Convex Optimisation for Sound Field Synthesis

Paul Teal
Technical Report for CARlab
University of Sydney

8 March 2010

Abstract

An approach and algorithm for computing the coefficients for sparsity-based sound field synthesis is outlined. It is shown that the problem can be formulated as a Second Order Cone Program, and an approach to numerical solution of the problem is outlined. The solution reached is accurate, and the algorithm is very fast: possibly even implementable in real time.

1 Introduction

In [4] an approach is suggested for finding the signals to send to an array of loudspeakers for reproduction of a particular sound field. The basic idea is that the original sound field may be able to be described by a sparse set of functions. The resulting solution has several robustness characteristics, particularly with regard to spatial aliasing.

This paper discusses an approach to numerical solution of this problem.

2 Problem Formulation

In [4] the compressive sampling sound field is synthesised using a decomposition based on plane waves. The desired coefficients are those of $\mathbf{g}_{\text{plw}}$ which is the solution for the following optimisation problem:

$$\begin{aligned}
&\text{minimise } \|\mathbf{g}_{\text{plw}}\|_1 \\
&\text{subject to } \frac{\|\mathbf{T}_{\text{plw/mic}}\mathbf{g}_{\text{plw}} - \mathbf{s}_{\text{mic}}\|_2}{\|\mathbf{s}_{\text{mic}}\|_2} \leq \epsilon_1 \\
&\text{and to} \frac{\|\mathbf{g}_{\text{plw}} - \text{pinv}\left(\mathbf{T}_{\text{plw/HOA}}\right)\mathbf{b}_{\text{HOA}}\|_2}{\|\,\text{pinv}\left(\mathbf{T}_{\text{plw/HOA}}\right)\mathbf{b}_{\text{HOA}}\|_2} \leq \epsilon_2
\end{aligned} \tag{1}$$

All of the vectors and matrices in (1) are complex. As a first step towards devising an algorithm for its solution, we convert the problem to one using real variables. If $\mathbf{T}_{\text{plw/mic}}$ is of size $M \times W$, and we set

$$\begin{aligned}
g &= \begin{pmatrix} \text{Re}\left(\mathbf{g}_{\text{plw}}\right) \\ \text{Im}\left(\mathbf{g}_{\text{plw}}\right) \end{pmatrix} \in \mathbb{R}^{2W\times 1} \\
s_m &= \begin{pmatrix} \text{Re}\left(\mathbf{s}_{\text{mic}}\right) \\ \text{Im}\left(\mathbf{s}_{\text{mic}}\right) \end{pmatrix} \in \mathbb{R}^{2M\times 1} \\
T_r &= \text{Re}\left(\mathbf{T}_{\text{plw/mic}}\right) \\
T_i &= \text{Im}\left(\mathbf{T}_{\text{plw/mic}}\right) \\
T &= \begin{pmatrix} T_r & -T_i \\ T_i & T_r \end{pmatrix} \in \mathbb{R}^{2M\times 2W} \\
g_t &= \begin{pmatrix} \text{Re}\left(\text{pinv}\left(\mathbf{T}_{\text{plw/HOA}}\right)\mathbf{b}_{\text{HOA}}\right) \\ \text{Im}\left(\text{pinv}\left(\mathbf{T}_{\text{plw/HOA}}\right)\mathbf{b}_{\text{HOA}}\right) \end{pmatrix} \in \mathbb{R}^{2W\times 1}
\end{aligned} \tag{2}$$

then the problem may be re-expressed as

$$\begin{aligned} &\text{minimise } \sum_{i=1}^{W} \sqrt{g_i^2 + g_{i+W}^2} \\ &\text{subject to } \|Tg - s_m\| \leq \|s_m\|\epsilon_1 \\ &\text{and to} \|g - g_t\| \leq \|g_t\|\epsilon_2 \end{aligned} \tag{3}$$

From here on we use the notation $\|\cdot\|$ to denote Euclidean norm unless otherwise specified.

Note that (3) faithfully reproduces the complex ℓ_1-norm of (1). The use of the ℓ_1-norm is designed to produce sparsity in the solution $\mathbf{g}_{\text{plw}}$. Finding a solution which minimises the ℓ_1 norm of the real vector g does not necessarily produce sparsity in the vector $\mathbf{g}_{\text{plw}}$, since sparsity of the real or imaginary components may not coincide on the same components of the vector.

The problem may be re-expressed in the equivalent form

$$\begin{aligned} &\min \sum_{i=1}^{W} t_i \\ &\text{subject to } \left\| \begin{pmatrix} g_i \\ g_{i+W} \end{pmatrix} \right\| \leq t_i \qquad i = 1, \ldots, W \\ &\|Tg - s_m\| \leq \|s_m\|\epsilon_1 \\ &\|g - g_t\| \leq \|g_t\|\epsilon_2 \end{aligned} \tag{4}$$

We now define

$$\begin{aligned} y &= \begin{pmatrix} \text{Re}\,(\mathbf{g}_{\text{plw}}) \\ \text{Im}\,(\mathbf{g}_{\text{plw}}) \\ t \end{pmatrix} = \begin{pmatrix} g \\ t \end{pmatrix} \in \mathbb{R}^{3W \times 1} \\ A_i &= \begin{pmatrix} e_i & 0 \\ 0 & e_i \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{3W \times 2} \qquad i = 1, \ldots, W \\ b_i &= \begin{pmatrix} 0 \\ 0 \\ e_i \end{pmatrix} \in \mathbb{R}^{3W \times 1} \qquad i = 1, \ldots, W \\ A_{W+1} &= \begin{pmatrix} T^T \\ 0 \end{pmatrix} \in \mathbb{R}^{3W \times 2M} \\ A_{W+2} &= \begin{pmatrix} I \\ 0 \end{pmatrix} \in \mathbb{R}^{3W \times 2W} \end{aligned} \tag{5}$$

where $e_i \in \mathbb{R}^{W \times 1}$ is a vector of zeros, except with a one in the ith component.

This enables the problem to be formulated as

$$\begin{aligned} &\min f^T y \\ &\text{subject to } \|A_i^T y\| \leq b_i^T y \qquad i = 1, \ldots, W \\ &\|A_{W+1}^T y - s_m\| \leq \|s_m\|\epsilon_1 \\ &\|A_{W+2}^T y - g_t\| \leq \|g_t\|\epsilon_2 \end{aligned} \tag{6}$$

where $f \in \mathbb{R}^{3W \times 1}$ has zeros for the first $2W$ components and ones in the remaining W components. This is recognisable as a second order cone program (SOCP).

We have only to define a few additional variables:

$$\begin{aligned} N =& W + 2 \\ c_i =& 0 \in \mathbb{R}^{2\times 1} \qquad i = 1, \ldots, W \\ c_{W+1} =& -s_m \in \mathbb{R}^{2M\times 1} \\ c_{W+2} =& -g_t \in \mathbb{R}^{2W\times 1} \\ b_{W+1} =& 0 \in \mathbb{R}^{3W\times 1} \\ b_{W+2} =& 0 \in \mathbb{R}^{3W\times 1} \\ d_i =& 0 \in \mathbb{R} \qquad i = 1, \ldots, W \\ d_{W+1} =& \|s_m\|\epsilon_1 \in \mathbb{R} \\ d_{W+2} =& \|g_t\|\epsilon_2 \in \mathbb{R} \end{aligned}$$

in order to express the problem in the conventional SOCP manner:

$$\begin{aligned} &\min\ f^T y \\ &\text{subject to } \|A_i^T y + c_i\| \le b_i^T y + d_i \qquad i = 1, \ldots, N \end{aligned} \tag{7}$$

3 Second Order Cone Program

A constraint of the form

$$\|A^T y + c\| \le b^T y + d \tag{8}$$

where $A \in \mathbb{R}^{n\times k}$ is called a second order cone constraint, because it is equivalent to requiring the function $(A^T y + c, b^T y + d)$ to lie in the second order cone in $\mathbb{R}^{k+1}$.

There are various equivalent definitions of a second order cone of dimension k, but the one we use here is

$$\mathcal{K} = \left\{ x = (x_0, x_1) \in \mathbb{R} \times \mathbb{R}^{k-1} : x_0 - \|x_1\| \ge 0 \right\} \tag{9}$$

This means that the first element of the vector x must be larger than or equal to the Euclidean norm of the remaining elements (some authors use a definition where the last element is greater than or equal to the remainder of the first $k-1$ components).

There are more general definitions of second order cones which can be rotated, or more general definitions still for cones which are not second order, but they are not needed for this problem.

If we define

$$\begin{aligned} s_{i0} =& b_i^T y + d_i \in \mathbb{R} \\ s_{i1} =& A_i^T y + c_i \in \mathbb{R}^{k_i - 1} \end{aligned} \tag{10}$$

then it is apparent that each s_i conforms to the definition of a second order cone of dimension k_i, i.e.,

$$\mathcal{K}_i = \left\{ s = (s_{i0}, s_{i1}) \in \mathbb{R} \times \mathbb{R}^{k_i - 1} : s_0 - \|s_1\| \ge 0 \right\} \tag{11}$$

and

$$s = \begin{pmatrix} s_{10} \\ s_{11} \\ \vdots \\ s_{N0} \\ s_{N1} \end{pmatrix} = \begin{pmatrix} b_1^T y + d_1 \\ A_1^T y + c_1 \\ \vdots \\ b_N^T y + d_N \\ A_N^T y + c_N \end{pmatrix} = \begin{pmatrix} b_1^T \\ A_1^T \\ \vdots \\ b_N^T \\ A_N^T \end{pmatrix} y + \begin{pmatrix} d_1 \\ c_1 \\ \vdots \\ d_N \\ c_N \end{pmatrix} = \widehat{A}^T y + c \tag{12}$$

where

$$\widehat{A} = \begin{pmatrix} b_1 & A_1 & \ldots & b_N & A_N \end{pmatrix} \tag{13}$$

and

$$c = \begin{pmatrix} d_1 \\ c_1 \\ \vdots \\ d_N \\ c_N \end{pmatrix} \tag{14}$$

If we define the product of all the cones to be

$$\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2 \times \cdots \times \mathcal{K}_N \tag{15}$$

then the problem can be specified as

$$\min \left\{ f^T y : s = \widehat{A}^T y + c, s \in \mathcal{K} \right\} \tag{16}$$

If we define $b = -f$ and $A = -\widehat{A}$, then we have the equivalent problem

$$\max \left\{ b^T y : A^T y + s = c, s \in \mathcal{K} \right\} \tag{17}$$

This is the dual form of the SOCP accepted by many numerical solvers.

This can be solved using various numerical solvers, such as SeDuMi[1]. The parameter describing the constraint dimensions for SeDuMi [9] is the vector of cone dimensions, which in this case is

$$K.q = \left(\underbrace{3, 3, \ldots, 3}_{W}, 2M + 1, 2W + 1 \right) \tag{18}$$

SeDuMi ranks as one of the better solvers for SOCP in the comparison conducted in [6]. In that comparison SeDuMi achieved identical solutions to MOSEK[2], but was slightly slower. SeDuMi is released under the GNU/GPL license and is very easy to compile and install, whereas MOSEK is non-free.

If the dimensions of each of the N cones are $k_1, k_2, \ldots, k_N$, we define

$$K = k_1 + \cdots + k_N \tag{19}$$

(Note that SeDuMi can accept A and/or c which are complex, so it may be possible to reduce the dimensions of A to approximately $2W$ instead of $3W$, although the manual only says that components of x may be complex — not components of y. This has not been investigated further.)

4 Algorithms

In this Section some issues associated with implementation of an algorithm for solving a SOCP are discussed.

There are many different algorithms for solving SOCP. One of those tried was [7]. A modification of this is used in [1, Ch 14].

The primal form of the problem examined in these references is:

$$\min \left\{ c^T x : Ax = b, x \in \mathcal{K} \right\} \tag{20}$$

and the dual form

$$\max \left\{ b^T y : A^T y + s = c, s \in \mathcal{K} \right\} \tag{21}$$

[1] sedumi.ie.lehigh.edu
[2] www.mosek.com

Note that the cone in which x is constrained is exactly the same cone as that in which s is constrained, i.e., each cone $\mathcal{K}_i$ is *self dual*.

For a problem with a simple non-negativity constraint, each constraint variable (e.g., x_i) and it's corresponding Lagrange multiplier (e.g., s_i) must be greater than or equal to zero, and at optimality, their product must be zero (the "Karush-Kuhn-Tucker" (KKT) condition). This means that either the constraint is inactive ($x_i > 0$) and so the multiplier is 'irrelevant' ($s_i = 0$) or else the constraint is active ($x_i = 0$) and so the multiplier is applied ($s_i > 0$).

If we define a matrix $X = \text{diag}(x)$, then the KKT condition amounts to

$$Xs = Sx = 0 \tag{22}$$

The second order cone constraint is a generalisation of the same idea. In fact a non-negativity constraint can be considered as a cone constraint of dimension 1. For the SOCP, rather than each x_i and s_i being a single non-negative element, they are now vectors satisfying $x_i \in \mathcal{K}_i$ and $s_i \in \mathcal{K}_i$.

If we define the matrix

$$X_i = \begin{pmatrix} x_{i0} & x_{i1}^T \\ x_{i1} & x_{i0} I \end{pmatrix} \tag{23}$$

then the condition at optimality is $X_i s_i = S_i x_i = 0$. If we further define X as $\text{diag}(X_1, \ldots, X_n)$, and S as $\text{diag}(S_1, \ldots, S_n)$ then the condition at optimality for the entire problem is again (22).

The central path following algorithms do not attempt to simultaneously satisfy

$$\begin{aligned} Ax &= b \\ A^T y + s &= c \\ Xs = Sx &= 0 \end{aligned} \tag{24}$$

in one step. Instead they solve the slightly relaxed problem of

$$\begin{aligned} Ax &= b \\ A^T y + s &= c \\ Sx + Xs &= \nu e \end{aligned} \tag{25}$$

where $e \in \mathbb{R}^K$ is a vector all of ones and $\nu > 0$.

This is done by repeatedly solving the systems of equations

$$\begin{aligned} S\Delta x + X\Delta s &= \nu e - Xs \\ A^T \Delta y + \Delta s &= c - s - A^T y \\ A\Delta x &= b - Ax \end{aligned} \tag{26}$$

and then updating x, s and y according to

$$\begin{aligned} x^{k+1} &= x_k + \Delta x_k \\ s^{k+1} &= s_k + \Delta s_k \\ y^{k+1} &= y_k + \Delta y_k \end{aligned} \tag{27}$$

Each iteration of the algorithm consists of solving the system of equations for gradually decreasing values of the parameter ν. ν is given by $\nu = \sigma\mu$ where

$$\mu = \frac{x^T s}{N} \tag{28}$$

In [7] it is proven that if the initial values of x, s and y lie within a neighbourhood $\mathcal{N}_2(\gamma)$ of the central path, and σ and γ satisfy certain conditions, then each iteration of this algorithm will preserve the cone-satisfying property of x and s.

In practice it was found that although the algorithm will always converge to a solution satisfying (24), the solution usually (though not always) deviates from $s, x \in K$. Presumably the theorem of [7] is valid, and therefore the initial conditions, although strictly feasible, are not close enough to the central path. It is not trivial to obtain such an initial condition using the algorithm itself.

The modification proposed in [1] is a line search along the Newton direction to ensure that the cone constraint remains satisfied at every step: i.e., find a value $\alpha \in (0, 1)$ so that if

$$\begin{aligned} x^{k+1} &= x_k + \alpha \Delta x_k \\ s^{k+1} &= s_k + \alpha \Delta s_k \\ y^{k+1} &= y_k + \alpha \Delta y_k \end{aligned} \tag{29}$$

then $x \in \mathcal{K}$ and $s \in \mathcal{K}$. It was found in practice that the required value of α often rapidly approaches zero, and so the modification prevents progress towards minimising the objective function.

The approach discussed in [5] uses a barrier method (see Section 4.2.1) and so is slightly different from the "MZ-direction" used above, but seems to have a similar problem. The method proposed to deal with them is a *plane* search rather than a linear search, so that a different value of α is used for y from that used for x and s. An implementation of this method was commenced but has not been completed.

It seems as though the best solution to this problem, and the one used in both SeDuMi and MOSEK, is to use the following modification to (26):

$$\begin{aligned} \tilde{S} G^T \Delta x + \tilde{X} G^{-1} \Delta s &= \nu e - \tilde{X} \tilde{s} \\ A^T \Delta y + \Delta s &= c - s - A^T y \\ A \Delta x &= b - Ax \end{aligned} \tag{30}$$

where $\tilde{x} = G^T x$, $\tilde{s} = G^1 s$, and $\tilde{X}, \tilde{S}$ are the block diagonal matrices defined using (23) corresponding to $\tilde{x}$ and $\tilde{s}$. The particular choice of G used in the above mentioned solvers is the "NT-direction"[8] where G is a positive semi-definite matrix satisfying $Gx = G^{-1}s$. This has not yet been implemented.

4.1 Matrix Size Reduction

It is worth noting that (26) can be considerably simplified.

We can arrange

$$S \Delta x + X \Delta s = \nu e - Xs \tag{31}$$

as

$$\Delta x = S^{-1} \left(\nu e - Xs - X \Delta s \right) \tag{32}$$

to eliminate Δx.

This results in $A \Delta x = b - Ax$ becoming

$$\begin{aligned} A(S^{-1}(\nu e - Xs - X \Delta s)) &= b - Ax \\ AS^{-1} X \Delta s &= AS^{-1} \nu e - AS^{-1} Xs + Ax - b \\ &= AS^{-1} \nu e - b \end{aligned} \tag{33}$$

and so the system of equations has become

$$\begin{bmatrix} I & A^T \\ AS^{-1}X & 0 \end{bmatrix} \begin{bmatrix} \Delta s \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - s - A^T y \\ AS^{-1} \nu e - b \end{bmatrix} \tag{34}$$

The matrix on the left of this equation can be made symmetric replacing Δs with $XS^{-1} \Delta s$:

$$\begin{bmatrix} X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} S^{-1} X \Delta s \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - s - A^T y \\ AS^{-1} \nu e - b \end{bmatrix} \tag{35}$$

This can be solved using Cholesky decomposition.

Alternatively, using the matrix identity

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BHCA^{-1} & -A^{-1}BH \\ -HCA^{-1} & H \end{bmatrix} \qquad H = (D - CA^{-1}B)^{-1} \tag{36}$$

we can find that

$$\begin{bmatrix} S^{-1}X\Delta s \\ \Delta y \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ (AS^{-1}XA^T)^{-1}AS^{-1}X & -(AS^{-1}XA^T)^{-1} \end{bmatrix} = \begin{bmatrix} c - s - A^T y \\ AS^{-1}\nu e - b \end{bmatrix} \tag{37}$$

and so

$$\begin{aligned} \Delta y =& (AS^{-1}XA^T)^{-1}AS^{-1}X(c - s - A^T y) - (AS^{-1}XA^T)(AS^{-1}\nu e - b) \\ =& (AS^{-1}XA^T)^{-1}\left(AS^{-1}X(c - s - A^T y) - AS^{-1}\nu e + b\right) \end{aligned}$$

Since the matrix $AS^{-1}XA^T$ has size $3M \times 3M$ and is somewhat sparse, and $AS^{-1}X(c - s - A^T y) - AS^{-1}\nu e + b$ is a vector, this can be calculated quite efficiently. Δs can then be obtained by back substitution, and thence Δx.

The above simplifications have assumed that the matrix S^{-1} is available. The structure of S means that it is readily found. Since S is block diagonal, its inverse is simply the block diagonal consisting of the inverses of each of the blocks of S. i.e.,

$$S^{-1} = \text{diag}\left(S_1^{-1}, \ldots, S_N^{-1}\right) \tag{38}$$

Each of these blocks has the structure

$$S_i = \begin{pmatrix} s_{i0} & s_{i1}^T \\ s_{i1} & s_{i0}I \end{pmatrix} \tag{39}$$

If we define $\Delta = s_{i1}^s i1 - si0^2$, then it is easy to show that

$$S^{-1} = \begin{bmatrix} -s_{i0}/\Delta & s_{i1}/\Delta \\ s_1/\Delta & (I - s_{i1}s_{i1}^T/\Delta)/s_{i0} \end{bmatrix} \tag{40}$$

There can be some numerical issues caused by the cancellation of terms along the diagonal of the above expression, so it is better to compute each of the sums (e.g., $\sum_{j=2,j\neq l}^{k_i} s_{ij}^2 - s_{i0}^2$) separately.

4.2 Other methods

We briefly mention some other approaches to SOCP algorithms:

4.2.1 Barrier Function for Generalised Inequalities

The idea of barrier methods is to add to the objective continuous function that approximates the constraints.

This is discussed in Example 11.8 of [2, p599]

4.2.2 Standard Barrier

An interesting formulation of the SOCP as

$$\begin{aligned} &\min\ f^T y \\ &\text{subject to } \frac{\|A_i^T y + c_i\|^2}{b_i^T y + d_i} \leq b_i^T y + d_i \qquad i = 1, \ldots, N \\ &\qquad b_i^T y + d_i \geq 0 \qquad i = 1, \ldots, N \end{aligned} \tag{41}$$

is discussed in Problem 11.5 of [2, p623]

5 Cardinality and the ℓ-1 norm

The gaol of (1) is actually to minimise the number of non-zero components of the vector $\mathbf{g}_{\text{plw}}$, which is also known as its *cardinality*: $\text{card}(\mathbf{g}_{\text{plw}})$.

This is also sometimes termed the ℓ-0 norm, $\|\mathbf{g}_{\text{plw}}\|_0$, although it is not really a norm.

It is known (e.g., [3]) that minimisation of the ℓ-1 norm has a very large probability of achieving the same solution as the cardinality minimising solution.

An algorithm that can improve the solution is outlined below:

set $\mathbf{w} = \mathbf{1}$
repeat
  minimise $\|\operatorname{diag}(\mathbf{w})\mathbf{x}\|_1$ subject to $\mathbf{x} \in \mathcal{C}$
  $w_i = 1/(\epsilon + |x_i|)$
until convergence to a local point

For real positive x this algorithm can be interpreted as approximating cardinality by

$$\text{card}\, x \approx \log(1 + x/\epsilon) \tag{42}$$

The algorithm usually converges in 5 or fewer steps, and often gives a modest improvement in the solution (i.e., reduction in cardinality). This is very easy to implement for the algorithm discussed above by scaling the non-zero elements of the vector b.

A card4sound.m

Following is a listing of a Matlab function that implements the SOCP described above using SeDuMi.

```
function g = card4sound(T1,s_mic,g2,epsilon1,epsilon2,epsilon_sedumi,eps_card)
% g = card4sound(T1,s_mic,g2,epsilon1,epsilon2,epsilon_sedumi)
%
% Solve compressive sampling problem for sound field synthesis using sedumi
% i.e., solve the problem min ||g||_1 (complex l-1 norm)
% constrained by ||T1*g-s||_2/||s||_2 <= epsilon
% and by ||g-g2||_2/||g2||_2 <= epsilon2
% epsilon_sedumi is the numerical accuracy achieved sedumi stops iterating
%
% iterates to reduce the cardinality (set eps_card=0 to not iterate)
% Paul Teal 8.3.2010
%

if nargin<7
  eps_card = 1e-5;
end

[M, W] = size(T1);
T1R = real(T1);
T1I = imag(T1);
T1R = [T1R -T1I;T1I T1R];
s   = [real(s_mic); imag(s_mic)];
gt  = [real(g2)  ;imag(g2)];

K.q = [3*ones(W,1); M+M+1; W+W+1];
b = sparse([zeros(W+W,1); ones(W,1)]);

A = speye(3*W);
ndx = [ [2:3:3*W] [3:3:3*W] [1:3:3*W] ];
A = A(ndx,:);
c = sparse(3*W,1);

A = [A c [T1R'; sparse(W,M+M)] ];
A = [A c [eye(W+W); sparse(W,W+W)] ];
```

```
c = [c; norm(s) *epsilon1; -s ];
c = [c; norm(g2)*epsilon2; -gt];

pars.eps = 1e-8;
if nargin>5
  pars.eps = epsilon_sedumi;
end
disp('starting sedumi');

cardlast = W;
if eps_card==0
  cardlast = 0;
end
for i=1:10
  [x,y,info] = sedumi(-A,-b,c,K,pars);
  g = y(1:W)+1j*y(W+1:W+W);
  ag = abs(g);
  card = sum(abs(g)>epsilon_sedumi*10);
  disp(sprintf('cardinality = %d',card));
  if card >= cardlast
    break;
  end
  cardlast = card;
  b(W+W+1:3*W) = 1./(eps_card+ag);
end
```

References

[1] A. Antoniou and W. Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer, 2007.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] E.J. Candès, J.K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. Pure and Applied Mathematics*, LIX:1207–1223, 2006.

[4] N. Epain, C. Jin, and A. van Schaik. The application of compressive sampling to the analysis and synthesis of spatial sound fields. In *Audio Eng. Soc. Cenvention*, New York, Oct 2009.

[5] M.S. Lobo, L. Vandenberghe, and S. Boyd. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228, 1998.

[6] H.D. Mittelmann. An independent benchmarking of sdp and socp solvers. *Math. Program. Ser. B*, 95:407–430, 2003.

[7] R.D.C. Monteiro and T. Tsuchiya. Polynomial convergence of primal-dual algorithms for the second-order ccone program based on the MZ-family of directions. *Math. Programming*, 88:61–83, 2000.

[8] Y.E. Nesterov and M. Todd. Self-scaled barriers and interior point methods for convex programming. *Math. Operations Research*, 22(1):1–42, 1997.

[9] Imre Pólik. Addendum to the sedumi user guide, Jun 2005. Version 1.1.