delta\theta$.

We can also say something more, namely that:

$$|\theta(G) - \theta^*| \leq \Delta\theta \tag{4.32}$$

if we are not in the unlucky case where $\theta^*$ is between $\theta_{min}(g)$ and $\theta_{max}(g)$ for any $g = 0, 1, \dots, G$, as in this case we need for all the $G$ periods given by Proposition 4.4.2. In fact, in all the other cases, the final period $G$ is achieved whenever

the improvement described by Eq. 4.29 is such that $\theta^*$ is between $\theta_{min}(G)$ and $\theta_{min}(G-1)$ if we are above the line, or is between $\theta_{max}(G)$ and $\theta_{max}(G-1)$ if we are below the line. Hence, $|\theta(G) - \theta^*| \leq \Delta\theta$. □

Thus, choosing $\Delta\theta$ small implies to obtain a final error on the slope of the line small in turn. However, it also holds that the smaller $\Delta\theta$, the larger the total number of periods $G$, as discussed in Section 4.4.2.

### 4.4.2 Rate of convergence

We can derive an upper bound on the maximum number of periods $G$ necessary to reach a convergence.

**Proposition.** *$G$ is upper bounded by*

$$G \leq 2\left(\frac{\theta_{max} - \theta_{min}}{\Delta\theta}\right). \tag{4.33}$$

*Proof.* We consider the worst case, namely the one where two assumptions hold:

1. the updating of the $n_1$, $n_2$, $n_3$ and $n_4$ does not give any improving on the estimates;
2. the initial bounds of the true theta, $\theta$, i.e. $[\theta_{min}(1), \theta_{max}(1)]$ are such that $\theta = \frac{\theta_{min}(1)+\theta_{max}(1)}{2}$.

At each period iteration $g$, we improve the estimate of either $\theta_{min}$ or $\theta_{max}$ of $\Delta\theta$, and every two iterations we improve both of them. So, if the true $\theta$ is the average of the two initial bounds, we need $\frac{\theta_{max}-\theta_{min}}{\Delta\theta}$ for one side and $\frac{\theta_{max}-\theta_{min}}{\Delta\theta}$ for the other (since the agent moves zigzagging). Since we also know that for $\theta^*(g=1) = \theta_{max} - \Delta\theta$, the very worst case is when the first assumption holds but in the second one $\frac{\theta_{min}(1)+\theta_{max}(1)}{2} - \Delta\theta \leq \theta \leq \frac{\theta_{min}(1)+\theta_{max}(1)}{2}$. □

**Note**: Proposition 4.4.2 means that Algorithm 2 converges in at most $2\left(\frac{\theta_{max}-\theta_{min}}{\Delta\theta}\right)$ periods.

### 4.4.3 Accuracy of the estimate of the intercept parameter

As far as the parameter $q$, we cannot say that it converges as in the sense of Eq. 4.2, as in fact it does not. Indeed, Algorithm 2 just cares of $\theta$ and if we have a final error in $\theta$, even if it is small (as small as the interval 4.32 allows), this small error may lead to move the agent quite far from the line if we look in long-distance terms, and so the intercept with the $x$ axis at $j = +\infty$, namely $q_\infty$, may be very different from the actual one.
However we can obtain a good estimate $\hat{q}$ not by looking at the final point of the agent but considering $\hat{q}$ as in Eq. 4.30 (estimating it in correspondence of the last change of side $G$), namely we can obtain convergence as in the sense of Eq. 4.3. We try to find the residual error of $\hat{q}$; from Eq. 4.30, the point $(x_{3,4_G}, z_{3,4_G}) = \frac{n_{3_G}+n_{4_G}}{2}$ deviates from the point belonging to the true line and intercepting the line between $n_{3_G}$ and $n_{4_G}$, call it $\left(x^*_{3,4_G}, z^*_{3,4_G}\right)$, of a quantity $(\tilde{x}_{3,4_G}, \tilde{z}_{3,4_G})$ such that:

$$(x_{3,4_G}, z_{3,4_G}) = \left(x^*_{3,4_G} + \tilde{x}_{3,4_G}, z^*_{3,4_G} + \tilde{z}_{3,4_G}\right). \tag{4.34}$$

At the same time, also $\hat{m}$ is affected by some error, as $\hat{\theta}$ does. Hence, we write:

$$\hat{m} = m^* + \tilde{m}. \tag{4.35}$$

Thus, knowing that $\hat{q} = q^* + \tilde{q}$:

$$\begin{aligned}
\tilde{q} &= \hat{q} - q^* = z_{3,4_G} - \hat{m}x_{3,4_G} - \left(z^*_{3,4_G} - m^*x^*_{3,4_G}\right) \\
&= z^*_{3,4_G} + \tilde{z}_{3,4_G} - (m^* + \tilde{m})\left(x^*_{3,4_G} + \tilde{x}_{3,4_G}\right) - \left(z^*_{3,4_G} - m^*x^*_{3,4_G}\right) \\
&= z^*_{3,4_G} + \tilde{z}_{3,4_G} - (m^* + \tilde{m})\left(x^*_{3,4_G} + \tilde{x}_{3,4_G}\right) - z^*_{3,4_G} + m^*x^*_{3,4_G} \\
&= \tilde{z}_{3,4_G} - m^*x^*_{3,4_G} - m^*\tilde{x}_{3,4_G} - \tilde{m}x^*_{3,4_G} - \tilde{m}\tilde{x}_{3,4_G} + m^*x^*_{3,4_G} \\
&= \tilde{z}_{3,4_G} - m^*\tilde{x}_{3,4_G} - \tilde{m}x^*_{3,4_G} - \tilde{m}\tilde{x}_{3,4_G}.
\end{aligned} \tag{4.36}$$

Now, given the fact that the error in $m$ is the result of the propagation of the error in $\theta$ through the tangent function, we can find that, if $\hat{\theta} \in \theta^* \pm \tilde{\theta}$, then the error in $\hat{m} = \tan\hat{\theta}$ is given by:

$$\left[\frac{d}{d\theta}\tan(\theta)\right]_{\hat{\theta}} \tilde{\theta} = \frac{\tilde{\theta}}{\cos(\hat{\theta})^2} = \frac{\Delta\theta}{\cos(\hat{\theta})^2}. \tag{4.37}$$

This expression remains small as long as $\theta$ does not approach $\pm\frac{\pi}{2}$. Indeed, $\Delta\theta$ is in general chosen small. Hence, if $\tilde{m}$ is small as much as we can, Eq. 4.36 yields:

$$\tilde{q} = \tilde{z}_{3,4_G} - m^* \tilde{x}_{3,4_G}, \tag{4.38}$$

and so the error in $\hat{q}$ only depends on the error of the point $(x_{3,4_G}, z_{3,4_G})$, that can be at most equal to $\frac{d(n_{3_G}, n_{4_G})}{2}$. Hence to have a good estimate of $q$, we can just impose a small sampling time $T$, or if it is fixed, small velocities, to make sure that the distance between two consecutive points is close enough.

**Note**: the analysis on the propagation of the error from $\theta$ to its $m$ has been computed in according to Linear Uncertainty Propagation, (see Appendix 8.9). It suggests us to be careful whenever the line to be estimated approaches slopes with $\theta \approx \pm\frac{\pi}{2}$, limiting the size of $\Delta\theta$ in such cases.

## 4.5 Simulations and results

In Figure 4.4, an example of the simulation of Algorithm 2, with a max velocity and a sampling time quite large, in order to also show the transient part, because otherwise the convergence rate is quite fast and does not allow to see in a proper way the oscillations of the trajectory around the line. The simulation's parameters for the agent's dynamics have been chosen as:

$$v_{max} = 2\,\mathrm{m\,s^{-1}}\ ,\ T = 1\,\mathrm{s}\ ,\ \Delta\theta = 1^\circ\ ,\ \Delta v = 0.5\,\mathrm{m\,s^{-1}}. \tag{4.39}$$

The resulted trajectory approaches the line and it does so very fast, since the distance between consecutive points is short and hence the estimation of the bounds for $m$ are close by. Moreover, the fact that no errors in the measurements of the labels are present ease the task.

We also consider a Monte Carlo approach to understand the behaviour of the error of the estimates of the simulation of Algorithm 2.
We run $n = 100$ simulations varying the angle of the line, $\theta$, from $[-70^\circ, 70^\circ]$, adding at each simulation an increment of the angle of $\frac{70^\circ-(-70^\circ)}{n} = 1.4^\circ$.
The initial two given points have been kept fixed in a location such that all the $n = 100$ different lines are between them. Such locations are: $P_{0_1} = (2, 10)$, $P_{0_2} = (5, 85)$.

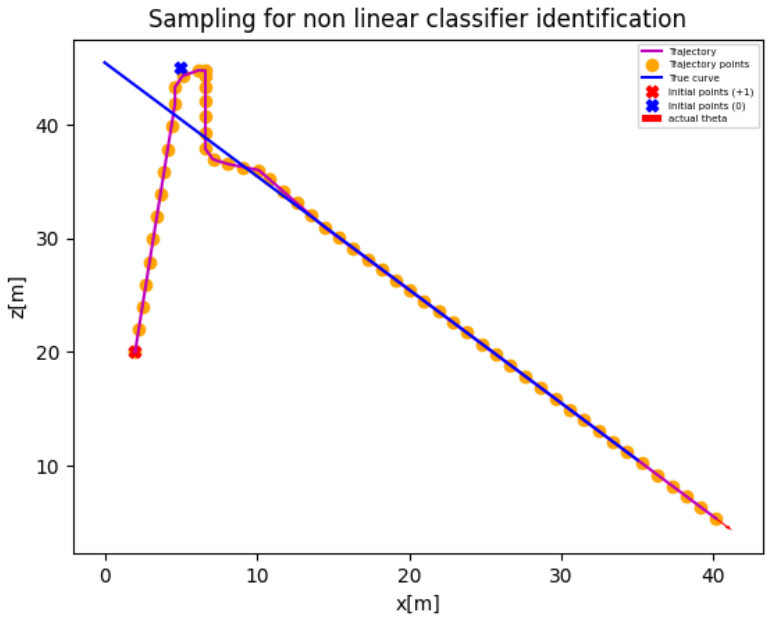


Figure 4.4: Estimate of a line by using Algorithm 2.

The histograms of the errors of the estimates of the parameters $\theta$ and $q$, namely $err_\theta$ and $err_q$ are presented below:

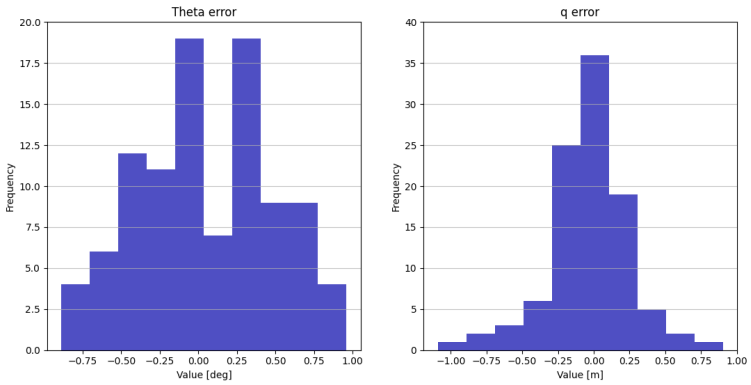


Figure 4.5: Histograms of the parameters' errors $m$ and $q$.

In the table 4.1 we also report the average of the absolute values of the errors, namely:

$$avg_{err_\theta} = \frac{1}{n}\sum_{i=1}^{n}|err_{\theta_i}| \text{ and } avg_{err_q} = \frac{1}{n}\sum_{i=1}^{n}\left|err_{q_i}\right|, \tag{4.40}$$

and the averages of the relative total errors, namely:

$$avg_{err_{rel_\theta}} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{err_{\theta_i}}{\theta_{true_i}}\right| \text{ and } avg_{err_{rel_q}} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{err_{q_i}}{q_{true_i}}\right|. \tag{4.41}$$

The results show how the Algorithm 2 works well achieving good performances

| $avg_{err_\theta}$ [°] | $avg_{err_{rel_\theta}}$ [°] | $avg_{err_q}$ [m] | $avg_{err_{rel_q}}$ [m] |
|---|---|---|---|
| 0.350 | 0.021 | 0.208 | 0.004 |

Table 4.1: Empirical errors.

in terms of final estimated parameters. The maximum errors for $\hat{\theta}$ are in fact inside the maximum interval prefixed by the setup of the Algorithm, namely $[-\Delta\theta+\theta^*, \Delta\theta+\theta^*]$, or, in other terms, their errors are always smaller, in absolute value, than $\Delta\theta = 1°$.
Moreover, arguments of Section 4.4.3 hold as the estimate $\hat{q}$ is in general very accurate.

**Note**: The choice on the sampling time, maximum velocities can influence the accuracy of the results, of course, as the agent may not have reached converge when the termination criterion has been reached. So, the termination criterion must be chosen according to the choice of the other parameters.
**Final note**: The proposed solution, from a control point of view, can be seen as an event-triggered control (see [6]). The direction theta is the only variable to be controlled, where the event that triggers a new control on theta is the observation of a different label, or of a series of consecutive labels. Otherwise, there is still a control that feeds our agents but it follows a constant control law. Moving the direction of the agent implies moving also its physical position. But is this enough to ensure that the agent track the separative line? The answer, as proved in this chapter is yes, as the possible controls on the direction of the agent make it to move oscillating around the separative line between two bounds on $\theta$ that are shrinking as long as the agents moves. This makes the agent approaching the line, namely reducing its physical distance from it, even though we have not knowledge on the actual position of this line to be tracked.

# 5

# Identification of linear classifiers with noisy data

## 5.1 Intro

In this chapter, the identification of linear classifiers with noisy data is investigated. In particular we want our agent to collect data navigating around the separative line, in order to get an estimate of the line. However, the collected data are no longer ideal, namely without errors, but the label have some random errors, thus the problem of following the separative line and get a good estimate becomes more involved. Three algorithms are developed in this chapter to solve the problem, with the relative theoretical and simulated results.

## 5.2 Noise on the labels measurements

In practice, the measurements that the sensors of an agent is provided, are influenced by some errors, and this is often unavoidable. Therefore, the extension of our setup to a more practical one is needed.
In our environment errors can happen for two main reasons:

- random uniformly distributed errors happen as all the electronic, optic, vision devices or sensors are not perfect;
- the distance from the separative margin is very short and so the accuracy of the measurements is not perfect; in fact sensors can be analog filters with a cutting frequency that is never a perfect classifier.

Therefore we can analyze these two cases separately, by using two different type of noise error models.

## 5.3 Uniform distributed errors

We consider the following noise error measurement model:

$$y(k) = \begin{cases} +\operatorname{sign}(h(\mathbf{x}(k)), & \text{if } \epsilon \geq e \text{ , where } \epsilon \sim \mathcal{U}(0\,,1) \\ -\operatorname{sign}(h(\mathbf{x}(k)), & \text{otherwise} \end{cases}, \tag{5.1}$$

where $e$ is the probability of measuring a wrong label, called error probability, that in the Appendix Section 8.6.3 and in general in the literature is referred as $q$, but since such symbol has been already used for the intercept of the line, we rename it as $e$.
We can also rewrite such error model as follows:

$$y(k) = \begin{cases} \text{True label}, & \text{if } \epsilon \geq e \text{ , where } \epsilon \sim \mathcal{U}(0\,,1) \\ \text{Fake label}, & \text{otherwise} \end{cases}, \tag{5.2}$$

that better shows the 0-1 (or on/off) relation between the value of $\epsilon$ and the final measurement $y(k)$.
In our analysis, we consider error probabilities that span from $0.1$ to $0.3$.

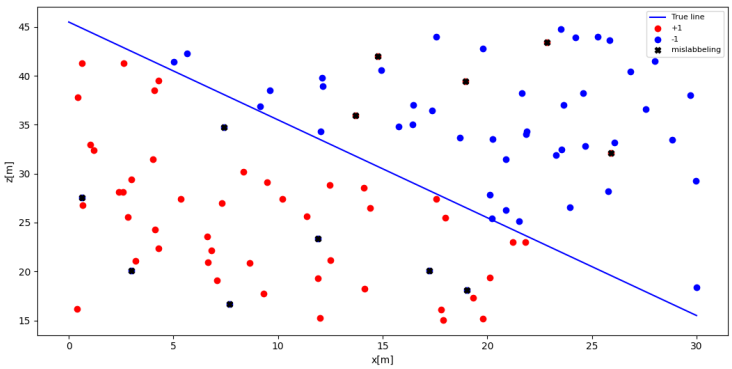


Figure 5.1: 100 sampled data points with a uniform noise error on the labelling, $e = 0.1$.

### 5.3.1 Problem formulation

The problem formulation is the same as in Section 4.2, so we aim make the agent to converge to the line with a finite error, thus estimating it in real-time. The only difference is that, in this new considered scenario, the measurements of the labels are done adopting Eq. 5.1, namely with the presence of uniform distributed noise.

### 5.3.2 Proposed solution

Since now the measurement outcome is a Bernoulli random variable (see Section 8.6.3, where the event $True\ label$ can be meant as $1$ and $Fake\ label$ can be meant as $0$, we know that the expected value and the variance of a measurement are:

$$\begin{aligned} \mathrm{E}[y(k)] &= \Pr(y(k)=1)\cdot 1 + \Pr(y(k)=0)\cdot 0 = p\cdot 1 + e\cdot 0 = p, \\ \mathrm{Var}[y(k)] &= \mathrm{E}\left[y(k)^2\right] - \mathrm{E}[y(k)]^2 = \mathrm{E}[y(k)] - \mathrm{E}[y(k)]^2 \\ &= p - p^2 = p(1-p) = pe, \ . \end{aligned} \tag{5.3}$$

To understand what is the expected value and the variance of measuring two consecutive data points, three consecutive data points, and so on and so forth, we have to think of these events as Binomial random variables (see 8.6.4). So, let's consider for instance $e = 0.1$ and $e = 0.3$, $n = 2$ and $n = 3$. The expected values are:

$$\begin{aligned} &\mathrm{E}_{e=0.1}[y(k), y(k+1)] = n\cdot p = 2\cdot 0.9 = 1.8, \\ &\mathrm{E}_{e=0.3}[y(k), y(k+1)] = n\cdot p = 2\cdot 0.7 = 1.4, \\ &\mathrm{E}_{e=0.1}[y(k), y(k+1), y(k+2)] = n\cdot p = 3\cdot 0.9 = 2.7, \\ &\mathrm{E}_{e=0.3}[y(k), y(k+1), y(k+2)] = n\cdot p = 3\cdot 0.7 = 2.1, \end{aligned} \tag{5.4}$$

whereas, the probabilities of getting $n = 2$ and $n = 3$ consecutive errors are:

$$\begin{aligned}
&\Pr_{e=0.1}(y(k)=0, y(k+1)=0) = \begin{pmatrix} n \\ t \end{pmatrix} p^t (1-p)^{n-t} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} 0.9^0 (0.1)^2 = 0.01, \\
&\Pr_{e=0.1}(y(k)=0, y(k+1)=0, y(k+2)=0) = \begin{pmatrix} 3 \\ 0 \end{pmatrix} 0.9^0 (0.1)^3 = 0.001, \\
&\Pr_{e=0.3}(y(k)=0, y(k+1)=0) = \begin{pmatrix} 2 \\ 0 \end{pmatrix} 0.7^0 (0.3)^2 = 0.09, \\
&\Pr_{e=0.3}(y(k)=0, y(k+1)=0, y(k+2)=0) = \begin{pmatrix} 3 \\ 0 \end{pmatrix} 0.7^0 (0.3)^3 = 0.027.
\end{aligned} \tag{5.5}$$

Interesting are also the probabilities of getting $n = 2$ and $n = 3$ consecutive success outcomes:

$$\begin{aligned}
&\Pr_{e=0.1}(y(k)=0, y(k+1)=0) = \begin{pmatrix} n \\ t \end{pmatrix} p^t (1-p)^{n-t} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} 0.9^2 (0.1)^0 = 0.81, \\
&\Pr_{e=0.1}(y(k)=0, y(k+1)=0, y(k+2)=0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} 0.9^3 (0.1)^0 = 0.729, \\
&\Pr_{e=0.3}(y(k)=0, y(k+1)=0) = \begin{pmatrix} 2 \\ 2 \end{pmatrix} 0.7^2 (0.3)^0 = 0.49, \\
&\Pr_{e=0.3}(y(k)=0, y(k+1)=0, y(k+2)=0) = \begin{pmatrix} 3 \\ 3 \end{pmatrix} 0.7^3 (0.3)^0 = 0.343.
\end{aligned} \tag{5.6}$$

Therefore, we notice that, given any tuple or triple of consecutive data points the probabilities of measuring all errors or all success outcomes are given by the previous two equations.
Now, a strategy that can arise from these simple results to move the agent on the space is the following: we can move it from the first initial given point, to the second one, and during the path we can observe some different labelled data points, but we say that we trust a different label only if there have been at least $n$ consecutive data points with same label, and just in that case we update the optimization problem to move the agent to the next state, otherwise we keep

following the same optimization problem ignoring the different label data point that we assume is not reliable.
The reason for this strategy is as follows: by observing Eq. 5.5 and Eq. 5.6 the probability of measuring $n$ consecutive mislabelled data, that could lead to a failure in the case this would happen far from the separative line, is very low, whereas the probability of measuring $n$ consecutive true labels is quite large, so that we can have a reasonable trigger on when we need to update the optimization problem, namely we need to change $\theta_{target}$.
However, we also notice that the probability of measuring $n$ consecutive true labels decreases as well as $n$ increases, so that we cannot use to many consecutive samples to decree whether the margin has been crossed or not, because this would imply to move too far away from the line.
Summing up, we formalize here the rules that the new algorithm to navigate and estimate a line with uniform distributed errors, named Algorithm 3, must obey:

1. we use $n = 4$ consecutive points to establish whether the margin has been crossed. In this way, we ensure to have a high likely probability that the estimate of the two bounds $\theta_{min}$ and $\theta_{max}$ is good enough;

2. in the first crossing, we save as $n_1$ the last but one data point sampled with label $+1$, namely on the side below the curve, of the sequence of at least $n$ consecutive points with label $+1$ (this because, if we saved the last one, there might be the possibility that some errors around the points $n_1$, $n_2$, $n_3$ and $n_4$ would make the bounds wrong);

3. in the other crossings, $g \geq 1$ and if we are on the side with label $-1$, namely above the line in our setting, we save as $n_3$ the last but one data point sample with label $-1$, namely on the side below the curve, of the sequence of at least $n$ consecutive points with label $-1$;

4. in the other crossings, $g \geq 1$ and if we are on the side with label $+1$, namely below the line in our setting, we save as $n_4$ the last but one data point sample with label $+1$, namely on the side above the curve, of the sequence of at least $n$ consecutive points with label $+1$.

The other steps we follow are the same of Algorithm 2 but with this new rules of trust a change in the navigation just when $n$ consecutive data points are observed with opposite label from the one we started the previous period.
To describe write down an algorithm, we update or define the following functions:

- $labelling()$ (see 8.11, Algorithm 3);

- $find\_prec()$ (see 8.11, Algorithm 3), that finds, in a change of side, the previous reliable point of the opposite label when we detect $n$ consecutive points, as we cannot take the $(k-n)-th$ point because some errors might have occurred;
- $update\_flag\_plus()$ (see 8.11, Algorithm 3);
- $find\_bounds()$ (see 8.11, Algorithm 3), taking into account the convergence achievement;
- $find\_theta\_target()$ (see 8.11, Algorithm 3).

Thus, we can state the Algorithm 3 as:

**Algorithm 3** Line navigation and identification with uniform distributed errors

**Require:** $X_{max} = 100,\ \Delta\theta = 1,\ \Delta v_{max} = 0.1,\ T = 1,\ v_{max} = 2,\ n = 4$ {Can be set differently}
  $g \leftarrow 0,\ k \leftarrow 0,\ P(k) \leftarrow P_{0_1},\ P_{opposite} \leftarrow P_{0_2},\ \theta(k) \leftarrow None$ {$\theta$ is not defined for k=0}
  $\zeta(k) = (P(k), \theta(k))$
  $y(k) \leftarrow y_{P_{opposite}}$
  $flag_+ \leftarrow y(k) == +1$
  **while** $flag_+$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_1()$ {see Eq. 4.25}
    $k \leftarrow k + 1$
    $\zeta(k) \leftarrow A\zeta(k-1) + B\mathbf{u}(k-1)$
    $update\ y(\mathbf{x}(k))$ {see Eq. 5.1 or code snippet 8.11}
    $flag_+ \leftarrow y(k) == +1$
    $flag\_break \leftarrow True$
    **for** $i = 0;\ i+ = 1;\ i < n$ **do**
      **if** $y(k-i) \times y_{P_{opposite}} < 0$ **then**
        $flag\_break \leftarrow False$ {If this flag remains True we have crossed the margin}
        $break$
    $flag_+ \leftarrow flag\_break$
  $flag\_convergence \leftarrow False,\ flag\_improv \leftarrow False$
  $n_1 \leftarrow find\_prec()$ {see 8.11}, $n_2 \leftarrow \mathbf{x}(k)$
  **while** $x(k) < X_{max}$ **do**
    **if** $g \geq 1$ **then**
      $find\_bounds()$ {See 8.11}
    $find\_theta\_target()$ {See 8.11}
    **while** $x(k) < X_{max}$ and $flag_+ == False$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) \leftarrow A\zeta(k-1) + B\mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow update\_flag\_plus()$ {see 8.11}
    **if** $m \geq 1$ **then**
      $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3$
    $n_3 \leftarrow find\_prec(),\ n_4 \leftarrow \mathbf{x}(k)$
    $g \leftarrow g + 1$
    $find\_bounds(),\ find\_theta\_target()$
    **while** $x(k) < X_{max}$ and $flag_+ == True$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) \leftarrow A\zeta(k-1) + B\mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow update\_flag\_plus()$
    $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3,$
    $n_3 \leftarrow \mathbf{x}(k),\ n_4 \leftarrow find\_prec()$
    $g \leftarrow g + 1$

### 5.3.3 Convergence/Guarantee of results

We notice that in this new setup, the fact that the agent converges as in Section 4.2 is not anymore a deterministic event but depends on the number of con-

secutive errors the agent samples during its trajectory. We state here a proposition on the relation between convergence of Algorithm 3, the number of data points $N$ needed to reach the convergence interval $[\theta^* - \Delta\theta, \theta^* + \Delta\theta]$ and the error probability $e$, as follows:

**Proposition.** *Let $\boldsymbol{P} \in \mathbb{R}^{(n+1)\times(n+1)}$ be the transition matrix of the absorbing Markov chain describing the probability of getting $n = 4$ consecutive errors. If Algorithm 3 terminates in N data points, then the probability of convergence, meant as in Section 4.2 is:*

$$\mathrm{P}(N, e, n) = 1 - [1, 0, 0, 0, 0]\, \boldsymbol{P}^N\, [0, 0, 0, 0, 1]' \,. \tag{5.7}$$

*Proof.* For the proof we use Markov chain theory (see Section 8.6.6).
First of all, we claim that once the agent has reached the bounds such that $\theta_{max}(G) - \theta_{min}(G) \leq 2\Delta\theta$, then all the following direction theta targets do not exit the interval. In fact, by construction of the algorithm, the bounds are not updated anymore once $G$ has been reached for the first time.
Hence, if the agent reaches the convergence interval, then it cannot exit it.
After that, we need to prove the Markov reasoning.
To do so, let us define the $5$ states $s_0, s_1, s_2, s_3, s_4$ as:

- $s_0$ represents the event: $0$ error labelling;
- $s_1$ represents the event: $1$ error labelling;
- $s_2$ represents the event: $2$ consecutive error labellings;
- $s_3$ represents the event: $3$ consecutive error labellings;
- $s_4$ represents the event: $4$ consecutive error labellings.

Once the state $s_4$ has been reached we cannot escape from it: this means it is an absorbing state. From the perspective of the agent's trajectory, if it reaches the absorbing state, its control law on the direction $\theta$ is wrongly triggered (as if it were in an opposite label's side) and it starts moving in a wrong direction as in the example of Figure **??**5.5.
The one-step transition probability is $P_{ij}^{n,n+1} = P_{ij}$, as the process is stationary (namely, the probability of going from one state to another is fixed with respect to time). In our case,

$$P_{ij} = \Pr\{X_{n+1} = j \mid X_n = i\} = \begin{cases} p, & \text{if } X_n = 0, 1, 2, 3 \text{ and } X_{n+1} = 0 \\ e, & \text{if } X_n = 0, 1, 2, 3 \text{ and } X_{n+1} = X_n + 1 \\ 1, & \text{if } X_n = 4 \text{ and } X_{n+1} = X_n \end{cases}, \tag{5.8}$$

and, of course, all the other case, have probabilities equal to zero as we already have in 5.8 that the sum of all the one-step transition probabilities from one state are equal to 1 (namely the sum of any row of the relative transition matrix is equal to 1). We draw the graph describing such Markov process:

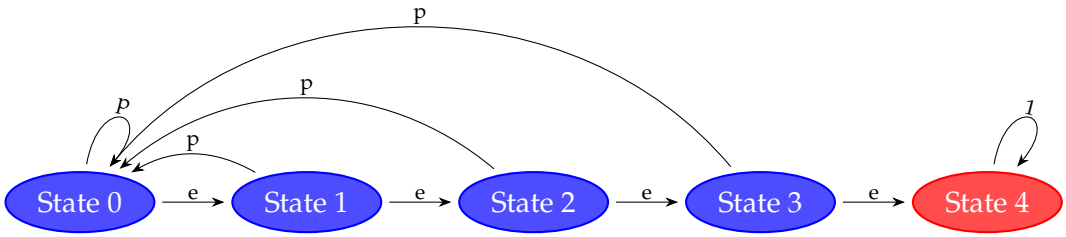


Figure 5.2: Markov chain of the considered process.

This is an absorbing Markov chain and its transition matrix is:

$$\mathrm{P} = \begin{pmatrix} p & e & 0 & 0 & 0 \\ p & 0 & e & 0 & 0 \\ p & 0 & 0 & e & 0 \\ p & 0 & 0 & 0 & e \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5.9}$$

The probability of measuring 4 consecutive wrong labels in $N$ possible data points with error probability $e$ is equal to the probability that, starting from the state 0, the Markov chain 5.2 ends up in the absorbing state 4 after $N$ transitions. Such probability is given by:

$$[1,0,0,0,0]\,\mathbf{P}^N\,[0,0,0,0,1]'\,. \tag{5.10}$$

This is explainable with the fact that such probability can be found on the cell of the matrix $\mathbf{P}$ elevated to the $N$ (as $N$ steps are computed) in the first row and in the last column (as is the probability of going from the first state of the chain to the last one). Hence the probability of not measuring these consecutive wrong labels, that would imply the failure of the Algorithm 3, as we would not have any guarantee about the estimated bounds and the consecutive $\theta^*$, is:

$$\mathrm{P}(N,e,n) = 1 - [1,0,0,0,0]\,\mathbf{P}^N\,[0,0,0,0,1]'\,. \tag{5.11}$$

□

**Remark**: this proof tells us the probability of convergence of Algorithm 3 knowing, or estimating the number of data points $N$ needed to reach the convergence interval. Such number depends on different factors, as pointed out in Section 5.3.5.

Let's suppose for instance $N = 100$, $e = 0.1$. Then, the probability of success of Algorithm 3 (where $n = 4$) is:

$$\begin{aligned}
\mathrm{P}(100, 0.1, 4) &= 1 - [1,0,0,0,0]\,\mathbf{P}^{100}\,[0,0,0,0,1]' \\
&= 1 - [1,0,0,0,0] \begin{pmatrix} 0.9 & 0.1 & 0 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 & 0 \\ 0.9 & 0 & 0 & 0.1 & 0 \\ 0.9 & 0 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{100} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
&= 1 - [1,0,0,0,0] \begin{pmatrix} 0.8922 & 0.08923 & 0.00892 & 0.000893 & 0.0087 \\ 0.8914 & 0.08915 & 0.00892 & 0.000892 & 0.0096 \\ 0.8834 & 0.08835 & 0.00884 & 0.000884 & 0.0185 \\ 0.8031 & 0.08032 & 0.00803 & 0.000803 & 0.1078 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
&= 1 - 0.00871 = 0.99129.
\end{aligned} \tag{5.12}$$

If instead, still $N = 100$, $e = 0.1$, but we consider $n = 3$ consecutive errors:

$$\begin{aligned}
\mathrm{P}(100, 0.1, 3) &= 1 - [1,0,0,0]\,\mathbf{P}^{100}\,[0,0,0,1]' \\
&= 1 - [1,0,0,0] \begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 \\ 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{100} [0,0,0,1]' \\
&= 1 - [1,0,0,0] \begin{pmatrix} 0.8245 & 0.0825 & 0.0083 & 0.0848 \\ 0.8170 & 0.0818 & 0.0082 & 0.0930 \\ 0.7427 & 0.07434 & 0.0074 & 0.1755 \\ 0 & 0 & 0 & 1 \end{pmatrix} [0,0,0,1]' \\
&= 1 - 0.0848 = 0.9152.
\end{aligned} \tag{5.13}$$

### 5.3.4 Comments on the success of the Algorithm 3 and on the choice the simulation's parameters

It is useful to analyze the meaning of Proposition 5.3.3 in terms of how to exploit this result to choose some simulation's parameters.
We plot the behaviour of the Algorithm 3 in function of the number of data points $N$ with different error probabilities $e$ and with different "trust's rule", namely different amounts of consecutive data points with same label that can provoke (as in most of the cases happens) the failure of the algorithm. We display the 4 plots with the following pairs:

$$\begin{aligned} &n = 3 \text{ and } e = 0.1, \quad n = 4 \text{ and } e = 0.1 \\ &n = 3 \text{ and } e = 0.3, \quad n = 4 \text{ and } e = 0.3 \end{aligned} \tag{5.14}$$

and we obtain:

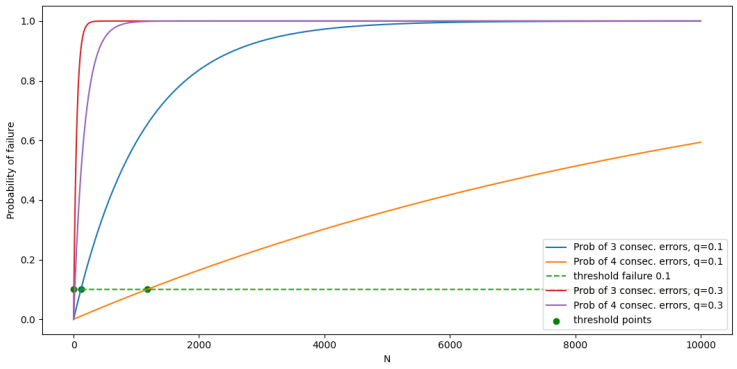


Figure 5.3: Rate of failure of Algorithm 3 with different implementation's choices.

It is evident that the more we increase $e$ the more is likely our algorithm fails, as a large error probability leads to too many mislabelled measurements that make too difficult to understand the side where we are and to make a good enough policy to navigate along the line.
Moreover, the choice of $n$ influence the rate of probability of a failure: the larger it is, the smaller the probability of getting $n$ consecutive mislabellings, and the larger the number of data points $N$ that the agent can collect while still being in a safe condition in terms of success.
However, on the other hand, we notice that the larger $n$, the larger is the range

created by the two bounds $\theta_{min}$ and $\theta_{max}$, as the points $n_1$, $n_2$, $n_3$ and $n_4$ are farther by, leading to an increasing of $N$ for reaching the safe interval of $[\theta^* - \Delta\theta, \theta^* + \Delta\theta]$. This problem is not evident with small $n$, like in our choices of Section 5.3.5, and so, as a general theoretic law, we can say that, for small values like $3$,$4$,$5$, the larger $n$, the better it is for our purposes.
We also notice that the choice of the sampling time $T$, and of the bounds on the velocities is important and in particular, larger values of them lead to data points with large distance, and so less data points in a portion of the space, but at the same time, the total number of points to reach convergence increases in this case as the updates of the bounds are worse.
A good strategy might be, to use larger velocities and sampling time in the initial part, where the agent is moving among the two initial given points, and then speed down and increase the sampling frequency (if allowed, as some agents can have fixed values), to increase the accuracy of the bounds, paying the price of enlarging $N$ and the probability of incurring into a failure's condition.
Finally, we report below the number of points allowed to reach convergence with the different implementation of 5.14, with a risk of failure of, at most $0.1$ to have a numeric idea of the behaviour of the plots:

| $n = 3,\ e = 0.1$ | $n = 4,\ e = 0.1$ | $n = 3,\ e = 0.3$ | $n = 4,\ e = 0.3$ |
|---|---|---|---|
| 117 | 1171 | 5 | 19 |

Table 5.1: Maximum number of points $N$ to reach the safe interval with probability $\leq 0.1$.

### 5.3.5 Simulations and results

We consider a Monte Carlo approach to understand the behaviour of the error of the estimates of the simulation of Algorithm 2.
The simulation's parameters for the agent's dynamics have been chosen as:

$$v_{max} = 1\,\mathrm{m\,s^{-1}}\ ,\ T = 0.3\,\mathrm{s}\ ,\ \Delta\theta = 0.5^\circ\ ,\ \Delta v = 0.1\,\mathrm{m\,s^{-1}}. \tag{5.15}$$

We run $100$ simulations and we vary the angle of the line, $\theta$, from $[-70^\circ, 70^\circ]$, adding at each simulation an increment of the angle of $\frac{70^\circ-(-70^\circ)}{n} = 1.4^\circ$.
The initial two given points have been kept fixed in a location such that all the $n = 100$ different lines are between them. Such locations are: $P_{0_1} = (2, 10)$, $P_{0_2} =$

$(5, 85)$.
An example of a trajectory, that achieved convergence, is the following one:

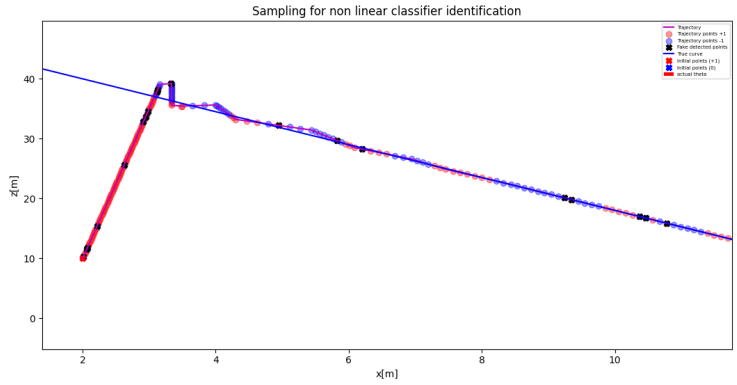


Figure 5.4: Example of trajectory achieving convergence using Algorithm 3 with 3 consecutive points rule. In black the data points that have been measured with wrong labels.

An example, of a trajectory that could not converge, is the following one:

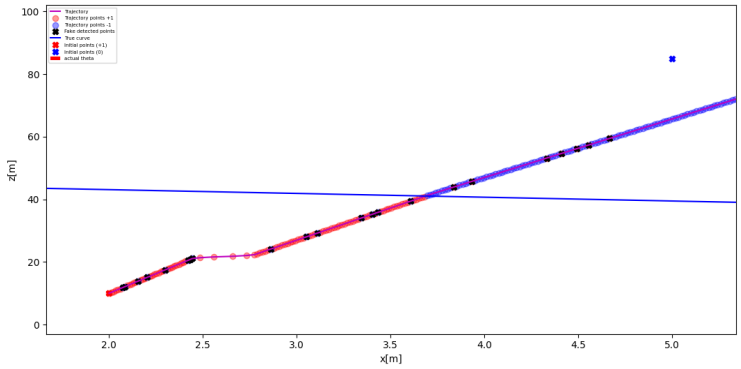


Figure 5.5: Example of trajectory not achieving convergence using Algorithm 3 with 3 consecutive points rule.

The histograms of the errors of the estimates of the parameters $theta$ and $q$, namely $err_\theta$ and $err_q$ are presented below:

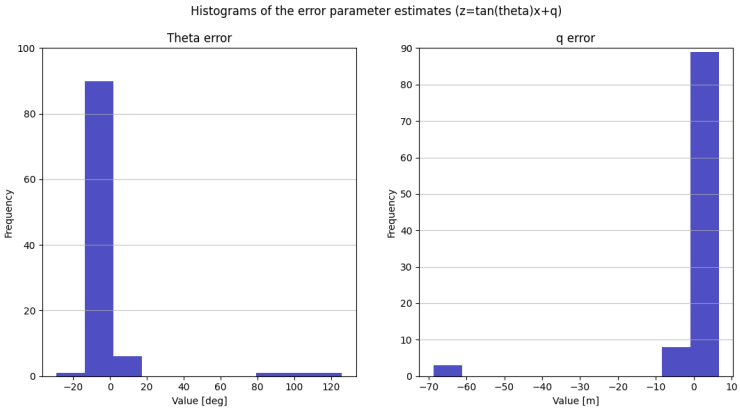


Figure 5.6: Histograms of $100$ runs of Algorithm 3 with $3$ consecutive points rule.

We notice how the peaks are around the errors both for $\theta$ and $q$ parameters, meaning that the Algorithm 3 works most of the times.
However some errors are present. In fact, by knowing that the agent with the chosen simulation's parameters of Eq. 5.15 and the initial points locations the agent needs a total number of data points $N$ such that $100 \leq N \leq 300$, and we use just $n = 3$ consecutive points to trust a measurement, the probability of convergence $\mathrm{P}(N, e, n)$ is, by observing Figure 5.3, such that $0.762 \leq \mathrm{P}(N, e, n) \leq 0.9152$.
If instead we use the rule of considering $n = 4$ consecutive data points, keeping all the other simulation's parameters as before, namely as 5.15, and we repeat the same $100$ experiments, we obtain the following histograms:

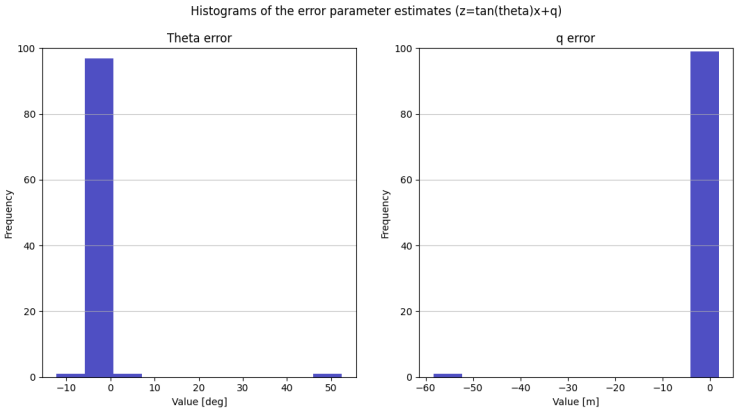


Figure 5.7: Histograms of $100$ runs of Algorithm 3 with $4$ consecutive points rule.

We notice how the results are better with respect to Fig. 5.6 and the $\theta$ error

is inside the interval $[\theta_{true} - \Delta\theta, \theta_{true} + \Delta\theta]$.
In particular in $100$ simulations, there has been just one error, and this is according to the probability of failure with such an error probability, namely $e = 0.1$, and with such consecutive number law's choice, namely $n = 4$, (see 5.3).
However the estimates of the $q$ parameter have some outliers with bigger errors. But this, is nothing unexpected, and, recalling the arguments of Section 4.4.3, can be explained as $\hat{q}$ now has been estimated taking $n_{3_G}$ and $n_{4_G}$ after, at least $n$ consecutive data points with same label, hence augmenting the distance between them and as a consequence also worsening the estimate of $\hat{q}$.
However this could also be resolved by using a smaller sampling time $T$ or a smaller velocities, at the price of having a larger number $N$ of data points to reach convergence.
Otherwise, other estimating techniques may be used, for instance an estimation of $q$ that considers more points, such as a regression approach, at the price of increase the final computational cost (whose smallness was our goal).

## 5.4 Gaussian distributed errors

We extend now our algorithm to estimate and navigating a line in the case where noise is present on the measurement of the labels. We model the error as follows:

$$y(k) = \text{sign}(h(\mathbf{x}(k)) + \epsilon), \text{ where } \epsilon \sim \mathcal{N}(0\,, \sigma^2)\,. \tag{5.16}$$

We recall Section 8.6 to a better understanding of the Gaussian distribution. Since the function $h(\mathbf{x}(k))$ is deterministic, then we have that:

$$(h(\mathbf{x}(k)) + \epsilon) \sim \mathcal{N}(h(\mathbf{x}(k))\,, \sigma^2). \tag{5.17}$$

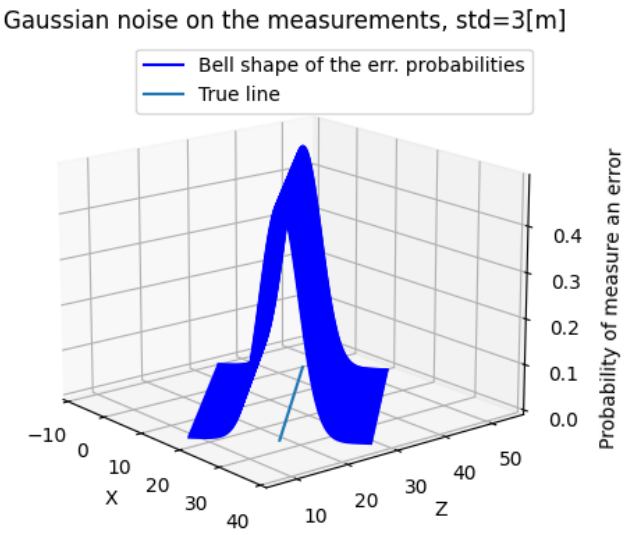


Figure 5.8: Shape of the error probabilities with a gaussian distributed noise in the labelling.

Thus, we can see that the probability of an error is proportional to the distance of the current position from the line. In particular, the larger the distance of the agent from the line, the lower the probability of get an error from the measurement.
If, for instance we have a point on the line, the probability of getting a mislabelled observation is maximum and is equal to $0.5$, as the $sign()$ function has an input that is a guassian random variable with mean the value of $h(\mathbf{x}(k))$, where $\mathbf{x}(k)$ is

a point on the line, and since any sample realization from this distribution can be with probability $0.5$ on one side of the bell, and with probability $0.5$ on the other one, in turns this is reflected on the probability of getting either a correct or wrong label.
This is a concrete modelling of the error as, in practice, a binary decision sensor commits more errors when the agent is close to the margin, while it does not commit any error when the agent is far away from it.
The parameter $\sigma^2$ is the variance of the normal distribution, and of course the larger it is, the larger is the field where an error can occur.
Below, an image representing the sampling of $150$ points whose labelling is subjected to gaussian noise, with $\sigma = 3.5\,\mathrm{m}$, with some errors:

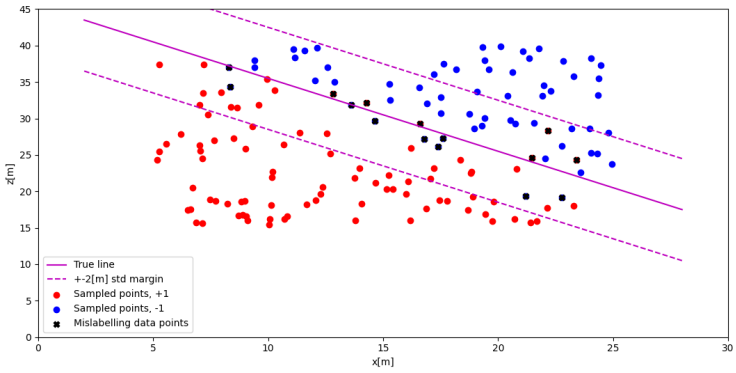


Figure 5.9: Sampling with gaussian noise labelling, $\sigma = 3.5\,\mathrm{m}$. In black the wrong labelled data points.

### 5.4.1 Problem formulation

For the nature of the noise in this scenario, the convergence of the trajectory computed by the agent to the separative line seems unfeasible unless we consider a tracking error very large, as large as proportional to the noise's standard deviation $\sigma$. Hence, the problem formulation is:
To move the agent collecting data points so that eventually is possible to get a good estimate of the separative curve, whereas navigating around the line and with a distance from it of less than a certain $\epsilon_\sigma$, namely:

$$\lim_{j\to\infty} \left| z_j - m^* x_j - q^* \right| \leq \epsilon_\sigma. \tag{5.18}$$

### 5.4.2 Proposed solution

The solution of Section 4.4, namely the one for the linear case with no error (ideal), is not feasible in this new setting.
This happens of course because we cannot trust the measurements as before, because some errors are present. For instance, if we followed the same strategy of change $\theta^*$ as soon as we detect an opposite label from the previous one, if this label has been measured wrongly, then we are going far away from the line, losing track of it.
At the same time, we cannot even use the procedure proposed in Section 5.3.2 in the uniform noise case, as in this new setup the probability of measuring an error is too large (at most $0.5$) when the sampled points are close to the separative curve. Therefore, we have to make some modifications.
The most important things to consider are:

1. Firstly, we need to define a new probabilistic-based strategy to consider the position of the agent enough reliable;
2. Then, we need to redefine an algorithm to move the agent through the two sides, in order to make it achieving the tracking of the line;
3. Lastly, we need to understand if we can converge to the true line in the same or in a different sense with respect to the type of convergence defined in Section 4.4.1.

As for the first point we analyze the impact of the errors on the previous procedure. So, let's consider the probability of performing an error, at iteration $k$ and position $\mathbf{x}(k) = (x(k), z(k))$:

$$\begin{aligned} P(err(k)) &= P\left(|z(k) - mx(k) - q| < \epsilon\right) = \\ &= P\left(z(k) - mx(k) - q < \epsilon \mid z(k) - mx(k) - q > 0\right) + \\ &+ P\left(-\epsilon < z(k) - mx(k) - q \mid z(k) - mx(k) - q < 0\right). \end{aligned} \tag{5.19}$$

From the latter equation is evident that the probability of incur into an error is as small as the distance from the point $\mathbf{x}(k)$ is large.
The probability of performing an error when the point is located at a distance of $3$ standard deviations is about $0.03\%$, and we approximate this to zero, as usually done in statistics.
So the strategy is to consider as reliable just the measurements made outside this range of $3$ standard deviations or, as we will see later on, outside a range large

enough. In fact, often times there are no error way before than being distant $3$ standard deviations from the line.

As for the second point, by moving the agent up and down vertically, namely using as possible $\theta_{target}$ just a value between $\{-\pi/2, +\pi/2\}$, we can claim with confidence that it is on a specific side only when some consecutive samples are labeled with same label, meaning that we are far away from the margin.
This claiming is possible because the probability of measure some consecutive samples with same labels (for instance $5$ consecutive wrong samples), given the fact that we are on a specific side (pointed out by the index $\pm 1$), is very low even in the case we are very close to the margin, as described by the following bound (that is also very relaxing):

$$\begin{aligned} &P_{+1}(e_1, e_2, e_3, e_4, e_5) = \\ &= P_{+1}(e_5|e_4, e_3, e_2, e_1)\, P_{+1}(e_4|e_3, e_2, e_1)\, P_{+1}(e_3|e_2, e_1)\, P_{+1}(e_2|e_1)\, P_{+1}(e_1) \\ &\leq \sum_{i=1}^{5} P_{+1}(e_i) \leq P_{+1}(e_{true})^5 \leq 0.5^5 = 0.03125 \end{aligned} \tag{5.20}$$

where $e_i$ is a random variable indicating the probability of point $P_i$ to measure a wrong error, $+1$ indicates that we are on the specific labeled side where the true label is $+1$, and $e_{true}$ indicates the probability of a point on the true line to be labelled as an error, and it has maximum probability.
The conditional probability of event $e_i$ depends of course on $e_{i-1,i-2,\ldots}$, as the $e_i$ depends on the position of the relative point and this point has been reached also depending on the $e_{i-1,i-2,\ldots}$, and this probability is of course less or equal than the probability that just a single point sampled on the true line is labelled wrongly, that is equal to $0.5$.
However, Eq. 5.20 represents just the error probability of a single set of $5$ consecutive points. But to understand what is the probability of getting $5$ consecutive errors along the trajectory we need to use the absorbing Markov process described in Section 5.3.3, where the variables $p$ and $e$ are not constant anymore, but depend on the position of the state they belong to. Hence, we have $p_0$, $p_1$, $p_2$, $p_3$, $p_4$ and $e_0$, $e_1$, $e_2$, $e_3$, $e_4$. It holds that $p_i = 1 - e_i$.
We can consider the very worst case, namely the case where $p_i = e_i = 0.5$, that coincides to a situation in which the agent is moving in a portion of space sufficiently small around the separative curve, where the probability of measuring

an error can be approximated to $0.5$. In this case, the error probability of measuring $5$ consecutive wrong labels is described by the following curve:

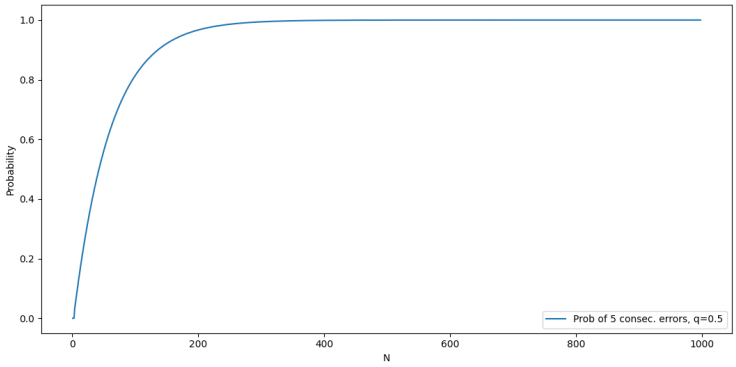


Figure 5.10: Probability of sampling $5$ consecutive wrong labels with gaussian noise in worst case scenario.

It is visible that already with a small number of data points the error probability is not small. However, this is the worst case scenario and, in any case, we could also increase the number of consecutive data points to be sampled to change target direction, as in the gaussian case we do not aim to converge to the true line but only to navigate it.

As for the third point, considering the type of trajectory we want to impose to the agent, with a certain oscillating behaviour around the line depending on the largeness of the variance, on the maximum allowed velocities and on the sampling time, we guess that we are not able to converge in any sense to the line more than the a margin that is large $2$-$3$ standard deviations. Therefore, we can establish an $\epsilon_\sigma$, that accomplishes Eq. 5.18 in the problem 5.4.1, as:

$$\epsilon_\sigma = 3\sigma + N_{cons} * v_{max}. \tag{5.21}$$

The latter value for $\epsilon_\sigma$ is always good because it considers the worst case in which the agent measures a wrong label after $3\sigma$ (after that we consider the error probability equal to $0$), and so it has to move for other $N_{cons}$ (the number of consecutive points as trusting law to change target direction), and it does so, in the worst case, with maximum allowed velocity (in norm sense, so for any direction is valid).

However, it also holds that this oscillating behaviour around the true line can be exploited to have a faster estimate of a non linear curve, as will be discussed in section 5.4.3.

In order to write down an algorithm, we re-define or define the functions:

- $find\_theta\_target()$ (see 8.11, Algorithm 4);
- $update\_flag\_plus()$ (see 8.11, Algorithm 4);
- $labelling()$ (see 8.11, Algorithm 4).

Consequently, Algorithm 4 is as follows:

**Algorithm 4** Line navigation with gaussian errors

**Require:** $X_{max} = 100, \Delta\theta = 1, \Delta v_{max} = 0.1, T = 1, v_{max} = 2$ {Can be set differently}
  $g \leftarrow 0,\ k \leftarrow 0,\ P(k) \leftarrow P_{0_1},\ P_{opposite} \leftarrow P_{0_2},\ \theta(k) \leftarrow None$ {$\theta$ is not defined for k=0}
  $\zeta(k) = (\mathbf{x}_P(k), \theta(k))$
  $y(k) \leftarrow h(\mathbf{x}_P(k))$ {see Eq. 5.16 or Snippet 8.11}
  $flag_+ \leftarrow y(k) == +1$
  **while** $True$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_1()$ {see Eq. 4.25}
    $k \leftarrow k + 1$
    $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
    $update\ y(\mathbf{x}(k))$ {see Eq. 5.16}
    $flag_+ \leftarrow y(k) == +1$
    $flag\_break = True$
    **for** $i = 0;\ i+ = 1;\ i < 5$ **do**
      **if** $y(k-i) \times h(\mathbf{x}_{P_{opposite}}) < 0$ **then**
        $flag\_break = False$ {If this flag remains True we have crossed the margin}
        $break$
    **if** $flag\_break$ **then**
      $break$ {we have crossed the margin so the loop ends}
  $n_1 \leftarrow \mathbf{x}(k-1),\ n_2 \leftarrow \mathbf{x}(k)$
  **while** $x(k) < X_{max}$ **do**
    **if** $m \geq 1$ **then**
      $find\_bounds()$ {See Eq. 8.11}
    $find\_theta\_target()$ {See Eq. 8.11}
    **while** $x(k) < X_{max}$ and $flag_+ == False$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow update\_flag\_plus()$
    **if** $g \geq 1$ **then**
      $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3$
    $n_3 \leftarrow \mathbf{x}(k-1),\ n_4 \leftarrow \mathbf{x}(k)$
    $g \leftarrow g + 1$
    $find\_bounds(),\ find\_theta\_target()$
    **while** $x(k) < X_{max}$ and $flag_+ == True$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
      $update\ y(\mathbf{x}(k))$
      $flag_+ \leftarrow update\_flag\_plus()$
    $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3,$
    $n_3 \leftarrow \mathbf{x}(k),\ n_4 \leftarrow \mathbf{x}(k-1)$
    $g \leftarrow g + 1$

### 5.4.3 Simulations and results

In this section the implementation of the solution proposed in 5.4.2 is presented, where the final estimate of the line has been done by a linear regression approach. In fact, this seems the best solution among other estimation approaches, such as linear classification. Linear classification is not well suited because of the error data points. Indeed, the only data points available to distinguish the two regions are around the separative curve and hence, even few errors in the labels, can lead to a very bad estimation of the line by classification. Instead, the linear regression does not account for the labels but just for the position of the points, and since they are close to the line, the estimation can be good. Below, the trajectory of the agent made with the following parameters:

$$v_{max} = 1\,\mathrm{m\,s^{-1}}\ ,\ T = 0.1\,\mathrm{s}\ ,\ \Delta v = 0.2\,\mathrm{m\,s^{-1}},\ \sigma = 3.5\,\mathrm{m}. \tag{5.22}$$

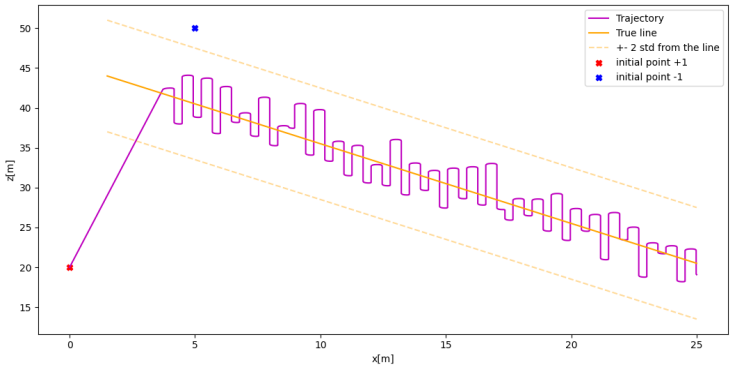


Figure 5.11: Trajectory made by the agent to navigate the line with gaussian noise in the measurements of the labels.

The estimated line has been computed by a linear regression algorithm.

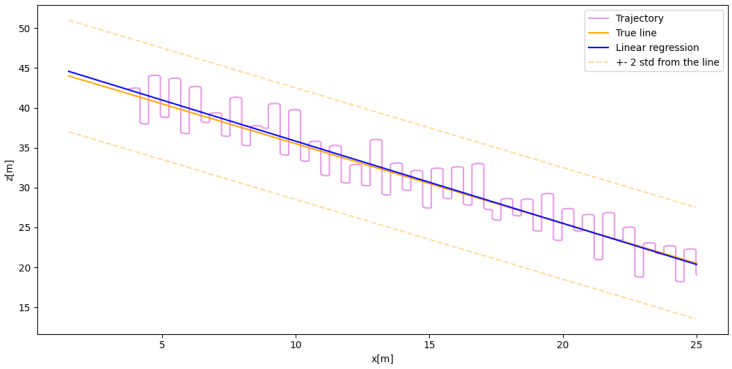


Figure 5.12: Estimation of the line by using a linear regression algorithm on the trajectory's data points.

The final estimate has the following parameters:

$$\hat{\theta} = -45.874^\circ, \quad \hat{q} = 46.126\,\mathrm{m}, \tag{5.23}$$

leading to an error of:

$$err_\theta = 0.874^\circ, \quad err_q = 1.126\,\mathrm{m}. \tag{5.24}$$

We also run $n = 100$ simulations as in Section 4.5, in order to estimate the expected value of the error of the parameters.

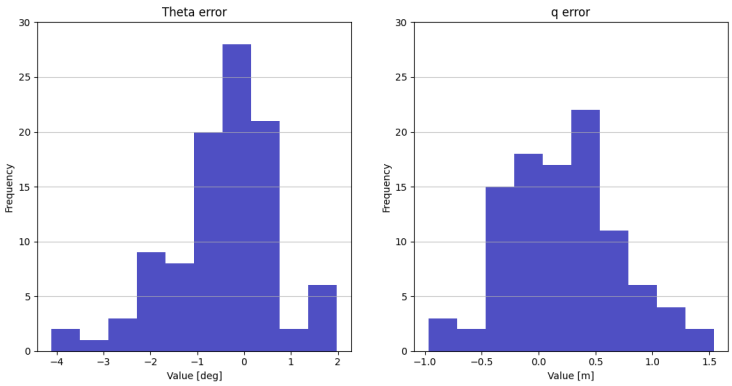


Figure 5.13: Histograms of the error on the estimates of the parameters $\theta$ and $q$, in $n = 100$ different linear regression estimations.

The results show how the estimates are quite good, with errors around the $0$

values, but with some outliers.
However, the quality of the estimates can be improved by enlarging the trajectory of the agent through a longer horizon, since the uncertainty on the slope and on the intercept decrease as long as the agent collect more data points around the true line.

Overall, the estimations of the separative line in the gaussian noise case are good. However, they required the implementation of a linear regression algorithm, whose complexity is normally $O(nd^2 + d^3)$, hence increasing the total computational time with respect the uniform noise case or the ideal noiseless one.

### 5.4.4 A different solution

The solution presented so far for the gaussian noisy model is good but requires an additional part for the estimation that involves all the $N$ data points collected by the trajectory, some of them not so useful as quite far away from the separative line.
We can slightly modify the approach we have followed in Section 5.4.2, by making the following observations:

- the closer the agent navigates to the separative line, the higher the number of errors;
- if the agent computes a symmetric linear piece-wise trajectory with respect to the separative line, recalling that the gaussian noise is also symmetric, then the expected value of the measurements (that can be only $\pm 1$) in that segment of trajectory is equal to zero;
- if instead the agent computes an asymmetric linear piece-wise trajectory, then the expected value of the measurements is different from zero, and depends in particular on the extra data points belonging to the segment that make it asymmetric.

Considering the discrete random variables given by the measurements of the labels in a piece-wise segment of the trajectory that passes through the separative line, say $\{y_i\}$, where $i \in [1, I]$, and recalling the fact that every single measurement is a function of a random variable so that:

$$y_i = g(X_i), \quad X_i \sim \mathcal{N}\left(z_i - mx_i - q, \sigma^2\right), \quad g(\cdot) = sgn(\cdot), \tag{5.25}$$

the expected value of the sum of the $I$ outcomes is (see [1]):

$$\mathbb{E}\left[\sum_{i=1}^{I} y_i\right] = \sum_{i=1}^{I} \mathbb{E}\left[y_i\right], \tag{5.26}$$

assuming the data points independent. But then we know that, in general, the piece-wise segment is not symmetric with respect to the separative line, so:

$$\mathbb{E}\left[\sum_{i=1}^{I} y_i\right] = c. \tag{5.27}$$

Since the segment passes through the separative line, there is a symmetric part, made of $2\bar{I}$ data points such that:

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{2\bar{I}} \bar{y}_i\right] = &\left[\sum_{i=1}^{\bar{I}}(+1)P(\bar{X}_i > 0) + \sum_{i=1}^{\bar{I}}(-1)P(\bar{X}_i < 0)\right] + \\ &\left[\sum_{i=\bar{I}}^{2\bar{I}}(-1)P(\bar{X}_i < 0) + \sum_{i=\bar{I}}^{2\bar{I}}(+1)P(\bar{X}_i > 0)\right] = 0,\end{aligned} \tag{5.28}$$

where $P(X_i > 0)$ is the probability that the continuous gaussian random variable $X_i$ (see 8.53) is bigger than zero and it can be obtained by the cumulative distributive function, that can be obtained by integrating the probability density function. The last equality holds because (see [9]), in general for independent random variables, $Z = \sum_{i=1}^{I} X_i$, is a guassian random variable such that:

$$Z \sim \left(\sum_{i=1}^{I} \mu_i, \sum_{i=1}^{I} \sigma_i^2\right), \tag{5.29}$$

thus the two parts cancel out each other as the two parts are symmetric with respect the separative line. Hence, overall:

$$\mathbb{E}\left[\sum_{i=1}^{I} y_i\right] = \sum_{i=1}^{I} \mathbb{E}\left[y_i\right] = \sum_{i=1}^{2\bar{I}} \mathbb{E}\left[\bar{y}_i\right] + \sum_{i=1}^{\tilde{I}} \mathbb{E}\left[\tilde{y}_i\right] = c. \tag{5.30}$$

If we further notice that the points belonging to the asymmetric part, namely $\{\tilde{y}_i\}_{\{i=1,\dots,\tilde{I}\}}$, are far away from the separative line (as close by the line there is the symmetric part) and lay on the same side (on the side where there segment

has more points), then it holds that:

$$c = \pm \tilde{I}, \tag{5.31}$$

where + is for the cases where the asymmetric part is on the side with labels $+1$ and $-$ for the other case. The latter equation holds because the errors in the measurements are not present when the observations are taken far away from the line, where the variance of the gaussian noise has not effect anymore and so, recalling that expected value of a function $g(X)$ of a discrete random variable $X$ with probability mass function $f_X(X)$ is $\mathbb{E}[g(X)] = \sum_{x \in X} g(x) f_X(x)$ and $sign() \in \{\pm 1\}$, then it holds that:

$$\sum_{i=1}^{\tilde{I}} \mathbb{E}\left[\tilde{y}_i\right] = \sum_{i=1}^{\tilde{I}} +1\, P\left(\tilde{X}_i \geq 0\right) + (-1)\, P\left(\tilde{X}_i < 0\right) = \pm \tilde{I}, \tag{5.32}$$

as either $P\left(\tilde{X}_i \geq 0\right) = 1$ and $P\left(\tilde{X}_i < 0\right) = 0$ for any $i = 1, \ldots, \tilde{I}$, or $P\left(\tilde{X}_i \geq 0\right) = 0$ and $P\left(\tilde{X}_i < 0\right) = 1$ for any $i = 1, \ldots, \tilde{I}$, since the asymmetric part lays on just one side (so they have same sign) and far away from the line (so all their value is unitary as the uncertainty of the gaussian noise is not present in such locations). All of this, allows us to claim that a variation of $\pm \tilde{I}$ (with the correct sign) in our measurements, implies that:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{I} y_i\right] = c \\
&\Rightarrow \mathbb{E}\left[\sum_{i=1}^{I} y_i\right] - c = 0 \\
&\Rightarrow \mathbb{E}\left[\sum_{i=1}^{I} y_i\right] - \mathbb{E}[c] = 0 \\
&\Rightarrow \mathbb{E}\left[\sum_{i=1}^{I} y_i - c\right] = 0 \\
&\Rightarrow \mathbb{E}\left[\sum_{i=1}^{I-\tilde{I}} y_i\right] = 0,
\end{aligned} \tag{5.33}$$

where, in the last implication, the set $I - \tilde{I}$ is the initial set minus the observations that are not symmetric.

This suggests to choose a point $P$, as an estimate of a point on the separative line, selected as:

$$P = (x_P, z_P) = \begin{cases} \left( \frac{x_{\tilde{I}\pm\tilde{I}} + x_{\tilde{I}+\tilde{I}+1}}{2}, \frac{z_{\tilde{I}\pm\tilde{I}} + z_{\tilde{I}+\tilde{I}+1}}{2} \right), & \text{if } I \text{ is even} \\ \left( x_{\tilde{I}\pm\tilde{I}+1}, z_{\tilde{I}\pm\tilde{I}+1} \right), & \text{if } I \text{ is odd} \end{cases}, \tag{5.34}$$

where $\pm I$ must be chosen according to the side where the agent is and to the sign of $c$.

Hence the point $P$ is a gaussian random variable with mean $0$ (as is a good estimate of a point belonging to the separative line) and variance $\sigma^2$. By picking up other points with the same distribution is then possible to perform a linear regression estimation of the true separative line, by just using these points and not all the points sampled by the agent during its trajectory, thus reducing the computational cost.

This permits us to write down a new algorithm to estimate a separative line in the case of gaussian noise in the measurements of the labels.

To find a piece-wise linear segment of the trajectory such that the asymmetric part has constant probability equal to $1$ (i.e. its points are far enough from the line) we can take $9$ consecutive data points sampled with same label to discriminate between one the two sides, hence changing target direction (alternating between $\{-\frac{\pi}{2}, \frac{\pi}{2}\}$) only when the agent is far away from the line. Therefore, if the agent computes $G$ up and down periods, the number of piece-wise segment collected is $G + 1$.

Thus the Algorithm 5 uses only $G + 1$ points to make the final estimate.

We define a function $find\_point()$ (see 8.11, Algorithm 5) that finds out the index in which to take the point $P$.

Here the Algorithm 5 as follows:

**Algorithm 5** Line navigation for estimation with G+1 data points

**Require:** $X_{max} = 100, \Delta\theta = 1, \Delta v_{max} = 0.1, T = 1, v_{max} = 2$ {Can be set differently}
$g \leftarrow 0,\ k \leftarrow 0,\ P(k) \leftarrow P_{0_1},\ P_{opposite} \leftarrow P_{0_2},\ \theta(k) \leftarrow None$ {$\theta$ is not defined for k=0}
$\zeta(k) \leftarrow (\mathbf{x}_P(k), \theta(k)),\ update\ y(\mathbf{x}(k))$ {see Eq. 5.16 or Snippet 8.11}
$doubt\_points \leftarrow [\,],\ c \leftarrow 0,\ sure\_flag \leftarrow True,\ final\_points \leftarrow [\,],\ flag_+ \leftarrow y(k) == +1,\ flag\_break \leftarrow False$
**while** $flag\_break == False$ **do**
  $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_1()$ {see Eq. 4.25}
  $k \leftarrow k + 1$
  $\zeta(k) \leftarrow A\zeta(k-1) + B\mathbf{u}(k-1),\ update\ y(\mathbf{x}(k)),\ flag_+ \leftarrow y(k) == +1$
  **if** $k \geq 8$ **then**
    $flag_+ \leftarrow update\_flag\_plus()$ {see 8.11}
    **if** $sure\_flag == True\ and\ y[-1] == -1$ **then**
      $sure\_flag \leftarrow False$
    **if** $all\ the\ 9\ last\ consecutive\ points\ have\ label\ -1$ **then**
      $flag\_break \leftarrow True,\ sure\_flag \leftarrow True$
  **if** $sure\_flag == False\ and\ break\_flag == False$ **then**
    $doubt\_points.append(\zeta(k)),\ c \leftarrow c + y(-1)$
$n_1 \leftarrow \mathbf{x}(k-1),\ n_2 \leftarrow \mathbf{x}(k),\ final\_points.append(find\_point())$ {See 8.11}
**while** $x(k) < X_{max}$ **do**
  **if** $m \geq 1$ **then**
    $find\_bounds()$ {See Eq. 8.11}
  $doubt\_points \leftarrow [\,],\ c \leftarrow 0,\ find\_theta\_target()$ {See Eq. 8.11}
  **while** $x(k) < X_{max}$ and $flag_+ == False$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
    $k \leftarrow k + 1$
    $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1),\ update\ y(\mathbf{x}(k)),\ flag_+ \leftarrow update\_flag\_plus()$
    **if** $sure\_flag == True\ and\ y[-1] == +1$ **then**
      $sure\_flag \leftarrow False$
    **if** $all\ the\ 9\ last\ consecutive\ points\ have\ label\ +1$ **then**
      $sure\_flag \leftarrow True$
    **if** $sure\_flag == False$ **then**
      $doubt\_points.append(\zeta(k)),\ c \leftarrow c + y(-1)$
  **if** $g \geq 1$ **then**
    $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3$
  $n_3 \leftarrow \mathbf{x}(k-1),\ n_4 \leftarrow \mathbf{x}(k)$
  $g \leftarrow g + 1$
  $find\_bounds(),\ find\_theta\_target(),\ final\_points.append(find\_point()),\ doubt\_points \leftarrow [\,],\ c \leftarrow 0$
  **while** $x(k) < X_{max}$ and $flag_+ == True$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
    $k \leftarrow k + 1$
    $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1),\ update\ y(\mathbf{x}(k)),\ flag_+ \leftarrow update\_flag\_plus()$
    **if** $sure\_flag == True\ and\ y[-1] == -1$ **then**
      $sure\_flag \leftarrow False$
    **if** $all\ the\ 9\ last\ consecutive\ points\ have\ label\ -1$ **then**
      $sure\_flag \leftarrow True$
    **if** $sure\_flag == False$ **then**
      $doubt\_points.append(\zeta(k)),\ c \leftarrow c + y(-1)$
  $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3,$
  $n_3 \leftarrow \mathbf{x}(k),\ n_4 \leftarrow \mathbf{x}(k-1)$
  $g \leftarrow g + 1$
  $final\_points.append(find\_point())$ {See 8.11}

### 5.4.5 Simulations and results

We run 100 simulations and we compare the errors on the estimates of the parameters $m$ and $q$ between the Algorithm 4 and Algorithm 5. However, to make a fairer comparison, also in Algorithm 4 we use 9 consecutive points as trusting law before changing target direction.

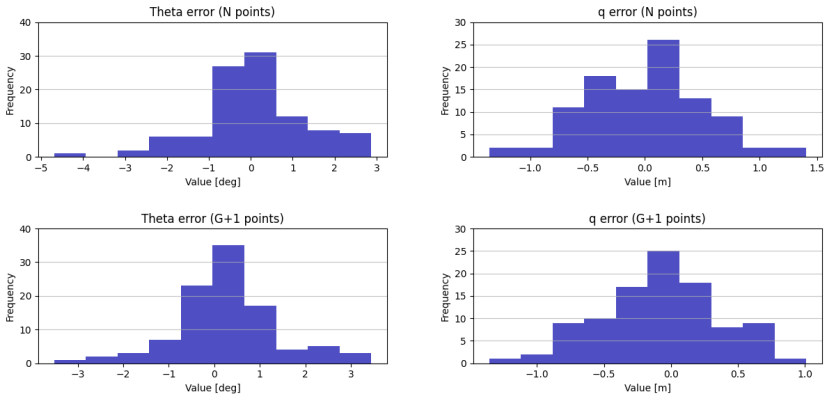


Figure 5.14: Histograms of the parameters' errors between Algorithm 4 and Algorithm 5.

Notice how the estimates with the second algorithm are distinctly better than the first one, both for the $\theta$ error and for the $q$ error.

The errors of the averages, computed as in Eq. 4.40 and in Eq. 4.41, are presented in the following table:

| ALGORITHM 4 | | | |
|---|---|---|---|
| $avg_{err_\theta}$ [°] | $avg_{err_{rel_\theta}}$ [°] | $avg_{err_q}$ [m] | $avg_{err_{rel_q}}$ [m] |
| 0.747 | 0.074 | 0.365 | 0.008 |

| ALGORITHM 5 | | | |
|---|---|---|---|
| $avg_{err_\theta}$ [°] | $avg_{err_{rel_\theta}}$ [°] | $avg_{err_q}$ [m] | $avg_{err_{rel_q}}$ [m] |
| 0.497 | 0.044 | 0.225 | 0.005 |

Table 5.2: Averages of the final errors of the Algorithm 4 and Algorithm 5.

The reason for the better performance of Algorithm 5 can be found on the fact that, while some oscillations of the trajectory can be slightly discorded from the

separative margin, leading to some points that worsen the trajectory, Algorithm 5 tries to overcome this issue estimating a single data point that is, in expected value sense, belonging to the separative line, hence yielding to a better estimation.

We summarize the results obtained in the last two chapters, Chapter 4 and Chapter 5, with the following table: The ideal noiseless case is the one that gives the

| Noiseless data | | | |
|---|---|---|---|
| **Convergence** | **Navigation** | **Estimation** | **Estimation's complexity** |
| ✓ | ✓ | ✓ | $O(1)$ |

| Uniform noisy label data | | | |
|---|---|---|---|
| **Convergence** | **Navigation** | **Estimation** | **Estimation's complexity** |
| ✓ $in$ $\mathrm{P}(N,e,n)$ | ✓ $in$ $\mathrm{P}(N,e,n)$ | ✓ $in$ $\mathrm{P}(N,e,n)$ | $If\ success$, $O(1)$ |

| Gaussian noisy label data | | | |
|---|---|---|---|
| **Convergence** | **Navigation** | **Estimation** | **Estimation's complexity** |
| ✗ | ✓ | ✓ | $O(N)$ $or$ $O(G+1)$ |

Table 5.3: Chapters 4 and 5's results

better results, but does not consider any error model. As far as the uniform noise model, the success of the relative developed procedure is probabilistic, however we have seen in Figure 5.3 how the success/failure probability changes varying the parameters $N$, $e$ and $n$. If there is success, namely there have not been $n$ consecutive errors in the transient part, than the estimation is "for free", as it is given as in the noiseless case by the convergence. As far as the gaussian noise case, we cannot make the agent converge to the line as there are too many errors around the margin, but we can make it move around it following the line, hence reaching the navigation task and eventually also the estimation one, paying the price of using again the $N$ points (that is the total number of points sampled by the agent, as in Algorithm 4), or using just the $G+1$ points (as in Algorithm 5) saving some computational effort.
Notice that for estimation's complexity we mean the complexity needed to estimate the separative line once the agent has ended its trajectory and has already

collected the data points. So the estimation part only regards the manipulation of these data to obtain the final estimate.

# 6

# Identification of special nonlinear classifiers with noiseless and noisy data

## 6.1 Intro

Extending the problem to the identification and navigation of non linear separative curve is a natural extension of the previous linear problem, as in practice most of the curve are in fact non linear. In this chapter, we analyze such an environment, while keeping fixed all the other assumptions.

## 6.2 Non linear ideal case

The main issue of the non linear case is the fact that we do not have an equation $z = mx + q$, but we have a more complex equation whose derivative on space is not constant anymore, but is a time-varying function as well.
Either we can have a non linear function:

$$z = h(x), \tag{6.1}$$

or we can have a parametric curve:

$$(x(t), z(t)) = (h_1(t), h_2(t)) . \tag{6.2}$$

So, we need a tool to capture the variation of the function (or of the param. curve), that coincides with the derivative (or with the gradient), to make the agent understand where it is moving and follow it.
In the work [4], a tracking of a non linear curve, whose shape separates two different areas is made, but there an important assumption on the gradient of the concentration of the salinity (that determines the discriminant law in the labelling) is made, hence having more knowledge about the variation of the curve. Instead, in this work, we do not change the assumptions on the information the agent is provided.
Based on the work of [5], we can think the separative curve as a set of finite piece-wise constant linear segments, where each segment has its own constant derivative value, and so we can use a similar algorithm to Algorithm 2 to make the agent navigating the line, by updating the estimate of the variation of the curve in an online fashion. Whereas in [5] three points of a curve are used to estimate the curvature of the curve in one point, we want to use two estimated points of the separative curve, retrieved by the trajectory of the agent, to estimate the slope of the segment between those two points, in order to obtain a measure of the variation of the curve.
To do so, we require and we assume the curve has a variation that is not too abrupt with respect to the capability in terms of speed of convergence of the agent to track the segment, and of course this depends on the velocities that the agent can achieve, on the maximum variation between two consecutive velocities and on the sampling time.
In this new scenario, we cannot make the state coordinate $\theta$ to converge to something, as this would primarily imply that we would enlarge too much the length of the periods $m$, loosing the capability of following the variations of the curve, and also $\theta$ must follow somehow, the current variation of the curve ans so it cannot be shrunk to a fixed constant value.
Therefore, the strategy we adopt is to make the agent to trust just the previous estimate of the range of $[\theta_{min}, \theta_{max}]$, given by $n_1$, $n_2$, $n_3$ and $n_4$, without considering the previous estimates as the curve keeps changing.
We know then that the true angular coefficient $m$ is such that $m \in [m_{min}, m_{max}]$, where:

$$m_{min} = \frac{z_{n_2} - z_{n_4}}{x_{n_2} - x_{n_4}} \text{ and } m_{max} = \frac{z_{n_1} - z_{n_3}}{x_{n_1} - x_{n_3}}, \tag{6.3}$$

or in terms of the true theta, $\theta$, such that $\theta \in [\theta_{min}, \theta_{max}]$, where:

$$\theta_{min} = \arctan(m_{min}) \text{ and } \theta_{max} = \arctan(m_{max}). \tag{6.4}$$

Then, we update $\theta^*$ by enlarging this interval of $2\Delta\theta$, so that:

$$\theta^*(g) = \begin{cases} \theta_{max}(g) + \Delta\theta, & \text{if } g > 0 \, is \, odd \\ \theta_{min}(g) - \Delta\theta, & \text{if } g > 0 \, is \, even \end{cases} \tag{6.5}$$

**Note**: we stress the fact that here we are enlarging the interval where the actual $\theta$ can be, allowing a variation of the curve that can be, in the worst case, equal to $\Delta\theta$.
Instead, in the linear case we were able to shrink the interval close to the true actual direction of the line.

Therefore, with these modifications on the definition of $\theta^*$ we can still use Algorithm 2.

### 6.2.1 Case of non linear curves that are not functions

We also consider the case where the curve that we want our agent to follow is not a function, e.g. an ellipse.
This case is more involved because the agent must move also backward, so the range of its possible $\theta$ directions must include all the degrees in $[0, 2\pi]$.
Thus, we redefine the third row of Eq. 4.4, as:

$$\theta(k+1) = \begin{cases} \arctan\left(\frac{v_2(k)}{v_1(k)}\right), & \text{if } v_1 \geq 0, \ k = 1, 2, \ldots, \\ \arctan\left(\frac{v_2(k)}{v_1(k)}\right) + \pi, & \text{if } v_1 < 0, \ k = 1, 2, \ldots, \end{cases} \tag{6.6}$$

and we redefine the function $find\_theta\_target()$ in 8.11 as in 8.11.
Let's suppose we know our curve is an ellipse and this has a certain equation, parameterized by $t$, as follows:

$$\left(2\cos\left(\pi t_i\right), \sin\left(\pi t_i\right)\right), t_i \in [0, 1]. \tag{6.7}$$

Suppose the agent will track such a curve going around its points and getting $G$ total crossings of the curve, where this number depends on the type of curve,

on the dynamics of the agent and on its inputs.
To be sure to be able to navigate the curve we want that two consecutive segments of the curve, taken among the three consecutive points belonging both to the separative curve and to the trajectory, do not change to much in slope. Hence, let's consider the points $\boldsymbol{x}_{g-1}$, $\boldsymbol{x}_g$ and $\boldsymbol{x}_{g+1}$ estimated as
$\hat{\boldsymbol{x}}_{g-1} = avg\,(n_1(g-1), n_2(g-1))$, $\hat{\boldsymbol{x}}_g = avg\,(n_3(g-1), n_4(g-1))$, and $\hat{\boldsymbol{x}}_{g+1} = avg\,(n_3(g), n_4(g))$.
We want to understand whether, given the fact that we are in position $\boldsymbol{x}_i$, we are able to reach the point $\boldsymbol{x}_{i+1}$, provided that we can turn at most of $\Delta\theta$ in the worst case, as in such a case we have approximated the two bounds $m_{max}$ and $m_{min}$ as a unique one given by $\frac{z_{\hat{x}_{g-1}} - z_{\hat{x}_g}}{x_{\hat{x}_{g-1}} - x_{\hat{x}_g}}$.
Moreover, since:

$$\hat{\boldsymbol{x}}_{g+1} = (x_{g+1}, z_{g+1}) = \left( x_g + \sum_{j=j_g}^{j_{g+1}} v_{1_j},\ z_g + \sum_{j=j_g}^{j_{g+1}} v_{2_j} \right), \tag{6.8}$$

we have to ensure that:

$$\left|\theta_g - \theta_{g+1}\right| \leq \Delta\theta, \tag{6.9}$$

namely:

$$\left| \arctan \frac{z_{g+1} - z_g}{x_{g+1} - x_g} - \arctan \frac{z_g - z_{g-1}}{x_g - x_{g-1}} \right| \leq \Delta\theta. \tag{6.10}$$

Assuming that we already are at the point $g$,

$$c - \Delta\theta \leq \arctan \frac{\sum_{j=j_g}^{j_{g+1}} v_{2_j}}{\sum_{j=j_g}^{j_{g+1}} v_{1_j}} \leq c + \Delta\theta, \tag{6.11}$$

where $c = \arctan \frac{z_g - z_{g-1}}{x_g - x_{g-1}}$ and is already given.
Thus, we can see that the velocities fed to the agent in the period from $g$ to $g+1$ must respect a specific bound.
We stop here this analysis because further investigation is quite difficult, but we can say that, as general rule, the slower the agent moves, the better is able to track a curve, as the curvature between two consecutive points is smaller as they are closer by.

### 6.2.2 Simulations and results

We simulate Algorithm 2 with the modifications of 8.11 and 6.6 for the tracking and identification of a sinusoidal separative function, described by the following equation:

$$z(x) = -5cos(0.2x) + 10. \tag{6.12}$$

We use as initial points $P_{0_1} = (2, 3)$ and $P_{0_2} = (5, 15)$, a sampling time of $T = 0.05\,\text{sec}$, $\Delta\theta = 20°$, $v_{max} = 1\,\text{m/sec}$ and $\Delta v_{max} = 0.1\,\text{m/sec}$.

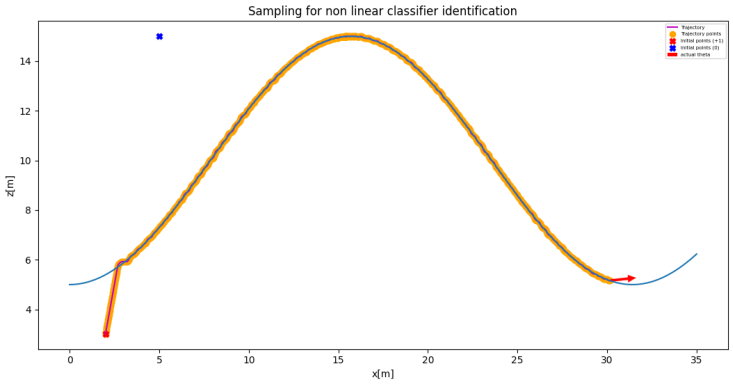


Figure 6.1: Trajectory performed for the tracking of a sinusoidal function without errors in the label measurements.

A zoomed section of the trajectory, showing its oscillating behaviour around the separative curve is provided next:

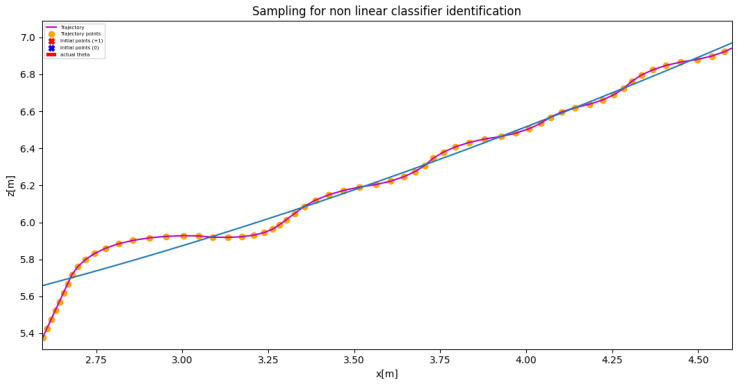


Figure 6.2: Zoomed on the trajectory performed for the tracking of a sinusoidal function without errors in the label measurements.

Finally, we also simulate the tracking and identification of an ellipse, that is not

a function, and hence the re-formulation of Algorithm... are needed. The ellipse is given by the following equation:

$$\frac{(x - x_{center})^2}{a^2} + \frac{(z - z_{center})^2}{b^2} = 1, \tag{6.13}$$

where we choose $x_{center} = 10\,\mathrm{m}$, $z_{center} = 25\,\mathrm{m}$, $a = 25\,\mathrm{m}$ and $b = 20\,\mathrm{m}$. The other parameters are: initial points $P_{0_1} = (2, 2)$ and $P_{0_2} = (5, 15)$, sampling time of $T = 0.2\,\mathrm{sec}$, $\Delta\theta = 20°$, $v_{max} = 1\,\mathrm{m/sec}$ and $\Delta v_{max} = 0.1\,\mathrm{m/sec}$.

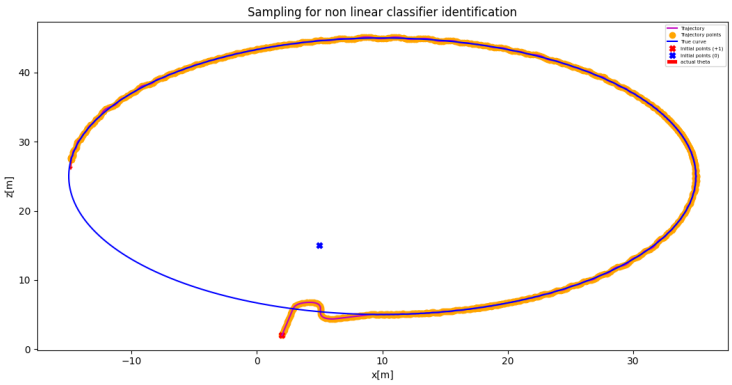


Figure 6.3: Trajectory performed for the tracking of an ellipse curve without errors in the label measurements.

We see how the tracking has success and the trajectory is very close to the true curve.

### 6.2.3 Non linear case with gaussian error noise

We consider a non linear function $z = h(x)$ such that, if we move the agent with vertical target directions, namely using as $\theta^*$ the set $\{-\pi/2, +\pi/2\}$, we are sure of finding the line.
To be confident that we cross the line, we use as criterion the sampling of a number $n$ of consecutive data points with the same label.
When we track a large enough part of the curve, we stop the agent and we start the estimation part.

The following algorithm, Algorithm 6, shows how to move the agent in this setting:

**Algorithm 6** Curve navigation with gaussian errors

**Require:** $X_{max} = 100, \Delta\theta = 1, \Delta v_{max} = 0.1, T = 1, v_{max} = 2$ {Can be set differently}
  $g \leftarrow 0,\ k \leftarrow 0,\ P(k) \leftarrow P_{0_1},\ P_{opposite} \leftarrow P_{0_2},\ \theta(k) \leftarrow None$ {$\theta$ is not defined for k=0}
  $\zeta(k) = (P(k), \theta(k))$
  $y(k) \leftarrow \text{sign}\,(h(\mathbf{x}(k)))$ {see Eq. 5.16 or Snippet 8.11}
  $flag_+ \leftarrow y(k) == +1$
  **while** $True$ **do**
    $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_1()$ {see Eq. 4.25}
    $k \leftarrow k + 1$
    $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
    $y(k) \leftarrow \text{sign}\,(h(\mathbf{x}(k)))$
    $flag_+ \leftarrow y(k) == +1$
    $flag\_break \leftarrow True$
    **for** $i = 0;\ i+ = 1;\ i < 5$ **do**
      **if** $y(k-i) \times labelling(P_{opposite}) < 0$ **then**
        $flag\_break = False$ {If this flag remains True we have crossed the margin}
        $break$
    **if** $flag\_break$ **then**
      $break$ {we have crossed the margin so the loop ends}
  $n_1 \leftarrow \mathbf{x}(k-1),\ n_2 \leftarrow \mathbf{x}(k)$
  **while** $x(k) < X_{max}$ **do**
    **if** $m \geq 1$ **then**
      $find\_bounds()$ {See Eq. 8.11}
    $find\_theta\_target()$ {See Eq. 8.11}
    **while** $x(k) < X_{max}$ and $flag_+ == False$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) = A\zeta(k-1) + B\mathbf{u}(k-1)$
      $y(k) \leftarrow \text{sign}\,(h(\mathbf{x}(k)))$
      $flag_+ \leftarrow update\_flag\_plus()$
    **if** $g \geq 1$ **then**
      $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3$
    $n_3 \leftarrow \mathbf{x}(k-1),\ n_4 \leftarrow \mathbf{x}(k)$
    $g \leftarrow g + 1$
    $find\_bounds(),\ find\_theta\_target()$
    **while** $x(k) < X_{max}$ and $flag_+ == True$ **do**
      $\mathbf{u}(k) \leftarrow solve\_opt\_prob\_2()$ {See Eq. 4.26}
      $k \leftarrow k + 1$
      $\zeta(k) \leftarrow A\zeta(k-1) + B\mathbf{u}(k-1)$
      $y(k) \leftarrow \text{sign}\,(h(\mathbf{x}(k)))$
      $flag_+ \leftarrow update\_flag\_plus()$
    $n_1 \leftarrow n_4,\ n_2 \leftarrow n_3,$
    $n_3 \leftarrow \mathbf{x}(k),\ n_4 \leftarrow \mathbf{x}(k-1)$
    $g \leftarrow g + 1$

### 6.2.4 Estimation

At this stage, we have navigated the line and we have collected quite a lot of points, part of them with mislabelled measurements.
We present and compare in this subsection different approaches that can be used to estimate the margin separative line.
The considered methods are:

- Non linear SVM regression (see 8.4.3);
- Non linear SVM classification (see 8.3.5);
- Moving average (see 8.7);
- Butterworth filter (see 8.8).

The first method is a classical estimation algorithm from machine learning field. It has a lot of good properties, but also some drawbacks, such as the high computational time that is $O(n^2 d)$ for the training time, the necessity of accurately tuning the hyperparameters of the kernel and of the regularization, that is normally made by cross-validation. Other ML methods could be used but they also suffer of the the problematic computational cost.
Therefore, we adopt two new strategies, very common in the signal processing field, that are both filters, and in particular low-pass filters.
The first one is a Moving average filtering and the second is a Butterworth analogic low-pass filter.
The reason to use a filter is given by the trajectory of the agent around the true curve. Indeed, the continuous "periodic" oscillations around the true curve made us think of dealing with a sort of disturbed signal, in which the noise has higher frequency. From this observation, is easy to look for something like a filter.
Moreover other two advantages made us to adopt, as we anticipate here, these two tools as the best one.
The first advantage is the facility in tuning the filter. In fact, given the trajectory we easily see how the periods are made and so we can just count the number of points belonging to one period and then multiply for some more periods using a trial and error approach. This gives us an easy tool to tune the window size of the Moving average filter, and the cut-off frequency of the Butterworth filter.
The second advantage is about the computational cost of the filtering procedure: both the filter are linear and have complexity, respectively, $O(n + w)$ and $O(nk)$,

where $w$ is the window size of the Moving average filter and $k$ is the order of the Butterworth filter.

### 6.2.5 Simulations and results

We now simulate Algorithm 6 to track a sinusoidal function in presence of gaussian error noise, as in Eq. 5.16, in the label's measurements.
The function has equation:

$$z(x) = -10cos(0.6x) + 20, \tag{6.14}$$

the initial points are $P_{0_1} = (0, 5)$ and $p_{0_2} = (5, 45)$.
Other parameters are:

$$T = 0.05\,\text{sec},\ v_{max} = 1\,\text{m/sec},\ \Delta v = 0.2\,\text{m/sec},\ \sigma = 3.5\,\text{m}. \tag{6.15}$$

**Note**: the choice of the sampling time is very important and determines both the total number of points, that are useful for the estimation part, the computational time, the distance from the line in which the agent detects a changing on the side where it is (namely $5$ consecutive label points in Algorithm 4). We also plot the true curve with an offset of $\pm 2$ standard deviations to visualize how far the agent moves from the line.

After the agent ends to move and has collected the data points, we perform a smoothing of the trajectory using the moving average approach, described in Section 8.7, with a window size of $\left[8\frac{count_1}{G-2}\right]$, where the square brackets represent here the $round$ operator, and $count_1$ is the number of data points from the point where the agent crosses the line for the first time until the end. With this choice, we use in average, all the points between $8g$, namely all the points among $8$ side variations.
This choice has been made by trial and error and seems one of the best as it is a good trade-off between smoothing the noisy trajectory and avoiding to incorporate too many periods, that would imply obtaining a curve with a smaller scale factor. Here it is the results of the application of this filter to the noisy signal, both with respect the $x$ coordinate and with respect the $z$ coordinate:

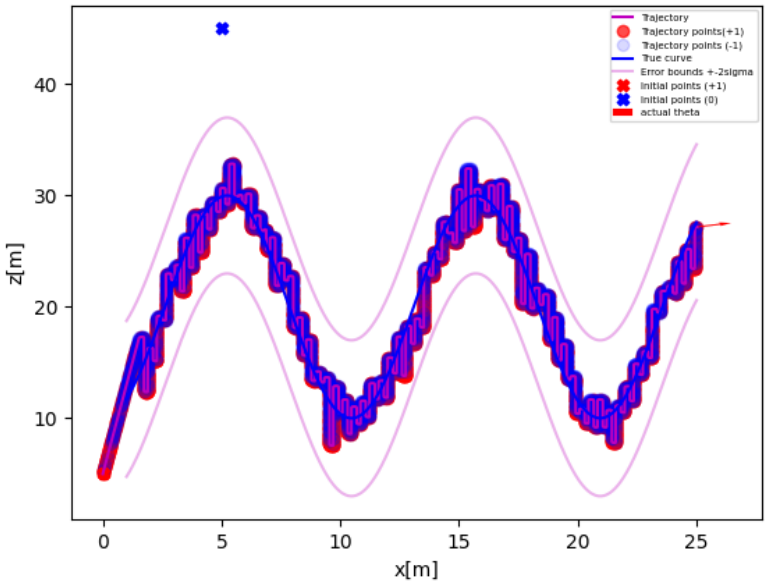


Figure 6.4: Trajectory performed for the tracking of a sinusoidal function with gaussian error in the label measurements.

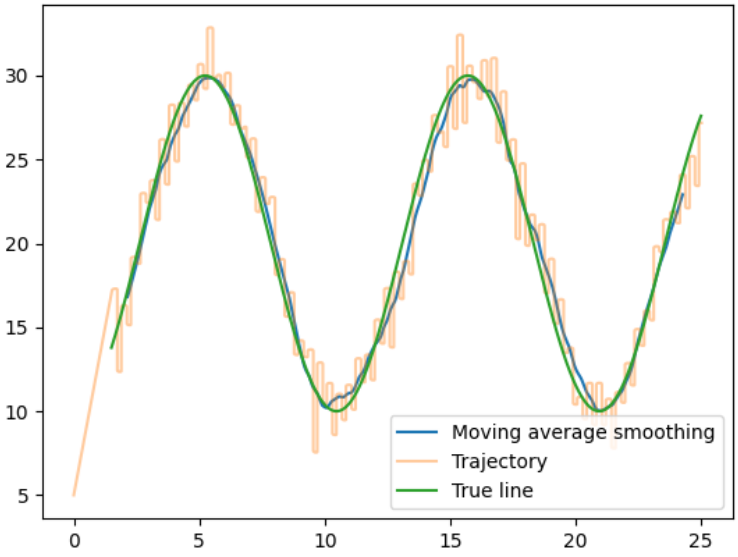


Figure 6.5: Moving average behaviour.

Then, we also use the Butterworth low-pass filter. To design it, we need to choose the order of the filter and the normal cut-off frequency.

We select an order of $n = 2$, as it is enough to filter the sinusoidal curves and most of the not too complex curves.
The normalized cut-off frequency is defined as follows:

$$f_{c_{norm}} = \frac{f_c}{f_n}, \tag{6.16}$$

where $f_c$ is the cut-off frequency and $f_n$ is the Nyquist frequency, that is equal to:

$$f_n = \frac{f_s}{2}, \tag{6.17}$$

where $f_s$ is the sample rate.
Thus we can keep fixed the cut-off frequency, let's say equal to $1$, and varying the sampling rate. After some trials, it comes out that one of the best choices is, as for the Moving average filter, to set the sampling rate as $f_s = \left[8\frac{count_1}{G-2}\right]$, in such a way that the Nyquist frequency contains about $4g$, that is considered enough to get the necessary information to smooth the signal.

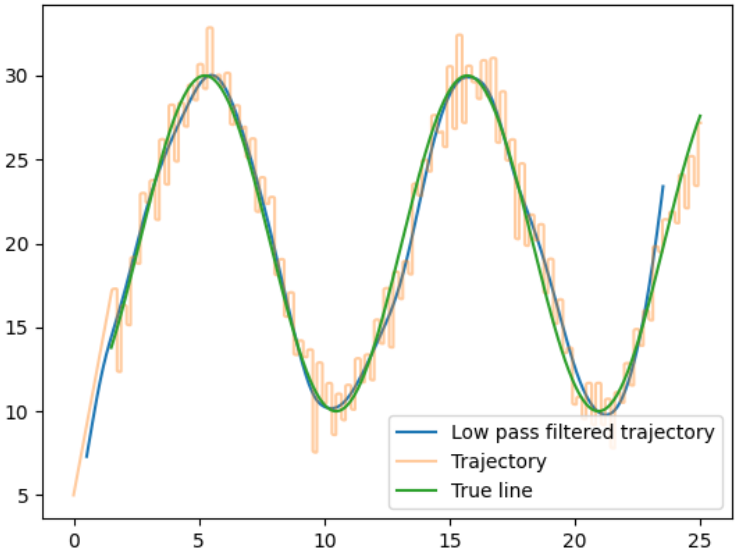


Figure 6.6: Butterworth low-pass filtering.

We notice how the Butterworth filter really nicely smooths the signal.

In the following image we compare different choiches of hyperparameters for

the non linear SVM regression algorithm, where we used just the "rbf" kernel, namely the gaussian one, as in general the polynomial kernel does not offer meaningful improvements.
We tested the algorithm using the Python library scikit-learn.SVM.
We tested the algorithm with different types of hyperparameters, both for $\gamma$ and for $C$, namely the kernel width and the regularization term. The kernel function has equation:

$$K(\mathbf{x}, \mathbf{x'}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x'}\|^2\right), \tag{6.18}$$

therefore, the smaller $\gamma$, the smoother the estimated curve is.
We use:

$$\gamma = 0.1,\ \gamma = 1,\ \gamma = 10,\ \gamma = 100, \tag{6.19}$$

as kernel width possible parameters, whereas:

$$C = 0.1,\ C = 1,\ C = 100,\ C = 1000, \tag{6.20}$$

as regularization term, where the smaller the term $C$ is, the higher the regularization strength.

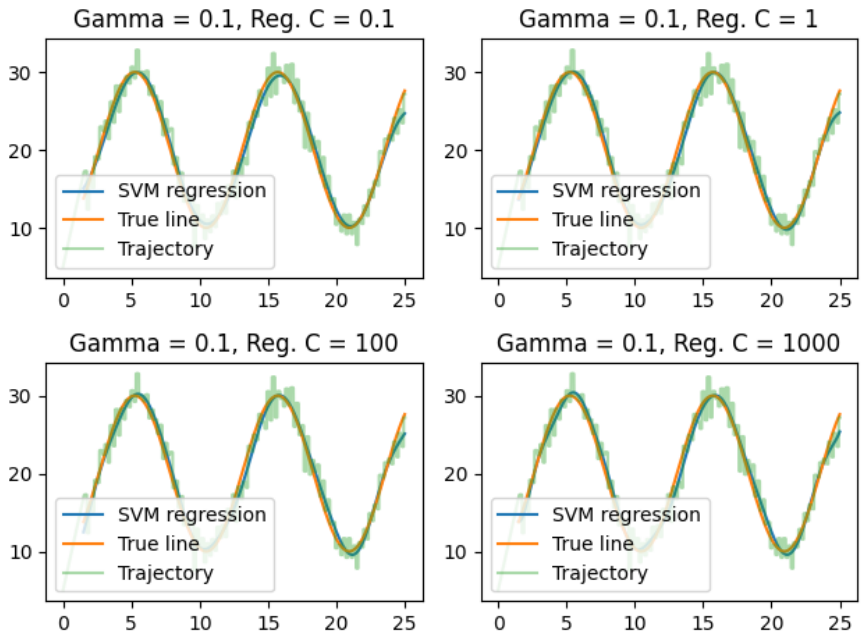


Figure 6.7: SVM regression gamma = 0.1 different C.

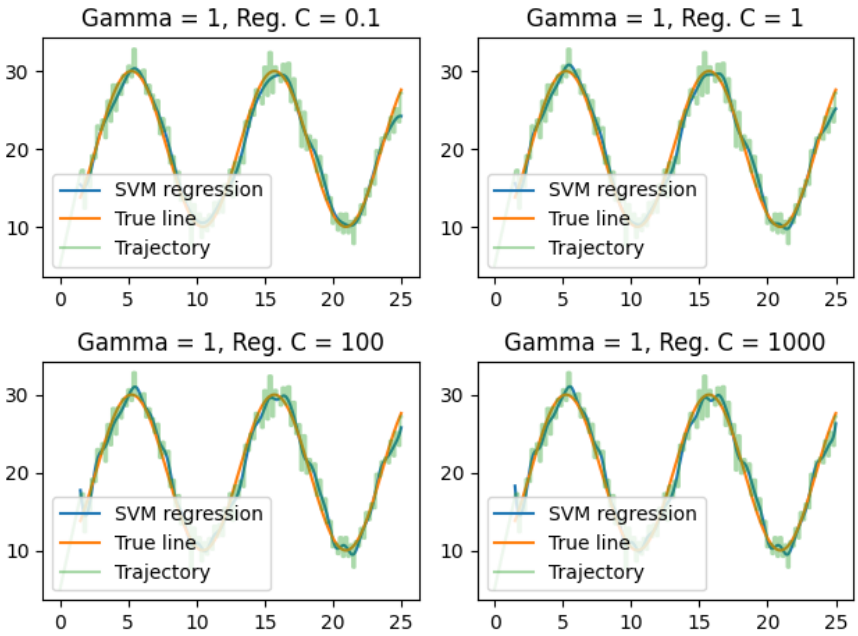


Figure 6.8: SVM regression gamma = 1 different C.

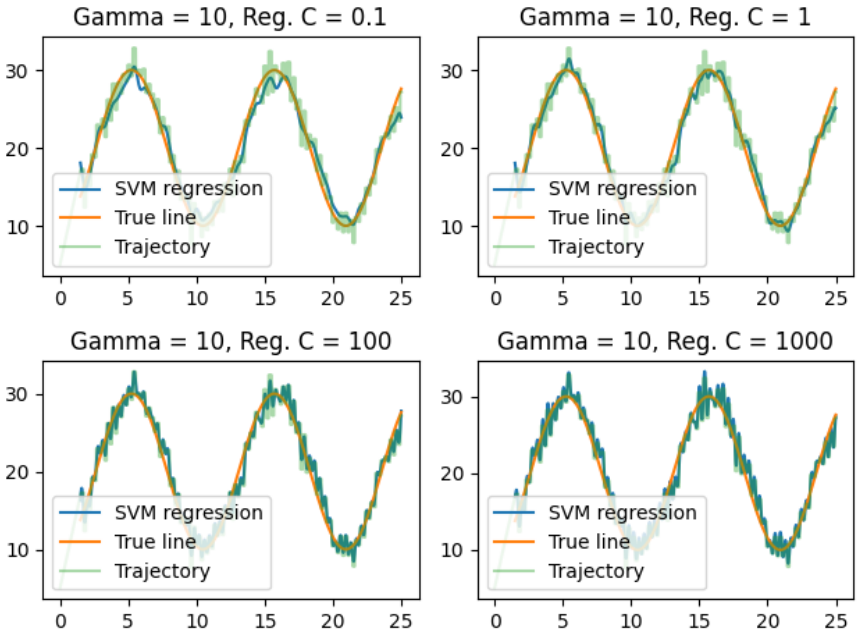


Figure 6.9: SVM regression gamma = 10 different C.

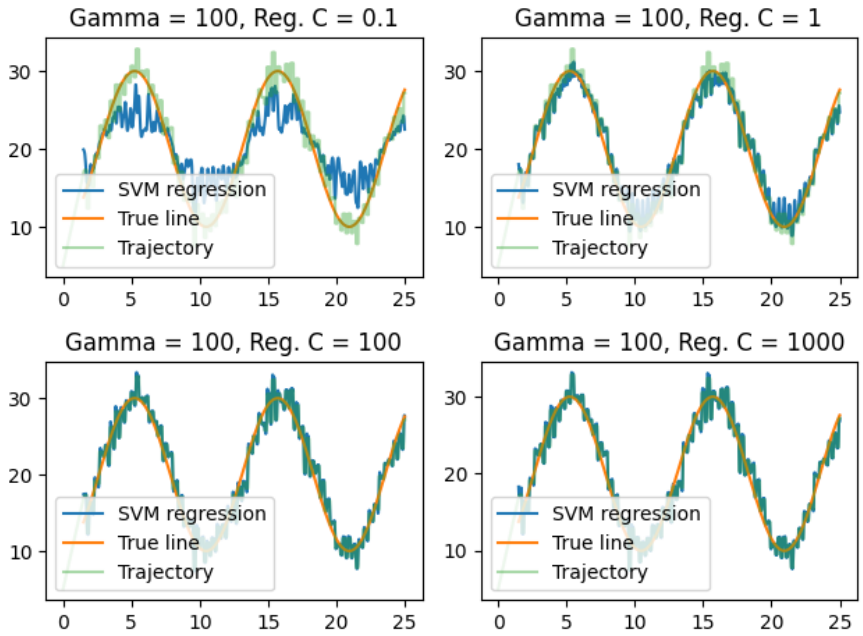


Figure 6.10: SVM regression gamma = 100 different C.

The results show how the best "width" parameter's setting for $\gamma = 0.1$, as with larger values, the estimated curve captures too many oscillations of the agent's trajectory. As, for the regularization parameter $C$, if we pick $\gamma = 0.1$, any value is fine, but if we chose larger values of $\gamma$, then $C = 0.1$, would be the best one among the $4$ that have been tested, namely a large regularization strength.

Moreover, we also try the SVM classification algorithm, with a gaussian kernel and with same choice of parameters as in Eq. 6.19 and in Eq 6.20.

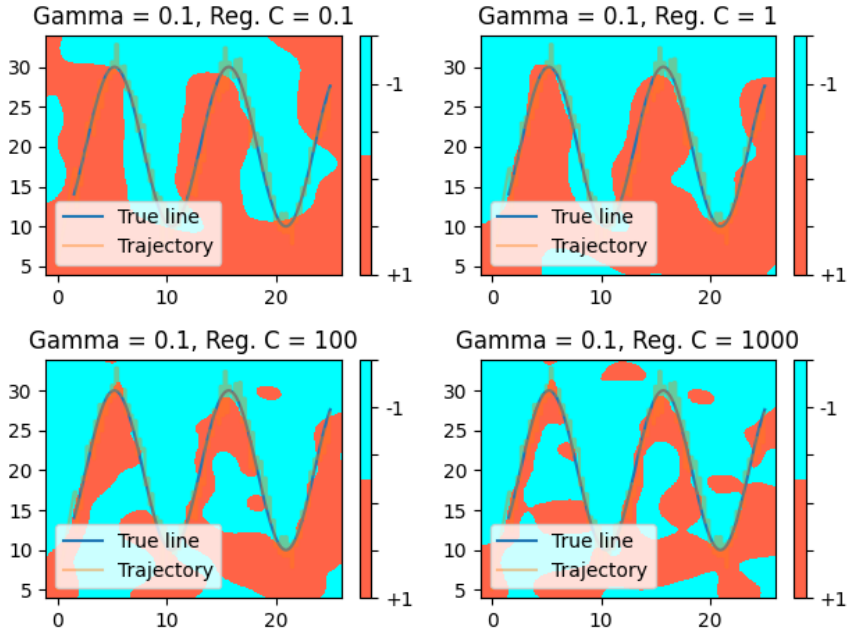


Figure 6.11: SVM classification gamma = 01 different C.

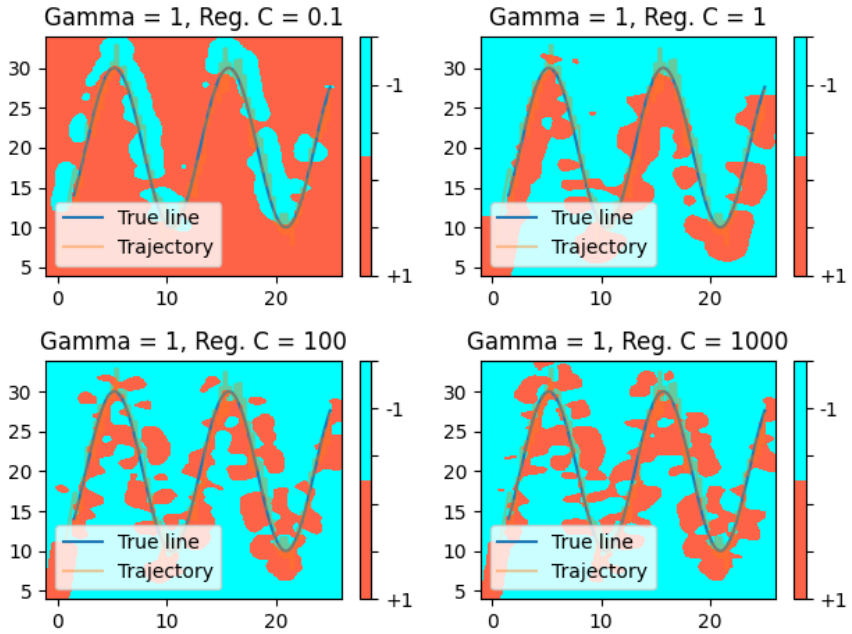


Figure 6.12: SVM classification gamma = 1 different C.

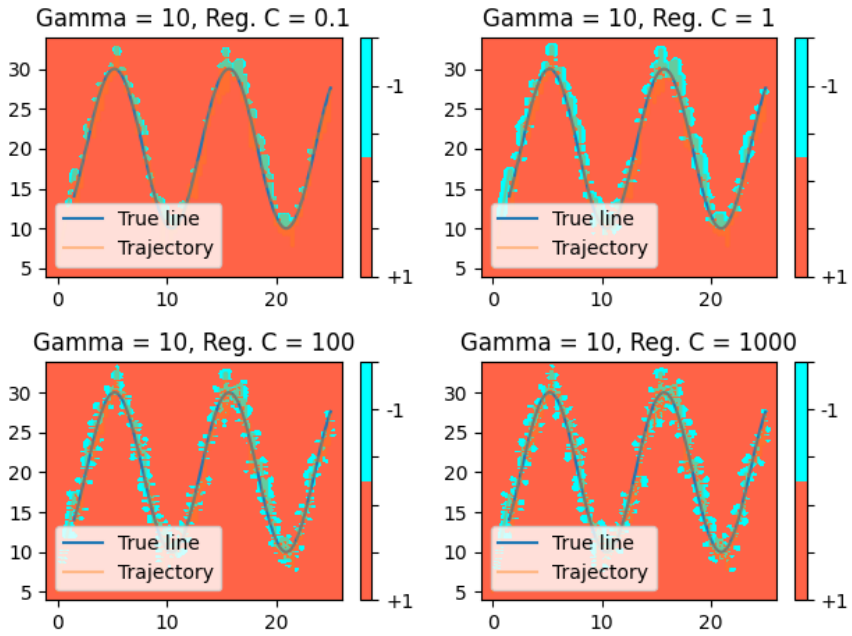


Figure 6.13: SVM classification gamma = 10 different C.

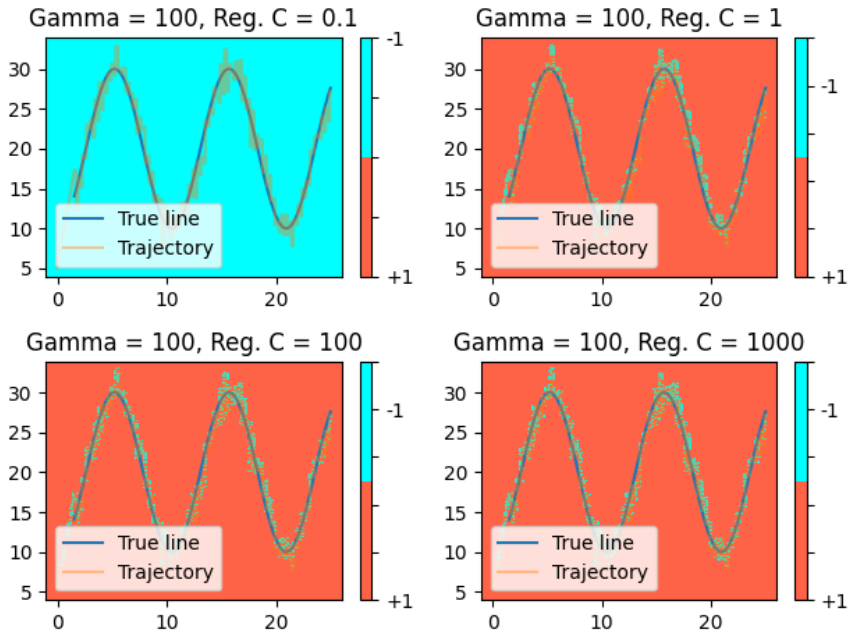


Figure 6.14: SVM classification gamma = 100 different C.

The results are quite bad. This because the classification accuracy is very sensitive to errors in the labels. The only significant results are with $\gamma = 0.1$. So, with such value's choice for this hyperparameter, we filter the data by selecting only

the ones that have 5 consecutive neighboring point with the same label, in order to remove corrupted data. We obtain the following estimates:

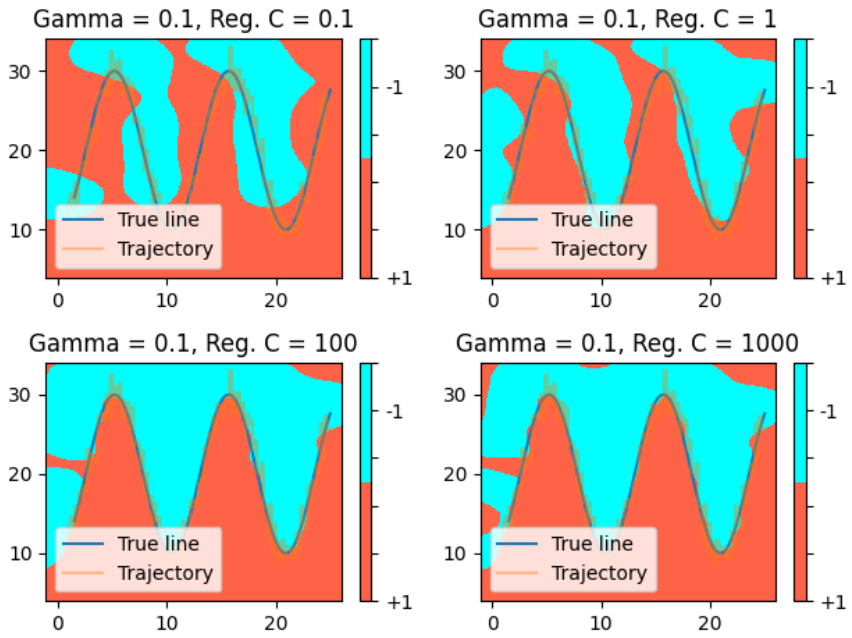


Figure 6.15: Improved estimates of SVM classification via processing the data.

We notice how for small values of $C$, the results are very bad, as the regularization term tends to make a smoother trajectory but some mislabelled points can still be present, and hence the smooth. On the other hand, with large values of $C$, we obtain better results, as the penalty to be on the wrong side is not too high.

Next, in the following image, we compare the best models of every approach, but ignoring the classification model, as it is not very well performing:

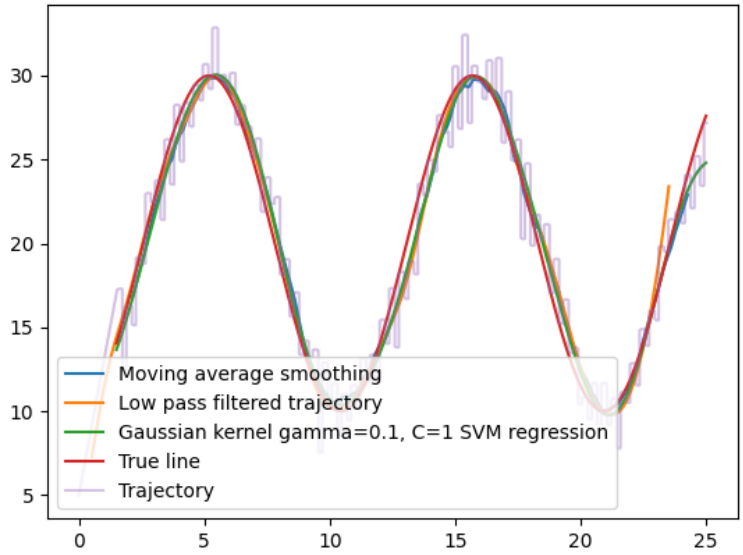


Figure 6.16: Comparison between different estimation approaches.

We notice how the two low-pass filter estimate achieves very nice results, fairly comparable with the SVM regression model. Moreover, as previously discussed, the tuning of these two methods are way easier than the one of a machine learning approach, and in particular does not depend on the type of curve but, rather, on the dynamics of the agent, meaning on the number of periods and the distance between them. Instead, for a machine learning algorithm, the tuning of the hyperparameters strongly depends on the type of curve, and a classic cross-validation approach is not valid as there are a lot of data points with wrong labels with respect the actual separative curve. Besides that, the important difference in the computational cost leads us to prefer the two low-pass filters instead of a classic regression or classification method.

We also notice that the sampling time $T$ influences not only the total number of data points, but also how far the agent moves away from the separative curve. Indeed, the larger it is, the larger are the oscillations made by the agent around the separative curve. This can be seen as having a noise with larger variance.

### 6.2.6 Non linear case with uniform distributed error noise

We consider now the case where the separative curve is non linear and the measurements of the labels are affected by uniform distributed noises, as in Section 5.3.
To achieve navigation and estimation of the curve by the robot, we can use again Algorithm 6 with the relative $labelling()$ function for the uniform distributed error, as in 5.3.2.
For the simulation, we use the same parameters as in Section 6.2.5 for the gaussian case, and for the estimation we use the two filters and the SVM regression with gaussian kernel (with parameters' choice of $\gamma = 0.1$, $C = 1$).
The trajectory obtained is the following:

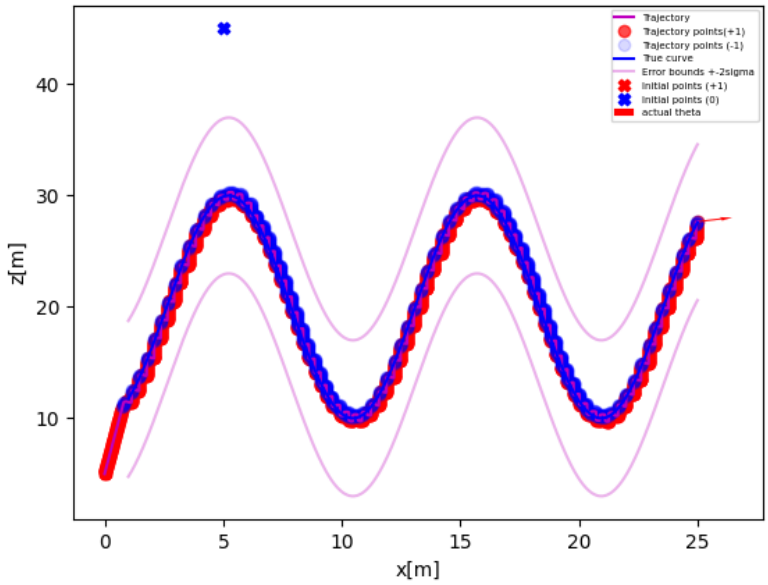


Figure 6.17: Trajectory of the agent with unif. noise in the measurements.

The agent navigates correctly around the separative curve.

Moreover, the estimated curves are drawn in the following plot:

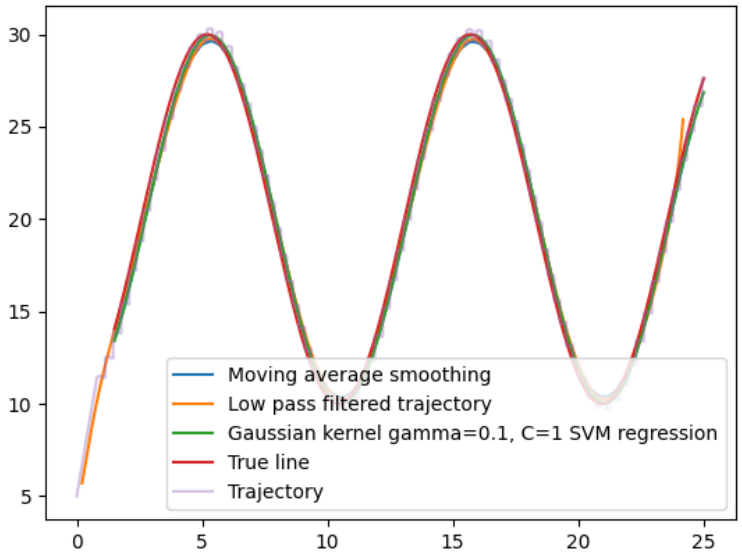


Figure 6.18: Estimated curves with 3 different approaches.

To compare the results, we run $n = 100$ simulations, slightly changing the curve as follows:

$$z(x) = -10\,cos\left(0.6\,\frac{i+1}{i+20}x\right) + 20, \tag{6.21}$$

where $i$ goes from $0$ to $99$.
At each iteration we compute the Root Mean Square Error (RMSE), described in Section 8.10, both for the Moving Average (MA) estimate, for the Butterwoth (BU) estimate and for the SVM estimate and we store the results in a list in order to realize $3$ final histograms.

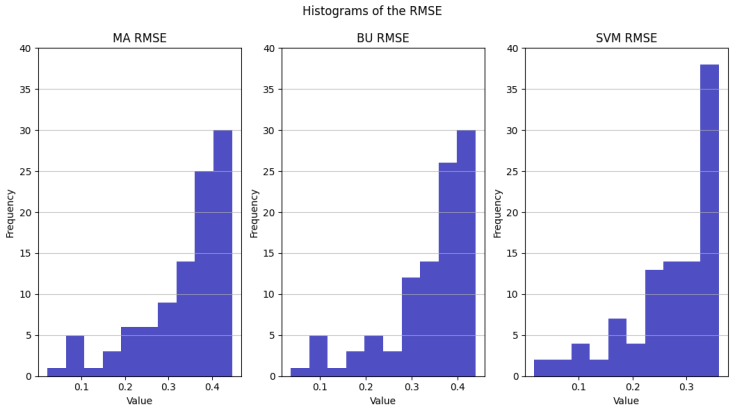


Figure 6.19: Histograms with the RMSE of the estimated curves with the 3 different approaches.

All the three estimates are quite accurate, but we prefer the two filtering techniques rather than the SVM algorithm as it is less computational intensive.
We notice that the choice of the curve of Eq. 6.21 gives a set of sinusoidal curves that are more flat in the first iterations ($i$ small) and becomes less smooth in the latest iterations when the frequency of the wave function is higher. So we expect that the estimation will be better in the first iterations, and this is in fact showed by the next figure:

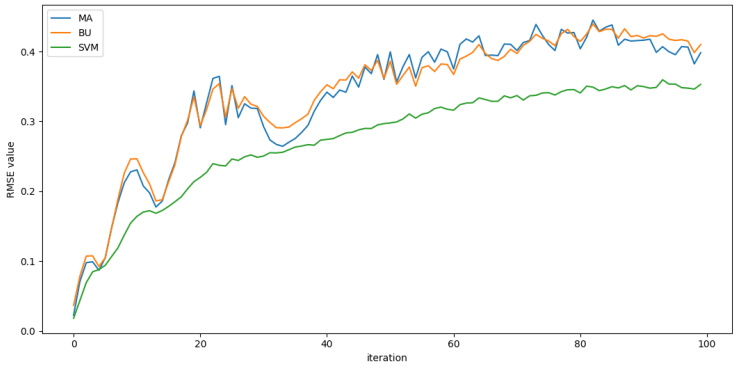


Figure 6.20: RMSE behaviour in function of the iterations and in turn of the regularity of the curve.

The figure 6.20 shows how the RMSE increases as long as the cosinusoidal function presents more variability in terms of its derivative. The three curves are not monotonically increasing, but this can be explained by the fact that the observations of the labels are subjected to some randomic error, uniformed distributed

in this case. Moreover the three curves are converging to some value as the ratio $\frac{i+1}{i+20}$ tends to $1$ as $i$ goes to large values, hence yielding to the same curve to be estimated.

As final comment, we notice how the SVM estimates are better than the other two approaches. On the other hand, all the $3$ estimates are quite accurate and the MA and the BU have complexity linear on the number of data points, whereas the complexity of SVM is at least $O(n^2)$.

# 7

# Conclusions and Future Work

## 7.1 Conclusions

Resuming what has been done in this work, we can say that we have gone through the problem of estimating a separative linear or non linear curve between two labeled regions in the space.
We firstly attached the problem from a pure sampling point of view without thinking of a possible dynamics of an agent, but just thinking of the possibility of freely sampling any point in the space. We have developed an algorithm, namely Algorithm 1, capable of estimating with accuracy a point belonging to such a separative curve, in any dimensional space. Moreover, we extended the problem to the non linear shape of the curve, before analysing the possibility of using the kernel trick to make linearly separable non linear curves mapping the original space in a higher dimensional space, but we saw that this way was not productive as then we cannot go back in the lower dimensional space and retrieve the separative curve since the map is not invertible. Thus, we reasoned before assuming to have access to the shape of the separative curve, as in practice this can also hold (for example if we know that an object is circular and we want to detect it), and in this case we achieved good estimates using an elliptic shape and estimating it by solving a linear algebraic system as in Eq. 3.17, once we have had estimated the $5$ necessary points with Algorithm 1. We then relaxed the assumption of knowing the separative shape, and we think of the problem as a non linear regression problem, or better as an interpolation problem, and we estimated the same ellipse, noticing that the larger the number of estimated

points, the better the final curve estimate.
After that, we moved to the dynamics part, where the goal was not only the estimation of the curve, but also the navigation along it. We adapted a single discrete-time integrator adopting an additional state coordinate, namely the direction of our agent, useful for the navigation's procedures. We first solved the problem for the linear case with no measurement errors, developing a control strategy able to track the line and to converge to the same slope with a certain error, as large as the variable $\Delta\theta$ is chosen.
Then, we adopted a uniform distributed error noise model on the measurements of the labels, and we developed Algorithm 3 able to solve the problem in a randomic setting, and we retrieved a relation between the total number of points necessary to reach convergence and the probability of the success of such algorithm, exploiting the Markov properties.
We also adopted a gaussian distributed error model in the measurement of the labels and here we could not achieve convergence, but we have been able to develop a procedure to navigate around a separative curve. We then used different estimation's approaches to finally estimate the separative curve, relaxing the computational effort that was quite hard with ML methods, such as SVM, but way less demanding with some simpler filters, like the MA and the BU one, and achieving pretty much the same levels of accuracy.
We finally discussed also the non linear separative curve case, that is the hardest one. This latter setup is not easily solvable with an agent like ours that is capable to sample just the data points where it is laying on, and therefore the convergence of the agent's trajectory to the separative curve looks infeasible. However, is still possible to navigate around the separative curve, both in the ideal no errors case and in the more complex error modelled case (both uniform and gaussian distributed), and we developed, for instance, Algorithm 4 to navigate around a non linear sinusoidal curve.
We also discussed about the max curvature's problem to capture the information on the variation of the curvature between two consecutive estimated points on the separative curve and using it to strategically speed up or speed down the agent, depending on how much the curve is rapidly varying.

In general, we have developed different algorithms and strategies to move a simple agent, from the theoretic perspective, in an innovative way that differs quite

a lot from the methods present in this field, such as dual control or reinforcement learning, or motion planning algorithms rooted on random sampling (e.g. RRT*). In fact whereas all these methods rely on a huge amount of sampling of data points on the space, before of estimate a good strategy to move, our approaches are more direct and aimed to online update the control input of our agent to make it moving around the separative curve. Moreover, a lot of methods present in literature for robotic or motion plannings rely on observation data from the feature position, namely from sensors that sample data points in front of it, whereas our agent, at every iteration, uses just the only point where it is laying on, and with it and with the previous ones, tries to find the best way to move.

## 7.2 Future work

Further research follows-up from this work.
We would like to better formalize all our algorithms from a theoretic and mathematical point of view, and compare them with other already existing algorithms in the same scenario.
We would also like to extend our algorithms to an agent with more practical dynamics, such as a unicycle, bicycle or an aerial multi-rotor agent.
An interesting extension could be use a $3D$ space and try to simulate all the algorithms. Of course, new challenges will arise, such as the increment of the computational time.
Another interesting extension might also be to use Algorithm 1 to higher dimensions, in research topic that may be even slightly different, touching different areas of ML and in general data science.

# 8 Appendix

## 8.1 Intro

In this chapter, some concepts that have been useful during the development of the other chapters of the thesis are presented. They are about control, machine learning, probability, estimation methods. Besides, the code snippets used to implement the algorithms are reported in the last part of this chapter.

## 8.2 Kernel's methods

### 8.2.1 Inner product space

An inner product is a map $\langle \cdot, \cdot \rangle_{\mathbb{X}} : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{K}$ satisfying the following axioms:

- Symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbb{X}} = \langle \mathbf{y}, \mathbf{x} \rangle_{\mathbb{X}}$
- Bilinearity: $\langle a\mathbf{x} + b\mathbf{y}, c\mathbf{z} + d\mathbf{w} \rangle_{\mathbb{X}} = ac\langle \mathbf{x}, \mathbf{z} \rangle_{\mathbb{X}} + ad\langle \mathbf{x}, \mathbf{w} \rangle_{\mathbb{X}} + bc\langle \mathbf{y}, \mathbf{z} \rangle_{\mathbb{X}} + bd\langle \mathbf{y}, \mathbf{w} \rangle_{\mathbb{X}}$
- Non-negativity: $\langle \mathbf{x}, \mathbf{x} \rangle_{\mathbb{X}} \geq 0$
- Positive definiteness: $\langle \mathbf{x}, \mathbf{x} \rangle_{\mathbb{X}} = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$

The standard inner product in the Euclidean space, $\mathbf{x} \in \mathbb{R}^d$ and $d \in \mathbb{N}$, is called the dot product: $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbb{R}^m} = \sum_{i=1}^{m} x_i y_i$.

### 8.2.2 KERNELS

**Definition (Positive semi-definite kernel)**
$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is positive semi-definite if:

- $\forall(\mathbf{x}, \mathbf{x'}) \in \mathbb{R}^d \times \mathbb{R}^d$, $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$.
- $\forall m \in \mathbb{N}, \forall \xi_1, ..., \xi_m \in \mathbb{R}, \forall \mathbf{x}_1, ..., \mathbf{x}_m \in \mathbb{R}^d$, $\sum_{i,j}^{m} \xi_i \xi_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$.

**Theorem (Moore-Aronsjan (1950))**
To every positive semi-definite kernek k, there exists a Hilbert space $\mathbb{H}$ and a feature map $\phi : \mathbb{R}^d \to \mathbb{H}$ such that for all $x_i$, $x_j$ we have
$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i, \phi(\mathbf{x}_j) \rangle_{\mathbb{H}}$.

**Operation on kernels** Let $k_1$ and $k_2$ be positive semi-definite, and $\lambda_{1,2} > 0$ then:

1. $\lambda_1 k_1$ is a valid kernel.
2. $\lambda_1 k_1 + \lambda_2 k_2$ is positive semi-definite.
3. $k_1 k_2$ is positive semi-definite.
4. $exp(k_1)$ is positive semi-definite.
5. $g(\mathbf{x}_i) g(\mathbf{x}_j)$ is positive semi-definite, with $g : \mathbb{R}^d \to \mathbb{R}$.

**Some examples of kernels**
The most common kernels are:

- **Polynomial kernel** $k(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + q)^p$, where $p$ is the order and $q$ is the bias.
- **Gaussian kernel** $k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2})$. It can be also further generalized picking a different distance from the euclidean one.

**Why kernels are important in ML?**
Traditionally, theory and algorithms of machine learning and statistics has been very well developed for the linear case. Real world data analysis problems, on the other hand, often require nonlinear methods to detect the kind of dependencies that allow successful prediction of properties of interest. By using a positive definite kernel, one can sometimes have the best of both worlds. The kernel corresponds to a dot product in a (usually

high-dimensional, some times even infinite large) feature space. In this space, our estimation methods are linear, but as long as we can formulate everything in terms of kernel evaluations, we never explicitly have to compute in the high-dimensional feature space. This "trick", is called **kernel trick**.
A kernel is usually seen as a measure of similarity between two samples. It reflects in some sens, how two samples are similar and this allows us to construct algorithms in dot product spaces.
In practice, it is possible to define kernels using some a priori information of our data. For instance: in image classification. It is possible to build kernels that includes information from the spatial domain.

## 8.3 Classification background

In machine learning, classification is a predictive modeling problem where the class label is anticipated for an example of input data.
It can be part of both supervised and unsupervised machine learning. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier.
Several classifiers have been developed in the last decades to solve the classification task. For the supervised learning there exist, for instance, neural networks, decision trees, linear regression, and support vector machines classifiers.
For the unsupervised learning there exist Hidden Markov models, k-means, hierarchical clustering, and Gaussian mixture models classifiers.
In addition to the two above methods, there exist another type of learning, that is reinforcement learning, that is neither supervised nor unsupervised learning as it doesn't require labeled data and not even a training set as it is based on the interaction with the environment through positive or negative rewards (also called feedbacks) to learn a specific goal.
Common algorithms in this field include temporal difference, deep adversarial networks, and Q-learning.

### 8.3.1 Binary linear classification

**Problem's statement**
Given a data set, also called training set, $\{x_i, y_i\}_{i=1...m}$, where $x_i \in \mathbb{R}^d$

$y_i \in \{-1, 1\}$ the goal is to find $w \in \mathbb{R}^{d \times m}$ and $b \in \mathbb{R}^m$, where $d$ is the dimension of the feature space and $m$ is the number of data points, in such a way that $y_i(w^T x_i + b) > 0 \; \forall i$, i.e. all points are correctly classified. In binary classification of course $d = 2$.

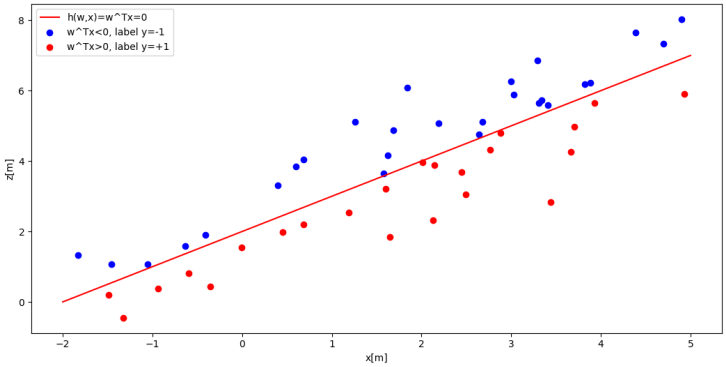


Figure 8.1: Geometrical representation in $2D$ case of the classification's problem.

Indeed, as depicted in Figure 8.1, the points with label $y = +1$ are laying above the hyperplane and hence $w^T x + b > 0$ whereas the points with label $y = -1$ are laying below the hyperplane and hence $w^T x + b < 0$. If such a $w$ and $b$ exist, than the dataset is called linearly separable. Often times the dataset is not linearly separable.

**Note**: all the Figures represent hyperplane with just 2 dimensions, namely the feature's space has just 2 dimensions as the drawing of bigger feature space is almost impossible.

As the geometrical view of the problem is meaningful to understand the problem, the next section is dedicated to a further description of the classification task from a geometrical and algebraic point of view.

### 8.3.2 Geometry of linear classification

Given a point $x | w^T x + b = 0$, namely a point belonging to the separable hyperplane, we wonder how the orthogonal decomposition of such a point, that can be meant as a vector in the feature space, is made. In other words, we want to understand how the vectors $x_\perp$ and $x_\parallel$ can be seen. If $w^T x + b = 0$ and $x = x_\perp + x_\parallel$, then $w^T x = -b$, namely $w^T(x_\perp + x_\parallel) = -b$. But since $w^T x_\parallel = 0$, then $w^T x_\perp = -b$, that is $-b = \langle w \; , x_\perp \rangle = \pm \|w\| \|x_\perp\|$.

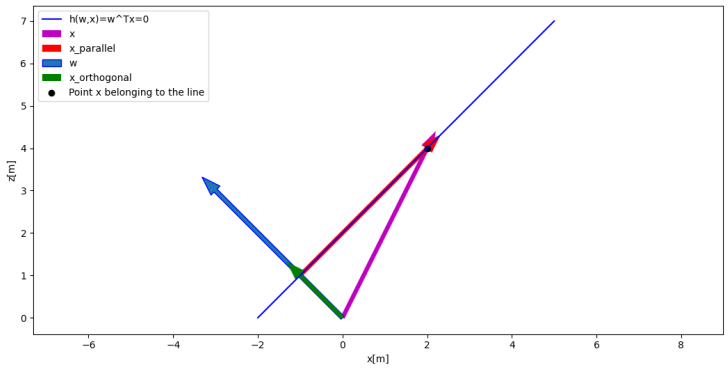


Figure 8.2: Orthogonal projections of $x$ belonging to the separable hyperplane.

Now a question arise spontaneously: what happens when $x|w^T x + b \neq 0$ ?

Let $x = x_\perp + x_\parallel + \tilde{x}$, by using the same reasoning of before,
$w^T x + b = w^T(x_\perp + x_\parallel + \tilde{x}) = w^T \tilde{x}$, as $w^T(x_\perp + x_\parallel) + b = 0$. Thus
$w^T x + b = w^T \tilde{x} = \pm \|w\| \|\tilde{x}\|$, where the sign depends on the orientation of $w$, $\tilde{x}$ (+ if they have the same orientation).

**Note**: $\|\tilde{x}\| = d(x, w^T x + b)$, where $d(\cdot, \cdot)$ is the euclidean distance.

Eventually, $\frac{w^T x + b}{\|w\|} = \pm \|\tilde{x}\|$, and by observing that the parameters $w$ and $b$ can always be normalized such that $\|w\| = 1$ (since the hyperplane's equation actually is $c(w^T x + b) = 0$, $c \neq 0$), $w^T x + b = \pm \|\tilde{x}\|$.

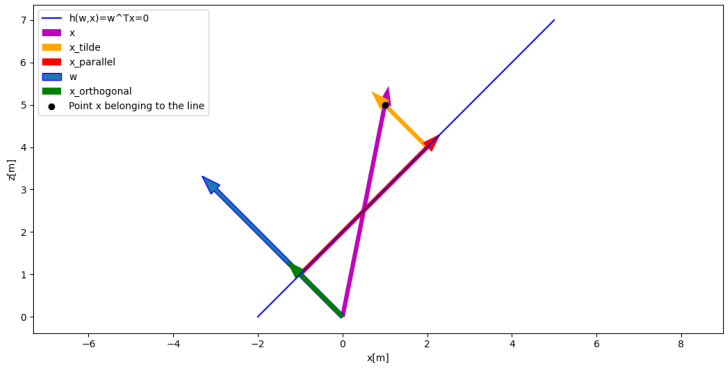


Figure 8.3: Orthogonal projection of $x$ not belonging to the separable hyperplane.

### 8.3.3 Perceptron algorithm

Perceptron algorithm is the first example of neural network (with just one neuron) and is a two-class (binary) classification machine learning algorithm. It was invented in 1943 by McCulloch and Pitts. The aim of the algorithm is to find an hyperplane that separates two data sets given their labels. More in details, the classification rule is given by the function:

$$h_{w,b}(x) = \begin{cases} +1, & \text{if } w^T x + b \geq 0 \\ -1, & \text{if } w^T x + b \leq 0 \end{cases} = sign(w^T x + b) \tag{8.1}$$

From now on, let's consider the notation $h_w(x) = sign(w^T x)$, where we make use of the so-called extended space, namely $w = \begin{bmatrix} w & b \end{bmatrix}$ and $x = \begin{bmatrix} x & 1 \end{bmatrix}^T$.
The goal of the algorithm is, given a training data set $\{x_i, y_i\}_{i=1...m}$, to find $w$ such that $w^T x = 0$, namely the training examples are correctly separated.
**Note**: the algorithm works only with linearly separable data.

**Algorithm 7** Perceptron algorithm (Rosenblat 1958)

$w^0 \leftarrow 0 \ (\in \mathbb{R}^d)$
**while** $True$ **do**
  select an index $i \in [m]$ s.t. $y_i (w^k)^T x_i \leq 0$
  **if** such an index cannot be found **then**
    $break$
  $w^{(k+1)} = w^k + y_i x_i$

### 8.3.4 Logistic regression

Logistic regression, although the confusing name, is a type of classification that is built on the fact that $w^T x$ is proportional to the distance of point $x$ from the separable hyperplane. The following probabilistic model for classification is introduced:

$$\begin{aligned} &P[y = 1|x] \propto e^{(w^T x)}, \\ &P[y = -1|x] \propto e^{-(w^T x)}, \\ &P[y = 1|x] + P[y = -1|x] = 1. \end{aligned} \tag{8.2}$$

This implies that:

$$\begin{aligned} P[y=1|x] &= \frac{e^{(w^T x)}}{e^{(w^T x)} + e^{-(w^T x)}} = \frac{1}{1+e^{-2(w^T x)}}, \\ P[y=-1|x] &= \frac{e^{-(w^T x)}}{e^{(w^T x)} + e^{-(w^T x)}} = \frac{1}{1+e^{2(w^T x)}}. \end{aligned} \tag{8.3}$$

By renaming $2w \rightarrow w$:

$$P[y=1|x] = \frac{1}{1+e^{-(w^T x)}}, \tag{8.4}$$

$$P[y=-1|x] = \frac{1}{1+e^{(w^T x)}}. \tag{8.5}$$

Since $y \in \{-1, +1\}$, then:

$$P_w[y|x] = \frac{1}{1+e^{-(yw^T x)}}, \tag{8.6}$$

where the latter equation can be seen as a composition between the well known sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}}$, and a function $f(y, x, w) = yw^T x$, whose sign tells us whether the data point $x$ is correctly classified or not.

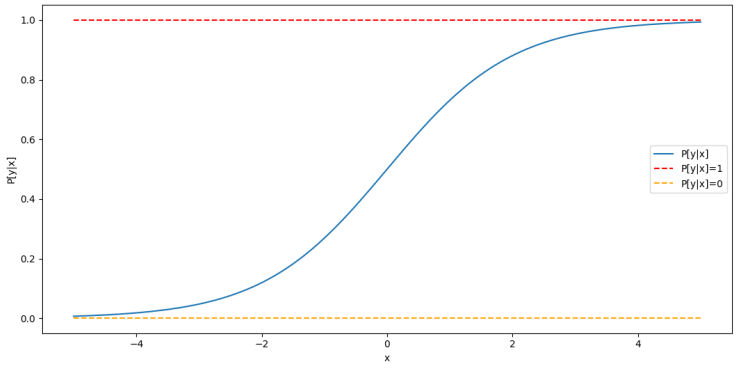


Figure 8.4: Decision's function for logistic regression.

So, in training logistic regression we aim to find the best $w$, call it $\tilde{w}_s$, so that:

$$P_{\tilde{w}_s}[y|x] = \frac{1}{1+e^{-(y\tilde{w}_s^T x)}}. \tag{8.7}$$

Provided $(x_i, y_i)$ independent,

$$P_w[y_1, y_2, ..., y_m | x_1, x_2, ..., x_m] = \prod_{i=1}^{m} P_w[y_i|x_i] = \prod_{i=1}^{m} \frac{1}{1 + e^{-(y_i w^T x_i)}} \tag{8.8}$$

is the likelihood function, and considering a maximum likelihood estimator, we obtain:

$$\tilde{w}_{ML} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{1 + e^{-(y_i w^T x_i)}}. \tag{8.9}$$

By using the classical reasoning about maximization of the likelihood function,

$$\begin{aligned} \tilde{w}_{ML} &= \underset{w}{\operatorname{argmax}} \log \prod_{i=1}^{m} P_w[y_i|x_i] \\ &= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{m} \log P_w[y_i|x_i] \\ &= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{m} (log1 - log(1 + e^{-(y_i w^T x_i)})) \\ &= \underset{w}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} log(1 + e^{-(y_i w^T x_i)}) \end{aligned} \tag{8.10}$$

This allows to get a new loss function,

$$L(w, x, y) = \frac{1}{m} \sum_{i=1}^{m} log(1 + e^{-(y_i w^T x_i)}) \tag{8.11}$$

whose minimization gives us the best $w$ for the given data set.
How to minimize it?
It can be proven that this loss function is convex and then the optimal $\tilde{w}_{ML}$ comes out setting the gradient of the loss function equal to zero.

The last point to complete the logistic regression task is how do we classify a new data point $x$?

we just follow the classification rule defined as:

$$\tilde{y} = \underset{y \in \{1,-1\}}{\operatorname{argmax}} P_{\tilde{w}_{ML}}[y|x] \tag{8.12}$$

#### Logistic regression for multiclass classification

Consider the case where $y = \{0, 1, ..., k-1\}$, there are $k$ different classes.
The natural extension of the logistic regression presented before, is:
$\sum_{j=0}^{k-1} P[y = j|x] = 1$ and $P[y = j|x] \propto e^{w_j^T x}$, $w_j \in \mathbb{R}^d$.
This leads to:

$$P[y = j|x] = \frac{e^{w_j^T x}}{\sum_{i=0}^{k-1} e^{w_i^T x}} \tag{8.13}$$

#### Non linear extension

If we are in the non linear case, we can just use:

$$P[y = j|x] = \frac{e^{h_j(x)}}{\sum_{i=0}^{k-1} e^{h_i(x)}} \tag{8.14}$$

### 8.3.5 Support Vector Machine (SVM) classification

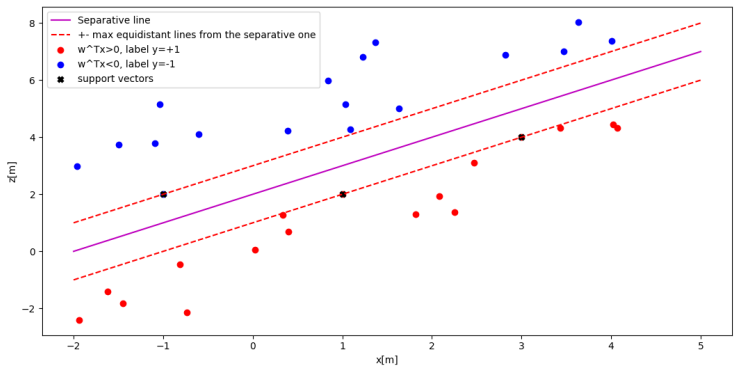


Figure 8.5: Geometrical representation in $2D$ of SVM in the linear separable case.

SVM algorithm (see [11]) is a powerful tool to make classification. Indeed, while the Perceptron algorithm finds an hyperplane that separates the training data without knowing exactly which one among the infinitely many that do the same job, and the logistic regression makes a "probabilistic" classification, SVM finds $\tilde{w}$ and $\tilde{b}$ such that we maximize the quantity $\min_i d(z_i, line(w, b))$.
The statement of the problem, more formally, is as follows:

$$\tilde{w}, \tilde{b} = \underset{w,b}{\operatorname{argmax}} \min_i d_i, \tag{8.15}$$

where $d_i = \frac{y_i(w^T x_i + b)}{||w||}$ is the signed distance of point $x_i$ from the line.

$$\begin{aligned} \tilde{w}, \tilde{b} &= \underset{w|\ ||w||=1, b}{\operatorname{argmax}} \min_i y_i(w^T x_i + b), \\ &= \underset{w|\ ||w||=1, b, y_i(w^T x_i + b) \geq \gamma}{\operatorname{argmax}} \gamma. \end{aligned} \tag{8.16}$$

We define now, $\bar{w} = \frac{w}{\gamma}$, $\bar{b} = \frac{b}{\gamma}$, so that $||\bar{w}|| = \frac{1}{\gamma}$ (as $||w|| = 1$).

Thus, the optimization problem becomes:

$$\begin{aligned} \tilde{w}, \tilde{b} &= \underset{\bar{w}, \bar{b}, y_i(\bar{w}^T x_i + \bar{b}) \geq 1}{\operatorname{argmax}} \frac{1}{||\bar{w}||^2}, \\ &= \underset{\bar{w}, \bar{b}, y_i(\bar{w}^T x_i + \bar{b}) \geq 1}{\operatorname{argmin}} \frac{1}{2}||\bar{w}||^2. \end{aligned} \tag{8.17}$$

In the latter equation there are $m$ linear constraints and the function to be minimized is quadratic; this, allows to solve the problem with Lagrange duality.

Let's define the lagrangian function as (renaming $\bar{w}$ with $w$ and same for $b$):

$$\Lambda(w, b, \mu) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} \mu_i f(w, b), \tag{8.18}$$

where $f(w, b) = 1 - y_i(w^T x_i + b) \leq 0$.

The next step consists in minimizing $\Lambda$ over $w, b$.

$$\begin{aligned} \frac{\partial \Lambda(w, b, \mu)}{\partial b} &= -\sum_{i=1}^{m} \mu_i y_i = 0, \\ \frac{\partial \Lambda(w, b, \mu)}{\partial w} &= w - \sum_{i=1}^{m} \hat{\mu}_i y_i x_i. \end{aligned} \tag{8.19}$$

**Remark**: if $\sum_{i=1}^{m} \mu_i y_i \neq 0$, then $\inf_b \Lambda(w, b, \mu) = -\inf$.

$$
\begin{aligned}
g(\mu) &= \inf_{w,b} \Lambda(w, b, \mu) \\
&= \begin{cases} -\inf, & \text{if } \sum_{i=1}^{m} \mu_i y_i \neq 0 \\ \frac{1}{2}(\sum_{i=1}^{m} \mu_i y_i x_i)^T (\sum_{j=1}^{m} \mu_j y_j x_j) + \sum_{i=1}^{m} \mu_i - w^T \sum_{i=1}^{m} \mu_i y_i x_i - (\sum_{i=1}^{m} \mu_i y_i) b, & \text{if } \sum_{i=1}^{m} \mu_i y_i = 0 \end{cases} \\
&= \begin{cases} -\inf, & \text{if } \sum_{i=1}^{m} \mu_i y_i \neq 0 \\ \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i,j}^{m} \mu_i \mu_j y_i y_j {x_i}^T x_j, & \text{if } \sum_{i=1}^{m} \mu_i y_i = 0 \end{cases}
\end{aligned}
\tag{8.20}
$$

The last step consists on maximize $g(\mu)$:

$$
\begin{aligned}
\hat{\mu} &= \underset{\mu \in \mathbb{R}^m, \mu_i \geq 0}{\operatorname{argmax}} \; g(\mu) \\
&= \underset{\mu \in \mathbb{R}^m, \mu_i \geq 0, \sum_{i=1}^{m} \mu_i y_i = 0}{\operatorname{argmax}} \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i,j}^{m} \mu_i \mu_j y_i y_j {x_i}^T x_j
\end{aligned}
\tag{8.21}
$$

**Note**: we notice at this point that $\bar{\mu_i} = 0$ for those points that are not touching the boundary (in fact the optimization problem doesn't involve those points).

Thus, we have found out that:

$$
\begin{aligned}
\hat{w} &= \sum_{i=1}^{m} \bar{\mu_i} y_i x_i \\
&= \sum_{i \in \mathbb{I}} \hat{\mu}_i y_i x_i, \quad \mathbb{I} = \{i \; s.t. \; \hat{\mu}_i \geq 0\}
\end{aligned}
\tag{8.22}
$$

The solution $\hat{w}$ depends only on data points $z_i = (x_i, y_i)$ that lie on the boundary.

How to find $\hat{b}$?

For $x_i$ at the boundary holds that $1 - y_i(w^T x_i + b) = 0$, thus it's possible to retrieve $\hat{b}$ from that expression.

**Note**: this type of SVM is also called Hard SVM as is valid just for the separable case. In the next subsection the non separable case is treated.

### Soft SVM & Kernels

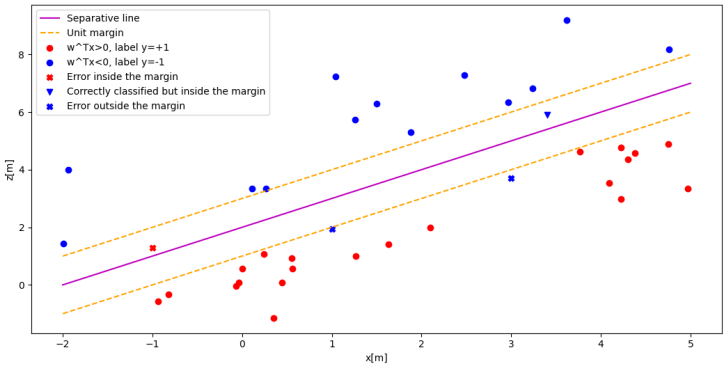


Figure 8.6: Geometrical representation in $2D$ of soft SVM.

To deal with the non separable case we need to introduce the so called slack variables $\xi_i$, that are useful to add a penalization for those points that are over the boundary given by the hyperplane. The optimization problem becomes:

$$\begin{aligned} \underset{w,b,\xi_i}{\text{argmin}} \quad & \frac{1}{2}||w||^2 + \lambda \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, ..., m \\ & \xi_i \geq 0 \end{aligned} \tag{8.23}$$

At the optimum:

- $\xi_i = 0 \; \forall$ points $i$ s.t. $y_i(w^T x_i + b) \geq 1$
- $0 < \xi_i < 1 \; \forall$ points $i$ s.t. $y_i(w^T x_i + b) \geq 0$ (that means they are correctly classified) and $y_i(w^T x_i + b) < 1$ (means they are inside the margin)
- $\xi_i > 1$ for incorrectly classified points
- $\xi_i = max\{0, 1 - y_i(w^T x_i + b)\}$

we can now rewrite (8.23) as follows:

$$\underset{w,b}{\text{argmin}} \quad \frac{1}{2\lambda m}||w||^2 + \frac{1}{m}\sum_{i=1}^{m} l(1 - y_i(w^T x_i + b)) \tag{8.24}$$

### Kernel SVM

The most straightforward way to deal with kernel SVM is given by the following procedure:

1. Introduce a feature map, namely a non linear transformation of my domain "$x$":
   $z = \phi(x) = \begin{bmatrix} \varphi_1(x) \\ \vdots \\ \varphi_N(x) \end{bmatrix} \in \mathbb{R}^N$
2. Find the solution $\hat{\mu}$ of (8.21), where now $x$ is substituted by the feature map $z$
3. Compute $\hat{w} = \sum_{i=1}^{m} \hat{\mu}_i y_i z_i$
4. Build up the decision function:
   $h_{\hat{w},\hat{b}}(z) = \hat{w}^T z + \hat{b} = \sum_{i=1}^{m} (\hat{\mu}_i y_i z_i^T z) + \hat{b} = \sum_{i=1}^{m} (\alpha_i \langle z_i, z \rangle) + \hat{b}$

**Note**: with the use of a feature map non linear maps can be achieved.

**Note**: we never need to compute explicitely $z = \phi(x)$ provided we can compute $\langle z_i, z_j \rangle \ \forall i, j$ and $\langle z_i, z \rangle \ \forall i, z$. we can directly define the inner product using a so-called "kernel function", that is a function of $x$ and not of $\phi(x)$:
$\langle z, z' \rangle = \langle \phi(x), \phi(x') \rangle = k(x, x')$.

**Note**: the kernel SVM is a non-linear classifier in the input space $\mathbb{R}^d$, but is still linear in the feature space (the space induced by the kernel function).

**Note**: the hyperparameters of the kernel must be tuned by some validation test (e.g. cross-validation).

### Multiclass SVM

Multiclass SVM can be performed using two approaches that collect in different ways binary classifiers:

- One versus All: each class is compared with all the other ones, leading to $m$ binary classifiers.
- One versus One: each class is classified with respect to any other single class, leading to $m(m-1)/2$ classifiers.

## 8.4 Regression background

Regression's task in ML formalizes and solves the problem of finding out a mathematical relation between measured variables based on a sample of data, called data set.
Let's consider a linear model

$$h_{w,b}(x) = w^T \mathbf{x} + b, \tag{8.25}$$

where $x \in \mathcal{R}^d$, $w \in \mathcal{R}^d$ and $b \in \mathcal{R}$.
$h()$ is an instance of the more general model class defined as:

$$\mathcal{H} := \left\{ h_w(\mathbf{x}) : \exists w \in \mathcal{R}^d,\ h_w(x) = \sum_{i=1}^{m} w_i \mathbf{x_i} \right\}. \tag{8.26}$$

The goal of the regression's task is to find $\hat{h}$ among the possible $h \in \mathcal{H}$ such that a certain loss function $\mathcal{L}(h)$ is minimized.

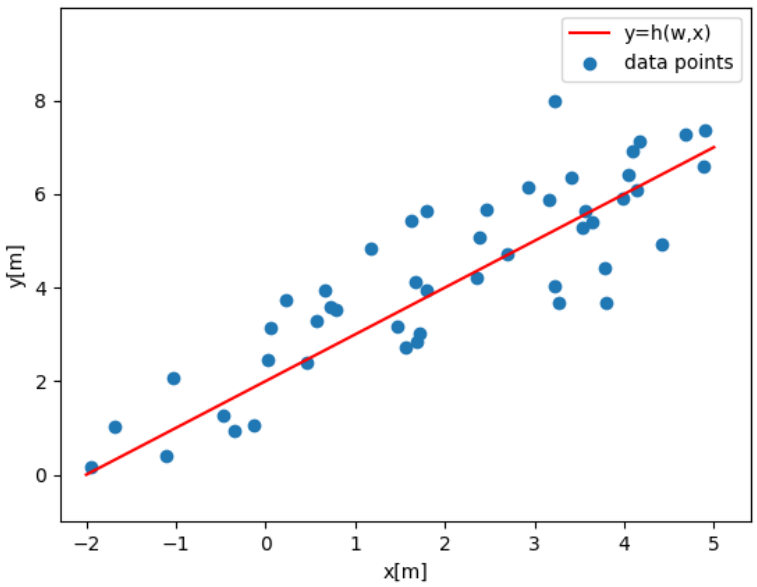


Figure 8.7: Regression's task.

We simplify the notation as follows:

$$\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \in \mathcal{R}^{d+1}, \quad \bar{w} = \begin{bmatrix} w \\ b \end{bmatrix}, \tag{8.27}$$

so that:

$$h_{w,b}(x) = w^T x + b = \begin{bmatrix} w^T & b \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \bar{w}^T \bar{\mathbf{x}}. \tag{8.28}$$

To simplify the notation, sometimes we use $\mathbf{x}$ to mean $\bar{\mathbf{x}}$, and $w$ to mean $\bar{w}$.
We adopt a quadratic loss, defined as follows:

$$\mathcal{L}(h, \mathbf{x}) = (y - h(\mathbf{x}))^2 = (y - \bar{w}^T \mathbf{x})^2. \tag{8.29}$$

In the case of linear regression and quadratic loss the solution of the problem, namely finding $\hat{w}$ and $\hat{b}$ that optimally describe the model $h_{w,b}(\mathbf{x})$ with a relative $\hat{h}_{\hat{w},\hat{b}}((x))$, is the solution given by the Empirical Risk Minimization Empirical Risk Minimization (ERM), namely:

$$\underbrace{\hat{w}}_{\in \mathcal{R}^{d+1}} = \underset{w}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \left( y_i - (\bar{w}^T x_i) \right)^2. \tag{8.30}$$

In the simplest case, where $\bar{w} \in \mathcal{R}^2$ and $\bar{\mathbf{x}} \in \mathcal{R}^2$, we find $\hat{b}$ imposing:

$$\frac{d\mathcal{L}(b)}{db} = 0. \tag{8.31}$$

So,

$$\begin{aligned} \frac{d\mathcal{L}(b)}{db} =& \frac{d}{db} \left( \sum_{i=1}^{m} \left( y_i - (w^T x_i - b) \right)^2 \right) = 0, \\ & - 2 \left( \sum_{i=1}^{m} \left( y_i - (w^T x_i - b) \right) \right) =, \\ & \sum_{i=1}^{m} y_i - w \sum_{i=1}^{m} x_i - mb =, \\ & \Rightarrow \hat{b} = \frac{\sum_{i=1}^{m} y_i - w \sum_{i=1}^{m} x_i}{m} = \bar{y} - w\bar{x}, \end{aligned} \tag{8.32}$$

where $\bar{y}$ and $\bar{x}$ are the arithmetic mean.

As far as $w \in \mathcal{R}$ it is sufficient to impose:

$$\begin{aligned}
\frac{d\mathcal{L}(w)}{dw} &= \frac{1}{m}\sum_{i=1}^{m} 2(y_i - wx_i - b)(-x_i) = 0, \\
&\Rightarrow \sum_{i=1}^{m}(-y_i x_i) + (wx_i^2) + (bx_i) = 0, \\
&\Rightarrow \sum_{i=1}^{m}(-y_i x_i) + (wx_i^2) + (\bar{y} - w\bar{x})x_i = 0, \\
&\Rightarrow \sum_{i=1}^{m} x_i y_i - (\bar{y} - w\bar{x})x_i - wx_i^2 = 0, \\
&\Rightarrow \sum_{i=1}^{m} x_i y_i - \bar{y}x_i + w\bar{x}x_i - wx_i^2 = 0, \\
&\Rightarrow \sum_{i=1}^{m} (x_i y_i - \bar{y}x_i) - w\sum_{i=1}^{m}\left(x_i^2 - \bar{x}x_i\right) = 0, \\
\hat{w} &= \frac{\sum_{i=1}^{m}(x_i y_i - \bar{y}x_i)}{\sum_{i=1}^{m}\left(x_i^2 - \bar{x}x_i\right)} = \frac{m\sum_{i=1}^{m}(x_i y_i) - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{m\sum_{i=1}^{m}(x_i^2) - \left(\sum_{i=1}^{m} x_i\right)^2}.
\end{aligned} \tag{8.33}$$

And also $\hat{b}$ can be rewritten as:

$$\hat{b} = \frac{m\sum_{i=1}^{m} y_i \sum_{i=1}^{m} x_i^2 - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} x_i y_i}{m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2}. \tag{8.34}$$

In the more general case, $\bar{w} \in \mathcal{R}^{d+1}$, $\acute{\mathbf{x}} \in \mathcal{R}^{d+1}$, it is needed to use partial derivatives:

$$\nabla_w \mathcal{L}(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_{d+1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial w_1}\left(\sum_{i=1}^{m}\left(y_i - w^T x_i\right)^2\right) \\ \vdots \\ \frac{\partial L}{\partial w_{d+1}}\left(\sum_{i=1}^{m}\left(y_i - w^T x_i\right)^2\right) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{8.35}$$

and knowing that:

$$\begin{aligned}\frac{\partial}{\partial w_j}\left(\sum_{i=1}^{m}\left(y_i - w^T x_i\right)^2\right) &= \frac{1}{m}\sum_{i=1}^{m} 2(y_i - w^T x_i)\left(-(x_i)_j\right) \\ &= \frac{2}{m}\left(\sum_{i=1}^{m} y_i\left(-(x_i)_j\right) + \underbrace{w^T x_i\left((x_i)_j\right)}_{\text{recall } (AB)^T = B^T A^T}\right) \\ &= \frac{2}{m}\left(-\sum_{i=1}^{m} y_i\left((x_i)_j\right) + \left((x_i)_j\right) x_i^T w\right),\end{aligned} \tag{8.36}$$

it holds that:

$$\begin{aligned}&\sum_{i=1}^{m} y_i x_i = \left(\sum_{i=1}^{m} x_i x_i^T\right) w \\ &\Rightarrow \hat{w} = \left(\sum_{i=1}^{m} x_i x_i^T\right)^{-1} \sum_{i=1}^{m} y_i x_i = \left(X^T X\right)^{-1} X^T Y,\end{aligned} \tag{8.37}$$

in the case $X^T X$ is invertible, and:

$$X := \begin{bmatrix} \bar{x}_1^T \\ \vdots \\ \bar{x}_m^T \end{bmatrix} \in \mathcal{R}^{m \times d}, \text{ and } Y := \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathcal{R}^m. \tag{8.38}$$

**Note**: in the case when $X^T X$ is not invertible, the Singular Value Decomposition Singular Value Decomposition (SVD) must be used to find a solution.

### 8.4.1 Ridge regression

In order to avoid overfitting, that happens whenever the learned model is too accurate and hence it performs well only with data sets similar to the training one, a regularization technique is needed. One very important regularization technique, suitable for the regression task, is Ridge Regression (RR).
The main difference from the normal regression is the introduction of a penalization term in the cost function to be minimized (that before was made

just by the Empirical Risk), as follows:

$$J_\lambda(h) = \mathcal{L}(h) + \lambda P(h), \tag{8.39}$$

where $P(h)$ is called penalty function, and is greater or equal than $0$ in norm, whereas $\lambda \geq 0$ is the regularization parameter controlling the trade-off between the fitting error (i.e. the Empirical Risk) and the penalty function. Let's find out the solution $\hat{w}_R(\lambda)$ in the RR problem, stated as:

$$\begin{aligned} \hat{w}_R(\lambda) &= \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} J_\lambda(w) \\ &= \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \left( y_i - (\bar{w}^T x_i) \right)^2 + \lambda ||w||^2. \end{aligned} \tag{8.40}$$

The solution is found setting the gradient of $J_\lambda(w)$ with respect to $w$ equal to zero, in fact, after rewriting the cost function in matrix form, we obtain:

$$\begin{aligned} J_\lambda(w) &= \frac{1}{m} (Y - Xw)^T (Y - Xw) + \lambda w^T w \\ &= \frac{1}{m} \left( Y^T Y - w^T X^T Y - Y^T X w + w^T X^T X w \right) + \lambda w^T w, \end{aligned} \tag{8.41}$$

hence:

$$\begin{aligned} \nabla_w J_\lambda(w) &= -\frac{1}{m} X^T Y - \frac{1}{m} X^T Y + \frac{1}{m} \left( X^T X w + X^T X w \right) + \lambda w + \lambda w \\ &= 2 \left[ \left( \frac{X^T X}{m} + \lambda I \right) w - \frac{X^T Y}{m} \right]. \end{aligned} \tag{8.42}$$

Thus, the solution of problem 8.40 is:

$$\hat{w}_R(\lambda) = \left( \frac{X^T X}{m} + \lambda I \right)^{-1} \frac{X^T Y}{m}. \tag{8.43}$$

From the latter equation, it is possible to notice that the matrix $\left( \frac{X^T X}{m} + \lambda I \right)$ is always invertible whenever $\lambda > 0$, and that if $\lambda = 0$, we obtain the same solution of the simple linear regression.
The optimal value of $\lambda$, that becomes another parameter to be estimated, can be found by a validation test. In general, it holds that for small values of $\lambda$ there is no regularization, and hence the outliers data points will have a lot of

importance, instead with very large values of $\lambda$ makes the solution $\hat{w}_R(\lambda)$ going to *zero* leading to a bad final model as well. Hence the optimal solution has to be sought somewhere in the middle of these two values.

### 8.4.2 Non linear regression

We want to extend the above description to a non linear model, as:

$$\mathcal{H} := \left\{ h_w(\mathbf{x}) : \exists w \in \mathcal{R}^d,\ h_w(x) = \sum_{i=1}^{m} w_i \Phi(\mathbf{x_i}) \right\}, \tag{8.44}$$

where it has been inserted a feature map, $\Phi(\mathbf{x})$, capable of describe a non linear behaviour of the data.
Then, the non linear regression goal is to find a model $\hat{h}(x)$ such that:

$$\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} (y_i - h(x_i))^2 . \tag{8.45}$$

Let's define:

$$Y := \begin{bmatrix} y_i \\ \vdots \\ y_m \end{bmatrix},\ \Phi := \begin{bmatrix} \Phi^T(x_1) \\ \vdots \\ \Phi^T(x_m) \end{bmatrix}, \tag{8.46}$$

so that we can eventually write the following optimization problem, very close to 8.30:

$$\hat{w} = \underset{w \in \mathcal{R}^d}{\operatorname{argmin}} \frac{1}{m} ||Y - \Phi w||^2, \tag{8.47}$$

whose solution is given by:

$$\hat{w} = \left(\Phi^T \Phi\right)^{-1} \Phi^T Y. \tag{8.48}$$

Also in this case, it is possible to use Ridge Regression RR to regularize the model, similarly to 8.4.1.

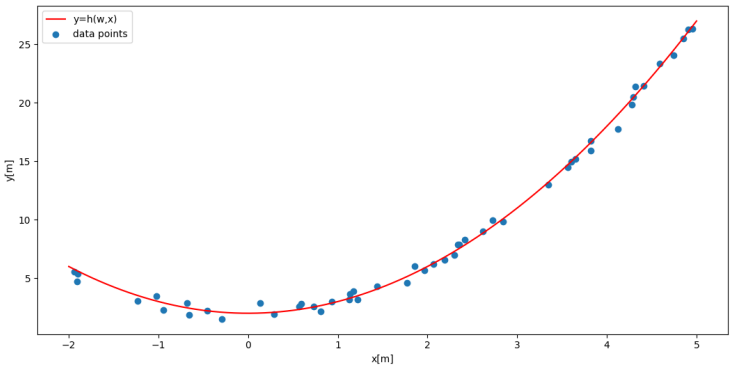


Figure 8.8: Non linear regression's example.

### 8.4.3 SVM REGRESSION

SVM algorithm can be extended to solve the regression task, just slightly changing the cost function to be minimized. In this case, in fact, we can minimize the following function:

$$\hat{w}, \hat{b} = \underset{w,b}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} l\left(y_i - (w^T x_i + b)\right) + \lambda ||w||^2, \tag{8.49}$$

that can be rewritten, as done in the SVM classification, as:

$$\hat{w}, \hat{b} = \underset{w,b,\text{ s.t. } |y_i - \langle w, x_i \rangle - b| \leq \epsilon}{\operatorname{argmin}} \frac{1}{2} ||w||^2, \tag{8.50}$$

where $x_i$ is the training sample with target value $y_i$. The inner product plus intercept $\langle w, x_i \rangle + b$ is the prediction for that sample, and $\epsilon$ is a free parameter that serves as a threshold: all predictions have to be within an $\epsilon$ range of the true predictions. Slack variables are usually added into the above to allow for errors and to allow approximation in the case the above problem is infeasible. Moreover, the term $||w||^2$ can be weighted in different ways using kernel reasonings to exploit some a priori knowledge of the curve to be learned.

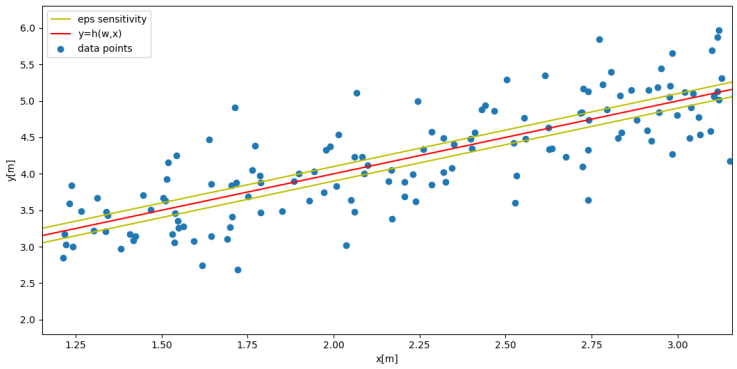


Figure 8.9: SVM regression's interpretation.

## 8.5 Control background

### 8.5.1 Reachability

Reachability. The reachability problem is to "find the set of all the final states $\mathbf{x}(t_1)$ reachable starting from a given initial state $\mathbf{x}(t_0)$ ": - A state $\mathbf{x}(t_1)$ of a dynamic system is reachable from the state $\mathbf{x}(t_0)$ in the time interval $[t_0, t_1]$ if it exists an input function $\mathbf{u}(\cdot) \in \mathcal{U}$ such that $\mathbf{x}(t_1) = \psi(t_0, t_1, \mathbf{x}(t_0), \mathbf{u}(\cdot))$. - Let $\mathcal{X}^+(t_0, t_1, \mathbf{x}(t_0))$ denote the "set of all the final states $\mathbf{x}(t_1)$ reachable at time $t_1$ starting from the initial state $\mathrm{x}(t_0)$".

Let us consider the following discrete time-invariant linear system:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k)$$

Reachability - The set $\mathcal{X}^+(k)$ of all the states reachable from the origin in $k$ steps is equal to the set of all the states $\mathbf{x}(k)$ obtained starting from the initial condition $\mathbf{x}(0) = 0$ and considering only the forced evolution of the system:

$$\mathbf{x}(k) = \sum_{j=0}^{k-1} \mathbf{A}^{(k=-\bar{j}-1)}\mathbf{B}\mathbf{u}(j) = \left[\mathbf{B}\mathbf{A}\mathbf{B}\ldots\mathbf{A}^{k-1}\mathbf{B}\right] \begin{bmatrix} \mathbf{u}(k-1) \\ \mathbf{u}(k-2) \\ \vdots \\ \mathbf{u}(0) \end{bmatrix}$$

and varying the input $\mathbf{u}(0), \mathbf{u}(1), \ldots, \mathbf{u}(k-1)$ in all the possible ways. -

Definition. Reachability matrix in $k$ steps:

$$\mathcal{R}^+(k) \triangleq \left[\mathbf{BAB}\ldots\mathbf{A}^{k-1}\mathbf{B}\right]$$

- The set $\mathcal{X}^+(k)$ of all the states reachable from the origin in $k$ steps is a vectorial space which is equal to the image of matrix $\mathcal{R}^+(k)$ :

$$\mathcal{X}^+(k) = \mathrm{Im}\left[\mathcal{R}^+(k)\right]$$

- The subspaces $\mathcal{X}^+(k)$ reachable in $1, 2, \ldots, k$ steps satisfy the following chain of inclusions ( $n$ is the dimension of the state space):

$$\mathcal{X}^+(1) \subseteq \mathcal{X}^+(2) \subseteq \ldots\ldots \subseteq \mathcal{X}^+(n) = \mathcal{X}^+(n+1) = \ldots$$

- The maximum reachable subspace $\mathcal{X}^+(n)$ is obtained, at the most, in $n$ steps.

- Definition. Reachability matrix of the system:

$$\mathcal{R}^+ \triangleq \mathcal{R}^+(k)\big|_{k=n} = \mathcal{R}^+(n) = \left[\mathbf{BAB}\ldots\mathbf{A}^{n-1}\mathbf{B}\right]$$

- The subspace $\mathcal{X}^+$of all the state reachable from the origin in a time interval however long is equal to the image of matrix $\mathcal{R}^+$:

$$\mathcal{X}^+ = \mathrm{Im}\left[\mathbf{BAB}\ldots\mathbf{A}^{n-1}\mathbf{B}\right] = \mathrm{Im}\,\mathcal{R}^+$$

- Definition. A system is reachable if the subspace $\mathcal{X}^+$of all the reachable states from the origin is equal to the whole state space $\mathbf{X}$ :

$$\mathcal{X}^+ = \mathbf{X}$$

- Necessary and sufficient condition for a system to be reachable is:

$$\mathrm{rank}\left(\mathcal{R}^+\right) = n$$

- For discrete, time-invariant linear systems the set $\mathcal{X}^+(k, \mathbf{x}_0)$ has the structure of a "linear variety":

$$\mathcal{X}^+(k, \mathbf{x}_0) = \mathbf{A}^k\mathbf{x}_0 + \mathrm{Im}\,\mathcal{R}^+(k)$$

## 8.6 Probability Background

### 8.6.1 Conditional probability

The conditional probability mass function $p_{X|Y}(x \mid y)$ of $X$ given $Y = y$ is defined by

$$p_{X|Y}(x \mid y) = \frac{\Pr\{X = x \text{ and } Y = y\}}{\Pr\{Y = y\}} \quad \text{if } \Pr\{Y = y\} > 0,$$

and is not defined, or is assigned an arbitrary value, whenever $\Pr\{Y = y\} = 0$. In terms of the joint and marginal probability mass functions $p_{XY}(x, y)$ and $p_Y(y) = \Sigma_x p_{XY}(x, y)$, respectively, the definition is:

$$p_{X|Y}(x \mid y) = \frac{p_{XY}(x, y)}{p_Y(y)} \quad \text{if } p_Y(y) > 0; \quad x, y = 0, 1, \ldots \tag{8.51}$$

Observe that $p_{X|Y}(x \mid y)$ is a probability mass function in $x$ for each fixed $y$, i.e., $p_{X|Y}(x \mid y) \geq 0$ and $\Sigma_\xi p_{X|Y}(\xi \mid y) = 1$, for all $x, y$. The law of total probability takes the form:

$$\Pr\{X = x\} = \sum_{y=0}^{\infty} p_{X|Y}(x \mid y) p_Y(y) \tag{8.52}$$

Notice in 8.52 that the points $y$ where $p_{X|Y}(x \mid y)$ is not defined are exactly those values for which $p_Y(y) = 0$, and hence, do not affect the computation.

### 8.6.2 Gaussian distribution

In statistics, a normal distribution or Gaussian distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x - \mu}{\sigma}\right)^2\right) \tag{8.53}$$

The parameter $\mu$ is the mean or expectation of the distribution (and also its median and mode), while the parameter $\sigma$ is its standard deviation. The variance of the distribution is $\sigma^2$. A random variable with a Gaussian distribution is said to be normally distributed, and is called a normal deviate. The Standard Normal distribution is a particular type of Normal distribution

with parameters $\mu = 0$ and $\sigma^2 = 1$.

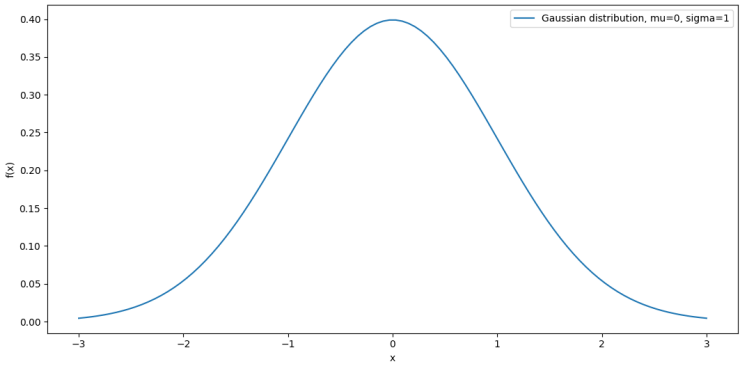


Figure 8.10: Plot of 8.53 with parameters of the Standard Normal distribution.

Normal distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known.
Their importance is partly due to the **central limit theorem**. It states that, under some conditions, the average of many samples (observations) of a random variable with finite mean and variance is itself a random variable—whose distribution converges to a normal distribution as the number of samples increases. Therefore, physical quantities that are expected to be the sum of many independent processes, such as measurement errors, often have distributions that are nearly normal.
Moreover, Gaussian distributions have some unique properties that are valuable in analytic studies. For instance, any linear combination of a fixed collection of normal deviates is a normal deviate. Many results and methods, such as propagation of uncertainty and least squares parameter fitting, can be derived analytically in explicit form when the relevant variables are normally distributed.

### 8.6.3 Bernoulli distribution

The Bernoulli distribution, named after Swiss mathematician Jacob Bernoulli, is the discrete probability distribution of a random variable which takes the value $1$ with probability $p$ and the value $0$ with probability $q = 1 - p$. Less formally, it can be thought of as a model for the set of possible outcomes of any

single experiment that asks a 0-1 question. Such questions lead to outcomes that are boolean-valued: a single bit whose value is success/yes/true/one with probability p and failure/no/false/zero with probability q.
Its basic parameters are:

$$0 \leq \ p \leq 1, \text{ and } q = 1 - p.$$

The expected value of a Bernoulli random variable $X$ is

$$\mathrm{E}[X] = p$$

This is due to the fact that for a Bernoulli distributed random variable $X$ with $\Pr(X = 1) = p$ and $\Pr(X = 0) = q$ we find

$$\mathrm{E}[X] = \Pr(X = 1) \cdot 1 + \Pr(X = 0) \cdot 0 = p \cdot 1 + q \cdot 0 = p.$$

The variance of a Bernoulli distributed $X$ is

$$\mathrm{Var}[X] = pq = p(1 - p).$$

In fact,

$$\mathrm{E}\left[X^2\right] = \Pr(X = 1) \cdot 1^2 + \Pr(X = 0) \cdot 0^2 = p \cdot 1^2 + q \cdot 0^2 = p = \mathrm{E}[X],$$

and from this follows that:

$$\mathrm{Var}[X] = \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2 = \mathrm{E}[X] - \mathrm{E}[X]^2 = p - p^2 = p(1 - p) = pq.$$

with this result it is easy to prove that, for any Bernoulli distribution, its variance will have a value inside $[0, 1/4]$.
If $X_1, \ldots, X_n$ are independent, identically distributed (i.i.d.) random variables, all Bernoulli trials with success probability $p$, then their sum is distributed according to a binomial distribution with parameters $n$ and $p$ :

$$\sum_{k=1}^{n} X_k \sim \mathrm{B}(n, p) \text{ (binomial distribution).}$$

The Bernoulli distribution is simply $\mathrm{B}(1, p)$, also written as $\mathrm{Bernoulli}(p)$.

### 8.6.4 Binomial distribution

The binomial distribution with parameters $n$ and $p$ is the discrete probability distribution of the number of successes in a sequence of $n$ independent Benrnoulli experiments, also called Bernoulli trials or Bernoulli experiments. For $n = 1$, it coincides with the Bernoulli distribution.
In general, if the random variable $X$ follows the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$, we write $X \sim \mathrm{B}(n, p)$. The probability of getting exactly $k$ successes in $n$ independent Bernoulli trials is given by the probability mass function:

$$f(k, n, p) = \Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, \ldots, n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is the binomial coefficient, hence the name of the distribution. The formula can be understood as follows: $k$ successes occur with probability $p^k$ and $n - k$ failures occur with probability $(1-p)^{n-k}$. However, the $k$ successes can occur anywhere among the $n$ trials, and there are $\binom{n}{k}$ different ways of distributing $k$ successes in a sequence of $n$ trials.
For $k > n/2$, the probability can be calculated by its complement as

$$f(k, n, p) = f(n - k, n, 1 - p).$$

Looking at the expression $f(k, n, p)$ as a function of $k$, there is a $k$ value that maximizes it. This $k$ value can be found by calculating

$$\frac{f(k+1, n, p)}{f(k, n, p)} = \frac{(n-k)p}{(k+1)(1-p)}$$

and comparing it to 1 . There is always an integer $M$ that satisfies [2]

$$(n+1)p - 1 \leq M < (n+1)p.$$

$f(k, n, p)$ is monotone increasing for $k < M$ and monotone decreasing for $k > M$, with the exception of the case where $(n + 1)p$ is an integer. In this case, there are two values for which $f$ is maximal: $(n + 1)p$ and $(n + 1)p - 1$. $M$ is the most probable outcome (that is, the most likely, although this can still be unlikely overall) of the Bernoulli trials and is called the mode.
Example
Suppose a biased coin comes up heads with probability 0.3 when tossed. The probability of seeing exactly 4 heads in 6 tosses is

$$f(4, 6, 0.3) = \binom{6}{4} 0.3^4(1 - 0.3)^{6-4} = 0.059535.$$

If $X \sim B(n, p)$, that is, $X$ is a binomially distributed random variable, $n$ being the total number of experiments and $p$ the probability of each experiment yielding a successful result, then the expected value of $X$ is: [5]

$$\mathrm{E}[X] = np.$$

This follows from the linearity of the expected value along with the fact that $X$ is the sum of $n$ identical Bernoulli random variables, each with expected value $p$. In other words, if $X_1, \ldots, X_n$ are identical (and independent) Bernoulli random variables with parameter $p$, then $X = X_1 + \cdots + X_n$ and

$$\mathrm{E}[X] = \mathrm{E}[X_1 + \cdots + X_n] = \mathrm{E}[X_1] + \cdots + \mathrm{E}[X_n] = p + \cdots + p = np.$$

The variance is:

$$\mathrm{Var}(X) = npq = np(1 - p).$$

This, similarly, follows from the fact that the variance of a sum of independent random variables is the sum of the variances.

### 8.6.5 Monte Carlo estimation

Monte Carlo methods can be used to solve any problem having a probabilistic interpretation. By the law of large numbers, integrals described by the expected value of some random variable can be approximated by taking the empirical mean (a.k.a. the 'sample mean') of independent samples of the variable.

By definition, Monte Carlo is the art of approximating an expectation by the sample mean of a function of simulated random variables. This definition is broad enough to cover everything that has been called Monte Carlo, and yet makes clear its essence in very familiar terms: Monte Carlo is about invoking laws of large numbers to approximate expectations.

In more mathematical terms: Consider a (possibly multidimensional) random variable $X$ having probability mass function or probability density function $f_X(x)$ which is greater than zero on a set of values $\mathcal{X}$. Then the expected value of a function $g$ of $X$ is

$$\mathbb{E}(g(X)) = \sum_{x \in \mathcal{X}} g(x) f_X(x)$$

if $X$ is discrete, and

$$\mathbb{E}(g(X)) = \int_{x \in \mathcal{X}} g(x) f_X(x) dx$$

if $X$ is continuous. Now, if we were to take an $n$-sample of $X$ 's, $(x_1, \ldots, x_n)$, and we computed the mean of $g(x)$ over the sample, then we would have the Monte Carlo estimate

$$\tilde{g}_n(x) = \frac{1}{n} \sum_{i=1}^{n} g\left(x_i\right)$$

of $\mathbb{E}(g(X))$. we could, alternatively, speak of the random variable

$$\tilde{g}_n(X) = \frac{1}{n} \sum_{i=1}^{n} g(X)$$

which is called the Monte Carlo estimator of $\mathbb{E}(g(X))$. If $\mathbb{E}(g(X))$, exists, then the weak law of large numbers tells us that for any arbitrarily small $\epsilon$

$$\lim_{n \to \infty} P\left(\left|\tilde{g}_n(X) - \mathbb{E}(g(X))\right| \geq \epsilon\right) = 0.$$

This tells us that as $n$ gets large, then there is small probability that $\widetilde{g_n}(X)$ deviates much from $\mathbb{E}(g(X))$. For our purposes, the strong law of large numbers says much the same thing - the important part being that so long as $n$ is large enough, $\widetilde{g_n}(x)$ arising from a Monte Carlo experiment shall be close to $\mathbb{E}(g(X))$, as desired. One other thing to note at this point is that $\tilde{g}_n(X)$ is

unbiased for $\mathbb{E}(g(X))$ :

$$\mathbb{E}\left(\tilde{g}_n(X)\right) = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} g\left(X_i\right)\right) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left(g\left(X_i\right)\right) = \mathbb{E}(g(X)).$$

In practice, many quantities of interest may be cast as expectations. Most importantly for applications in statistical genetics, it is possible to express all probabilities, integrals, and summations as expectations.

### 8.6.6 Markov processes

A Markov process (see [8]) $\{X_t\}$ is a stochastic process with the property that, given the value of $X_t$, the values of $X_s$ for $s > t$ are not influenced by the values of $X_u$ for $u < t$. In words, the probability of any particular future behavior of the process, when its current state is known exactly, is not altered by additional knowledge concerning its past behavior. A discrete-time Markov chain is a Markov process whose state space is a finite or countable set, and whose (time) index set is $T = (0, 1, 2, \ldots)$. In formal terms, the Markov property is that

$$\begin{aligned}\Pr\{X_{n+1} &= j \mid X_0 = i_0, \ldots, X_{n-1} = i_{n-1}, X_n = i\} \\ &= \Pr\{X_{n+1} = j \mid X_n = i\}\end{aligned} \tag{8.54}$$

for all time points $n$ and all states $i_0, \ldots, i_{n-1}, i, j$. It is frequently convenient to label the state space of the Markov chain by the non negative integers $\{0, 1, 2, \ldots\}$, which we will do unless the contrary is explicitly stated, and it is customary to speak of $X_n$ as being in state $i$ if $X_n = i$.
The probability of $X_{n+1}$ being in state $j$ given that $X_n$ is in state $i$ is called the one-step transition probability and is denoted by $P_{ij}^{n,n+1}$. That is,

$$P_{ij}^{n,n+1} = \Pr\{X_{n+1} = j \mid X_n = i\} \tag{8.55}$$

The notation emphasizes that in general the transition probabilities are functions not only of the initial and final states but also of the time of transition as well. when the one-step transition probabilities are independent of the time variable $n$, we say that the Markov chain has stationary transition probabilities. Since the vast majority of Markov chains that we shall encounter have stationary transition probabilities, we limit our discussion to this case. Then,

$P_{ij}^{n,n+1} = P_{ij}$ is independent of $n$, and $P_{ij}$ is the conditional probability that the state value undergoes a transition from $i$ to $j$ in one trial. It is customary to arrange these numbers $P_{ij}$ in a matrix, in the infinite square array:

$$\mathbf{P} = \left\| \begin{array}{ccccc} P_{00} & P_{01} & P_{02} & P_{03} & \cdots \\ P_{10} & P_{11} & P_{12} & P_{13} & \cdots \\ P_{20} & P_{21} & P_{22} & P_{23} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \\ P_{i0} & P_{i1} & P_{i2} & P_{i3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{array} \right\|,$$

and refer to $\mathbf{P} = \left\| P_{ij} \right\|$ as the Markov matrix or transition probability matrix of the process.

The $i$-th row of $\mathbf{P}$, for $i = 0, 1, \ldots$, is the probability distribution of the values of $X_{n+1}$ under the condition that $X_n = i$. If the number of states is finite, then $\mathbf{P}$ is a finite square matrix whose order (the number of rows) is equal to the number of states. Clearly, the quantities $P_{ij}$ satisfy the conditions:

$$P_{ij} \geq 0 \quad \text{for } i, j = 0, 1, 2, \ldots \tag{8.56}$$

$$\sum_{j=0}^{\infty} P_{ij} = 1 \quad \text{for } i = 0, 1, 2, \ldots \tag{8.57}$$

The condition 8.57 merely expresses the fact that some transition occurs at each trial. (For convenience, one says that a transition has occurred even if the state remains unchanged.)

A Markov process is completely defined once its transition probability matrix and initial state $X_0$ (or, more generally, the probability distribution of $X_0$ ) are specified. we shall now prove this fact. Let $\Pr\{X_0 = i\} = p_i$. It is enough to show how to compute the quantities

$$\Pr\{X_0 = i_0, X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n\} \tag{8.58}$$

since any probability involving $X_{j_1}, \ldots, X_{j_k}$, for $j_1 < \cdots < j_k$, can be obtained, according to the axiom of total probability, by summing terms of the form 8.58.

By the definition of conditional probabilities, we obtain:

$$\begin{aligned}&\Pr\{X_0 = i_0, X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n\} \\ &\quad = \Pr\{X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\} \\ &\quad\quad \times \Pr\{X_n = i_n \mid X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\}.\end{aligned} \tag{8.59}$$

Now, by the definition of a Markov process,

$$\begin{aligned}&\Pr\{X_n = i_n \mid X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\} \\ &\quad = \Pr\{X_n = i_n \mid X_{n-1} = i_{n-1}\} = P_{i_{n-1}, i_n}\end{aligned} \tag{8.60}$$

Substituting 8.60 into 8.59 gives:

$$\begin{aligned}&\Pr\{X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n\} \\ &\quad = \Pr\{X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}\} P_{i_{n-1}, i_n}.\end{aligned}$$

Then, upon repeating the argument $n - 1$ additional times, 8.58 becomes

$$\begin{aligned}&\Pr\{X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n\} \\ &\quad = p_{i_0} P_{i_0, i_1} \cdots P_{i_{n-2}, i_{n-1}} P_{i_{n-1}, i_n}.\end{aligned} \tag{8.61}$$

This shows that all finite-dimensional probabilities are specified once the transition probabilities and initial distribution are given, and in this sense, the process is defined by these quantities.

Related computations show that 8.54 is equivalent to the Markov property in the form

$$\begin{aligned}&\Pr\{X_{n+1} = j_1, \ldots, X_{n+m} = j_m \mid X_0 = i_0, \ldots, X_n = i_n\} \\ &\quad = \Pr\{X_{n+1} = j_1, \ldots, X_{n+m} = j_m \mid X_n = i_n\}\end{aligned} \tag{8.62}$$

for all time points $n, m$ and all states $i_0, \ldots, i_n, j_1, \ldots, j_m$. In other words, once 8.62 is established for the value $m = 1$, it holds for all $m \geq 1$ as well.

**Example**: a Markov chain $X_0, X_1, X_2, \ldots$ has the following transition probability matrix:

$$\mathbf{P} = \begin{Vmatrix} 0.1 & 0.1 & 0.8 \\ 0.2 & 0.2 & 0.6 \\ 0.3 & 0.3 & 0.4 \end{Vmatrix}. \tag{8.63}$$

The states are $0, 1, 2$. Determine the conditional probability:

$$\Pr\{X_1 = 1, X_2 = 1 \mid X_0 = 0\}$$

The fact that we are conditioning the two states $X_1$ and $X_2$ on the state $X_0$ and we know that $X_0 = 0$ allows us to solve the problem as follows:

$$\Pr\{X_1 = 1, X_2 = 1 \mid X_0 = 0\} = P_{01}P_{11} = 0.1 \times 0.2 = 0.02.$$

#### 8.6.6.0.1 Markov chains

A Markov chain or Markov process is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.
In mathematical terms, let $\{X_0, X_1, \ldots\}$ be a sequence of random variables. Then, $\{X_0, X_1, \ldots\}$ is a Markov chain if it satisfies the Markov property 8.54 for all $t = 1, 2, 3, \ldots$ and for all states $s_0, s_1, s_2, \ldots$.
A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain Discrete-time Markov chain (DTMC). A continuous-time process is called a continuous-time Markov chain Continuous-time Markov chain (CTMC).
A Markov chain is completely defined by its one-step transition probability matrix and the specification of a probability distribution on the state of the process at time 0 . The analysis of a Markov chain concerns mainly the calculation of the probabilities of the possible realizations of the process. Central in these calculations are the $n$-step transition probability matrices $\mathbf{P}^{(n)} = \left\| P_{ij}^{(n)} \right\|$. Here, $P_{ij}^{(n)}$ denotes the probability that the process goes from state $i$ to state $j$ in $n$ transitions. Formally,

$$P_{ij}^{(n)} = \Pr\{X_{m+n} = j \mid X_m = i\}. \tag{8.64}$$

Observe that we are dealing only with temporally homogeneous processes having stationary transition probabilities, since otherwise the left side of 8.64 would also depend on $m$.
The Markov property allows us to express 8.64 in terms of $\left\| P_{ij} \right\|$ as stated in the following theorem.

**Theorem**.
The n-step transition probabilities of a Markov chain satisfy:

$$P_{ij}^{(n)} = \sum_{k=0}^{\infty} P_{ik} P_{kj}^{(n-1)}, \tag{8.65}$$

where we define

$$P_{ij}^{(0)} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}. \tag{8.66}$$

From the theory of matrices, we recognize the relation 8.65 as the formula for matrix multiplication so that $\mathbf{P}^{(n)} = \mathbf{P} \times \mathbf{P}^{(n-1)}$. By iterating this formula, we obtain

$$\mathbf{P}^{(n)} = \underbrace{\mathbf{P} \times \mathbf{P} \times \cdots \times \mathbf{P}}_{n \text{ factors}} = \mathbf{P}^n. \tag{8.67}$$

In other words, the $n$-step transition probabilities $P_{ij}^{(n)}$ are the entries in the matrix $\mathbf{P}^n$, the $n$-th power of $\mathbf{P}$.

*Proof.* The proof proceeds via a first step analysis, a breaking down, or analysis, of the possible transitions on the first step, followed by an application of the Markov property. The event of going from state $i$ to state $j$ in $n$ transitions can be realized in the mutually exclusive ways of going to some intermediate state $k (k = 0, 1, \ldots)$ in the first transition, and then going from state $k$ to state $j$ in the remaining $(n-1)$ transitions. Because of the Markov property, the probability of the second transition is $P_{kj}^{(n-1)}$ and that of the first is clearly $P_{ik}$. If we use the law of total probability, then 8.65 follows. The steps are

$$\begin{aligned} P_{ij}^{(n)} &= \Pr\{X_n = j \mid X_0 = i\} = \sum_{k=0}^{\infty} \Pr\{X_n = j, X_1 = k \mid X_0 = i\} \\ &= \sum_{k=0}^{\infty} \Pr\{X_1 = k \mid X_0 = i\} \Pr\{X_n = j \mid X_0 = i, X_1 = k\} \\ &= \sum_{k=0}^{\infty} P_{ik} P_{kj}^{(n-1)}. \end{aligned} \tag{8.68}$$

If the probability of the process initially being in state $j$ is $p_j$, i.e., the distribution law of $X_0$ is $\Pr\{X_0 = j\} = p_j$, then the probability of the process

being in state $k$ at time $n$ is

$$p_k^{(n)} = \sum_{j=0}^{\infty} p_j P_{jk}^{(n)} = \Pr\{X_n = k\}. \tag{8.69}$$

□

**Example**: a Markov chain $\{X_n\}$ om the states $0, 1, 2$ has the following transition probability matrix:

$$\mathbf{P} = \begin{Vmatrix} 0.1 & 0.2 & 0.7 \\ 0.2 & 0.2 & 0.6 \\ 0.6 & 0.1 & 0.3 \end{Vmatrix}. \tag{8.70}$$

The two-step transition matrix $P^2$ is:

$$\mathbf{P}^2 = \begin{Vmatrix} 0.1 & 0.2 & 0.7 \\ 0.2 & 0.2 & 0.6 \\ 0.6 & 0.1 & 0.3 \end{Vmatrix} \times \begin{Vmatrix} 0.1 & 0.2 & 0.7 \\ 0.2 & 0.2 & 0.6 \\ 0.6 & 0.1 & 0.3 \end{Vmatrix} = \begin{Vmatrix} 0.47 & 0.13 & 0.4 \\ 0.42 & 0.14 & 0.44 \\ 0.26 & 0.17 & 0.57 \end{Vmatrix}$$

The probability of

$$\Pr\{X_3 = 1 \mid X_1 = 0\}$$

is simply equal to:

$$\Pr\{X_3 = 1 \mid X_1 = 0\} = \mathbf{P}_{01}^2 = 0.13.$$

#### 8.6.6.0.2 Absorbing Markov chains

An absorbing Markov chain is a Markov chain in which every state can reach an absorbing state. An absorbing state is a state that, once entered, cannot be left.
**Definition**:A Markov chain is an absorbing chain if:

1. there is at least one absorbing state and;
2. it is possible to go from any state to at least one absorbing state in a finite number of steps.

In an absorbing Markov chain, a state that is not absorbing is called transient.
Let's analyze the canonical form of an absorbing Markov chain.
Let an absorbing Markov chain with transition matrix $P$ have $t$ transient states and $r$ absorbing states. Unlike a typical transition matrix, the rows of $P$

represent sources, while columns represent destinations. Then

$$P = \begin{bmatrix} Q & R \\ \mathbf{0} & I_r \end{bmatrix}$$

where $Q$ is a $t$-by-t matrix, $R$ is a nonzero $t$-by- $r$ matrix, 0 is an $r$-by-t zero matrix, and $I_r$ is the $r$-by- $r$ identity matrix. Thus, $Q$ describes the probability of transitioning from some transient state to another while $R$ describes the probability of transitioning from some transient state to some absorbing state. The probability of transitioning from $i$ to $j$ in exactly $k$ steps is the $(i, j)$-entry of $P^k$, further computed below. when considering only transient states, the probability found in the upper left of $P^k$, the $(i, j)$-entry of $Q^k$.

**Expected number of visits to a transient state**

A basic property about an absorbing Markov chain is the expected number of visits to a transient state $j$ starting from a transient state $i$ (before being absorbed). This can be established to be given by the $(i, j)$ entry of so called fundamental matrix **N**, obtained by summing $Q^k$ for all $k$ (from 0 to $\infty$ ). It can be proven that:

$$N := \sum_{k=0}^{\infty} Q^k = (I_t - Q)^{-1}$$

where $I_t$ is the $t$-by-t identity matrix. The computation of this formula is the matrix equivalent of the geometric series of scalars, $\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}$.
with the matrix $N$ in hand, also other properties of the Markov chain are easy to obtain.

**Expected number of steps before being absorbed**

The expected number of steps before being absorbed in any absorbing state, when starting in transient state $i$ can be computed via a sum over transient states.
The value is given by the $i - th$ entry of the vector:

$$\mathbf{t} := N\mathbf{1},$$

where **1** is a length-t column vector whose entries are all 1.

**Absorbing probabilities**

By induction,

$$P^k = \begin{bmatrix} Q^k & \left(1 - Q^k\right) NR \\ \mathbf{0} & I_r \end{bmatrix}$$

The probability of eventually being absorbed in the absorbing state $j$ when starting from transient state $i$ is given by the $(i, j)$-entry of the matrix:

$$B := NR.$$

The number of columns of this matrix equals the number of absorbing states $r$. An approximation of those probabilities can also be obtained directly from the $(i, j)$-entry of $P^k$ for a large enough value of $k$, when $i$ is the index of a transient, and $j$ the index of an absorbing state. This is because:

$$\left(\lim_{k\to\infty} P^k\right)_{i,t+j} = B_{i,j}.$$

**Transient visiting probabilities**
The probability of visiting transient state $j$ when starting at a transient state $i$ is the $(i, j)$-entry of the matrix:

$$H := (N - I_t)\left(N_{\mathrm{dg}}\right)^{-1},$$

where $N_{\mathrm{dg}}$ is the diagonal matrix with the same diagonal as $N$.
**Variance on number of transient visits**
The variance on the number of visits to a transient state $j$ with starting at a transient state $i$ (before being absorbed) is the $(i, j)$-entry of the matrix:

$$N_2 := N\left(2N_{\mathrm{dg}} - I_t\right) - N_{\mathrm{sq}}$$

where $N_{\mathrm{sq}}$ is the Hadamard product of $N$ with itself (i.e. each entry of $N$ is squared).
**Variance on number of steps**
The variance on the number of steps before being absorbed when starting in transient state $i$ is the ith entry of the vector

$$(2N - I_t)\,\mathbf{t} - \mathbf{t}_{\mathrm{sq}},$$

where $\mathbf{t}_{\mathrm{sq}}$ is the Hadamard product of $\mathbf{t}$ with itself (i.e., as with $N_{\mathrm{sq}}$, each entry of $\mathbf{t}$ is squared).

## 8.7 Moving average filter

In statistics, a moving average (see [2]) (rolling average or running average) is an average calculation to analyze data points by creating a series of averages of different selections of the full data set. It is also called a moving mean or rolling mean and is a type of finite impulse response filter.
Given a series of numbers and a fixed subset size, the first element of the moving average is obtained by taking the average of the initial fixed subset of the number series. Then the subset is modified by "shifting forward"; that is, excluding the first number of the series and including the next value in the subset.

A moving average is commonly used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. The threshold between short-term and long-term depends on the application, and the parameters of the moving average will be set accordingly. It is also used in economics to examine gross domestic product, employment or other macroeconomic time series. Mathematically, a moving average is a type of convolution and so it can be viewed as an example of a low-pass filter used in signal processing.
Viewed simplistically it can be regarded as smoothing the data.
In financial applications a simple moving average (SMA) is the unweighted mean of the previous $k$ data-points. However, in science and engineering, the mean is normally taken from an equal number of data on either side of a central value. This ensures that variations in the mean are aligned with the variations in the data rather than being shifted in time. An example of a simple equally weighted running mean is the weighted mean over the last $w$ entries of a data-set containing $n$ entries. Let those data-points be $P_1, P_2, P_3, \ldots, P_n$.

Let's denote the mean over the last $w$ data points as $SMA_w$, defined as follows:

$$\begin{aligned} SMA_w &= \frac{P_{n-w+1} + P_{n-w+2} + \cdots + P_n}{w} \\ &= \frac{1}{w} \sum_{k=n-w+1}^{n} P_k \end{aligned}$$

To calculate the next mean $SMA_{w,next}$ with same sampling width w (also called window), the new range from $n - w + 2$ to $n + 1$ is considered. A new value $P_{n+1}$ comes into the sum and the oldest value $P_{n-w+1}$ is dropped out. This allows to simplify the calculations, in a recursive way:

$$\begin{aligned} SMA_{w,\,\text{next}} &= \frac{1}{w} \sum_{k=n-w+2}^{n+1} p_k \\ &= \frac{1}{w} (\underbrace{p_{n-w+2} + p_{n-w+3} + \cdots + p_n + p_{n+1}}_{\sum_{k=n-w+2}^{n+1} p_k} + \underbrace{p_{n-w+1} - p_{n-w+1}}_{=0}) \\ &= \underbrace{\frac{1}{w} (p_{n-w+1} + p_{n-w+2} + \cdots + p_n)}_{=SMA_{w,\,\text{prev}}} - \frac{p_{n-w+1}}{w} + \frac{p_{n+1}}{w} \\ &= SMA_{w,\,\text{prev}} + \frac{1}{w} (p_{n+1} - p_{n-w+1}) \end{aligned}$$

This simplification is very important in terms of computational complexity as it speeds up the algorithm from an initial $O(nw)$, that is almost quadratic, to a way better $O(n + w)$, that is basically linear.

## 8.8 Low pass filters

A low-pass filter is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The exact frequency response of the filter depends on the filter design.
An ideal low-pass filter completely eliminates all frequencies above the cutoff frequency while passing those below unchanged; its frequency response is a rectangular function and is a brick-wall filter. The transition region present in

practical filters does not exist in an ideal filter. An ideal low-pass filter can be realized mathematically (theoretically) by multiplying a signal by the rectangular function in the frequency domain or, equivalently, convolution with its impulse response, a sinc function, in the time domain.
However, the ideal filter is impossible to realize without also having signals of infinite extent in time, and so generally needs to be approximated for real ongoing signals, because the sinc function's support region extends to all past and future times. The filter would therefore need to have infinite delay, or knowledge of the infinite future and past, to perform the convolution. It is effectively realizable for pre-recorded digital signals by assuming extensions of zero into the past and future, or, more typically, by making the signal repetitive and using Fourier analysis.
Real filters for real-time applications approximate the ideal filter by truncating and windowing the infinite impulse response to make a finite impulse response; applying that filter requires delaying the signal for a moderate period of time, allowing the computation to "see" a little bit into the future. This delay is manifested as phase shift. Greater accuracy in approximation requires a longer delay.
Truncating an ideal low-pass filter result in ringing artifacts via the Gibbs phenomenon, which can be reduced or worsened by the choice of windowing function. Design and choice of real filters involves understanding and minimizing these artifacts. For example, simple truncation of the sync function will create severe ringing artifacts, which can be reduced using window functions that drop off more smoothly at the edges.

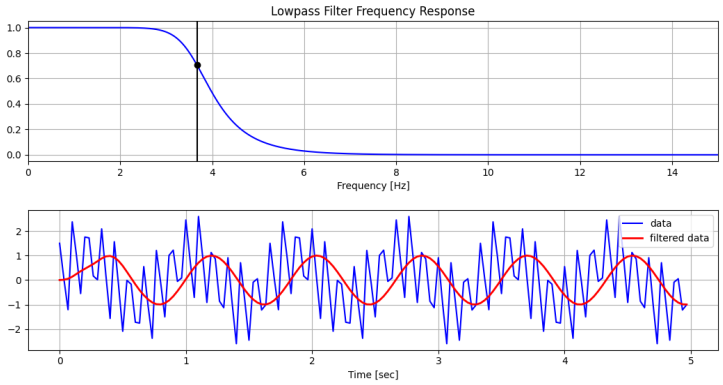


Figure 8.11: Example of a filtered processed signal.

### 8.8.1 Band-pass filters

The four common filters (see [12]):

- low-pass filter, passes signals with a frequency lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency;
- high-pass filter, passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency;
- a band-pass filter can be formed by cascading a high-pass filter and a low-pass filter;
- a band-reject filter is a parallel combination of low-pass and high-pass filters.

### 8.8.2 Butterworth Filter

The Butterworth filter is a type of signal processing filter designed to have a frequency response that is as flat as possible in the passband.
Hence its one of the most popular low pass filter.
The Nyquist rate or frequency is the minimum rate at which a finite bandwidth signal needs to be sampled to retain all of the information. If a time series is sampled at regular time intervals $\Delta t$, then the Nyquist rate is just $\frac{1}{2\Delta t}$.

## 8.9 Error's propagation

In statistics, propagation of uncertainty (or propagation of error) is the effect of variables' uncertainties, that can be errors or random errors, on the uncertainty of a function based on them.
When the variables are the values of experimental measurements they have uncertainties due to measurement limitations (e.g., instrument precision) which propagate due to the combination of variables in the function.

The general expressions for a scalar-valued function $f$ are:

$$
\begin{aligned}
f &= \sum_{i}^{n} a_i x_i = \mathbf{a}\mathbf{x}, \\
\sigma_f^2 &= \sum_{i}^{n} \sum_{j}^{n} a_i \Sigma_{ij}^{x} a_j = \mathbf{a}\mathbf{\Sigma}^{x}\mathbf{a}^{\mathrm{T}},
\end{aligned}
\tag{8.71}
$$

where $\mathbf{a}$ is a row vector, the unitary element $x_i$ of $\mathbf{x}$ can be seen as the mean of a gaussian random variable, with standard deviation $\sigma_i$. Each covariance term $\sigma_{ij}$ can be expressed in terms of the correlation coefficient $\rho_{ij}$ by $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$, so that an alternative expression for the variance of $f$ is

$$
\sigma_f^2 = \sum_{i}^{n} a_i^2 \sigma_i^2 + \sum_{i}^{n} \sum_{j(j\neq i)}^{n} a_i a_j \rho_{ij} \sigma_i \sigma_j. \tag{8.72}
$$

In the case that the variables in $x$ are uncorrelated, this simplifies further to

$$
\sigma_f^2 = \sum_{i}^{n} a_i^2 \sigma_i^2. \tag{8.73}
$$

In the simple case of identical coefficients and variances, we find

$$
\sigma_f = \sqrt{n}|a|\sigma. \tag{8.74}
$$

For the arithmetic mean, $a = 1/n$, the result is the standard error of the mean:

$$
\sigma_f = \frac{\sigma}{\sqrt{n}}. \tag{8.75}
$$

In the non linear case instead, neglecting correlations or assuming independent variables yields a common formula among engineers and experimental scientists to calculate error propagation, the variance formula:

$$
s_f = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 s_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 s_y^2 + \left(\frac{\partial f}{\partial z}\right)^2 s_z^2 + \cdots}, \tag{8.76}
$$

where $s_f$ represents the standard deviation of the function $f$, $s_x$ represents the standard deviation of $x$, $s_y$ represents the standard deviation of $y$, and so forth. It is important to note that this formula is based on the linear characteristics of

the gradient of $f$ and therefore it is a good estimation for the standard deviation of $f$ as long as $s_x, s_y, s_z, \ldots$ are small enough. Specifically, the linear approximation of $f$ has to be close to $f$ inside a neighbourhood of radius $s_x, s_y, s_z, \ldots$.
For example, any non-linear differentiable function, $f(a, b)$, of two variables, $a$ and $b$, can be expanded as

$$f \approx f^0 + \frac{\partial f}{\partial a}a + \frac{\partial f}{\partial b}b$$

now, taking variance on both sides, and using the formula for variance of a linear combination of variables:

$$\operatorname{Var}(aX + bY) = a^2 \operatorname{Var}(X) + b^2 \operatorname{Var}(Y) + 2ab * \operatorname{Cov}(X, Y)$$

hence:

$$\sigma_f^2 \approx \left|\frac{\partial f}{\partial a}\right|^2 \sigma_a^2 + \left|\frac{\partial f}{\partial b}\right|^2 \sigma_b^2 + 2\frac{\partial f}{\partial a}\frac{\partial f}{\partial b}\sigma_{ab}$$

where $\sigma_f$ is the standard deviation of the function $f$, $\sigma_a$ is the standard deviation of $a$, $\sigma_b$ is the standard deviation of $b$ and $\sigma_{ab} = \sigma_a \sigma_b \rho_{ab}$ is the covariance between $a$ and $b$. See [7] for more details.

## 8.10 Root mean square error

The Root mean square error (RMSE), is a measure of the difference between the true values of a model and the estimated ones of an estimator of such a model. RMSE represents the square root of the second sample moment of the differences between the predicted values and the observed/measured ones. In other words, is the quadratic mean of these differences, that are also called the residuals.
Another interpretation of the RMSE is that it can be seen as the standard deviation of the error.
The lower the value of the RMSE, the better the model is. A perfect model (a hypothetic model that would always predict the exact expected value) would have a RMSE value of $0$.
The Root Mean Squared Error has the advantage of representing the amount of error in the same unit as the predicted column making it easy to interpret. If

you are trying to predict an amount in meters, then the RMSE can be interpreted as the amount of error in meters.

## 8.11 Code snippets

**Algorithm 2:**

```
def find_bounds():
                if g < 2: # called just when M = 1, initialization
                    m_low = (n2[1] - n4[1]) / (n2[0] - n4[0])
                    m_high = (n1[1] - n3[1]) / (n1[0] - n3[0])

                else:
                    if (n2[1] - n4[1]) / (n2[0] - n4[0]) > m_low:
                        m_low = (n2[1] - n4[1]) / (n2[0] - n4[0])
                    if (n1[1] - n3[1]) / (n1[0] - n3[0]) < m_high:
                        m_high = (n1[1] - n3[1]) / (n1[0] - n3[0])
```

Code 8.1: find_bounds function.

```
def find_theta_target():
        if g == 0:
            return -pi/2
        if flag_plus:
            if math.atan(m_high)- delta_theta > math.atan(m_low):
                return math.atan(m_high)- delta_theta
            else:
                return math.atan(m_high)
        else:
            if math.atan(m_low) + delta_theta < math.atan(m_high):
                return math.atan(m_low) + delta_theta
            else:
                return math.atan(m_low)
```

Code 8.2: find_theta_target function.

**Algorithm 3**:

```
def labelling(a):
    # a is a 2D vector
    if np.random.uniform() >= e:
        if -a[1] + m_coeff * a[0] + q_coeff >= 0:
            return +1
```

```
        else:
            return -1
    else:
        if -a[1] + m_coeff * a[0] + q_coeff >= 0:
            return -1
        else:
            return +1
```

Code 8.3: *labelling_with_unif_noise*() function.

```
def find_prec():
    # y[-4] till y[-1] is on the new side
    t=0
    while True:
        if y[-5-t]*y[-1]<0 and y[-6-t]*y[-1]<0 and y[-7-t]*y[-1]<0
    and y[-8-t]*y[-1]<0:
            return [x[-6-t], z[-6-t]]
        t += 1
```

Code 8.4: *find_prec*() function.

```
def update_flag_plus():
    if y[-1]>0 and y[-2]>0 and y[-3]>0 and y[-4]>0:
        return True
    if y[-1]<0 and y[-2]<0 and y[-3]<0 and y[-4]<0:
        return False
```

Code 8.5: *update_flag_plus*() function.

```
def find_bounds():
    if not flag_convergence:
        if g >= 2:
            if (P2[1] - P4[1]) / (P2[0] - P4[0]) > m_low:
                m_low = (P2[1] - P4[1]) / (P2[0] - P4[0])
                flag_improv = True
            if (P1[1] - P3[1]) / (P1[0] - P3[0]) < m_high:
                m_high = (P1[1] - P3[1]) / (P1[0] - P3[0])
                flag_improv = True
        else:
            m_low = (P2[1] - P4[1]) / (P2[0] - P4[0])
            m_high = (P1[1] - P3[1]) / (P1[0] - P3[0])
            if flag_convergence==False and m_high-m_low <= 2*
    delta_theta:
                j_convergence = j
```

```
                flag_convergence = True
```

Code 8.6: *find_bounds( )* function.

```
def find_theta_target():
        if g == 0:
            return -pi/2
        if flag_plus:
            if math.atan(m_high)- delta_theta > math.atan(m_low) and
   not flag_improv:
                return math.atan(m_high)- delta_theta
            else:
                flag_improv = False
                return math.atan(m_high)

        else:
            if math.atan(m_low) + delta_theta < math.atan(m_high) and
    not flag_improv:
                return math.atan(m_low) + delta_theta
            else:
                flag_improv = False
                return math.atan(m_low)
```

Code 8.7: *find_theta_target()* function.

**Algorithm 4**:

```
def find_theta_target():
        if g == 0:
            return -pi/2
        if flag_plus:
            return pi/2
        else:
            return -pi/2
```

Code 8.8: New *find_theta_target()* function.

```
def update_flag_plus():
    if y[-1]>0 and y[-2]>0 and y[-3]> 0 and y[-4]> 0 and y[-5]> 0 and
    y[-6]> 0 and y[-7]> 0 and y[-8]> 0 and y[-9]> 0:
        return True
    if y[-1]<0 and y[-2]<0 and y[-3]< 0 and y[-4]< 0 and y[-5]< 0 and
    y[-6]< 0 and y[-7]< 0 and y[-8]< 0 and y[-9]< 0:
        return False
```

Code 8.9: *update_flag_plus()* function.

```
def labelling(a): #a is a 2D vector
        if -a[1]  -m*a[0] - q + np.random.normal(scale=sigma) > 0:
            return +1
        else:
            return -1
```

Code 8.10: New *labelling*() function.

**Algorithm 5**:

```
def find_point()
    llen=len(doubt_points_x)
        if flag_plus == False:
            if llen %2==0:
                index = llen/2  + (doubt_points_y)-1 #--1 for index 0
                return avg(p[index], p[index+1])
            else:
                index = (llen-1)/2 + (doubt_points_y) -1
                return p[index]
        else:
            if llen %2==0:
                index = llen/2  - (doubt_points_y) -1 #--1 for index
   0
                return avg(p[index], p[index+1])
            else:
                index = (llen-1)/2 - (doubt_points_y) -1
                return p[index]
```

Code 8.11: *find_point*() function.

**Non linear curves**:

```
def find_theta_target():
    if g == 0:
        return -pi/2
    if flag_plus:
        if (P3[0]-P1[0]<0):
            return atan(m_high) + delta_theta + pi
        elif P3[0]-P1[0]>0:
            return atan(m_high) + delta_theta
    else:
        if  P4[0] - P2[0] > 0:
            return atan(m_low) - delta_theta
        else:
            return atan(m_low) - delta_theta + pi
```

Code 8.12: find_theta_target function.

# References


[1] Patrick Billingsley. "Probability and measure. Wiley Series in Probability and Mathematical Statistics". In: *New York: John Wiley & Sons*. 1195, p. 76.

[2] Ya-lun Chou. "Statistical Analysis". In: *Holt International*. 1975.

[3] Joana Fonseca et al. "Algal Bloom Front Tracking Using an Unmanned Surface Vehicle: Numerical Experiments Based on Baltic Sea Data". In: *OCEANS 2021: San Diego–Porto*. IEEE. 2021, pp. 1–7.

[4] Joana Fonseca et al. "Cooperative Circumnavigation for a Mobile Target Using Adaptive Estimation". In: *CONTROLO 2020*. Ed. by José Alexandre Gonçalves, Manuel Braz-César, and João Paulo Coelho. Cham: Springer International Publishing, 2021, pp. 33–48.

[5] B.& Chen& J. L. Hamann. "Data point selection for piecewise linear curve approximation." In: *CLEF (Working Notes)*. 1994, pp. 2302–2318.

[6] W.P.M.H. Heemels, K.H. Johansson, and P. Tabuada. "An introduction to event-triggered and self-triggered control". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. 2012, pp. 3270–3285. DOI: 10.1109/CDC.2012.6425820.

[7] James. Kirchner. "Data Analysis Toolkit 5: Uncertainty Analysis and Error Propagation". In: *Berkeley Seismology Laboratory. University of California.*

[8] D.Baum L.Breuer. "An introduction to queueing theory and matrix-analytic methods". In: Springer.

[9] Don S. Lemons. "An Introduction to Stochastic Processes in Physics". In: *The Johns Hopkins University Press*. 2002, p. 34.

[10] Dilip K. Prasad and Maylor K.H. Leung. “Methods for Ellipse Detection from Edge Maps of Real Images”. In: *Machine Vision*. Ed. by Fabio Solari, Manuela Chessa, and Silvio P. Sabatini. Rijeka: IntechOpen, 2012. Chap. 7. DOI: 10.5772/35150. URL: https://doi.org/10.5772/35150.

[11] Bernhard Schölkopf and Alexander J Smola. “Learning with kernels: support vector machines, regularization, optimization, and beyond”. In: MIT press, 2002.

[12] B. A. Shenoi. “Introduction to Digital Signal Processing and Filter Design”. In: *John Wiley & Sons*. 2005.