REAL-TIME IMPLEMENTATIONS OF SPARSE LINEAR PREDICTION FOR SPEECH PROCESSING

Tobias Lindstrøm Jensen[1], *Daniele Giacobello*[2], *Mads Græsbøll Christensen*[3], *Søren Holdt Jensen*[1], *Marc Moonen*[4]

[1]Dept. of Electronic Systems, Aalborg Universitet, Denmark
[2]Office of the CTO, Broadcom Corporation, Irvine, CA, USA
[3]Audio Analysis Lab, Dept. of Architecture, Design and & Technology, Aalborg Universitet, Denmark
[4]Dept. of Electrical Engineering (ESAT-SCD) and iMinds Future Health Dept., KU Leuven, Belgium

{tlj,shj}@es.aau.dk, giacobello@broadcom.com, mgc@create.aau.dk, marc.moonen@esat.kuleuven.be

ABSTRACT

Employing sparsity criteria in linear prediction of speech has been proven successful for several analysis and coding purposes. However, sparse linear prediction comes at the expenses of a much higher computational burden and numerical sensitivity compared to the traditional minimum variance approach. This makes sparse linear prediction difficult to deploy in real-time systems. In this paper, we present a step towards real-time implementation of the sparse linear prediction problem using hand-tailored interior-point methods. Using compiled implementations the sparse linear prediction problems corresponding to a frame size of $20\,\mathrm{ms}$ can be solved on a standard PC in approximately $2\,\mathrm{ms}$ and orders faster than with general purpose software.

Index Terms— Sparse linear prediction, convex optimization, real-time implementation, speech analysis.

1. INTRODUCTION

Linear prediction (LPC) is, arguably, the most used parametric modeling technique for the analysis and coding of speech signals [1]. Minimum variance LPC with the 2-norm criterion has found a widespread use, mostly for its amenability of producing an optimization problem that is attractive both theoretically and computationally. Theoretically, this method corresponds to the maximum likelihood (ML) approach when the prediction error signal is considered to be i.i.d. Gaussian, making it mathematically tractable [2]. Furthermore, according to Parseval's theorem, minimizing the 2-norm of the prediction error in the time-domain is equivalent to minimizing the error between the true and estimated spectra, thus giving LPC an easy spectral interpretation. Computationally, the minimization of the 2-norm of the prediction error results in the Yule-Walker equations which can be solved efficiently via the Levinson recursion. Stability is intrinsically guaranteed by the construction of the problem [3] and can be easily preserved by the numerical robustness of the Levinson recursion. Nevertheless, in LPC of speech, sparsity criteria have been shown to provide a valid alternative to the 2-norm minimization criterion, overcoming most of its deficiencies in modeling and coding [4–8]. In particular, in [6], a new formulation for speech coding is introduced that provides not only a sparse approximation of the prediction error, which allows for a simple coding strategy, but also a sparse approximation of a high-order predictor which successfully models jointly short-term and long-term redundancies.

Sparse LPC can be formulated as a convex optimization problem, specifically as a linear programming (LP) problem. In order to be deployed in real-time applications, it requires its convex optimization core to be embedded directly in the algorithm that runs online and where strict real-time constraints apply[1]. While convex optimization problems can be efficiently solved, both in theory, with worst-case polynomial complexity [9] and in practice, see e.g., [10, 11], it is rarely limited in its implementation by real-time constraints. Its employment in optimization problems is generally limited to design purposes, e.g., finding the coefficients of finite impulse response filters [12] or offline signal processing, e.g., image denoising [13]. However, modern algorithms along with technology advances in processing power, have dramatically reduced solution times. This introduces the possibility of embedding convex optimization directly in signal processing algorithms that run online, with strict real-time constraints [14, 15].

In this paper, we propose a LP implementation of the sparse LPC problem using interior point methods. By hand-tailoring the solver to this particular problem we are able to obtain a solution time that is orders faster than with general purpose software. The paper is structured as follows. In Sec. 2 we define our notation and give an introduction to sparse LPC. In Sec. 3 we give two methods for solving the sparse LPC problem and provide details to the implementation. In Sec. 4 we provide experimental data of the timing benchmarks and discuss and conclude on the results in Sec. 5.

2. SPARSE LINEAR PREDICTION

We consider the following speech production model, where a sample of speech $x[t]$ is written as a linear combination of K past samples

$$x[t] = \sum_{k=1}^{K} \alpha_k x[t-k] + r[t], \tag{1}$$

where $\{\alpha_k\}$ are the prediction coefficients and $r[t]$ is the prediction error. Considering this model for a segment of T speech samples $x[t], t = 1, 2, \ldots, T$ in matrix form

The work of T. L. Jensen is supported by The Danish Council for Strategic Research under grant number 09-067056.

[1]LPC in traditional speech coders is usually performed every $5 - 10\,\mathrm{ms}$ [1].

$$x = X\alpha + r, \tag{2}$$

where

$$x = \begin{bmatrix} x[T_1] \\ \vdots \\ x[T_2] \end{bmatrix}, X = \begin{bmatrix} x[T_1-1] & \cdots & x[T_1-n] \\ \vdots & & \vdots \\ x[T_2-1] & \cdots & x[T_2-n] \end{bmatrix}. \tag{3}$$

The general LPC problem is then written as

$$\underset{\alpha\in\mathbb{R}^n}{\text{minimize}} \quad \|x - X\alpha\|_p^p + \gamma\|\alpha\|_k^k. \tag{4}$$

The starting and ending points T_1 and T_2 can be chosen in various ways by assuming $x[t] = 0$ for $t < 1$ and $t > T$. Here, we will use the most common choice of $T_1 = 1$ and $T_2 = T + K$, which is equivalent, when $p = 2$ and $\gamma = 0$, to the *autocorrelation method* [16]. With $m = K + T$ and $n = K$ then $x \in \mathbb{R}^m$, $X \in \mathbb{R}^{m\times n}$, $\alpha \in \mathbb{R}^n$. The introduction of the regularization term with γ in (4) can be seen as being related to the prior knowledge of the prediction coefficients vector α. While sparsity is often measured by the cardinality, we will use the more computational tractable 1-norm $\|\cdot\|_1$, which is known throughout the sparse recovery literature (see, e.g., [17]) to perform well as a relaxation of the cardinality measure with equivalence in certain cases. The problem then becomes

$$\underset{\alpha\in\mathbb{R}^n}{\text{minimize}} \quad \|x - X\alpha\|_1 + \gamma\|\alpha\|_1. \tag{5}$$

By exploiting the sparsity of the high-order predictor and the sparse prediction error, or *residual*, we are able to define a very simple speech coding scheme. For more on this we refer the reader to [6, 18].

3. METHODS

There are many methods for solving the sparse LPC (5). In this paper, we will focus on primal-dual interior-point methods. The purpose is twofold. Firstly, these are well known efficient methods for convex optimization used in real-time convex optimization [19, 20]; Secondly, by comparing the proposed solver with general-purpose software based on interior-point methods [11, 21], we can measure the improvement of hand-tailoring the algorithms for the same class of methods.

We note that for real-time convex optimization, there does exist specialized software. For small/sparse problems it is possible to use automatic tools for C code generation of a primal-dual interior-point method for a specific problem family which results in fast implementations [14, 19]. The sparse LPC problem is small but X is in general not sparse and even $m, n \approx 40$ will result in the number of coefficients in the problem to grow above the suggested limit of 4000 coefficients that the system can handle (as reported on cvxgen.com, software page of [19]). This effectively limits T and K in the sparse LPC to the smallest values considered in sparse LPC. In the numerical experiments, we will consider this method only for a small problem instance. The work in [20] presents a primal-dual algorithm for the sparse LPC problem ($\gamma = 0$). We extend this work to the sparse coefficients problem ($\gamma > 0$), and also consider compiled implementations, and timing based benchmarking among state-of-the-art methods.

The key element in interior-point methods is a fast and stable procedure for solving a linear system of equations in each iteration [10]. In the following, we will review two different methods for solving the problem (5) using interior-point methods and analyze the associated linear system. The details of the algorithms are given in [22, 23], respectively.

3.1. Dual Based Approach

Consider one standard form LP [22]

$$\begin{array}{ll} \text{minimize} & \bar{c}^T\bar{x} \\ \text{subject to} & \bar{A}\bar{x} = \bar{b} \\ & \bar{x} \succeq 0 \end{array} \tag{6}$$

where $\bar{c}, \bar{x} \in \mathbb{R}^{\bar{N}}$, $\bar{b} \in \mathbb{R}^{\bar{N}}$ and $\bar{A} \in \mathbb{R}^{\bar{M}\times\bar{N}}$ with the dual problem

$$\begin{array}{ll} \text{maximize} & \bar{b}^T\bar{\lambda} \\ \text{subject to} & \bar{A}^T\bar{\lambda} + \bar{s} = \bar{c} \\ & \bar{s} \succeq 0 \end{array} \tag{7}$$

where $\bar{\lambda} \in \mathbb{R}^{\bar{M}}$ and $\bar{s} \in \mathbb{R}^{\bar{N}}$. Then the sparse LPC problem (5) is a LP on the form (6) with

$$\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_4 \end{bmatrix}, \bar{c} = \begin{bmatrix} \mathbb{1} \\ \mathbb{1} \\ \gamma\mathbb{1} \\ \gamma\mathbb{1} \end{bmatrix}, \bar{A} = \begin{bmatrix} -I & I & -X & X \end{bmatrix}, \bar{b} = 2x \tag{8}$$

where $\mathbb{1} = [1, 1, \ldots, 1]^T$ of appropriate size and the original variable is related to the optimization variable by $\alpha = \frac{1}{2}(-\bar{x}_3 + \bar{x}_4)$. Following [22], we form the system of equations (normal-equations form) that that we must solve in each iteration as

$$\bar{A}\bar{D}\bar{A}^T\Delta\bar{\lambda} = -\bar{r} \tag{9}$$

where $\bar{D} = \mathbf{diag}([\bar{d}_1, \bar{d}_2, \bar{d}_3, \bar{d}_4]^T) \in \mathbb{R}^{(2m+2n)\times(2m+2n)}$ is a diagonal positive definite matrix and $\bar{r} \in \mathbb{R}^m$ is some right hand side. With $\bar{D}_1 = \mathbf{diag}(\bar{d}_1)$, $\bar{D}_2 = \mathbf{diag}(\bar{d}_2) \in \mathbb{R}^{m\times m}$ and $\bar{D}_3 = \mathbf{diag}(\bar{d}_3)$, $\bar{D}_4 = \mathbf{diag}(\bar{d}_4) \in \mathbb{R}^{n\times n}$ the coefficient matrix in (9) is then

$$\bar{A}\bar{D}\bar{A}^T = (\bar{D}_1 + \bar{D}_2) + X(\bar{D}_3 + \bar{D}_4)X^T. \tag{10}$$

Notice that this a $m \times m$ system ($m = \bar{M}$) connected to the dual variable $\bar{\lambda}$. We will call this approach the *dual* based approach and the corresponding algorithm is referred to as the dual based algorithm. This is a dense system which can be formed and solved in $\mathcal{O}(m^2n + m^3)$ operations via Cholesky factorization. Changing the linear systems of equations to normal-equations form and solving it via Cholesky factorization is regarded as the fastest method [10]. We prefer this method since we are aiming for speed. The dual based approach is then [22, Algorithm 14.3].

3.2. Primal Based Approach

In this second approach, we instead follow the standard LP form

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T\tilde{x} \\ \text{subject to} & \tilde{G}\tilde{x} \preceq \tilde{h} \end{array} \tag{11}$$

where $\tilde{c}, \tilde{x} \in \mathbb{R}^{\tilde{N}}$, $\tilde{h} \in \mathbb{R}^{\tilde{M}}$ and $\tilde{G} \in \mathbb{R}^{\tilde{M}\times\tilde{N}}$. The sparse LP problem (5) can be formulated as

$$\underset{\alpha\in\mathbb{R}^n}{\text{minimize}} \quad \|\tilde{A}\alpha - \tilde{b}\|_1 \ , \ \tilde{A} = \begin{bmatrix} -X \\ \gamma I \end{bmatrix}, \tilde{b} = \begin{bmatrix} -x \\ 0 \end{bmatrix} \tag{12}$$

which is on the form (11) with

$$\tilde{G} = \begin{bmatrix} \tilde{A} & -I \\ -\tilde{A} & I \end{bmatrix}, \quad \tilde{h} = \begin{bmatrix} \tilde{b} \\ -\tilde{b} \end{bmatrix}, \quad \tilde{c} = \begin{bmatrix} 0 \\ \mathbb{1} \end{bmatrix}. \tag{13}$$

Following, the primal-dual interior-point algorithm in [23], we need to solve problems of the form

$$\begin{bmatrix} 0 & \tilde{G}^T \\ \tilde{G} & -\tilde{D}^{-1} \end{bmatrix} \begin{bmatrix} \Delta\tilde{x} \\ \Delta\tilde{z} \end{bmatrix} = - \begin{bmatrix} r_{\mathrm{c}} \\ r_{\mathrm{h}} - \tilde{D}' r_{\mathrm{s}} \end{bmatrix} \tag{14}$$

where $\tilde{D} \in \mathbb{R}^{(2m+2n)\times(2m+2n)}$ is a positive definite diagonal matrix and $\tilde{D}' \in \mathbb{R}^{(2m+2n)\times(2m+2n)}$ is a diagonal matrix that changes in each iteration ($r_{\mathrm{c}} \in \mathbb{R}^{n+m}$ and r_{h}, $r_{\mathrm{s}} \in \mathbb{R}^{2m+2n}$ changes as well). Equation (14) can be reduced to the normal-equation form

$$\tilde{G}^T \tilde{D} \tilde{G} \Delta\tilde{x} = -r_{\mathrm{c}} - \tilde{G}^T \tilde{D}(r_{\mathrm{h}} - \tilde{D}' r_{\mathrm{s}}) \tag{15}$$

$$\Delta\tilde{z} = \tilde{D}(\tilde{G}\Delta x + r_{\mathrm{h}} + \tilde{D}' r_{\mathrm{s}}). \tag{16}$$

Since $r_{\mathrm{c}} = G^T \hat{z}$ where $\hat{z} \in \mathbb{R}^{2n+2m}$ is the current dual iterate the above system can with a $y \in \mathbb{R}^{2n+2m}$ be interpreted as

$$\tilde{G}^T \tilde{D} \tilde{G} \Delta\tilde{x} = -\tilde{G}^T y. \tag{17}$$

With (13) we can write (17) as

$$\begin{bmatrix} \tilde{A}^T & -\tilde{A}^T \\ -I & -I \end{bmatrix} \begin{bmatrix} \tilde{D}_1 & 0 \\ 0 & \tilde{D}_2 \end{bmatrix} \begin{bmatrix} \tilde{A} & -I \\ -\tilde{A} & -I \end{bmatrix} \begin{bmatrix} \Delta\alpha \\ \Delta v \end{bmatrix} = - \begin{bmatrix} \tilde{A}^T g_1 \\ g_2 \end{bmatrix} \tag{18}$$

with explicitly defined $g_1 \in \mathbb{R}^{n+m}$, $g_2 \in \mathbb{R}^n$ and $\Delta\tilde{x} = [\Delta\alpha,\ \Delta v]^T$. Following [24, §11.8.2], this can be reduced to a linear system of equations of the form

$$\tilde{A}^T \breve{D} \tilde{A} \Delta\alpha = -\tilde{A}^T \tilde{g} \tag{19}$$

where $\tilde{g} \in \mathbb{R}^{m+n}$ and $\breve{D} \in \mathbb{R}^{(m+n)\times(m+n)}$ is a positive definite diagonal matrix. The step Δv is then given as a function of $\Delta\alpha$ [24, §11.8.2]. The system (19) can be efficiently formed using (12) as

$$\tilde{A}^T \breve{D} \tilde{A} = X^T \breve{D}_1 X + \gamma^2 \breve{D}_2 \tag{20}$$

where $\breve{D} = \mathbf{diag}(\breve{d}) = [\mathbf{diag}(\breve{d}_1),\ \mathbf{diag}(\breve{d}_2)]^T$ and $\breve{D}_1 = \mathbf{diag}(\breve{d}_1) \in \mathbb{R}^{m\times m}$ and $\breve{D}_2 = \mathbf{diag}(\breve{d}_2) \in \mathbb{R}^{n\times n}$. The system in (19) is an $n \times n$ system connected to the *primal* variable α. The coefficient matrix of this system is positive definite if $\gamma > 0$ or the signal is full rank $\mathbf{rank}(X) = n$. We will call this approach the *primal* based approach and the corresponding algorithm is referred to as the primal based algorithm. This should be compared to the dual approach which leads to an $m \times m$ positive definite system. This implies that the efficiency of the primal and dual based approaches depends on the values of m and n. The linear system of equations in the primal based method can be formed and solved in $\mathcal{O}(n^2 m + n^3)$ operations via Cholesky factorization. The primal based algorithm is then designed following [23] without self-dual embedding.

The system (19) corresponds to the normal-equations for the weighted linear least-squares problem [24, §11.8.2]

$$\underset{\Delta\alpha\in\mathbb{R}^n}{\text{minimize}} \quad \| \breve{D}^{\frac{1}{2}} \left(\tilde{A}\Delta\alpha + \breve{D}^{-1}\tilde{g} \right) \|_2^2. \tag{21}$$

Since $\tilde{A}$ is of the form (12), the problem (21) can be reformulated as the generalized Tikhonov problem

$$\underset{\Delta\alpha\in\mathbb{R}^n}{\text{minimize}} \ \| \breve{D}_1^{\frac{1}{2}} \left(\breve{D}_1^{-1}\tilde{g}_1 - X\Delta\alpha \right) \|_2^2 + \gamma^2 \| \breve{D}_2^{\frac{1}{2}} \left(\Delta\alpha + \breve{D}_2^{-1}\tilde{g}_2 \right) \|_2^2 \tag{22}$$

where $\tilde{g} = [\tilde{g}_1,\ \tilde{g}_2]^T$. Notice the similarity between problem (22) and (5).

3.3. Implementation

The proposed algorithms are implemented in M (Matlab) and C++. The C++ implementation uses the LAPACK and BLAS library from the Intel Math Kernel Library (MKL). More specific operations, such as diagonal-times-vector and diagonal-times-matrix, are implemented using Hadamard products and `bsxfun` in Matlab, respectively. In C++, diagonal-times-vector is implemented as a loop and diagonal-times-matrix as a loop with some BLAS calls. The structure in the multiplication with the constraint matrices is also exploited, e.g., from (8)

$$\bar{A}\bar{x} = \bar{x}_2 - \bar{x}_1 + X(\bar{x}_4 - \bar{x}_3). \tag{23}$$

and it is then only necessary to apply matrix-vector multiplication with X ones for each matrix-vector multiplication with $\bar{A}$. The matrix-vector multiplication can also be implemented as a filtering operation but we prefer matrix-vector multiplication since it makes it possible to employ the highly optimized BLAS library MKL. Many computing units, such as the central processing unit (CPU) on a standard PC, are capable of performing more single precision operations than double precision operations per second. This is exploited in the implementations where the first iterations are executed in single precision. Single precision execution stops when the stopping conditions [10, p. 226] are satisfied with $\epsilon = 10^{-3}$. Double precision execution stops with $\epsilon = 10^{-6}$. If the Cholesky factorization fails in double precision the algorithms add 10^{-6} to the diagonal and retry the factorization.

We will call `Mprimal` the primal based algorithm implemented in double precision with M (Matlab) and `Cprimal` the one implemented in C++. Similarly for the dual based algorithms `Mdual` and `Cdual`. Algorithms using the single/double strategy are named `Cprimal(s/d)` and `Cdual(s/d)`. Implementations are available [2].

4. EXPERIMENTAL RESULTS

Benchmarking is performed with 3 settings denoted #1, #2 and #3 using a $\approx 2.5\,\mathrm{s}$ long vocalized speech signal sampled at $8\,\mathrm{kHz}$. In setting #1 and #2 each frame is $20\,\mathrm{ms}$ ($T = 160$ samples) with order $K = 100$, $K = 40$ respectively. Setting #3 processes $5\,\mathrm{ms}$ speech frames ($T = 40$) with order $K = 10$. Setting #3 may not be practical but is included to allow for a comparison with state-of-the-art methods and exemplifies the relation between scaling and speed.

The time from call of the solver to return is measured, excluding the time to form the data (matrices) that are given as input. The POSIX function `gettimeofday` is used to measure the execution time of the proposed algorithms in C++. For the simulations in Matlab we do a warm-start before measuring the timing [25]. The timing is measured over 100 solves of each frame to average out possible system processes (note that each frame is then static and the solvers

[2] Implementations with a Matlab mex interface can be obtained from `sparsesampling.com/sparse_lp`

then run with the exact same input). The setting $\gamma = 0.1$ was experimentally found to be a reasonably choice and fixed for all simulations. The solutions from the algorithms are validated by comparing the objective of all the solutions (no solution has an objective that is more than 0.004% larger than the smallest objective). The simulations are executed on an Intel(R) Dual Core(TM) i5-2410M CPU at 2.3 GHz with Ubuntu Linux kernel 3.2.0-32-generic, MKL 10.3 and Matlab 7.13.0.564. The algorithms implemented with C++ are compiled using gcc-4.6 and the `-Os -march=native` optimization option. The binaries and Matlab are executed with highest priority. We compare the implementation with the general purpose software `Mosek` 6.0 [11], `CVX+SeDuMi` 1.21 [21, 26], both via a Matlab interface, and a code generated solver from `CVXGEN` [19].

Methods	#1	#2	#3
`CVX+SeDuMi`	**416.39** 279.17/520.12	**344.73** 246.25/428.54	**172.29** 148.10/199.95
`Mosek`	**38.40** 28.05/44.00	**17.12** 14.15/41.06	**4.56** 3.60/4.82
`Mprimal`	**25.24** 14.41/35.48	**11.47** 6.32/14.54	**4.27** 2.26/6.08
`Mdual`	**23.49** 13.09/30.19	**13.55** 7.78/19.67	**3.15** 2.14/4.84
`CVXGEN`	N/A	N/A	**0.56** 0.38/0.72
`Cprimal`	**10.63** 6.70/13.58	**2.30** 1.51/2.75	**0.24** 0.14/0.41
`Cdual`	**13.79** 7.36/17.70	**5.52** 3.07/8.61	**0.41** 0.28/0.64
`Cprimal(s/d)`	**8.02** 5.29/10.64	**1.96** 1.36/2.29	**0.23** 0.15/0.30
`Cdual(s/d)`	**10.22** 5.08/14.69	**4.60** 2.23/6.96	**0.39** 0.24/0.63

Table 1. Timing (across frames) in milliseconds. Format: **Average** over min/max. The three settings are #1 $T = 160$, $K = 100$ ($m = 260$, $n = 100$), #2 $T = 160$, $K = 40$ ($m = 200$, $n = 40$), #3 $T = 40$, $K = 10$ ($m = 50$, $n = 10$).

The results are shown in Table 1. It is not possible to generate a solver from CVXGEN for setting #1 and #2 but setting #3 is small enough for CVXGEN to handle. From the table observe that the primal based algorithms are faster than the dual based algorithm. This is due to the fact that for all settings $m > n$ and then its computational cheaper to form and solve the linear system of equations in the primal based algorithm. We also see a modest decrease in timing for the larger problems when going from a double precision solver such as `Cprimal` to a mixed (single/double) precision solver such as `Cprimal(s/d)`. A speed-up is observed from M scripts to compiled C++ implementations. Specifically, considering the `Mprimal` and `Cprimal` algorithms there is a speed-up of #1: 2.4, #2: 5.0 and #3 17.8, i.e., a speed-up that increases as the optimization problem becomes smaller. This demonstrates that for small problems it is necessary with compiled implementations. Observe that the compiled algorithms using C++ compare favorably to `CVXGEN` for setting #3. Note that the fastest algorithm for setting #3 provide solutions in sub-milliseconds and is three orders faster than the slowest algorithm. `CVX+SeDuMi` is a highly used optimization software for prototyping and is only added here to highlight the potential speed-up that a hand-tailored algorithm can achieve. Minimum and maximum solve time is also presented in Table. 1, which provides important information for designing systems with hard time constraints. In these simulations we have fixed $\epsilon = 10^{-6}$ but to reduce maximum solve time, the algorithm could be altered to return after a certain fixed time with a less accurate solution.

5. DISCUSSION AND CONCLUSIONS

The first attempt to find a faster solution to the sparse LPC problem can be found in [8] where, acknowledging the impractical usage of the LP formulation in real-time systems, the sparse LPC problem is approximated using the Burg method for prediction parameters estimation based on the least absolute forward–backward error. In this approach, however, the sparsity is not preserved and this approximation only solves (5) for $\gamma = 0$. The work in [20], to the authors' knowledge, is the first to introduce a LP formulation for the sparse LPC problem (5) to reduce the complexity of its solution. Also in this approach, the solution is only defined for $\gamma = 0$. In this work, we extended and generalized the LP solution of (5) for $\gamma > 0$ and provide algorithmic details for a real-time solution of the sparse LPC problem for speech processing. In particular, exploiting the structure of the problem we can bring the size of the linear system of equations to the smallest possible (m or n). We would like to note that in general interior-point methods are not implemented using fixed-point do to requirements on high precision arithmetic [10].

6. REFERENCES

[1] J. H. L. Hansen, J. G. Proakis, and J. R. Deller Jr, *Discrete-time processing of speech signals*, Prentice-Hall, 1987.

[2] F. Itakura and S. Saito, "Analysis synthesis telephony based on the maximum likelihood method," in *Proc. 6th Int. Congress Acoust.*, 1968, vol. 17, pp. C17–C20.

[3] L. Knockaert, "Stability of linear predictors and numerical range of shift operators in normal spaces," *IEEE Trans. Inf. Theory*, vol. 38, no. 5, pp. 1483 –1486, Sep. 1992.

[4] C.-H. Lee, "On robust linear prediction of speech," *IEEE Trans. Acoust, Speech, Signal Process.*, vol. 36, no. 5, pp. 642 –650, May 1988.

[5] M. N. Murthi and B. D. Rao, "Towards a synergistic multi-stage speech coder," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 1998, pp. 369–372.

[6] D. Giacobello, M. G. Christensen, M. N. Murthi, S. H. Jensen, and M. Moonen, "Sparse linear prediction and its applications to speech processing," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 5, pp. 1644–1657, Jul. 2012.

[7] J. Makhoul, "Linear prediction: A tutorial review," *Proc. IEEE*, vol. 63, no. 4, pp. 561–580, Apr. 1975.

[8] E. Denoel and J.-P. Solvay, "Linear prediction of speech with a least absolute error criterion," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 33, no. 6, pp. 1397 – 1403, Dec. 1985.

[9] Yu. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Methods in Convex Programming*, SIAM, 1994.

[10] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, 1997.

[11] E. D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Math. Program. Series B*, pp. 249–277, Feb. 2003.

[12] L. Rabiner, "Linear program design of finite impulse response (FIR) digital filters," *IEEE Trans. Audio Electroacoust.*, vol. 20, no. 4, pp. 280 – 288, Oct. 1972.

[13] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006.

[14] J. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *IEEE Signal Process. Mag., Special Section – Convex Optimization in Signal Processing*, vol. 27, no. 3, pp. 50–61, May 2010.

[15] B. Defraene, T. van Waterschoot, H. J. Ferreau, M. Diehl, and M. Moonen, "Real-time perception-based clipping of audio signals using convex optimization," *IEEE Tran. Audio Speech Lang. Process.*, vol. 20, no. 10, pp. 2657–2671, Dec. 2012.

[16] P. Stoica and R. L. Moses, *Spectral analysis of signals*, Pearson/Prentice Hall, 2005.

[17] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization," *Proc. Natl. Acad. Sci. USA*, vol. 4, no. 5, pp. 2197–2202, Mar. 2003.

[18] D. Giacobello, M. G. Christensen, M. N. Murthi, S. H. Jensen, and M. Moonen, "Speech coding based on sparse linear prediction," in *Proc. Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2009, pp. 2524–2528.

[19] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, Mar. 2012.

[20] G. Alipoor and M. H. Savoji, "Wide-band speech coding based on bandwidth extension and sparse linear prediction," in *Int. Conf. Telecommun. Signal Process. (TSP)*, Jul. 2012, pp. 454–459.

[21] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optim. Methods Softw.*, vol. 11-12, pp. 625–653, 1999.

[22] J. Nocedal and S. Wright, *Numerical Optimization*, Springer Verlag, 1999.

[23] L. Vandenberghe, "The CVXOPT linear and quadratic cone program solvers," 2010, Available from `abel.ee.ucla.edu/cvxopt/documentation/coneprog.pdf`.

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

[25] T. Larsen, G. Pryor, and J. Malcolm, "Jacket: GPU powered MATLAB acceleration," in *NVIDIA Computing Gems: Jade Edition*, W.-M. W. Hwu, Ed., chapter 28. Morgan-Kaufman, Oct. 2011.

[26] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," `http://cvxr.com/cvx/`, Apr. 2011.