
ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Linear Algebra and its Applications 366 (2003) 295–315

LINEAR ALGEBRA AND ITS APPLICATIONS

www.elsevier.com/locate/laa

Efficient implementation of a structured total least squares based speech compression method☆

Philippe Lemmerling [a,*,1], Nicola Mastronardi [b], Sabine Van Huffel [a]

[a] *ESAT-SISTA/COSIC, Department of Electrical Engineering, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee, Leuven, Belgium*

[b] *Isituto per le Applicazioni del Calcolo "M. Picone", Sez. Bari, Via G. Amendola, 122/I, I-70126 Bari, Italy*

Received 29 January 2002; accepted 12 June 2002

Submitted by D.A. Bini

Abstract

We present a fast implementation of a recently proposed speech compression scheme, based on an all-pole model of the vocal tract. Each frame of the speech signal is analyzed by storing the parameters of the complex damped exponentials deduced from the all-pole model and its initial conditions. In mathematical terms, the analysis stage corresponds to solving a structured total least squares (STLS) problem. It is shown that by exploiting the displacement rank structure of the involved matrices the STLS problem can be solved in a very fast way. Synthesis is computationally very cheap since it consists of adding the complex damped exponentials based on the transmitted parameters.

The compression scheme is applied on a speech signal. The speed improvement of the fast vocoder analysis scheme is demonstrated. Furthermore, the quality of the compression scheme

☆ This paper presents research results of the Belgian Programme on Interuniversity Poles of Attraction (IUAP P4-02 and P4-24), initiated by the Belgian State, Prime Minister's Office—Federal Office for Scientific, Technical and Cultural Affairs, of the Brite Euram Programme, Thematic Network BRRT-CT97-5040 'Niconet', of the Concerted Research Action (GOA) projects of the Flemish Government MEFISTO-666 (Mathematical Engineering for Information and Communication Systems Technology) and of the FWO (Fund for Scientific Research Flanders) projects G0200.00 and G078.01.

* Corresponding author. Tel.: +32-16-321796; fax: +32-16-321970.

E-mail address: philippe.lemmerling@esat.kuleuven.ac.be (P. Lemmerling).

[1] Philippe Lemmerling is supported by a post-doctoral K.U. Leuven scholarship.

0024-3795/03/$ - see front matter © 2003 Elsevier Science Inc. All rights reserved.
doi:10.1016/S0024-3795(02)00465-2

is compared with that of a standard coding algorithm, by using the segmental signal-to-noise ratio.
© 2003 Elsevier Science Inc. All rights reserved.

Keywords: Displacement rank; Complex damped exponentials; Speech compression; Generalized Schur algorithm

1. Introduction

This paper presents a fast implementation of a recently proposed speech compression scheme [15]. The compression scheme belongs to the class of vocoders that use an all-pole model for modeling the vocal tract. The resulting minimum phase model is sufficient for preserving the exact magnitude spectrum, whereas phase information is lost [11]. Most linear predictive coding (LPC) based techniques make the additional assumption that the input to the auto regressive (AR) model is white noise, represented by the vector e. If we represent the speech signal by a vector s and assume a model of order n, the modeling of the ith frame of the speech signal can be recasted as the following optimization problem:

$$\begin{aligned} &\min_{a(l),\, l=1,\ldots,n} \sum_{j=1+(i-1)N}^{iN} (e(j))^2, \\ &\text{where } s(k) - e(k) = \sum_{l=1}^{n} a(l)s(k-l), \\ &\quad k = 1 + (i-1)N + n, \ldots, iN, \end{aligned} \tag{1}$$

where N equals the number of samples per frame, $a(l),\ l = 1, \ldots, n$ are the so-called prediction coefficients. Note that we adopt a Matlab-like notation, where $v(i)$ indicates the ith element of vector v, and $v(i : j)$ represents the subvector of v, starting at the ith element and ending at the jth element of vector v.

A closer look at (1) reveals that the problem is in fact a least squares (LS) problem. This is the basic scheme used by well-known LPC based algorithms such as LPC-10 [20] or CELP [10] (in practice however, the prediction coefficients are not determined by solving (1), but by using an equivalent autocorrelation method). At the receiver side, the speech is synthesized using the all-pole model based on the transmitted model parameters. In the case of a voiced frame, the input to the filter will be a periodic pulse with the transmitted pitch frequency, while in the unvoiced case the input is white noise. In the case of CELP the excitation is chosen out of a series of standardized noise-like sequences in order to obtain the best synthesis.

The recently proposed approach [15] is still based on the all-pole model but instead of solving (1), the following problem for the ith frame is solved:

$$\min_{\substack{\Delta s(j),\ j=1+(i-1)N,\ldots,iN,\\ a(l),\ l=1,\ldots,n}} \sum_{j=1+(i-1)N}^{iN} (\Delta s(j))^2 \tag{2}$$

$$\text{such that } s(k) + \Delta s(k) = \sum_{l=1}^{n} a(l)(s(k-l) + \Delta s(k-l)),$$

$$k = n + 1 + (i-1)N, \ldots, iN.$$

So instead of minimizing a prediction error, as in (1), we determine for each sample $s(k)$ a correction $\Delta s(k)$, such that the corrected signal $s(k) + \Delta s(k)$ exactly satisfies an AR model, with the correction as small as possible in L_2 norm.

It is interesting to note that this approach is related to what is known as sinusoidal coders where a frame of the speech signal is approximated by a sum of sinusoids. This can be seen as follows. The constraints in (2) basically require the Toeplitz matrix containing $s + \Delta s$ (starting in the upper right corner and ending in the lower left corner) to be rank deficient. It is well known that such a rank-deficient Toeplitz matrix can be parametrized by the parameters of the complex damped exponentials for which

$$s(t) + \Delta s(t) = \sum_{l=1}^{n} c_l z_l^t$$

with $\mathrm{j} = \sqrt{-1}$, $c_l = b_l \mathrm{e}^{\mathrm{j}p_l}$ a complex amplitude and $z_l = \mathrm{e}^{(\mathrm{j}2\pi f_l + d_l)}$ a complex signal pole, holds. The latter follows from the fact that due to the rank deficiency of the Toeplitz matrix, $s + \Delta s$ satisfies a linear prediction equation, represented by the prediction coefficients $a(l),\ l = 1, \ldots, n$. Starting from these coefficients, we can determine the frequencies $f_l,\ l = 1, \ldots, n$ and the dampings $d_l,\ l = 1, \ldots, n$ of the underlying complex damped exponentials (see e.g. [21]). The amplitudes $b_l,\ l = 1, \ldots, n$ and phases $p_l,\ l = 1, \ldots, n$ are obtained by solving a system of linear equations based on the above calculated frequencies and dampings. Note that in this paper we only consider real signals. Therefore the complex signal poles and amplitudes appear in complex conjugated pairs. Assuming that $f_l \neq 0,\ l = 1, \ldots, n$, only $(n/2)4$ real parameters are necessary to reconstruct $s(t) + \Delta s(t)$.

Summarizing, it can be seen that our new approach is related to sinusoidal coders since also sinusoidal components are used to approximate the speech signal. However, in our approach the sinusoidal components are damped, thereby requiring much less components (and thus higher compression ratio) than pure sinusoidal coders to obtain the same quality.

The representation of the frames of the speech signal by the parameters of the corresponding complex damped exponentials also yields many advantages from the practical point of view. First of all, synthesis of the frames is very cheap and furthermore this parametrization allows progressive speech compression. Progressive speech compression allows a variable degree of analysis/synthesis at the emitter/receiver depending on the availability of channel capacity and/or the requirements of the specific application. Furthermore the quality of the reconstructed signal may

also be subject to the specifications of the receiver and in particular to its ability to cope with increased computational load. With the representation of the approximated frame of the speech signal as a sum of complex damped exponentials this can easily be done by determining/transmitting/reconstructing a varying number of complex damped exponentials.

In the following section we describe the vocoder analysis scheme based on our new approach, by developing the kernel algorithm followed by the outline of the complete compression scheme. Section 4 presents numerical results and a qualitative comparison with a standard speech compression method, using a speech signal. We discuss the quality performance and the efficiency of the new approach. We end with conclusions.

2. Description of the vocoder analysis scheme

As already mentioned in the introduction, the kernel problem of our new approach can be formulated as in (2). It is easy to recast this optimization problem in a matrix framework:

$$\min_{\substack{\Delta s(j),\ j=1+(i-1)N,\ldots,iN,\\ x(l),\ l=1,\ldots,n}} \sum_{j=1+(i-1)N}^{iN} (\Delta s(j))^2 \tag{3}$$
$$\text{such that } (A+\Delta A)x = b + \Delta b,$$

where we used the convention that the vector $s(1+(i-1)N : iN)$ can be read from the first row and the first column of the Toeplitz matrix $[A\ b]$ (and the same convention for $\Delta s(1+(i-1)N : iN)$ and $[\Delta A\ \Delta b]$), by starting in the upper right corner and ending in the lower left corner. Furthermore a in (2) and x in (3) are related as follows:

$$[-1\ a^{\mathrm{T}}] \equiv [x^{\mathrm{T}}\ -1]/(-x(1)).$$

Note that problem (3) is called a Toeplitz structured total least squares (STLS) problem since both the matrices $[A\ b]$ and $[\Delta A\ \Delta b]$ have a Toeplitz structure. For ease of notation the first frame of the speech signal is considered in the remainder of this section:

$$[\Delta A \quad \Delta b] = \begin{bmatrix} \Delta s(n+1) & \Delta s(n) & \ldots & \Delta s(2) & \Delta s(1) \\ \Delta s(n+2) & \Delta s(n+1) & \ddots & \ddots & \Delta s(2) \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \Delta s(m+n) & \Delta s(m+n-1) & \ldots & \Delta s(m+1) & \Delta s(m) \end{bmatrix}. \tag{4}$$

As can be seen from (3) the STLS problem is a constrained optimization problem with a quadratic objective function and nonlinear constraints. Therefore, taking into

account the nature of the STLS problem, it should come as no surprise that all the algorithms for solving it will be iterative.

The first basic algorithm described here follows the same lines as the heuristic algorithm developed in [18] for a similar (namely A Toeplitz but b unstructured) STLS problem: in each iteration the equality constraints of (3) are linearized around the current solution point (a solution point being determined by Δs and x). The Toeplitz STLS problem considered here is also treated in [18], but in that case a penalty function approach is proposed. The latter means that the constrained optimization problem (3) is transformed into an unconstrained optimization problem, simply by adding the weighted constraints to the original objective function, resulting in an unconstrained optimization problem (see e.g. [7,8]). However, for the unconstrained problem to be equivalent to the constrained problem, large weights need to be introduced, yielding ill-conditioned matrices and thus inaccurate results. Applying methods to overcome this ill-conditioning (see e.g. [1]) makes it difficult to develop fast algorithms. We therefore stick to the constrained optimization formulation of (3) and the above mentioned linearization of the constraints.

Before describing the algorithm, we introduce some notation. Let us represent small perturbations on Δs and x by $\Delta\tilde{s} \in \mathbb{R}^{(m+n)\times 1}$ and $\Delta\tilde{x} \in \mathbb{R}^{n\times 1}$ respectively. Furthermore let

$$r(\Delta s, x) = (A + \Delta A)x - b - \Delta b$$

and $X \in \mathbb{R}^{m\times(m+n)}$ is defined by

$$X\,\Delta s = [\Delta A \quad \Delta b]\begin{bmatrix} x \\ -1 \end{bmatrix}.$$

When $[\Delta A\ \Delta b]$ is Toeplitz this yields

$$X = \begin{bmatrix} -1 & x(n) & \cdots & x(1) & 0 & \cdots & \cdots & 0 \\ 0 & -1 & x(n) & \cdots & x(1) & 0 & & \vdots \\ \vdots & & \ddots & \ddots & & \ddots & & \vdots \\ 0 & & & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & -1 & x(n) & \cdots & x(1) \end{bmatrix}.$$

The iterative algorithm is obtained by replacing in (3) Δs and x by $\Delta s + \Delta\tilde{s}$ and $x + \Delta\tilde{x}$ respectively, followed by a linearization of the constraints around the current solution point $[\Delta s^{\mathrm{T}}\ x^{\mathrm{T}}]^{\mathrm{T}}$ (simply omit second order terms that occur). We then obtain the following algorithm:

Algorithm STLS1

Input: $[A\ b] \in \mathbb{R}^{m\times(n+1)}$
Output: the parameter vector $x \in \mathbb{R}^{n\times 1}$ and $\Delta s \in \mathbb{R}^{(m+n)\times 1}$ (i.e. the minimal representation of the matrix $[\Delta A\ \Delta b]$)

Step 1: Initialize Δs and x
Step 2: while *stopcriterion* not satisfied
Step 2.1: Solve the following equality constrained LS problem:

$$\min_{\Delta\tilde{s},\Delta\tilde{x}} \|\Delta s + \Delta\tilde{s}\|_2^2$$
$$\text{such that } r(\Delta s, x) + J \begin{bmatrix} \Delta\tilde{s} \\ \Delta\tilde{x} \end{bmatrix} = 0$$

Step 2.2: $\Delta s \leftarrow \Delta s + \Delta\tilde{s}$
$x \leftarrow x + \Delta\tilde{x}$
end

where $J = [X \;\; A + \Delta A]$ is the Jacobian of the constraints $r(\Delta s, x)$ w.r.t. $v \equiv [\Delta s^{\mathrm{T}} \;\; x^{\mathrm{T}}]^{\mathrm{T}}$. The choice of the stop criterion depends on the application at hand. In the remainder of the paper the following stop criterion is used:

$$\|\Delta\tilde{s}\|_2 < \text{tol}.$$

Note that in Section 4 tol is set to 0.01. For solving the equality constrained LS problem in Step 2.1 of Algorithm STLS1, we make use of the generalized RQ (GRQ) factorization [2,3]. The latter GRQ factorization is a two-step procedure consisting of one RQ factorization followed by a QR factorization. For the RQ factorization a fast implementation exploiting the low displacement rank of the involved matrices could be developed. However, the consecutive QR factorization does not operate on one of the originally structured matrices but on one of the original matrices multiplied with an orthogonal matrix to the right. Since the latter matrix is typically unstructured, it is not possible to improve the performance of this second step. We will however use this algorithm in the comparison of Section 4, since this algorithm is in fact a general algorithm that with only minor modifications can deal with any type of linearly STLS problem. The price to pay for this generality is the decrease in efficiency.

In order to be able to exploit the structure of the Toeplitz STLS problem we follow a different approach. Instead of eliminating the constraints of the STLS problem (3), we apply the Newton method for unconstrained optimization to the Lagrangian L of problem (3):

$$L(\Delta s, x, \lambda) = 1/2\, \Delta s^{\mathrm{T}}\, \Delta s - \gamma^{\mathrm{T}}(b - Ax - X\, \Delta s),$$

where $\gamma \in \mathbb{R}^{m\times 1}$ is a vector of Lagrange multipliers. The straightforward application of the Newton method (see e.g. [7,8]) on the Lagrangian L yields the second basic algorithm:

Algorithm STLS2

Input: $[A\ b] \in \mathbb{R}^{m\times(n+1)}$
Output: the parameter vector $x \in \mathbb{R}^{n\times 1}$ and $\Delta s \in \mathbb{R}^{(m+n)\times 1}$ (i.e. the minimal representation of the Toeplitz matrix $[\Delta A\ \Delta b]$)

Step 1: Initialize Δs, x and γ
Step 2: while *stopcriterion* not satisfied
Step 2.1: Solve the following system of equations:

$$\begin{bmatrix} S & J^{\mathrm{T}} \\ J & 0 \end{bmatrix} \begin{bmatrix} \Delta\tilde{s} \\ \Delta\tilde{x} \\ \Delta\tilde{\gamma} \end{bmatrix} = - \begin{bmatrix} g + J^{\mathrm{T}}\gamma \\ r(\Delta s, x) \end{bmatrix}$$

Step 2.2: $\Delta s \leftarrow \Delta s + \Delta\tilde{s}$
$x \leftarrow x + \Delta\tilde{x}$
$\gamma \leftarrow \gamma + \Delta\tilde{\gamma}$
end

where $g = \begin{bmatrix} \Delta s \\ 0 \end{bmatrix} \in \mathbb{R}^{(m+2n)\times 1}$ is the gradient of the objective function in (3) and $J = [X \;\; A + \Delta A]$ is the Jacobian of the constraints $r(\Delta s, x)$ in (3), all w.r.t. $v \equiv [\Delta s^{\mathrm{T}} \; x^{\mathrm{T}}]^{\mathrm{T}}$. Furthermore, for optimal convergence rate (superlinear) the matrix S should be set to $\nabla^2_{vv} L(\Delta s, x, \gamma)$. The latter means that in this case

$$S = \begin{bmatrix} I_{m+n} & 0 \\ 0 & 0 \end{bmatrix} - \sum_{i=1}^{m} \gamma(i) \nabla^2_{vv} r. \tag{5}$$

As shown in e.g. [7], S can also be chosen to be a positive definite approximation of $\nabla^2_{vv} L$, without changing the final solution of problem (3). Inclusion of the second term in (5) would render the structure of S rather complicated. We therefore only retain the first term in (5):

$$S = \begin{bmatrix} I_{m+n} & 0 \\ 0 & 0 \end{bmatrix}. \tag{6}$$

It is interesting to notice that Step 2.1 of algorithm STLS1 is basically the same as Step 2.1 of algorithm STLS2 (when the above approximation of S (see (6)) is used), although both algorithms are derived from a different starting point. The proof is easily obtained by applying the method of Lagrange multipliers to the subproblem described in Step 2.1 of algorithm STLS1. The system of equations that results from it is exactly the same as the system of equations that needs to be solved in Step 2.1 of algorithm STLS2 with the approximation of S described in (6). The latter observation also yields some additional insight in the convergence properties that can be expected [2] for algorithms STLS1 and STLS2. By omitting the curvature of the constraints in (5) the convergence rate is no longer quadratic but superlinear (see [7,8]). However, this is largely compensated by the fact that one iteration can be implemented in a very fast way, exploiting the low displacement rank structure of the matrices involved in Step 2.1 of Algorithm STLS2 *and* by taking advantage of the sparsity of the corresponding generators.

[2] The same conclusion applies to the similar but different STLS problem described in [18].

In order to develop a fast implementation for the kernel problem (i.e. Step 2.1) of algorithm STLS2 note that it corresponds to solving a system of linear equations:

$$Mz = b_1 \tag{7}$$

with

$$M = \begin{bmatrix} I_{m+n} & 0_{(m+n)\times n} & X^{\mathrm{T}} \\ 0_{n\times(m+n)} & 0_{n\times n} & \Lambda^{\mathrm{T}} \\ X & \Lambda & 0_{m\times m} \end{bmatrix}, \tag{8}$$

where $\Lambda = A + \Delta A \equiv \text{toeplitz}(\lambda(n : m+n-1), \lambda(n : -1 : 1))$, with $\text{toeplitz}(c, r)$ a shorthand notation for the Toeplitz matrix having c as its first column and r as its first row. The solution of (7) can be obtained by computing the LDL^{T} factorization of M, where L is lower triangular and D is a signature matrix. The solution is then found by solving the following linear systems:

$$\begin{aligned} &Lz_2 = b_1, \\ &Dz_1 = z_2, \\ &L^{\mathrm{T}}z = z_1. \end{aligned} \tag{9}$$

The latter factorization can be obtained in a fast way by an appropriate implementation of the generalized Schur algorithm. As will be described in the next subsection, a high computational efficiency is obtained by exploiting the low displacement rank of the Toeplitz-block-like matrix M *and* by taking advantage of the sparsity of the corresponding generators (note that a similar approach can be used to solve efficiently a different STLS problem described in [17]).

2.1. The generalized Schur algorithm

In this section we introduce the generalized Schur algorithm to compute the LDL^{T} factorization of a symmetric matrix A, where L is an upper triangular matrix and D is a signature matrix. A more extensive description of the algorithm can be found in [12]. Given a strongly regular [3] $n \times n$ matrix A, and define

$$D_A = A - ZAZ^{\mathrm{T}}$$

we say that the displacement rank of A is α if $\text{rank}(D_A) = \alpha$, where Z is a lower triangular matrix of order n. The choice of Z depends on the matrix A, e.g. if A is a Toeplitz matrix, Z is chosen equal to the shift matrix. If A is a block-Toeplitz matrix, Z is chosen equal to the block-shift matrix (for a more general choice of the matrix Z, see [12]). Clearly, D_A will have a decomposition of the form

$$D_A = G^{\mathrm{T}} J_A G,$$

[3] A square matrix A is said to be *strongly regular* if all its principal minors are different from zero.

where

$$G = \begin{bmatrix} g_1^{\mathrm{T}} \\ \vdots \\ g_p^{\mathrm{T}} \\ g_{p+1}^{\mathrm{T}} \\ \vdots \\ g_\alpha^{\mathrm{T}} \end{bmatrix}, \quad J_A = I_p \oplus -I_q, \quad q = \alpha - p,$$

where

$$P \oplus Q \equiv \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}.$$

The matrix $G \in \mathbb{R}^{\alpha \times n}$ and the vectors g_i, $i = 1, \ldots, \alpha$, are called the generator matrix and the generators of A, respectively. The generators $g_1, \ldots, g_p$ are said to be *positive*, the generators $g_{p+1}, \ldots, g_\alpha$ are said to be *negative*. The pair (p, q) is called the *displacement inertia* of D_A. A matrix Θ is said to be J_A-orthogonal if $\Theta^{\mathrm{T}} J_A \Theta = J_A$.

A generator matrix is not unique. In fact, if G is a generator matrix of A and Θ is a J_A-orthogonal matrix, then ΘG is a generator matrix of A too. A generator matrix is said to be in *proper* form if its first nonzero column has a single nonzero entry, i.e.

$$G = \begin{bmatrix} 0 & \ldots & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & * & \cdots & * \\ 0 & 0 & * & * & \cdots & * \\ 0 & 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & \ldots & 0 & * & \cdots & * \end{bmatrix},$$

where the elements denoted by "$*$" are generally different from zero, moreover, the corresponding row is called *pivot*.

The number of steps of the generalized Schur algorithm is equal to the order of the matrix A. Let $G_0 = G$ and denote by G_{i-1} the generator matrix at the beginning of the ith step. A J_A-orthogonal matrix Θ_i is chosen such that $H_{i-1} = \Theta_i G_{i-1}$ is in proper form. More precisely, denote by f_i the ith column of G_{i-1}. The index of the pivot has to be within $\{1, \ldots, p\}$ if $f_i^{\mathrm{T}} J f_i > 0$ (positive step), within $\{p+1, \ldots, \alpha\}$ if $f_i^{\mathrm{T}} J f_i < 0$ (negative step).

Denote this index by k. Then, the generator matrix G_i is updated in the following way:

$$G_i(k, :) = H_{i-1}(k, :) Z^{\mathrm{T}},$$

$$G_i([1 : k-1, k+1 : \alpha], :) = H_{i-1}([1 : k-1, k+1 : \alpha], :).$$

Furthermore, $H_{i-1}(k,:)^{\mathrm{T}}$ becomes the ith column of L. If $f_i^{\mathrm{T}} J f_i > 0$, we set $D(i,i) = 1$. If $f_i^{\mathrm{T}} J f_i < 0$, we set $D(i,i) = -1$. Observe that the case $f_i^{\mathrm{T}} J f_i = 0$ does not occur due to the strong regularity of A [12]. Since in general the matrix Θ_i is given by the product of a number of Givens and hyperbolic rotations proportional to α, the computational cost at the ith step is $\mathrm{O}(\alpha(n-i+1))$. Hence the computational cost of the generalized Schur algorithm is $\mathrm{O}(\alpha n^2)$.

2.1.1. The generalized Schur algorithm applied to M

Before applying the generalized Schur algorithm to M we observe that the matrix M is not strongly regular. In fact $\det(M(1:i,1:i)) = 0,\ i = m+n+1,\ldots,m+2n$. Hence a permutation matrix P is considered in order to transform M into the Toeplitz-block matrix $\tilde{K}$, i.e.

$$\tilde{K} = PMP^{\mathrm{T}} = \begin{bmatrix} I_{m+n} & X^{\mathrm{T}} & 0_{(m+n)\times n} \\ X & 0_{m\times m} & \Lambda \\ 0_{n\times(m+n)} & \Lambda^{\mathrm{T}} & 0_{n\times n} \end{bmatrix}.$$

It is easy to prove that $\tilde{K}$ is strongly regular. Considering the Schur complement of $I_{(m+n)\times(m+n)}$ in $\tilde{K}$ we can obtain the following partial LDL^{T} decomposition of $\tilde{K}$ without any additional cost (of course the product XX^{T} is not explicitly computed),

$$\tilde{K} = \begin{bmatrix} I_{m+n} & & \\ X & I & \\ 0_{n\times(m+n)} & & I \end{bmatrix} \begin{bmatrix} I_{m+n} & & \\ & -XX^{\mathrm{T}} & \Lambda \\ & \Lambda^{\mathrm{T}} & 0_{n\times n} \end{bmatrix} \times \begin{bmatrix} I_{m+n} & X^{\mathrm{T}} & 0_{(m+n)\times n} \\ & I & \\ & & I \end{bmatrix},$$

where the matrix

$$\hat{K} = \begin{bmatrix} -XX^{\mathrm{T}} & \Lambda \\ \Lambda^{\mathrm{T}} & 0_{n\times n} \end{bmatrix}$$

of order $m+n$ is the Schur complement of I_{m+n} in the matrix $\tilde{K}$. Then the problem is reduced to computing the LDL^{T} decomposition of $\hat{K}$.

Let $Z = Z_m \oplus Z_n$ be a shift-block matrix, where

$$Z_k = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \in \mathbb{R}^{k\times k}.$$

Then the displacement rank of $\hat{K}$ with respect to Z is 4. Denote by $v_1 = X(1,:)^{\mathrm{T}}$, and $v = v_1/\|v_1\|_2$. Let $y = -Xv$ and $w = [\lambda(n), \lambda(n-1), \ldots, \lambda(1)]^{\mathrm{T}}/\|v_1\|_2$. Then the generators of $\hat{K}$ are defined in the following way:

$$
\begin{aligned}
g_1 &= [y^{\mathrm{T}}, w^{\mathrm{T}}]^{\mathrm{T}},\\
g_2 &= [0, y(2:m)^{\mathrm{T}}, w^{\mathrm{T}}]^{\mathrm{T}},\\
g_3 &= [0, \lambda(m+n-1), \lambda(m+n-2), \ldots, \lambda(n+1), 0.5, 0, \ldots, 0]^{\mathrm{T}},\\
g_4 &= [0, \lambda(m+n-1), \lambda(m+n-2), \ldots, \lambda(n+1), -0.5, 0, \ldots, 0]^{\mathrm{T}},
\end{aligned}
$$

where g_2 and g_3 are positive, g_1 and g_4 are negative.

Since the order of the matrix $\hat{K}$ is $m+n$ the computational cost of the generalized Schur algorithm should be proportional to $(m+n)^2$. In the next section we will show that, exploiting the particular structure of the generators of $\hat{K}$ the computational cost of the generalized Schur algorithm can be reduced to $\mathrm{O}(mn+n^2)$.

We observe that the matrix $\hat{K}$ is indefinite. However, analyzing the generators and the Schur complement of $-XX^{\mathrm{T}}$ in $\hat{K}$ we are able to say a priori that the steps of the algorithm for $i=1,\ldots,m$ are negative, the steps for $i=m+1,\ldots,m+n$, are positive. Hence, taking into account what we already said at the beginning of this section, the diagonal matrix D of the LDL^{T} factorization of $\hat{K}$ is

$$
D = \operatorname{diag}\left(\underbrace{-1,\ldots,-1}_{m}, \underbrace{1,\ldots,1}_{n}\right).
$$

Remark 2.1. We observe that $g_1(1:m)$, $g_2(1:m)$ are the generators for the symmetric negative definite Toeplitz matrix $-XX^{\mathrm{T}}$. If we denote by $\hat{G}_1 = \begin{bmatrix} g_1^{\mathrm{T}}(1:m) \\ g_2^{\mathrm{T}}(1:m) \end{bmatrix}$, the generator matrix for $-XX^{\mathrm{T}}$, by $\hat{G}_i$ and $\hat{f}_i$ the updated generator matrix and the ith column of $\hat{G}_i$, respectively, obtained at the ith step for the computation of the LDL^{T} factorization of $-XX^{\mathrm{T}}$, we have that $\hat{f}_i^{\mathrm{T}} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \hat{f}_i < 0$.

2.1.2. Description of the algorithm

As introduced in Section 2.1, at each step i, we look for a J-orthogonal matrix Θ_i in order to eliminate all elements of f_i, the ith column of G_i with exception of one element. This can be done by choosing J-orthogonal matrices Φ such that $\Phi \begin{bmatrix} f_i(j) \\ f_i(k) \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$. Φ can be either a Givens rotation (updating) if $\{j,k\} \in \{1,4\}$ or $\{j,k\} \in \{2,3\}$, or a hyperbolic rotation (downdating) elsewhere. Proceeding in this way we can eliminate all the entries of f_i with exception of a single pivot element. Since the pivot can arbitrarily be chosen either between the positive generators in case of a positive step or between the negative generators in case of a negative step, we fix the index of the pivot equal to 1 in case of a positive step, equal to 2 otherwise. We perform the downdating by means of a *mixed* hyperbolic rotation [4,19].

We divide the algorithm in four phases:

- 1st phase: step `for` $i = 1$,
- 2nd phase: steps `for` $i = 2,\ldots,m-n$,

- 3rd phase: steps `for` $i = m - n + 1, \ldots, m$,
- 4th phase: steps `for` $i = m + 1, \ldots, m + n$.

2.1.3. 1st phase: step for $i = 1$

$g_1^{(0)}$ is the only vector with the first entry different from zero. Then we set $L(:, 1) = g_1^{(0)}$, $D(1, 1) = -1$, $g_1^{(1)}(2 : m + n) = g_1^{(0)}(1 : m + n - 1)$, $g_1^{(1)}(1) = 0$, $g_1^{(1)}(m + 1) = 0$.

2.1.4. 2nd phase: steps for $i = 2 : m - n$

Before describing this phase we observe that the vectors $g_3^{(i-1)}$ and $g_4^{(i-1)}$ differ only for the $(m + 1)$th entry. Then we will see that the updating of $g_1^{(i-1)}$ with $g_4^{(i-1)}$ and the downdating with $g_3^{(i-1)}$ modifies only the $(m + 1)$th entry of $g_1^{(i-1)}$. Hence $g_1^{(i-1)}(1 : m)$ and $g_2^{(i-1)}(1 : m)$ continue to be the generator vectors at the beginning of the ith step for the LDL^{T} factorization of $-XX^{\mathrm{T}}$. Thus each step of this phase is a negative one since $f_i^{\mathrm{T}} J f_i < 0$. Now we describe how the generators are modified at each step of this phase. We have to update $g_1^{(i-1)}$ with $g_4^{(i-1)}$ and downdate with $g_3^{(i-1)}$. These vectors are

$$g_1^{(i-1)} = \left[\underbrace{0, \ldots, 0}_{i-1}, \xi_i, \ldots, \xi_{n+i}, \underbrace{0, \ldots, 0}_{m-n-i}, \xi_{m+1}, \ldots, \xi_{m+n} \right]^{\mathrm{T}}, \tag{10}$$

$$g_4^{(i-1)} = \left[\underbrace{0, \ldots, 0}_{i-1}, \zeta_i, \ldots, \zeta_m, \zeta_{m+1}, \zeta_{m+2} \ldots, \zeta_{m+n} \right]^{\mathrm{T}},$$

$$g_3^{(i-1)} = \left[\underbrace{0, \ldots, 0}_{i-1}, \zeta_i, \ldots, \zeta_m, \mu_{m+1}, \zeta_{m+2} \ldots, \zeta_{m+n} \right]^{\mathrm{T}}.$$

The Givens rotation used to update $g_1^{(i-1)}$ with $g_4^{(i-1)}$ is

$$G = \begin{bmatrix} c_G^{(i-1)} & s_G^{(i-1)} \\ -s_G^{(i-1)} & c_G^{(i-1)} \end{bmatrix}$$

$$\text{with } c_G^{(i-1)} = \frac{\xi_i}{\sqrt{\xi_i^2 + \zeta_i^2}} \text{ and } s_G^{(i-1)} = \frac{\zeta_i}{\sqrt{\xi_i^2 + \zeta_i^2}}.$$

The updated vectors $\tilde{g}_1^{(i-1)}$ and $\tilde{g}_4^{(i-1)}$ are

$$\tilde{g}_1^{(i-1)} = c_G^{(i-1)} g_1^{(i-1)} + s_G^{(i-1)} g_4^{(i-1)}, \tag{11}$$

$$\tilde{g}_4^{(i-1)} = -s_G^{(i-1)} g_1^{(i-1)} + c_G^{(i-1)} g_4^{(i-1)} \tag{12}$$

with

$$\tilde{g}_1^{(i-1)} = \left[\underbrace{0, \ldots, 0}_{i-1}, \tilde{\xi}_i, \ldots, \tilde{\xi}_{m+n}\right]^{\mathrm{T}} \quad \left(\tilde{\xi}_i = \sqrt{\xi_i^2 + \zeta_i^2}\right).$$

Moreover,

$$\tilde{g}_4^{(i-1)}(n+i+1:m) = c_G^{(i-1)} g_4^{(i-1)}(n+i+1:m) \tag{13}$$

since $g_1^{(i-1)}(j) = 0,\ j = n+i+1, \ldots, m$. Finally, the generators $\tilde{g}_1^{(i-1)}$ with $g_3^{(i-1)}$ are multiplied by the mixed hyperbolic rotation

$$H = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\rho^2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \rho \\ 0 & 1 \end{bmatrix},$$

where ρ is such that $H[\tilde{\xi}_i, \zeta_i]^{\mathrm{T}} = [\hat{\tilde{\xi}}_i, 0]^{\mathrm{T}}$. Taking (11) into account, it is straightforward to see that

$$\rho = -s_G^{(i-1)} \quad \text{and} \quad \sqrt{1-\rho^2} = c_G^{(i-1)}.$$

The downdated vectors $\hat{g}_1^{(i-1)}$ and $\tilde{g}_3^{(i-1)}$ are

$$\hat{g}_1^{(i-1)} = \frac{\tilde{g}_1^{(i-1)} - s_G^{(i-1)} g_3^{(i-1)}}{c_G^{(i-1)}} = g_1^{(i-1)} - s_G^{(i-1)} \frac{g_3^{(i-1)} - g_4^{(i-1)}}{c_G^{(i-1)}} \tag{14}$$

and

$$\tilde{g}_3^{(i-1)} = -s_G^{(i-1)} \hat{g}_1^{(i-1)} + c_G^{(i-1)} g_3^{(i-1)}. \tag{15}$$

Hence,

$$\begin{aligned} \tilde{g}_3^{(i-1)} &= -s_G^{(i-1)} \hat{g}_1^{(i-1)} + c_G^{(i-1)} g_3^{(i-1)} \\ &= -s_G^{(i-1)} g_1^{(i-1)} + \frac{c_G^{(i-1)^2} g_3^{(i-1)} - s_G^{(i-1)^2} (g_4^{(i-1)} - g_3^{(i-1)})}{c_G^{(i-1)}} \\ &= -s_G^{(i-1)} g_1^{(i-1)} + \frac{g_3^{(i-1)} - (1 - c_G^{(i-1)^2}) g_4^{(i-1)}}{c_G^{(i-1)}}. \end{aligned} \tag{16}$$

From (15) and (12), $\tilde{g}_3^{(1)}$ and $\tilde{g}_4^{(1)}$ continue to be equal, except for the $(m+1)$th entry. Furthermore, from (14), we observe that $g_1^{(i-1)}$ and $\hat{g}_1^{(i-1)}$ differ in their $(m+1)$th entry. Since $g_1^{(i-1)}(m+1) = 0$, $\hat{g}_1^{(i-1)}(m+1) = -s_G^{(i-1)}(g_3^{(i-1)}(m+1) - g_4^{(i-1)}(m+1))/c_G^{(i-1)}$. We observe that it is not necessary to compute the whole

vector in (13) since, at the next step, the corresponding entries of $g_1^{(i)}(n+i+2:m+1)$ are equal to 0. We need only to store the partial product

$$c_G^{(i-2)} \cdots c_G^{(2)} c_G^{(1)} \tag{17}$$

into a temporary variable, and multiply $g_4^{(i)}(n+i+1)$ with this variable at the beginning of the ith step.

To finish, the step $\hat{g}_1^{(i-1)}$ has to be downdated with $g_2^{(i-1)}$. This computation does not destroy the structure of the vectors since

$$\hat{g}_1^{(i-1)} = \left[\underbrace{0,\ldots,0}_{i-1},\underbrace{*,\ldots,*,*}_{n+1},\underbrace{0,\ldots,0}_{m-n-i},\underbrace{*,\ldots,*}_{n}\right]^{\mathrm{T}},$$

$$g_2^{(i-1)} = \left[\underbrace{0,\ldots,0}_{i-1},\underbrace{*,\ldots,*}_{n},\underbrace{0,\ldots,0}_{m-n-i+1},\underbrace{*,\ldots,*}_{n}\right]^{\mathrm{T}}.$$

Let $\tilde{H}$ be the stabilized hyperbolic rotation such that

$$\tilde{H}\begin{bmatrix}\hat{g}_1^{(i-1)}(i:m+n)\\ g_2^{(i-1)}(i:m+n)\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}\check{g}_1^{(i-1)}(i:m+n)\\ \check{g}_2^{(i-1)}(i:m+n)\end{bmatrix}^{\mathrm{T}}$$

with $\check{g}_2^{(i-1)}(i)=0$. Then $\check{g}_1^{(i-1)}$ becomes the ith column of L, $D(i,i)=-1$, and, for the next step, the updated vectors are

$$g_1^{(i)} = [0, \hat{g}_1^{(i-1)}(i+1:m+n)],\ g_1^{(i)}(m+1)=0,$$

$$g_2^{(i)} = \check{g}_2^{(i-1)},$$

$$g_4^{(i)} = \tilde{g}_4^{(i-1)},$$

$$g_3^{(i)} = [\tilde{g}_4^{(i-1)}(1:m);\ \gamma;\ \tilde{g}_4^{(i-1)}(m+2:m+n)],$$

where $\gamma = c_G^{(i-1)} g_3^{(i-1)}(m+1) - s_G^{(i-1)} \hat{g}_3^{(i-1)}(m+1)$. The number of flops of this phase is $18mn - 18n^2$.

2.1.5. 3rd phase: steps for $i = m-n+1:m$

The steps of this phase are very similar to those of the previous one. However, we do not need to store the product of the Givens coefficients $c_G^{(i-1)}$ into a temporary variable, since $g_1^{(i-1)}(k) \neq 0,\ k = i,\ldots,m+n$. We recall that $g_3^{(i-1)}$ and $g_4^{(i-1)}$ continue to be equal with exception of the $(m+1)$th entry. Thus $g_1^{(i-1)}(i:m)$ and $g_2^{(i-1)}(i:m)$ are the generator vectors at the ith step of the LDL^{T} factorization

of $-XX^{\mathrm{T}}$. Hence each step of this phase continues to be a negative step ($D(i,i) = -1$). The number of flops of this phase is $13.5n^2$.

2.1.6. 4th phase: steps for $i = m+1 : m+n$

In this phase the vectors $g_3^{(i-1)}(m+1:m+n)$ and $g_4^{(i-1)}(m+1:m+n)$ are different. Now we observe that the vectors $g_i^{(m)}$, $i = 1,\ldots,4$ are the generators for the Schur complement of $-XX^{\mathrm{T}}$ in the matrix $\hat{K}$ [13], that is, the generators for $\Lambda^{\mathrm{T}}(XX^{\mathrm{T}})^{-1}\Lambda$, a symmetric and positive definite matrix. Then each step of this phase is a positive step, meaning that $D(i,i) = 1$ and the ith column of L is $\breve{g}_2^{(i-1)}$. The number of flops of this phase is $9n^2$.

2.2. Stability of LDL^{T} factorization

The stability of the proposed generalized Schur algorithm is studied in [16]. The stability properties of the algorithm for the considered problem depend on the implementation of the hyperbolic rotations.

In [16] it is proved that the following results holds for the LDL^{T} factorization of $\hat{K}$, provided the hyperbolic rotations are implemented in a stable way [4,6].

Theorem 2.2. *Let G be the generator matrix of $\hat{K}$. Let L and D be the matrices of the LDL^{T} factorization of $\hat{K}$ computed by means of the generalized Schur algorithm applying a sequence of Givens rotations and two mixed hyperbolic rotations per step. Then*

$$\|\hat{K} - LDL^{\mathrm{T}}\|_F \leqslant 62(m+n-1)(m+n)\varepsilon\left(2\sqrt{m+n}\|\hat{K}\|_F + \|G_1\|_F^2\right).$$

Hence the proposed algorithm is weakly stable. [4]

2.3. Solution of the linear systems

In this subsection we evaluate the computational cost of the solution of the linear system of equations (9).

Having computed the following factorization of $\tilde{K}$ in $\mathrm{O}(mn+n^2)$ flops,

$$\tilde{K} = \begin{bmatrix} I & & \\ X & I & \\ 0_{n\times(m+n)} & & I \end{bmatrix}\begin{bmatrix} I & \\ & R^{\mathrm{T}} \end{bmatrix}\begin{bmatrix} I & \\ & D \end{bmatrix}\begin{bmatrix} I & \\ & R \end{bmatrix} \times \begin{bmatrix} I & X^{\mathrm{T}} & 0_{(m+n)\times n} \\ & I & \\ & & I \end{bmatrix} = L_1L_2D_1L_2^{\mathrm{T}}L_1^{\mathrm{T}}$$

[4] An algorithm for solving linear equations is *weakly stable* for a class of matrices $\mathscr{A}$ if for each well conditioned $A \in \mathscr{A}$ and for each b the computed solution $\hat{x}$ to $Ax = b$ is such that $\|\hat{x} - x\|/\|x\|$ is small [5].

we need now to solve five linear systems, with coefficient matrices L_1, L_2, D_1, L_2^{T}, L_1^{T}, respectively. The solution of the systems with coefficient matrix L_1 and L_1^{T} can be computed in $\mathrm{O}(mn)$ flops. The solution of the linear system with coefficient matrix D_1 is obtained by changing the sign of the entries $m+n+1,\ldots,2m+n$, of b_1. Furthermore, the solution of the linear systems with coefficient matrix L_2 and L_2^{T} can be computed in $\mathrm{O}(mn+n^2)$ flops since

$$R = \begin{bmatrix} * & * & * & * & & & * & * & * \\ & * & * & * & * & & * & * & * \\ & & * & * & * & * & * & * & * \\ & & & * & * & * & * & * & * \\ & & & & * & * & * & * & * \\ & & & & & * & * & * & * \\ & & & & & & * & * & * \\ & & & & & & & * & * \\ & & & & & & & & * \end{bmatrix},$$

where the first row is

$$R(1,:) = \begin{bmatrix} \underbrace{*,\ldots,*}_{n+1}, \underbrace{0,\ldots,0}_{m-n-1}, \underbrace{*,\ldots,*}_{n} \end{bmatrix}^{\mathrm{T}}.$$

Hence the solution of the linear system (9) has the same computational complexity as the LDL^{T} factorization of $\hat{K}$.

3. Outline of speech compression scheme

Since (3) is a nonlinear optimization problem, the use of good starting values is of utmost importance for convergence within a reasonable amount of time. A method which yields very good starting values in this respect is HTLS [21]. This is a sub-optimal (it does not give the closest fit) subspace based harmonic retrieval method, that approximates the signal s by a sum of n complex damped exponentials. Straightforward calculations based on the parameters of these exponentials yield the initial x and $[\Delta A\ \Delta b]$ in (3). After solving the latter Toeplitz STLS problem, we could analyze each frame by storing the vector x and the first n values of $s+\Delta s$ for that particular frame. Since this procedure will lead to large reconstruction errors at the receiver side, we apply a TLS-ESPRIT-like algorithm, HTLS, to the obtained Toeplitz data matrix $[A+\Delta A\ b+\Delta b]$. Since the corrected data matrix $[A+\Delta A\ b+\Delta b]$ is rank deficient and real, HTLS gives an exact fit and the resulting $2n$ parameters of the complex damped exponentials can be used. The vocoder analysis and synthesis algorithms, applied to the ith frame, can thus be summarized as follows:

Vocoder Analysis Algorithm

Input: ith frame of the speech signal: $s(k),\ k = 1 + (i-1)N, \ldots, iN$, with N the number of samples per frame, n the order of the AR filter
Output: $f_k, d_k, a_k, p_k,\ k = 1, \ldots, n/2$, representing the frequencies, dampings, amplitudes and phases of the complex damped exponentials, satisfying $\sum_{k=1}^{n/2} c_k z_k^j + \overline{c_k}\,\overline{z}_k^j = s(j) + \Delta s(j),\ j = 1 + (i-1)N, \ldots, iN$.

Step 1: Initialize $\Delta s(j),\ j = 1 + (i-1)N, \ldots, iN$ and
$a(l),\ l = 1, \ldots, n$ with the result of HTLS
applied to $s(1 + (i-1)N : iN)$.
Step 2: Solve Toeplitz STLS problem (3)
Step 3: Apply HTLS to
$s(1 + (i-1)N : iN) + \Delta s(1 + (i-1)N : iN)$,
to extract $f_k, d_k, a_k, p_k,\ k = 1, \ldots, n/2$

Vocoder Synthesis Algorithm

Input: $f_k, d_k, a_k, p_k,\ k = 1, \ldots, n/2$, representing the frequencies, dampings, amplitudes and phases of the complex damped exponentials.
Output: $s(1 + (i-1)N : iN) + \Delta s(1 + (i-1)N : iN)$, the rank-deficient speech signal that lies closest to $s(1 + (i-1)N : iN)$ in L_2 norm.

Step 1: $s(j) + \Delta s(j) \leftarrow \sum_{k=1}^{n/2} c_k z_k^j + \overline{c_k}\,\overline{z}_k^j$,
$j = 1 + (i-1)N, \ldots, iN$.

With $c_k = a_k \mathrm{e}^{(\sqrt{-1}p_k)}$, $z_k = \mathrm{e}^{(2\sqrt{-1}\pi f_k + d_k)\Delta t}$, Δt being the sampling interval and $\overline{x}$ indicating the complex conjugate of x.

4. Experimentation testing

In this section we apply the speech compression scheme to a speech signal sampled at 8 kHz, using 8 bits per sample. It contains 14,749 samples (approximately 2 s of speech) and is a phonetically balanced French sentence, uttered by a male speaker. The sentence is an enumeration of geographical places:

Paris, Bordeaux, Le Mans, Saint-Leu, Léon, Loudun

which has the following phonetic transcription (according to the International Phonetic Association's rules [14]):

paʁ i, bɔʁdo, ləm$\tilde{a}$, s$\tilde{\varepsilon}$ lø, le$\tilde{ɔ}$, lud$\tilde{œ}$

In the first subsection the speed improvement of the fast implementation of the kernel problem of the vocoder analysis scheme is demonstrated as well as its dependence on

the problem size. The second subsection compares the exact AR modeling approach to the CELP standard algorithm.

4.1. Computational performance of the vocoder analysis scheme

In this subsection we compare the efficiency of three implementations. The first one is the fast implementation of the STLS2 algorithm as described in Section 2. It will be referred to as STLS2f. We also consider a straightforward implementation of algorithm STLS2, referred to as STLS2s, in which Step 2.1 of algorithm STLS2 is solved by Gaussian elimination with partial pivoting [9]. No use is made of the particular structure of the matrix involved in this system of equations. The third implementation, referred to as STLS1s, is a straightforward implementation of the algorithm STLS1, without any optimization with respect to the structure of the involved matrices. This means that we use a standard GRQ algorithm [5] to solve the equality constrained LS problem in Step 2.1 of Algorithm STLS1. As shown in Section 2, the computational complexity of STLS2f is $O(mn + n^2)$. A more rigid flop count based on the program code, yields a theoretical flop count of $40mn + 71m - 36n - 13n^2 + 90$. The implementations STLS1s and STLS2s obviously [6] have a computational complexity of respectively $O(m^3)$ and $O((m + n)^3)$ per iteration.

To investigate the dependence of the computational cost of the different implementations on the size of the problem, we vary the parameters of the vocoder analysis scheme. The three implementations are applied using the following framelengths and estimates orders:

- framelength = 504, estimated order = 4 (this corresponds to $m = 500$, $n = 4$),
- framelength = 254, estimated order = 4 (this corresponds to $m = 250$, $n = 4$),
- framelength = 508, estimated order = 8 (this corresponds to $m = 500$, $n = 8$).

Per frame, the three implementations STLS1s, STLS2s and STLS2f require the same (but varying) number of iterations. The important number is thus the number of flops per iteration. For this particular problem these numbers are displayed in Table 1. We clearly see the drastically improved computational performance obtained with STLS2f. Also note that the cells in the column of the fast STLS2f implementation are split in two parts: the upper part contains the flop count as obtained by the Matlab function *flops* whereas the lower part (bold number) contains the theoretically obtained flop count (i.e. flop count based on the program code). We see that there is a strong correspondence between both numbers.

[5] As mentioned before, the first step (RQ) of the GRQ algorithm applied to the Toeplitz STLS problem can be made more efficient. However, all structure is lost in the second step (QR), which will dominate the computational cost.

[6] For STLS1s the computational cost is mainly due to the Householder reflections used in the RQ and QR factorizations, whereas for STLS2s the computational cost is that of the Gaussian elimination scheme with partial pivoting.

Table 1
This table shows the increased performance of the implementation STLS2f compared to the straightforward implementations STLS1s and STLS2s, for different problem sizes

Frame length	Estimated order	$m \times n$	STLS1s	STLS2s	STLS2f	$\text{flops}_{\text{STLS1s}}/\text{flops}_{\text{STLS2f}}$	$\text{flops}_{\text{STLS2s}}/\text{flops}_{\text{STLS2f}}$
504	4	500×4	9.624e8	6.916e8	117,854 **116,162**	8166	5868
254	4	250×4	1.262e8	8.968e7	59,104 **57,488**	2135	1517
508	8	500×8	1.002e9	7.080e8	2.056e5 **194,470**	4874	3444

The bold numbers represent the theoretical flop count whereas the other numbers represent the flop count obtained with the Matlab function *flops*. $\text{flops}_{\text{STLS1s}}$, $\text{flops}_{\text{STLS2s}}$ and $\text{flops}_{\text{STLS2f}}$ represent the number of flops per iteration for respectively the implementations STLS1s, STLS2s and STLS2f.

By comparing the number of flops for the different problem sizes we note the following. Going from the second to the first line of the table, m is doubled. As could be expected from the theoretical flop count, the number of flops per iteration for STLS1s and STLS2s is approximately multiplied by 8, whereas the number of flops per iteration for STLS2f is only doubled. Going from the first to the third line of the table, n is doubled and m remains constant. As could be expected from the theoretical flop count estimate, doubling n does not really change the number of flops per iteration for implementations STLS1s and STLS2s. For the fast implementation we see that the number of flops is not really doubled, because of the large linear term $(71m)$ in the theoretical flop count. Theoretically we would expect the flop count to be multiplied by 1.67, which is very close to the factor 1.74 found when using the results obtained by the matlab command *flops*.

4.2. Qualitative performance of the vocoder

In this subsection the STLS speech coding scheme is compared to a standard method namely CELP. For the CELP algorithm, we used a Fortran implementation of the Federal Standard 1016 4800 bps CELP vocoder [10] with a compression ratio $\approx$13.33. For the exact AR modeling approach we use the vocoder algorithm described in Sections 2 (i.e. STLS2f is used) and 3. We set the frame length N to 301, the model order n to 6, yielding a compression ratio [7]

$$\approx \frac{301(\text{samples/frame})}{12(\text{parameters/frame})} \approx 25.$$

To assess the quality of the compressed speech, we use the following segmental signal-to-noise ratio (SNR) definition:

[7] Note that for reasons of simplicity, there is no quantization of the parameters included.

$$\mathrm{SNR}_{\mathrm{seg}} \equiv 10 \log_{10} \frac{1}{F} \sum_{j=1}^{F} \frac{\sum_{i=1}^{p} (s_j(i))^2}{\sum_{i=1}^{p} (s_j(i) - \hat{s}_j(i))^2}, \tag{18}$$

where F represents the number of frames, p is the frame length used for averaging, $s_j = s(1 + (j-1)p : jp)$, $\hat{s}_j = \hat{s}(1 + (j-1)p : jp)$ and $\hat{s}$ represents the synthesized signal. Here p is chosen equal to 60 but the result is rather insensitive with respect to p. For the CELP result, this gives a $\mathrm{SNR}_{\mathrm{seg}} = 12.8$ dB. This value results from a comparison between the highpass filtered input and the nonpostfiltered output (standard CELP applies at the end an adaptive postfilter routine to reduce perceptual coder noise). For the STLS based scheme a $\mathrm{SNR}_{\mathrm{seg}}$ of 13 dB is obtained. Note however that this result should be interpreted with care. On purpose, we used a simple vocoder scheme to illustrate the use of the STLS approach in a vocoder. The result does not mean that the STLS-based vocoder scheme with this tuning of the parameters obtains approximately the same audio-quality as the CELP scheme at twice the compression rate of CELP. The signal obtained with the STLS-based vocoder scheme and these parameter settings yields worse audio-quality than CELP, which is explained by the fact that $\mathrm{SNR}_{\mathrm{seg}}$ is mainly a mathematical measure and does not give a quantification of the audio-quality. Furthermore, quantization [8] of the parameters in the STLS-based vocoder scheme would lead to a further degradation of the $\mathrm{SNR}_{\mathrm{seg}}$ measure. Nevertheless, current research has shown that most of these shortcomings can be solved using a subband scheme.

5. Conclusions

In this paper we presented a fast implementation of the vocoder analysis scheme of a recently proposed speech compression scheme. The approach is based on the application of the method of Lagrange multipliers to the Toeplitz STLS problem that occurs in the vocoder analysis scheme. The kernel problem that needs to be solved in each iteration of the iterative algorithm is the solution of a system of equations. By exploiting the low displacement rank of the involved matrices a fast implementation can be developed. By taking advantage of the sparsity of the corresponding generators we are able to even further improve the computational efficiency. The computational complexity for each iteration is $\mathrm{O}(mn + n^2)$ whereas straightforward implementations have a computational complexity of $\mathrm{O}(m^3)$ or $\mathrm{O}((m+n)^3)$. These numbers are confirmed by applying the different implementations in the proposed speech compression scheme that contains a Toeplitz STLS problem as kernel problem. Furthermore a preliminary qualitative comparison of this compression scheme to a standard method is made.

[8] Quantization means that the parameters that result from the vocoder analysis scheme $(f_k, d_k, a_k, p_k,\ k = 1, \ldots, n/2)$ have to be represented by a finite number of bits before they can be transmitted. Obviously this will lead to so-called quantization errors.

References

[1] A.A. Anda, H. Park, Self-scaling fast rotations for stiff least squares problems, Linear Algebra Appl. 234 (1996).

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide, SIAM, Philadelphia, 1995.

[3] E. Anderson, Z. Bai, J.J. Dongarra, Generalized QR factorization and its applications. Technical Report CS-91-131 (LAPACK Working Note 31), Computer Science Department, University of Tennessee, Knoxville, 1991.

[4] A.W. Bojanczyk, R.P. Brent, P. Van Dooren, F.R. De Hoog, A note on downdating the Cholesky factorization, SIAM J. Sci. Statist. Comput. 1 (1980) 210–220.

[5] J. Bunch, The weak and strong stability of algorithms in numerical linear algebra, Linear Algebra Appl. 88/89 (1987) 49–66.

[6] S. Chandrasekaran, A.H. Sayed, Stabilizing the generalized Schur algorithm, SIAM J. Matrix Anal. Appl. 17 (1996) 950–983.

[7] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, New York, 1987.

[8] P.E. Gill, W. Murray, M.H. Wright, Practical Optimization, Academic Press, London, 1981.

[9] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, 1996.

[10] J.P. Campbell Jr., T.E. Tremain, V.C. Welch, The federal standard 1016 4800 bps celp voice coder, Digital Signal Processing, vol. 1(3), Academic Press, 1991, pp. 145–155.

[11] John R. Deller Jr., John G. Proakis, J.H.L. Hansen, Discrete-Time Processing of Speech Signals, MacMillan Publishing Company, Englewood Cliffs, NJ, 1993.

[12] T. Kailath, Displacement structure and array algorithms, in: T. Kailath, A.H. Sayed (Eds.), Fast Reliable Algorithms for Matrices with Structure, SIAM, Philadelphia, 1999.

[13] T. Kailath, A.H. Sayed, Displacement structure:theory and applications, SIAM Rev. 37 (1995) 297–386.

[14] P. Ladefoged, A Course in Phonetics, Harcourt Brace Jovanovich, New York, 1975.

[15] P. Lemmerling, I. Dologlou, S. Van Huffel, Speech compression based on exact modeling and structured total least norm optimization. in: Proceedings of ICASSP 98, vol. I, Seattle, Washington, USA, May 1998, pp. 353–356.

[16] N. Mastronardi, P. Van Dooren, S. Van Huffel, On the stability of the generalized Schur algorithm. in: Numerical Analysis and Its applications Rousse, Bulgaria, June 2000, Lecture Notes in Computer Science, Springer, number 1988, June 2001, pp. 560–567.

[17] N. Mastronardi, P. Lemmerling, S. Van Huffel, Fast structured total least squares algorithm for solving the basic deconvolution problem, SIAM J. Matrix Anal. Appl. 22 (2) (2000) 533–553.

[18] J.B. Rosen, H. Park, J. Glick, Total least norm formulation and solution for structured problems, SIAM J. Matrix Anal. Appl. 17 (1) (1996) 110–128.

[19] M. Stewart, P. Van Dooren, Stability issues in the factorization of structured matrices, SIAM J. Matrix Anal. Appl. 18 (1997) 104–118.

[20] T.E. Tremain, The government standard linear predictive coding algorithm: Lpc-l0. Speech Technology Magazine, April 1982, pp. 40–49.

[21] S. Van Huffel, H. Chen, C. Decanniere, P. Van Hecke, Algorithm for time-domain NMR data fitting based on total least squares, J. Magn. Reson. A 110 (1994) 228–237.