On Piecewise Linear Approxiamtion of Continuous functions and Application to Control
Final Proyect ECE 417 - Fall 2000

Kurt Plarre

December 8, 2000

1 Introduction

This paper is concerned with the piecewise linear approximation of a nonlinear function $f^* : [0,1]^d \to \Re$ and its application to the control of nonlinear systems.
It is very common in automatic control, to use linear approximations to nonlinear functions in order to design a suitable controller for a dynamic system. If the nonlinear system model is known to the designer, a common technique is to approximate the nonlinearities in the model by a first order Taylor series around an equilibrium point x_e, or an equlibrium trayectory $x^*(t)$ and design a controller for the resulting linear model. This technique is usefull for regulation purposes, i.e., when the control goal is to maintain the state of the system at the equilibrium point or when the desired trayectory of the system does not deviate much from $x^*(t)$. In other words, linearization around an equilibrium point yields an approximating model that is valid locally. If the system is known to be linear, but its parameters are unknown, then a parameter identification scheme can be used in order to find the value of the unknown parameters.
If the system is nonlinear and the control goal is to track a signal that does not stay close to some specific trayectory, or the inicial conditions are such that $x(0)$ is far from x_e, then the linear approximation might not give the desired results. In this case other techinques have to be used in order to obtain an approximating model that is valid over a large region of $\Re^d$.
This paper suggests a method for finding piecewise linear approximations of nonlinear functions based on a set of n datapoints $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in [0,1]^d$ and $y_i = f^*(x_i) \in \Re$. This method is based on the Hinge Finding Algorithm proposed in [2].
In section 2 the problem is formally stated and the notation is introduced. Section 3 presents the approximation criteria, i.e. the specific cost function to be used. In section 4 the Hinge Finding Algorithm is presented and and section 5 shows its equivallence with Newton's method applied to the sum of squared errors (see [4]). Section 6 shows the suggested extension to the Hinge Finding Algorithm. Section 7 shows an example of the application of this method and Section 8 discusses some of the difficulties encountered when using this method. Finally, section 9 presents some conclusions and possible extensions.

2 The problem

Consider a possibly nonlinear continuous function $f^* : [0,1]^d \to \Re$.
The goal is to find a function $\hat{f} \in \mathcal{F}$, where $\mathcal{F}$ is a prespecified class of functions, such that

$$\hat{f} = \underset{g \in \mathcal{F}}{argmin} J(f^*, g) \tag{1}$$

$J(f^*, \cdot) : \mathcal{F} \to \Re$ is a cost function that induces an order in $\mathcal{F}$. For $g_1, g_2 \in \mathcal{F}$ we say g_1 is better than g_2, if and only if $J(f^*, g_1) < J(f^*, g_2)$. The particular choice of J used in this paper is presented in section 3.
The class of approximating functions $\mathcal{F}$ considered here is the class of all continuous piecewise linear functions, which will be denoted by $\mathcal{F}_{pl}$. Each function $f_{pl} \in \mathcal{F}_{pl}$ is specified by a set of planes $\{p_i(x) = a_i^T x + b_i\}_{i=1}^m$, and a set of "cells" $\{C_i \subset [0,1]^d\}_{i=1}^m$ that represents a partition of $[0,1]^d$, i.e.,

$$\begin{aligned} C_i \cap C_j &= \emptyset \qquad \text{when } i \neq j \\ \bigcup_{i=1}^{m} C_i &= [0,1]^d \end{aligned} \tag{2}$$

Given $\mathcal{P}$ and $\mathcal{C}$ the piecewise linear function is defined as

$$f_{pl}(x) = p_i(x) \quad \text{iff} \quad x \in C_i \tag{3}$$

Given the set of hyperplanes $\mathcal{P}$ there are infinite many possible choices for $\mathcal{C}$ .In this paper the cells are restricted to be such that f_{pl} is continuous, but they are not restricted to be convex. The description of the cells is not discussed here, because the method for fitting a piecewise linear function to the given data, presented in section 6 does not make use of this information.

2.1 Notation

In this article, two different notations are used to describe a (d+1)-dimensional hyperplane $y = p(x)$. The first one shows explicitly the constant term b: $p(x) = a^T x + b$, where $a \in \Re^d$ and $b \in \Re$. The second one is used to simplify the calculations and replaces x by the (d+1)-dimensional vector $\begin{pmatrix} x \\ 1 \end{pmatrix}$ and a by $\begin{pmatrix} a \\ b \end{pmatrix}$. Using this notation, the hyperplane can be written as $p(x) = a^T x$. Both notations are used interchangeably.

3 Approximation critera.

Let $f^* : [0,1]^d \rightarrow \Re$ be a continuous function. In order to find a good approximating function $\hat{f} : [0,1]^d \rightarrow \Re$ it is necessary to find a criterion to decide when one approximating function is better than another. This leads to the definition of a cost function $J(f^*, \hat{f})$.
For $\hat{f}$ linear, i.e., $\hat{f}(x) = a^T x + b$, some possible choiches of J are:

1. If f^* is differentiable, $J(f^*, a, b) = \left[a^T x + b - f^*(x_0)\right]^2 + [a - \nabla f^*(x_0)]$ produces the approximating function $\hat{f} = f(x_0) + \nabla f(x_0)^T (x - x_0)$, i.e., a first order Taylor series approximation on x_0.

2. $\min\limits_{a \in \Re^d, b \in \Re} J(f^*, a, b)$, where $J(f^*, a, b) = \int\limits_{[0,1]^d} \frac{[a^T x + b - f^*(x)]^2}{\|a\|^2 + 1} dx$ produces a plane that minimizes the squared distances from the points $\begin{pmatrix} x \\ f^*(x) \end{pmatrix}$ to the plane.

3. The minimization of $J(f^*, a, b) = \int\limits_{[0,1]^d} [a^T x + b - f^*(x)]^2 dx$ produces a plane that minimizes the squared vertical distances from the points on the surface $\{(x, f^*(x)), x \in \Omega\}$ to the plane.

In this paper the last of the above mentioned cost functions is used. The approximation of the function f^* will be based on a set of data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \Re^d$ and $y_i = f^*(x_i) \in \Re$. The cost function will take the form

$$J(f^*, a, b) = \sum_{i=1}^{n} [a^T x_i + b - y_i]^2 \tag{4}$$

For a piecewise linear function f_{pl} given by the planes $\mathcal{P} = \{p_1(a) = a_1^T x + b_1, p_2(x) = a_2^T x + b_2, \ldots, p_m(x) = a_m^T x + b_m\}$ and cells $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, the cost function takes the form

$$J(f^*, \hat{f}) = \sum_{k=1}^{m} \sum_{x_i \in C_i} \left[a_k^T x_i + b_k - y_i\right]^2 \tag{5}$$

4 The Hinge Finding Algorithm

The simplest piecewise linear function is a plane itself. In this case $m = 1$, $\mathcal{C} = \{C_1\}$ and $\mathcal{P} = \{p_1\}$.
The second most simple case arises when $m = 2$. Then $\mathcal{P} = \{p_1 = a1^T x, p_2 = a_2^T x\}$. The $(d-1)$ dimensional hyperplane given by $(a_1 - a_2)^T x = 0$ is called the "hinge". Letting $\Delta = a_1 - a_2$, there are two choices for the cells

$$\begin{array}{ccc} C_1 = \{x : \Delta^T x \leq 0\} & \text{and} & C_2 = \{x : \Delta^T x > 0\} \\ & \text{or} & \\ C_1 = \{x : \Delta^T x > 0\} & \text{and} & C_2 = \{x : \Delta^T x \leq 0\} \end{array} \tag{6}$$

In the first case, the piecewise function defined is concave, in the second case it is convex.
In [2] a method for fitting hinging hyperplanes to a set of data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ is presented. The procedure, which is called the "Hinge Finding Algorithm" can be described as follows:

1. Let $\Delta_0 \in \Re^d$ be a random vector.
2. Let $C_1^k = \{x \in [0,1]^d : \Delta_k^T x \leq 0\}$ and $C_2^k = \{x \in [0,1]^d : \Delta_k^T x > 0\}$.
3. Let $a_1^k = \underset{a \in \Re^{d+1}}{argmin} \sum_{x_i \in C_1^k} \left[a^T x_i - y_i\right]^2$ and $a_2^k = \underset{a \in \Re^{d+1}}{argmin} \sum_{x_i \in C_2^k} \left[a^T x_i - y_i\right]^2$.
4. Let $\Delta_{k+1} = a_1^k - a_2^k$. In general $\Delta_{k+1} \neq \Delta_k$.
5. Repeat 2 until $\Delta_{k+1} = \Delta_k$.

By the definitions of Δ_k and C_1, C_2 the piecewise linear function fitted in this way is concave. Changing the definition of the cells to $C_1^k = \{x \in [0,1]^d : \Delta_k^T x > 0\}$ and $C_2^k = \{x \in [0,1]^d : \Delta^T x \leq 0\}$ produces a concave function.
This procedure might not converge, but it is shown in [2] that if $y_i = h(x_i)$, where h is given by hinging hyperplanes, i.e.,

$$\begin{array}{rcll} h(x) & = & min\{a_1^T x, a_2^T x\} & \text{(concave hinging hyperplanes), or} \\ h(x) & = & max\{a_1^T x, a_2^T x\} & \text{(convex hinging hyperplanes)} \end{array} \tag{7}$$

then the $\Delta_k \rightarrow \Delta^*$, the true hinge.
One motivation for the use of this method is the fact that it is fast, even in high dimensions.

5 Fitting planes

This section shows the equivalence of three different approaches to fitting hinging hyperplanes:

1. Refitting the hinge (Hinge Finding Algorithm).
2. Reffiting the planes.
3. Newton's method applied to the sum of squared errors.

The procedures shown here are designed for fitting concave hinging hyperplanes, but with minor changes they can be used to fit convex hinging hyperplanes aswell.

5.1 Refitting the hinge

This algorithm was presented in section 4, but is repeated here in condensed form.

1. $\Delta_0 :=$ random vector
2. $a_1^k := \underset{a \in \Re^{d+1}}{argmin} \sum_{\Delta_k^T x_i \leq 0} \left[a^T x_i - y_i\right]^2$

   $a_2^k := \underset{a \in \Re^{d+1}}{argmin} \sum_{\Delta_k^T x_i > 0} \left[a^T x_i - y_i\right]^2$
3. $\Delta_{k+1} := a_1^k - a_2^k$, repeat from 2 until $\Delta_{k+1} = \Delta_k$.

The solution to the equations in point 2 are:

$$\begin{aligned} a_1^k &= \left[\sum_{\Delta_k^T x_i \leq 0} x_i x_i^T\right]^{-1} \sum_{\Delta_k^T x_i \leq 0} x_i y_i \\ a_2^k &= \left[\sum_{\Delta_k^T x_i > 0} x_i x_i^T\right]^{-1} \sum_{\Delta_k^T x_i > 0} x_i y_i \end{aligned} \tag{8}$$

5.2 Refitting the planes

This algorithm is a minor modification to the previous one, but is presented here because this approach will be used later to extend the hinging hyperplane fitting algorithm.

1. $a_1^0 :=$ random vector
   $a_2^0 :=$ random vector

2. $a_1^{k+1} := \underset{a \in \Re^{d+1}}{argmin} \sum_{\left(a_1^k\right)^T x_i \leq \left(a_2^k\right)^T x_i} \left[a^T x_i - y_i\right]^2$
   $a_2^{k+1} := \underset{a \in \Re^{d+1}}{argmin} \sum_{\left(a_1^k\right)^T x_i > \left(a_2^k\right)^T x_i} \left[a^T x_i - y_i\right]^2$

3. Repeat from 2 until $a_1^{k+1} = a_1^k$ and $a_2^{k+1} = a_2^k$.

The solution to the equations in 2 is the same as that given by equations 8, considering that $\Delta_k^T x \leq 0$ if and only if $x^T a_1^k \leq x^T a_2^k$ and $\Delta_k^T x > 0$ if and only if $x^T a_1^k > x^T a_2^k$.

5.3 Newtons method

Given a twice differentiable function $g : \Re^d \rightarrow \Re$. Newton's method is a searching algorithm that finds solutions to $\nabla g(x^*) = 0$. If the starting point x_0 provided to the algorithm is close to a local minimum of g, then the algorithm will converge to the local minimum.
The algorithm is as follows:

1. Let $x_0 \in \Re^d$ be close to a local minimum of g.

2. Let $x_{k+1} = x_k - \mathcal{G}(x_k)^{-1} \nabla g(x_k)$, where $\mathcal{G}(x_k)$ is the Hessian matrix of g at the point x_k.

3. Repeat 2 until covergence is detected (according to the chosen stopping criterion).

Applied to finding a local minimum of the sum of squared errors this algorithm takes the form:

1. Let $a^0 := \begin{pmatrix} a_1^0 \\ a_2^0 \end{pmatrix}$ a random vector in $\Re^{2d}$.

2. Define the cost: $J(a) := \sum_{i=1}^{n} \left[min\{a_1^T x_i, a_2^T x_i\} - y_i\right]^2$, where $a := \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$.

3. Let $a^{k+1} := a^k - \mathcal{J}_a(a^k)^{-1}\nabla_a J(a^k)$, where $\mathcal{J}_a(a^k)$ is the Hessian of $J(a)$ with respect to a at a^k.

4. Repeat 3 until convergence is detected.

It is important to notice that the cost function defined in step 2 of the described method is continuous, but not differentiable. In fact $J(a)$ is a piecewise quadratic function and is infinitely differentiable except when a data point lies exactly on the hinge. So, in defining the gradient and Hessian of J at a^k it is assumed that there exists no point x_i such that $x_i^T a_1^k = x_i^T a_2^k$. The fact that J is piecewise quadratic can cause problems to the convergence of the algorithms. This problem will be discussed in section 8.
The cost J can be rewritten as

$$J(a) = \sum_{a_1^T x_i \leq x_i^T a_2} \left[a_1^T x_i - y_i\right]^2 + \sum_{a_1^T x_i > x_i^T a_2} \left[a_2^T x_i - y_i\right]^2 \tag{9}$$

From here the gradient of J can be computed:

$$\nabla_a J(a^k) = \begin{pmatrix} \sum_{x_i^T a_1^k \leq x_i^T a_2^k} 2\left[x_i^T a_1^k - y_i\right] x_i \\ \sum_{x_i^T a_1^k > x_i^T a_2^k} 2\left[x_i^T a_2^k - y_i\right] x_i \end{pmatrix} \tag{10}$$

The Hessian can also be obtained:

$$\mathcal{J}_a(a^k) = 2 \begin{bmatrix} \sum_{x_i^T a_1^k \leq x_i^T a_2^k} x_i x_i^T & 0 \\ 0 & \sum_{x_i^T a_1^k > x_i^T a_2^k} x_i x_i^T \end{bmatrix} \tag{11}$$

Using the expressions 10 and 11, in step 2 of the application of Newton's method the following solutions for a_1^{k+1} and a_2^{k+1} are obtained

$$\begin{aligned} a_1^{k+1} &= x_1^k - \left[\sum_{x_i^T a_1^k \leq x_i^T a_2^k} x_i x_i^T\right]^{-1} \sum_{x_i^T a_1^k \leq x_i^T a_2^k} \left[x_i^T a_1^k - y_i\right] x_i \\ a_2^{k+1} &= a_2^k - \left[\sum_{x_i^T a_1^k > x_i^T a_2^k} x_i x_i^T\right]^{-1} \sum_{x_i^T a_1^k > x_i^T a_2^k} \left[x_i^T a_1^k - y_i\right] x_i \end{aligned} \tag{12}$$

Note that

$$\left[\sum_{x_i^T a_1^k \le x_i^T a_2^k} x_i x_i^T\right]^{-1} \sum_{x_i^T a_1^k \le x_i^T a_2^k} (x_i^T a_1^k) x_i \tag{13}$$

is the same expression obtained when fitting a plane to the points given by $\{(x_i, x_i^T a_1^k)\}_{x_i^T a_1^k \le x_i^T a_2^k}$, i.e., fitting a plane to the points on a plane. This means that the expression in 13 is equal to a_1^k. Considering this fact, the first equation in 12 can be simplified to

$$a_1^{k+1} = \left[\sum_{x_i^T a_1^k \le x_i^T a_2^k} x_i x_i^T\right]^{-1} \sum_{x_i^T a_1^k \le x_i^T a_2^k} y_i x_i \tag{14}$$

This expression is the same as the first equation in 8. The same analysis is valid for a_2^{k+1}.
This shows that the three methods considered are really the same but they give different insights on the behaviour of the refitting method.
Let $a^* = \begin{pmatrix} a_1^* \\ a_2^* \end{pmatrix}$ be a local minimum of the cost $J(a)$ and suppose that there is no data point x_i, such that $x_i^T a_1^* = x_i^T a_2^*$. Then a^* is a stationary point of Newton's method, which means that a^* is also a stationary point of the refitting algorithm. In other words

$$\begin{aligned} a_1^* &= \underset{a}{argmin} \sum_{x_i^T a_1^k \le x_i^T a_2^k} \left[a^T x_i - y_i\right]^2 \\ a_2^* &= \underset{a}{argmin} \sum_{x_i^T a_1^k > x_i^T a_2^k} \left[a^T x_i - y_i\right]^2 \end{aligned} \tag{15}$$

On the other hand, any stationary point of the refitting algorithm is a local minimum of the error J. This observation presents a motivation for the use of refitting methods. One of the difficulties of these methods is that it can easily happen that at the local minimum of J a data point lies exactly at the hinge. This problem will be studied in section 8.

6 Extension of the refitting algorithm

In section 4, three different interpretations of the refitting algorithm were presented. The first one is a refitting of the hinge (Hinge Finding Algorithm). The second one corresponds to refitting the planes, which is just a way of rewriting the first algorithm. The third one uses Newton's method to find a local minimum of the sum of squared errors, and leads to an algorithm that is equivallent to the previous ones.
In this section, the third form of the refitting algorithm is used to find a method for fitting more than two planes simultaneously.
The piecewise linear function to be fitted will be described as follows:
Consider a set of m planes $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ where $p_j(x) = a_j^T x$, and a set of m cells $\mathcal{C} = \{C_1(a), C_2(a), \ldots, C_3(a)\}$, where $C_j(a) \subset [0,1]^d$ and

$$\begin{array}{lcll} C_i(a) \cap C_j(a) & = & \emptyset & \text{when } i \neq j \\ \bigcup\limits_{j=1}^{m} C_j(a) & = & [0,1]^d & \end{array} \tag{16}$$

Here a denotes the vector of parameters $a = (a_1, a_2, \ldots, a_m)^T$. The piecewise linear function defined by $\mathcal{P}$ and $\mathcal{C}$ is given by

$$f_{pl}(x) = p_j(x) \quad \text{iff} \quad x \in C_j(a) \tag{17}$$

In this case the intersections $\Delta_{l\bar{l}}(a) = \{x \in \Re^d : a_l^T x = a_{\bar{l}}^T x\}$, for $l, \bar{l} \in \{1, 2, \ldots, m\}$ will be called the "hinges". A third condition will be added to the ones stated in 16: Given $\mathcal{P}$, $\mathcal{C}$ has to be chosen in such a way that f_{pl} is continuous.
Since the goal of this scheme is to approximate a set of datapoints $\mathcal{D}$ by means of a refitting algorithm, it will be necessary to generate a sequence of functions f_{pl}^k, defined by $\mathcal{P}^k = \{p_1(x) = x^T a_1^k, p_2(x) = x^T a_2^k, \ldots, p_m(x) = x^T a_m^k\}$ and $\mathcal{C}^k = \{C_1(a^k), C_2(a^k), \ldots, C_m(a^k)\}$. The computation of $\mathcal{C}^k$ such that f_{pl}^k is continuous, is hard and computationally demanding. Later in this section a 2 dimensional example will be provided.
The cost function can be written as

$$\begin{array}{rcl} J(a) & = & \sum\limits_{q=1}^{m} \sum\limits_{x_i \in C_i(a)} \left[a_q^T x - y_i\right]^2 \\ & = & \sum\limits_{q=1}^{m} J_q(a_q) \end{array} \tag{18}$$

where $J_q(a)$ is the cost on cell $C_q(a)$. Assuming that for $a = a^k$ no datapoint lies on one of the hinges there is a neighbourhood $\mathcal{N}$ of a^k, such that $J_q(a)$ is a function only of a_q when $a \in \mathcal{N}$. The gradient and Hessian of J can be then computed as

$$\begin{aligned} \nabla_a J(a) &= \begin{pmatrix} \nabla_{a_1} J_1(a_1) \\ \nabla_{a_2} J_2(a_2) \\ \dots \\ \nabla_{a_m} J_m(a_m) \end{pmatrix} \\ \mathcal{J}_a(a) &= \begin{bmatrix} \mathcal{J}^1_{a_1}(a_1) & 0 & \dots & 0 \\ 0 & \mathcal{J}^2_{a_2}(a_2) & & 0 \\ \dots & & & \\ 0 & \dots & 0 & \mathcal{J}^m_{a_m}(a_m) \end{bmatrix} \end{aligned} \tag{19}$$

It can be seen from 19 that applying Newton's algorithm to J is equivalent to applying Newton's algorithm to each J_q, considering the cells fixed. This is again equivalent to just reffiting the plane p_q to the datapoints contained in cell C_q. So, the following method is obtained:

1. Initialize a_q^0 , $q = 1, 2, \dots, m$ with an initial guess.
2. Compute the cells $C_q(a^0)$, , $q = 1, 2, \dots, m$.
3. Let $a_q^{k+1} = \left(\mathcal{J}^q_{a_q}(a_q^k)\right)^{-1} \nabla_{a_q} J_q(a_q^k)$.
4. Compute the new cells $C_q(a^{k+1})$.
5. Repeat from 3

In the algorithm presented below, no description of the cells is used. the only information used at each step is the subset of the datapoints that is contained in each cell, i.e., for each point x_i there will be a number $c_i(a^k)$ such that x_i is defined to be in $C_{c_i(a^k)}(a^k)$. Basically the numbers $c_i(a^k)$ define the cells. The algorithm is implemented in the following way:

1. Initialize a_q^0 , $q = 1, 2, \dots, m$ with an initial guess.
2. Find the cells $C_q(a^0)$, and for all datapoint x_i, compute $c_i(a^0)$, such that $x_i \in C_{c_i(a^0)}(a^0)$. This is the only step at which the cells define the values of $c_i(a^0)$.

3. Let $a_q^{k+1} = \left(\mathcal{J}_{a_q}^q(a_q^k)\right)^{-1} \nabla_{a_q} J_q(a_q^k)$ and $a^{k+1} = \{a_1^k, a_2^k, \ldots, a_m^k\}$.

4. Let $a_\alpha = (1-\alpha)a^k + \alpha a^{k+1}$ for $\alpha \in \Re$.
   For each $q \in \{1, 2, \ldots, m\}$ and for each point x_i, such that $c_i(a^k) = q$, let $\alpha_{qi} = inf\{\alpha : x_i \in \Delta_{qj}(a_\alpha), \text{for some } j \in \{1, 2, \ldots, m\}, j \neq q\}$. Let $\alpha_q = min\{\alpha_{qi} : \alpha_{qi} \geq 0\}$. In other words, α_q is the maximum value of α such that al points x_i with $c_i(a^k) = q$ are contained in $C_q(a_\alpha)$.

5. Let $\alpha^* = min\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. Then, $\forall \alpha < \alpha^*, c_i(a_\alpha) = c_i(a^k), i = 1, 2, \ldots, m$. Let $\alpha_{q^*i^*} = \alpha^*$. This means that point x_i, which is assigned to cell q is the first point that crosses the hinge Δqj for some j. Now a decision has to be taken wether to set $c_i(a_{\alpha^*}) = q$ or $c_i(a_{\alpha^*}) = j$, i.e., eiter leave x_i in cell C_q or reasigne it to cell C_j. The decision algorithm will not be described here.

6. Set $\bar{\alpha} = \alpha^* + \epsilon$, where ϵ is computed in such a way that no other point leaves the cell to which it is assigned.

7. Let $a^{k+1} = a_{\bar{\alpha}}$

8. Repeat from 3

This procedure is applied iteratively, by fitting initialy just a plane, and then splitting the cell which achieves the maximum error (thus adding one plane at a time) and applying the algorithm until a stationary point is found.
Section 7 provides a two dimensional exaple of the application of this algorithm.

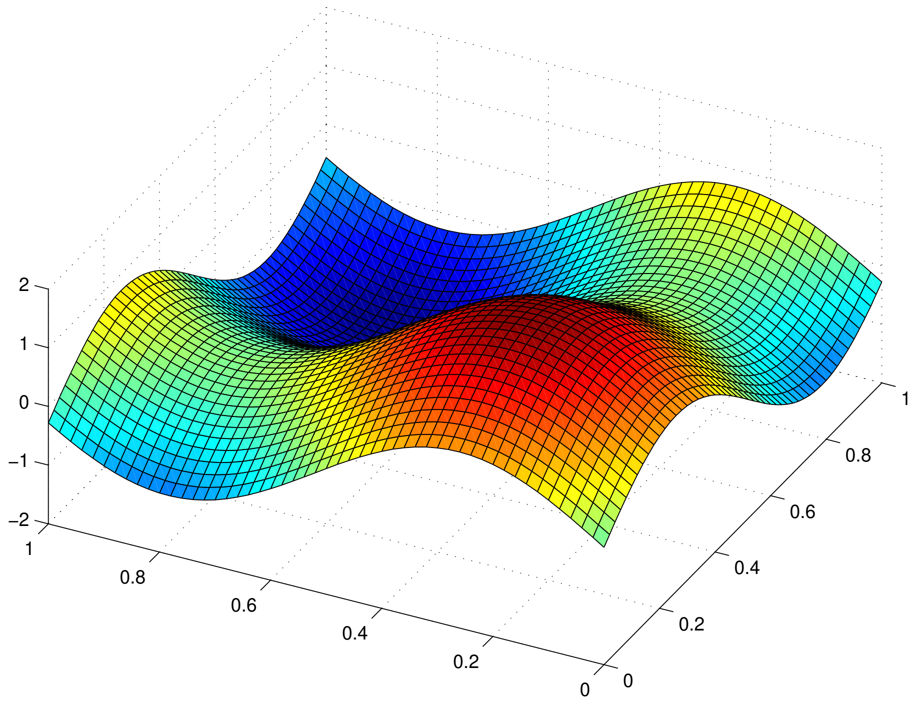

Figure 1: Function to be approximated.

7 Example

This section provides a two dimensional example of the application of the algorithm presented in section 6.

The function to be approximated is $f^*(x_1, x_2) = sin(6x_1) + sin(6x_2)$ and is shown in figure 1. The set of datapoints is $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{1000}$, where x_i is uniformly distributed in $[0, 1]^d$, and $y_i = f^*(x_i)$. The approximating functions for $m = 2$, $m = 3$, $m = 5$ and $m = 9$ are shown in figure 2.

To observe the effect of approximating a dynamic system using the algorithm presented, the following system was considered:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f^*(x_1 + \pi/6, x_2 + \pi/6) \end{aligned} \tag{20}$$

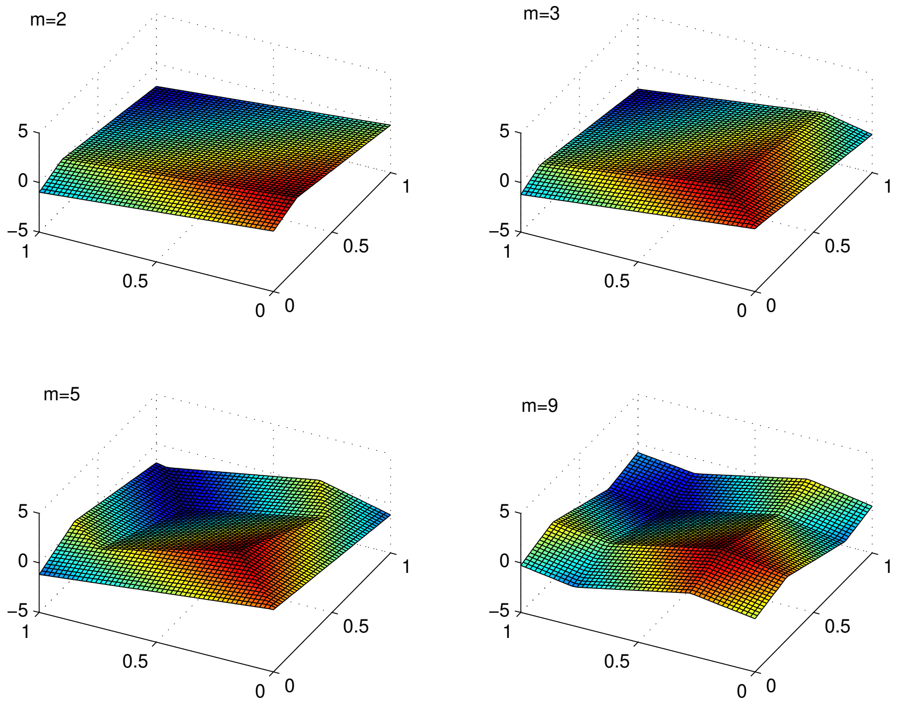

Figure 2: Approximating functions for m=2,3,5 and 9.

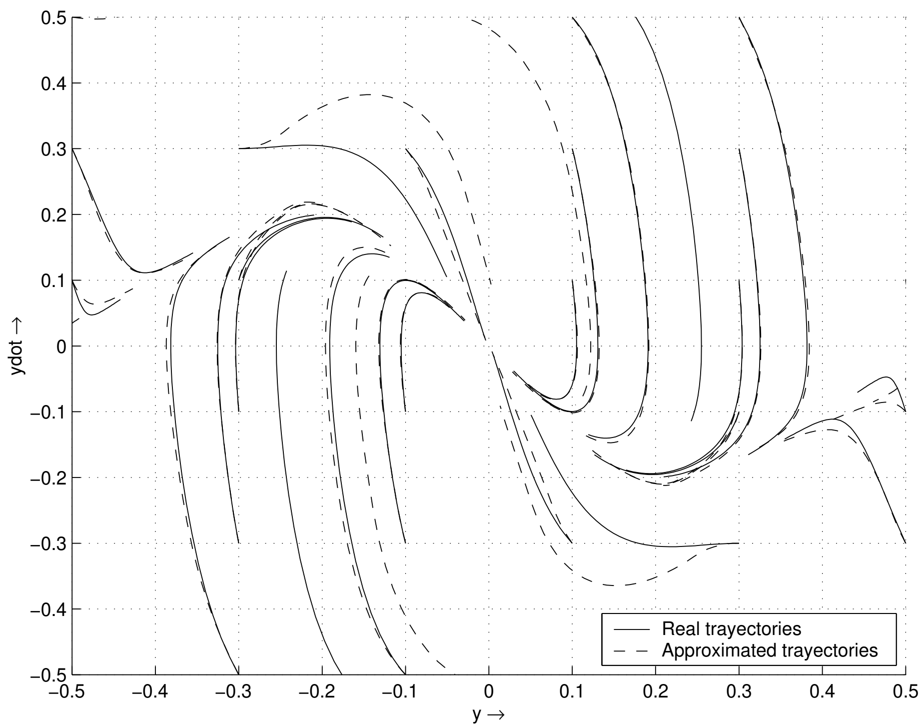

Figure 3: Real and approximated trayectories in state space.

Note that the variables are shifted in such a way that the origin is an equlibrium point of 20. The approximated system is

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \hat{f}(x_1 + \pi/6, x_2 + \pi/6) \end{aligned} \tag{21}$$

where $\hat{f}$ is the nine plane approximating function shown in figure 2. The real and approximated trayectories in state space, for different initial conditions are shown in figure 3. It can be seen that the approximated system behaves very similarly to the real system.

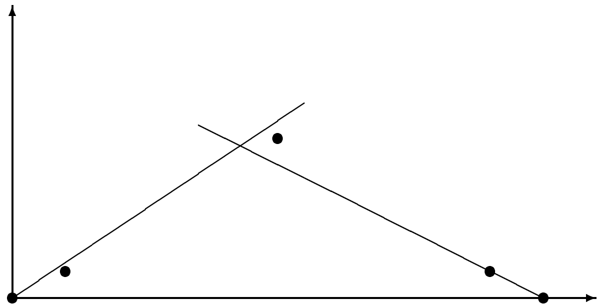

Figure 4: Example of a "bouncing point".

8 Problems of refitting algorithms

One of the greatest difficulties of the refitting methods considered in section 5 is the fact that the solution to $a^* = \underset{a}{argmin} J(a)$ could be such that $x_i^T a_1^* = x_i^T a_2^*$ for some datapoint x_i. In this case the error is not differentiable at a^*, which means that Newton's method can not be applied and the optimal planes given by $p_1(x) = x^T a_1^*$ and $p_2(x) = x^T a_2^*$ are not locally optimal. It is easy to construct examples of such a behaviour, even in dimension 1:
Cosider the data points $\mathcal{D} = \{(x_1, y_1) = (0,0), (x_2, y_2) = (1,1), (x_3, y_3) = (5,6), (x_4, y_4) = (9,1), (x_5, y_5) = (10,0)\}$ (see figure 4). Let $m = 2$

It is easy to see that if $x_1, x_2, x_3 \in C_1$ and $x_4, x_5 \in C_2$, the hinge will lie to the left of x_3. Also, if $x_1, x_2 \in C_1$ and $x_4, x_5, x_3 \in C_2$, the hinge will lie to the right of x_3. So, if the refitting algorithm is applied to $\mathcal{D}$ x_3 will bounce back and fort between C_1 and C_2, i.e., the algorithm does not converge. This happens because for this data set, the optimal hinge is the point $\Delta^* = 5$.
A point like (x_3, y_3) will be called "bouncing point". It is important to notice that in the previous example, it is not necessary that $x_3 = 5$. In fact, for $y_3 = 6$, and any $x_3 \in [4.08, 5.92]$, (x_3, y_3) is a bouncing point.

The existence of bouncing points makes the it very difficult to find even a local minimum of $J(a)$.

9 Conclusions

A possible method for approximating a set of datapoints by a piecewise linear function was suggested and discussed. Some of the difficulties of using such a method were also discussed. The main difficulties at this point are

1. The representation of the picewise linear surface. In particular, the cells can be nonconvex and difficult to handle. One way of overcomming this problem could be to use a representation of f_{pl} as a difference of two convex piecewise linear functions (see [3]).

2. The optimization of $J(a)$. Since $J(a)$ is a nondifferentiable function of a, the probem of finding even a local minimum is difficult. In fact, it seems (from experience) that in general $J(a^*)$ will be nondifferentiable.

In view of this, the ideas presented are by no means pretended to be conclusive, or suggested to be applicable. They are just a first exploration of the application of previous work done, to automatic control.

References

[1] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[2] L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, MAY 1993.

[3] D. Meltzer. On the expressibility of piecewise-linear continuous functions as the difference of two piecewise-linear convex functions. *Mathematical Programming Study*, 29:118–134, 1986.

[4] P. Pucar and J. Sjoberg. On the hinge-finding algorithm for hinging hyperplanes. *IEEE Transactions of Information Theory*, 44(3):1310–1319, MAY 1998.