Comparison of Orthogonal Matching Pursuit Implementations

Sturm, Bob L.; Christensen, Mads Græsbøll

Published in:
Proceedings of the European Signal Processing Conference

Publication date:
2012

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
COMPARISON OF ORTHOGONAL MATCHING PURSUIT IMPLEMENTATIONS

Bob L. Sturm and Mads Græsbøll Christensen

Dept. of Architecture, Design and Media Technology, Aalborg University, Denmark
E-mail: bst,mgc@create.aau.dk

ABSTRACT
We study the numerical and computational performance of three implementations of orthogonal matching pursuit: one using the QR matrix decomposition, one using the Cholesky matrix decomposition, and one using the matrix inversion lemma. We find that none of these implementations suffer from numerical error accumulation in the inner products or the solution. Furthermore, we empirically compare the computational times of each algorithm over the phase plane.

Index Terms— Orthogonal matching pursuit, algorithms

1. INTRODUCTION
We wish to efficiently model a signal \( \mathbf{u} \) by atoms defined in a dictionary \( \mathcal{D} := \{ \varphi_\omega \in \mathbb{C}^M : \omega \in \Omega := \{1, 2, \ldots, N\} \} \) by

\[
\mathbf{u} = \Phi \mathbf{x} + \mathbf{n} \tag{1}
\]

where \( \Phi := [\varphi_1 | \varphi_2 | \cdots | \varphi_N] \), and \( \mathbf{n} \) is noise. We wish our model \( \hat{\mathbf{x}} \) to have \( s < M \ll N \) non-zero elements (sparsity), such that \( \| \mathbf{u} - \Phi \hat{\mathbf{x}} \| \) is on the order of \( \| \mathbf{n} \| \). This problem is sparse approximation or representation [1, 2], and many algorithms have been designed to solve it [1–3]. In this work, we are concerned only with one: the orthogonal greedy algorithm, orthogonal matching pursuit (OMP) [4]. OMP allows one to directly tune the sparsity or order of the approximation, and its performance is competitive with other more complex algorithms, e.g., convex optimization [5, 6].

OMP is simple and straightforward to implement in a naive manner, and its computational complexity can be reduced for a region of problem dimensions using matrix decomposition, e.g., the QR or Cholesky decomposition [4,7,8]. The numerical behavior of OMP in its different implementations, however, have not been studied. In this paper, we review three computationally efficient implementations of OMP. We are not concerned with guarantees of sparse solutions for OMP, but only with the numerical behavior and computational complexity of each implementation as a function of problem size. Furthermore, we do not consider approximate implementations of OMP, such as gradient pursuit [9], or cyclic matching pursuit [10].

Let \( \Omega_k \subset \Omega \) denote an ordered set of \( k \) indices into \( \mathcal{D} \). We denote the \( i \)th element of a set \( \mathcal{I} \) by \( \mathcal{I}(i) \). The matrix \( \Phi_k \) is composed of the ordered atoms indexed by \( \Omega_k \). We denote \( \mathbf{P}_k := \Phi_k \Phi_k^\dagger \), where we assume \( \text{rank}(\Phi_k) = k \), and thus \( \Phi_k^\dagger := (\Phi_k^\dagger \Phi_k)^{-1} \Phi_k^\dagger \). The projection matrix \( \mathbf{P}_k := \mathbf{I} - \mathbf{P}_k \) is the orthogonal projection onto space orthogonal to the span of \( \mathcal{D}_{\Omega_k} \), or equivalently, the left null space of \( \Phi_k \). We denote the inner product between two vectors in \( \mathbb{C}^M \) as \( \langle \mathbf{u}_1, \mathbf{u}_2 \rangle := \mathbf{u}_1^H \mathbf{u}_2 \), where \( H \) denotes the complex conjugate transpose.

The vector \( \mathbf{r}_k \) is the \( k \)th element of the standard basis of \( \mathbb{R}^N \), i.e., all zeros with a 1 in its \( k \)th row. Finally, unless explicitly noted, all norms are the \( \ell_2 \)-norm, i.e., \( \| \mathbf{x} \|^2 := \langle \mathbf{x}, \mathbf{x} \rangle \).

2. COMPUTATIONAL VARIETIES OF OMP
If in (1) \( \| \mathbf{n} \| = 0 \), OMP attempts to solve

\[
\min_{\mathbf{x}} \| \mathbf{x} \|_0 \text{ subject to } \mathbf{u} = \Phi \mathbf{x} \tag{2}
\]

where \( \| \mathbf{x} \|_0 \) counts the number of non-zeros in \( \mathbf{x} \). When there is noise, OMP attempts to solve

\[
\min_{\mathbf{x}} \| \mathbf{x} \|_0 \text{ subject to } \| \mathbf{u} - \Phi \mathbf{x} \|^2 \leq \epsilon^2 \tag{3}
\]

where \( \epsilon^2 > 0 \). In its \( k \)th iteration, OMP augments \( \Omega_{k-1} \) by \( \Omega_k := \Omega_{k-1} \cup \{n_k\} \) using the selection criterion

\[
n_k := \arg \min_{\beta \in \mathbb{C}, \mathbf{n} \in \Omega} \| \mathbf{r}_{k-1} - \beta \varphi_n \| = \arg \max_{n \in \Omega} \frac{|\langle \mathbf{r}_{k-1}, \varphi_n \rangle|}{\| \varphi_n \|} \tag{4}
\]

OMP then orthogonalizes the residual, \( \mathbf{r}_{k-1} := \mathbf{P}_{k-1}^\perp \mathbf{u} \), and updates the solution, initialized as \( \hat{\mathbf{x}}_0 := 0 \), becomes

\[
[\hat{\mathbf{x}}_k]_{\Omega_k} := \Phi_{k-1}^\dagger \mathbf{u} \tag{5}
\]

where \( [\hat{\mathbf{x}}_k]_{\Omega_k} \) are those elements of \( \hat{\mathbf{x}}_k \) at indices \( \Omega_k \). OMP is initialized \( \Omega_0 := \emptyset \) and \( \hat{\mathbf{x}}_0 := 0 \).

As has been shown before, e.g., [4,7,8], one can make OMP efficient through matrix decomposition. After we review the naive implementation of OMP, we review two approaches using matrix decomposition, and one approach employing the matrix inversion lemma. For our evaluations of algorithmic complexity, we assume that multiplies and adds have the same cost, as done in, e.g., [1,9,11]; but we keep only dominating terms, and do not carry coefficients. Table
The complexity of these implementations are multivariate, big-O analysis of complexity is troublesome [12]; however, we find support from regression analysis for how the parameters affect the computation times we observe. Since we are concerned with practical implementations, we only consider $M < N < \infty$, and assume we can compute and store $\Phi$, and its Gramian $\Phi^H\Phi$.

### 2.1. OMP the Naïve Way

In its $k$th iteration, a naïve implementation of OMP creates and searches through the set

$$I_{k-1} := \left\{ (r_{k-1}, \varphi_n) / \| \varphi_n \| \right\}_{n \in \Omega}$$

which has a complexity of $O(MN)$. After selecting a new atom, OMP orthogonalizes the residual $r_k := \Phi_k^H u = u - \Phi_k (\Phi_k^H \Phi_k)^{-1} \Phi_k^H u$. Assuming that the cost of inverting a complex $k \times k$ matrix is at least $O(k^3)$, this procedure has a complexity of $O(Mk + Mk^2 + k^3)$. We find this from evaluating $[u - (\Phi_k [(\Phi_k^H \Phi_k)^{-1} \Phi_k^H] u)]_{2[3]}$, with subscripts denoting the order of operations. Thus, the $k$th iteration of the naïve OMP implementation has complexity $O(NM + Mk + Mk^2 + k^3)$, which shows that orthogonalization dominates complexity. The memory required for the naïve approach is $O(NM)$ for the dictionary and inner products. While this implementation has a higher computational complexity than the approaches below, it has the lowest memory requirement.

### 2.2. OMP by the Cholesky Decomposition

This implementation is used in several software implementations of OMP, e.g., [13, 14]. Consider that in its $k$th step, OMP has the set of inner products in (6), the set of atom norms $\{ \| \varphi_n \| \}_{n \in \Omega}$, the set of initial inner products

$$I_0 := \left\{ (u, \varphi_n) / \| \varphi_n \| \right\}_{n \in \Omega}$$

(7)

the set of Gramian products for $l \in \Omega$

$$\Gamma_l := \left\{ (\varphi_l, \varphi_n) / \| \varphi_n \| \right\}_{n \in \Omega}$$

(8)

and, finally, the Cholesky decomposition of the Gramian $\Phi_{k-1}^H \Phi_{k-1} = L_{k-1} L_{k-1}^H$. Being positive semidefinite, any Gramian has a Cholesky decomposition.

OMP selects the new atom index $n_k$ from $I_{k-1}$ (6), and updates the Cholesky factorization as follows. Notice

$$L_k L_k^H = \Phi_k^H \Phi_k = \left[ \Phi_{k-1}^H \Phi_{k-1} \Phi_{k-1}^H \Phi_{k-1} \Phi_{k-1}^H \varphi_{n_k} \| \varphi_{n_k} \|^2 \right]$$

$$= \left[ L_{k-1} v^H \ 0 \right] \left[ L_{k-1}^H \ 0 \ b^* \right]$$

(9)

where $\| v \|^2 + | b |^2 = \| \varphi_{n_k} \|^2$. Thus, we first need to solve for $v$ in the triangular system $L_{k-1} v = \Phi_k^H \varphi_{n_k}$ — the right-hand side coming from $\Gamma_{n_k}$ in (8) — and then update the Cholesky factor by adding a row and column, i.e.,

$$L_k := \left[ L_{k-1} \ 0 \right] \left[ v^H \ \sqrt{\| \varphi_{n_k} \|^2 - \| v \|^2} \right].$$

(10)

Since solving a triangular system of size $k$ has complexity $O(k^2)$, the complexity of the search and update is $O(N + k^2)$. Defining the least squares solution $x_k := \Phi_k^H u$, then

$$\Phi_k^H \Phi_k x_k = L_k L_k^H x_k = \Phi_k^H u$$

(11)

where the right-hand side comes from $I_0$. OMP now only needs to solve two triangular systems by back-substitution

$$L_k y = \Phi_k^H u$$

(12)

$$L_k^H x_k = y$$

(13)

with a complexity $O(k^2)$. The new residual energy is then

$$\| r_k \|^2 = \| u \|^2 - \| y \|^2.$$  

(14)

Finally, OMP can update the inner products (6) in one of two ways. For the first (Chol-1), OMP explicitly computes the residual signal $r_k := u - \Phi_k x_k$ with a cost of $O(Mk)$, and then directly computes the inner products $I_k := \{ (r_k, \varphi_n) / \| \varphi_n \| \}_{n \in \Omega}$ with a cost of $O(NM)$. Thus, the total complexity of the $k$th iteration is $O(NM + Mk + k^2)$. In the second case (Chol-2), OMP updates the initial products (7)

$$I_k := \left\{ (r_k, \varphi_n) / \| \varphi_n \| \right\}_{n \in \Omega} = \left\{ I_0(n) - \sum_{l=1}^{k} x_l \Gamma_{n_l}(n) \right\}_{n \in \Omega}$$

(15)

where $x_l$ is the $l$th element of $x_k$. This entire update has a complexity of $O(Nk)$, and thus the total complexity of the $k$th iteration in the second case is $O(Nk + k^2)$. The iteration when the second approach becomes more complex than the first one is $k \approx MN/(N - M)$. In both cases, the memory requirements of OMP is $O(N^2 + NM + k^2 + k)$.

### 2.3. OMP by the QR Decomposition

This implementation is suggested in [8]. Consider that in the $k$th step of OMP we have the set of current inner products (6), initial inner products (7), as well as the QR decomposition

$$http://code.soundsoftware.ac.uk/projects/smallbox
where \( q \) and \( r \) find the solution. Since the least squares solution satisfies \( K \) avoids explicitly computing the solution and residual during the complexity of this approach exceeds QR-1 is \( O \). Thus, the total complexity of the update the residual energy using (14) with the solution of QR decomposition, with \( \Phi \) is the new direction contributed by the atom. OMP updates \( Q_{k-1} \) by adding this unit-norm direction

\[
Q_k := \left[ \begin{array}{c} Q_{k-1} \frac{\varphi_{n_k} - Q_{k-1} w}{\sqrt{\| \varphi_{n_k} \|^2 - \| w \|^2}} \end{array} \right].
\] (16)

Since \( \Phi_k = Q_k R_k = [\Phi_k \varphi_{n_k}] \), we see that OMP can update the other factor by

\[
R_k := \left[ \begin{array}{c} R_{k-1} \frac{w}{\sqrt{\| \varphi_{n_k} \|^2 - \| w \|^2}} \end{array} \right].
\] (17)

With the search, updating these matrices has complexity \( O(N + Mk) \). OMP computes the residual energy recursively

\[
\| r_k \|^2 = \| u - Q_k \varphi_{n_k} \|^2 - \| \Phi_k^H u \|^2 = \| r_{k-1} \|^2 - |\langle u, q_k \rangle|^2
\] (18)

where \( q_k \) is the last column of \( Q_k \).

OMP then updates the projections (6) in one of two ways. First (QR-1), notice OMP need not compute the residual since \( r_k = r_{k-1} - q_k \varphi_{n_k}^H u \). Thus, the updated projections are

\[
I_k := \left\{ \begin{array}{c} (r_k, \varphi_{n_k}) \end{array} \right\} = \left\{ I_{k-1}(n) - \langle u, q_k \rangle \frac{\varphi_{n_k}}{\| \varphi_{n_k} \|} \right\}
\] (19)

for \( \{ n \in \Omega \} \). This has complexity \( O(NM) \). OMP thus avoids explicitly computing the solution and residual during decomposition. Only after the final iteration \( K \) does OMP find the solution. Since the least squares solution satisfies \( \Phi_k^H \Phi_k x_K = \Phi_k^H u \), we see that by substituting the QR decomposition on the left side, OMP finds \( x_K \) by solving

\[
R_k^H y = \Phi_k^H u \quad (20)
\]

\[
R_k x_K = y \quad (21)
\]

where the right hand side of the first system comes from \( I_0 \) (7). Solving these with back-substitution has complexity \( O(K^2) \). In this way, the total complexity of the kth iteration of QR-1 is \( O(NM + Mk) \). Its memory requirements are \( O(NM + Mk + k^2) \) since it need not store the Gramian.

In the second approach (QR-2), OMP updates the projections (6) using (15) after solving (20) and (21). It can also update the residual energy using (14) with the solution of (20). This reduces the complexity at the cost of computing the solution at each iteration and storing the dictionary Gramian (8). Thus, the total complexity of the kth of QR-2 is \( O(Nk + Mk + k^2) \). The cross-over point where the complexity of this approach exceeds QR-1 is \( k(k + N) \approx NM \). Its memory requirements are \( O(N^2 + NM + Mk + k^2) \).

**2.4. OMP by the Matrix Inversion Lemma**

This implementation is suggested in [4]. Consider that in its kth iteration, OMP has the set of current inner products (6), initial inner products (7), the set of Gramian products (8), the current solution \( x_{k-1} \), and the biorthogonal basis of \( \Phi_{k-1} \), i.e., the columns of the matrix

\[
\Psi_{k-1} := \Phi_{k-1} (\Phi_{k-1}^H \Phi_{k-1})^{-1}.
\] (22)

This means that \( \Psi_{k-1}^H \Phi_{k-1} = \Phi_{k-1}^H \Psi_{k-1} = I \). OMP guarantees that \( \Phi_k \) has full column rank when \( k \leq M \), and so \( \Psi_k \) always exists when \( k \leq M \). When it selects the new atom index \( n_k \) from \( I_{k-1} \) (6), OMP must update the biorthogonal basis and solution. Using the matrix inversion lemma, we see

\[
(\Phi_k^H \Phi_k)^{-1} = \left[ (\Phi_{k-1}^H \Phi_{k-1})^{-1} + \lambda \Phi_{k-1}^H \Psi_{k-1} \right]^{-1}
\] (23)

where \( h := \Psi_{k-1} \varphi_{n_k} \), and \( \lambda^{-1} := \| \varphi_{n_k} \|^2 - \| \Phi_{k-1} h \|^2 \).

OMP thus computes the new biorthogonal basis by

\[
\Phi_k := \Phi_k (\Phi_k^H \Phi_k)^{-1} = \left[ \Psi_{k-1} - (\Phi_{k-1}^H \Psi_{k-1}) h \right] / |v|
\] (24)

where \( v := \varphi_{n_k} - \Phi_{k-1} h \). To update the biorthogonal basis then, OMP need only compute \( h \) and \( \lambda \) at a complexity of \( O(N + Mk) \), including the search. It is not necessary until the last step, but OMP can compute the new solution by

\[
x_k = \Psi_k^H u = \begin{bmatrix} x_k \ 0 \ -\Delta \ h \\ I_{k-1} \end{bmatrix}
\] (25)

where

\[
\Delta := \lambda v^H u = \lambda \langle \varphi_{n_k} - \Phi_{k-1} h \rangle^H u
\]

\[
= \lambda \left[ I_0(n_k) - \sum_{l=1}^{k-1} h_l I_0(n_l) \right].
\] (26)

OMP can then update the set of projections for \( \{ n \in \Omega \} \) by

\[
I_k := \left\{ \begin{array}{c} \frac{\langle r, \varphi_{n} \rangle}{\| \varphi_{n} \|} \end{array} \right\} = \left\{ \langle u, \Phi_k x_K, \varphi_{n} / \| \varphi_{n} \| \rangle \right\}
\] (27)

which becomes

\[
I_k := \left\{ I_{k-1}(n) - \Delta G(n_k) + \Delta \sum_{l=1}^{k-1} h_l G(n_l) \right\}. \quad (28)
\]

OMP can also use \( x_k \) in (15) at the same computational cost.

Updating the residual energy is simply done by

\[
| r_k |^2 = \| u - \Phi_k x_k \|^2 = \| u - \Phi_k \left( \begin{bmatrix} x_{k-1} \ 0 \ -\Delta \ h \\ I_{k-1} \end{bmatrix} \right) \|^2
\]

\[
= | r_{k-1} |^2 + |\Delta|^2 / \lambda - 2Re\{\Delta I_{k-1}(n_k)\}
\] (29)

since \( r_{k-1} \) is orthogonal to all selected atoms except the last. In total, the kth iteration of OMP with the matrix inversion lemma has a complexity of \( O(Nk + Mk) \), and memory requirements of \( O(N^2 + MN + Mk) \).
We now test the numerical performance and computational time of the three implementations (QR-1, Chol-2, MIL), and compare with the naive implementation. In every experiment, we sample $\Phi$ from the uniform spherical ensemble, meaning we sample each entry independently from a Normal distribution, and then normalize each column. We test sparse vectors with non-zero entries sampled from either a Rademacher distribution, i.e., equiprobable in $\{-1, 1\}$, or a Normal distribution. We use MATLAB on a 64-bit machine.

In our first experiment, we investigate the accumulation of numerical errors in the recursive computation of the inner products and solution, compared to the values given by the naive implementation. We perform 50 independent trials for signals with sparsity $K = 200$ in ambient dimension $N = 400$, and projected into a space of dimension $M = 400$. In this experiment, $M = N = 400$, but smaller values of $M$ produce the same results. For each iteration, we compute the mean relative error of the inner products, e.g., we compute for the $i$th trial of the Cholesky approach the normalized squared $k$th difference

$$ h_i^{(\text{Chol})}(k) := \frac{\sum_{n=1}^{N} |T_k^{(\text{Chol})}(n) - T_k^{(\text{Naive})}(n)|^2}{\sum_{n=1}^{N} |T_k^{(\text{Naive})}(n)|^2} $$

and find the mean of $\{h_i^{(\text{Chol})}\}$ over several trials.

Figure 1(a) shows how the mean relative error of the inner products slightly increases as a function of iteration in terms of the sparsity of the signal. At the iteration $k/K = 1$, the algorithm has performed as many iterations as there are non-zero elements in the true solution. It is clear that the differences between implementations are inconsequential, but we see a small difference between signal distributions. In Fig. 1(b) we plot the mean relative error in the solutions (between the implementations and not between the true and the solution built by OMP), e.g., for the $i$th trial of the Cholesky approach the normalized squared $k$th difference

$$ e_i^{(\text{Chol})}(k) := \|X_k^{(\text{Chol})} - X_k^{(\text{Naive})}\|^2 / \|X_k^{(\text{Naive})}\|^2 $$

We see little difference between the approaches and the distributions. Furthermore, we see little differences between different sparsities $\rho$ or signal dimension $N$.

In our second experiment, we measure for each implementation the computation time of 10 independent runs of $K$ iterations (sparsity) for $\Phi$ of size $M \times N$. We use as similar code as possible to be fair in the comparisons, and measure times using tic/toc in MATLAB. For two different $N$, Fig. 2 shows $\log_{10}(T_{\text{imp}}(K, M, N) / T_{\text{naive}}(K, M, N))$. For small $N$, $M$ and $K$ the naive approach is the fastest; but as $N$ increases, the other implementations become faster. For small numbers of iterations, the Cholesky implementation appears the fastest; but at larger iterations, the QR approach is faster.

Finally, to evaluate the results in Table 1, we find predictors for each implementation of the mean computation times for $N \in \{100, 200, \ldots, 800\}$, and 15 different $M$ and $K$ (as in Fig. 2). We perform least squares regression with a non-negative constraint solving

$$ \min_w \|\mathbf{T} - [1] \mathbf{X} w\|_2 \text{ subject to } w \succeq 0 $$

where $\mathbf{T}$ is a matrix of measurements $T_{\text{imp}}(K, M, N)$, and $\mathbf{X}$ is a predictor matrix with values from each test, such as $M$, $MK$, and $NK^2$, with each column made to have unit variance. Figure 3 shows the weights of the various terms for Naive, QR-1, Chol-2, and MIL. We see the naive implementation is dominated by $k^2, MNk^2, (kM)^2$ and $MNk^3$. For Chol-2, we see only $NK^2, (NK)^2$ and $k^3$ are the strong predictors. QR-1 is dominated by $MK^2, MNk^2$ and $(kM)^2$. And MIL appears dominated by $k^2$ and $(kM)^2$. The differences between these complexities and those in Table 1 are possibly a result of assuming big-O complexity applies to multivariable processes [12].

---

2Our experiments and the figures in this paper can be reproduced with the code at http://imi.aau.dk/~bst.
4. CONCLUSION

We have investigated three different matrix factorization implementations of OMP, and compared them to the naive implementation. We find that, though their variables are updated recursively, the accumulation of error is insignificant in each implementation. Depending on the problem size, any of the four implementations can be the fastest. However, as the ambient dimension $N$ increases, the computation time of the naive approach becomes much longer than for the other three. For particularly large problem sizes, as the number of iterations increases, the QR implementation appears to be the fastest. Future work will explore to what extents the QR implementation decreases the computational complexity of the Cholesky implementation.

5. REFERENCES