Improving the Semidefinite Stand-and-Hit Algorithm for Detecting Necessary Linear Matrix Inequalities

Graham Mueller[1]
Oklahoma State University
Stillwater, Ok 74077

Abstract

Large semidefinite programming problems can be reduced in size by indentifying necessary linear matrix inequalities, or LMIs. One such method of finding the necessary LMIs is known as the semidefinite stand-and-hit algorithm(SSH). The SSH algorithm is a Monte Carlo algorithm, which uses randomly selected points to identify the necessary LMIs. The SSH algorithm is used to find the necessary constraints in a system of linear matrix inequalities. The size of a semidefinite programming problem can be significantly reduced, since unnecessary, or redundant constraints only add addtional computational time to a problem. In this paper different methods of improving the SSH algorithm, specifically using faster and more efficient algorithms to detect eigenvalues are examined as well as the implementation of weighted analytic center and repelling limits.

[1]advised by Dr. Shafiu Jibrin, Department of Mathematics and Statistics, Northern Arizona University

§1 Introduction

1.1 Basics from Linear Algebra

Definition 0.1 *Let A be an $n \times n$ symmetric matrix. A is* positive definite, *denoted by $A \succ 0$, if all its eigenvalues are positive.*

A is positive definite if and only if $\det(A_{ij}) > 0$ for each principal submatrix A_{ij} of A.

Definition 0.2 *A is* positive semidefinite, *denoted by $A \succeq 0$, if all of its eigenvalues are non-negative.*

A is positive semi-definite if and only if $\det(\ A_{ij}\ ) \geq 0$ for each principal submatrix A_{ij} of A.

Definition 0.3 - Square Root Decompostion - *a symmetric, positive definite matrix can be decomposed into the following form:*

$A = A^{\frac{1}{2}}A^{\frac{1}{2}}$ where A is determined by the Schur decomposition ($A^{\frac{1}{2}} = QD^{\frac{1}{2}}Q^T$ where D is a diagonal matrix with the same eigenvalues as A.) Also $A \succ 0$ if and only if $A^{\frac{1}{2}} \succ 0$.

Definition 0.4 - Cholesky Factorization - *a symmetric, positive definite matrix can be decomposed into the following form:*
$A = LL^T$ where L is a lower triangular matrix

Definition 0.5 - Symmetric Tridiagonal Form - *A matrix is symmetric tridiagonal if the matrix has non-zero entries only on the diagonal, the super-diagonal, and the sub-diagonal and is symmetric.*

For instance, the following matrix is symmetric tridiagonal.

$$\begin{pmatrix} a_1 & b_1 & 0 & 0 \\ b_1 & a_2 & \ddots & 0 \\ 0 & \ddots & \ddots & b_{n-1} \\ 0 & 0 & b_{n-1} & a_n \end{pmatrix}$$

1.2 Semidefinite Programming

Definition 0.6 - **Semidefinite Programming Problem** *is a problem of the type*

minimize $c^T x$

$$\textit{subject to } A^{(j)}(x) := A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0, j = 1, 2, \ldots, q.$$

where $A_i^{(j)}, i = 0, 1, \ldots, n$ are $m_j \times m_j$ symmetric matrices. The number of constraints is given by q and $c, x \in \mathbb{R}^n$.

Example of a system of linear matrix inequalities:

$$A^{(1)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \succeq 0$$

$$A^{(2)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix} + x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0$$

$$A^{(3)}(x) = \begin{bmatrix} 5 & 0 \\ 0 & 2 \end{bmatrix} + x_1 \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \succeq 0$$

$$A^{(4)}(x) = \begin{bmatrix} 5 & -1 \\ -1 & 2 \end{bmatrix} + x_1 \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \succeq 0$$

$$A^{(5)}(x) = \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix} + x_1 \begin{bmatrix} -0.25 & 0 \\ 0 & -0.25 \end{bmatrix} + x_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0$$

The SDP problem, i.e., the problem of minimizing an objective function subject to a linear matrix inequality, is an important problem for many reasons. Semidefinite Programming has many applications in systems and control theory and also provides a valuable technique in bounding solutions from many NP-Combinatorial Optimization problems. In addition to its wide-range of applications, Semidefinite Programming provides a generalization of linear programming.

1.3 Redundancy in Semidefinite Programming

Definition 0.7 -*The* Feasible Region, $\mathcal{R}$, *is defined as*

$$\mathcal{R} = \{x \mid A^{(j)}(x) \succeq 0, 1 \leq j \leq q\}$$

Definition 0.8 - *An LMI constraint is* redundant *if its removal does not change the feasible region. A constraint is called* necessary *if its removal from the system changes the feasible region.*

Identifying the redundant linear matrix inqualities in a particular SDP problem can significantly reduce the time needed to solve the SDP problem and is therefore a primary goal in the study of semidefinite programming.

There are three different Monte-Carlo algorithms for detecting necessary linear matrix inequalities. [1] They are:

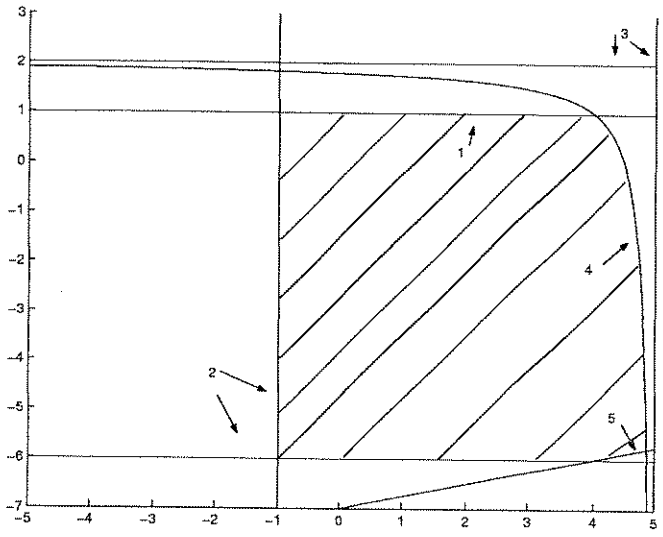

Figure 1: The feasible region corresponding to example of LMIs

Semidefinite Stand-and-Hit Algorithm - The SSH algorithm is based on the following ideas. Fix a standing point, x_0, so that $x_0 \in \mathcal{R}$. A random direction vector, s, is chosen. Then let $x_0 + \sigma s$ and $x_0 - \sigma s$ denote two line segments whose intersection with the boundary of $\mathcal{R}$ identify a necessary LMI constraint. This process is continued until a stopping point is reached.

Semidefinite Hypersphere Direction (SHD) algorithm - The SHD algorithm is very similiar to the SSH algorithm, with one major difference. In SHD the standing point is not fixed, rather a new standing point is chosen uniformly along the current line segment each iteration.

Semidefinite Coordinate Direction (SCD) algorithm - SCD is similiar to SHD in that a new standing point is chosen each iteration. However, in the SCD algorithm the direction vector, s, is chosen randomly from one of the coordinate axis.

§2 The SSH Algorithm

As mentioned in the introduction, the SSH algorithm fixes a standing point x_0 somewhere in the interior of $\mathcal{R}$. A random direction vector,s is then generated. Given x_0 and s let us define the symmetric matrix at x_0 as follows:

$$B_j(s, x_0) = -A^{(j)}(x_0)^{-\frac{1}{2}} \left(\sum_{i=1}^{n} s_i A_i^j \right) A^{(j)}(x_0)^{-\frac{1}{2}} \text{ where } (1 \leq j \leq q).$$

Theorem 0.9 - *Since x_0 is in the interior of $\mathcal{R}$, $A^j(x_0) \succ 0$ for each j and $A^j(x_0)$ has a square root decompostion and can then the distances from x_0 to the boundary of $\mathcal{R}$ along the line $\{x_0 + \sigma s \mid \sigma \in \mathbb{R}\}$ are given by*

$\sigma_+ = min\{\frac{1}{\lambda_{max}(B_j(s))} \mid j = 1, 2, \ldots, q\}$ *and* $\sigma_- = max\{-\frac{1}{\lambda_{min}(B_j(s))} \mid j = 1, 2, \ldots, q\}$

Proof:

If $x \in \partial\mathcal{R}$. where $\partial\mathcal{R}$ denotes the boundary of $\mathcal{R}$, then at least one eigenvalue must be zero

$\Rightarrow det[A^{(j)}(x_0 + \sigma s)] = det[A^{(j)}(x_0) + \sigma \sum_{i=1}^{n} s_i A_i] = 0.$ multiplying we have

$\Rightarrow det[[A^{(j)}(x_0)^{-\frac{1}{2}}][A^{(j)}(x_0) + \sigma \sum_{i=1}^{n} s_i A_i][A^{(j)}(x_0)^{-\frac{1}{2}}]]=0$

$\Rightarrow det[I + \sigma A^{(j)}(x_0)^{-\frac{1}{2}} \sum_{i=1}^{n} s_i A_i A^{(j)}(x_0)^{-\frac{1}{2}}] = 0$

$\Rightarrow det[\frac{1}{\sigma} I - B_j(s, x_0)] = 0$

$\frac{1}{\sigma}$ is an eigenvalue of $B_j(s, x_0)$. The problems of finding a necessary constraint is now reduced to finding the extreme eigenvalues of $B_j(s, x_0)$

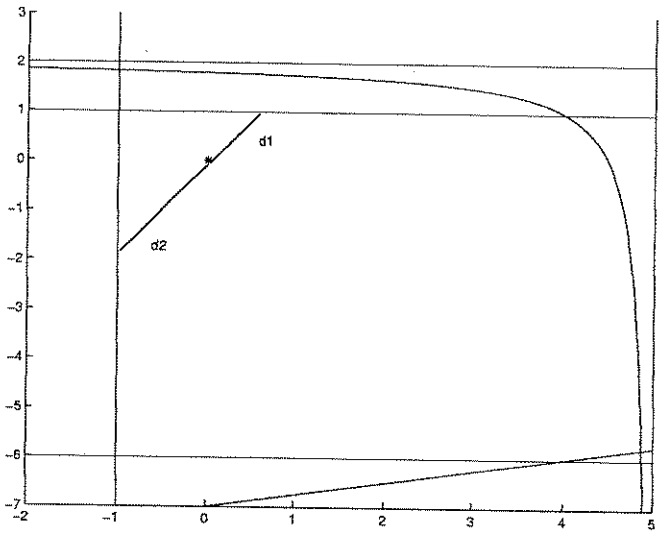

Figure 2: The distance to the boundary from a standing point

SSH Algorithm

Let $\mathcal{J}$ denote the set of necessary constraints.

1. Initialize $\mathcal{J} = \emptyset$. Choose an interior point, x_0 of $\mathcal{R}$. Calculate $A^{(j)}(x_0)^{-\frac{1}{2}}$ for $1 \leq j \leq q$.
2. Repeat : Choose s_0 randomly from N(0,1) and set $s = \frac{s_0}{\|s_0\|}$.
3. Repeat : Calculate $B_j(s, x_0)$ and find both $\sigma_+^{(j)}$ and $\sigma_-^{(j)}$. Calculate $\sigma_+ = \min\{\sigma_+^{(j)} | 1 \leq j \leq q\}$ and $\sigma_- = \min \sigma_-^{(j)} | 1 \leq j \leq q\}$.

4. For $1 \leq k \leq q$ check if $\sigma_+^{(k)} = \sigma_+$ or if $\sigma_-^{(k)} = \sigma_-$ if so set $\mathcal{J} = \mathcal{J} \cup k$

5. Continue until a stopping criterion is reached.

Note that since the standing point, x_0 is fixed, $A^{(j)}(x_0)^{-\frac{1}{2}}$ is only calculated once during the course of the algorithm. The main source of computational complexity for the SSH algorithm is finding λ_{max} and λ_{min} of the matrix $B_j(s, x_0)$.

2.2 Computation of Eigenvalues

Several different methods for computing λ_{max} and λ_{min} of the matrix $B_j(s, x_0)$ were examined.

QR Algorithm - The QR Algorithm is one of the fastest and most efficient numerical algorithms used to find the eigenvalues of a matrix. However, the QR Algorithm finds all eigenvalues of a particular matrix and for the SSH algorithm we are only interested in finding λ_{max} and λ_{min}. One method of finding these two eigenvalues is by simply finding all the eigenvalues using QR and then finding the maximum and minimum eigenvalues from the results.
The QR Algorithm begins by reducing the original matrix, A, into a tridiagonal matrix, T, by a series of orthogonal transformations. The QR Algorithm then applies a series of orthogonal transformations to T, so that T converges to a diagonal matrix, D, so that the diagonal elements of D are equal to the eigenvalues of T. At each iteration the QR Decomposition ($A = QR$ where $Q^TQ = I$ and R is upper triangular) of a matrix is computed.

The QR Algorithm

$$i = 0 \ , \ A_0 = A$$
repeat until stopping criterion is reached.
$$\text{Factor } A_i = Q_i R_i$$
$$A_{i+1} = R_i Q_i$$
$$i = i + 1$$

Power Method with shifting and accelerated convergence - Assume the eigenvalues of a matrix A are $\lambda_1, \lambda_2, \ldots, \lambda_n$ where $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Let $v = c_1v_1 + c_2v_2 + \ldots + c_nv_n$ where each v_i denotes a corresponding eigenvector. Then.

$$Av = c_1v_1 + c_2v_2 + \ldots + c_nv_n = \lambda_1(c_1v_1 + c_2\tfrac{\lambda_2}{\lambda_1}v_2 + \ldots + c_n\tfrac{\lambda_n}{\lambda_1}v_n)$$

This iteration is repeated until we have:

$$A^pv = \lambda_1^p(c_1v_1 + c_2(\tfrac{\lambda_2}{\lambda_1})^pv_2 + \ldots + c_n(\tfrac{\lambda_n}{\lambda_1})^pv_n)$$

for large p $A^pv \cong \lambda_1^p(c_1v_1)$ where c_1v_1 is the eigenvector of λ_1. Thus

$$\lambda_1 = \lim \frac{(A^{p+1}v)_r}{(A^pv)_r}$$

where $r = 1, 2, \ldots, n$ and denotes the index of the component of the vector.
Power Method Algorithm- Given a matrix A and a vector y_k, we form two other vectors y_{k+1} and z_{k+1}.

$$z_{k+1} = Ay_k$$
$$y_{k+1} = z_{k+1}$$
$$\alpha_{k+1} = max_r|(z_{k+1})_r|$$

Continuing with this iteration one arrives at the eigenvector of the largest eigenvalue. Note that y_0 is usually chosen so that all its components are equal to one. So we have the largest eigenvalue,λ_{max}, how can we now find the smallest eigenvalue? This can be done easily with shifting. Replace the original matrix, A, by $A - \lambda_{max}I$ and repeat the power method on this new matrix. This finds the smallest eigenvalue.

The power method seems to be slower than the QR algorithm for finding the maximum and minimum eigenvalues. However, we can improve upon the power method by accelerating the convergence of the sequence formed by the algorithm. For accelerating convergence we use what is known as Aitken's $\triangle^2$ method.

Suppose we have some sequence, p_n, that is linearly convergent with the limit p. A new sequence, $\overline{p}_n$ can be formed that converges faster to p than p_n. This new sequence is defined in the following manner:

$$\overline{p}_n = p_n - \frac{(p_{n+1}-p_n)^2}{p_{n+2}-2p_{n+1}+p_n}$$

Implementing Aitkens $\triangle^2$ method allows us to find the maximum and minimum eigenvalues much more quickly than with the ordinary power method.

Bisection method - The bisection method for finding eigenvalues is advantagous to use when only selected eigenvalues are needed. The bisection method can only be applied to symmetric tridiagonal matrices, however any symmetric matrix may be transformed into this form by the Householder transformation. If it is necessary to find the *k-th* largest eigenvalue of a matrix T, in this case we are interested in the smallest (1st largest) and the largest (nth largest) eigenvalues of T we can use the method of bisection.

Suppose we wish to find $\lambda_k(T)$ then if we can find a y and z such that $\lambda_k(T)\epsilon[y, z]$ then the method of bisection can be used to approximate λ_k.
Let T_r denote the leading *r-by-r* principal submatrix of T. Define a polynomial

$$p_r(x) = \det(T_r - xI) \; r = 1, 2, \ldots, n \text{ with } p_0(x) = 1$$

Theorem 0.10 - *(Sturm Sequence Property) If a tridiagonal matrix has no zero subdiagonal entries then the eigenvalues of T_{r-1} seperate the eigenvalues of T_r:*

$$\lambda_r(T_r) < \lambda_{r-1}(T_{r-1}) < \lambda_{r-1}(T_r) < \ldots < \lambda_2(T_r) < \lambda_1(T_{r-1}) < \lambda_1(T_r)$$

Let us denote the number of sign changes in the sequence $\{p_0(\lambda), p_1(\lambda), \ldots, p_n(\lambda)\}$ by $a(\lambda)$, then $a(\lambda)$ is the number of eigenvalues that are less than λ.

It is possible to find y and z such that $\lambda_k(T)\epsilon[y, z]$ where $y = \min\ a_i - |b_i| - |b_{i-1}|$ and $z = \max\ a_i + |b_i| + |b_{i-1}|$. Also define $b_0 = b_n = 0$. With these initial values the following iteration produces a sequence that converges to $\lambda_k(T)$.

while $|z-y| > TOL(|y|+|z|)$
    $x = (y+z)/2$
    if $a(x) \geq n-k$
        $z = x$
    else
        y=z
    end
end

2.3 Experimental Data

The following table displays the size of the random matrix used for finding λ_{max} and λ_{min} and the time in seconds it took each method to find λ_{max} and λ_{min}.

Matrix Size	QR	Bisection	Power
15×15	2.6930	37.5330	15.2110
10×10	1.7020	26.0380	12.3680
10×10	1.7320	25.7270	12.0680
9×9	1.7220	22.9430	11.5960
8×8	1.7530	20.9700	10.0440
7×7	1.7120	18.8670	9.6340
6×6	1.5020	16.6340	8.4020
5×5	1.1220	15.3620	7.9210
5×5	1.1620	15.3120	8.3320
4×4	1.0820	13.2990	7.0000

2.4 Remarks

All the experiments were run using MATLAB Version 6.0. MATLAB has a built in function, eig, to find the eigenvalues of a matrix. The eig function uses the QR Algorithm. In our tests we used the eig function along with the built-in max and min functions. The results show that the QR Algorithm is simply the fastest method for finding λ_{max} and λ_{min}. However, these built-in functions have been highly optimized by MATLAB's programmers for speed and accuracy. There may exist ways to improve the Bisection and Power methods so that they are able to compete with the QR Algorithm.

§3 Analytic Center and Repelling Limits

Another method of identifying necessary constraints in a semidefinite programming problem is to utilize the concepts of weighted analytic center and repelling limits.

3.1 A Weighted Analytic Center for LMIs

Consider the previous example of linear matrix inequalities and its feasible region, $\mathcal{R}$. Let us define the barrier function $\phi_w : \mathbb{R}^n \to \mathbb{R}$

$$\phi_w(x) = \begin{cases} \sum_{j=1}^{q} w_j logdet[(A^{(j)}(x))^{-1}] & \text{if } x \in \mathcal{R} \\ \infty & \text{otherwise} \end{cases}$$

The analytic center can be determined in the following manner:
given $x \in \mathcal{R}$ and tolerance $\epsilon > 0$

1. $s = -H(x)^{-1}\nabla\phi_w(x)$ where $H(x)$ is the Hessian of ϕ_w and $\nabla\phi_w(x)$ is the gradient.
2. set $d = \sqrt{s^T H(x) s}$
3. if $d > 0.25$ set $h = \frac{1}{1+d}$ else let $h = 1$. This step ensures staying within the feasible region.
4. $x = x + hs$ until $d \leq \epsilon$

Definition 0.11 - *Let w_k be a positive weight vector in $\mathbb{R}^q$ with the kth component α and all other components equal to one. Then the* repelling limit *associated with the kth LMI constraint is the trajectory of $s^{(k)}(\alpha) = x_{ac}(w_k)$ as $\alpha \to \infty$ or $\alpha \to 0$.*

3.2 Necessary Constraints and Repelling Limits

The necessary constraints of a system of LMIs can be identified by using repelling limits. The algorithm for using repelling limits to get "good" standing points is as follows:

1. Assign a large weight(or small) to one constraint while holding the others constant at 1.
2. Calculate weighted analytic center
3. Repeat

3.3 Results of repelling limit algorithm:

We apply this algorithm to our system of linear matrix inequalities given above and the resulting ten points, eight of which are unique are shown below:

weight	x	y
$w_1 \to \infty$	1.9088	-5.9993
$w_1 \to 0$	0.1520	-2.924
$w_2 \to \infty$	1.7498	0.9992
$w_2 \to 0$	0.2249	-5.6631
$w_3 \to \infty$	-0.9999	-5.9999
$w_3 \to 0$	1.0494	-3.5541
$w_4 \to \infty$	-0.9994	-5.9989
$w_4 \to 0$	1.1365	-3.4793
$w_5 \to \infty$	1.9090	-5.9998
$w_5 \to 0$	.4704	-3.7643

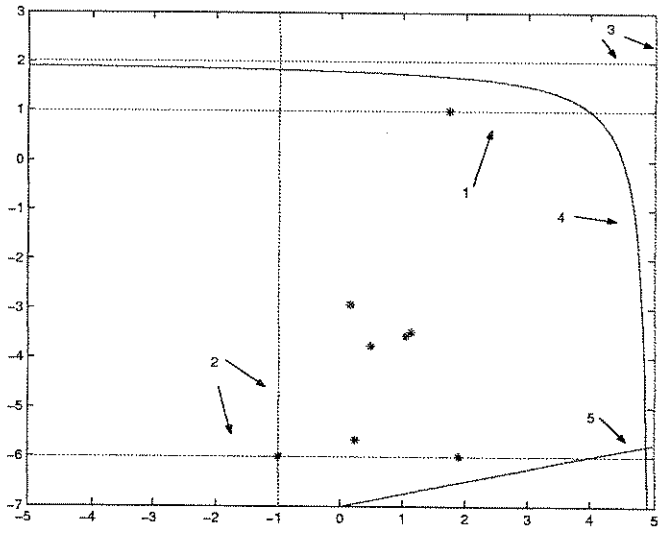

Figure 3: Feasible region with weighted analytic center points

Once the weighted analytic centers are known we can check whether each point is an interior point or if it is on the boundary of $\mathcal{R}$. One way of doing this is by checking the eigenvalues. However this method can be very slow as seen earlier. A faster method of determining the type of point is by checking whether or not the matrix has a Cholesky decomposition. Remember that only positive definite matrices can be decomposed into Cholesky form. Therefore, if the matrix has a Cholesky decomposition, it is an interior point. Otherwise it is on the boundary. However this method proved not to be feasible, in that the points are not actually on the boundary. A method that can be easily checked with a computer is checking whether or not the determinant of the matrix is tending towards zero.

Consider the following repelling limit: $(1.9088, -5.9993)$ Plugging this into each constraint we have:

$$\det[A^{(1)}(x)] = \det \begin{bmatrix} 2.9088 & 0 \\ 0 & 4.9993 \end{bmatrix} = 20.3596$$

$$\det[A^{(2)}(x)] = \det \begin{bmatrix} 2.9099 & 0 \\ 0 & 0.0007 \end{bmatrix} = 0.0020$$

$$\det[A^{(3)}(x)] = \det \begin{bmatrix} 3.0912 & 0 \\ 0 & 7.9993 \end{bmatrix} = 24.7274$$

$$\det[A^{(4)}(x)] = \det \begin{bmatrix} 3.0912 & -1 \\ -1 & 7.9993 \end{bmatrix} = 23.7274$$

$$\det[A^{(5)}(x)] = \det \begin{bmatrix} .5235 & 0 \\ 0 & .5235 \end{bmatrix} = 0.2741$$

It is clear that det $A^{(2)}(x)$ is tending towards zero and it is safe to say that (1.9088, −5.9993) is a boundary point.

Now let us consider the repelling limit (0.152, −2.924)

$$\det[A^{(1)}(x)] = \det \begin{bmatrix} 1.152 & 0 \\ 0 & 3.924 \end{bmatrix} = 4.5204$$

$$\det[A^{(2)}(x)] = \det \begin{bmatrix} 1.152 & 0 \\ 0 & 3.076 \end{bmatrix} = 3.5436$$

$$\det[A^{(3)}(x)] = \det \begin{bmatrix} 4.847 & 0 \\ 0 & 4.924 \end{bmatrix} = 23.8716$$

$$\det[A^{(4)}(x)] = \det \begin{bmatrix} 4.847 & -1 \\ -1 & 4.924 \end{bmatrix} = 22.8716$$

$$\det[A^{(5)}(x)] = \det \begin{bmatrix} 4.038 & 0 \\ 0 & 4.038 \end{bmatrix} = 16.3054$$

Each determinant is greater than zero, so we infer that (0.152, −2.924) is an interior point.

By checking the remaining determinants we find that three points are on the bounday and the remaining five points are all interior points. The constraints identified by the boundary points are constraints 1 and 2. The remaining five points are all good standing points that can be used to identify the remaining necessary constraints.

The interior points generated by the repelling limits algorithm are especially "good" standing points. These points are pushed towared a particular boundary by increasing or decreasing their weight. Since these points are "biased" toward the boundary they are more likely to indentify necessary constraints. One way to take advantage of this property is to use each interior point identified through the repelling limits algorithm as a standing point.

Using this method, we travel from one interior point to the next and run SSH on each point. The advantage to this, is of course, an increased likelihood that SSH will find the all the necessary constraints. However, remember that SSH must first calculate $A^{(j)}(x_0)^{-\frac{1}{2}}$ for $1 \leq j \leq q$. This means that for each new standing point the inverse must be recalculated. This is a drawback in that it increases the time needed to find each necessary constraint.

§4 Conclusion and Further Research

The Semidefinite Stand-and-Hit Algorithm is a valuable tool for identifying necessary linear matrix inequalities in a semidefinite programming problem. SSH Algorithm identifies the necessary LMIs by finding the maximum and minimum eigenvalues of a particular matrix. This is the main source of computational complexity for the SSH Algorithm. Therefore, in order to decrease the run-time of SSH, one must minimize the amount of time it takes to solve for the maximum and minimum eigenvalues. In our experiments we found that the fastest method is the QR Algorithm, although there may exist ways to optimize the power method or bisection method so that they are able to compete with QR. Another idea that might be able to improve SSH is the use of repelling limits, calculated by finding a weighted analytic center. These repelling limits produce points "pushed" toward the boundaries of the feasible region, making it more likely to hit the necessary constraints. These repelling limits can be implemented into the SSH Algorithm to reduce the time required to identify all the necessary linear matrix inequalities.

§5 Acknowledgements

I would like to thank Dr. Shafiu Jibrin for assisting me in the completion of this work.

References

[1] Jibrin, Shafiu and Pressman, Irwin. *Monte Carlo Algorithms for the Detection of Necessary Linear Matrix Inequalities.*

[2] Boyd, Steven and Vandenberghe, Lieven. *Semidefinite Programming.*

[3] Golub, Gene H. and Van Loan, Charles F. *Matrix Computations.*

[4] Jibrin, Shafiu and Pressman, Irwin S. *A Weighted Analytic Center for Linear Matrix Inequalities*