FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Nathan Chappell

Minkowski-Weyl Theorem

Department of Applied Mathematics

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....   signature of the author

Dedication.

Title: Minkowski-Weyl Theorem

Author: Nathan Chappell

Department: Department of Applied Mathematics

Supervisor: Hans Raj Tiwary, Department of Applied Mathematics

Abstract: The Minkowski-Weyl Theorem is proven for polyhedra by first showing the proof for cones, then the reductions from polyhedra to cones. The proof follows Ziegler [1], and uses Fourier-Motzkin elimination. A C++ implementation is given for the enumeration algorithm suggested by the proof.

Keywords: Minkowski-Weyl Theorem polyhedra Fourier-Motzkin C++

Contents

Introduction

Polyhedra are fundamental mathematical objects. Two ways of describing poly-hedra are:

1. A finite intersection of half-spaces

2. The *Minkowski-Sum* of the *convex-hull* of a finite set of rays and a finite set of points

The Minkowski-Weyl Theoremis a fundamental result in the theory of polyhedra. It states that both means of representation are equivalent. The proof given here is algorithmic in nature, using a technique known as *Fourier-Motzkin elimination*. The algorithm implied by the proof is then implemented in C++.

1. Minkowski-Weyl Theorem

1.1 Polyhedra

Definition 1 (Non-negative Linear Combination)**.** Let $U \in \mathbb{R}^{d \times p}$, $\mathbf{t} \in \mathbb{R}^p, \mathbf{t} \geq \mathbf{0}$, then $\sum_{1 \leq j \leq p} t_j U^j = U\mathbf{t}$ is called a *non-negative linear combination of U*.

Definition 2 (V-Cone)**.** Let $U \in \mathbb{R}^{d \times p}$. The set of all non-negative linear combinations of U is denoted $\mathrm{cone}(U)$. Such a set is called a *V-Cone.*

Definition 3 (Convex Combination)**.** Let $V \in \mathbb{R}^{d \times n}$, $\boldsymbol{\lambda} \in \mathbb{R}^n, \boldsymbol{\lambda} \geq \mathbf{0}, \sum_{1 \leq j \leq n} \lambda_j = 1$, then $\sum_{1 \leq j \leq n} \lambda_j V^j$ is called a *convex combination of V. The set of all convex combinations of V is denoted* $\mathrm{conv}(V)$.

Definition 4 (V-Polyhedron)**.** Let $V \in \mathbb{R}^{d \times n}$, $U \in \mathbb{R}^{d \times p}$. Then the set

$$\{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \mathrm{cone}(U), \mathbf{y} \in \mathrm{conv}(V)\}$$

is called a *V-Polyhedron.*

Definition 5 (H-Polyhedron)**.** Let $A \in \mathbb{R}^{m \times d}$, $\mathbf{b} \in \mathbb{R}^m$. Then the set

$$\left\{ \mathbf{x} \in \mathbb{R}^d \,\middle|\, A\mathbf{x} \leq \mathbf{b} \right\}$$

is called an *H-Polyhedron.*

Definition 6 (H-Cone)**.** Let $A \in \mathbb{R}^{m \times d}$. Then the set

$$\left\{ \mathbf{x} \in \mathbb{R}^d \,\middle|\, A\mathbf{x} \leq \mathbf{0} \right\}$$

is called an *H-Cone.*

A simple but useful property of cones is that they are closed under addition and positive scaling.

Proposition 1. *Let C be either an H-Cone or a V-Cone, for each i $\mathbf{x}^i \in C$, and $c_i \geq 0$. Then:*

$$\sum_i c_i \mathbf{x}^i \in C$$

Proof. First we prove Proposition 1 for H-Cones, then for V-Cones. If, for each i, $A\mathbf{x}^i \leq \mathbf{0}$, then $A(c_i \mathbf{x}^i) = t_i A\mathbf{x}^i \leq \mathbf{0}$, and

$$A \left(\sum_i c_i \mathbf{x}^i \right) = \sum_i A(c_i \mathbf{x}^i) = \sum_i c_i A\mathbf{x}^i \leq \sum_i \mathbf{0} \leq \mathbf{0}$$

So, $\sum_i c_i \mathbf{x}^i \in C$ when C is an H-Cone. Next, suppose that $C = \mathrm{cone}(U)$, and for each i, $\exists \mathbf{t}_i \geq \mathbf{0} : \mathbf{x}^i = U\mathbf{t}_i$. Then $c_i \mathbf{t}_i \geq \mathbf{0}$, and $\sum_i c_i \mathbf{t}_i \geq \mathbf{0}$. Therefore

$$\sum_i c_i \mathbf{x}^i = \sum_i c_i U\mathbf{t}_i = \sum_i U(c_i \mathbf{t}_i) = U \left(\sum_i c_i \mathbf{t}_i \right)$$

So, $\sum_i c_i \mathbf{x}^i \in C$ when C is a V-Cone. $\qquad \square$

This proposition will be used in the following way: if we wish to show that $\sum_i c_i \mathbf{x}^i$ in a member of some cone C, it suffices to show that, for each i, $c_i \geq 0$ and $\mathbf{x}^i \in C$.

1.2 Minkowski-Weyl Theorem

The following theorem is the basic result to be proved in this thesis, which states that V-Polyhedra and H-Polyhedra are two different representations of the same objects.

Theorem 1 (Minkowski-Weyl Theorem). *Every V-Polyhedron is an H-Polyhedron, and every H-Polyhedron is a V-Polyhedron.*

The proof proceeds by first showing that V-Cones are representable as H-Cones, and H-Cones are representable as V-Cones. Then it is shown that the case of polyhedra can be reduced to cones.

Theorem 2 (Minkowski-Weyl Theorem for Cones). *Every V-Cone is an H-Cone, and every H-Cone is a V-Cone.*

2. Proof of the Minkowski-Weyl Theorem

2.1 Every V-Cone is an H-Cone

Definition 7 (Coordinate Projection)**.** Let I be the identity matrix. Then the matrix I' formed by deleting some rows from I is called a **coordinate-projection**.

The proof rests on the following two propostions:

(V1) Every V-Cone is a coordinate-projection of an H-Cone.

(V2) Every coordinate-projection of an H-Cone is an H-Cone.

Proof. Given (V1) and (V2), the proof follows simply. Given a V-Cone, we use (V1), to get a description involving coordinate-projection of an H-Cone. Then we can apply (V2) in order to get an H-Cone. $\square$

Proof of (V1). We prove that every V-Cone is a coordinate-projection of an H-Cone, by giving an explicit formula. Let $U \in \mathbb{R}^{d \times p}$, and observe that

$$\text{cone}(U) = \{U\mathbf{t} \mid \mathbf{t} \in \mathbb{R}^p, \, \mathbf{t} \geq \mathbf{0}\} = \left\{\mathbf{x} \in \mathbb{R}^d \mid (\exists \mathbf{t} \in \mathbb{R}^p) \, \mathbf{x} = U\mathbf{t}, \, \mathbf{t} \geq \mathbf{0}\right\}$$

We will collect $\mathbf{t}$ and $\mathbf{x}$ on the left side of the inequality, treating $\mathbf{t}$ as a variable and expressing its contraints as linear inequalities, then project away the coordinates corresponding to $\mathbf{t}$. The following expression takes one step:

$$\mathbf{t} \geq \mathbf{0} \Leftrightarrow -I\mathbf{t} \leq \mathbf{0} \tag{2.1}$$

And using the equality: $a = 0 \Leftrightarrow a \leq 0 \wedge -a \leq 0$, and block matrix notation, we take the second step.

$$\mathbf{x} = U\mathbf{t} \Leftrightarrow \mathbf{x} - U\mathbf{t} = \mathbf{0} \Leftrightarrow \begin{pmatrix} I & -U \\ -I & U \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0} \tag{2.2}$$

Comparing (2.1) and (2.2), we define a new matrix $A' \in \mathbb{R}^{(p+2d) \times (d+p)}$:

$$A' = \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix} \tag{2.3}$$

then we can rewrite $\text{cone}(U)$:

$$\text{cone}(U) = \left\{\mathbf{x} \in \mathbb{R}^d \,\middle|\, A' \begin{pmatrix} \mathbf{x} \\ \mathbf{t} \end{pmatrix} \leq \mathbf{0}\right\}$$

Let $\Pi \in \{0,1\}^{d \times (d+p)}$ be the identity matrix in $\mathbb{R}^{(d+p) \times (d+p)}$, but with the last p-rows deleted. Then Π is a coordinate projection, and the above expression can be written:

$$\text{cone}(U) = \Pi\left(\left\{\mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0}\right\}\right) \tag{2.4}$$

This is a coordinate projection of an H-Cone, and (V1) is shown. $\square$

To prove (V2), we use two separate propositions.

Proposition 2. *Let $B \in \mathbb{R}^{m' \times (d+p)}$, B' be B with the last p columns deleted, and Π the identity matrix with the last p rows deleted (i.e. $B' = \Pi B$). Furthermore, suppose that the last p columns of B are $\mathbf{0}$. Then*

$$\Pi \left(\left\{ \mathbf{y} \in \mathbb{R}^{d+p} \mid B\mathbf{y} \leq \mathbf{0} \right\} \right) = \left\{ \mathbf{x} \in \mathbb{R}^d \mid B'\mathbf{x} \leq \mathbf{0} \right\}$$

Proof. Recall that $B\mathbf{y} \leq \mathbf{0}$ means that $(\forall i) \langle B_i, \mathbf{y} \rangle \leq 0$. By the way we've defined B, any row B_i of B can be written $(B'_i, \mathbf{0})$, with $\mathbf{0} \in \mathbb{R}^p$. Rewriting $\mathbf{y} \in \mathbb{R}^{d+p}$ as $(\mathbf{x}, \mathbf{w})$ with $\mathbf{x} \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^p$, so that $\mathbf{x} = \Pi(\mathbf{y})$. Then

$$\langle B, \mathbf{y} \rangle = \langle (B'_i, \mathbf{0}), (\mathbf{x}, \mathbf{w}) \rangle = \langle B'_i, \mathbf{x} \rangle = \langle B'_i, \Pi(\mathbf{y}) \rangle$$

It follows that

$$\langle B_i, \mathbf{y} \rangle \leq 0 \Leftrightarrow \langle B'_i, \Pi(\mathbf{y}) \rangle \leq 0$$

Since B_i is an arbitrary row of B, the proposition is shown. $\square$

In order to use the above proposition, we need a matrix with $\mathbf{0}$ columns. The next proposition shows us how to do so, one column at a time.

Proposition 3. *Let $B \in \mathbb{R}^{m_1 \times (d+p)}$, $1 \leq k \leq p$, and $\mathbf{x} = \sum_{i \neq k} x_i \mathbf{e}_i$. Then there exists a matrix $B' \in \mathbb{R}^{m_2 \times (d+p)}$ with the following properties:*

1. *Every row of B' is a positive linear combination of rows of B.*

2. *m_2 is finite.*

3. *The k-th column of B' is $\mathbf{0}$.*

4. *$(\exists t) B(\mathbf{x} + t\mathbf{e}_k) \leq \mathbf{0} \Leftrightarrow B'\mathbf{x} \leq \mathbf{0}$*

Proof. Partition the rows of B as follows:

$$
\begin{aligned}
P &= i \mid B_i^k > 0 \\
N &= j \mid B_j^k < 0 \\
Z &= l \mid B_l^k = 0
\end{aligned}
$$

Then let B' be a matrix with rows of the following forms:

$$
\begin{aligned}
C_l &= B_l & \mid l \in Z \\
C_{ij} &= B_i^k B_j - B_j^k B_i \mid i \in P, j \in N
\end{aligned}
$$

1 and *2* are clear. *3* can be seen from:

$$\langle C_l, \mathbf{e}_k \rangle = 0$$

$$\langle C_{ij}, \mathbf{e}_k \rangle = \left\langle B_i^k B_j - B_j^k B_i, \mathbf{e}_k \right\rangle = B_i^k B_j^k - B_j^k B_i^k = 0 \tag{2.5}$$

6

The right direction of 4 is shown in the following calculations. Because $B_l^k = 0$:

$$\langle B_l, \mathbf{x} + t\mathbf{e}_k \rangle = \langle B_l, \mathbf{x} \rangle + tB_l^k = \langle B_l, \mathbf{x} \rangle = \langle C_l, \mathbf{x} \rangle$$

So:

$$\langle B_l, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0 \Rightarrow \langle C_l, \mathbf{x} \rangle \leq 0$$

For rows indexed by P, N, we observe (2.15), and have:

$$\left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} + t\mathbf{e}_k \right\rangle = \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle$$

Now, we use property 1:

$$\langle B_i, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0, \ \langle B_j, \mathbf{x} + t\mathbf{e}_k \rangle \leq 0 \Rightarrow \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} + t\mathbf{e}_k \right\rangle \leq 0$$

Therefore

$$\left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle \leq 0$$

Now suppose that $B'\mathbf{x} \leq \mathbf{0}$. The task is to find a t so that $B\mathbf{x} \leq \mathbf{0}$. Looking at (2.15), any choice of t we make will be okay for rows indexed by Z. So the task is to find a t so that the inequality holds for rows indexed by P and N. Observe

$$\begin{aligned}
\forall i \in P, \forall j \in N \quad & \left\langle B_i^k B_j - B_j^k B_i, \mathbf{x} \right\rangle \leq 0 \Leftrightarrow \\
\forall i \in P, \forall j \in N \quad & \left\langle B_i^k B_j, \mathbf{x} \right\rangle \leq \left\langle B_j^k B_i, \mathbf{x} \right\rangle \Leftrightarrow \\
\forall i \in P, \forall j \in N \quad & \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq \left\langle B_i / B_i^k, \mathbf{x} \right\rangle \Leftrightarrow \\
& \min_{j \in N} \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq \max_{i \in P} \left\langle B_i / B_i^k, \mathbf{x} \right\rangle
\end{aligned}$$

Note that the third inequality changes directions because $B_j^k < 0$. Now we choose t to lie in this last interval, and show that we can use it to satisfy all of the constraints given by B. So, we have a t such that

$$\min_{j \in N} \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq t \geq \max_{i \in P} \left\langle B_i / B_i^k, \mathbf{x} \right\rangle$$

In particular,

$$\begin{aligned}
(\forall j \in N) \quad & \left\langle B_j / B_j^k, \mathbf{x} \right\rangle \geq t \Rightarrow \\
(\forall j \in N) \quad & \langle B_j, \mathbf{x} \rangle - B_j^k t \leq 0
\end{aligned}$$

Again, the inequality changes directions because $B_j^k < 0$. Now consider a row B_j from B:

$$\langle B_j, \mathbf{x} - t\mathbf{e}_k \rangle = B_j \mathbf{x} - B_j^k t \leq 0$$

Similarly,

$$\begin{aligned}
(\forall i \in P) \quad & t \geq B_i / B_i^k \mathbf{x} \Rightarrow \\
(\forall i \in P) \quad & 0 \geq B_i \mathbf{x} - B_i^k t
\end{aligned}$$

Now consider a row B_i from B:

$$\langle B_i, \mathbf{x} - t\mathbf{e}_k \rangle = B_i \mathbf{x} - B_i^k t \leq 0$$

So, we've demonstrated that $\mathbf{x} - t\mathbf{e}_k$ satisfies all the constraints from B, and the left implication is shown. So 4 holds. □

Now to prove:

(V2) Every coordinate-projection of an H-Cone is an H-Cone.

proof of (V2). Here we prove the case that the coordinate projection is onto the first d of $d + p$ coordinates. Let $\left\{ \mathbf{y} \in \mathbb{R}^{d+p} : A'\mathbf{y} \leq \mathbf{0} \right\}$ be the H-Cone we need to project, and Π the coordinate-projection we need to apply (the identity matrix with the last p columns deleted). For each $1 \leq k \leq p$ we can use proposition 3 in an incremental manner, starting with A'.

> let $B_0 := A'$
> for $1 \leq k \leq p$
>   let $B_k :=$ result of proposition 2 applied to B_{k-1}, $\mathbf{e}_{d+k}$
> endfor
> return B_p

Consider the resulting B. Property *2* holds throughout, so B is finite. After each iteration, property *3* holds for k, so the k-th column is $\mathbf{0}$. Since each iteration only results from non-negative combinations of the result of the previous iteration (property *1*), once a column is $\mathbf{0}$ it remains so. Therefore, at the end of the process, the last p columns of B are all $\mathbf{0}$. Then, by proposition 2, we can apply Π to B by simply deleting the last p columns of B. Denote this resulting matrix A. We still need to check:

$$A'\mathbf{y} \leq \mathbf{0} \Leftrightarrow A(\Pi(\mathbf{y})) \leq \mathbf{0} \tag{2.6}$$

$$(\exists t_1) \ldots (\exists t_p) A'(\mathbf{x} + t_1 \mathbf{e}_{d+1} + \cdots + t_p \mathbf{e}_{d+p}) \leq \mathbf{0} \Leftrightarrow A\mathbf{x} \leq \mathbf{0} \tag{2.7}$$

Then, using (2.6) and (2.7), it is easy to see that:

$$\Pi \left\{ \mathbf{y} \in \mathbb{R}^{d+p} \mid A'\mathbf{y} \leq \mathbf{0} \right\} = \left\{ \mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0} \right\} \tag{2.8}$$

The key observation of this verification utilizes property *4* of proposition 3:

$$(\exists t) B(\mathbf{x} + t\mathbf{e}_k) \leq \mathbf{0} \Leftrightarrow B'\mathbf{x} \leq \mathbf{0}$$

In what follows, let $\mathbf{x} = \sum_{1 \leq j \leq d} x_j \mathbf{e}_j$. The above property is applied sequentially to the sets B_k as follows:

$$
\begin{aligned}
(\exists t_p)(\exists t_{p-1}) \ldots (\exists t_1) \quad & B_0(\mathbf{x} + t_1\mathbf{e}_p + t_2\mathbf{e}_{p-1} + \cdots + t_p\mathbf{e}_d) \leq \mathbf{0} \Leftrightarrow \\
(\exists t_p) \ldots (\exists t_2) \quad & B_1(\mathbf{x} + t_2\mathbf{e}_{d+2} + \cdots + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} \quad \Leftrightarrow \\
\vdots \quad\quad\quad & \quad\quad\quad \vdots \quad\quad\quad\quad\quad\quad\quad \vdots \\
(\exists t_p) \quad & B_{p-1}(\mathbf{x} + t_p\mathbf{e}_{d+p}) \leq \mathbf{0} \quad\quad\quad \Leftrightarrow \\
& B_p\mathbf{x} \leq \mathbf{0} \quad\quad\quad\quad\quad\quad\quad\quad \Leftrightarrow
\end{aligned}
$$

Because $A' = B_0$, and A is B_p with the last p columns deleted, (2.6) and (2.7) hold, therefore (2.8) holds, and the proof of (V2) is complete, and we've shown that a coordinate projection of an H-Cone is again an H-Cone. $\square$

With (V1) and (V2) proven, we are now certain that any V-Cone is also an H-Cone.

2.2 Every H-Cone is a V-Cone

Definition 8 (Coordinate Hyperplane)**.** A set of the form

$$\left\{ \mathbf{x} \in \mathbb{R}^{d+m} \mid \langle \mathbf{x}, \mathbf{e}_k \rangle = 0 \right\} = \left\{ \mathbf{x} \in \mathbb{R}^{d+m} \mid x_k = 0 \right\}$$

is called a *coordinate-hyperplane.*

We will use coordinate-hyperplanes in the following way. We consider a V-Cone intersected with some coordinate hyperplanes, and write it in the following way:

$$\left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U' \mathbf{t} \right\} \tag{2.9}$$

If we suppose that $U' \subset \mathbb{R}^{d+m}$, and Π is the identity matrix with the last m rows deleted, then this is just a convenient way of writing:

$$\Pi\Big(\operatorname{cone}(U') \cap \{x_{d+1} = 0\} \cap \cdots \cap \{x_{d+m} = 0\} \Big) \tag{2.10}$$

The proof rests on the following three propostions:

H1 Every H-Cone is a coordinate-projection of a V-Cone intersected with some coordinate hyperplanes.

H2 Every V-Cone intersected with a coordinate-hyperplane is a V-Cone

H3 Every coordinate-projection of a V-Cone is an V-Cone.

Proof. Given *H1*, *H2*, and *H3*, the proof follows simply. Given an H-Cone, we use *H1* to get a description involving the coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. We apply *H2* as many times as necessary to elimintate the intersections, then we can apply *H3* in order to get a V-Cone. $\qquad\square$

Proof of H1. Let $A \in \mathbb{R}^{m \times d}$, we now show that the H-Cone

$$\left\{ \mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0} \right\}$$

can be written as the projection of a V-Cone intersected with some hyperplanes. Define U':

$$U' = \left\{ \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix}, \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}, \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix}, 1 \leq j \leq d,\, 1 \leq i \leq m \right\}$$

Then we claim:

$$\left\{ \mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{0} \right\} = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U' \mathbf{t} \right\} \tag{2.11}$$

First, considering (2.9) and (2.10), observe that this is a coordinate-projection of a V-Cone intersected with some coordinate-hyperplanes. Next, we note that

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \leq j \leq d} x_j \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}$$

9

We can write this as a sum with all positive coefficients if we split up the x_j as follows:

$$x_j^+ = \begin{cases} x_j & x_j \geq 0 \\ 0 & x_j < 0 \end{cases} \qquad x_j^- = \begin{cases} 0 & x_j \geq 0 \\ -x_j & x_j < 0 \end{cases}$$

Then we have

$$\begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} = \sum_{1 \leq j \leq d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq j \leq d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} \tag{2.12}$$

where $x_j^+, x_j^- \geq 0$. Also observe that

$$A\mathbf{x} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \geq \mathbf{0}) \mid A\mathbf{x} + \mathbf{w} = \mathbf{0}$$

This can also be written

$$A\mathbf{x} \leq \mathbf{0} \Leftrightarrow (\exists \mathbf{w} \geq \mathbf{0}) \mid \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} \tag{2.13}$$

(2.12) and (2.13) together show

$$A\mathbf{x} \leq \mathbf{0} \Rightarrow (\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}$$

Conversely, suppose

$$(\exists \mathbf{t} \geq 0) \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} = U'\mathbf{t}$$

We would like to show that $A\mathbf{x} \leq \mathbf{0}$. Let x_j^+, x_j^-, w_i take the values of $\mathbf{t}$ that are coefficients of $\begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix}$, $\begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix}$, and $\begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix}$ respectively, and denote $x_j = x_j^+ - x_j^-$. Then we have

$$\begin{aligned} \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix} &= \sum_{1 \leq j \leq d} x_j^+ \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq j \leq d} x_j^- \begin{pmatrix} -\mathbf{e}_j \\ -A^j \end{pmatrix} + \sum_{1 \leq i \leq n} w_i \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix} \\ &= \sum_{1 \leq j \leq d} x_j \begin{pmatrix} \mathbf{e}_j \\ A^j \end{pmatrix} + \sum_{1 \leq i \leq n} w_i \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_i \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{x} \\ A\mathbf{x} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{w} \end{pmatrix} \end{aligned}$$

where $\mathbf{w} \geq \mathbf{0}$. By (2.13) we have $A\mathbf{x} \leq \mathbf{0}$. So (2.11) holds. $\qquad \square$

Note that U' can be written:

$$U' = \begin{pmatrix} \mathbf{0} & I & -I \\ I & A & -A \end{pmatrix} \tag{2.14}$$

The proof of *H2* relies upon the following proposition.

Proposition 4. *Let $Y \in \mathbb{R}^{(d+m) \times n_1}$, $1 \leq k \leq m$, and $\mathbf{x}$ satisfy $x_k = 0$. Then there exists a matrix $Y' \in \mathbb{R}^{(d+m) \times n_2}$ with the following properties:*

1. *Every column of Y' is a postive linear combination of rows of B.*

2. *n_2 is finite.*

3. *The k-th row of Y' is $\mathbf{0}$.*

4. *$(\exists \mathbf{t} \geq \mathbf{0})\mathbf{x} = Y\mathbf{t} \Leftrightarrow (\exists \mathbf{t}' \geq \mathbf{0})\mathbf{x} = Y'\mathbf{t}'$*

Recall that Y^i is the i-th column of Y, and Y_k^i is the element of Y in the i-th column and k-th row.

Proof. We partition the columns of Y:

$$P = i \ \mid Y_k^i > 0$$
$$N = j \ \mid Y_k^j < 0$$
$$Z = l \ \mid Y_k^l = 0$$

We then define Y':

$$Y' = \left\{ Y^l \mid l \in Z \right\} \cup \left\{ Y_k^i Y^j - Y_k^j Y^i \mid i \in P, j \in N \right\}$$

1 and *2* are clear. *3* can be seen from:

$$\left\langle Y'^l, \mathbf{e}^k \right\rangle = 0$$

$$\left\langle Y'^{ij}, \mathbf{e}^k \right\rangle = \left\langle Y_k^i Y^j - Y_k^j Y^i, \mathbf{e}^k \right\rangle = Y_k^i Y_k^j - Y_k^j Y_k^i = 0 \tag{2.15}$$

Before moving on to the proof, we first note how to write our vectors.

$$Y\mathbf{t} = \sum_{l \in Z} t_k Y^k + \sum_{i \in P} t_i Y^i + \sum_{j \in N} t_j Y^j$$

$$Y'\mathbf{t} = \sum_{l \in Z} t_k Y^k + \sum_{\substack{i \in P \\ j \in N}} t_{ij}(Y_k^i Y^j - Y_k^j Y^i)$$

Then, by proposition 1, to show that the proposition is true, we need only show that, given any $t_i, t_j \geq 0$ ($t_{ij} \geq 0$), there exists $t_{ij} \geq 0$ ($t_i, t_j \geq 0$) such that

$$\sum_{i \in P} t_i Y^i + \sum_{j \in N} t_j Y^j = \sum_{\substack{i \in P \\ j \in N}} t_{ij}(Y_k^i Y^j - Y_k^j Y^i) \tag{2.16}$$

Proposition 5. *Suppose that*

$$\sum_{i \in P} t_i Y_{d+1}^i + \sum_{j \in N} t_j Y_{d+1}^j = 0 \qquad Y_k^j < 0 < Y_k^i$$

Then the following holds

$$(t_i, t_j \geq 0) \Rightarrow (\exists t_{ij} \geq 0) \quad \text{such that (2.16) holds}$$
$$(t_{ij} \geq 0) \Rightarrow (\exists t_i, t_j \geq 0) \text{ such that (2.16) holds}$$

Proof. First note that if all $t_i = 0, t_j = 0$, then choosing $t_{ij} = 0$ satisfies (2.16), likewise if all $t_{ij} = 0$, then $t_i = 0, t_j = 0$ satisfies (2.16). So suppose that some $t_i \neq 0, t_j \neq 0, t_{ij} \neq 0$.

The right hand side of (2.16) can be written

$$\sum_{j \in N} \left(\sum_{i \in P} t_{ij} Y_k^i \right) Y^j + \sum_{i \in P} \left(- \sum_{j \in N} t_{ij} Y_k^j \right) Y^i$$

This means, given $t_{ij} \geq 0$, we can choose $t_j = \sum_{i \in P} t_{ij} Y_k^i$, and $t_i = - \sum_{j \in N} t_{ij} Y_k^j$, both of which are greater than 0.

Now suppose we have been given $t_i \geq 0, t_j \geq 0$. First observe:

$$0 = \sum_{i \in P} t_i Y_k^i + \sum_{j \in N} t_j Y_k^j \Rightarrow \sum_{i \in P} t_i Y_k^i = - \sum_{j \in N} t_j Y_k^j$$

Denote the value in this equality as σ, and note that $\sigma > 0$. Then

$$\sum_{i \in P} t_i Y^i = \frac{-\sum_{j \in N} t_j Y_k^j}{\sigma} \sum_{i \in P} t_i Y^i = \sum_{\substack{i \in P \\ j \in N}} -\frac{t_i t_j}{\sigma} Y_k^j Y^i$$

$$\sum_{j \in N} t_j Y^j = \frac{\sum_{i \in P} t_i Y_k^i}{\sigma} \sum_{j \in N} t_j Y^j = \sum_{\substack{i \in P \\ j \in N}} \frac{t_i t_j}{\sigma} Y_k^i Y^j$$

Combining these results, we have

$$\sum_{i \in P} t_i Y^i + \sum_{j \in N} t_j Y^j = \sum_{\substack{i \in P \\ j \in N}} \frac{t_i t_j}{\sigma} (Y_k^i Y^j - Y_k^j Y^i)$$

$\square$

Finally, we can conclude that, given $\mathbf{t} \geq \mathbf{0}$, if $Y\mathbf{t}$ has a 0 in the final coordinate, then we can write it as $Y'\mathbf{t}'$ where $\mathbf{t}' \geq \mathbf{0}$, and any non-negative linear combination of vectors from Y' can be written as a non-negative linear combination of vetors from Y, and will necessarily have the k-th coordinate be 0 by property *3*. So property *4* holds. $\square$

Proof of H2. In proposition 4, the assumption that $x_k = 0$ in property *4* creates the set $\text{cone}(Y) \cap \{\mathbf{x} \mid x_k = 0\}$. This set, by property *4*, is $\text{cone}(Y')$. $\square$

Proof of H3. We shall prove that the coordinate-projection of a V-Cone is again a V-Cone. Let Π be the relevant projection, then we have:

$$\Pi \{U\mathbf{t} \mid \mathbf{t} \geq \mathbf{0}\} = \{\Pi(U\mathbf{t}) \mid \mathbf{t} \geq \mathbf{0}\} = \{\Pi(U)\mathbf{t} \mid \mathbf{t} \geq \mathbf{0}\}$$

The last equality follows from associativity of matrix multiplication. Therefore,

$$\Pi\Big(\text{cone}(U)\Big) = \text{cone}\Big(\Pi(U)\Big)$$

$\square$

2.3 Reducing Polyhedra to Cones

Definition 9 (Hyperplane). Let $\mathbf{y} \in \mathbb{R}^d$, $c \in \mathbb{R}$. Then a set of the form

$$\left\{ \mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{y}, \mathbf{x} \rangle = c \right\}$$

is called a *hyperplane.*

2.3.1 H-Polyhedra $\leftrightarrow$ H-Cones

We show that an H-Polyhedron can be represented as the projection of an H-Cone intersected with a hyperplane. We begin by re-writing the expression:

$$A\mathbf{x} \leq \mathbf{b} \Leftrightarrow -\mathbf{b} + A\mathbf{x} \leq \mathbf{0} \Leftrightarrow \left[-\mathbf{b}|A \right] \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \leq \mathbf{0} \tag{2.17}$$

Proposition 6. *Every H-Polyhedron can be written as an H-Cone intersected with the set $\{\mathbf{x} \mid x_0 = 1\}$, and any H-Cone intersected with the set $\{\mathbf{x} \mid x_0 = 1\}$ is an H-Polyhedron.*

Proof. We extend (2.17):

$$\mathbf{x} \in \left\{ \mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} \leq \mathbf{b} \right\} \Leftrightarrow \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in \left\{ \mathbf{y} \in \mathbb{R}^{d+1} \mid \left[-\mathbf{b}|A \right] \mathbf{y} \leq \mathbf{0} \right\}$$

We conclude, given an H-Polyhedron, we can add an extra coordinate and prepend the vector $\mathbf{b}$ to the left of A, and later we can just move this column back to the right side of the inequality and drop the extra coordinate. $\square$

2.3.2 V-Polyhedra $\leftrightarrow$ V-Cone

We show that a V-Polyhedra can be reprented as a projection of a V-Cone intersected with the hyperplane $\left\{ \mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1 \right\}$. Given two sets $V \in \mathbb{R}^{d \times n}$ and $U \in \mathbb{R}^{d \times p}$, the V-Polyhedron is given by:

$$P_V = \{ \mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \text{cone}(U), \ \mathbf{y} \in \text{conv}(V) \}$$

It isn't hard to see that

$$\mathbf{x} \in P_V \Leftrightarrow \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \in \text{cone} \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ U & V \end{pmatrix}$$

For the value 1 to appear in the first coordinate, a convex combination of the vectors from $(\mathbf{1}, V)$ must be taken. After that, any non-negative combination of $(\mathbf{0}, U)$ added to this vector won't affect the 1 in the first coordinate.

It is more difficult to show that, given a V-Cone, that you can intersect it with the hyperplane $\left\{ \mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1 \right\}$ and get a V-Polytope out of it. So let

$$C_V = \text{cone}(U) \cap \left\{ \mathbf{y} \in \mathbb{R}^{d+1} \mid y_0 = 1 \right\}$$

We partition U into the sets:

$$P = i \mid U_0^i > 0$$
$$N = j \mid U_0^j < 0$$
$$Z = l \mid U_0^l = 0$$

And define two new sets:

$$U' = \left\{ U^l \mid l \in Z \right\} \cup \left\{ U_0^i U^j - U_0^j U^i \mid i \in P, j \in N \right\}$$
$$V = \left\{ U^i / U_0^i \mid i \in P \right\}$$

Then I claim that

$$C_V = \{ \mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \mathrm{cone}(U'), \mathbf{y} \in \mathrm{conv}(V) \} \tag{2.18}$$

Say $\mathbf{x} \in \mathrm{cone}(U')$, $\mathbf{x}$ can be written

$$\mathbf{x} = \sum_{l \in Z} t_l U^l + \sum_{\substack{i \in P \\ j \in N}} t_{ij} (U_0^i U^j - U_0^j U^i)$$

$$= \sum_{l \in Z} t_l U^l + \sum_{j \in N} \left(\sum_{i \in P} t_{ij} U_0^i \right) U^j + \sum_{i \in P} \left(\sum_{j \in N} -t_{ij} U_0^j \right) U^i$$

So $\mathbf{x} \in \mathrm{cone}(U)$. Furthermore,

$$\langle \mathbf{e}_0, \mathbf{x} \rangle = \sum_{l \in Z} t_l U_0^l + \sum_{\substack{i \in P \\ j \in N}} t_{ij} (U_0^i U_0^j - U_0^j U_0^i) = 0$$

So $x_0 = 0$. Similarly, for $\mathbf{y}$,

$$\mathbf{y} = \sum_{i \in P} \lambda_i U^i / U_0^i, \quad \sum_{i \in P} \lambda_i = 1$$

So $\mathbf{y} \in \mathrm{cone}(U)$, and then $\mathbf{x} + \mathbf{y} \in \mathrm{cone}(U)$. Furthermore,

$$\langle \mathbf{e}_0, \mathbf{y} \rangle = \sum_{i \in P} \lambda_i U_0^i / U_0^i = 1$$

So $y_0 = 1$ and $x_0 + y_0 = 1$. Then, by proposition 1, $\mathbf{x} + \mathbf{y} \in C_V$.

Next, suppose that $\mathbf{z} \in C_V$, then $\mathbf{z}$ can be written

$$\mathbf{z} = \sum_{l \in Z} t_l U^l + \sum_{i \in P} t_i U^i + \sum_{j \in N} t_j U^j$$

It will be convenient to use shorter notation for these sums. Define the following:

$$\sigma_Z = \sum_{l \in Z} t_l U^l, \quad \sigma_l = \sum_{l \in Z} t_l U_0^l = 0$$
$$\sigma_P = \sum_{i \in P} t_i U^i, \quad \sigma_i = \sum_{i \in P} t_i U_0^i$$
$$\sigma_N = \sum_{j \in N} t_j U^j, \quad \sigma_j = \sum_{j \in N} t_j U_0^j$$

14

Then it holds that

$$\langle \mathbf{e}_0, \mathbf{z} \rangle = \sigma_l + \sigma_i + \sigma_j = \sigma_i + \sigma_j = 1 \quad \Rightarrow \quad -\sigma_j/\sigma_i = 1 - 1/\sigma_i$$

$$\sigma_P = \sigma_P/\sigma_i + (1 - 1/\sigma_i)\sigma_P = \sigma_P/\sigma_i - (\sigma_j/\sigma_i)\sigma_P$$

Using the new notation, we can rewrite $\mathbf{z}$:

$$\mathbf{z} = \sigma_Z + \sigma_P + \sigma_N = \sigma_Z + \frac{\sigma_P}{\sigma_i} - \frac{\sigma_j}{\sigma_i}\sigma_P + \frac{\sigma_i}{\sigma_i}\sigma_N = \sigma_Z + \frac{\sigma_P}{\sigma_i} + \frac{\sigma_i\sigma_N - \sigma_j\sigma_P}{\sigma_i}$$

Using proposition 1, we need only show that

1. $\sigma_Z \in \mathrm{cone}(U')$

2. $(\sigma_i\sigma_N - \sigma_j\sigma_P)/\sigma_i \in \mathrm{cone}(U')$

3. $\sigma_P/\sigma_i \in \mathrm{conv}(V)$

Since each $U^l : l \in Z$ is in C_V, (1) holds. We also have:

$$\sigma_i\sigma_N - \sigma_j\sigma_P = \sum_{i\in P} t_i \sum_{j\in N} t_j U_0^i U^j - \sum_{j\in N} t_j \sum_{i\in P} t_i U_0^j U^i = \sum_{\substack{i\in P \\ j\in N}} t_i t_j (U_0^i U^j - U_0^j U^i)$$

So (2) holds. Finally,

$$\sigma_P/\sigma_i = \sum_{i\in P} t_i U^i /\sigma_i = \sum_{i\in P} (t_i U_0^i/\sigma_i)(U^i/U_0^i)$$

Since $\sum_{i\in P}(t_i U_0^i/\sigma_i) = \sigma_i/\sigma_i = 1$, it follows that $\sigma_P/\sigma_i \in \mathrm{conv}(V)$.

2.4   Picture of the Proof

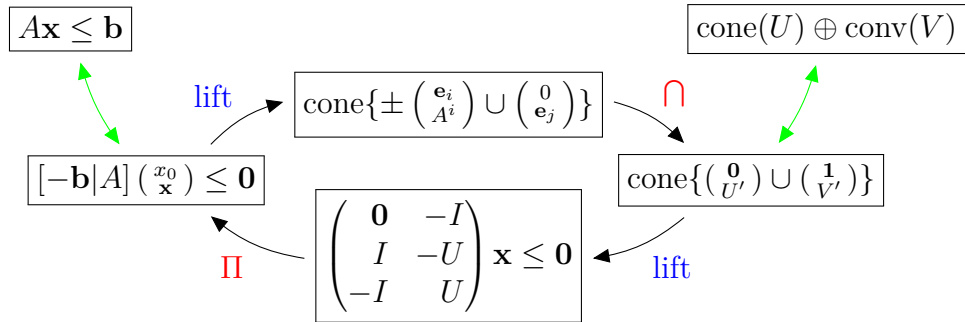Here we show a diagram that represent the proof of the Minkowski-Weyl Theorem.

Figure 2.1: Diagram of the proof $P_H \leftrightarrow P_V$

Figure 2.1 shows the flow from an H-Polyhedron to a V-Polyhedron and back. The colored arrows are the transformations back and forth from polyhedra to cones. The black arrows show the transformation between cones. V-Cones are lifted to H-Cones which need to be projected (Π), and H-Cones are lifted to V-Cones which need to be intersected ($\cap$) with some coordinate-hyperplanes then projected.

3. C++ Implementation

The above transformation have been implemented in C++. Program `main.cpp`
takes on argument specifying the type of input object. It reads the description of
the object from standard input, and writes the result of the implied transforma-
tion to standard output. If no arguments are supplied, then a `usage` message is
given. The `usage` message, which also contains the input format for the objects,
is:

```
usage: ./main input_type

The input object is read on stdin, and the result of the transform to sent to
stdout.  input_type determines the type of input and output:

    -vc  # transforms a vcone into an hcone
    -vp  # transforms a vpolyhedron into an hpolyhedron
    -hc  # transforms an hcone into a vcone
    -hp  # transforms an hpolyhedron into a vpolyhedron

input format is as follows:
  hcone := dimension   (vector ws)*
  vcone := dimension   (vector ws)*
  hpoly := dimension+1 (vector ws constraint ws)*
  vpoly := dimension   ('U' | 'V') ws vpoly_vecs*

  ws          := whitespace, as would be read by "cin >> ws;"
  dimension   := a positive integer.  For hpoly, add one to the dimension of
                  the space (this extra dimension is for the constraint)
  vector      := (dimension) doubles separated by whitespace
  constraint := a double (the value b_i in <A_i,x> <= b_i)
  'V' | 'U'   := the literal character 'U' or 'V'
  vpoly_vecs := (['U'] ws vector) | (['V'] ws vector)

VPOLY ONLY:

  vpoly contains two matrices:
    U - contains the rays of the vpolyhedron
    V - contains the points of the vpolyhedron

  On input, enter 'U' or 'V' to indicate which matrix should receive the vecto
  that follow.  You can switch back and forth as you like, but either 'U' or ''
  must be entered before starting to input vectors.

EXAMPLES:

$ ./main -vc <<< "2 1 0"

OUTPUT:
```

```
2
-0 -1
0 1
0 0
-1 0

$ ./main -hc <<< "2 1 0 0 1"

OUTPUT:

2
-1 0
0 0
0 0
0 -1

$ ./main -vp <<< "2 U 1 0 V 0 0 1 1"

OUTPUT:

3
0 0 -0
0 0 -0
0 1 1
0 0 1
-1 1 -0
-1 0 -0
0 0 -0
0 -1 -0
$ ./main -hp <<< "3 0 -1 0 0 1 1 -1 1 0"

OUTPUT:

2
U
1 0
0 0
0 0
0 0
V
0 0
1 1
```

   The files pertaining to the implementation will be discussed in the following
sections, but here is a table showing the include dependencies followed by a short
summary of the files.

file	includes
`linear_algebra.h`	
`fourier_motzkin.h`	`linear_algebra.h`
`polyhedra.h`	`fourier_motzkin.h`
`main.cpp`	`polyhedra.h`
`test_functions.h`	`linear_alebra.h`
`test.cpp`	`test_functions.h`, `polyhedra.h`

Here is a very brief summary of the files mentioned in the above table, more details are given in sequent sections.

- `linear_algebra.h`
  Types `Vector` and `Matrix`, and some basic functionality for them

- `fourier_motzkin.h`
  Fourier Motzkin elimination, Minkowski-Weyl Theoremfor cones

- `polyhedra.{cpp,h}`
  Transforms between polytopes and polyhedra, Minkowski-Weyl Theorem

- `test_functions.h`
  Types and functions for testing the algorithms, e.g. if an h-polyhedra contains some points

- `test.cpp`
  Test cases for functions from `test_functions.h` and algorithms for Minkowski-Weyl Theorem

3.1 Code

The relevant code will be displayed with commentary below. Some of the code relating to C++ specific technicalities and I/O is ommitted.

3.1.1 `linear_algebra.h`

The types `Vector` and `Vatrix` are used in the representation of polyhedra. The `std::valarray` template is used because it has built-in vector-space operations (sum and scaling). `std::vector`, is used, however other sequence containers could be used.

```
10  using Vector = std::valarray<double>;
11  using Vectors = std::vector<Vector>;
```

The `class Matrix` implements a subset of what a *C++ Container* should. It is the primary type for representing polyhedra, and directly represents Cones, as well as H-Polyhedra. The interface is designed to enforce the following invariant:

invariant: $(\forall v \in \texttt{vectors})\, \texttt{v.size() == d}$

The *factory* function `read_Matrix` is provided to read a `Matrix` from an `istream`. It is necessary because the value of `d` can't be known before reading some of the stream.

```
13  class Matrix {
14  // invariant: d >= 0
15  // invariant: (forall valid i) vectors[i].size() == d
16  public:
17    const size_t d; // size of all Vectors
18  private:
19    Vectors vectors;
20  public:
21    // needed for back_insert_iterator
22    using value_type = Vector;
23
24    Matrix(size_t d);
25    Matrix(std::initializer_list<Vector>&&);
26    bool check() const; // checks each Vector has size d
27
28    //defaults don't work because of const member
29    Matrix(const Matrix&);
30    Matrix(Matrix&&);
31    Matrix &operator=(const Matrix&);
32    Matrix &operator=(Matrix&&);
33    Matrix &operator=(std::initializer_list<Vector>&&);
34
35    static Matrix read_Matrix(std::istream&);
36
37    Vectors::iterator       begin();
38    Vectors::iterator       end();
39    Vectors::const_iterator begin() const;
40    Vectors::const_iterator end()   const;
41
42    bool    empty() const;
43    size_t  size()  const;
44    Vector& back();
45
46    Vector& add_Vector();
47    void push_back(const Vector &v);
48    void push_back(Vector &&v);
49  };
```

The struct VPoly gather two Matrixs needed to represent a V-Polyhedron. The Matrix U corresponds to the rays that generate the cone, and the Matrix V corresponds to the points, i.e.

$$\text{Vpoly vpoly} = \{p + q \mid p \in \text{cone}(\texttt{vpoly.U}), q \in \text{conv}(\texttt{vpoly.V})\}$$

```
51  struct VPoly {
52    const size_t d;
53    Matrix U; // rays
54    Matrix V; // points
55
56    VPoly(size_t d) : d{d}, U{d}, V{d} {}
57    VPoly(std::initializer_list<Vector>&&,
58          std::initializer_list<Vector>&&);
```

```
59    bool check() const;
60
61    static VPoly read_VPoly(std::istream&);
62 };
```

The class input_error is thrown to indicate an invalid input to the program, and provide some clue as to why it failed. Here are two command line examples:

```
$ ./main -vc <<< "0"
terminate called after throwing an instance of 'input_error'
  what():  bad d: 0
Aborted (core dumped)
$ ./main -vc <<< "2 1"
error reading matrix, vector 1
terminate called after throwing an instance of 'input_error'
  what():  failed to read vector: istream failed
Aborted (core dumped)
```

```
64 class input_error : public std::runtime_error {
65 public:
66    input_error(const char*s);
67    input_error(const std::string &s);
68 };
```

operator>> and operator<< implement the input format described in usage.txt.

```
70 std::istream& operator>>(std::istream&, Vector&);
71 std::istream& operator>>(std::istream&, Matrix&);
72 std::istream& operator>>(std::istream&, VPoly&);
```

```
74 std::ostream& operator<<(std::ostream& o, const Vector&);
75 std::ostream& operator<<(std::ostream& o, const Matrix&);
76 std::ostream& operator<<(std::ostream& o, const VPoly&);
```

usage() outputs the usage message shown above.

```
78 int usage();
```

3.1.2 linear_algebra.cpp

e_k creates the canonical basis Vector $\mathbf{e}_k \in \mathbb{R}^d$.

```
232 Vector e_k(size_t d, size_t k) {
233    Vector result(d);
234    result[k] = 1;
235    return result;
236 }
```

concatentate takes the Vectors l $\in \mathbb{R}^{\text{l.size()}}$ and r $\in \mathbb{R}^{\text{r.size()}}$ and returns the Vector (l,r) $\in \mathbb{R}^{\text{l.size() + r.size()}}$

```
239 Vector concatenate(const Vector &l, const Vector &r) {
240    Vector result(l.size() + r.size());
241    copy(begin(l), end(l), begin(result));
```

```
242    copy(begin(r), end(r), next(begin(result), l.size()));
243    return result;
244 }
```

get_column returns the k-th column of the Matrix M. Note that while a Matrix may logically represent either a collection of row or column Vectors, get_column is only used in the function transpose, where this distinction is unimportant.

```
249 Vector get_column(const Matrix &M, size_t k) {
250    if (!(0 <= k && k < M.d)) {
251       throw std::out_of_range("k < 0 || M.d <= k");
252    }
253    Vector result(M.size());
254    size_t result_row{0};
255    for (auto &&row : M) {
256       result[result_row++] = row[k];
257    }
258    return result;
259 }
```

transpose returns the transpose of Matrix M.

```
262 Matrix transpose(const Matrix &M) {
263    if (M.empty()) {
264       return M;
265    }
266    Matrix result{M.size()};
267    // for every column of M,
268    for (size_t k = 0; k < M.d; ++k) {
269       result.push_back(get_column(M,k));
270    }
271    return result;
272 }
```

An slice object can be used to conveniently obtain a subset of an valarray. slice_matrix returns the Matrix obtained by applying the slice s to each Vector of the Matrix.

```
275 Matrix slice_matrix(const Matrix &M, const std::slice &s) {
276    Matrix result{s.size()};
277    transform(M.begin(), M.end(), back_inserter(result),
278       [s](const Vector &v) { return v[s]; });
279    return result;
280 }
```

3.1.3 fourier_motzkin.h

Lift is a function pointer typedef that is used in a generic cone-transformation function.

```
43 typedef Matrix(*Lift)(const Matrix&);
```

3.1.4 `fourier_motzkin.cpp`

A `slice` object is determined by three fields: `start`, `size`, and `stride`, and implicitly represents all indices of the form:

$$\sum_{0 \le k < \texttt{size}} \texttt{start} + k \cdot \texttt{stride}$$

Therefore:

$$i \in \texttt{slice} \Leftrightarrow i - \texttt{start} \equiv 0 \mod (\texttt{stride}), \quad \texttt{start} \le i \le \texttt{start} + \texttt{stride} \cdot \texttt{size}$$

```
11  bool index_in_slice(size_t index, const slice &s) {
12    return ((index - s.start()) % s.stride() == 0) &&
13            s.start() <= index &&
14            index <= s.start() + s.stride()*(s.size()-1);
15  }
```

`fourier_motzkin` takes a `Matrix M` and a coordinate `k` and creates the set which either corresponds to a projection of an H-Cone (without actually doing the projection), or the intersection of a V-Cone with a coordinate-hyperplane.

```
20  Matrix fourier_motzkin(Matrix M, size_t k) {
21    Matrix result{M.d};
22    // Partition into Z,P,N
23    const auto z_end = partition(M.begin(), M.end(),
24        [k](const Vector &v) { return v[k] == 0; });
25    const auto p_end = partition(z_end, M.end(),
26        [k](const Vector &v) { return v[k] > 0; });
27    // Move Z to result
28    move(M.begin(), z_end, back_inserter(result));
29    // convolute vectors from P,N
30    for (auto p_it = z_end; p_it != p_end; ++p_it) {
31      for (auto z_it = p_end; z_it != M.end(); ++z_it) {
32        result.push_back(
33          (*p_it)[k]*(*z_it) - (*z_it)[k]*(*p_it));
34      }
35    }
36    return result;
37  }
```

The lines:

```
23    const auto z_end = partition(M.begin(), M.end(),
24        [k](const Vector &v) { return v[k] == 0; });
25    const auto p_end = partition(z_end, M.end(),
26        [k](const Vector &v) { return v[k] > 0; });
```

Partition `M` into logical sets Z, P, N that satisfy the following:

set	range	property
Z	[`M.begin()`, `z_end`)	`it` $\in Z \Leftrightarrow$ `(*it)[k]` $= 0$
P	[`z_end`, `p_end`)	`it` $\in P \Leftrightarrow$ `(*it)[k]` > 0
N	[`p_end`, `M.end()`)	`it` $\in N \Leftrightarrow$ `(*it)[k]` < 0

The line:

```
28    move(M.begin(), z_end, back_inserter(result));
```

Moves Z into the result. The lines:

```
30    for (auto p_it = z_end; p_it != p_end; ++p_it) {
31      for (auto z_it = p_end; z_it != M.end(); ++z_it) {
32        result.push_back(
33          (*p_it)[k]*(*z_it) - (*z_it)[k]*(*p_it));
34      }
35    }
```

Convolutes the vectors in the way described in Propositions 3 and 4 (concerning projecting an H-Cone and intersecting a V-Cone with a coordinate-hyperplane), and push them into the result `Matrix`. In particular, it creates the sets which correspond to

$$B_i^k B_j - B_j^k B_i \mid i \in P, \, j \in N$$

`sliced_fourier_motzkin` applies `fourier_motzkin` to `Matrix M` for each $k \notin$ `slice s`, then slices the resulting `Matrix` using `slice_matrix` and `s`. This is the realization of the algorithms indicated by the proofs of either direction of the Minkowski-Weyl Theorem for cones.

```
40  Matrix sliced_fourier_motzkin(Matrix M, const slice &s) {
41    for (size_t k = 0; k < M.d; ++k) {
42      if (!index_in_slice(k,s)) {
43        M = fourier_motzkin(M, k);
44      }
45    }
46    return slice_matrix(M, s);
47  }
```

When transforming an H-Cone to a V-Cone, it first must be written as a V-Cone of a new matrix, then it is intersected with coordinate-hyperplanes and projected. Similarly, when a V-Cone is transformed into an H-Cone, it must be written as and H-Cone of a new matrix then projected with coordinate-projections. The transformations are described in (2.3) and (2.14), and summarized here:

$$A \to \begin{pmatrix} \mathbf{0} & -I \\ I & -U \\ -I & U \end{pmatrix} \quad U \to \begin{pmatrix} \mathbf{0} & I & -I \\ I & A & -A \end{pmatrix}$$

Note that the tranformation of U can be written:

$$U \to \begin{pmatrix} \mathbf{0} & I \\ I & A \\ -I & -A \end{pmatrix}^T$$

Remembering that a `Matrix` is either a collection of row *or* column `Vector`s, it is not surprising that these two transformations can be written as one function of a `Matrix` and some coefficients. In `generalizedlift`, the coefficients are given as an `array<double, 5> C`, so the overall transformation can be illustrated as:

$$\texttt{Matrix M} \to \begin{pmatrix} \mathbf{0} & \texttt{C[0]}I \\ \texttt{C[1]}I & \texttt{C[2]M} \\ \texttt{C[3]}I & \texttt{C[4]M} \end{pmatrix}$$

Where `Matrix M` is a collection of row `Vectors`, or

$$\texttt{Matrix M} \rightarrow \begin{pmatrix} \mathbf{0} & \texttt{C[1]}I & \texttt{C[3]}I \\ \texttt{C[0]}I & \texttt{C[2]M} & \texttt{C[4]M} \end{pmatrix}$$

Where `Matrix M` is a collection of column `Vectors`.

```
64  Matrix generalized_lift(const Matrix &cone,
65                          const array<double,5> &C) {
66    const size_t d = cone.d;
67    const size_t n = cone.size();
68    Matrix result{d+n};
69    Matrix cone_t = transpose(cone);
70    // |0   C[0]*I|  |0     |
71    //               |C[0]*I|
72    for (size_t i = 0; i < n; ++i) {
73      result.add_Vector()[d+i] = C[0];
74    }
75    size_t k = 0;
76    // |C[1]*I C[2]*U|  |C[1]*I|
77    //                  |C[2]*A|
78    for (auto &&row_t : cone_t) {
79      result.push_back(
80        concatenate(C[1]*e_k(d,k++), C[2]*row_t));
81    }
82    k = 0;
83    // |C[3]*I C[4]*U|  |C[3]*I|
84    //                  |C[4]*A|
85    for (auto &&row_t : cone_t) {
86      result.push_back(
87        concatenate(C[3]*e_k(d,k++), C[4]*row_t));
88    }
89    return result;
90  }
```

`lift_vcone` and `lift_hcone` implement the appropriate transformation using `generalized_lift` and providing the appropriate coefficients in `array<double, 5> C`.

```
98   Matrix lift_vcone(const Matrix &vcone) {
99     return generalized_lift(vcone, {-1,1,-1,-1,1});
100  }
```

```
107  Matrix lift_hcone(const Matrix &hcone) {
108    return generalized_lift(hcone, {1,1,1,-1,-1});
109  }
```

`cone_transform` consolidates the logic of the V-Cone → H-Cone and H-Cone → V-Cone transformations by accepting a `Matrix cone` and a `Lift`.

```
112  Matrix cone_transform(const Matrix &cone, Lift lift) {
113    if (cone.empty()) {
114      throw logic_error{"empty cone for transform"};
115    }
116    // the idea of the entire mwt is this one line
```

```
117    return sliced_fourier_motzkin(
118      lift(cone), slice(0, cone.d, 1));
119  }
```

vcone_to_hcone and hcone_to_vcone specialize cone_transform by providing the appropriate Lift.

```
121  Matrix vcone_to_hcone(Matrix vcone) {
122    return cone_transform(vcone,lift_vcone);
123  }
```

```
125  Matrix hcone_to_vcone(Matrix hcone) {
126    return cone_transform(hcone,lift_hcone);
127  }
```

3.1.5 polyhedra.cpp

hpoly_to_hcone and hcone_to_hpoly implement the Matrix transforms:

$$\texttt{hpoly\_to\_hcone} : (A|b) \to (-b|A), \quad \texttt{hcone\_to\_hpoly} : (-b|A) \to (A|b)$$

These very simple transforms are done with the cshift function, which "circularly shifts" the elements of a Vector (provided as part of the interface to valarray).

```
13  Matrix hpoly_to_hcone(Matrix hpoly) {
14    transform(hpoly.begin(), hpoly.end(), hpoly.begin(),
15        [](Vector v) {
16          v[v.size()-1] *= -1;
17          return v.cshift(-1);
18        });
19    return hpoly;
20  }
```

```
24  Matrix hcone_to_hpoly(Matrix hcone) {
25    transform(hcone.begin(), hcone.end(), hcone.begin(),
26        [](Vector v) {
27          v[0] *= -1;
28          return v.cshift(1);
29        });
30    return hcone;
31  }
```

vpoly_to_vcone implements the VPoly transform:

$$\texttt{vpoly} \to \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ \texttt{vpoly.U} & \texttt{vpoly.V} \end{pmatrix}$$

```
36  Matrix vpoly_to_vcone(VPoly vpoly) {
37    //requires increase in dimension
38    Matrix result{vpoly.d+1};
39    for (auto &&u : vpoly.U) {
40      result.push_back(concatenate({0},u));
41    }
```

```
42    for (auto &&v : vpoly.V) {
43      result.push_back(concatenate({1},v));
44    }
45    return result;
46  }
```

normalized_P calculates the V in (2.18). Let Π is the identity matrix with the 0-th row deleted, and $P = \{\mathbf{u} \in U : u_0 > 0\}$. then this is the result of:

$$\Pi(\text{cone}(P) \cap \{x_0 = 1\})$$

```
50  Matrix normalized_P(const Matrix &U) {
51    if (U.d <= 1) {
52      throw std::logic_error{"can't normalize U!"};
53    }
54    Matrix result{U.d-1};
55    std::slice s{1,result.d,1};
56    for (auto &&v : U) {
57      // select the vectors with positive 0-th coordinate
58      if (v[0] <= 0) { continue; }
59      // normalize the selected vectors,
60      result.push_back(v[0] == 1 ? v[s] : (v / v[0])[s]);
61    }
62    return result;
63  }
```

vcone_to_vpoly implements the full tranformation in (2.18).

```
67  VPoly vcone_to_vpoly(Matrix vcone) {
68    VPoly result{vcone.d-1};
69    result.U = sliced_fourier_motzkin(
70                vcone, slice(1,vcone.d-1,1));
71    result.V = normalized_P(vcone);
72    return result;
73  }
```

hpoly_to_vpoly and vpoly_to_hpoly implement the complete transformations promised by the file.

```
77  VPoly hpoly_to_vpoly(Matrix hpoly) {
78    return vcone_to_vpoly(
79            hcone_to_vcone(
80              hpoly_to_hcone(move(hpoly))));
81  }
```

```
83  Matrix vpoly_to_hpoly(VPoly vpoly) {
84    return hcone_to_hpoly(
85            vcone_to_hcone(
86              vpoly_to_vcone(move(vpoly))));
87  }
```

3.1.6 test_functions.h

```
 7  struct hcone_test_case {
 8    std::string name;
 9    Matrix hcone; // vectors for H or V cone
10    Matrix key; // minimal genearating set of dual vectors
11
12    bool run_test() const;
13  };

15  struct vcone_test_case {
16    std::string name;
17    Matrix vcone; // vectors for H or V cone
18    Matrix key; // minimal genearating set of dual vectors
19
20    bool run_test() const;
21  };

23  struct hpoly_test_case {
24    std::string name;
25    Matrix hpoly; // vectors for H-Polyhedron
26    VPoly key;    // minimal genearating set of dual vectors
27
28    bool run_test() const;
29  };

31  struct vpoly_test_case {
32    std::string name;
33    VPoly  vpoly; // vectors for V-Polyhedron
34    Matrix key;   // minimal genearating set of dual vectors
35
36    bool run_test() const;
37  };
```

3.1.7 `test_functions.cpp`

```
28  double operator*(const Vector &l, const Vector &r) {
29    if (l.size() > r.size()) {
30      throw runtime_error{"inner product: l > r"};
31    }
32    return inner_product(begin(l), end(l), begin(r), 0.);
33  }

35  double norm(const Vector &v) {
36    return sqrt(v*v);
37  }

39  bool approximately_zero(double d) {
40    const double error = .000001;
41    bool result = d < error;
42    if (d != 0 && result) {
43      char d_buf[64]{};
44      std::sprintf(d_buf, "%.9e", d);
45      log("approximately_zero "s + d_buf, 1);
46    }
```

```
47      return result;
48    }

50    bool approximately_lt_zero(double d) {
51      return d < 0 || approximately_zero(d);
52    }

55    bool approximately_zero(const Vector &v) {
56      return approximately_zero(norm(v));
57    }

59    bool is_equivalent(const Vector &l, const Vector &r) {
60      if (l.size() != r.size()) return false;
61      if (norm(l) == 0 || norm(r) == 0) {
62        return norm(l) == 0 && norm(r) == 0;
63      }
64      return approximately_zero(l / norm(l) - r / norm(r));
65    }

67    bool is_equal(const Vector &l, const Vector &r) {
68      if (l.size() != r.size()) return false;
69      return approximately_zero(l - r);
70    }

72    bool has_equivalent_member(const Matrix &M,
73                               const Vector &v) {
74      if (!any_of(M.begin(), M.end(),
75        [&](const Vector &u) { return is_equivalent(u,v); })) {
76        ostringstream oss;
77        oss << dashes << " no equivalent member found for:\n"
78            << v << endl;
79        log(oss.str(),1);
80        return false;
81      }
82      return true;
83    }

85    bool has_equal_member(const Matrix &M, const Vector &v) {
86      if (!any_of(M.begin(), M.end(),
87        [&](const Vector &u) { return is_equal(u,v); })) {
88        ostringstream oss;
89        oss << dashes << " no equal member found for:\n"
90            << v << endl;
91        log(oss.str(),1);
92        return false;
93      }
94      return true;
95    }

98    bool subset_mod_eq(const Matrix &generators,
99                       const Matrix &vcone) {
100     return all_of(generators.begin(), generators.end(),
101       [&](const Vector &g) {
```

```
102        return has_equivalent_member(vcone, g); });
103 }


106 bool subset(const Matrix &generators,
107             const Matrix &vcone) {
108   return all_of(generators.begin(), generators.end(),
109     [&](const Vector &g) {
110       return has_equal_member(vcone, g); });
111 }


115 bool ray_satisfied(const Vector &constraint,
116                    const Vector &ray) {
117   if (constraint.size() != ray.size() &&
118       constraint.size()-1 != ray.size()) {
119     throw runtime_error{"bad ray vs constraint"};
120   }
121   double ip = ray * constraint;
122   if (!(approximately_lt_zero(ip))) {
123     ostringstream oss;
124     oss << dashes << " ray not satisfied!\n"
125         << "ray: " << ray
126         << "\nconstraint: " << constraint
127         << "\n ray * constraint = " << ip << endl;
128     log(oss.str(), 1);
129     return false;
130   }
131   return true;
132 }


134 bool ray_satisfied(const Matrix &constraints,
135                    const Vector &ray) {
136   return all_of(constraints.begin(), constraints.end(),
137     [&](const Vector &cv) {
138       return ray_satisfied(cv, ray); });
139 }


141 bool rays_satisfied(const Matrix &constraints,
142                     const Matrix &rays) {
143   return all_of(rays.begin(), rays.end(),
144     [&](const Vector &ray) {
145       return ray_satisfied(constraints, ray); });
146 }


149 bool vec_satisfied(const Vector &constraint,
150                    const Vector &vec) {
151   size_t cback_i = constraint.size()-1;
152   if (cback_i != vec.size()) {
153     throw runtime_error{"bad vec vs constraint"};
154   }
155   double ip = vec * constraint;
156   if (!(approximately_lt_zero(ip - constraint[cback_i]))) {
157     ostringstream oss;
```

```
158      oss << dashes << " vec not satisfied!\n"
159          << "vec: " << vec
160          << "\nconstraint: " << constraint
161          << "\n vec * constraint = " << ip << endl;
162      log(oss.str(), 1);
163      return false;
164    }
165    return true;
166 }

168 bool vec_satisfied(const Matrix &constraints,
169                    const Vector &vec) {
170    return all_of(constraints.begin(), constraints.end(),
171      [&](const Vector &cv) {
172        return vec_satisfied(cv, vec); });
173 }

175 bool vecs_satisfied(const Matrix &constraints,
176                     const Matrix &vecs) {
177    return all_of(vecs.begin(), vecs.end(),
178      [&](const Vector &vec) {
179        return vec_satisfied(constraints, vec); });
180 }

184 bool equivalent_cone_rep(const Matrix &cone,
185                          const Matrix &key,
186                          const Matrix &alt_rep) {
187    return rays_satisfied (cone, alt_rep) &&
188           subset_mod_eq  (key, alt_rep);
189 }

191 bool equivalent_hpoly_rep(const Matrix &hpoly,
192                           const VPoly  &key,
193                           const VPoly  &vpoly) {
194    return rays_satisfied (hpoly, vpoly.U) &&
195           vecs_satisfied (hpoly, vpoly.V) &&
196           subset_mod_eq  (key.U, vpoly.U) &&
197           subset         (key.V, vpoly.V);
198 }

200 bool equivalent_vpoly_rep(const VPoly  &vpoly,
201                           const Matrix &key,
202                           const Matrix &hpoly) {
203    return rays_satisfied (hpoly, vpoly.U) &&
204           vecs_satisfied (hpoly, vpoly.V) &&
205           subset_mod_eq  (key, hpoly);
206 }

208 bool hcone_test_case::run_test() const {
209    auto alt_rep = hcone_to_vcone(hcone);
210    if (!equivalent_cone_rep(hcone, key, alt_rep)) {
211      log(dashes + "hcone TEST FAILED: " + name, 0);
212      ostringstream oss;
```

```cpp
213        oss << "hcone: " << hcone << "key: " << key
214            << "alt_rep: " << alt_rep;
215        log (oss.str(), 2);
216        return false;
217      }
218      return true;
219    }

221    bool vcone_test_case::run_test() const {
222      auto alt_rep = vcone_to_hcone(vcone);
223      if (!equivalent_cone_rep(vcone, key, alt_rep)) {
224        log(dashes + "vcone TEST FAILED: " + name, 0);
225        ostringstream oss;
226        oss << "vcone: " << vcone << "key: " << key
227            << "alt_rep: " << alt_rep;
228        log (oss.str(), 2);
229        return false;
230      }
231      return true;
232    }

234    bool hpoly_test_case::run_test() const {
235      auto vpoly = hpoly_to_vpoly(hpoly);
236      if (!equivalent_hpoly_rep(hpoly, key, vpoly)) {
237        log(dashes + "hpoly TEST FAILED: " + name, 0);
238        ostringstream oss;
239        oss << "hpoly: " << hpoly << "key: " << key
240            << "vpoly: " << vpoly;
241        log (oss.str(), 2);
242        return false;
243      }
244      return true;
245    }

247    bool vpoly_test_case::run_test() const {
248      auto hpoly = vpoly_to_hpoly(vpoly);
249      if (!equivalent_vpoly_rep(vpoly, key, hpoly)) {
250        log(dashes + "vpoly TEST FAILED: " + name, 0);
251        ostringstream oss;
252        oss << "vpoly: " << vpoly << "key: " << key
253            << "hpoly: " << hpoly;
254        log (oss.str(), 2);
255        return false;
256      }
257      return true;
258    }
```

Bibliography

[1] ZIEGLER, Gunter. *Lectures on Polytopes*. Springer-Verlag, New York, 1995. ISBN 0-387-94329-3.