OPTIMIZATION OF VECTOR FUNCTIONS USING THE MAX NORM

E. SHASKA AND T. SHASKA

Abstract. Minimizing vector-valued functions is a fundamental problem in optimization, with applications spanning numerical analysis, machine learning, and number theory. The choice of norm plays a crucial role in both the theoretical framework and computational methods used for optimization. While the Euclidean norm (ℓ_2) is commonly employed due to its smoothness and compatibility with gradient-based techniques, the max norm (ℓ_∞) presents unique challenges due to its inherent non-differentiability.

This paper investigates the minimization of vector functions under the max norm, a problem with significant implications in mathematical optimization and algebraic geometry. In number theory, the max norm corresponds to the *height* of a polynomial—the maximum absolute value of its coefficients—an essential concept in diophantine geometry and computational algebra. We explore various mathematical and algorithmic approaches to overcoming the difficulties posed by the max norm, including subgradient methods, smoothing approximations, coordinate descent, and proximal operator techniques.

A special emphasis is placed on applications in symmetric polynomials and graded vector spaces, where minimizing the max norm leads to interesting geometric and algebraic structures. Additionally, we discuss connections to classical problems in minimizing polynomial heights, including recent developments in machine learning approaches to Julia reduction.

By providing a systematic framework for max-norm minimization, this work bridges classical mathematical techniques with modern optimization and machine learning strategies. The results presented contribute to both theoretical advancements and practical computational methods in algebra, number theory, and optimization.

Contents

Date: February 14, 2025.

1. Introduction

Minimizing vector-valued functions is a central problem in mathematical optimization, with applications ranging from numerical analysis to modern machine learning. Given a vector function $F : \mathbb{R}^n \to \mathbb{R}^m$, the choice of norm significantly influences both theoretical analysis and computational methods. While the Euclidean norm (ℓ_2) is widely used due to its smoothness and compatibility with gradient-based optimization techniques, the max norm (ℓ_∞) presents distinct challenges due to its inherent non-differentiability.

This paper focuses on the problem of minimizing vector functions using the max norm, an area that has received relatively less attention compared to Euclidean-based methods. The max norm, defined as the largest absolute value among the components of a vector, is particularly relevant in areas such as number theory, where it corresponds to the *height* of a polynomial—the maximum absolute value of its coefficients. Heights play a crucial role in diophantine geometry, arithmetic dynamics, and algebraic number theory, making the study of max-norm-based optimization not only of theoretical interest but also of practical significance in mathematical applications. Minimizing heights of binary forms is a classical topic in mathematics due to the work of Hermite and many others, reaching its peak with Julia reduction. For a machine learning approach to Julia reduction, see [?2024-06]. Another source where max norm is used to minimize heights and weighted max norm to minimize weighted heights is [?2024-03].

We explore various approaches to handling the difficulties posed by the max norm in optimization. These include subgradient methods, smoothing approximations, coordinate descent, and proximal operator techniques, each offering different trade-offs between computational efficiency and theoretical guarantees. In addition, we examine specific applications in symmetric polynomials, where minimizing norms of coefficient vectors connects naturally to classical problems in algebraic geometry. Finally, we extend the discussion to graded vector spaces, introducing a structured framework for defining and computing max norms in this broader setting.

By developing a systematic approach to max-norm minimization, this work aims to bridge classical mathematical techniques with modern optimization and machine learning strategies. The insights gained here contribute not only to computational mathematics but also to fundamental questions in algebra and number theory.

2. Preliminaries

We start with some basic definitions from Calculus.

A **vector function** $F : \mathbb{R}^n \to \mathbb{R}^m$ is a mapping from an n-dimensional vector space to an m-dimensional vector space. It assigns to each vector $x = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$ a vector

$$F(x) = [f_1(x), f_2(x), \ldots, f_m(x)] \in \mathbb{R}^m.$$

Each component $f_i(x)$, where $i = 1, 2, \ldots, m$, is a scalar-valued function, meaning $f_i : \mathbb{R}^n \to \mathbb{R}$. We can think of $F(x)$ as a collection of m scalar functions, each dependent on the n input variables $x_1, x_2, \ldots, x_n$. Vector functions are essential tools in various fields, including physics (e.g., vector fields), computer graphics (e.g., transformations), and machine learning (e.g., representing model outputs). In essence, they allow us to represent and manipulate multiple related quantities simultaneously.

The gradient is a fundamental concept in calculus and optimization, particularly when dealing with scalar-valued functions of multiple variables. Let $g(x) : \mathbb{R}^n \to \mathbb{R}$ be a scalar function, where $x = [x_1, x_2, \ldots, x_n]$. The gradient of g at a point x, denoted by $\nabla g(x)$, is a vector whose components are the partial derivatives of g with respect to each variable x_i:

$$\nabla g(x) = \left(\frac{\partial g}{\partial x_1}(x), \frac{\partial g}{\partial x_2}(x), \ldots, \frac{\partial g}{\partial x_n}(x) \right)$$

The partial derivative $\frac{\partial g}{\partial x_i}(x)$ represents the instantaneous rate of change of the function g with respect to the variable x_i, while holding all other variables constant. Geometrically, the gradient vector $\nabla g(x)$ points in the direction of the greatest rate of *increase* of the function g at the point x. The magnitude of the gradient, $||\nabla g(x)||$, represents the rate of change of the function in that direction. In optimization, we are often interested in finding the direction of the greatest *decrease*, which is given by the negative gradient, $-\nabla g(x)$. The gradient plays a central role in optimization algorithms like gradient descent, which iteratively updates the input x by taking steps in the direction opposite to the gradient to find a local minimum of the function.

A **norm** is a function that assigns a non-negative real number, representing a "length" or "magnitude," to a vector. It generalizes the concept of absolute value for scalars to vectors. A norm $||\cdot||$ on a vector space must satisfy the following properties:

(1) **Non-negativity:** $||x|| \geq 0$ for all vectors x, and $||x|| = 0$ if and only if x is the zero vector.
(2) **Homogeneity:** $||\alpha x|| = |\alpha| \cdot ||x||$ for all scalars α and vectors x.
(3) **Triangle inequality:** $||x + y|| \leq ||x|| + ||y||$ for all vectors x and y.

In this note, we are particularly interested in two specific norms:

Euclidean Norm (L2 Norm): Also known as the L2 norm, the Euclidean norm is the most common and intuitive norm. For a vector $x = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$, the Euclidean norm is defined as:

$$||x||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Geometrically, $||x||_2$ represents the standard Euclidean distance from the origin to the point represented by the vector x.

Max Norm (Infinity Norm): Also known as the infinity norm or the L∞ norm, the max norm is defined as:

$$||x||_\infty = \max_{1 \leq i \leq n} \{|x_i|\}$$

In other words, the max norm is the largest absolute value among the components of the vector x. It's sometimes referred to as the "supremum norm" when dealing with functions.

Proof that the Max Norm is a Norm: Since the max norm will be the main focus of this paper we provide a proof that this is indeed a norm. We need to show that the max norm satisfies the three properties of a norm. Let $x = [x_1, \ldots, x_n]$ and $y = [y_1, \ldots, y_n]$ be vectors in $\mathbb{R}^n$, and let α be a scalar.

(1) **Non-negativity:** Since $|x_i| \geq 0$ for all i, their maximum is also non-negative: $||x||_\infty = \max_{1\leq i\leq n} |x_i| \geq 0$. If $||x||_\infty = 0$, then $|x_i| = 0$ for all i, which implies $x_i = 0$ for all i, so x is the zero vector. Conversely, if x is the zero vector, then all $x_i = 0$, so $||x||_\infty = 0$.
(2) **Homogeneity:** $||\alpha x||_\infty = \max_{1\leq i\leq n}\{|\alpha x_i|\} = \max_{1\leq i\leq n}\{|\alpha||x_i|\} = |\alpha| \max_{1\leq i\leq n}\{|x_i|\} = |\alpha|||x||_\infty$.
(3) **Triangle Inequality:**

$$\begin{aligned} ||x+y||_\infty &= \max_{1\leq i\leq n}\{|x_i + y_i|\} \\ &\leq \max_{1\leq i\leq n}\{|x_i| + |y_i|\} \quad \text{(Triangle inequality for scalars)} \\ &\leq \max_{1\leq i\leq n}\{|x_i|\} + \max_{1\leq i\leq n}\{|y_i|\} \quad \text{(Property of maximum)} \\ &= ||x||_\infty + ||y||_\infty \end{aligned}$$

Since all three properties are satisfied, the max norm is indeed a norm. □

The choice of norm depends on the specific problem being considered. The Euclidean norm is often used when we are interested in the overall magnitude or "energy" of a vector. The max norm, on the other hand, is useful when we are concerned with the largest individual component of the vector. Each norm induces a different notion of "closeness" or "distance" between vectors, which can affect the behavior of optimization algorithms.

3. Minimizing Vector Functions via Gradient Methods (Euclidean Norm)

Our goal is to minimize $||F(x)||_2$, where

$$F(x) = [f_1(x), f_2(x), \ldots, f_m(x)]$$

is a vector function. We define the scalar function

$$g(x) = ||F(x)||_2 = \sqrt{\sum_{i=1}^{m} f_i(x)^2}.$$

Minimizing $||F(x)||_2$ is equivalent to minimizing $g(x)$. This is a basic multivariable calculus problem. The idea is to leverage the gradient of $g(x)$ to find points where the function might attain a minimum. These points are called *critical points*.

Calculate the Gradient: The gradient of $g(x)$, denoted by $\nabla g(x)$, is a vector of the partial derivatives of $g(x)$ with respect to each component of x:

$$\nabla g(x) = \left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \ldots, \frac{\partial g}{\partial x_n}\right)$$

Using the chain rule, we can express the partial derivatives of $g(x)$ in terms of the partial derivatives of the component functions $f_i(x)$:

$$\frac{\partial g}{\partial x_j} = \frac{1}{2\sqrt{\sum_{i=1}^m f_i(x)^2}} \cdot 2\sum_{i=1}^{m} f_i(x)\frac{\partial f_i}{\partial x_j} = \frac{\sum_{i=1}^m f_i(x)\frac{\partial f_i}{\partial x_j}}{||F(x)||_2}$$

If $F(x) \neq 0$, which is usually the case, the gradient is well-defined.

Find Critical Points: Critical points are points x where the gradient is either zero or undefined. In our case, since we are assuming $F(x) \neq 0$, we are looking for points where the gradient is zero. We solve the system of equations:

$$\nabla g(x) = 0$$

This is equivalent to solving the system of n equations:

$$\frac{\partial g}{\partial x_j} = 0, \quad j = 1, 2, \ldots, n$$

Or, more compactly:

$$\sum_{i=1}^{m} f_i(x) \frac{\partial f_i}{\partial x_j} = 0, \quad j = 1, 2, \ldots, n$$

The solutions to this system of equations are the critical points of $g(x)$.

Second Derivative Test (Optional): To determine whether a critical point is a local minimum, local maximum, or a saddle point, we can use the second derivative test. This involves computing the Hessian matrix of $g(x)$ (the matrix of second partial derivatives) and analyzing its eigenvalues at the critical point. If all eigenvalues are positive, the critical point is a local minimum.

Boundary Analysis: If the domain of $F(x)$ has a boundary, we need to consider the behavior of $g(x)$ on the boundary as well. This often involves parameterizing the boundary and then finding the critical points of the resulting function.

Global Minimum: To find the global minimum of $g(x)$ on the entire domain, we compare the values of $g(x)$ at all the critical points (both interior and boundary) and any corner points of the domain. The point where $g(x)$ attains the smallest value is the global minimum.

3.1. **Machine Learning Methods.** Here we briefly describe some of the machine learning methods used to find the minimum

3.1.1. *Gradient Descent (GD):.* Gradient descent is the foundational algorithm for minimizing differentiable functions. It's an iterative process that updates the input vector x by taking steps in the direction *opposite* to the gradient of the function at the current point. The rationale is that the negative gradient points in the direction of the steepest *decrease* of the function.

The update rule is:

$$x_{k+1} = x_k - \eta \nabla g(x_k)$$

where:

- x_{k+1} is the updated input vector at iteration $k + 1$.
- x_k is the current input vector at iteration k.
- η is the learning rate (a small positive scalar). The learning rate controls the step size taken in the direction of the negative gradient. It's a crucial hyperparameter. If it's too small, convergence can be very slow. If it's too large, the algorithm might overshoot the minimum and oscillate or even diverge.
- $\nabla g(x_k)$ is the gradient of the function $g(x)$ evaluated at x_k.

Variants of Gradient Descent:

- *Batch Gradient Descent:* The gradient is computed using the *entire* dataset. This can be computationally expensive for large datasets.
- *Stochastic Gradient Descent (SGD):* The gradient is computed using only *one* randomly chosen data point. This is much faster than batch gradient descent but introduces more noise into the updates.
- *Mini-Batch Gradient Descent:* A compromise between batch and stochastic gradient descent. The gradient is computed using a small, randomly sampled subset of the data called a "mini-batch." This offers a good balance between computational efficiency and stability of the updates.

3.1.2. *Backpropagation:* Backpropagation is not an optimization algorithm itself; rather, it's an extremely efficient algorithm for computing gradients in neural networks. Neural networks are complex functions composed of multiple layers of interconnected nodes. The loss function (which we want to minimize) depends on the network's weights and biases. Backpropagation uses the chain rule of calculus to compute the gradient of the loss function with respect to *all* the weights and biases in the network. It starts from the output layer and propagates the gradient backward through the network, layer by layer. This allows us to efficiently calculate the gradients needed for gradient descent (or its variants) to update the network's parameters.

3.1.3. *Stochastic Gradient Descent (SGD):.* As mentioned above, SGD is a variant of gradient descent where the gradient is computed using a single data point (or a mini-batch). The update rule is the same as gradient descent, but the gradient is now a "noisy" estimate of the true gradient. This noise can actually be beneficial, as it can help the algorithm escape local minima and find better solutions.

3.1.4. *Adaptive Optimizers:* Adaptive optimization algorithms improve upon standard gradient descent by adjusting the learning rate for each parameter individually. This is particularly useful because different parameters might have different sensitivities to changes in the learning rate. Some popular adaptive optimizers include:

- **Adam (Adaptive Moment Estimation):** Adam keeps track of both the first and second moments of the gradients. The first moment is an exponentially decaying average of the gradients, and the second moment is an exponentially decaying average of the squared gradients. These moments are used to adapt the learning rate for each parameter. Adam is often a good default choice for optimization problems.
- **RMSprop (Root Mean Square Propagation):** RMSprop is similar to Adam, but it only uses the exponentially decaying average of the squared gradients to adapt the learning rate.
- **Adagrad (Adaptive Gradient Algorithm):** Adagrad adapts the learning rate based on the cumulative sum of squared gradients. Parameters that have received large gradients in the past will have smaller learning rates, and vice versa. However, Adagrad's learning rates can become very small over time, which can hinder convergence.
- **Adadelta (Adaptive Delta):** Adadelta addresses the diminishing learning rate problem of Adagrad by using a decaying average of past squared gradients.

Adaptive optimizers often converge faster and are less sensitive to the choice of initial learning rate than standard gradient descent. However, they also have their own hyperparameters that need to be tuned.

4. Minimizing Vector Functions using Max Norm (Mathematical Approaches)

The max norm,

$$||F(x)||_\infty = \max_{1\leq i\leq m}\{|f_i(x)|\},$$

presents a significant challenge for optimization: it is not differentiable everywhere. Specifically, it's non-differentiable at points where two or more of the magnitudes $|f_i(x)|$ are equal and achieve the maximum. This non-differentiability prevents the direct application of standard gradient-based optimization methods, which rely on the existence of a gradient.

The core idea behind the mathematical approach for minimizing with the max norm is to break the problem down into smaller, more manageable pieces by analyzing each component function individually.

4.1. **Component-wise Analysis:** This is the heart of the method. We analyze each component function $f_i(x)$ separately to find its potential extrema (maximum and minimum values).

4.1.1. *Interior Critical Points:* For each $i = 1, \ldots, m$, we want to find points x in the interior of the domain where $|f_i(x)|$ could potentially have a maximum or minimum. This involves two steps:

(1) *Critical Points of $f_i(x)$:* Find points x where the partial derivatives of $f_i(x)$ are all zero:

$$\frac{\partial f_i}{\partial x_j}(x) = 0, \quad j = 1, 2, \ldots, n$$

These are the standard critical points of the function $f_i(x)$ itself. At these points, $f_i(x)$ could have a local maximum, local minimum, or a saddle point.

(2) *Points where $f_i(x) = 0$:* Because we are dealing with $|f_i(x)|$, we also need to consider points where $f_i(x) = 0$. At these points, $|f_i(x)|$ will have a minimum.

4.1.2. *Boundary Analysis:* The maximum or minimum of $|f_i(x)|$ could also occur on the boundary of the domain. To find these extrema, we typically:

(1) *Parameterize the Boundary:* Express the boundary of the domain using parametric equations. For example, if the domain is a circle, we could parameterize it using polar coordinates.

(2) *Restrict $f_i(x)$ to the Boundary:* Substitute the parametric equations into $f_i(x)$ to get a function of the parameters.

(3) *Find Extrema on the Boundary:* Use single-variable or multi-variable calculus techniques (depending on the complexity of the boundary) to find the maximum and minimum values of the restricted function.

4.2. **Combining Results:** After analyzing each component function $f_i(x)$, we have a set of candidate points x where $|f_i(x)|$ could have a maximum or minimum. This set includes:

- Critical points of $f_i(x)$ in the interior.
- Points where $f_i(x) = 0$.
- Points on the boundary where $|f_i(x)|$ attains a maximum or minimum.

For each of these candidate points x, we evaluate *all* the component functions

$$|f_1(x)|, |f_2(x)|, \ldots, |f_m(x)|$$

and then calculate the max norm:

$$g(x) = ||F(x)||_\infty = \max_{1\leq i\leq m}\{|f_i(x)|\}$$

4.3. **Comparison:** Finally, we compare the values of $g(x)$ that we calculated for all the candidate points. The smallest value of $g(x)$ is the minimum of the max norm, and the corresponding point x is the minimizer. The largest value is the maximum.

5. Machine Learning Methods for Minimizing Vector Functions using Max Norm

The non-differentiability of the max norm

$$||F(x)||_\infty = \max_{1\leq i\leq m} |f_i(x)|$$

presents a significant hurdle for traditional gradient-based optimization. While the Euclidean norm's smoothness allows for direct application of gradient descent and backpropagation, the max norm requires specialized techniques. Here's a detailed exploration of these methods:

5.1. **Subgradient Methods.** Subgradient methods offer a way to generalize the concept of a gradient to non-differentiable functions. At a point x where $g(x)$ is differentiable, the subgradient is simply the gradient $\nabla g(x)$. However, at points of non-differentiability, the subgradient becomes a *set* of vectors. Any vector within this set can serve as a "generalized gradient."

A vector v is a subgradient of $g(x)$ at x_0 if it satisfies the following inequality for all x:

$$g(x) \geq g(x_0) + v^T(x - x_0)$$

Subgradient descent updates the input vector using a subgradient at each iteration:

$$x_{k+1} = x_k - \eta_k v_k$$

where v_k is a subgradient of $g(x)$ at x_k, and η_k is the learning rate at iteration k. Choosing an appropriate sequence of learning rates (η_k) is crucial for convergence. Often, diminishing learning rates are used (e.g., $\eta_k \propto 1/k$ or $\eta_k \propto 1/\sqrt{k}$).

As advantage is that subgradient methods can handle non-differentiable functions directly. Disadvantages are that convergence can be significantly slower and less stable than with gradient descent for smooth functions. The choice of learning rate schedule is critical and can greatly impact performance. Subgradients can be computationally expensive to compute for some functions.

Comparison to Euclidean Norm Methods is that with the Euclidean norm, we have a unique gradient, leading to more directed and often faster convergence. The set of subgradients for the max norm introduces ambiguity, making the optimization process less precise.

Bundle methods, which use information from multiple subgradients, can sometimes improve convergence compared to simple subgradient descent.

5.2. **Smoothing Approximations.** This approach bypasses the non-differentiability issue by approximating the max function with a smooth, differentiable function. A common choice is the "soft-max" or "log-sum-exp" function:

$$g_{\text{approx}}(x) = \frac{1}{k} \log \left(\sum_{i=1}^{m} \exp(k|f_i(x)|) \right)$$

where k is a large positive constant. As $k \to \infty$, $g_{\text{approx}}(x)$ approaches the true max function.

To implement this approach we replace the max norm in your objective function with $g_{\text{approx}}(x)$. Then, you can use standard gradient-based optimization techniques (like backpropagation for neural networks).

Some advantages are that this method allows the use of efficient gradient-based methods and readily available optimization libraries.

Disadvantages are that it introduces an approximation error. The choice of k is crucial. A larger k leads to a better approximation but can also cause numerical instability during optimization. Too small a k leads to a smoother function but a larger approximation error.

This method is similar to Euclidean norm methods in that we can use gradients; however, we are now minimizing an approximation, not the true objective.

Gradually increasing the smoothing parameter k during training (starting with a smaller k and increasing it over time) can sometimes improve performance. This allows the optimization to explore a smoother landscape initially and then refine the solution as the approximation gets closer to the true max function.

5.3. **Coordinate Descent.** Coordinate descent updates the variables one at a time (or in small blocks). For each variable (or block), the objective function is minimized with respect to that variable, while holding the others fixed. This can be particularly useful when dealing with non-smooth functions, as it avoids the need to compute a full gradient.

The implementation depends heavily on the specific form of the objective function. For each variable, you need to find the minimum of the function with respect to that variable. This might involve solving a one-dimensional optimization problem.

Advantages are that it can be more robust to non-smoothness than gradient-based methods. Can be simpler to implement in some cases. Disadvantages are that it can be much slower than gradient-based methods, especially for high-dimensional problems. The convergence behavior can be sensitive to the ordering of the coordinate updates.

Coordinate descent is quite different from gradient-based methods used with the Euclidean norm. It doesn't rely on gradients and explores the search space by changing one variable at a time.

Stochastic coordinate descent, where the coordinates are chosen randomly, can sometimes improve convergence.

5.4. **Proximal Operator Methods.** Proximal operator methods are a class of optimization algorithms specifically designed for problems with non-smooth terms in the objective function. They use the concept of the *proximal operator*, which is a generalization of the projection operator. The proximal operator of a function at a point is a point that is "close" to the original point and also minimizes a combination of the function value and the distance to the original point.

Proximal operator methods can be more complex to implement than other methods. They require computing the proximal operator of the non-smooth part of the objective function, which can be challenging depending on the function.

Such methods can handle non-smoothness very effectively. Often have good convergence properties. However, they are more complex to implement. Computing the proximal operator can be computationally expensive.

Proximal operator methods are designed specifically for non-smooth optimization, whereas standard methods for the Euclidean norm rely on smoothness. There are various proximal operator methods, such as proximal gradient descent and ADMM (Alternating Direction Method of Multipliers), each with its own strengths and weaknesses. The choice of method depends on the specific problem.

Which Method to Choose? The "best" method depends on the specifics of the problem, the size of the data, and the computational resources available. Here's a general guideline:

- **Small to medium-sized datasets, relatively simple functions:** Smoothing approximations are often a good starting point due to their ease of implementation and compatibility with existing optimization libraries.

- **Large datasets, complex functions:** Subgradient methods or stochastic variants of other methods might be necessary.
- **Problems where smoothness is a significant issue:** Proximal operator methods or coordinate descent can be more robust.
- **Neural Networks:** Smoothing approximations are generally the most practical choice, as they allow the use of backpropagation.

6. Specific Case: Symmetric Polynomials

In this section, we apply the theory to symbolic polynomials. We focus on minimizing norms of coefficient vectors, connecting this to minimizing norms of symmetric polynomials evaluated at the roots.

Let

$$P(z) = c_n z^n + c_{n-1} z^{n-1} + \cdots + c_1 z + c_0$$

be a symbolic polynomial of degree n with complex coefficients $c_i \in \mathbb{C}$. Let $a = [a_1, a_2, \ldots, a_n]$ be a vector of n complex numbers, representing the roots of a polynomial. The k-th elementary symmetric polynomial, denoted by $s_k(a)$, is defined as the sum of all possible products of k distinct roots:

$$\begin{aligned} s_1(a) &= a_1 + a_2 + \cdots + a_n \\ s_2(a) &= a_1a_2 + a_1a_3 + \cdots + a_{n-1}a_n \\ &\vdots \\ s_n(a) &= a_1a_2 \ldots a_n \end{aligned}$$

We define the vector function $F(a) = [s_1(a), s_2(a), \ldots, s_n(a)]$. Our goal is to find the vector of complex roots a that minimizes the norm of $F(a)$, considering both the Euclidean and max norms

- **Euclidean Norm:** $||F(a)||_2 = \sqrt{|s_0|^2 + |s_1|^2 + \cdots + |s_n|^2}$
- **Max Norm:** $||F(a)||_\infty = \max_{0 \le i \le n} |s_i|$

6.1. Minimization with Euclidean Norm.

To minimize

$$||F(a)||_2 = \sqrt{\sum_{k=1}^{n} |s_k(a)|^2},$$

we define $g(a) = ||F(a)||_2$. Because the symmetric polynomials are differentiable functions of the roots (in the complex sense), $g(a)$ is also differentiable (except possibly at $F(a) = 0$). Therefore, we can use gradient-based optimization methods:

(1) **Complex Gradient:** Calculate the gradient of $g(a)$ with respect to the complex variables a_j. This involves complex differentiation. The partial derivatives will be complex-valued.
(2) **Critical Points:** Solve $\nabla g(a) = 0$ to find critical points. This will involve solving a system of n complex equations.
(3) **Optimization:** Use a suitable optimization algorithm (e.g., gradient descent with complex numbers, Adam with complex numbers). Most modern deep learning frameworks support complex numbers and automatic differentiation, making this step relatively straightforward.

6.2. **Minimization with Max Norm.** To minimize

$$||F(a)||_\infty = \max_{1\le k\le n} |s_k(a)|,$$

we encounter the non-differentiability of the max norm.

(1) **Component-wise Analysis:** For each $k = 1, \ldots, n$, analyze $|s_k(a)|$ individually. This involves finding critical points of $|s_k(a)|$ in the interior and on the boundary of the domain (which might be constrained, e.g., roots in the upper half-plane).
(2) **Combining Results:** For each candidate point a found in the component-wise analysis, compute $||F(a)||_\infty = \max_{1\le k\le n} |s_k(a)|$.
(3) **Comparison:** Compare these values to find the minimum.

6.3. **Machine Learning Approaches for Max Norm.** The machine learning approaches discussed earlier (subgradient methods, smoothing approximations, coordinate descent, proximal operator methods) are all applicable here. However, smoothing approximations are often the most practical due to their ease of implementation and compatibility with complex differentiation in deep learning frameworks.

Because the roots are complex, all calculations, including gradients and optimization updates, must be performed using complex numbers. Deep learning frameworks handle this seamlessly.

Fundamental Domain: The conjecture that shifting the n-gon formed by the roots towards a fundamental domain tends to minimize the max norm is a fascinating area for further investigation. Defining the fundamental domain precisely and rigorously proving (or disproving) this conjecture is a significant challenge. This might involve connections to modular forms or other areas of complex analysis.

Smoothing Approximation (Recommended): Using the smooth approximation

$$g_{\text{approx}}(a) = \frac{1}{k} \log\left(\sum_{i=1}^{n} \exp(k|s_i(a)|)\right)$$

is often the most practical approach for machine learning. It allows the use of backpropagation and complex differentiation.

Geometric Interpretation: The geometric interpretation of minimizing the norm of symmetric polynomials is complex and depends on n. It's related to making the n-gon formed by the roots "as regular as possible" in some sense, but the precise meaning of "regular" depends on the chosen norm. Numerical experiments and visualizations can be very helpful in developing intuition.

Example 1 (Binary Quadratics)**.** *Consider $P(z) = c_2 z^2 + c_1 z + c_0$. To minimize $||C||_\infty$, we can use the smoothing approximation:*

$$||C||_\infty \approx \frac{1}{k} \log(\exp(k|c_0|) + \exp(k|c_1|) + \exp(k|c_2|))$$

Express c_0 and c_1 in terms of the roots a_1 and a_2 using Vieta's formulas and then use gradient descent with respect to a_1 and a_2.

7. Max Norm on Graded Vector Spaces

Yes, there is a natural way to define a max norm (or infinity norm) on graded vector spaces. However, it's important to understand what "graded" means in this context and how it affects the definition.

7.1. **Graded Vector Spaces.** A graded vector space V is a vector space that is decomposed into a direct sum of subspaces:

$$V = \bigoplus_{i \in \mathbb{Z}} V_i$$

where the index set (in this case, the integers $\mathbb{Z}$) represents the "degree" or "grading" of the subspaces. Elements of V_i are said to be *homogeneous* of degree i. Not all vector spaces are graded. A graded vector space provides additional structure.

7.2. **Defining the Max Norm.** The crucial point is that the max norm on a graded vector space needs to respect the grading. You can't just take the max of the norms of arbitrary components because elements of different degrees belong to different subspaces and shouldn't be directly compared in this way.

Here's how you can define the max norm on a graded vector space V:

(1) **Choose a Norm on Each Subspace:** For each degree $i \in \mathbb{Z}$, choose a norm $||\cdot||_i$ on the subspace V_i. These norms can be different for different degrees.
(2) **Decompose the Vector:** Let $v \in V$. Since V is a direct sum, v can be uniquely decomposed as a sum of homogeneous components:

$$v = \sum_{i \in \mathbb{Z}} v_i$$

where $v_i \in V_i$. Note that only finitely many of these v_i will be non-zero because the sum is direct.
(3) **Define the Max Norm:** The max norm of v is then defined as the maximum of the norms of its homogeneous components:

$$||v||_\infty = \max_{i \in \mathbb{Z}} ||v_i||_i$$

7.3. **Why this Definition Makes Sense.**

- **Respects the Grading:** We are comparing norms of elements within the *same* degree subspace.
- **Reduces to Standard Max Norm:** If all elements of a vector space are considered to have the same degree (so that there is only one subspace), this definition reduces to the usual max norm.
- **Satisfies Norm Properties:** One can show that this definition satisfies the properties of a norm (non-negativity, homogeneity, and the triangle inequality). The triangle inequality, in particular, will follow from the triangle inequalities of the norms $||\cdot||_i$ on the individual subspaces.

7.4. **Example.** Consider the graded vector space of polynomials in one variable, where the degree of a monomial is the degree of the polynomial. Let $p(x) = 2x^3 + x^2 - 5x + 1$. The homogeneous components are $2x^3$ (degree 3), x^2 (degree 2), $-5x$ (degree 1), and 1 (degree 0). If we define the norm on each subspace V_i (polynomials of degree i) as the absolute value of the coefficient, then the max norm of $p(x)$ would be:

$$||p(x)||_\infty = \max(|2|, |1|, |-5|, |1|) = 5$$

Key Point: The specific norms chosen on the subspaces V_i influence the resulting max norm on the graded vector space. There isn't a single "the" max norm on a graded vector space until you specify the norms on the individual graded components.