Online ISSN 1848-3380, Print ISSN 0005-1144
ATKAFF 56(3), 359–366(2015)

Ivo Beroš, Nikica Hlupić, Mario Brčić

Computational Aspects of Efficient Estimation of Harmonically Related Sine-Waves

DOI 10.7305/automatika.2015.12.687
UDK 510.52:[621.391.018.72:543.42]

Original scientific paper

This paper proposes an actual implementation of a well-known method [1] for spectral analysis of signals composed of harmonically related sine waves. The method itself requires computations which carried out directly according to the theoretical formulas do not yield computationally efficient implementation. Thus, utilizing matrix factorizations and mathematical “shortcuts", several algorithms have been developed, which perform computations efficiently and make the method suitable for large-scale applications. Implementation details are clearly explained both theoretically and from computational point of view, and the achieved improvements have been proven by extensive simulations. Particular calculations applied will be equally efficient in all similar problems, which renders the proposed routines into widely useful building blocks.

Key words: signal analysis, algorithm, computation speed

Analiza harmoničkog signala s gledišta računske učinkovitosti. U članku se opisuje nekoliko računski učinkovitih algoritama za primjenu dobro poznate i prihvaćene metode za analizu spektra signala sastavljenog od harmoničkih valova [1]. Metoda zahtijeva proračune čija provedba izravno po formulama koje opisuju teorijsku pozadinu postupka vodi u računski neučinkovite algoritme, stoga se koristeći faktorizacije matrica i neke matematičke “prečice" postupno razvijaju učinkoviti algoritmi primjenjivi i u analizama s velikim brojem uzoraka i sastavnica signala. Poboljšanja su jasno objašnjena i teorijski i s gledišta primjene, a dokazuju se opsežnim simulacijama. Posebni načini izračunavanja pojedinih matematičkih izraza primjenjivi su i u drugim sličnim problemima, što ih čini zanimljivim i važnim u raznim područjima znanosti.

Ključne riječi: analiza signala, algoritam, složenost algoritma

1 INTRODUCTION

Estimation of sine wave parameters form noisy samples, as a basic step in many system identification or signal processing applications, has been in the focus of research interest during the past few decades. Since, under assumption of Gaussian noise, maximum likelihood (ML) principle [2] leads to nonlinear least squares (LS) problem, most of algorithms for computing ML estimates are iterative search routines in their nature [3–5]. As such, they are usually two-stage procedures, the first stage being a coarse initial estimation that provides seed values for the second stage, which iteratively approaches maximum likelihood solution.

It turns out that the critical parameter is frequency because once frequency is known, the subsequent calculation of amplitude and phase is straightforward. Although it is well known that ML estimator of frequency is given by the location of the peak of a periodogram [6], due to iterative procedures the computational effort is prohibitive in many instances. This motivated development of various alternative solutions, usually based on a kind of linearization of the problem [1, 7–14]. The improvement in processing speed, compared to “classical” ML algorithms, can be significant, but it is often the case that the gain is not as much obtained by theoretical results themselves as much by exploiting specific mathematical techniques in an actual algorithm implementation.

In this paper we consider one of methods belonging to this “alternative” group [1], and investigate several algorithms that arise from its theoretical foundation, with the final aim to achieve as fast as possible calculations (program execution). As we show shortly, the proposed algorithms notably outperform the one following directly from the theory, which we call Algorithm 1. Specifically, the complexity of initial algorithm is $\mathcal{O}(N^2h)$, where N is the number of samples and h is the number of signal components, while all improved variants are $\mathcal{O}(Nh^2)$, which can be significant difference when $h \ll N$. Mathematical "shortcuts" used are clearly explained both theoretically and from computational point of view, as well as are extensive simulations and their results. Particular calculations applied in tested algorithms will be equally efficient in all similar problems, like for instance in recent applications [15, 16], which renders these routines into widely useful building blocks.

2 PROBLEM STATEMENT AND SOLUTIONS

2.1 Review of theoretical background of the estimation procedure

As explained in [1], the method under consideration estimates parameters of a multi-tone signal, but it simplifies the problem introducing the constraint that all signal components are harmonically related (i.e., their frequencies are integer multiples of fundamental frequency). Thus, the signal model is

$$s(nT_s) = \sum_{k=-h}^{h} A_k \mathrm{exp}(jkn\omega_0 T_s), \qquad A_{-k} = A_k^*,\ n = 0, 1, \ldots, N-1, \tag{1}$$

where T_s is the sampling period, A_0 is DC offset, A_k are complex harmonic amplitudes ($k = 1, 2, \ldots, h$), and $\omega_0 = 2\pi f_0$ is fundamental frequency.

With unknown signal parameters arranged in vector $\mathbf{\Psi} = [\mathbf{\Theta}^{\mathrm{T}}, f_0]^{\mathrm{T}}$, where $\mathbf{\Theta} = [A_0, \mathrm{real}(A_1), \ldots, \mathrm{real}(A_h), \mathrm{imag}(A_1), \ldots, \mathrm{imag}(A_h)]^{\mathrm{T}}$, their estimates are obtained as solution of nonlinear least squares problem

$$\mathbf{\Psi}_{NLS} = \arg\min_{\mathbf{\Psi}} K_{NLS}(\mathbf{\Psi}) \tag{2}$$

where cost function $K_{NLS}(\mathbf{\Psi})$ is

$$K_{NLS}(\mathbf{\Psi}) = \sum_{n=0}^{N-1} \Big(s_m(nT_s) - \sum_{k=-h}^{h} A_k \mathrm{exp}(jkn\omega_0 T_s) \Big)^2. \tag{3}$$

For known f_0, the cost function (3) is linear in $\mathbf{\Theta}$ and this motivates a two stage estimation. The first stage is estimation of frequency, whereby nonlinear least square problem (2) in $\mathbf{\Psi}$ ($2h+2$ parameters) is reduced to a nonlinear least square estimation of a single parameter, frequency f_0. It is shown in [1] that the cost function in single parameter f_0 is

$$K_{NLS}(f_0) = [\mathbf{P}(f_0)\mathbf{S}_m]^{\mathrm{T}}[\mathbf{P}(f_0)\mathbf{S}_m], \tag{4}$$

where $\mathbf{P}(f_0) = \mathbf{I_N} - \mathbf{H}(f_0)\mathbf{H}(f_0)^+$, $\mathbf{H}(f_0) \in \mathbf{M}_{N,2h+1}$, with entries defined by

$$(\mathbf{H}(f_0))_{i,k+1} = \begin{cases} 1,\ k = 0, \\ 2\cos((i-1)k\omega_0 T_s), \\ \qquad k = 1, \ldots, h, \\ -2\sin((i-1)(k-h)\omega_0 T_s), \\ \qquad k = h+1, \ldots, 2h. \end{cases} \tag{5}$$

and $\mathbf{S}_m = [s_m(0), s_m(T_s), \ldots, s_m((N-1)T_s)]^{\mathrm{T}}$.

Here $\mathbf{I_N}$ is an $N \times N$ identity matrix $\mathbf{H}(f_0)^+$ is Moore-Penrose pseudoinverse of $\mathbf{H}(f_0)$ given by

$$\mathbf{H}(f_0)^+ = \Big(\mathbf{H}(f_0)^{\mathrm{T}}\mathbf{H}(f_0)\Big)^{-1}\mathbf{H}(f_0)^{\mathrm{T}}. \tag{6}$$

Minimization of the cost function (4) is obtained by Gauss-Newton method. Each iteration step has the form

$$[\mathbf{J}(f_0)^T\mathbf{J}(f_0)]\Delta f_0 = -\mathbf{J}(f_0)^T[\mathbf{P}(f_0)\mathbf{S}_m], \tag{7}$$

with $\mathbf{J}(f_0) \in \mathbf{R}^N$, defined by

$$\mathbf{J}(f_0) = -\Big(\mathbf{P}(f_0)\frac{\partial \mathbf{H}(f_0)}{\partial f_0}\mathbf{H}(f_0)^+ + \Big(\mathbf{P}(f_0)\frac{\partial \mathbf{H}(f_0)}{\partial f_0}\mathbf{H}(f_0)^+\Big)^{\mathrm{T}}\Big)\mathbf{S}_m, \tag{8}$$

and $\partial\mathbf{H}(f_0)/\partial f_0 \in \mathbf{M}_{N,2h+1}$, with entries

$$\left(\frac{\partial \mathbf{H}(f_0)}{\partial f_0}\right)_{i+1,k+1} = \begin{cases} 0, \qquad k = 0, \\ -4\pi ikT_s\sin(ik\omega_0 T_s), \\ \qquad k = 1, \ldots, h, \\ -4\pi i(k-h)T_s\cos(i(k-h)\omega_0 T_s), \\ \qquad k = h+1, \ldots, 2h. \end{cases} \tag{9}$$

Once f_0 is found, the second stage is estimation of $\mathbf{\Theta}$, but with f_0 known this is an ordinary linear LS problem with $2h+1$ unknowns whose solution is

$$\mathbf{\Theta}_{LS} = \mathbf{H}(f_0)^+\mathbf{S}_m. \tag{10}$$

Hence, the explained estimation method is an iterative routine summarized in Algorithm 1.

Algorithm 1 Initial algorithm

```
1: $f_0$ = initial (starting) estimate of fundamental frequency.
2: repeat
3:     By Gauss-Newton method (7) find better estimate $f_0^{new}$ of $f_0$.
4:     $f_0 = f_0^{new}$
5: until $f_0$ is "good enough"
6: Calculate vector $\mathbf{\Theta}_{LS}$ for given $f_0$ using (10).
```

The complexity of a single iteration of Algorithm 1 is determined by the step 3. Direct computation according to (4) – (9) is not the best approach because it implies computation of matrix $\mathbf{P}(f_0) \in \mathbf{M}_{N,N}$, and this requires matrix multiplication of $\mathbf{H}(f_0) \in \mathbf{M}_{N,2h+1}$ and $\mathbf{H}(f_0)^+ \in \mathbf{M}_{2h+1,N}$, which is $\mathcal{O}(N^2h)$ (for Oh notation see, for example, [17]). In this paper we utilize a few mathematical "shortcuts" jointly with well-known matrix factorizations to achieve better computational efficiency.

2.2 Algorithms

Theoretical foundation in section 2.1 offers several possibilities for actual algorithm design and now we explain few of them that will be compared. For the sake of brevity, in all formulas we omit parameter f_0 from vectors and matrices (i.e. we write $\mathbf{P}$, instead $\mathbf{P}(f_0)$, etc.).

The elements of matrices $\mathbf{H}$ and $\dfrac{\partial \mathbf{H}}{\partial f_0}$ have the form $\alpha \sin(r\omega_0 T_s)$ or $\alpha \cos(r\omega_0 T_s)$, where $\alpha \in \mathbf{R}$ and $r = 0, 1, \ldots, (N-1)h$, so the first speed-up possibility is to calculate $\sin(r\omega_0 T_s)$ and $\cos(r\omega_0 T_s)$ using standard three-term recurrence relation for sine and cosine [18]:

$$U_r = 2\cos(\omega_0 T_s)U_{r-1} - U_{r-2}, \quad r = 2, 3, \ldots. \qquad (11)$$

For $U_0 = 1$ and $U_1 = \cos(\omega_0 T_s)$, we obtain the sequence (11) is $U_r = \cos(r\omega_0 T_s)$, and for $U_0 = 0$ and $U_1 = \sin(\omega_0 T_s)$, the sequence is $U_r = \sin(r\omega_0 T_s)$.

Next, we observe that vector $\mathbf{J}$ in (8), whose calculation implies several matrix-matrix products, can be represented as the sum of two vectors, $\mathbf{J}_1$ and $\mathbf{J}_2$,

$$\mathbf{J} = -\underbrace{\mathbf{P}\frac{\partial \mathbf{H}}{\partial f_0}\mathbf{H}^+\mathbf{S}_m}_{\mathbf{J}_1} - \underbrace{\left(\mathbf{P}\frac{\partial \mathbf{H}}{\partial f_0}\mathbf{H}^+\right)^{\mathrm{T}}\mathbf{S}_m}_{\mathbf{J}_2}. \qquad (12)$$

Such form, computed from right to the left, replaces all matrix-matrix products in (8) by products of matrices and vectors only. Because single $(n \times m) \cdot (m \times p)$ matrix-matrix product requires more calculations than a few $(n \times m) \cdot (m \times 1)$ matrix-vector products, this will speed up the computation. Furthermore, by definition of $\mathbf{P}$, vector $\mathbf{J}_1$ becomes

$$\begin{aligned}\mathbf{J}_1 &= \mathbf{P}\frac{\partial \mathbf{H}}{\partial f_0}\mathbf{H}^+\mathbf{S}_m = (\mathbf{I}_N - \mathbf{H}\mathbf{H}^+)\frac{\partial \mathbf{H}}{\partial f_0}\mathbf{H}^+\mathbf{S}_m \\ &= \frac{\partial \mathbf{H}}{\partial f_0}\Big(\mathbf{H}^+\mathbf{S}_m\Big) - \mathbf{H}\Big(\mathbf{H}^+\Big(\frac{\partial \mathbf{H}}{\partial f_0}\Big(\mathbf{H}^+\mathbf{S}_m\Big)\Big)\Big) \qquad (13)\end{aligned}$$

Similarly, recalling that $\mathbf{P}$ is symmetric matrix, vector $\mathbf{J}_2$ becomes

$$\begin{aligned}\mathbf{J}_2 &= (\mathbf{H}^+)^{\mathrm{T}}\Big(\frac{\partial \mathbf{H}}{\partial f_0}\Big)^{\mathrm{T}}\mathbf{P}^{\mathrm{T}}\mathbf{S}_m = (\mathbf{H}^+)^{\mathrm{T}}\Big(\frac{\partial \mathbf{H}}{\partial f_0}\Big)^{\mathrm{T}}\mathbf{P}\mathbf{S}_m \\ &= (\mathbf{H}^+)^{\mathrm{T}}\Big(\frac{\partial \mathbf{H}}{\partial f_0}\Big)^{\mathrm{T}}\left(\mathbf{S}_m - \mathbf{H}\left(\mathbf{H}^+\mathbf{S}_m\right)\right). \qquad (14)\end{aligned}$$

In both (13) and (14), parenthesis are used to emphasize that all calculation necessary for a step of Gauss-Newton iteration (7) can be performed as matrix-vector product only. Having (12)–(14) in mind, we are able to write the first algorithm for actual calculation of Gauss-Newton iteration (7). We shall name it AlgorithmMPI, since its most distinctive property is computation of and direct calculations with Moore-Penrose pseudoinverse $\mathbf{H}^+$. For clarity, in the pseudo-codes that follow we introduce vectors $\mathbf{X}_i$ that are just auxiliary vectors used to form matrix-vector product and to make calculations neater.

Algorithm 2 AlgorithmMPI

```
1: Form matrices $\mathbf{H}$ and $\frac{\partial \mathbf{H}}{\partial f_0}$ according to (5) and (9), calculating $\sin(r\omega_0 T_s)$ and $\cos(r\omega_0 T_s)$, $r = 0, \ldots, (N-1)h$ as in (11).
2: Calculate matrix $\mathbf{H}^+$.
3: $\mathbf{X}_1 = \mathbf{H}^+\mathbf{S}_m$,
   $\mathbf{X}_2 = \frac{\partial \mathbf{H}}{\partial f_0}\mathbf{X}_1$,
   $\mathbf{X}_3 = \mathbf{H}^+\mathbf{X}_2$,
   $\mathbf{J}_1 = \mathbf{X}_2 - \mathbf{H}\mathbf{X}_3$.
4: $\mathbf{X}_4 = \mathbf{S}_m - \mathbf{H}\mathbf{X}_1$,
   $\mathbf{X}_5 = \frac{\partial \mathbf{H}}{\partial f_0}^{\mathrm{T}}\mathbf{X}_4$,
   $\mathbf{J}_2 = (\mathbf{H}^+)^{\mathrm{T}}\mathbf{X}_5$.
5: $\mathbf{J} = -\mathbf{J}_1 - \mathbf{J}_2$.
6: calculate new approximation $f_0^{new}$ of fundamental frequency $f_0^{new} = f_0 - \frac{\mathbf{J}^{\mathrm{T}}\mathbf{X}_4}{\mathbf{J}^{\mathrm{T}}\mathbf{J}}$.
```

In AlgorithmMPI the most time consuming operation is calculation of $\mathbf{H}^+$. Moreover, it can occasionally even be numerically unstable, and to improve numerical stability in [1] it is recommended to apply singular value decomposition (SVD) [19].

Algorithm 3 AlgorithmSVD

```
1: The same as in Algorithm MPI.
2: Calculate "economy size" singular value decomposition of $\mathbf{H}$, $\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$.
3: $\mathbf{X}_0 = \mathbf{U}^{\mathrm{T}}\mathbf{S}_m$,
   $\mathbf{X}_1 = \mathbf{V}(\mathbf{\Sigma}^{-1}\mathbf{X}_0)$,
   $\mathbf{X}_2 = \frac{\partial \mathbf{H}}{\partial f_0}\mathbf{X}_1$,
   $\mathbf{X}_3 = \mathbf{U}^{\mathrm{T}}\mathbf{X}_2$,
   $\mathbf{J}_1 = \mathbf{X}_2 - \mathbf{U}\mathbf{X}_3$.
4: $\mathbf{X}_4 = \mathbf{S}_m - \mathbf{U}\mathbf{X}_1$,
   $\mathbf{X}_5 = \frac{\partial \mathbf{H}}{\partial f_0}^{\mathrm{T}}\mathbf{X}_4$,
   $\mathbf{J}_2 = \mathbf{U}(\mathbf{\Sigma}^{-1}(\mathbf{V}^{\mathrm{T}}\mathbf{X}_5))$.
5: The same as in AlgorithmMPI.
6: The same as in AlgorithmMPI.
```

However, SVD and other decompositions can be utilized not only to obtain $\mathbf{H}^+$, but also in subsequent calculations as well. Decomposed according to SVD, matrix $\mathbf{H}$ can be written as $\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{U} \in \mathbf{M}_{N,2h+1}$ with orthonormal columns, and $\mathbf{\Sigma}, \mathbf{V} \in \mathbf{M}_{2h+1}$; $\mathbf{\Sigma}$ is diagonal matrix and $\mathbf{V}$ is orthogonal ("economy size" singular value decomposition in MATLAB). Then $\mathbf{H}^+ = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{\mathrm{T}}$, but more important, we have convenient expression $\mathbf{H}\mathbf{H}^+ = \mathbf{U}\mathbf{U}^\mathrm{T}$ so we can write AlgorithmSVD.

Another possibility is to use QR factorization [19] of matrix $\mathbf{H}$. By QR factorization, matrix $\mathbf{H}$ is written as $\mathbf{H} = \mathbf{Q}\mathbf{R}$ where $\mathbf{Q} \in \mathbf{M}_{N,2h+1}$ with orthonormal columns, and $\mathbf{R} \in \mathbf{M}_{2h+1}$ is upper triangular ("economy size" QR factorization in MATLAB). Having $\mathbf{Q}$ and $\mathbf{R}$, it follows that $\mathbf{H}^+ = \mathbf{R}^{-1}\mathbf{Q}^\mathrm{T}$ and $\mathbf{H}\mathbf{H}^+ = \mathbf{Q}\mathbf{Q}^\mathrm{T}$ so we arrive to AlgorithmQR.

Algorithm 4 AlgorithmQR

```
1: The same as in AlgorithmMPI.
2: Evaluate "economy size" QR factorization of $\mathbf{H}$, $\mathbf{H} = \mathbf{QR}$.
3: $\mathbf{X}_0 = \mathbf{Q}^\mathrm{T}\mathbf{S}_m$,
   $\mathbf{X}_1 = \mathbf{R}^{-1}\mathbf{X}_0$,
   $\mathbf{X}_2 = \frac{\partial \mathbf{H}}{\partial f_0}\mathbf{X}_1$,
   $\mathbf{X}_3 = \mathbf{Q}^\mathrm{T}\mathbf{X}_2$,
   $\mathbf{J}_1 = \mathbf{X}_2 - \mathbf{Q}\mathbf{X}_3$
4: $\mathbf{X}_4 = \mathbf{S}_m - \mathbf{Q}\mathbf{X}_0$,
   $\mathbf{X}_5 = \frac{\partial \mathbf{H}}{\partial f_0}^\mathrm{T}\mathbf{X}_4$,
   $\mathbf{X}_6 = (\mathbf{R}^{-1})^\mathrm{T}\mathbf{X}_5$,
   $\mathbf{J}_2 = \mathbf{Q}\mathbf{X}_6$.
5: The same as in AlgorithmMPI.
6: The same as in AlgorithmMPI.
```

Of course, instead evaluating an expression of the form $\mathbf{y} = \mathbf{R}^{-1}\mathbf{x}$, we can rather solve equation $\mathbf{R}\mathbf{y} = \mathbf{x}$, which is significantly faster because matrix $\mathbf{R}$ is upper triangular.

Even better approach is to avoid direct evaluation of expression of the form $\mathbf{y} = \mathbf{H}^+\mathbf{x} = (\mathbf{H}^\mathrm{T}\mathbf{H})^{-1}\mathbf{H}^\mathrm{T}\mathbf{x}$; rather, we can solve linear system $(\mathbf{H}^\mathrm{T}\mathbf{H})y = \mathbf{H}^\mathrm{T}\mathbf{x}$. In this approach, we first calculate symmetric matrix $\mathbf{H}_p = \mathbf{H}^\mathrm{T}\mathbf{H} \in \mathbf{M}_{2h+1}$ and then make Cholesky decomposition of $\mathbf{H}_p$, $\mathbf{H}_p = \mathbf{U}^\mathrm{T}\mathbf{U}$, where $\mathbf{U}$ is upper triangular matrix. This leads to AlgorithmChol.

In step 4 of AlgorithmChol we make use of the fact that $\mathbf{H}^\mathrm{T}\mathbf{H}$ is symmetric matrix, which yields $(\mathbf{H}^+)^\mathrm{T} = ((\mathbf{H}^\mathrm{T}\mathbf{H})^{-1}\mathbf{H}^\mathrm{T})^\mathrm{T} = \mathbf{H}((\mathbf{H}^\mathrm{T}\mathbf{H})^{-1})^\mathrm{T} = \mathbf{H}(\mathbf{H}^\mathrm{T}\mathbf{H})^{-1}$. Also, as in AlgorithmQR, we exploit triangular structure of matrix $\mathbf{U}$.

An alternative is to obtain Cholesky decomposition of matrix $\mathbf{H}_p$ by QR factorization of matrix $\mathbf{H}$. Namely, let $\mathbf{H} = \mathbf{QR}$ is QR decomposition of matrix $\mathbf{H}$. From $\mathbf{H}_p = \mathbf{H}^\mathrm{T}\mathbf{H} = \mathbf{R}^\mathrm{T}\mathbf{Q}^\mathrm{T}\mathbf{Q}\mathbf{R} = \mathbf{R}^\mathrm{T}\mathbf{R}$ and from the uniqueness of Cholesky decomposition it follows that matrix $\mathbf{U}$ in AlgorithmChol is equal to $\mathbf{R}$. Hence, we can use the same procedure (AlgorithmChol) but with step 2 modified in a way that we evaluate QR decomposition of matrix $\mathbf{H}$ and in the subsequent steps we use $\mathbf{R}$ instead $\mathbf{U}$. In what follows, we call this variant of computations AlgorithmCholQR.

Algorithm 5 AlgorithmChol

```
1: The same as in AlgorithmMPI.
2: Calculate matrix $\mathbf{H}_p = \mathbf{H}^\mathrm{T}\mathbf{H}$ and Cholesky decomposition of $\mathbf{H}_p = \mathbf{U}^\mathrm{T}\mathbf{U}$.
3: $\mathbf{X}_0 = \mathbf{H}^\mathrm{T}\mathbf{S}_m$,
   $\mathbf{X}_1 = \mathbf{U}^{-1}(\mathbf{U}^{-\mathrm{T}}\mathbf{X}_0)$,
   $\mathbf{X}_2 = \frac{\partial \mathbf{H}}{\partial f_0}\mathbf{X}_1$,
   $\mathbf{X}_{3a} = \mathbf{H}^\mathrm{T}\mathbf{X}_2$,
   $\mathbf{X}_3 = \mathbf{U}^{-1}(\mathbf{U}^{-\mathrm{T}}\mathbf{X}_{3a})$,
   $\mathbf{J}_1 = \mathbf{X}_2 - \mathbf{H}\mathbf{X}_3$.
4: $\mathbf{X}_4 = \mathbf{S}_m - \mathbf{H}\mathbf{X}_1$,
   $\mathbf{X}_5 = \frac{\partial \mathbf{H}}{\partial f_0}^\mathrm{T}\mathbf{X}_4$,
   $\mathbf{J}_2 = \mathbf{H}(\mathbf{U}^{-1}(\mathbf{U}^{-\mathrm{T}}\mathbf{X}_5))$.
5: The same as in AlgorithmMPI.
6: The same as in AlgorithmMPI.
```

However, if $\mathbf{H}$ has full rank, then matrix $\mathbf{H}_p$ is symmetric and positive definite, which indicated the usage of another decomposition [19], namely LU factorization $\mathbf{H}_p = \mathbf{L}_1\mathbf{U}_1$. Because of the same "structure" of Cholesky decomposition and LU factorization, we can use LU factorization within the "template" of AlgorithmChol by simply replacing matricies $\mathbf{U}$ and $\mathbf{U}^\mathrm{T}$ with $\mathbf{U}_1$ and $\mathbf{L}_1$, respectively. Detailed analysis reveals that there is no notable difference in performance of these two variants so, for conciseness, in the paper we report and comment results with LU factorization only, calling this routine AlgorithmLU.

3 SIMULATIONS AND ANALYSES

3.1 The design of experiment

Since, in this work, precision and absolute control over every single calculation step are of primary importance, all simulations used were programmed in FORTRAN, with all matrix computations performed by double precision LAPACK routines. This should ensure highest objectivity and reliability of reported results, as well as easy traceability. Specifically, QR decomposition is evaluated by using LAPACK routine `dgeqrf`, which does not form matrix $\mathbf{Q}$ explicitly, and for multiplications with $\mathbf{Q}$ routine `dormqr` is used.

The aim is to determine the computational complexity of the aforementioned algorithms with respect to the sample size and number of signal components, rather than to compare convergence and final results (which are the same, anyway). Therefore, we measure execution times of these routines (using FORTRAN `secnds` function) while changing the appropriate parameters. For our purpose, it is sufficient to have signal composed of at most DC + four components ($h = 1 \ldots 4$) and to vary sample size in the range $n = 2^5 \ldots 2^{10}$.

Because we also wish to determine the components of the total execution time, we construct regression model (explained shortly) and measure execution times of a single iteration, then the time needed for two iterations etc. up to eight iterations. Of course, measurement of time needed for just a few iterations would be practically impossible because it would be too short, so we measure *rep* consecutive executions of the same algorithm with the same preset simulation parameters and obtain the estimate of time needed for single execution as the quotient (*total time*)/*rep*. Hence, the general simulation structure is shown in Algorithm 6.

Algorithm 6 The general simulation structure.

```
1: for number of signal components $h = 1$ to $4$ do
2:    for sample size $n = 2^6$ to $2^{10}$ do
3:       for number of iterations $q = 1$ to $8$ do
4:          for $i = 1$ to $rep$ do
5:             $\mathrm{T_{MPI}(q, n, h)}$ = execution time of $q$ iterations of AlgorithmMPI with given $h$ and $n$
6:             $\mathrm{T_{SVD}(q, n, h)} = \ldots$ AlgorithmSVD
7:             $\mathrm{T_{QR}(q, n, h)} = \ldots$ AlgorithmQR
8:             $\mathrm{T_{LU}(q, n, h)} = \ldots$ AlgorithmLU
9:             $\mathrm{T_{Chol}(q, n, h)} = \ldots$ AlgorithmChol
10:            $\mathrm{T_{CholQR}(q, n, h)} = \ldots$ AlgorithmCholQr
11:         end for
12:         $q$ iterations take $T_{ALG}/rep$ time
13:      end for
14:   end for
15: end for
```

Once we acquire execution times, two main problems are to be dealt with. First of all, we need to distinguish the time spent on memory management (housekeeping) jobs [20] and the time effectively spent on routines under consideration. The housekeeping jobs entirely depend on operating system, computer hardware and translation of program code (compiler properties), so it is not possible to put them under control. Secondly, but related to the first problem, execution times measured on different systems (hardware) or in different program environments (let us say, Matlab, C++ etc.) would be quite misleading because of different low level memory management, thread servicing and other processor tasks that are out of our control and are not observable.

To overcome the second problem, we actually compare ratios of execution times, not their measured values. However, we firstly have to separate housekeeping and usefully spent time, and the solution we apply emanates from a number of preliminary simulations and analysis. In short, it turns out that the total execution time of an algorithm, as a function of number of iterations and with n and h as parameters, can be faithfully modeled linearly as

$$T_{Alg}(q; n, h) = \alpha_{Alg}(n, h) + \beta_{Alg}(n, h) \cdot q, \qquad (15)$$

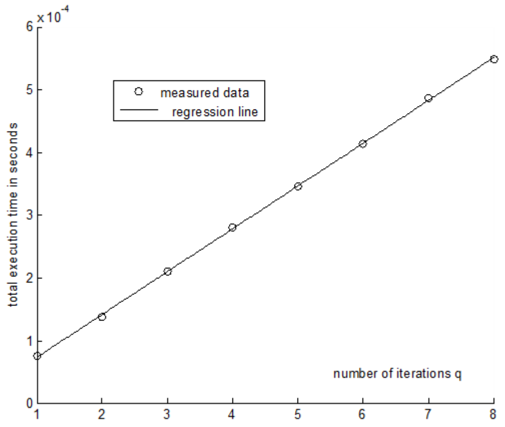


Fig. 1. Plot of total execution time for AlgorithmLU, for $N = 2^8$ and $h = 2$, as a function of number of iterations. This is a typical result because all other tested algorithms show the same behavior.

where $\alpha_{Alg}(n, h)$ is housekeeping contribution for given n and h, and $\beta_{Alg}(n, h)$ is time usefully spent on calculations required by a single iteration of the algorithm *Alg*. A visual support to a linear model is figure 1, which shows a typical simulation result. Correlation coefficients are typically over 0,99.

The linear model is also intuitively most acceptable because housekeeping jobs are in our case mostly related to memory allocation and as such are practically independent on the repetitions of the job that follows, while usefully spent time is, in contrast, directly proportional to the number of iterations. Thus, we are interested in ratios of model parameters β for different algorithms because although execution time of the same algorithm will vary in different environments, mutual relationships among different algorithms (ratios of β) should remain approximately the same.

3.2 Analysis of results

The time we measure depends on two parameters n and h, so we should naturally plot three-dimensional graph to present results of all simulations at once. However, plot of execution times for the six compared algorithms would consist of six close, partly interlaced surfaces and the complete graph would be hardly readable. Therefore, in order to simplify and make analysis clearer, we provide only typical results with one variable fixed. For convenience, at the end we show three-dimensional plot in the whole domain (all pairs (n, h)), but for a single algorithm.

The algorithm based on LU factorization has confirmed itself as the best solution in all instances, so we plot results for all other algorithms with respect to (in ratio with) the results of AlgorithmLU. Figure 2 shows the ratios of exe-

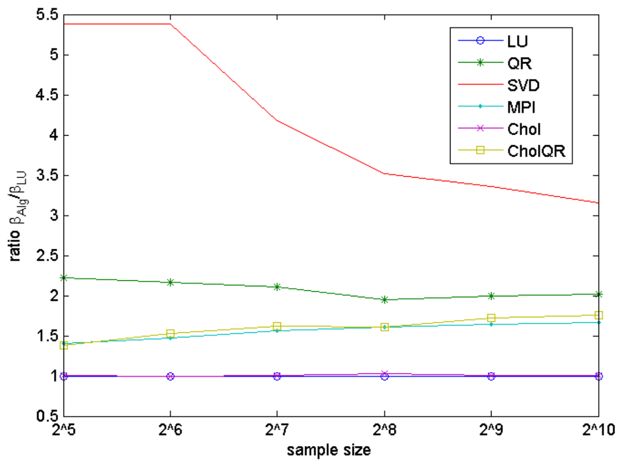


Fig. 2. Ratios of execution times of all algorithms and the time of AlgorithmLU, with the number of signal components fixed to $h = 4$.

cution times with the number of signal components fixed to $h = 4$.

The differences among compared algorithms are notable. AlgorithmLU and AlgorithmChol can be considered equaly fast and they set the lower limit. In the second level are AlgorithmMPI and AlgorithmCholQR, which are about 60% slower than AlgorithmLU (and AlgorithmChol). The AlgorithmQR is just a little slower than AlgorithmMPI, but the one utilizing AlgorithmSVD is convincingly the slowest, slower than AlgorithmLU three to five times. Recalling that SVD based computations are numerically the most stable among all compared ones [19], these results can be interpreted as an obvious confirmation of inherent trade-off between speed and numerical stability of computations.

However, a valuable observation accessible from Figure 2 is indication of limiting performances. Clearly, as the sample size grows, changes in performances for each algorithm get smaller, that is, plots tend to become horizontal lines. Theoretically, all these algorithms have equal worst-case complexity, specifically $\mathcal{O}(Nh^2)$ (see [21]), which is better than complexity of Algorithm 1 (for $N \gg h$). In AlgorithmMPI, AlgorithmLU and AlgorithmChol the complexity is determined by multiplication $\mathbf{H}_p = \mathbf{H}^T\mathbf{H}$, while for AlgorithmQR and AlgorithmCholQR it follows from QR factorization, and from SVD decomposition for AlgorithmSVD. However, their asymptotic [17] limits Θ are not the same and AlgorithmLU will be the fastest for all sample sizes as long as $h \ll N$. In the figure this fact is reflected by different end-points of lines.

Final confirmation of AlgorithmLU (and AlgorithmChol) based algorithm as the best choice regardless of n

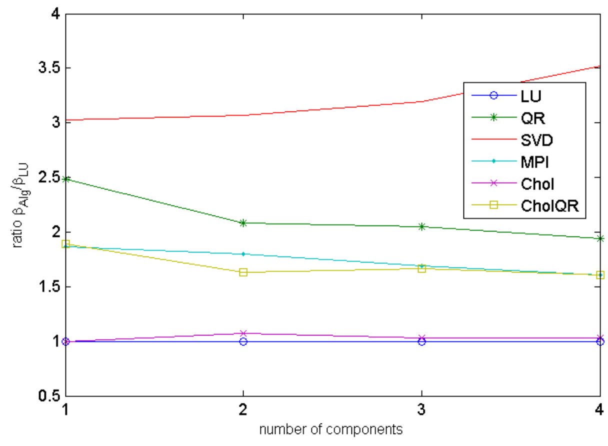


Fig. 3. Ratios of execution times of all algorithms and the time of AlgorithmLU, with sample size fixed to $n = 2^8$.

and h is in Figure 3.

We see that mutual relationships with fixed sample size are identical to those with fixed number of signal components, which confirms AlgorithmLU as the best solution in all setups in this experiment.

As a concluding result, let us consider Figure 4, which shows execution time of AlgorithmLU for all simulated pairs of sample size and number of signal components. The graph has been constructed by taking the shortest time (for $n = 2^5$ and $h = 1$) as a reference and plotting ratios of all other measurements, for particular pairs (n, h), with respect to the referential time. In other words, Figure 4 is plot of ratios $\beta_{LU}(n, h)/\beta_{LU}(2^5, 1)$.

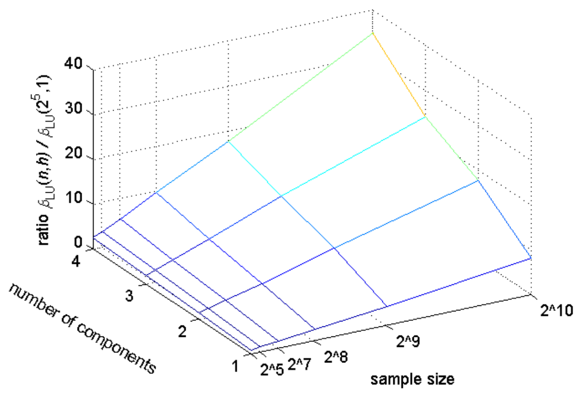


Fig. 4. Execution time of AlgorithmLU as a function of sample size n and number of signal components h. The graph is a plot of ratio of time for particular pair (n, h) to the shortest measured time, i.e., plot of ratios $\beta_{LU}(n, h)/\beta_{LU}(2^5, 1)$.

Not surprisingly, the growth of sample size and the number of signal components quickly increase computation time, but it is interesting to notice precisely the general form of execution time dependence on sample size and its dependence on signal complexity. In Figure 4 we see that the dependence of execution time on the sample size is approximately linear, and on the number of signal components dependence is quadratic, though this is somewhat harder to see in the figure because number of components is maximally four. This can be theoretically proven in a more involved numerical analysis [19].

4 CONCLUSION

In this paper it has been shown, on example of theoretically clear and commonly accepted estimation method, that firm theoretical foundation does not necessarily imply single and obvious algorithm for actual computations. The algorithm that strictly follows the theory (Algorithm 1) is gradually improved and eventually we arrive to several algorithms that notably outperform the Algorithm 1. All these improved algorithms are significantly faster than Algorithm 1 and no one of them can be declared absolutely the best because there is a trade-off between computational speed and numerical stability "inherited" from underlying matrix factorizations. Thus, the analyses in this paper can be a useful source during fine-tuning of an application by choosing the algorithm optimal with respect to the particular problem. Moreover, mathematical shortcuts like performing several matrix-vector products instead of a matrix-matrix product and specific usage of matrix decompositions can be applied in other similar problems, emphasizing the importance of deliberate algorithm design and providing useful guidelines.

REFERENCES

[1] R. Pintelon and J. Schoukens, "An improved sine-wave fitting procedure for characterizing data acquisition channels," *IEEE Transactions on Instrumentation and Measurement*, vol. 45, no. 2, pp. 588–593, 1996.

[2] S. M. Kay, *Fundamentals of statistical signal processing: estimation theory.* Upper Saddle River: Prentice-Hall PTR, 2010.

[3] D. Rife and R. Boorstyn, "Single tone parameter estimation from discrete-time observations," *IEEE Transactions on Information Theory*, vol. 20, no. 5, pp. 591–598, 1974.

[4] D. C. Rife and R. R. Boorstyn, "Multiple tone parameter estimation from discrete-time observations," *Bell System Technical Journal*, vol. 55, no. 9, p. 1389–1410, 1976.

[5] P. Stoica, R. Moses, B. Friedlander, and T. Soderstrom, "Maximum likelihood estimation of the parameters of multiple sinusoids from noisy measurements," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 3, pp. 378–392, 1989.

[6] P. Stoica and R. L. Moses, *Spectral analysis of signals*. Upper Saddle River, N.J.: Pearson/Prentice Hall, 2005.

[7] D. Tufts and R. Kumaresan, "Estimation of frequencies of multiple sinusoids: Making linear prediction perform like maximum likelihood," *Proceedings of the IEEE*, vol. 70, no. 9, pp. 975–989, 1982.

[8] S. Tretter, "Estimating the frequency of a noisy sinusoid by linear regression (corresp.)," *IEEE Transactions on Information Theory*, vol. 31, no. 6, pp. 832–835, 1985.

[9] S. Kay, "A fast and accurate single frequency estimator," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 12, pp. 1987–1990, 1989.

[10] J.-M. Valin, D. V. Smith, C. Montgomery, and T. B. Terriberry, "An iterative linearised solution to the sinusoidal parameter estimation problem," *Computers & Electrical Engineering*, vol. 36, pp. 603–616, July 2010.

[11] Z. Zhang, A. Jakobsson, M. Macleod, and J. Chambers, "On the efficient frequency estimation of a single tone," in *2005 IEEE/SP 13th Workshop on Statistical Signal Processing*, pp. 225–228, 2005.

[12] G. Campobello, G. Cannata, N. Donato, A. Famulari, and S. Serrano, "A novel low-complex and low-memory method for accurate single-tone frequency estimation," in *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pp. 1–6, 2010.

[13] S. Provencher, "Estimation of complex single-tone parameters in the DFT domain," *IEEE Transactions on Signal Processing*, vol. 58, no. 7, pp. 3879–3883, 2010.

[14] A. De Sabata and L. Toma, "Application of bandpass filtering in real tone frequency estimation," in *5th International Symposium on Applied Computational Intelligence and Informatics, 2009. SACI '09*, pp. 429–434, 2009.

[15] I. Orović, S. Stanković, and A. Draganić, "Time-frequency analysis and singular value decomposition applied to the highly multicomponent musical signals," *Acta Acustica united with Acustica*, vol. 100, no. 1, pp. 93–101, 2014.

[16] I. Orović and S. Stanković, "Time-frequency-based speech regions characterization and eigenvalue decomposition applied to speech watermarking," *EURASIP J. Adv. Signal Process*, vol. 2010, p. 4:1–4:10, Feb. 2010.

[17] T. H. Cormen, *Introduction to algorithms*. Cambridge, Mass.: MIT Press, 2001.

[18] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs and mathematical tables*. New York: Dover, 1972.

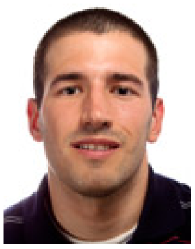
[19] G. H. Golub and C. F. Van Loan, *Matrix computations*. Baltimore: Johns Hopkins University Press, 1996.

[20] W. Stallings, *Operating systems*. Harlow: Prentice Hall, 2009.

[21] L. N. Trefethen, *Numerical linear algebra*. Philadelphia: Society for Industrial and Applied Mathematics, 1997.

Ivo Beroš received M.Sc. degree in 2001 at the Department of Mathematics, Faculty of Science, Zagreb, Croatia. At present, he is a senior lecturer at the VERN' University of Applied Sciences, Zagreb, Croatia. His scientific interests include numerical mathematics, optimization and statistics.

Nikica Hlupić graduated at the Faculty of Electrical Engineering and Computing (FER) in Zagreb, Croatia, in 1993, where he also received M.Sc. degree in 1997 and Ph.D. degree in 2001. At present he is associate professor at the Department of Applied Computing, FER, where he lectures several undergraduate and graduate courses. His scientific interests include estimation theory, optimization, statistics and general theory of algorithm design. He has written a number of scientific and professional publications and he participated in several projects. He is a memeber of IEEE.



Mario Brčić graduated in 2008 from University of Zagreb Faculty of Electrical Engineering and Computing, Croatia where he is, at present, Ph.D. student and works as a research assistant. Teaching activities include undergraduate and graduate courses such as Algorithms and Data Structures and Operational Research. His research interests include optimization, software engineering and artificial general intelligence. He is a member of IEEE, INFORMS and AGI Society.

AUTHORS' ADDRESSES

Ivo Beroš, M.S.
Department of Applied Mathematics and Computer Science
VERN, University of Applied Sciences
Trg bana Josipa Jelačića 3, HR-10000, Zagreb, Croatia
email: ivo.beros@vern.hr

Prof. Nikica Hlupić, Ph.D.
Mario Brčić, B.Sc.
Department of Applied Computing
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
email: {nikica.hlupic, mario.brcic}@fer.hr

Received: 2013-11-19
Accepted: 2015-01-25