# Constrained Fitting: A Gallery of Methods

Venkataram Edavamadathil Sivaram
veskimo123@gmail.com

Cupertino High School, CA

November 29, 2020

**Abstract**

Given a set of data, and a model with parameters which are subject to change, the goal of algorithms in the class of constrained fitting methods is to tune the parameters such that the model is as "accurate" as possible, where accuracy is a measure up to the analyst, and, most importantly, that the parameters satisfy a certain set of given constraints. In this paper, we consider specific variations of the constrained fitting problem and display a gallery of algorithms that work towards achieving this goal.

## 1 Introduction

A model in constrained fitting, with parameters $\boldsymbol{a}$, is subject to the constraints that restrict the parameters to a certain set $C(\boldsymbol{a})$, which we shall refer to as the *valid set* of $\boldsymbol{a}$. The general goal of constrained fitting algorithms is to generate an "accurate" model whose parameters, $\boldsymbol{a}$, are in $C(\boldsymbol{a})$. In most cases, this accuracy is measured with a *loss function*, $\mathcal{L}$.

In this paper, we shall restrict our attention to point-wise constrains, in which the valid set $C(\boldsymbol{a})$ is determined by a series of relations of the form

$$\forall i \in [1, n],\ f(x_i) \underset{i}{\sim} y_i.$$

Here, $D$ is a set of $n$ data points of the form $(x_i, y_i)$, and the relation can be arbitrary. In some of the algorithms, however, we restrict our attention further to the Four Relations (4R) problems, where the relations above are one of $>, \geq, <$ or $\leq$ .

We divide these problems even further: In the Monotone Constrains Problem (MCP), our constraints are of the form

$$\forall i \in [1, n],\ f(x_i) \sim y_i,$$

where $\sim$ is a relation which stays the same (hence the word monotone). On the other hand, we deal with the Free Constraints Problem (FCP), where this is not necessarily the case. Note that MCPs are a subset of FCPs.

In this paper, we consider four distinct algorithms for solving these problems:

- In Section 2 we go over the Support Polynomial (PSA) method, which constructs a polynomial model based on a single parameter. This algorithm, as we shall soon see, has its benefits against polynomial interpolation, but is not as powerful as gradient descent methods.
- In Section 3, we explore the more powerful gradient descent methods that can be used to solve this problem. In addition to the projected gradient descent method, we propose a new algorithm that is more specific to the 4R problems, and which can also be used on non-linear models.
- In Section 4 we introduce a set of functions with properties that allow us to create models satisfying 4R problems with ease and without much precomputation. We will see, however, that these functions have a drawback which can severely disrupt the accuracy of the model.
- Following the concepts and algorithms presented Section 4, in Section 5 we take a look at polar models which use similar functions to cluster data outside or inside the model.

# 2 Supported Polynomial (PSA) Method

The supported polynomial method, which we shall sometimes refer to as the PSA method, finds a linear relationship between the coefficients of a polynomial model using a selected subset of the given data points. It then uses these linear relationships to construct a restriction on the coefficients, from which it becomes easy to optimize the loss function, $\mathcal{L}$, defined as

$$\mathcal{L}(f) = \sum_{(x_i,y_i)\in D} (y_i - f(x_i;a))^2.$$

(Here, $a$ is the parameter of the model $f$.) This algorithm works not only for the 4R problems, but it also works whenever the relationship is some arbitrary point-wise relation.

## 2.1 Preliminaries

Let $D$ be the set of $m$ data points,

$$D := \{(x_0,y_0),(x_1,y_1),\ldots,(x_m,y_m)\},$$

and $\boldsymbol{y}$ be vector of all the $y$-values in the set,

$$\boldsymbol{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

Let $f(x)$ be the $n$th degree polynomial

$$f(x) = \sum_{k=0}^{n} p_k x^k.$$

This will be the model which we wish to fit above or below the set $D$. The parameters here are the coefficients $p_k$, and for convenience, let $\boldsymbol{p}$ be the vector with these parameters are components,

$$\boldsymbol{p} = \begin{bmatrix} p_n \\ p_{n-1} \\ \vdots \\ p_0 \end{bmatrix}.$$

**Definition.** *Denote $S_f$ to be the **support** of $f$, which is the subset of the points in $D$ which $f$ strictly goes through.*

The support of $f$ contains all the points which are used to create the linear relationships between the coefficients of $f$. As we will soon see, with $f$ being a degree $n$ polynomial with $n+1$ coefficients, we will need $n$ points in $S_f$ to construct a relationship between one coefficient and the rest.

**Definition.** *For an arbitrary set of points, $D$, denote $P(D)$ to be the **power matrix** or **matrix of powers over** $D$, defined as*

$$P = \begin{bmatrix} x_0^n & x_0^{n-1} & \dots & 1 \\ x_1^n & x_1^{n-1} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_m^n & x_m^{n-1} & \dots & 1 \end{bmatrix}$$

*or more concisely,*

$$[P]_{i,j} = x_i^j,$$

*where $(x_i, y_i)$ are points in $D$.*

The goal of the algorithm is to maintain satisfaction of constrains,

$$P(D)\boldsymbol{p} \sim \boldsymbol{y},$$

where $\sim$ represents the component wise relation, while trying to minimize the loss function,

$$\|\boldsymbol{y} - P(D)\boldsymbol{p}\|_2,$$

where $\|\boldsymbol{x}\|_2$ is the Euclidean norm of $\boldsymbol{x}$. (Note that this representation of the loss function is equivalent to one mentioned previously.)

There are two parts to this algorithm; the first part determines the substitution matrix $S$, which allows us to evaluate $\boldsymbol{p}$ from the two element vector

$$\boldsymbol{a} = \begin{bmatrix} p_n \\ 1 \end{bmatrix}.$$

The second part of the algorithm is responsible for determining $C(p_n)$, which allows the model $f$ under $p_n$ to adhere to the given point-wise constrains.

The third and final part of the algorithm is responsible for minimizing the value of

$$\|\boldsymbol{y} - P(D)\boldsymbol{p}\|_2,$$

by assessing the global optimum and the boundary points of the domain of $p_n$.

## 2.2 Evaluation of $S$, the Substitution Matrix

As pointed out in Section 2.1, the purpose of determining the substitution matrix $S$ is to be able to compute $\boldsymbol{p}$ with a single variable,

$$\boldsymbol{p} = S\boldsymbol{a} = S \begin{bmatrix} p_n \\ 1 \end{bmatrix}.$$

This way, we can represent the constraints of $\boldsymbol{p}$ through a constraint on $p_n$. Note that there is nothing particularly special about the coefficient $p_n$; any coefficient would work just as fine. Our general goal with $S$ is to shrink the work on multiple variables to a job on a single variable.

To determine $S$, we need $n$ linear equations. These equations, as pointed out in Section 2.1 come from the support, $S_f$. Denote the points in $S_f$ as of the form $(u_i, v_i)$, for $i \in [0, n)$, and let $\hat{\boldsymbol{y}}$ be the vector

$$\hat{\boldsymbol{y}} = \begin{bmatrix} v_n \\ v_{n-1} \\ \vdots \\ v_n \end{bmatrix}.$$

By our definition of the support, we required that $f$ goes through the points in $S_f$. In other words, we must have

$$P'\boldsymbol{p} = \hat{y},$$

where $P' = P(S_f)$. We shall split this as

$$P'_{(-1)}\boldsymbol{p}_{(-1)} + P'_{(+1)}p_n = \hat{y},$$

where $P'_{(-1)}$ is $P'$ without its first column, $P'_{(+1)}$ is the first column of $P'$ and $\boldsymbol{p}_{(-1)}$ is $\boldsymbol{p}$ without its first component. Elementary manipulations give us

$$P'_{(-1)}\boldsymbol{p}_{(-1)} = \begin{bmatrix} -P'_{(+1)} & \hat{\boldsymbol{y}} \end{bmatrix} \boldsymbol{p}_{(-1)} = S'\boldsymbol{a}.$$

Our goal now to isolate $\boldsymbol{p}_{(-1)}$ by transforming $P'_{(-1)}$ to the $n \times n$ identity matrix, $I_n$.

One can perform this transformation with an extension of Gauss-Jordan elimination; on the first pass, we turn $P'_{(-1)}$ into an upper triangular matrix, $U$, using Gauss Jordan's. In the second pass, we transform $U$ into a diagonal

matrix $D$ using Gauss-Jordan's again, from bottom upwards. Finally, to turn $D$ into the identity matrix $I_n$, we simply scale each row appropriately. Note that this process alters the elements of the augmented matrix

$$\begin{bmatrix} -P'_{(-1)} & \hat{\boldsymbol{y}} & S' \end{bmatrix},$$

and yields the matrix

$$\begin{bmatrix} I_n & (\hat{\boldsymbol{y}})' & S_{(-1)} \end{bmatrix},$$

in $\mathcal{O}(n^3)$ operations.

The result of this transformation is that we can turn the equation

$$P'_{(-1)}\boldsymbol{p}_{(-1)} = S'\boldsymbol{a}$$

into

$$\boldsymbol{p}_{(-1)} = S_{(-1)}\boldsymbol{a}.$$

From here, to get $S$ from $S_{(-1)}$, we simply append the row $\begin{bmatrix} 1 & 0 \end{bmatrix}$ to the top,

$$S = \begin{bmatrix} 1 & & 0 \\ & S_{(-1)} & \end{bmatrix}.$$

## 2.3 Determination of the Range on $p_n$

With the substitution matrix $S$, we can proceed to determine the restrictions on $p_n$. We can rewrite

$$P(D)S\boldsymbol{a} \sim \boldsymbol{y}$$

as

$$P(D)S\boldsymbol{a} = P(D)S \begin{bmatrix} p_n \\ 1 \end{bmatrix} \sim y.$$

We can then multiply the matrices $P(D)$ and $S$ to produce a new matrix $Q$, such that

$$Q \begin{bmatrix} p_n \\ 1 \end{bmatrix} \sim \boldsymbol{y}.$$

This becomes

$$Q'p_n + Q'' \sim \boldsymbol{y},$$

where $Q'$ and $Q''$ are the first and second columns of $Q''$. Subtraction by $Q''$ finally yields

$$Q'p_n \sim \boldsymbol{c},$$

where $\boldsymbol{c}$ is the difference $\boldsymbol{y} - Q''$.

From here, we can assess each component of $Q'$ and $\boldsymbol{c}$, along with the corresponding relation, to get the restrictions of $p_n$. If, in the process of determining the valid set of $p_n$, we reach a contradiction, it follows that there exists no model $f$ that goes through each point in $S_f$ while maintaining the required constraints over $D$.

## 2.4 Optimizing $p_n$

With the suitable range for $p_n$, our next step is to find the optimum such $p_n$. Specifically, we want to minimize the loss function,

$$\|\boldsymbol{y} - P\boldsymbol{p}\|_2 = \|\boldsymbol{c} - Q'p_n\|_2.$$

This is the same as minimizing the square of the norm,

$$\mathcal{L}(p_n) = \|\boldsymbol{c} - Q'p_n\|_2^2.$$

Recall that the vertex of a parabola, which is its only extrema, of the form $ax^2 + bx + c$ is $-\frac{b}{2a}$. With the basis of the above equation, it follows that

$$a = \sum_{k=1}^{m} (Q'_k)^2$$

and

$$b = -2\sum_{k=1}^{m} Q'_k \boldsymbol{c}_k,$$

where the subscripts denote the $k$th component of the vector. Thus, it follows that the global optimum of $\mathcal{L}$ is

$$\lambda = \frac{\sum_{k=1}^{m} Q'_k \boldsymbol{c}_k}{\sum_{k=1}^{m} (Q'_k)^2}.$$

To translate $\lambda$ into $p_n$, we need to project $\lambda$ onto the valid set of $p_n$. If $\lambda$ already exists in $C(p_n)$, then it follows that $p_n = \lambda$ is our optimal $p_n$. If $C(p_n)$ is a closed range $[a, b]$, then we have

$$p_n = \begin{cases} \lambda & \text{if } \lambda \in [a, b] \\ a & \text{if } |\lambda - a| < |\lambda - b| \\ b & \text{if } |\lambda - b| \leq |\lambda - a| \end{cases}.$$

On the other hand, if $C(p_n)$ is an open interval such as $(a, b)$, then we have

$$p_n = \begin{cases} \lambda & \text{if } \lambda \in (a, b) \\ \inf(a, b) & \text{if } |\lambda - a| < |\lambda - b| \\ \sup(a, b) & \text{if } |\lambda - b| \leq |\lambda - a| \end{cases}.$$

With the optimal $p_n$, we can use the substitution matrix $S$ to derive $\boldsymbol{p}$,

$$\boldsymbol{p} = S \begin{bmatrix} p_n \\ 1 \end{bmatrix}.$$

### 2.5 Trade-Offs and Extensions

The advantage of the PSA method is that it generates a polynomial that satisfies the required point-wise constraints, assuming that no contradiction arose. Using Horner's method, one can compute values of the polynomial in $\mathcal{O}(n)$ operations, where $n$ is the degree of the polynomial. This has better performance than the $\mathcal{O}(n \log n)$ complexity for Lagrange interpolation polynomials, as mentioned in [7].

As a result, if one requests a model whose support is the same as the given data, they will receive a polynomial that can be computed faster than the one generated by Lagrange interpolation, and which achieves the same accuracy. So, excluding the time it takes to generate the model, the PSA method is superior to Lagrange interpolation.

The time which the PSA method takes to generates the model is dominated by the computation of the substitution matrix. Thus, it follows that the PSA method takes $\mathcal{O}(n^3)$ time to generate the model.

However, the fact that the PSA method requires a support is its largest weakness. Although the study is still in progress, there is no method to generate an appropriate support of a given size. This implies that the PSA method cannot be used autonomously. While the PSA method is deterministic, as described in Section 1, one should note that it does not necessarily generate the most accurate models over data set and constraints, a consequence of having to select a support.

Currently, there is a crude way to extend the PSA method into higher dimensions, and it uses the same idea as the two-dimensional PSA method. Let $x_1, x_2, \ldots, x_n$ be the variables of our model $f$. Let $L_i$ be a function which evaluates a polynomial term with these variables,

$$L_i(x_1, x_2, \ldots, x_n) = \prod_{j=1}^{k} x_{a_j}.$$

Here, $\langle a_i \rangle$ is a sequence of $k$ integers from 1 to $n$. The model $f$ will have the general form of

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{n} p_i L_i(x_1, x_2, \ldots, x_n),$$

for some "degree" $n$, and as with the two-dimensional PSA method, we find a linear relationship with some $p_i$ and the rest, using the given support of $n$ points in $D$.

## 3 The Gradient Descent Methods

As noted in the previous section, the PSA method will yield a deterministic result, but this result does not always give the optimal solution. (In general,

algorithms which reduce the dimension of the original problem will fail to given globally optimal solutions.)

In this section, we cover the more general spectrum of algorithms, the gradient descent methods, which allows us to tackle optimizing models with multiple parameters even when there are constraints on these parameters.

## 3.1 The Projected Gradient Descent Method

The projected gradient descent method is the most popular method used for constrained global optimization problems. What distinguishes this algorithm from vanilla gradient descent is the fact that the updates are projected onto the valid set of the parameters, $C(\boldsymbol{a})$. Furthermore, it is required that $C(\boldsymbol{a})$ be a convex set.

As it uses gradients of function, the projected gradient descent algorithm works for multivariate functions $f : \mathbb{R}^n \to \mathbb{R}^m$. Each point in our data set thus corresponds to a pair $(\boldsymbol{x_i}, \boldsymbol{y_i})$, where $\boldsymbol{x_i} \in \mathbb{R}^n$ and $\boldsymbol{y_i} \in \mathbb{R}^n$.

At each iteration of the projected gradient descent algorithm, one computes the gradient of the loss function, $\nabla\mathcal{L}$. Then, as mentioned earlier, to ensure that the new parameters $\boldsymbol{a} + \nabla\mathcal{L}$ are in $C(\boldsymbol{a})$, it is projected onto $C(\boldsymbol{a})$. Thus, each update in the projected gradient descent method has the form

$$\boldsymbol{a} \leftarrow \text{proj}_{C(\boldsymbol{a})}(\boldsymbol{a} + \nabla\mathcal{L}).$$

In simple terms, the projection function $\text{proj}_{C(\boldsymbol{a})}(\boldsymbol{x})$ yields the element in $C(\boldsymbol{a})$ that is "closest", in some sense, to $\boldsymbol{x}$.

For more information on this algorithm, see [3].

## 3.2 The Four Relations Gradient Descent (4RGD) Method

The projected gradient descent algorithms work well for many practical applications and situations. However, as mentioned in the previous section, it does not directly deal with valid sets that are not convex. Such situations are likely to occur when dealing with non-linear models, and thus a few additional tweaks are required to have the algorithm solve constrained global optimizations problems for non-linear models. In this section, we present an algorithm that goes around this problem for constrains in the 4R class.

We assume that the initial parameters, $\boldsymbol{a}$, is already in $C(\boldsymbol{a})$. As with the projected gradient descent algorithm, we must compute the gradient $\nabla\mathcal{L}$ at each iteration. We then update our parameters as follows:

$$\boldsymbol{a} \leftarrow \boldsymbol{a} + \lambda\nabla\mathcal{L}.$$

Here, $\lambda$ is a scaling factor in the interval $[0, 1]$ which will guarantee than applying this update will not break the constrains. To be more precise, $\lambda$ is the largest constant in $[0, 1]$ which satisfies

$$\boldsymbol{a} + \lambda\nabla\mathcal{L} \in C(\boldsymbol{a}).$$

Note that the inclusion of $\lambda$ represents the projection step in the projected gradient descent algorithm represented earlier.

When the constrains are 4R, computation of $\lambda$ can be done quite simply. For this, however, we shall make the following definition:

**Definition.** *Let the* ***individual loss function****, $\mathcal{L}_i$ on the model $f$ and data set $D$ be defined as*

$$\mathcal{L}_i(\boldsymbol{a}) = \pm(\boldsymbol{y_i} - f(\boldsymbol{x_i}; \boldsymbol{a})).$$

*The sign is determined by the condition*

$$\mathcal{L}_i(\boldsymbol{a}) \geq 0 \iff f(\boldsymbol{x_i}; \boldsymbol{a}) \underset{i}{\sim} y_i.$$

*Here, the comparison $\geq$ is applied per component, meaning that each component of $\boldsymbol{y_i}$ should be non-negative. Thus, if $\mathcal{L}_i(\boldsymbol{a})$ is negative, then it implies that the condition does not hold.*

If $\mathcal{L}_i(\boldsymbol{a} + \nabla\mathcal{L})$ is non-negative to start with, then we set $\lambda$ to 1, so that the convergence of the algorithm is as high as possible.

If this is not the case, then because we assumed that $\mathcal{L}_i(\boldsymbol{a})$ is non-negative, it follows that there is a root of $\mathcal{L}_i$ on the line segment from the point $\boldsymbol{a}$ and the point $\boldsymbol{a} + \nabla\mathcal{L}$ (See Figure 1), by the Intermediate Value Theorem of calculus. It follows, then, that our goal should be to locate this root.

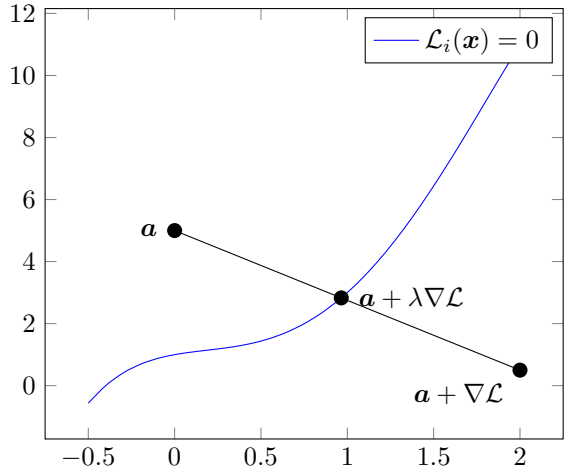


Figure 1: Optimal choice of $\lambda$ as the root of $\mathcal{L}_i$ which lies on the vector from $\boldsymbol{a}$ to $\boldsymbol{a} + \nabla\mathcal{L}$

The most appropriate algorithm for such a task are the bracketing methods, such as the Bisection Method [6], False Position [2], Ridder's [5], Brent's [1],

and many others. With these algorithms, we wish to refine the interval in which we are certain the root is in. Ideally, we would like to keep refining the interval $[\boldsymbol{c}, \boldsymbol{d}]$ such that $\mathcal{L}_i(\boldsymbol{c})$ is negative and $\mathcal{L}_i(\boldsymbol{d})$ is positive. This way, when we have reached a satisfactory amount of precision with respect to bracketing the root, we can set $\boldsymbol{a} + \lambda \nabla \mathcal{L}$ as $\boldsymbol{d}$, and the 4R conditions will be met.

To summarize, we present the 4RGD algorithm below:

**Algorithm 1:** The 4RGD Algorithm

**Data:** Model $f(\boldsymbol{x}; \boldsymbol{a})$ and initial estimate of parameters $\boldsymbol{a}$

**Precondition:** $\boldsymbol{a}$ satisfies $\mathcal{L}_i(\boldsymbol{a}) \geq 0$ for all $i \in [1, n]$

**Result:** Optimal parameters $\boldsymbol{a}$

**for** $k \leftarrow 1$ **to** rounds **do**
- $\boldsymbol{g} \leftarrow \nabla \mathcal{L}$
- $\hat{\boldsymbol{a}} \leftarrow \boldsymbol{a} + \boldsymbol{g}$
- `// Minimum` $\lambda$ `value`
- $\Lambda = 1$
- **for** $i \leftarrow 1$ **to** $n$ **do**
  - `// If the below condition is not met, i.e.` $\mathcal{L}_i(\hat{\boldsymbol{a}}) \geq 0$, `then there is no need to change the currrent` $\Lambda$
  - **if** $\mathcal{L}_i(\hat{\boldsymbol{a}}) < 0$ **then**
    - $[\boldsymbol{c}, \boldsymbol{d}] \leftarrow$`BracketRoot(`$[\boldsymbol{a}, \hat{\boldsymbol{a}}]$, iters, eps`)`
    - $\lambda \leftarrow \frac{\boldsymbol{d} - \boldsymbol{a}}{\boldsymbol{g}}$
    - $\Lambda \leftarrow \min(\lambda, \Lambda)$
- $\boldsymbol{a} \leftarrow \boldsymbol{a} + \Lambda \boldsymbol{g}$

**return** $\boldsymbol{a}$

In the above algorithm, we assume that the user has defined each $\mathcal{L}_i$ function, as well as the cost function $\mathcal{L}$. The `BracketRoot` algorithm would be implemented similar to below, using the Bisection method:

**Algorithm 2:** The `BracketRoot` algorithm

**Data:** Two points, $\boldsymbol{a}$ and $\boldsymbol{b}$, and a continuous function $f$

**Precondition:** $f(\boldsymbol{a}) \leq 0$ and $f(\boldsymbol{b} \geq 0$

**Result:** A more refined range $[\boldsymbol{c}, \boldsymbol{d}]$ such that $f(\boldsymbol{c}) \leq 0$ and $f(\boldsymbol{d}) \geq 0$

**for** $k \leftarrow 1$ **to** iters **do**
  **if** $\|\boldsymbol{a} - \boldsymbol{b}\| <$ eps **then**
    **return** $[\boldsymbol{a}, \boldsymbol{b}]$
  $\boldsymbol{m} \leftarrow \frac{\boldsymbol{a}+\boldsymbol{b}}{2}$
  **if** $f(\boldsymbol{m}) \geq 0$ **then**
    $\boldsymbol{b} \leftarrow \boldsymbol{m}$
  **else**
    $\boldsymbol{a} \leftarrow \boldsymbol{m}$

**return** $[\boldsymbol{c}, \boldsymbol{d}]$

In the above algorithms, we assume that the user has specified `rounds`, `iters` and `eps` as global variables; `rounds` should be the desired number of iterations of the 4RGD parameter update loop, `iters` is the maximum number of iterations that should be done in the `BracketRoot` algorithm, and `eps` is the desired precision or length of interval returned by the `BracketRoot` algorithm.

# 4 The Spike Methods

The spike methods, which we describe in the following sections, use the properties a certain class functions called *spike functions*. The models are created by adding these spike functions together. In light of this, we shall define the resolution of a model to be the number of spikes that must be summed to get the model. As we shall see, a good model trade resolution with computation time as well as overall error.

These algorithms can be used to create models that satisfy 4R MCP conditions, with processing times that vary among the algorithms. Before moving on, we shall state that we assume that no point has a null value; that is, there is no $(x_i, y_i)$ in $D$ such that $y_i$ is 0.

## 4.1 Spike Functions

We begin with a general definition of a spike function. (We shall omit the word "function" when referring to spike functions from now on.)

**Definition.** *A continuous, differentiable function $\psi : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a **spike** if it satisfies the following properties.*

1. *There exists a unique point $(h, k)$, called the **apex** of $\psi$, such that $\psi(h) = k$ and $\max \psi(x) = k$. (Spike property)*

2. *For all real $x$, we have $\psi(x+h) = \psi(x-h)$, where $h$ is defined as above. (Apex-symmetric)*

3. *For all real $a$ and $b$, if $|h-a| < |h-b|$, then $\psi(a) > \psi(b)$. (Decaying)*

Examples of spikes are $e^{-x^2}, \text{sech}(x)$ and $\frac{1}{1+x^2}$. (See Figure 2 for their graphs.) As we shall soon see, these spikes have additional properties.

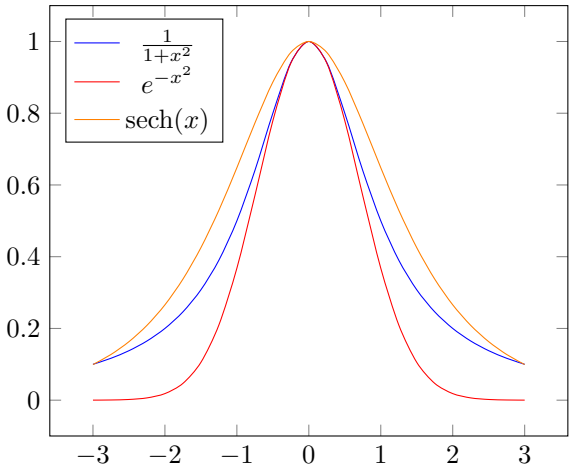


Figure 2: Examples of spike functions

We also introduce a few sets to accompany our understanding of the spike functions.

**Definition.** *Let $\psi$ be a spike. Then, define the following sets as specified:*

1. *Let $\mathbb{S}$ be the set of all spikes.*

2. *Let $\mathbb{B}_\psi$ be the set of spikes generated by $\psi$ and which is closed under the map*

$$\begin{aligned} f : \mathbb{B}_\psi &\to \mathbb{B}_\psi \\ \psi &\mapsto k\psi(a(x-h)), \end{aligned} \tag{1}$$

*where $a, h$ and $k$ are real numbers such that $k$ and $a$ are non-zero. This is the set of spikes with the same **base** as $\psi$, or, equivalently, the set of **bases** of $\psi$. If $\psi$ and $\theta$ have the same base, then we denote $\psi \equiv \theta$. Note that $\equiv$ is an equivalence relation.*

3. *Let $\mathbb{P}_\psi$ be the set of spikes generated by $\psi$ and which is closed under the map*

$$\begin{aligned} f : \mathbb{P}_\psi &\to \mathbb{P}_\psi \\ \psi &\mapsto k\psi(x-h), \end{aligned} \tag{2}$$

*where $h$ and $k$ are real numbers such that $k$ is non-zero. This is the set of **phases** of $\psi$ – we may also refer to this set as a **phase group** when $\psi$ is not specified. If $\psi$ and $\theta$ have the same phase, we say that $\theta$ is a **phase** of $\psi$ and we denote this by $\psi \sim \theta$. Note that $\sim$ is an equivalence relation. Additionally, let $\tau_\psi$ be the map*

$$\begin{aligned} \tau_\psi : \mathbb{R}^2 &\to \mathbb{P}_\psi \\ (h,k) &\mapsto k\psi(x-h), \end{aligned} \tag{3}$$

*which generates the phases of $\psi$. In fact, if $\psi \sim \theta$, then we will always have that $\tau_\theta$ generates $\mathbb{P}_\psi$, and furthermore, that $\mathbb{P}_\psi$ is identical to $\mathbb{P}_\theta$.*

While these sets are convenient on their own, it would be more useful to be able to uniquely identify them. Thus, we create the following terminology:

**Definition.** *Define the **kernel**, $\Phi$, of the set $\mathbb{P}_\psi$ to be the spike in this set such that the apex of $\Phi$ occurs precisely at $(0,1)$.*

Figure 2 above shows three examples of spikes which are also kernels.

**Definition.** *Define the **foundation**, $\Psi$, of the set $\mathbb{B}_\psi$ to be the spike which is the kernel of some phase group and which satisfies*

$$\int_{-\infty}^{\infty} \Psi(x)dx = 1.$$

As an example, the foundation of the bases of $e^{-x^2}$ is $e^{-\pi x^2}$.

## 4.2 The Lazy Method

The properties of the spike functions allow us to solve MCP on sparse data very well, and without much work.

If $\Phi$ is the kernel of our desired phase group, then the model generated by $D$ over this kernel (or over these phases) is

$$f(x) = \sum_{(x_i,y_i)\in D} \tau_\Phi(x_i, y_i) \tag{4}$$

when the 4R relation is $>, \geq$, and

$$f(x) = \max y_i - \sum_{(x_i,y_i)\in D} \tau_\Phi(x_i, y_i - \max y_i) \tag{5}$$

when the relation is otherwise. Note that at least one point term in the summation in Equation 5 vanishes because $y_i - \max y_i$ becomes 0. Following this, we can make a term vanish in the model in Equation 4 by shifting up by the minimum $y_i$,

$$f(x) = \min y_i + \sum_{(x_i, y_i) \in D} \tau_\Phi(x_i, y_i - \min y_i). \tag{6}$$

As an example, see Figure 3, which demonstrates how these models appear.

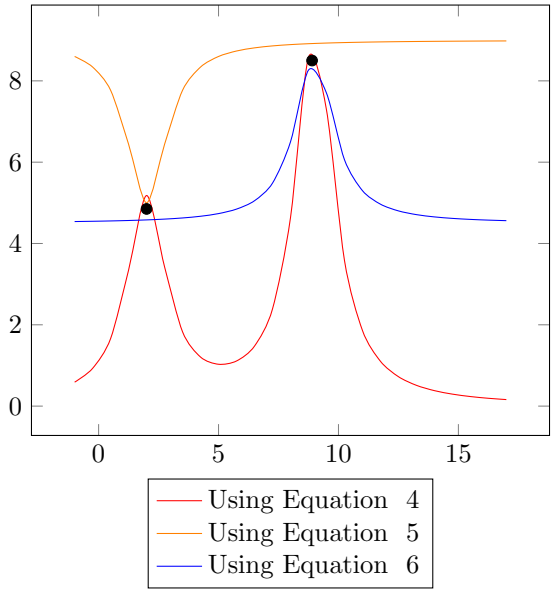


Figure 3: The Lazy Method applied to a sparse set of points

The naivety in this method, however, arises when our set of data becomes more dense – when the data points are closer to each other with respect to the inputs. See Figure 4; the closer the points are, the higher their combined peak is, and the more inaccurate and undesirable the model becomes. In the next section, we cover this effect more formally, and in the section following the next, we discuss how to avoid such issues.

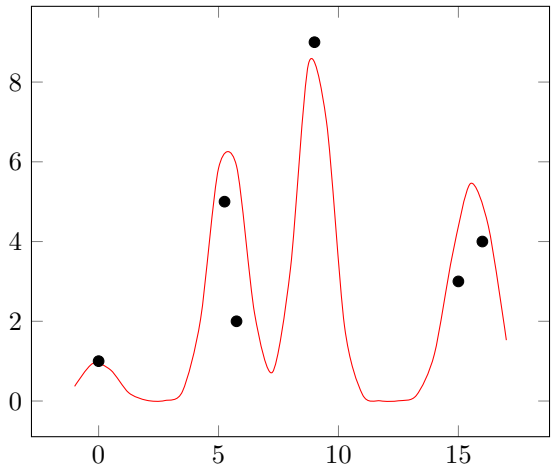


Figure 4: The Lazy Method applied to a dense set of points

### 4.3 Disturbance

The effect seen in Figure 4 is what we shall refer to as *disturbance*. It is a consequence of a spikes relatively slow rate of decrease. We formalize this rate with the following definition:

**Definition.** *For a kernel* $\Phi$*, of some arbitrary phase group, define* $\eta_\Phi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$*, the disturbance of* $\Phi$ *with* ***epoch*** $n$*, to be following:*

$$\eta_\Phi(n) = \frac{\Phi(n)}{n^2}.$$

*Furthermore, define the* ***unit disturbance*** *to be* $\zeta(\Phi)$*, which is equal to* $\eta_\Phi(1) = \Phi(1)$.

The function $\eta_\Phi$ is a measure of the amount of disturbance a spike creates at an epoch-distance (of $n$) from the apex. The reason we define it only for kernels is that the disturbances of other spikes in the phase group are simply scalar multiples of those of $\Phi$. Thus, we need only to define the disturbance for kernels, and the definition becomes clear for all spikes in $\mathbb{S}$. Also note that while $\eta_\Phi$ is a function of the epoch, $\zeta$ yields a numerical value.

As an example, consider the spikes $f(x) = e^{-x^2}$ and $g(x) = \frac{1}{1+x^2}$:

$$\eta_f(n) = \frac{e^{-n^2}}{n^2} = \frac{1}{n^2 e^{n^2}} = \Theta\left(\frac{1}{n^2 e^{n^2}}\right) \qquad \eta_g(n) = \frac{\frac{1}{1+n^2}}{n^2} = \frac{1}{n^2(1+n^2)} = \Theta\left(\frac{1}{n^4}\right)$$

$$\zeta(f) = f(1) = \frac{1}{e} \qquad \zeta(g) = g(1) = \frac{1}{2}$$

If two spikes, $f$ and $g$, have the same apex, then it follows that if $\zeta(f) > \zeta(g)$ then it implies that $\eta_f(n) > \eta_g(n)$ for all real $n \geq 1$, due to the decaying property of spike functions. In general, by the same reasoning, if $\eta_f(k) > \eta_g(k)$, then $\eta_f(n) > \eta_g(n)$ for all $n \geq k$. Therefore, it follows that $e^{-x^2}$ creates less disturbance than $\frac{1}{1+x^2}$. Thus, $\zeta$ and $\eta$ are both metrics which compare spikes.

To summarize the metrics and intuitions provided thus far, we compare the foundations of the spikes $e^{-x^2}$, $\frac{1}{1+x^2}$ and $\text{sech}(x)$ in Table 4.3 below. One can see that the foundation for $e^{-x^2}$ will create the least amount of disturbance for epochs greater than unit length, while the foundation of $\text{sech}(x)$ creates the largest disturbance.

| Spike, $\psi$ | $e^{-x^2}$ | $\frac{1}{1+x^2}$ | $\text{sech}(x)$ |
|---|---|---|---|
| Foundation, $\Psi$ | $e^{-\pi x^2}$ | $\frac{1}{1+(\pi x)^2}$ | $\text{sech}(\pi x)$ |
| Disturbance, $\eta_\Psi(n)$ | $\Theta\left(\frac{1}{e^{n^2}n^2}\right)$ | $\Theta\left(\frac{1}{n^4}\right)$ | $\Theta\left(\frac{2}{(e^{-n}+e^n)n^2}\right)$ |
| Unit Disturbance, $\zeta(\Psi)$ | $\frac{1}{e^\pi} \approx 0.04321$ | $\frac{1}{1+\pi^2} \approx 0.09199$ | $\frac{2e}{e^2+1} \approx 0.64805$ |

Table 1: Comparing Properties of Foundations

In practice, spikes with the foundation of $\frac{1}{1+x^2}$ should be used for computation, since they balance disturbance and computation time; they are relatively cheap to compute compared to the other spikes, while their disturbance is somewhat in between.

## 4.4 Working Around Disturbance

Disturbance appeared as a large issue when attempting to use the Lazy Method. However, there are ways in which we can acknowledge disturbance while creating

effective models. We propose one such method in this section.

One notes that disturbance accumulates as the number of spikes accumulates. Thus, we first attempt to reduce the number of spikes that we use. Formally, we would like to construct a sufficiently small set, $S$, of triples $(h_i, k_i, \eta_i)$. Each triple represents the spike $k_i\Psi(\eta_i(x - h_i))$, where $\Psi$ is the foundation of an arbitrary set of bases. The addition of the extra parameter $\eta_i$ will allow us to control the amount of disturbance the spike creates.

The creation of $S$ differs with respect to the monotone relation:

**Case 1.** All relations are either $>$ or $\geq$

Fix some value $\eta$. We will use this value as $\eta_i$ for all spikes in $S$. Let $y$ be the minimum value of $y_i$ for each point $(x_i, y_i)$ in $D$. Discard the corresponding spike from consideration.

Pick a point $(x_i, y_i)$ in $D$ which has the maximum $y_i$ value. Add the triple $(x_i, y_i - y, \eta)$ to $S$. Remove, from consideration, the points in $D$ which are under the added spike. Call this new set $D'$, and repeat the procedure with $D'$ instead of $D$ until there are no more points left to consider.

**Case 2.** All relations are either $<$ or $\leq$

Again, we fix a value $\eta$ which we shall use as $\eta_i$ for each spike in $S$. Let $y$ be the maximum value of $y_i$ for each point $(x_i, y_i)$ in $D$. Discard the corresponding spike from consideration.

Pick a point $(x_i, y_i)$ in $D$ which has the minimum $y_i$ value. Add the triple $(x_i, y_i - y, \eta)$ to $S$. Remove, from consideration, the points in $D$ which are above the added spike. Call this new set $D'$, and repeat the procedure with $D'$ instead of $D$ until there are no more points left to consider.

The corresponding algorithms for can be written as follows:

**Algorithm 3:** Construction of $S$ for *Case 1.*

**Data:** Set of points $D$, and fixed value $\eta$

**Result:** Set of spike parameters satisfying *Case 1.*

$S \leftarrow \varnothing$

$(x, y) \leftarrow \underset{(x_i, y_i) \in D}{\arg\min}\, y_i$

$D \leftarrow D \setminus (x, y)$

**while** $\#D \neq 0$ **do**

  $(h, k) \leftarrow \underset{(x_i, y_i) \in D}{\arg\max}\, y_i$

  $S \leftarrow S \cup (h, k - y, \eta)$

  **for** $(x, y) \in D$ **do**

    **if** $y > k\Psi(\eta(x - h))$ **then**

      $D \leftarrow D \setminus (x, y)$

**return** $S$

**Algorithm 4:** Construction of $S$ for *Case 2.*

**Data:** Set of points $D$, and fixed value $\eta$

**Result:** Set of spike parameters satisfying *Case 2.*

$S \leftarrow \varnothing$

$(x, y) \leftarrow \underset{(x_i, y_i) \in D}{\arg\max}\, y_i$

$D \leftarrow D \setminus (x, y)$

**while** $\#D \neq 0$ **do**

  $(h, k) \leftarrow \underset{(x_i, y_i) \in D}{\arg\min}\, y_i$

  $S \leftarrow S \cup (h, k - y, \eta)$

  **for** $(x, y) \in D$ **do**

    **if** $y > k\Psi(\eta(x - h))$ **then**

      $D \leftarrow D \setminus (x, y)$

**return** $S$

The difference between the algorithms for these two cases is very minor. (See the algorithms above; their differences have been highlighted.)

Now, we construct our model, $f$ as the sum of all the spikes in $S$, added to

$y$. Thus, our model should be

$$f(x) = y + \sum_{(h_i,k_i,\eta_i)\in S} k_i \Psi(\eta_i(x - h_i)).$$

The parameters of this model, $\boldsymbol{a}$, shall consist of all $k_i$ and $\eta_i$. Note that because this model satisfies the MCP condition (by construction of $S$), we have $\boldsymbol{a} \in C(\boldsymbol{a})$. Because we have an initial estimate of the parameters which already exists in $C(\boldsymbol{a})$, we can apply 4RGD to finish the job by optimizing the parameters of $f$.

Although the main goal of this data pre-processing is to reduce the number of spikes, and therefore the amount the disturbance emitted, its specific structure yields an initial model that satisfies the 4R MCP conditions. As mentioned in Section 3.2, this is necessary for the functioning of the 4RGD algorithm, which we then use to optimize the model.

## 4.5 Applications to Classical Regression

As we have seen, spike functions are quite useful for creating models or foundations of models that satisfy the 4R MCP conditions. However, due to their nice properties, it is also possible to create models for classical regression problems. We cover how this can be done in this section.

### 4.5.1 The Lazy Method and Gradient Descent

For an arbitrary set of points, $D$, we can construct our model as

$$f(x) = \sum_{(x_i,y_i)\in D} y_i \psi(x - x_i),$$

for some kernel $\psi$. However, as we saw in Section 4.2, this can yield a very bad model if the set of data is dense.

While this is true, it is still possible to salvage the model by multiplying it by a constant. Define the new model, $F$, as

$$F(x; k) = kf(x).$$

With $F$, we can apply vanilla gradient descent to the parameter $k$ to yield a model that is closer to the set of data, and more importantly, does not exhibit as much disturbance – see Figure **??**.

### 4.5.2 The Clustered Spike Method

We have already seen that adding many spikes together will at some point lead to undesirable models. The clustered spike method provides a way to select representative points with which to base our spikes.

The clustered spike method partitions the data set into $m$ of clusters, $D_i$, each being an interval of equal length. The representatives, $(u_i, v_i)$, are then chosen as the centroids of the points in each of these clusters.

We then calculate the average height of the centroids as $\mu$. With this information, we can construct the reduced model as

$$f(x) = \mu + \sum_{i=1}^{m} (v_i - \mu)\psi(x - u_i),$$

where $\psi$ is the kernel of our desired phase group

There are instance in which each cluster has a different number of points. In this case, we would want to prioritize the representative spikes whose clusters are larger. This leads us to create a weighted alternative of the above,

$$f(x) = \mu + \frac{m}{\#D} \sum_{i=1}^{m} \#C_i(v_i - \mu)\psi(x - u_i).$$

The multiplication by $m$ is added so that model does not flatten out.

Alternatively, we could parametrize the weights of each spike. In other words, we start with the model

$$f(x; \boldsymbol{k}) = \mu + \sum_{i=1}^{m} \boldsymbol{k}_i(v_i - \mu)\psi(x - u_i),$$

where $\boldsymbol{k}_i$ is the $i$th component of $\boldsymbol{k}$. We then apply gradient descent of this model with respect to $\boldsymbol{k}$ to find the optimal weights of each spike.

The advantage of picking representatives, as this method does, is that the resolution of the model decreases, making computation of our models faster.

## 4.6 Extensions to Higher Dimensions

We can extend the definitions made in earlier sections to higher dimensions. We begin with the definition of a multidimensional spike:

**Definition.** *A continuous, differentiable function $\psi : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, is a spike on $n$ dimensions if it satisfies the following properties.*

1. *There exists a unique point $(\boldsymbol{h}, k)$, where $\boldsymbol{h} \in \mathbb{R}^n$ and $k \in \mathbb{R}_{\geq 0}$, called the **apex** of $\psi$, such that $\psi(\boldsymbol{h}) = k$ and $\max_{\boldsymbol{x} \in \mathbb{R}^n} \psi(\boldsymbol{x}) = k$. (Spike property)*
2. *For all $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{R}^n$, if $\|\boldsymbol{a} - \boldsymbol{h}\| = \|\boldsymbol{b} - \boldsymbol{h}\|$, we must have $\psi(\boldsymbol{a}) = \psi(\boldsymbol{b})$, where $\boldsymbol{h}$ is defined as above. (Apex-symmetric)*
3. *For all $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{R}^n$, if $\|\boldsymbol{a} - \boldsymbol{h}\| < \|\boldsymbol{b} - \boldsymbol{h}\|$, then $\psi(\boldsymbol{a}) > \psi(\boldsymbol{b})$. (Decaying)*

Figure **??** shows the graphs of $e^{-(x^2+y^2)}$ and $\frac{1}{1+(x^2+y^2)}$; these are two examples of spikes on two dimensions.

Similarly, we can extend the definition of the sets we created and their unique representatives:

**Definition.** *Let $\psi$ be a spike on $n$ dimensions. Then, define the following sets as specified:*

1. *Let $\mathbb{S}^n$ be the set of all spikes on $n$ dimensions.*

2. *Let $\mathbb{B}_\psi^n$ be the set of bases, which is the set of spikes generated by $\psi$ and which is closed under the map*

$$\begin{aligned} f : \mathbb{B}_\psi &\to \mathbb{B}_\psi \\ \psi &\mapsto k\psi(a(x - \boldsymbol{h})), \end{aligned} \tag{7}$$

   *where $a$ and $k$ are real numbers such that $k$ and $a$ are non-zero, and $\boldsymbol{h}$ is in $\mathbb{R}^n$. The other notions from the one dimensional case are the same. The **foundation** of this set is the spike $\Psi$ which has an apex at the origin with unit and which satisfies*

$$\int_{\mathbb{R}^n} \cdots \int \Psi(\boldsymbol{x}) dx = 1.$$

3. *Let $\mathbb{P}_\psi$ be the phase group of $\psi$, which is the set of spikes generated by $\psi$ and which is closed under the map*

$$\begin{aligned} f : \mathbb{P}_\psi &\to \mathbb{P}_\psi \\ \psi &\mapsto k\psi(x - \boldsymbol{h}), \end{aligned} \tag{8}$$

   *where $h$ is in $\mathbb{R}^n$ and $k$ is a real number such that $k$ is non-zero. The other notions from the one-dimensional case are the same. The **kernel** of this set is the spike $\Phi$ which has an apex at the origin with unit height.*

Extending the definitions of disturbance is quite straightforward:

**Definition.** *For a kernel $\Phi$, of some arbitrary phase group, define $\eta_\Phi : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, the disturbance of $\Phi$ with **epoch** $\|\boldsymbol{n}\|$, to be following:*

$$\eta_\Phi(\boldsymbol{n}) = \frac{\Phi(\boldsymbol{n})}{\|\boldsymbol{n}\|^2}.$$

*Furthermore, define the **unit disturbance** to be $\zeta(\Phi)$, which is equal to $\eta_\Phi(\boldsymbol{u}) = \Phi(\boldsymbol{u})$, for some unit vector $\boldsymbol{u}$.*

Note that this definition is well defined since kernels are 0-symmetric.

With these extended definitions, one can quite trivially translate the one-dimensional algorithms mentioned in this section to higher dimensions.

# 5 The Polar Spike Method

The previous algorithms we have show-cased perform well in getting upper and lower bounds on the data. However, there are instances in which we wish to cluster the data, such that none or all of the points lie inside our model. In this section, we cover the polar spike methods, which are polar analogous of the spike methods that can be used to cluster data by "extruding" certain parts of polar curves to certain points.

This method, like the spike methods, can be used to solve the polar equivalents of 4R MCP, in which relations take the form

$$f(\theta_i) \underset{i}{\sim} r_i.$$

One should note that the domains of these polar curves will always be $[0, 2\pi]$, and that they will always be closed.

## 5.1 Extrusion of Polar Curves; The Polar Forms of Spike Functions

As mentioned earlier, the polar spike methods are based on being able to extrude parts of the curve. Before providing a formalization of how this is done, we give an example.

Suppose that at some $\theta = h$ on our polar curve, we wish to extrude the curve by $k$ units. We propose that the curve

$$r = k \sin f(\theta, h),$$

where $f$ is a Cartesian spike function with an apex at $(h, \frac{\pi}{2})$, performs our desired extrusion. An example of such a function $f$ is the spike

$$f(\theta, h) = \frac{\pi}{2} e^{-\alpha(\theta - h)^2},$$

where $\alpha$ is a parameter that can be tuned to adjust the disturbance created by the spike. Of course, this is not the only curve which performs our necessary extrusion; we would do just as well to replace sin with cos and adjust as needed.

We now formally describe how this done.

**Definition.** *A continuous differential function $\psi : [0, 2\pi] \to \mathbb{R}_{\geq 0}$ is a **polar spike** is it satisfies the following properties:*

1. *There exists a unique point $(\alpha, \rho)$, where $\alpha \in [0, 2\pi]$, called the **apex** of $\psi$, such that $\psi(\alpha) = k$ and $\max \psi(\alpha) = \rho$. (Spike property)*
2. *For all $\theta \in [0, 2\pi]$, we have $\psi(\theta + \alpha) = \psi(\theta - \alpha)$, where $\alpha$ is defined as above. (Apex-symmetric)*
3. *For all $\delta$ and $\omega$ in $[0, 2\pi]$, if $|\alpha - \delta| < |\alpha - \omega|$, then $\psi(\delta) > \psi(\omega)$. (Decaying)*

The difference between the definitions of the polar and Cartesian spikes is quite minuscule, which explains why we can construct polar spikes from Cartesian spikes, as done earlier.

The definitions of the sets $\mathbb{S}, \mathbb{B}_\psi$ and $\mathbb{P}_\psi$ for a polar spike $\psi$ are identical to the definitions for Cartesian spikes $\psi$. The same is true for kernels and foundations.

## 5.2 The Lazy Method for Polar Spikes

With definitions underway, we can present the first of the polar spike methods, which is the analog of the lazy method for Cartesian spikes.

If $\Phi$ is the kernel of our desired phase group, then the function

$$f(\theta) = \min r_i + \sum_{(r_i,\theta_i)\in D} \tau_\Phi(r_i - \min r_i, \theta_i)$$

serves a model which satisfies the MCP for the relations $>$ and $\geq$ . For the relations $<$ and $\leq$, we would instead use the model

$$f(\theta) = \max r_i - \sum_{(r_i,\theta_i)\in D} \tau_\Phi(r_i - \max r_i, \theta_i).$$

However, the issue of disturbance from performing the lazy method for Cartesian spikes is still relevant with polar spikes, and even more so when the relations are $<$ or $\leq$ . (There are cases in which, for a certain $\theta$, our model will yield a negative radius, which could lead to drastic results.)

## 5.3 4RD and the Lazy Method for Polar Spikes

As with the Cartesian lazy method, there is a very similar manner in which we can avoid the mishaps of disturbance. With Cartesian spikes, we applied the 4RGD algorithm as an additional step of the lazy method to yield a more accurate model. For the polar lazy method, we can do something very similar. The only change from the Cartesian algorithm is the selection of $S$, which only requires exchanging $x$ and $\theta$ and $y$ and $r$.

Then, with our selected polar foundation $\Psi$, we construct the basis of our model,

$$f(\theta) = r + \sum_{(\theta_i,r_i,\eta_i)\in S} r_i \Psi(\eta_i(\theta - \theta_i)),$$

and apply the 4RGD algorithm on the lengths of the extrusions ($\boldsymbol{r}$ consisting of $r_i$) and the disturbance parameters ($\boldsymbol{\eta}$ consisting of $\eta_i$).

# References


[1] R. P. Brent. *Algorithms for minimization without derivatives.* Prentice-Hall, Englewood Cliffs, N.J, 1972.

[2] Sérgio Galdino. A family of regula falsi root-finding methods. 2011.

[3] A. N. Iusem. On the convergence properties of the projected gradient method for convex optimization. *Computational Applied Mathematics*, 22:37 – 52, 00 2003.

[4] William Press. *Numerical recipes : the art of scientific computing*. Cambridge University Press, Cambridge, UK New York, 2007.

[5] C. J. F. Ridders. A new algorithm for computing a single root of a real continuous function. 1979.

[6] Mohd Rivaie, Muhammad Fakhri, Nujma Hayati, Nurul Ramli, and Ibrahim Jusoh. The n-th section method: A modification of bisection. *Malaysian Journal of Fundamental and Applied Sciences*, 13, 12 2017.

[7] H.-J. Stoss. The complexity of evaluating interpolation polynomials. *Theoretical Computer Science*, 41:319–323, 1985.