



Project Guide for Notions of Positivity Project
Written by Karim El-Sharkawy of Purdue Mathematics

Introduction

This document provides a simple map for anyone wanting to familiarize themselves with this project. This can be used for review or as a guide. I am working on this project with Darshini Rajamani ([LinkedIn](#)), Abbas Dohadwala ([LinkedIn](#)), and Luke Luschwitz ([LinkedIn](#)). Professor Thomas Sinclair ([Website](#)) of Purdue Mathematics supervises this project. You can access all the code at the [GitHub](#) and also find the profiles of contributors

An understanding of linear algebra concepts (norm, adjoint, isomorphism, etc) will be needed before tackling the theory of this project. Experience in a proof-based linear algebra course is helpful before reading the Professor's notes. Some concepts in the professor's notes are expanded upon below. Learning new material as you go is good so it's not overwhelming

Principal topics

1. Definition of Positive Matrices
 - a.  What Does It Mean For a Matrix to be POSITIVE? The Practical Guide to ...
 - b.  The Practical Guide to Semidefinite Programming (2/4)
2. Further explanations on other concepts are found in this book: [Convex Optimization](#)
[Euclidean Distance Geometry](#) ← Underlines indicate clickable links
 - a. Cones (77), Half-Space (58), PSD Cones (90), etc.
 - b. The links below have less condensed explanations of important concepts
3. Convex Optimization ([Wikipedia](#))
 - a. Also talks about extensions!
4. Ordered Vector Spaces ([Wikipedia](#))
 - a. Also talks about positive cones!
5. Preorders & Partial Orders ([Wikipedia](#))
6. Positive Lifting ([Paper](#))
 - a. From the abstract: "A lift of a convex set is a higher-dimensional convex set that projects onto the original set. Many convex sets have lifts that are dramatically

simpler to describe than the original set. Finding such simple lifts has significant algorithmic implications, particularly for optimization problems.”

- b. You don't have to read the paper, but thought this definition was best and explains why we care about them
7. Hyperplane ([Wikipedia](#))
- a. Read the 'Affine hyperplanes' section

The Code

1. Understanding linear programming: [YouTube: The Art of Linear Programming](#)
 - a. This is not necessary to watch, but it explains what we're essentially doing
2. Semidefinite Programming ([Wikipedia](#))
3. We use Python along with multiple packages that a basic understanding of is needed
4. NumPy: [YouTube: Python NumPy Tutorial for Beginners](#)
 - a. Follow along with the video: [GitHub - KeithGalli/NumPy](#)
 - b. [API Reference](#)
5. SciPy: [SciPy Manual](#)
 - a. More specifically, [scipy.optimize.linprog](#)
6. Matplotlib
 - a. [YouTube: Matplotlib Full Python Course - Data Science Fundamentals](#)
 - b. [Plot types](#)
7. Random: Used to generate random numbers
8. Scikit-learn will be explained later


Explaining the code

1. The first code block recursively creates a positive semidefinite matrix that satisfies the $E(2,2)$, linearity, and positivity constraints
 - a. Important note: Currently, we're only working on 4 x 4 matrices. The code will be generalized later for any $N \times N$ size

- b. The E(2,2) constraint checks that the sum of the first two values of every row equals the sum of the last two values
 - c. The linearity constraint checks that the sum of the first two rows equals the sum of the last two rows
 - d. The positivity constraint checks if the matrix is positive. Please watch the videos referenced above for the definition of a positive matrix
 - i. A matrix that isn't positive is discarded, since we need all matrices that are mapped to be positive
2. The second block checks for liftability (extendability in this case too), using the 'linprog' function in SciPy. How this works is that ...
 - a. We tell it how many mappings to create, and after creating all the mappings along with their identifications (extendable or non-extendable), it appends them to a list depending on whether they're extendable or not
 - b. It then saves the lists as files. This is used to decrease the wait time and with testing
 - c. 'c' is "The coefficients of the linear objective function to be minimized". This is then equated to something, 'mat' in this case.
 - d. 'A_ub' is "The inequality constraint matrix. Each row of A_ub specifies the coefficients of a linear inequality constraint on c" and 'b_ub' is "The inequality constraint vector. Each element represents an upper bound on the corresponding value of A_ub"
 - e. 'A_eq' is "The equality constraint matrix. Each row of A_eq specifies the coefficients of a linear equality constraint on c" and 'b_eq' is "The equality constraint vector. Each element of A_eq @ x must equal the corresponding element of b_eq"
3. The next few blocks save the mappings into files, load the files again if needed, and then print the number of extendable and non-extendable mappings
4. The next block finds the farthest non-extendable matrix from the extendable matrices. This is done through the use of the norm function (`np.linalg.norm`)

5. We'll look into the machine learning code shortly, make sure to understand what the code is doing and why it works. Check the last bullet point of the next section to see a summary of the ML code
6. Each block has the contributors' name on it, check the file to know who created what

Machine learning

1. There are different types of ways the computer learns, we'll be using the binary classification method
 - a. For how ML works, check out this [tutorial](#). This video is incredibly useful and will give you all the needed information
 - b. This goes over basic ML, has coding sections using NumPy and Matplotlib which would be good refreshers, uses Scikit-learn and imbalanced-learn, and covers important topics like Knn and Bayes' Law. You should watch at least up to 1:08:00 and from 1:29:12 to 1:47:57
2. Linear classifier ([Wikipedia](#))
3. Support Vector Machines ([Wikipedia](#))
 - a. Please watch 1:29:13 to 1:39:44 of the video in 1a
 - b. We are using the support vector classifier (SVC) from scikit-learn:
[sklearn.svm.SVC](#)
 - i. Watch the video in 4b first to learn about scikit-learn
 - c. [Formulating the Support Vector Machine Optimization Problem](#)
4. Scikit-learn
 - a. [scikit-learn documentation](#)
 - b.  [Scikit-learn Crash Course - Machine Learning Library for Python](#)
5. The last block of code uses SciKit-learn
 - a. It first ensures the (non)extendable mappings and farthest non-extendable matrix are seen as NumPy arrays
 - b. After adding the farthest non-extendable matrix to the extendable mappings (WHY DOES IT DO THIS?) and flattening all the matrices to a 1D array (vector), it generates labels of 0 or 1

- c. We use the support vector machine technique to classify the matrices. Finally, a hyperplane is created
- d.
- 6. The most important part of this code is understanding exactly what it is doing and how it sees the matrices. Just because it gives you an output, doesn't mean it's what you want
 - a. Note: while learning ML for the first time, you may need to be slightly familiar with other packages like matplotlib and pandas

Proof of working code

The most important part, after all the work, is to check whether what the code is giving us is correct. In order to do this, one has to check whether at least one of the true classifiers given by the code is actually a true classifier. The proof is below exists for one of the true classifiers, found in the 'trueClassifiersGood.npy' file

```
trueClassifiers[5]:
[[-0. -7. -7.  0.]
 [-0. -4. -4. -0.]
 [ 0. -0.  0. -0.]
 [-0. -10. -11.  0.]]
totalMappingsInClass1: 99964
totalMappingsInClass2: 36
```

$$\begin{array}{l} -7b_1 - 7c_1 \\ -4b_2 - 4c_2 \\ -10b_4 - 11c_4 \end{array} \leq 0 \quad *y$$

$$-a_j - b \leq c_j$$

1) If $b_i = c_i$, then everything is satisfied if both are Non-Neg

$$2) -10b_4 \leq 11c_4$$

$$c_4 \geq \frac{-11}{10}b_4 \text{ for } *y \text{ only}$$

3) $b_i = -c_i$ satisfies everything if b is non-pos

Theorem

If there exist two closed cones where one is contained in the other, then they're different exactly when there is a separating hyperplane

Proof

Definition:

Let $\mathcal{C}_2, \mathcal{C}_1$ be two cones, H is a Separating Hyperplane if $\langle x, n_H \rangle \geq 0 \forall x \in \mathcal{C}_1$ and $\langle y, n_H \rangle < 0$ for some $y \in \mathcal{C}_2$

\nwarrow
a normal vector

$$(\Leftarrow) \quad \forall x \in \mathcal{Y}_1, \langle x, r_H \rangle \geq 0, \exists y \in \mathcal{Y}_2 \text{ s.t. } \langle y, r_H \rangle < 0$$

— Hence, $\gamma \in \mathcal{G}_1$, and $\mathcal{G}_2 \neq \mathcal{G}_1$

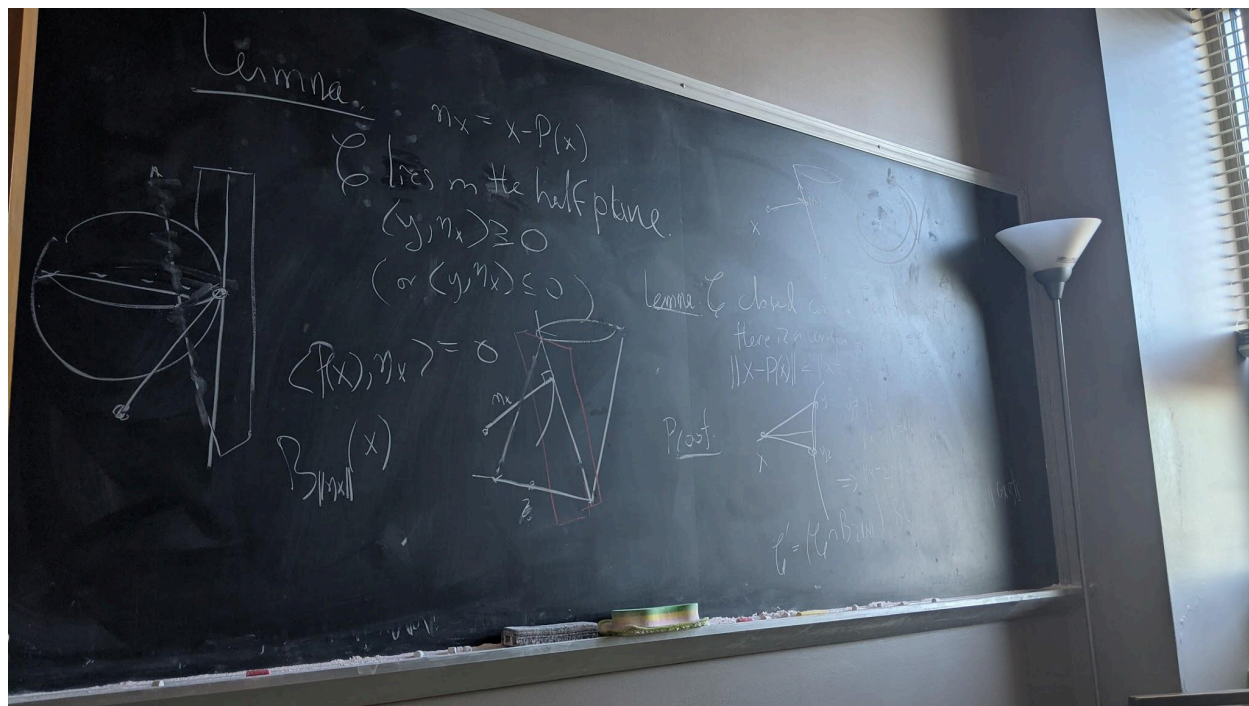
(\Rightarrow) Farkas' Lemma

Proved by Professor Sinclair, also a good video here: [YouTube Convex cones and Farkas' lemma](#)

1) The other direction is tough to prove. This is known as Farkas' Lemma ([Wikipedia](#))

(Proof)

Geometric proof/interpretation



Similarities

Something I noticed, when printing out their RREFs, almost all of them, whether extendable or not, had the same RREF:

$$[1, 0, 0, 1]$$

$$[0, 1, 0, 1]$$

$$[0, 0, 1, -1]$$

$$[0, 0, 0, 0]$$

With the 1st, 2nd, and 3rd rows being pivot rows (or columns?). I made 400 mappings, and 355 of them had this RREF form, regardless of their extendability. Doesn't help much, but it's an interesting beginning.

Inverses don't offer any differences either. Interestingly, in both extendable and non extendable matrices, there are (3~4 times) more singular (noninvertible) matrices than invertible ones

Interestingly, when finding the null spaces of the matrices, most of them, which are (suspected) to be the same ones with the RREF above, have a null space of either:

$$[-0.5]$$

$$[-0.5] (1)$$

$$[0.5]$$

$$[0.5]$$

$$\begin{bmatrix} 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{bmatrix} (2)$$

No matter the type of null space, they have the same RREF form. This still needs to be proven.

Still need to look into: linear dependence, column spaces,

Current/ongoing tasks

For the theory/practice

- + testing and proving that conjecture (untangle and apply a map, then the map must be extendable) - Abbas, karim, and Darsh
- + Detecting patterns: can you create them by hand? - Karim and Darsh
- + Review Bolzano weierstrass theorem - karim

For the code

- + Using the same vector to test multiple different sets to see if they still work - Luke and karim
- + Plotting the cones - karim

$P(x)$ = perfection

() Means norm

Define $n(x) = x - P(x)$

Goal: C lies in a half-space by proving $(y, n(x))$ is larger than or equal to OR smaller than or equal to 0

$(P(x), n(x)) = 0$ (the norm)