# APPLYING MACHINE LEARNING TECHNIQUES FOR ROBOT CONTROL WHILE ENSURING STABILITY

Vaibhav Balkrishna Wanere

AN INDEPENDENT STUDY

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Engineering

2023

Independent Study Advisor

____________________

Nadia Figueroa
Shalini and Rajeev Misra Presidential Assistant Professor
Mechanical Engineering and Applied Mechanics

Graduate Group Chairperson

____________________

Howard H. Hu
Master's Program Chair
Mechanical Engineering and Applied Mechanics

# TABLE OF CONTENTS

# CHAPTER 1

# STUDY PROPOSAL

## 1.1. A Dynamical System approach for robot control

Traditional approaches to robot control rely on offline path planning by modelling the environment and quantifying uncertainty related to it [1]. This requires accurate model of environment and a good model of uncertainty which may even change with time [1]. This study is based on the new approach presented in [1] which uses dynamical system and control theory to model the physics of the robot motion and machine learning techniques to learn the robot control law. Use of dynamical system for robot control allows for:

1. Online Reactivity: Resistance to external perturbations during task execution.

2. Ability to learn control law though human demonstrations.

3. A compliant robot during task execution which ensures human safety.

4. Incremental learning: learning trajectories while retaining the old ones.

## 1.2. Linear Time Invariant Dynamical System

A time invariant 1st order linear dynamical system is given by [1]:

$$\begin{aligned} f &: \mathbb{R}^N \to \mathbb{R}^N, \\ \dot{x} &= f(x), \\ f(x) &= Ax + b, \\ f(x) &\neq 0, \forall x \neq x^*, \\ \dot{x^*} &= f(x^*) = 0, \\ \lim_{x \to \infty} x &= x^*. \end{aligned} \tag{1.1}$$

The above equation expresses the end effector velocity as a function of its position where $x^*$ is the target or attractor. This function can be learned using a suitable machine learning technique given we have sufficient data points pairs of velocity and position ($\dot{x}$, $x$). But the learned function should respect the constraints such as stability of the dynamical system which restricts the use of machine learning techniques off the shelf [1]. This study will explore, analyse and compare the potential machine learning algorithms that can learn the function and respect the stability constraints at the same time. This study aims at surveying the potential machine learning techniques such as Gaussian Mixture Model, Support Vector Machines and Gaussian Process Regression and Neural Networks to learn the robot control law.

The problem of robot control becomes more complex as we move from linear to highly curvilinear trajectories as well as for self intersecting trajectories. I will study the approaches to mitigate this challenges.

# CHAPTER 2

# GAUSSIAN MIXTURE MODEL FOR REGRESSION

## 2.1. Mapping positions to velocities

The governing idea of the approach studied given in [1] is the mapping of robot end effector positions to the end effector linear velocities. Also, given sufficient number of demonstration data points (figure 2.1) that map positions to velocities, one can use a suitable ML algorithm to map the position space to the velocity space and predict velocities at the unknown positions. In other words, one can learn the Dynamical System using a suitable Machine Learning Algorithm.

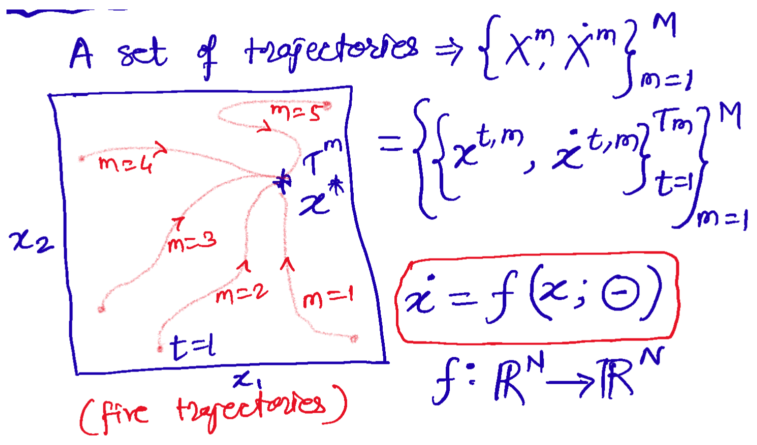
A set of trajectories ⇒ $\{X^m, \dot{X}^m\}_{m=1}^{M}$
$= \left\{\{x^{t,m}, \dot{x}^{t,m}\}_{t=1}^{T_m}\right\}_{m=1}^{M}$
m=5
m=4
$T^m$
$x^*$
m=3
m=2
m=1
t=1
$x_2$
$x_1$
(five trajectories)
$\boxed{\dot{x} = f(x; \Theta)}$
$f: \mathbb{R}^N \rightarrow \mathbb{R}^N$

Figure 2.1: Demonstrated Trajectories in $\mathbb{R}^{\nvDash}$

The obvious choice to do this is to use use regression algorithm. There are various regression algorithms such as linear regression, support vector regression, gaussian mixture regression, gaussian process regression, and even a neural network can be used to learn a regressive function. We are mapping from $f : \mathbb{R}^N \rightarrow \mathbb{R}^N$, but SVR and GPR map an input space from $\mathbb{R}^{\mathbb{N}}$ to $\mathbb{R}$, so while using SVR and GPR we have to map from input space to every element of the output space. In other words, we have to learn $N$ regressive functions, one for each $n$-th output dimension, i.e. $f(x) = [f_1(x), ..., f_n(x), ...f_N(x)]^T$. After studying [1], it was found that SVR, GPR cannot learn a stable dynamical system hence this study is focused on the use of GMM and its constrained version for learning a stable dynamical system.

## 2.2. Gaussian Mixture Model

As the name suggests, the Gaussian Mixture Models employs a mixture of Gaussian distributions and represents multimodal sample distributions. The probability density function of a Gaussian mixture model with $K$ distributions is given as the weighted sum of those distributions as:

$$p(x|\Theta) = \sum_{k=1}^{K} \gamma_k p(x|\mu^k, \Sigma^k) \tag{2.1}$$

where, $p(x|\mu^k, \Sigma^k)$ is the multivariate Gaussian pdf with mean $\mu^k$ and covariance $\Sigma^k$, which is given as:

$$\begin{aligned} p(x|\mu^k, \Sigma^k) &= \mathcal{N}(x|\mu^k, \Sigma^k) \\ &= \frac{1}{\sqrt{(2\pi)^N |\Sigma^k|}} \exp\left\{-\frac{1}{2}(x-\mu^k)^T (\Sigma^k)^{-1} (x-\mu^k)\right\} \end{aligned} \tag{2.2}$$

$\Theta = \{\theta_1, ....\theta_k\}$ is the complete set of parameters $\theta_k = \{\gamma_k, \mu^k, \Sigma^k\}$, where $\gamma_k$ are priors (or mixing weights) of each Gaussian Component, which satisfy the constraint $\sum_{k=1}^{K} \gamma_k = 1$. $\Theta$ can be estimated from training data using either a Maximum Likelihood parameter estimation approach through the iterative Expectation-Maximization (EM) algorithm or using Maximum APosterior (MAP) estimation with fixed values of K [1].

### 2.2.1. Finite Gaussian Mixture Model with EM-based Parameter Estimation

As we have already discussed, the GMM uses a mixture of $K$ Gaussians, thus each data-point $x_i \epsilon \mathbb{R}^{\mathbb{N}}$ is assigned to a cluster $k$ with the clustering assignment indicator variable $Z = \{z_1, ..., z_M\}$, where $i : z_i = k$.

The marginal distribution over $Z$ is defined by the mixing coefficients $\pi_k$, viewed as the prior probability of the cluster assignment indicator variable [1]: $p(z_i = k) = \pi_k$. To be more verbose, $p(z_i = k) = \pi_k$ indicates that a datapoint $x_i$ belongs to a cluster $k$ with the probability $\pi_k$.

The conditional distribution of a sample point $x_i$ given a cluster $k$ is obtained by multivariate normal distribution with mean $\mu^k$ and covariance $\Sigma^k$: $x_i|z_i = k \sim \mathcal{N}(\mu^k, \Sigma^k)$. Each data-point $x_i$ is generated by independently selecting the $k$-th cluster ($zi = k$) according to the mixing coefficients $\pi_k$, and then sampling from that $k$-th distribution, parametrized by $\mu^k$ and $\Sigma^k$.

Finally, the probability density function of the Gaussian Mixture Model is given as:

$$p(x|\Theta, \pi) = \sum_{k=1}^{K} p(z_i = k)p(x|k) = \sum_{k=1}^{K} \pi_k p(x|\mu^k, \Sigma^k) \tag{2.3}$$

with, $0 \le \pi_k \le 1$ and $\sum_{k=1}^{K} \pi_k = 1$. Given a set of datapoints, we can know the

## 2.3. Gaussian Mixture Regression

For Python implementation of GMR, please refer to [1] and https://github.com/AlexanderFabisch/gmr/tree/master/gmr.

We estimate a joint density of the inputs $x \in \mathbb{R}^N$ and outputs $\dot{x} \in \mathbb{R}^P$ through a K-component Gaussian mixture model as follows:

$$p(x, \dot{x}|\Theta) = \sum_{k=1}^{K} \pi_k p(x, \dot{x}|\mu^k \Sigma^k) \tag{2.4}$$

where, $\sum_{k=1}^{K} \pi_k = 1$ and $\mu^k = [\mu_x^k \;\; \mu_{\dot{x}}^k]^T$ and $\Sigma^k = \begin{bmatrix} \Sigma_{xx}^k & \Sigma_{x\dot{x}}^k \\ \Sigma_{\dot{x}x}^k & \Sigma_{\dot{x}\dot{x}}^k \end{bmatrix}$.

The conditional density is given by:

$$p(\dot{x}|x) = \sum_{k=1}^{K} \gamma_k p(y|x; \mu^k \Sigma^k) \tag{2.5}$$

$$\text{where,} \quad \gamma_k = \frac{\pi_k p(x|\mu_x^k, \Sigma_{xx}^k)}{\sum_{k=1}^{K} \pi_k p(x|\mu_x^k, \Sigma_{xx}^k)} \tag{2.6}$$

The regressive function $\dot{x} = f(x)$ is given by the expectation over this conditional density:

$$\dot{x} = f(x) = \mathbf{E}[p(\dot{x}|x)] = \sum_{k=1}^{K} \gamma_k(x)\tilde{\mu}^k(x) \tag{2.7}$$

$$\text{where,} \quad \tilde{\mu} = \mu_{\dot{x}}^k + \Sigma_{\dot{x}x}^k (\Sigma_{xx}^k)^{-1}(x - \mu_x^k) \tag{2.8}$$

Thus, the final expression becomes:

$$\begin{aligned} \dot{x} &= f(x) \\ &= \mathbf{E}[p(\dot{x}|x)] \\ &= \sum_{k=1}^{K} \frac{\pi_k p(x|\mu_x^k, \Sigma_{xx}^k)}{\sum_{k=1}^{K} \pi_k p(x|\mu_x^k, \Sigma_{xx}^k)} \left(\mu_{\dot{x}}^k + \Sigma_{\dot{x}x}^k (\Sigma_{xx}^k)^{-1}(x - \mu_x^k)\right) \end{aligned} \tag{2.9}$$

The figure 2.2 shown below represents as GMR with three components fitted on a function [2]:

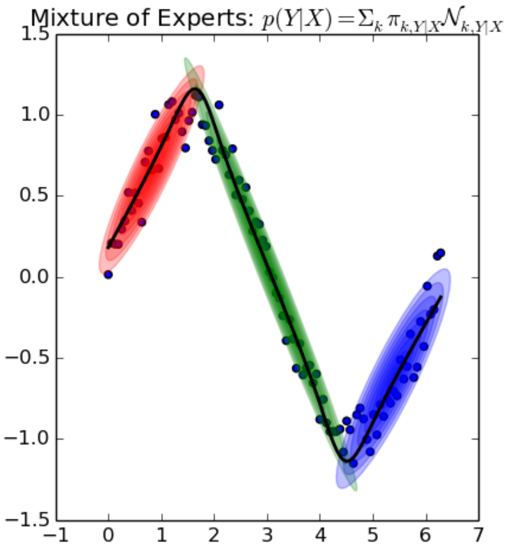

Figure 2.2: GMR with three components

The above function is learned using an Expectation Maximization algorithm, discussed in section 3.2. As we can notice, the GMR model can learn even a non linear function when a proper number of Gaussian components is chosen. The algorithm tries to adjust the means and covariances for each Gaussian component based on the dataset by maximizing the likelihood of dataset for the Gaussian components.

# CHAPTER 3

# LEARNING A NONLINEAR DYNAMICAL SYSTEM USING GMR

In this chapter we will use Gaussian Mixture Regression to learn the function $\dot{x} = f(x)$ which maps a two dimensional position space to a two dimensional velocity space.

## 3.1. LASA Handwriting Dataset

The LASAHandwriting Dataset consists of a library of 2D handwriting motions recorded from Tablet-PC [1]. The LASA Handwriting Dataset has now become a benchmark for evaluating and comparing different approaches to learn nonlinear DS [1]. It is available in the source repository: https://bitbucket.org/khansari/lasahandwritingdataset/src/master. The dataset is available in MATLAB file format. As this study is using Python for learning the Dynamical System, the dataset should be loaded in Python compatible format. pyLasaDataset is a Python library for loading and visualizeing LASA Handwriting dataset in Python [5].

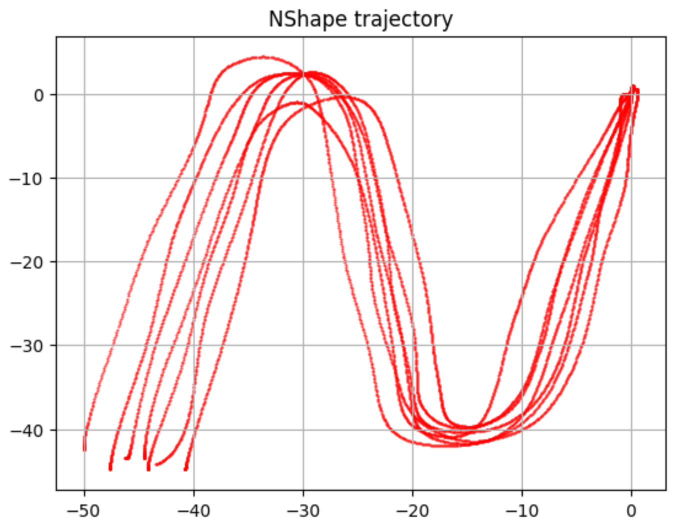

Figure 3.1: Visualizing NShape from LASA Dataset

The dataset consists of $M$ demonstrations with each trajectory of length $T_m$ where each datapoint is separated by a timestep 0.02 second. Thus, for all input datapoints $x \epsilon \mathbb{R}^N$, we have: $\{\mathbf{X}, \dot{\mathbf{X}}\} = \{X^m, \dot{X}^m\}_{m=1}^{M} = \{\{x^{t,m}, \dot{x}^{t,m}\}_{t=1}^{T_m}\}_{m=1}^{M}$. where, $m \epsilon \{1, ...., M\}$. The dataset can be explicitly written in the matrix form as:

$$X_m = \begin{bmatrix} x_1^{1,1} & x_2^{1,1} & ....... & x_N^{1,1} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{T_1,1} & x_2^{T_1,1} & ....... & x_N^{T_1,1} \\ x_1^{1,2} & x_2^{1,2} & ....... & x_N^{1,2} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{t,m} & x_2^{t,m} & ....... & x_N^{t,m} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{T_M,M} & x_2^{T_M,M} & ....... & x_N^{T_M,M} \end{bmatrix} \tag{3.1}$$

where each row represent a single datapoint $x \epsilon \mathbb{R}^N$ and total number of rows are $L = M \times T_m$.

## 3.2. Expectation Maximization Algorithm

For Python implementation of the algorithm, please refer to: https://github.com/vbwanere/SEDS_in_python/blob/main/gmm_lib/gaussian_mixture.py

Our goal is to find the parameters of the distribution which maximize the likelihood given a training dataset $x$ (as discussed in section 3.1):

$$\max_{\Theta,\pi} \log \mathcal{L}(\Theta, \pi|x) = \max_{\Theta,\pi} \log p(x|\Theta, \pi) \tag{3.2}$$

As we have $L$ number of datapoints, considering the datapoints as i.i.d samples, we can write the likelihood more explicitly as:

$$\log p(x|\Theta, \pi) = \sum_{i=1}^{L} \log(\sum_{k=1}^{K} \pi_k p(x_i|\mu^k, \Sigma^k)) \tag{3.3}$$

Which can be maximized w.r.t the parameters $\{\mu, \Sigma, \pi\}$ using the Expectation Maximization (EM) algorithm as given below:

1. Initialization Step: At $t = 0$, we initialize priors to uniform probabilities as: $\pi^{(0)} = [\pi_1^{(0)}, ...., \pi_K^{(0)}]^T$. For each cluster $k$ we initialize the means as: $\mu^{(0)} = \{\mu^{1(0)}, ..., \mu^{K(0)}\}$ and covariances as:

$\Sigma^{(0)} = \{\Sigma^{1(0)}, ..., \Sigma^{K(0)}\}$ from the K-means algorithm. For the Python implementation of K-Means algorithm, please refer to https://github.com/vbwanere/SEDS_in_python/blob/main/k_means_lib/kmeans.py.

2. Expectation Step (Membership probabilities): At each iteration t, we estimate, for each Gaussian k, the probability that this Gaussian is responsible for generating each point of the dataset. We use the expression for the posteriori probability for the $k$-th component given as:

$$p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)}) = \frac{\pi_k^{(t)} p(x_i|\mu^k, \Sigma^k)}{\sum_{k=1}^{K} \pi_k^{(t)} p(x_i|\mu^{k(t)}, \Sigma^{k(t)})} \tag{3.4}$$

These probabilities are computed for all the datapoints ($x_i \epsilon \mathbb{R}^n$) for every the cluster, which gives us a matrix of membership probabilities known as responsibility matrix $\mathcal{R}_k$:

$$\mathcal{R}_k = \begin{bmatrix} p(z_1 = 1) & p(z_1 = 2) & ...... & p(z_1 = K) \\ . & . & ...... & . \\ . & . & ...... & . \\ . & . & ...... & . \\ p(z_L = 1) & p(z_L = 2) & ...... & p(z_L = K) \end{bmatrix} \tag{3.5}$$

Each element of the reponsibility matrix gives the probability with which a datapoint belongs to a cluster $k$.

3. Maximization Step (Updating the Parameters): We update the priors as:

$$\pi_k^{(t+1)} = \frac{1}{L} \sum_{i=1}^{L} p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)}) \tag{3.6}$$

where, $p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)})$ is given by 3.4. The means are updated as:

$$\mu^{k(t+1)} = \frac{\sum_{i=1}^{L} p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)}) x_i}{\sum_{i=1}^{L} p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)})}. \tag{3.7}$$

The covariances are updated as:

$$\Sigma^{k(t+1)} = \frac{\sum_{i=1}^{L} p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)})(x_i - \mu^{k(t+1)})(x_i - \mu^{k(t+1)})^T}{\sum_{i=1}^{L} p(z_i = k|x_i, \Theta^{(t)}, \pi^{(t)})} \tag{3.8}$$

4. Repeat step 2 and then 3 till there is no further significant change in the liklihood.

## 3.3. Fitting LASA Dataset using GMR and EM

As mentioned in section 3.1, I have used pyLasaDataset [5] and a Python library ‘gmr‘ by Alexander Fabisch [2] to learn the trajectories given in the pyLasaDataset. Please refer to the Python notebook for implementation details at https://github.com/vbwanere/SEDS_in_python/blob/main/examples/GMR.ipynb

## 3.4. Using GMR to learn LASA Dataset

We will fit a GMR model on a *NShape* trajectory (figure 3.1) and few other trajectories from pyLasaDataset.

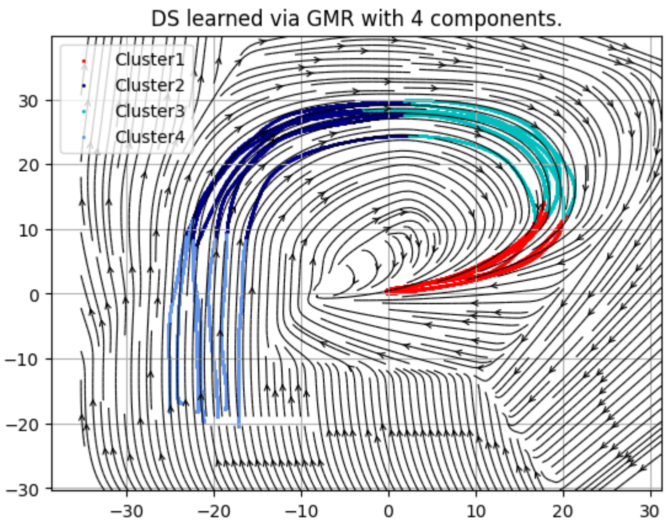

Figure 3.2: Visualizing a DS learned on *PShape* from LASA Dataset

It can be noticed from the figures 3.2 and 3.3 that only some velocity contours are pointing to attractor which shows that the learned dynamical system using GMR model is not stable. There is no guarantee of reaching to the attractor if we start at some arbitrary point in the task space, which is not desirable at all. The learned dynamical system should respect the stability constraints. The next chapter studies the approach to include the stability constraints in the Machine Learning model. It uses the Lyapunov stability theory and reformulates the original GMR problem to enforce the stability constraints derived using Lyapunov stability theory on the dynamical system matrix.

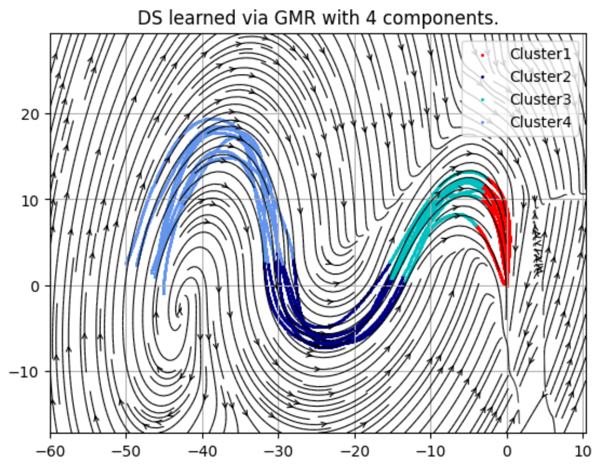

Figure 3.3: Visualizing a DS learned on *Sine* shape from LASA Dataset

The following figure 3.4 can give an idea of a stable dynamical system [1]. It can bee observed that the system is stable at the attractor i.e. all points in the task space have the velocity contours that eventually lead to the attractor.

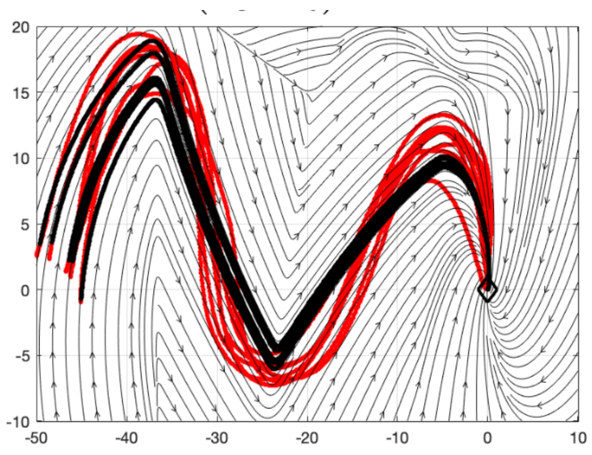

Figure 3.4: Visualizing a stable DS learned on *Sine* shape from LASA Dataset

# CHAPTER 4

# LEARNING A STABLE NONLINEAR DYNAMICAL SYSTEM

## 4.1. Nonlinear DS as a mixture of Linear DS

As given in [1], which is the central idea in using the GMR model for learning a nonlinear dynamical system, we can write a non linear dynamical system as a mixture of linear dynamical systems as follows:

$$\begin{aligned}\dot{x} &= f(x) \\ &= \sum_{k=1}^{K} \gamma_k(x)(A_x^k + b^k)\end{aligned} \tag{4.1}$$

where $A^k \epsilon \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$ and $b^k \epsilon \mathbb{R}^{\mathbb{N}}$ are the k-th linear system parameters. $\gamma_k(x) : \mathbb{R}^{\mathbb{N}} \rightarrow \mathcal{R}$ is the mixing function, also referred to as the activation function. Please refer to the figure below which depicts the idea in one dimension:

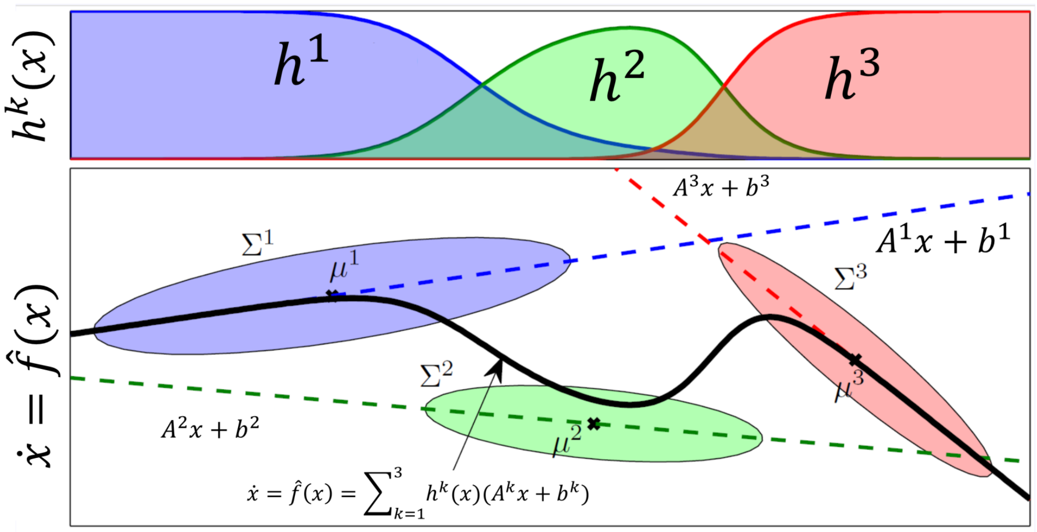

Figure 4.1: Illustration of parameters defined in equation 4.1 and their effects on $f(x)$ for a 1-D model constructed with 3 Gaussians. $h(x)$ is the activation function.

It should be noted that even if a given number of dynamical systems are stable, their mixture is not always stable [1].

## 4.2. Constrained GMR

As given in [1], using a change of variable and defining:

$$\begin{aligned} A^k &= \Sigma^k_{\dot{x}x}(\Sigma^k_x)^{-1} \\ b^k &= \mu^k_{\dot{x}} - A^k \mu^k_x \\ \gamma_k &= \frac{\pi_k p(x|\mu^k_x, \Sigma^k_{xx})}{\sum_{k=1}^K \pi_k p(x|\mu^k_x, \Sigma^k_{xx})} \end{aligned} \tag{4.2}$$

Substituting 4.2 in 2.7 and using 2.8 yields:

$$\begin{aligned} \dot{x} &= f(x; \Theta_{GMR}) \\ &= \mathbf{E}[p(\dot{x}|x)] = \sum_{k=1}^{K} \gamma_k(x)(A^k_x + b^k) \end{aligned} \tag{4.3}$$

We can apply Lyapunov stability theory on the above formulation of the nonlinear dynamical system.

## 4.3. SEDS Algorithm

This algorithm ensures that the dynamical system is globally asymptotically stable. We formulate the parameter estimation of the Gaussian mixtures as an optimization problem and enforce the stability constraints derived from Lyapunov stability theory. The likelihood objective functions with the necessary constraints is given as:

$$\begin{aligned} \min_{\Theta_{GMR}} \quad & J(\Theta_{GMR}) = -\frac{1}{L}\sum_{m=1}^{M}\sum_{t=0}^{T_m} \log p(x^{t,m}, \dot{x}^{t,m}|\Theta_{GMR}) \\ \text{subject to} \quad & (\mathbf{a})\ \ b^k = -A^k x^* \\ & (\mathbf{b})\ \ A^k + (A^k)^T \prec 0 \\ & (\mathbf{c})\ \ \Sigma^k \succ 0 \qquad\qquad \forall\ k = 1, ..., K \\ & (\mathbf{d})\ \ 0 < \pi_k \le 1 \\ & (\mathbf{e})\ \ \Sigma^K_{k=1}\pi_k = 1 \end{aligned} \tag{4.4}$$

For solving the above optimization problem we use interior point method [3] for which we need derivatives of the objective function with respect to the optimization parameters. The derivatives used are given by in [4]. For ongoing Python implementation of the SEDS algorithm using the *ipyopt* (a Python connector of interior point optimizer) [6], please refer to https://github.com/vbwanere/SEDS_in_python/blob/main/examples/GMR.ipynb.

### 4.3.1. Optimization Problem Formulation

There are four variants of the optimization problem as given in [4]:

1. Mean Square Error Optimization The optimization problem is given as:

$$
\begin{aligned}
\min_{\Theta_{GMR}} \quad & J(\Theta_{GMR}) = -\frac{1}{2L}\sum_{m=1}^{M}\sum_{t=0}^{T_m}(f(\dot{x}^{t,m}) - \dot{x}^{t,m})^T(f(\dot{x}^{t,m}) - \dot{x}^{t,m}) \\
\text{subject to} \quad & \textbf{(a)} \;\; b^k = -A^k x^* \\
& \textbf{(b)} \;\; A^k + (A^k)^T \prec 0 \\
& \textbf{(c)} \;\; \Sigma^k \succ 0 \qquad\qquad \forall\; k = 1, ..., K \\
& \textbf{(d)} \;\; 0 < \pi_k \leq 1 \\
& \textbf{(e)} \;\; \Sigma_{k=1}^{K}\pi_k = 1
\end{aligned}
\tag{4.5}
$$

2. Alternative Mean Square Error Optimization:

   Here, we transform the optimization parameters as:

$$
\begin{aligned}
\tilde{\pi}^k &= ln(\pi^k) \\
L^k_{xx} &= Chol(\Sigma^k_{xx})
\end{aligned}
\tag{4.6}
$$

   where, $L^k$ are $2d \times 2d$ lower triangular matrices. Also, from equation 4.3 and 4.5a:

$$
f(\dot{x}^{t,m}) = \sum_{k=1}^{K}\gamma_k(x)A^k(x - x^*)
\tag{4.7}
$$

From equation 4.12 and equation 4.7, the optimization problem becomes:

$$\begin{aligned} \min_{\Theta_{GMR}} \quad & J(\Theta_{GMR}) = -\frac{1}{2L}\sum_{m=1}^{M}\sum_{t=0}^{T_m}(f(\dot{x}^{t,m}) - \dot{x}^{t,m})^T(f(\dot{x}^{t,m}) - \dot{x}^{t,m}) \\ \text{subject to} \quad & A^k + (A^k)^T \prec 0 \qquad \forall\ k = 1, ..., K \end{aligned} \tag{4.8}$$

The GMM parameters can be reconstructed as:

$$\begin{aligned} \pi^k &= e^{\tilde{\pi}^k}/(\sum_{i=1}^{K} e^{\tilde{\pi}^i}) \\ \Sigma^k_{xx} &= L^k_{xx}(L^k_{xx})^T \\ \Sigma^k_{\dot{x}x} &= A^k\Sigma_{xx} \end{aligned} \tag{4.9}$$

3. Likelihood Optimization

   It is given by equation 4.5.

4. Alternative Likelihood Optimization

   Here also, we transform the optimization parameters as:

$$\begin{aligned} \tilde{\pi}^k &= ln(\pi^k) \\ L^k &= Chol(\Sigma^k) \end{aligned} \tag{4.10}$$

The optimization problem becomes:

$$\begin{aligned} \min_{\Theta_{GMR}} \quad & J(\Theta_{GMR}) = -\frac{1}{L}\sum_{m=1}^{M}\sum_{t=0}^{T_m}\log p(x^{t,m}, \dot{x}^{t,m}|\Theta_{GMR}) \\ \text{subject to} \quad & A^k + (A^k)^T \prec 0 \end{aligned} \tag{4.11}$$

The GMM parameters can be reconstructed as:

$$\begin{aligned} \pi^k &= e^{\tilde{\pi}^k}/(\sum_{i=1}^{K} e^{\tilde{\pi}^i}) \\ \Sigma^k &= L^k(L^k)^T. \end{aligned} \tag{4.12}$$

# CHAPTER 5

# CONCLUSION

The objective of this study was to learn about the different machine learning techniques available which can used for robot control in task space. After studying [1], Gaussian Mixture Regression was found to be a suitable machine learning that can be used to control a robotic manipulator. Regression algorithms such as SVM and GPR cannot be used to model a dynamical system because they fail to capture the physics of the problem i.e. stability. Python implementation of the Unstable Regressor for Dynamical System using Gaussian Mixture Regression has been completed as discussed in chapter 3. One important goal was to implement the SEDS algorithm in Python, which is underway. The said goal could not be achieved within the stipulated time due to the complexity of the problem and the material that had to be studied to have a Python implementation of SEDS. For the SEDS implementation, the alternative likelihood optimization problem objective function and derivatives formulation is completed in Python; now it is to be fed to a suitable optimization solver and the solutions are to be tested and verified.

## References

[1] Nadia Figueroa Aude Billard, Sina Mirrazavi. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach.* MIT Press, 2022.

[2] Alexander Fabisch. gmr: Gaussian mixture regression. *Journal of Open Source Software*, 6(62):3054, 2021.

[3] Debdas Ghosh. A primer to mathematical optimization, February 2022.

[4] S. Mohammad Khansari-Zadeh and Aude Billard. The derivatives of the seds optimization cost function and constraints with respect to the learning parameters. page 8, 2011.

[5] Saif Sidhik. pylasadataset, 2022. GitHub repository.

[6] Eric Xu. ipyopt, 2018. GitHub repository.