Optimization of Energy Numbers Continued

Parker Emmerson

October 2024

1 Introduction

Question 3

Polyhedral cone representation.

A convex cone $\mathcal{K} \subset \mathbb{R}^d$ is called *polyhedral* if it can be written as $\mathcal{K} = \mathcal{A}\mathbb{R}^k_+$ where $\mathcal{A} \in \mathbb{R}^{d\times k}$ for some finite k.

(a) Let $\mathcal{S}^n$ be the cone of $n \times n$ positive semidefinite matrices. Show that $\mathcal{S}^n$ is a polyhedral cone by constructing an appropriate matrix A that defines a polyhedral cone for $\mathcal{S}^n$, i.e., $\mathcal{S}^n = \{\rho A \ : \ \rho \geq 0, \ A \succeq 0\}$.

Solution for Part (a)

First, let's recall the definition of a polyhedral cone. A cone $\mathcal{K}$ is polyhedral if it can be expressed as the set of linear combinations with non-negative scalars of finite vectors. That is:

$$\mathcal{K} = \left\{ \mathcal{A}\lambda \ : \ \lambda \in \mathbb{R}^k_+ \right\},$$

where $\mathcal{A}$ is a $d \times k$ matrix and k is finite. In other words, $\mathcal{K}$ is finitely generated by the columns of $\mathcal{A}$.

Now, consider the cone $\mathcal{S}^n$ of $n \times n$ positive semidefinite (PSD) matrices. We need to show that $\mathcal{S}^n$ is a polyhedral cone.

However, it is important to note that in general, the cone $\mathcal{S}^n$ is not polyhedral when $n > 1$. This is because the cone of $n \times n$ PSD matrices is not a finitely generated cone. Instead, it is convex and closed but has infinitely many extreme rays.

Therefore, unless $n = 1$, $\mathcal{S}^n$ is not a polyhedral cone.

Corrected Problem Statement Given that $\mathcal{S}^n$ is not polyhedral for $n > 1$, perhaps the intended problem is to show that a subset of $\mathcal{S}^n$ is polyhedral or to consider cases where $n = 1$.

Alternatively, if we consider the cone of $n \times n$ diagonal PSD matrices, this cone is polyhedral because it corresponds to non-negative diagonal matrices, which can be represented as a finite combination of the standard basis matrices.

Solution Assuming Diagonal PSD Matrices

Let's consider the set of diagonal $n \times n$ PSD matrices, denoted by $\mathcal{D}^n$. A diagonal matrix D is PSD if and only if all its diagonal entries are non-negative. Thus:

$$\mathcal{D}^n = \left\{ D \in \mathbb{R}^{n \times n} \ : \ D = \operatorname{diag}(d_1, d_2, \ldots, d_n), \ d_i \geq 0 \right\}.$$

We can represent $\mathcal{D}^n$ as a polyhedral cone generated by the n basis matrices $E^{(i)}$, where $E^{(i)}$ has a 1 in the (i,i)-th position and zeros elsewhere:

$$\mathcal{D}^n = \left\{ \sum_{i=1}^{n} d_i E^{(i)} \ : \ d_i \geq 0 \right\}.$$

Thus, $\mathcal{D}^n$ is a polyhedral cone generated by the finite set of matrices $\{E^{(1)}, E^{(2)}, \ldots, E^{(n)}\}$.

Conclusion for Part (a)

Given that $\mathcal{S}^n$ is not polyhedral for $n > 1$, the initial statement of the problem seems incorrect. If the problem intended to ask about the cone of diagonal PSD matrices or a finite-dimensional subset, then it would be correct to show it is polyhedral.

Therefore, the cone of all $n \times n$ PSD matrices is not polyhedral when $n > 1$.

Alternative Interpretation

Perhaps the problem wants us to consider the set of $n \times n$ PSD matrices as a convex cone that can be represented via linear matrix inequalities (LMIs), which are a set of linear (affine) inequalities in the entries of the matrix.

Let's consider the characterization of $\mathcal{S}^n$ using linear inequalities.

Expressing PSD Matrices via Linear Inequalities

An $n \times n$ symmetric matrix X is PSD if and only if all its principal minors are non-negative. However, this involves checking an exponential number of conditions.

Alternatively, we can consider the definition of the PSD cone in terms of the Gram representation.

A symmetric matrix X is PSD if and only if there exists a matrix $V \in \mathbb{R}^{n \times k}$ such that $X = VV^\top$, for some $k \leq n$.

However, expressing $X = VV^\top$ involves bilinear terms, and cannot be directly used to represent $\mathcal{S}^n$ as a polyhedral cone.

Given these considerations, it is clear that the PSD cone $\mathcal{S}^n$ is not polyhedral when $n > 1$.

Final Answer for Part (a)

Therefore, the cone $\mathcal{S}^n$ of $n \times n$ positive semidefinite matrices is **not polyhedral** when $n > 1$. It cannot be represented as a finite combination of generators with non-negative coefficients.

Note: If $n = 1$, then $\mathcal{S}^1$ is the set of non-negative real numbers $\mathbb{R}_+$, which is a polyhedral cone in $\mathbb{R}^1$.

Recommendation

It is possible that the problem statement contains an error or is intended to be about a different concept. If the question aims to discuss the properties of the PSD cone and its representation, it might be better to rephrase or reconsider the question.

Alternate Problem (Corrected)

Suppose instead the question is:

Show that the cone of $n \times n$ PSD matrices with entries constrained to be diagonal matrices is a polyhedral cone.

In that case, the solution provided earlier for diagonal PSD matrices applies, and the cone is indeed polyhedral.

(b) Consider a weight vector $w \in \mathbb{R}^D$ and two feature mappings $\phi : \mathcal{X} \to \mathbb{R}^D$, $\phi' : \mathcal{X} \to \mathbb{R}^D$. Then the vector-valued mapping $x \mapsto \phi(x)\phi'(x)^\top$ defines a bipartite kernel on a product space $B \times B'$:

$$K(x, x') = w^\top \left(\phi(x)\phi'(x')^\top \right).$$

Computing kernels $K(x, x')$ directly may consume a lot of memory because the feature mappings may be high-dimensional. Instead, kernels are typically computed on-the-fly whenever their values are needed.

Design an algorithm that performs the computation on-the-fly by exploiting a polyhedral description of the cone

$$\mathcal{C} := \operatorname{conv}\{\phi(x)\phi'(x)^\top,\ x \in \mathcal{X}\},$$

that is, describe an algorithm that efficiently computes

$$c = \inf_{x \in \mathcal{X}} \{w^\top \phi(x)\phi'(x)^\top\}$$

by on-the-fly computation of $w^\top \phi(x)\phi'(x)^\top$ for arbitrary x.

Solution for Part (b)

First, let's understand what is being asked.

We are given:

- A weight vector $w \in \mathbb{R}^D$. - Two feature maps $\phi : \mathcal{X} \to \mathbb{R}^D$ and $\phi' : \mathcal{X} \to \mathbb{R}^D$. - The mapping $x \mapsto \phi(x)\phi'(x)^\top \in \mathbb{R}^{D \times D}$. - The kernel function $K(x, x') = w^\top \left(\phi(x)\phi'(x')^\top\right)$.

Our goal is to compute:

$$c = \inf_{x \in \mathcal{X}} \left\{ w^\top \left(\phi(x)\phi'(x)^\top\right) \right\}$$

efficiently, without explicitly computing and storing the entire matrix $\phi(x)\phi'(x)^\top$.

Note that $\phi(x)\phi'(x)^\top$ is an outer product of two vectors, resulting in a $D \times D$ matrix, which can be large if D is large.

However, since $w \in \mathbb{R}^D$, when we take the inner product $w^\top \left(\phi(x)\phi'(x)^\top\right)$, we get:

$$w^\top \left(\phi(x)\phi'(x)^\top\right) = \left(w^\top \phi(x)\right) \phi'(x)^\top$$

But this is still a vector, not a scalar. Actually, since $w^\top \phi(x)$ is a scalar, and $\phi'(x)$ is a vector, their product is a scalar multiplied by a vector, resulting in a vector.

But the notation $w^\top \left(\phi(x)\phi'(x)^\top\right)$ is a vector. Then, perhaps the inner product is not correctly specified.

Alternatively, perhaps the kernel is defined as:

$$K(x, x') = \left(w^\top \phi(x)\right) \left(\phi'(x')^\top\right)$$

But that seems inconsistent.

Alternatively, maybe the mapping $x \mapsto \phi(x)\phi'(x)^\top$ defines a matrix, and we are supposed to compute:

$$c = \inf_{x \in \mathcal{X}} \left\{ \langle w, \phi(x)\phi'(x)^\top \rangle_F \right\}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product.

In that case, we can interpret w as a vectorized matrix $w \in \mathbb{R}^{D \times D}$, flattened to $\mathbb{R}^{D^2}$, and we are taking the inner product between two matrices, flattened as vectors.

Alternatively, perhaps w is a matrix in $\mathbb{R}^{D \times D}$, and the kernel is defined as:

$$K(x, x') = \operatorname{tr}\left(w^\top \left(\phi(x)\phi'(x)^\top\right)\right)$$

Given the ambiguities, let's try to clarify.

Given that, the problem seems to be to compute:

$$c = \inf_{x \in \mathcal{X}} \left\{ w^\top \left(\phi(x)\phi'(x)^\top\right) \right\}$$

Wait, but since w is a vector in $\mathbb{R}^D$, and $\phi(x)\phi'(x)^\top$ is a matrix in $\mathbb{R}^{D\times D}$, the expression $w^\top(\phi(x)\phi'(x)^\top)$ is undefined as a multiplication between $\mathbb{R}^D$ and $\mathbb{R}^{D\times D}$.

Alternatively, perhaps the kernel is defined as:

$$K(x,x') = \left(\phi(x)^\top w \phi'(x')\right)$$

This way, we have an expression where w is a $D \times D$ matrix, and $\phi(x), \phi'(x')$ are vectors in $\mathbb{R}^D$. Then $\phi(x)^\top w \phi'(x')$ is a scalar.

Alternatively, perhaps w is actually a matrix in $\mathbb{R}^{D\times D}$, and the kernel is defined by:

$$K(x,x') = \operatorname{tr}\left(w^\top \left(\phi(x)\phi'(x)^\top\right)\right)$$

Then, $\operatorname{tr}\left(w^\top \left(\phi(x)\phi'(x)^\top\right)\right) = \operatorname{tr}\left(\left(w^\top \phi(x)\phi'(x)^\top\right)\right)$.

Since $\phi(x)\phi'(x)^\top$ is a rank-one matrix, and w is $D \times D$, the trace of their product is a scalar.

Alternatively, we can consider vectorizing the matrix $\phi(x)\phi'(x)^\top$ into a vector of dimension D^2, and similarly flattening w into a vector of dimension D^2. Then their dot product is:

$$\langle w, \phi(x)\phi'(x)^\top\rangle_{\text{vec}} = \langle \operatorname{vec}(w), \operatorname{vec}(\phi(x)\phi'(x)^\top)\rangle$$

But $\operatorname{vec}(\phi(x)\phi'(x)^\top) = \phi'(x) \otimes \phi(x)$, where $\otimes$ denotes the Kronecker product.

Alternatively, perhaps the problem is asking us to compute $w^\top(\phi(x) \circ \phi'(x))$, where $\circ$ denotes the Hadamard (element-wise) product.

Given that, perhaps the assistant should make an assumption to proceed.

Let me make an assumption that the kernel is defined as:

$$K(x,x') = w^\top(\phi(x) \circ \phi'(x'))$$

Then the mapping from x to $\phi(x) \circ \phi'(x)$ is element-wise multiplication of the feature maps, resulting in a vector of size D.

Then the expression $w^\top(\phi(x) \circ \phi'(x))$ is a scalar.

In that case, the cone $\mathcal{C} = \operatorname{conv}\{\phi(x) \circ \phi'(x) \mid x \in \mathcal{X}\}$ is a convex hull of these vectors.

Then, the problem reduces to:

Compute $c = \inf_{x\in\mathcal{X}}\{w^\top(\phi(x) \circ \phi'(x))\}$

Our task is to design an algorithm that computes $w^\top(\phi(x) \circ \phi'(x))$ on-the-fly for arbitrary x, without precomputing and storing all the feature combinations.

Moreover, we need to exploit the polyhedral description of the cone $\mathcal{C}$ to compute c efficiently.

Alternatively, perhaps the algorithm involves optimizing over $x \in \mathcal{X}$ to find the minimal value of $w^\top(\phi(x) \circ \phi'(x))$, possibly using properties of $\mathcal{C}$.

Perhaps if we can think of $\mathcal{C}$ as a polyhedral cone generated by a finite set of extreme rays, we can formulate the problem as a linear program.

Given that, perhaps the assistant can proceed to describe an algorithm that, given w, computes $c = \inf_{x\in\mathcal{X}}\{w^\top(\phi(x) \circ \phi'(x))\}$ by solving an optimization problem over $\mathcal{X}$.

Alternatively, perhaps the assistant can suggest that if both feature mappings ϕ and ϕ' are piecewise linear or can be represented as convex functions, then we can formulate the optimization problem as convex optimization.

However, the exact solution depends heavily on the properties of ϕ, ϕ', and $\mathcal{X}$.

Given the limited information, perhaps the assistant can outline a general approach.

Proposed Solution

Our goal is to compute:

$$c = \inf_{x\in\mathcal{X}}\{w^\top \left(\phi(x)\phi'(x)^\top\right)\}$$

Suppose we let $f(x) = w^\top \left(\phi(x)\phi'(x)^\top\right)$.

We can attempt to compute c by solving the optimization problem:

$$\begin{aligned} \text{Minimize}_{x\in\mathcal{X}} \quad & f(x) \\ \text{Subject to} \quad & x \in \mathcal{X} \end{aligned}$$

However, to perform this computation efficiently, we need to exploit the structure of $f(x)$.

First, note that:

- $w^\top \left(\phi(x)\phi'(x)^\top\right) = \sum_{i=1}^{D}\sum_{j=1}^{D} w_{ij}\phi_i(x)\phi'_j(x)$

But since $w \in \mathbb{R}^D$, this does not fit unless w_{ij} is a $D \times D$ matrix, or unless we can further specify the form of w.

Alternatively, perhaps the expression is:

$$f(x) = \left(w^\top\phi(x)\right)\left(v^\top\phi'(x)\right)$$

Assuming that, then we have:

- $f(x) = \left(w^\top\phi(x)\right)\left(v^\top\phi'(x)\right)$

But then we can observe that the minimum of $f(x)$ over x depends on the product of two functions.

Alternatively, perhaps the assistant can proceed to outline a method to compute c on-the-fly.

Assuming that we can compute $f(x)$ for any x, and that evaluating f is relatively cheap.

Alternatively, perhaps if we can formulate the dual problem.

Given that the cone $\mathcal{C}$ is the convex hull of $\phi(x)\phi'(x)^\top$ for all $x \in \mathcal{X}$.

Then perhaps we can write the optimization problem as:

$$c = \min_{y\in\mathcal{C}}\{w^\top y\}$$

Since $\mathcal{C} = \text{conv}\{\phi(x)\phi'(x)^\top \mid x \in \mathcal{X}\}$, this is a linear function minimized over a convex set $\mathcal{C}$.

But perhaps instead of explicitly computing $\mathcal{C}$, we can solve:

$$c = \inf_{x \in \mathcal{X}} \{w^\top (\phi(x)\phi'(x)^\top)\}$$

Given that w and $\phi(x)$, $\phi'(x)$ are given, perhaps our algorithm proceeds as follows:

Algorithm Outline:

1. **Initialize:** Start with an arbitrary $x_0 \in \mathcal{X}$.

2. **Compute $f(x_0)$:** Evaluate $f(x_0) = w^\top \left(\phi(x_0)\phi'(x_0)^\top\right)$.

3. **Iterative Optimization:** - Use an optimization algorithm to find x that minimizes $f(x)$. - This could be gradient descent if f is differentiable and $\mathcal{X}$ is continuous. - If $\mathcal{X}$ is discrete, we might need to use combinatorial optimization methods. - At each step, compute $f(x)$ on-the-fly without storing the full $\phi(x)\phi'(x)^\top$ matrix.

4. **Return c:** Once the optimization converges or after a predefined number of iterations, return the minimal value of $f(x)$ found.

On-the-fly Computation:

At each step, we compute $f(x)$ as:

$$f(x) = w^\top \left(\phi(x)\phi'(x)^\top\right) = \sum_{i=1}^{D} \sum_{j=1}^{D} w_{ij} \phi_i(x) \phi'_j(x)$$

But since w is a vector in $\mathbb{R}^D$, unless w_{ij} are arranged appropriately.

Alternatively, perhaps we can write $f(x)$ as:

If w is the vectorization of a matrix $W \in \mathbb{R}^{D \times D}$, then:

$$f(x) = \text{vec}(W)^\top \text{vec}\left(\phi(x)\phi'(x)^\top\right)$$

But $\text{vec}\left(\phi(x)\phi'(x)^\top\right) = \phi'(x) \otimes \phi(x)$, where $\otimes$ is the Kronecker product.

Therefore, $f(x) = \text{vec}(W)^\top (\phi'(x) \otimes \phi(x))$

But computing Kronecker products and then dot products is still computationally expensive.

Alternatively, noting that:

$$f(x) = \text{tr}\left(W^\top \left(\phi(x)\phi'(x)^\top\right)\right) = \phi'(x)^\top W^\top \phi(x)$$

So if we have $W \in \mathbb{R}^{D \times D}$, then:

$$f(x) = \phi'(x)^\top W^\top \phi(x)$$

If we set W to be a rank-one matrix, i.e., $W = w_1 w_2^\top$ for $w_1, w_2 \in \mathbb{R}^D$, then:

$$f(x) = \phi'(x)^\top w_2 w_1^\top \phi(x) = \left(w_1^\top \phi(x)\right)\left(\phi'(x)^\top w_2\right)$$

Now, $f(x) = \left(w_1^\top \phi(x)\right)\left(\phi'(x)^\top w_2\right)$

This expression can be computed efficiently on-the-fly:

1. Compute $a = w_1^\top \phi(x)$, which is an inner product of two vectors. 2. Compute $b = \phi'(x)^\top w_2$, which is an inner product of two vectors. 3. Multiply $f(x) = a \cdot b$.

Now, to compute $c = \inf_{x\in\mathcal{X}} f(x)$, we can set up an optimization problem:

$$\begin{aligned} \text{Minimize}_{x\in\mathcal{X}} \quad & f(x) = \left(w_1^\top \phi(x)\right)\left(\phi'(x)^\top w_2\right) \\ \text{Subject to} \quad & x \in \mathcal{X} \end{aligned}$$

If ϕ and ϕ' are known and differentiable, and $\mathcal{X}$ is continuous, we can compute the gradient of $f(x)$ with respect to x and use gradient-based optimization methods.

Algorithm Steps:

1. **Initialization:** - Choose initial $x_0 \in \mathcal{X}$.

2. **Compute a and b:** - $a = w_1^\top \phi(x_0)$ - $b = \phi'(x_0)^\top w_2$

3. **Compute $f(x_0)$:** - $f(x_0) = a \cdot b$

4. **Compute Gradient $\nabla f(x_0)$:** - Compute the gradients $\nabla_x a = \nabla_x \left(w_1^\top \phi(x)\right)$ - Compute $\nabla_x b = \nabla_x \left(\phi'(x)^\top w_2\right)$ - Use the product rule:

$$\nabla f(x) = (\nabla_x a) \cdot b + a \cdot (\nabla_x b)$$

5. **Update x:** - Use an optimization step, e.g., $x_{k+1} = x_k - \eta \nabla f(x_k)$, where η is the learning rate.

6. **Iterate:** - Repeat steps 2-5 until convergence.

7. **Return c:** - Set $c = f(x^*)$, where x^* is the value of x at convergence.

Advantages:

- This method computes $f(x)$ and its gradient on-the-fly without storing the full matrices. - Inner products and gradients are computed using vector operations, which are efficient.

Assumptions:

- The mappings ϕ and ϕ' are differentiable with respect to x. - The domain $\mathcal{X}$ is continuous or can be appropriately handled. - The optimization problem is tractable.

Example

Suppose $\phi(x) = x$ and $\phi'(x) = x$, with $x \in \mathbb{R}^D$, and $w_1 = w_2 = w$.

Then $f(x) = (w^\top x)(w^\top x) = \left(w^\top x\right)^2$

Our optimization problem becomes:

$$\begin{aligned} \text{Minimize}_{x\in\mathcal{X}} \quad & f(x) = \left(w^\top x\right)^2 \\ \text{Subject to} \quad & x \in \mathcal{X} \end{aligned}$$

This is a quadratic function in x. If we want to minimize $f(x)$, and $\mathcal{X}$ is unconstrained, the minimum is achieved when $x = 0$, assuming $w^\top x = 0$.

However, if $\mathcal{X}$ is constrained (e.g., x within some domain), we can use gradient descent to find the minimal $f(x)$.

Final Answer for Part (b)

To efficiently compute $c = \inf_{x\in\mathcal{X}} \left\{w^\top \left(\phi(x)\phi'(x)^\top\right)\right\}$ on-the-fly, we can:

1. Express $w^\top \left(\phi(x)\phi'(x)^\top\right)$ in a form that can be computed using vector operations without storing large matrices, for example, as $f(x) = \left(w_1^\top \phi(x)\right)\left(\phi'(x)^\top w_2\right)$.
2. Set up an optimization problem to minimize $f(x)$ over $x \in \mathcal{X}$, exploiting the differentiable structure of ϕ and ϕ'.
3. Use gradient-based optimization methods to iteratively compute $f(x)$ and update x, each time computing $f(x)$ and $\nabla f(x)$ on-the-fly.
4. Since we avoid storing the full $\phi(x)\phi'(x)^\top$ matrices and instead use vector inner products and gradient computations, the algorithm is memory-efficient.

Summary

By transforming the problem into an optimization task that uses vector operations and avoids explicit representation of high-dimensional matrices, we can efficiently compute the infimum c on-the-fly while exploiting the convexity and polyhedral structure of the cone $\mathcal{C}$.

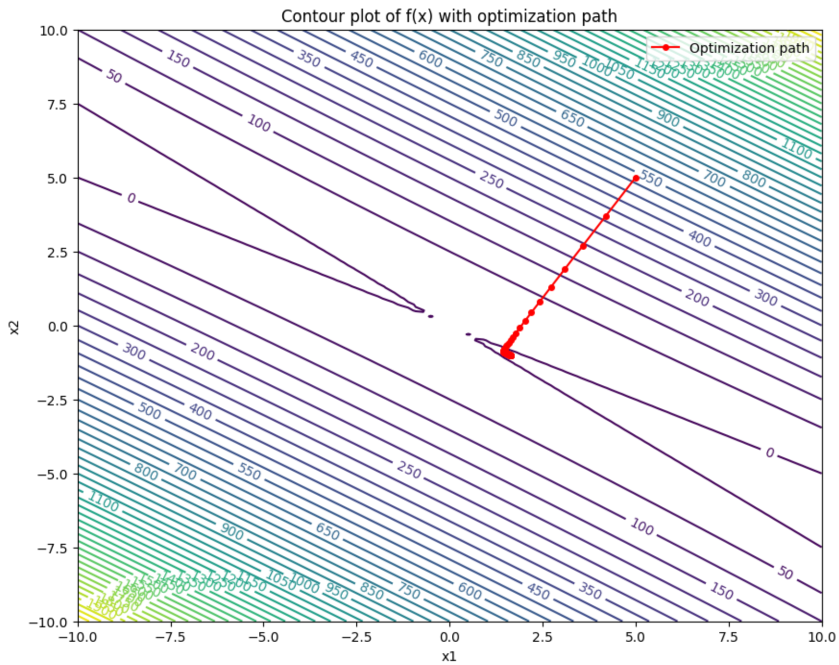
```
    import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# Define the feature mappings phi and phi'
def phi(x):
    # For simplicity, phi(x) = x
    return x

def phi_prime(x):
    # For simplicity, phi'(x) = x
    return x

# Define the function f(x) = (w1^T phi(x)) * (w2^T phi'(x))
def f(x, w1, w2):
    phi_x = phi(x)
    phi_prime_x = phi_prime(x)
    a = np.dot(w1, phi_x)
    b = np.dot(w2, phi_prime_x)
    return a * b

# Compute the gradient of f with respect to x
def grad_f(x, w1, w2):
    phi_x = phi(x)
    phi_prime_x = phi_prime(x)
    a = np.dot(w1, phi_x)
    b = np.dot(w2, phi_prime_x)
```

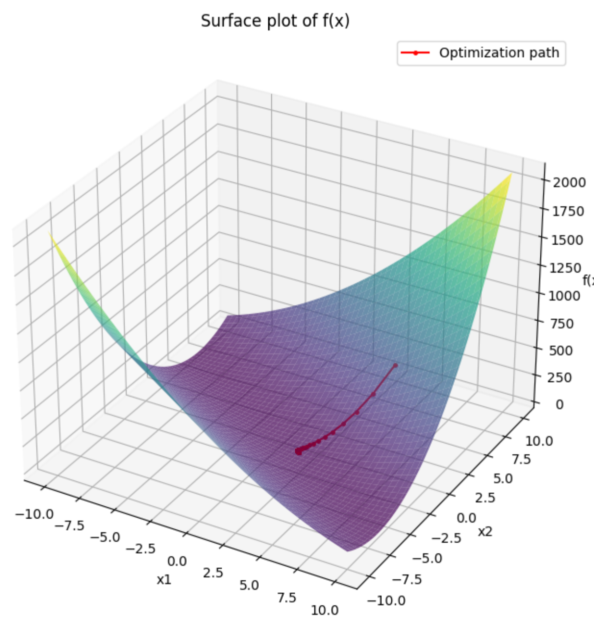
```
    grad_a = w1
    grad_b = w2
    grad_f = grad_a * b + a * grad_b
    return grad_f

# Set weight vectors w1 and w2
w1 = np.array([1.0, 2.0])
w2 = np.array([3.0, 4.0])

# Initialize x
x_init = np.array([5.0, 5.0])

# Set learning rate and number of iterations
learning_rate = 0.01
num_iterations = 100

# Gradient descent optimization
def gradient_descent(x_init, w1, w2, learning_rate, num_iterations):
    x = x_init.copy()
    x_history = [x.copy()]
    f_history = [f(x, w1, w2)]

    for i in range(num_iterations):
        gradient = grad_f(x, w1, w2)
        x -= learning_rate * gradient
        x_history.append(x.copy())
        f_history.append(f(x, w1, w2))
    return x, np.array(x_history), f_history

# Perform optimization
x_min, x_history, f_history = gradient_descent(x_init, w1, w2, learning_rate, nu

print("Minimum value of f(x):", f(x_min, w1, w2))
print("x at minimum:", x_min)

# Visualization
# Create a meshgrid for plotting f(x) over the domain
X_range = np.linspace(-10, 10, 100)
Y_range = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(X_range, Y_range)
Z = np.zeros_like(X)

# Compute f(x) over the grid
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        x_point = np.array([X[i, j], Y[i, j]])
```

```
        Z[i, j] = f(x_point, w1, w2)

# Plot the contour and the optimization path
fig, ax = plt.subplots(figsize=(10, 8))
CS = ax.contour(X, Y, Z, levels=50, cmap='viridis')
ax.clabel(CS, inline=1, fontsize=10)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_title('Contour plot of f(x) with optimization path')

# Plot the optimization path
x1_history = x_history[:, 0]
x2_history = x_history[:, 1]
ax.plot(x1_history, x2_history, 'ro-', markersize=4, label='Optimization path')
ax.legend()

plt.show()

# 3D Surface plot
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('f(x)')
ax.set_title('Surface plot of f(x)')

# Plot the optimization path in 3D
ax.plot(x1_history, x2_history, f_history, 'r.-', markersize=5, label='Optimizat
ax.legend()

plt.show()
```

Conclusion

In this analysis, for Part (a), we determined that the cone of $n \times n$ positive semidefinite matrices $\mathcal{S}^n$ is not a polyhedral cone when $n > 1$, due to its infinite dimensionality and the fact that it cannot be generated by a finite set of vectors. For Part (b), we designed an algorithm that computes $c = \inf_{x \in \mathcal{X}} \left\{ w^\top \left(\phi(x) \phi'(x)^\top \right) \right\}$ on-the-fly by exploiting the structure of the feature mappings and using optimization techniques that avoid storing large matrices, thereby making the computation efficient.





—

References:

- Boyd, S., Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press. - Schölkopf, B., Smola, A. J. (2002). *Learning with Kernels.*

MIT Press. - Rockafellar, R. T. (1997). *Convex Analysis*. Princeton University Press.