Sparse Solutions of Systems of Equations and Sparse Modeling of Signals and Images

Alfredo Nava-Tudela
ant@umd.edu
John J. Benedetto
Department of Mathematics
jjb@umd.edu

Abstract

We are interested in finding sparse solutions to systems of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is underdetermined and fully-ranked, as presented in [1]. In this report we examine an implementation of the *orthogonal matching pursuit* (OMP) algorithm, an algorithm to find sparse solutions to equations like the one described above, and present a logic for its validation and corresponding validation results.

We also test OMP in the study of the compression properties of $\mathbf{A}$ in the context of image processing. We make a small modification in the stopping criteria of OMP that results in better compression ratio vs image quality as measured by the structural similarity (SSIM) and mean structural similarity (MSSIM) indices.

1 Introduction and context

Let n and m be two positive natural numbers such that $n < m$, and consider a full rank matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$. Given a column vector $\mathbf{b} \in \mathbb{R}^n$, we know that there is an infinite number of solutions to the system of linear equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

where $\mathbf{x}$ is a column vector in $\mathbb{R}^m$ [5]. That is, there is an infinite number of column vectors $\mathbf{x} \in \mathbb{R}^m$ that satisfy equation (1). However, of this infinite number of possible solutions, we are interested in those solutions that are the *sparsest*. By this we mean solutions where $\mathbf{x}$ has the fewest number of non-zero entries. We shall make this concept more precise in section 2.

1.1 Why is this problem of interest?

Finding sparse solutions to systems of linear equations has many signal processing applications, among them, signal compression.

For example, the media encoding standard JPEG [2, 8] and its successor, JPEG-2000 [7], are both based on the notion of transform encoding. The JPEG standard uses the Discrete Cosine Transform (DCT), and the JPEG-2000 standard uses the Discrete Wavelet Transform (DWT). Both JPEG standards use properties of the DCT or the DWT, respectively, to achieve compression by creating approximations that represent the original image in a sparse way. We shall revisit this application as part of the testing protocol of this project.

2 Defining the problem of finding sparse solutions to $\mathbf{Ax} = \mathbf{b}$

Consider a full-rank matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ with $n < m$, and define the underdetermined system of linear equations $\mathbf{Ax} = \mathbf{b}$. From the infinite number of solutions, we shall narrow down the choice to a well-defined solution by introducing a real valued function $\mathcal{J}(\mathbf{x})$ to evaluate the desirability of a would-be solution $\mathbf{x} \in \mathbb{R}^m$, with smaller values of $\mathcal{J}$ being preferred. This way, we can define the general optimization problem $(P_{\mathcal{J}})$ as

$$(P_{\mathcal{J}}): \qquad \min_{\mathbf{x}} \mathcal{J}(\mathbf{x}) \quad \text{subject to} \quad \mathbf{Ax} = \mathbf{b}. \tag{2}$$

Selecting a strictly convex function $\mathcal{J}(\cdot)$ guarantees a unique solution. For example if $\mathcal{J}(\mathbf{x}) = \|\mathbf{x}\|_2^2$, the squared Euclidean norm of $\mathbf{x}$, the problem (P_2) that results from this choice has the unique minimum-norm solution $\hat{\mathbf{x}}$ given by

$$\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{b} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b}.$$

We know that the squared ℓ^2 norm is a measure of energy; we are interested in measures of *sparsity*. As was mentioned before, a vector $\mathbf{x}$ is sparse if there are few nonzero elements in the possible entries in $\mathbf{x}$. As such we shall introduce the ℓ^0 "norm"

$$\|\mathbf{x}\|_0 = \#\{i : x_i \neq 0\}.$$

Thus, if $\|\mathbf{x}\|_0 \ll n$, then $\mathbf{x}$ is sparse.

Consider the problem (P_0) obtained from the general prescription (2) that results from choosing $\mathcal{J}(\mathbf{x}) = \|\mathbf{x}\|_0$, viz.,

$$(P_0): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \mathbf{Ax} = \mathbf{b},$$

or its approximation (P_0^ϵ),

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{Ax} - \mathbf{b}\|_2 < \epsilon. \tag{3}$$

Unfortunately, the discrete and discontinuous nature of the ℓ^0 norm impedes the application of the standard convex analysis ideas that were at the core of the solution of (P_2). Moreover, it has been proven that finding a solution to (P_0^ϵ) is NP-hard [3], p. 228. However, the solution of (P_0^ϵ) can still be obtained by *greedy algorithms* when a sufficiently sparse solution exists. We introduce one such greedy algorithm next.

3 Orthogonal Matching Pursuit

In the first half of this project, we are interested in implementing and validating one of the many greedy algorithms (GAs) that attempt to solve (P_0). The general idea is as follows. Starting from $\mathbf{x}^0 = \mathbf{0}$, a greedy strategy iteratively constructs a k-term approximation $\mathbf{x}^k$ by maintaining a set of active columns—initially empty—and, at each stage, expanding that set by one additional column. The column chosen at each stage maximally reduces the residual ℓ^2 error in approximating $\mathbf{b}$ from the current set of active columns. After constructing an approximation including the new column, the residual error ℓ^2 is evaluated; if it now falls below a specified threshold, the algorithm terminates.

Orthogonal Matching Pursuit (OMP)—a GA for approximating the solution of (P_0):

Task: Approximate the solution of $(P_0) : \min_{\mathbf{x}} \|\mathbf{x}\|_0$ subject to $\mathbf{Ax} = \mathbf{b}$.

Parameters: We are given the matrix $\mathbf{A}$, the vector $\mathbf{b}$, and the threshold ϵ_0.

Initialization: Initialize $k = 0$, and set

- The initial solution $\mathbf{x}^0 = \mathbf{0}$.
- The initial residual $\mathbf{r}^0 = \mathbf{b} - \mathbf{Ax}^0 = \mathbf{b}$.
- The initial solution support $\mathcal{S}^0 = \mathit{Support}\{\mathbf{x}^0\} = \emptyset$.

Main Iteration: Increment k by 1 and perform the following steps:

- **Sweep:** Compute the errors $\epsilon(j) = \min_{z_j} \|z_j \mathbf{a}_j - \mathbf{r}^{k-1}\|_2^2$ for all j using the optimal choice $z_j^* = \mathbf{a}_j^T \mathbf{r}^{k-1} / \|\mathbf{a}_j\|_2^2$.
- **Update Support:** Find a minimizer j_0 of $\epsilon(j)$: $\forall j \notin \mathcal{S}^{k-1},\ \epsilon(j_0) \leq \epsilon(j)$, and update $\mathcal{S}^k = \mathcal{S}^{k-1} \cup \{j_0\}$.
- **Update Provisional Solution:** Compute $\mathbf{x}^k$, the minimizer of $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ subject to $\mathit{Support}\{\mathbf{x}\} = \mathcal{S}^k$.
- **Update Residual:** Compute $\mathbf{r}^k = \mathbf{b} - \mathbf{Ax}^k$.
- **Stopping Rule:** If $\|\mathbf{r}^k\|_2 < \epsilon_0$, stop. Otherwise, apply another iteration.

Output: The proposed solution is $\mathbf{x}^k$ obtained after k iterations.

This algorithm is known in the literature of signal processing by the name *orthogonal matching pursuit* (OMP), and this is the algorithm we have implemented and validated. OMP solves, in essence, (P_0^ϵ) for $\epsilon = \epsilon_0$, a given positive threshold. See (3) above for details.

4 An OMP implementation

For a given matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, if the approximation delivered by OMP has k_0 zeros, the method requires $\mathcal{O}(k_0 mn)$ flops in general; this can be dramatically better than the exhaustive search, which requires $\mathcal{O}(nm^{k_0} k_0^2)$ flops.

4.1 Least-squares approximation by QR decomposition

We would like to make the following observations about the OMP algorithm described in section 3. The step that updates the provisional solution seeks to minimize $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, subject to $Support\{\mathbf{x}\} = \mathcal{S}^k$. This is equivalent to solving the least squares approximation problem $\min_{\tilde{\mathbf{x}}} \|\mathbf{A}^{(k)}\tilde{\mathbf{x}} - \mathbf{b}\|_2^2$ for the matrix $\mathbf{A}^{(k)}$ that results from using only the k active columns of $\mathbf{A}$ defined by $\mathcal{S}^k$, and $\tilde{\mathbf{x}}$ is the vector in $\mathbb{R}^k$ whose i-th entry corresponds to the column of $\mathbf{A}$ that was chosen during the i-th iteration of the main loop. See figure 1.

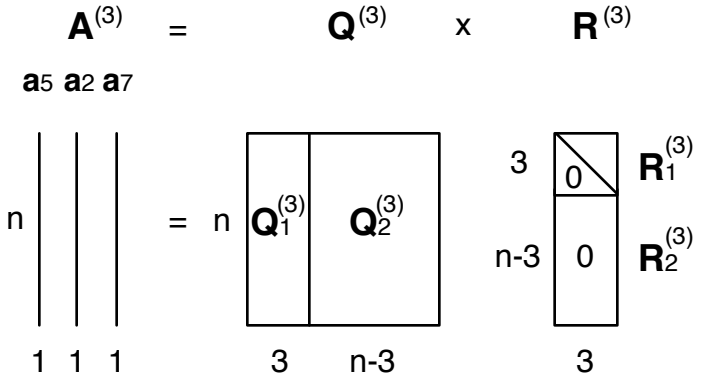


Figure 1: Suppose that after $k = 3$ iterations of the main loop, OMP has chosen, in the following order, columns $\mathbf{a}_5$, $\mathbf{a}_2$, and $\mathbf{a}_7$ from matrix $\mathbf{A}$. We form sub-matrix $\mathbf{A}^{(3)} = (\mathbf{a}_5\ \mathbf{a}_2\ \mathbf{a}_7)$, and its QR decomposition $\mathbf{A}^{(3)} = \mathbf{Q}^{(3)}\mathbf{R}^{(3)}$, which we use to solve the least-squares problem $\|\mathbf{A}^{(3)}\tilde{\mathbf{x}} - \mathbf{b}\|_2^2 = 0$, with $\tilde{\mathbf{x}} \in \mathbb{R}^3$.

For the case when $\mathbf{A}$ is a relatively small matrix, we can solve this problem, for example, by factorizing $\mathbf{A}^{(k)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$ with the QR-algorithm, and then observing that

$$\mathbf{A}^{(k)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)} + \mathbf{Q}_2^{(k)}\mathbf{R}_2^{(k)} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)} + \mathbf{0} = \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}, \tag{4}$$

where—using Matlab notation—

$$\mathbf{Q}_1^{(k)} = \mathbf{Q}^{(k)}(:, 1{:}k),\ \mathbf{Q}_2^{(k)} = \mathbf{Q}^{(k)}(:, k{+}1{:}n),\ \mathbf{R}_1^{(k)} = \mathbf{R}^{(k)}(1{:}k, :),\ \text{and}\ \mathbf{R}_2^{(k)} = \mathbf{R}^{(k)}(k{+}1{:}n, :).$$

Then, from equation (4), we have

$$\begin{aligned}
\mathbf{A}^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{b} &\Leftrightarrow \mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{b} \\
&\Rightarrow \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{Q}_1^{(k)}\mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b} \\
&\Leftrightarrow \mathbf{R}_1^{(k)}\tilde{\mathbf{x}}_0 = \mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b} \\
&\Leftrightarrow \tilde{\mathbf{x}}_0 = (\mathbf{R}_1^{(k)})^{-1}\mathbf{Q}_1^{(k)\mathrm{T}}\mathbf{b},
\end{aligned}$$

where $\tilde{\mathbf{x}}_0 \in \mathbb{R}^k$ is the solution to the equivalent minimization problem described above, and the inverse of $\mathbf{R}_1^{(k)}$ exists because $\mathbf{A}^{(k)}$ is full rank. Finally, when OMP returns successfully after k_0 iterations, we embed $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{k_0}$ in $\mathbf{0} \in \mathbb{R}^m$ "naturally" to obtain the solution $\mathbf{x}_0 \in \mathbb{R}^m$

to the initial least-squares approximation problem $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ subject to the final active column set $\mathcal{S}^{k_0}$. The natural embedding refers to setting the j-th entry of $\mathbf{0} \in \mathbb{R}^m$ equal to the i-th entry in $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{k_0}$ if during the i-th loop of the main algorithm, OMP chose the j-th column of $\mathbf{A}$.

4.2 Implementation fine tuning and speedup

We went through a series of code iterations to speedup our original implementation `ompQR`, initially done from a simplistic reading of the OMP algorithm described in section 3. We also had a generic implementation and a couple of dedicated implementations. In table 1 we show the speedup results for the generic version, which can take any full-rank matrix $\mathbf{A}$ as input. The dedicated implementations exploited the structure of known input matrices used during OMP testing with further speedup gains, as in resorting to the FFT as part of the internal calculations, for example.

Algorithm	Seconds	Speedup
`ompQR`	617.802467	—
`ompQRf`	360.192118	1.715
`ompQRf2`	308.379138	1.168
`ompQRf3`	298.622174	1.032

Table 1: Algorithm performance. The speedup column refers to the speedup from the immediately previous implementation. To compute the total speedup from first to last implementations multiply all speedup values together. Total speedup from `ompQR` to `ompQRf3` is 2.068, which means we doubled the speed of our implementation for the generic matrix version of our code. We used Matlab version R2010b Service Pack 1 to run "experiment.m" which performs many OMP calls on randomized input.

The first improvement came from computing $\|\mathbf{r}^k\||\cos(\theta_j)|$ during the *Sweep* portion of the algorithm. In this case θ_j is the angle between $\mathbf{a}_j$ and the residue $\mathbf{r}_{k-1}$. This number reflects how good an approximation $z_j\mathbf{a}_j$ to the residue is, and it is faster to compute than $\epsilon(j)$. During this step we also kept track of the best approximation to the residue so that during the *Update Support* stage we could more efficiently update $\mathcal{S}^k$ compared to what was done in `ompQR`. Finally, we do the sweep only on the set of columns that have not been added to the support set, resulting in further time gains on the *Sweep* stage whenever $k > 1$. All these changes where incorporated into `ompQRf`.

For the next round of improvements, we stop building $\mathbf{A}_k$ at each iteration as explained in section 4.1. Rather, we initialize $\mathbf{Q} = \mathbf{I}_n$ and $\mathbf{R} = \emptyset$, where $\mathbf{I}_n \in \mathbb{R}^{n\times n}$ is the identity, and for consistency we define $\mathbf{A}_0 = \mathbf{I}_n \cdot \emptyset = \emptyset$. Subsequently, we update $\mathbf{Q}$ and $\mathbf{R}$ each time we add a column vector $\mathbf{a}_j$ of $\mathbf{A}$ in the following way. Suppose that at step $k > 0$ we have a QR decomposition of $\mathbf{A}_{k-1} = \mathbf{QR}$, and that column $\mathbf{a}_{j_k}$ is chosen from the *Update Support* step. Set $\mathbf{w} = (\mathbf{a}_{j_k}^{\mathrm{T}}\mathbf{Q})^{\mathrm{T}}$ and let $\mathbf{H}$ be a Householder reflexion such that $\mathbf{Hw} = \mathbf{v}$, where $\mathbf{v} = (\#, \ldots, \#, 0, \ldots, 0)^{\mathrm{T}}$ has $n - k$ zeros after the first k entries. Then, since $\mathbf{H}^{\mathrm{T}} = \mathbf{H}$,

$\mathbf{H}^2 = \mathbf{I}_n$, and $\mathbf{HR} = \mathbf{R}$, it is easy to see that

$$\begin{aligned}\mathbf{Q}\mathbf{H}^{\mathrm{T}}\big(\mathbf{R}|\mathbf{H}^{\mathrm{T}}\mathbf{w}\big) &= \mathbf{Q}\big(\mathbf{H}^{\mathrm{T}}\mathbf{R}|\mathbf{H}^2\mathbf{w}\big) = \big(\mathbf{Q}\mathbf{R}|\mathbf{Q}\mathbf{w}\big) \\ &= \big(\mathbf{A}_{k-1}|\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\mathbf{a}_{j_k}\big) = \big(\mathbf{A}_{k-1}|\mathbf{a}_{j_k}\big) = \mathbf{A}_k.\end{aligned}$$

Therefore, if we set $\mathbf{Q}' = \mathbf{Q}\mathbf{H}^{\mathrm{T}}$, and $\mathbf{R}' = \big(\mathbf{R}|\mathbf{H}^{\mathrm{T}}\mathbf{w}\big)$, we would have found a QR decomposition of $\mathbf{A}_k = \mathbf{Q}'\mathbf{R}'$ as a function of $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{a}_{j_k}$. The implementation of this update results in faster code compared to the implementation that computes a QR decomposition of $\mathbf{A}_k$ from scratch for each k. This new approach was implemented in `ompQRf2`.

A final time improvement came simply from allocating all required variables as opposed to have them grow dynamically as needed. This was implemented in the final version `ompQRf3`.

5 OMP validation protocol and validation results

In this section we present the validation protocol that we followed to verify the correctness of our OMP implementation.

5.1 Theoretical results that motivate and justify the protocol

The following results provide the foundation for the validation protocol that we adopted. This protocol can be used to validate any OMP implementation.

Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ with $n < m$, we can compute its mutual coherence defined as follows.

Definition 1. *The mutual coherence of a given matrix* $\mathbf{A}$ *is the largest absolute normalized inner product between different columns from* $\mathbf{A}$*. Denoting the k-th column in* $\mathbf{A}$ *by* $\mathbf{a}_k$*, the mutual coherence is given by*

$$\mu(\mathbf{A}) = \max_{1\le k,j\le m,\ k\neq j} \frac{|\mathbf{a}_k^T\mathbf{a}_j|}{\|\mathbf{a}_k\|_2 \cdot \|\mathbf{a}_j\|_2}. \tag{5}$$

The mutual coherence gives us a simple criterion by which we can test when a solution to (1) is the unique sparsest solution available. In what follows, we assume that $\mathbf{A} \in \mathbb{R}^{n\times m}$, $n < m$, and $\mathrm{rank}(\mathbf{A}) = n$.

Lemma 1. *If* $\mathbf{x}$ *solves* $\mathbf{A}\mathbf{x} = \mathbf{b}$*, and* $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$*, then* $\mathbf{x}$ *is the sparsest solution. That is, if* $\mathbf{y} \neq \mathbf{x}$ *also solves the equation, then* $\|\mathbf{x}\|_0 < \|\mathbf{y}\|_0$*.*

This same criterion can be used to test when OMP will find the sparsest solution.

Lemma 2. *For a system of linear equations* $\mathbf{A}\mathbf{x} = \mathbf{b}$*, if a solution* $\mathbf{x}$ *exists obeying* $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$*, then an OMP run with threshold parameter* $\epsilon_0 = 0$ *is guaranteed to find* $\mathbf{x}$ *exactly.*

The proofs of these lemmas can be found or are inspired by results in [1]. In light of these lemmas, we can envision the following roadmap to validate an implementation of OMP. We have a simple unified theoretical criterion to guarantee both solution uniqueness and OMP convergence. The following theorem simply unifies the previous lemmas into one statement.

Theorem 3. *If* $\mathbf{x}$ *is a solution to* $\mathbf{Ax} = \mathbf{b}$*, and* $\|\mathbf{x}\|_0 < \frac{1}{2}\big(1 + \mu(\mathbf{A})\big)$*, then* $\mathbf{x}$ *is the unique sparsest solution to* $\mathbf{Ax} = \mathbf{b}$*, and OMP will find it.*

In light of this result, we can establish the following protocol to validate any implementation of OMP.

5.2 Validation protocol

Given a full-rank matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, with $n < m$, compute $\mu(\mathbf{A})$, and find the largest integer k smaller than or equal to $\frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$. That is, $k = \big\lfloor \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big) \big\rfloor$.

Then, build a vector $\mathbf{x}$ with exactly k non-zero entries and produce a right hand side vector $\mathbf{b} = \mathbf{Ax}$. This way, you have a known sparsest solution $\mathbf{x}$ to which to compare the output of any OMP implementation.

Pass $\mathbf{A}$, $\mathbf{b}$, and ϵ_0 to OMP to produce a solution vector $\mathbf{x}_{OMP} = \text{OMP}(\mathbf{A}, \mathbf{b}, \epsilon_0)$.

If OMP terminates after k iterations (or less), and $\|\mathbf{Ax}_{OMP} - \mathbf{b}\| < \epsilon_0$, for all possible $\mathbf{x}$ and $\epsilon_0 > 0$, then the OMP implementation would have been validated.

5.3 Validation results

Call $\kappa_{\mathbf{A}} = \frac{1}{2}\big(1 + 1/\mu(\mathbf{A})\big)$, the constant dependent on $\mathbf{A}$ that guarantees the results of Theorem 3 for matrix $\mathbf{A}$. To test our implementation, we ran two experiments involving two random matrices.

1. $\mathbf{A}_1 \in \mathbb{R}^{100 \times 200}$, with entries in the Gaussian distribution $N(0, 1)$, i.i.d., for which its mutual coherence turned out to be $\mu(\mathbf{A}_1) = 0.3713$, corresponding to $k = 1 = \lfloor \kappa_{\mathbf{A}_1} \rfloor$.

2. $\mathbf{A}_2 \in \mathbb{R}^{200 \times 400}$, with entries in the Gaussian distribution $N(0, 1)$, i.i.d., for which its mutual coherence turned out to be $\mu(\mathbf{A}_2) = 0.3064$, corresponding to $k = 2 = \lfloor \kappa_{\mathbf{A}_2} \rfloor$.

We first note that, with probability 1, $\mathbf{A}_i$, $(i = 1, 2)$, is a full-rank matrix [1]. Second, we would like to mention that for full-rank matrices $\mathbf{A}$ of size $n \times m$, the mutual coherence satisfies $\mu(\mathbf{A}) \geq \sqrt{(m - n)/(n \cdot (m - 1))}$, with the equality being sharp [6]. We used these results to guide us into obtaining matrix $\mathbf{A}_2$ for which $k = 2 = \lfloor \kappa_{\mathbf{A}_2} \rfloor > 1$.

For each matrix $\mathbf{A}_i$, $(i = 1, 2)$, we chose 100 compatible vectors with k non-zero entries whose positions were chosen at random, and whose entries were in the Gaussian distribution $N(0, 1)$, i.i.d..

Then, for each such vector $\mathbf{x}$, we built a corresponding right hand side vector $\mathbf{b} = \mathbf{A}_i\mathbf{x}$. Each of these vectors would then be the unique sparsest solution to $\mathbf{A}_i\mathbf{x} = \mathbf{b}$, and OMP should be able to find them.

Finally, given $\epsilon_0 > 0$, if our implementation of OMP were correct, it should stop after k steps (or less), and if $\mathbf{x}_{OMP} = \text{OMP}(\mathbf{A}_i, \mathbf{b}, \epsilon_0)$, then $\|\mathbf{b} - \mathbf{A}_i\mathbf{x}_{OMP}\|_2 < \epsilon_0$.

We ran these experiments for twelve values of ϵ_0 equal to 10, 1, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, 10^{-14}, 10^{-15}, and 10^{-16}. For each of these values of ϵ_0 we built 100 vectors as described above, with their respective right hand side vectors, both of which were fed to OMP together with the tolerance ϵ_0 being tested.

We kept track of how many iterations it took OMP to stop, and the value of the norm of the residue $\|\mathbf{b} - \mathbf{A}_i\mathbf{x}_{OMP}\|_2$ at the end of each run. We mention that our implementation

of OMP had as stopping condition that either the residue would be less than the tolerance ϵ_0 given, or that n iterations of the main loop would have been executed.

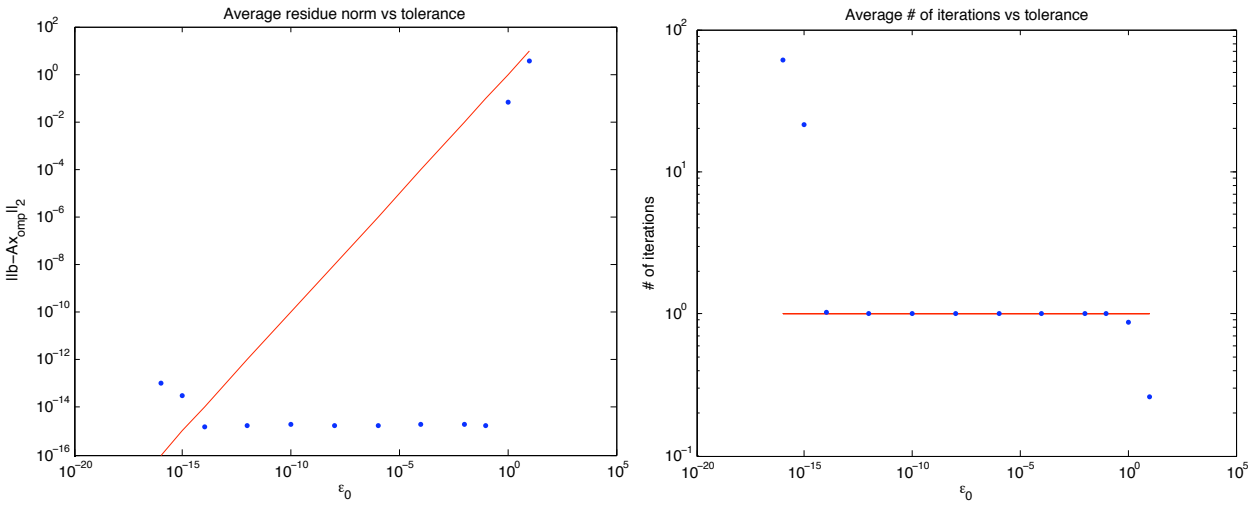


Figure 2: OMP behavior for a matrix $\mathbf{A}$ with $\mu(\mathbf{A}) = 0.3713$, which corresponds to $k_0 = 1$.

Figure 2 shows the summary of the results for matrix $\mathbf{A}_1$. It contains two graphs, the top graph represents the average of the norm of the residue over the 100 experiments executed for a given tolerance, versus the 12 tolerances chosen. The red line represents the identity in this case. The second graph is the same but for the average number of iterations it took OMP to stop vs the tolerances chosen. The red line in this case is the expected number of iterations $k = \lfloor \kappa_{\mathbf{A}_1} \rfloor$ at stop time. Figure 3 is the same as figure 2, but for matrix $\mathbf{A}_2$.

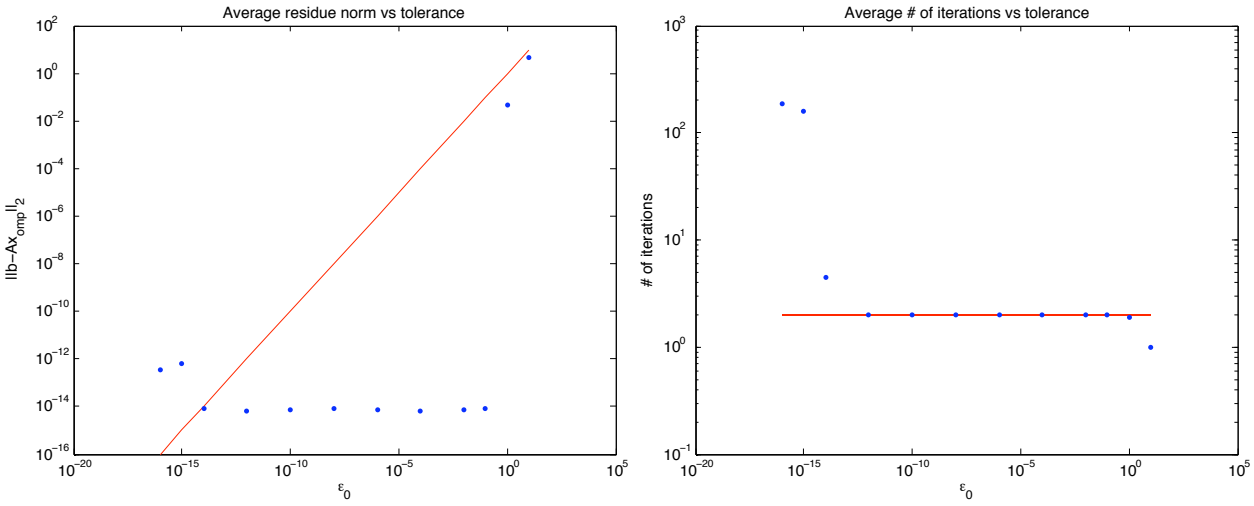


Figure 3: OMP behavior for a matrix $\mathbf{A}$ with $\mu(\mathbf{A}) = 0.3064$, which corresponds to $k_0 = 2$.

One can observe in both cases that there are three modal behaviors of OMP. The rightmost points in each graph correspond to tolerances ϵ_0 that are "too large". For them, OMP converges, but it does not have to do much work necessarily, since the default initial solution $\mathbf{x} = \mathbf{0}$ is already close to the right hand side $\mathbf{b}$. The typical behavior corresponds to points

in the middle of the graph, they represent the cases when OMP converges in exactly k iterations to the sparsest solution within machine precision. And, finally, the leftmost points, they represent when OMP fails to converge because the tolerances ϵ_0 are too close to machine precision, basically trampling OMP efforts to converge due to roundoff and truncation errors.

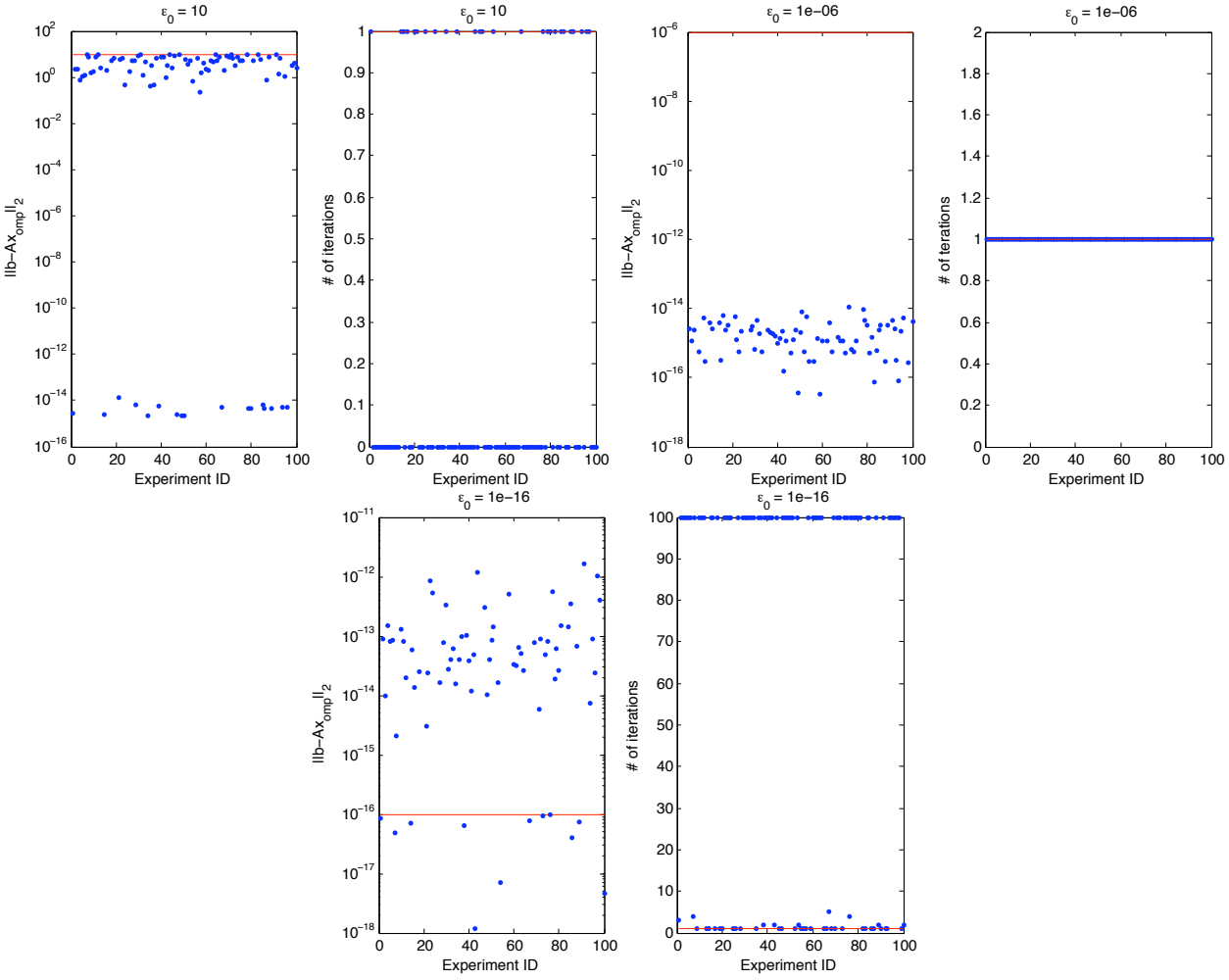


Figure 4: The three modal behaviors, dependent on ϵ_0, observed for the matrix $\mathbf{A}$ used in Fig. 2.

In figure 4 we exemplified each of the three modal behaviors with three values of ϵ_0 typical of each mode. The figure contains three graphs, the top graph is for $\epsilon_0 = 10$, the middle graph is for $\epsilon_0 = 10^{-6}$, and the bottom graph is for $\epsilon_0 = 10^{-16}$. Each of the graphs shows the individual results for each of the 100 experiments ran for each tolerance ϵ_0. This figure corresponds to matrix $\mathbf{A}_1$. In figure 5 we have the same graphs but for matrix $\mathbf{A}_2$.

We can conclude then that the validation protocol and results confirm that our implementation of OMP is correct. This implementation will return a solution $\mathbf{x} = \text{OMP}(\mathbf{A}, \mathbf{b}, \epsilon_0)$ to $\mathbf{Ax} = \mathbf{b}$, within machine precision, whenever the tolerance $\epsilon_0 \geq 10^{-14}$, and provided $\|\mathbf{x}\|_0 \leq \kappa_{\mathbf{A}}$.

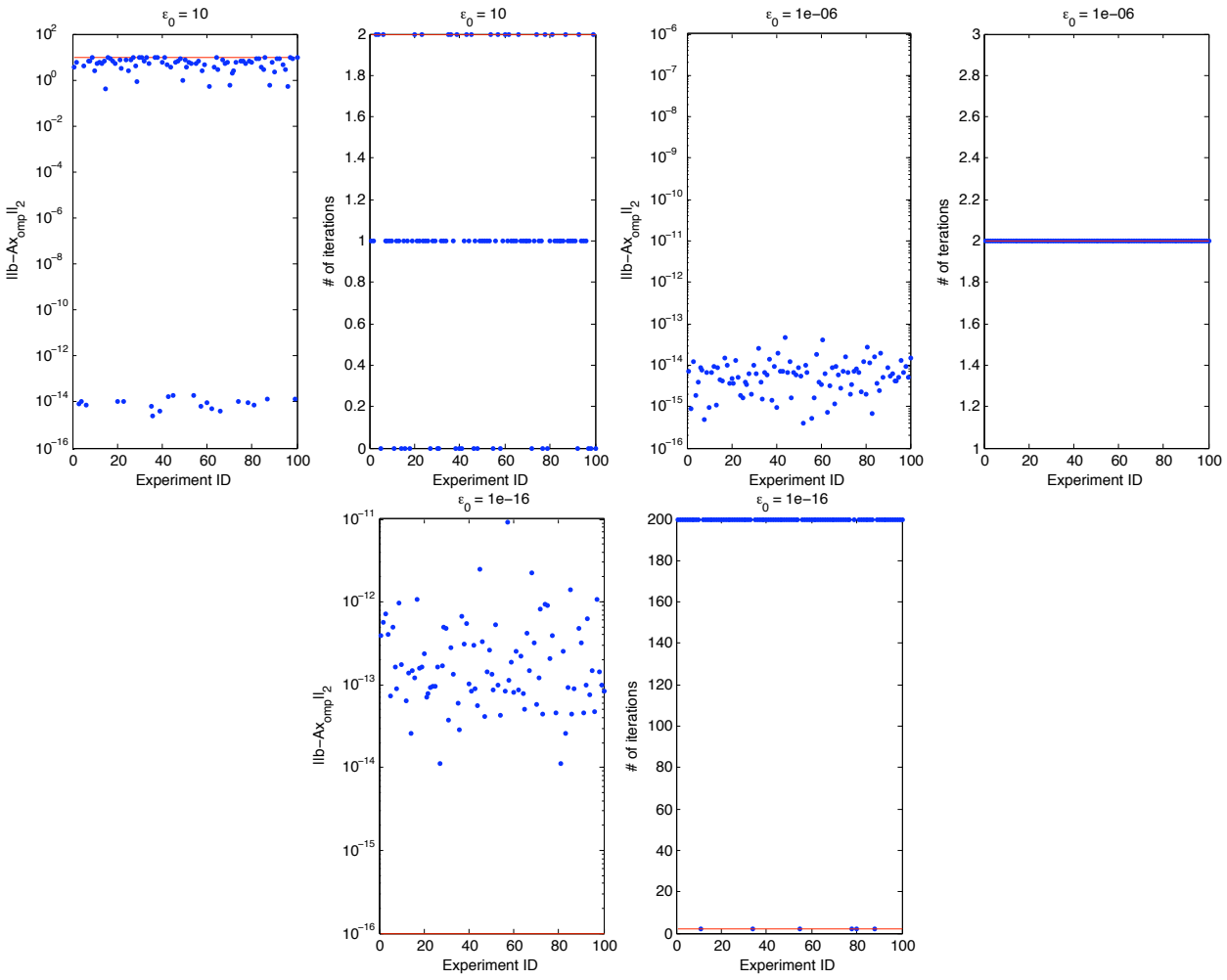


Figure 5: The three modal behaviors, dependent on ϵ_0, observed for the matrix $\mathbf{A}$ used in Fig. 3.

6 OMP testing protocol and results

For the first part of our testing protocol, we set to reproduce a portion of an experiment described in [1]. The second part deals with studying the image compression properties of a particular matrix $\mathbf{A}$ described in more detail below.

6.1 Reproducing previous results

In a SIAM Review article by A. M. Bruckstein, D. L. Donoho, and M. Elad [1], the following experiment is presented. We set to reproduce the portion corresponding to OMP.

Consider a random matrix $\mathbf{A}$ of size 100×200, with entries independently drawn at random from a Gaussian distribution of zero mean and unit variance, $\mathcal{N}(0, 1)$. It can be proven that, with probability 1, every solution for the system $\mathbf{Ax} = \mathbf{b}$ with less than 51 entries is necessarily the sparsest one possible, and, as such, it is the solution of (P_0). By randomly generating such sufficiently sparse vectors $\mathbf{x}$ (choosing the nonzero locations uniformly over the support in random and their values from $\mathcal{N}(0, 1)$), we generate vectors $\mathbf{b}$. This way, we know the sparsest solution to $\mathbf{Ax} = \mathbf{b}$, and we shall be able to compare this to the results given by OMP.

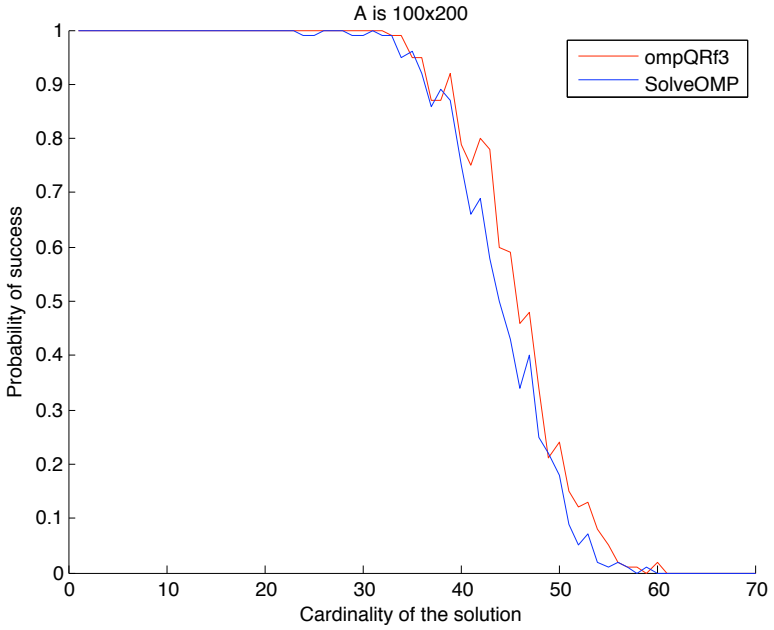


Figure 6: Reproduction of results in section 3.3.1 of [1] for OMP. Our implementation `ompQRf3` is slightly better at recovering with higher probability the sparsest solution to $\mathbf{Ax} = \mathbf{b}$ when compared to `SolveOMP`, an implementation publicly available at [4].

Since we set to reproduce the results that pertain to OMP in Figure 2 of page 56 of [1], we considered cardinalities in the range of 1 to 70—even though we knew that, with probability 1, only those solutions with cardinality equal to or less than 51 were uniquely the sparsest ones possible—, and we conducted 100 repetitions and averaged the results to obtain the

probability of the algorithm finding the solution with which we had generated the right hand side **b**. Comparing our results with results obtained by published OMP implementations, e.g., like the ones available at SparseLab [4], figure 6 shows that our implementation of OMP reproduces the published experiment, and it performs slightly better than the software found in [4].

6.2 Image compression

In the second part of our test suite, we explore image compression via sparsity. The basic idea is that if $\mathbf{Ax} = \mathbf{b}$, $\mathbf{b}$ is dense, and $\mathbf{x}$ is sparse, we can achieve compression by storing wisely $\mathbf{x}$ instead of $\mathbf{b}$.

In specific, suppose we have a signal $\mathbf{y} \in \mathbb{R}^n$ that usually requires a description by n numbers. However, suppose that we can solve equation (3), which we reproduce below,

$$(P_0^\epsilon): \qquad \min_{\mathbf{x}} \|\mathbf{x}\|_0 \quad \text{subject to} \quad \|\mathbf{Ax} - \mathbf{y}\|_2 < \epsilon,$$

and the solution $\mathbf{x}_0^\epsilon$ has k non-zeros, with $k \ll n$, then we would have obtained an approximation $\hat{\mathbf{y}} = \mathbf{Ax}_\mathbf{0}^\epsilon$ to $\mathbf{y}$ using k scalars, with an approximation error of at most ϵ. By increasing ϵ we obtain stronger compression with larger approximation error, and in this way we can obtain a compression ratio vs tolerance curve for this compression mechanism.

In our case, we consider the matrix $\mathbf{A} = [\text{DCT Haar}]$ that results from concatenating two basis. On the one hand we have the Discrete Cosine Transform (DCT) waveforms basis, and on the other hand the basis generated by the identity plus the Haar wavelet waveforms (Haar). In figure 7 we can see the first six representatives of each basis for $\mathbb{R}^{64}$, respectively.

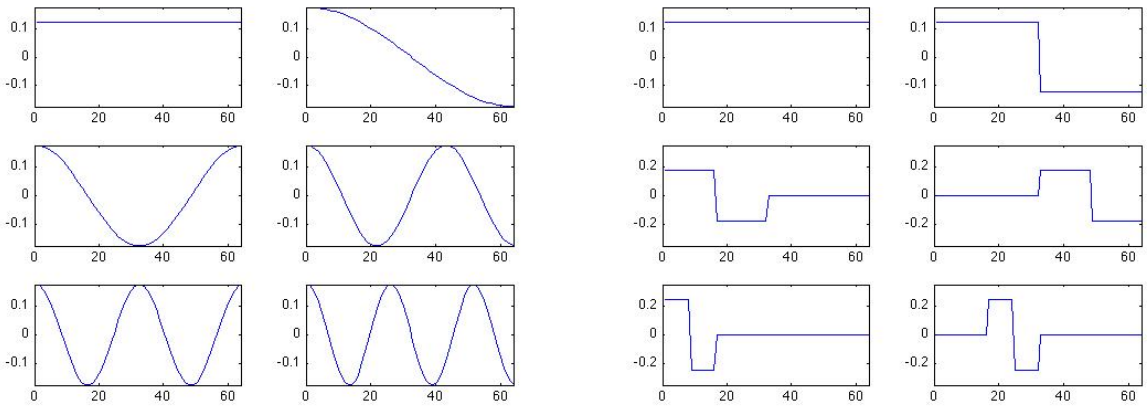


Figure 7: The first six waveforms of the DCT (left) and Haar (right) bases.

6.2.1 Image database

We select 5 natural images to test our compression algorithms. They are shown in figure 8. All images are 512 by 512, 8-bit grayscale images, which means they are composed of $512^2 = 262{,}144$ pixels that can take integer values from 0 (black) to 255 (white).

(a) Barbara

(b) Boat

(c) Elaine

(d) Lena

(e) Peppers

Figure 8: Images used for our compression algorithms based on sparse image representation.

6.2.2 Methodology

Following the approach to image processing at the core of the JPEG image compression standard [8], we subdivide each image in our database in 8 by 8 non-overlapping squares that will be treated individually. A sub-image $\mathbf{Y} \in \mathbb{R}^{8\times 8}$ of size 8 by 8 pixels can be linearized into a vector $\mathbf{y} \in \mathbb{R}^{64}$. There are many ways to do this, we tested two possible solutions. The first one consisted of concatenating one after the other the columns of $\mathbf{Y}$, we shall call this method c_1, which can be thought of as a bijection $c_1 : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ that maps $\mathbf{Y} \mapsto \mathbf{y}$ the way it was described above.

Yet another method would be to flip the ordering of the entries of every even column of $\mathbf{Y}$ and then concatenate its columns. That is, from $\mathbf{Y}$ first obtain the matrix $\mathbf{Y}'$, where $Y'_{i,2j} = Y_{8+1-i,2j}$, and $Y'_{:,:}$ and $Y_{:,:}$ are the entries of $\mathbf{Y}'$ and $\mathbf{Y}$, respectively. Then concatenate one after the other the columns of $\mathbf{Y}'$. Call this method c_2, also a bijection $c_2 : \mathbb{R}^{8\times 8} \to \mathbb{R}^{64}$ that maps $\mathbf{Y} \mapsto \mathbf{y}'$ the way just described.

After a series of tests, we chose c_2 over c_1 as it produces better results. This could be because on average, for a natural image, $|Y_{8,j} - Y_{8,j+1}| < |Y_{8,j} - Y_{1,j+1}|$ which makes $\mathbf{y}' = c_2(\mathbf{Y})$ change more slowly than $\mathbf{y} = c_1(\mathbf{Y})$. This translates into needing less Fourier coefficients of the DCT to describe, within a certain error ϵ, the signal $\mathbf{y}'$ compared to those needed for the signal $\mathbf{y}$.

We proceed in the following way. Given a tolerance $\epsilon_0 > 0$, and an image $\mathcal{I}$ that has been partitioned in 8 by 8 non-overlapping sub-images, say $\{\mathbf{Y}_l\}$ where $l = 1, \ldots, (512/8)^2$ for the images in our database, we obtain the approximation to $\mathbf{y}_l = c2(\mathbf{Y}_l)$ derived from the OMP algorithm, i.e., from the sparse $\mathbf{x}_l = \texttt{OMP}([\text{Dct Haar}], \mathbf{y}_l, \epsilon_0)$, compute $\tilde{\mathbf{y}}_l = [\text{DCT Haar}]\mathbf{x}_l$.

We know that $\|\tilde{\mathbf{y}}_l - \mathbf{y}_l\|_2 < \epsilon_0$, and we can count how many non-zero entries there are in $\mathbf{x}_l$, say $nnz_{\mathbf{x}_l}$. With this, we can define the *compression ratio*.

Definition 2 (Compression Ratio)**.** *Given the context above, the compression ratio for image $\mathcal{I}$—given compression matrix $\mathbf{A} = [DCT\ Haar]$, and tolerance ϵ_0—is the number*

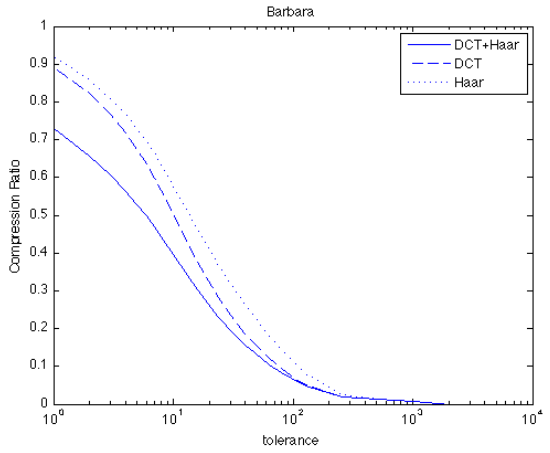
$$cr(\mathcal{I}, \mathbf{A}, \epsilon_0) = \frac{\sum_l nnz_{\mathbf{x}_l}}{S},$$

where S is the total number of pixels in image $\mathcal{I}$ ($S = 512^2$ in our case).

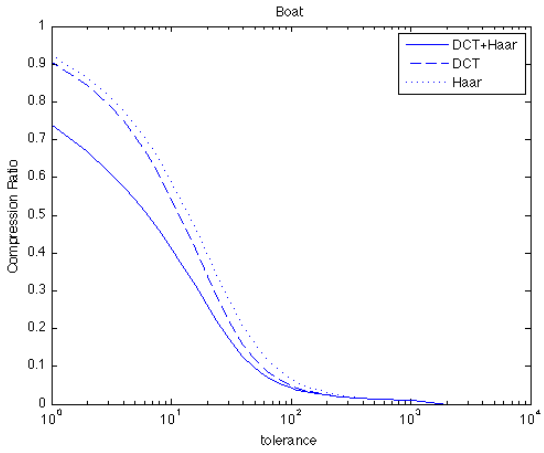
With this definition in hand, we can now proceed to plot compression ratio vs tolerance graphs—one of the main goals of the project—, which will allow us to gauge the compression properties of various compression matrices.

6.2.3 Compression ratio vs tolerance graphs

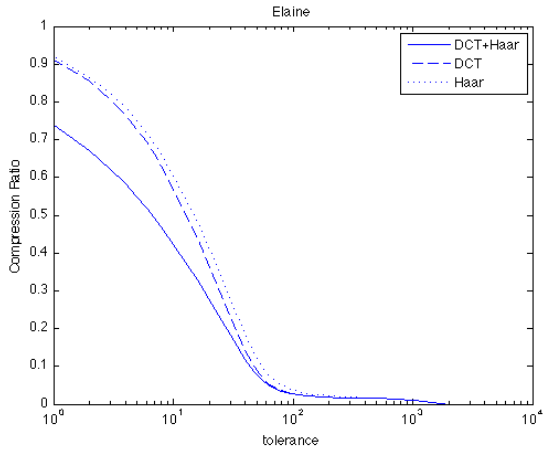
Since we are interested in finding out the compression properties of $\mathbf{A} = [\text{DCT Haar}]$, and compare them to those of $\mathbf{B}$ and $\mathbf{C}$ which use only the DCT or the Haar basis, respectively, we plot for all images in our database their respective compression ratio vs tolerance graphs. We took $\epsilon_0 \in T = \{2048, 1024, 512, 256, 128, 112, 96, 80, 64, 56, 48, 40, 32, 24, 16, 8, 7, 6, 5, 4, 3, 2, 1\}$ and for each image $\mathcal{I}$ in our image database we obtained the corresponding compression ratios $cr(\mathcal{I}, \mathbf{A}, \epsilon_0)$, $cr(\mathcal{I}, \mathbf{B}, \epsilon_0)$, and $cr(\mathcal{I}, \mathbf{C}, \epsilon_0)$ to obtain the following graphs.



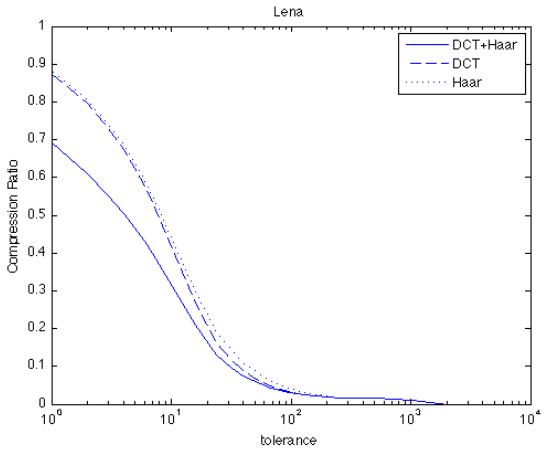
(a) Barbara



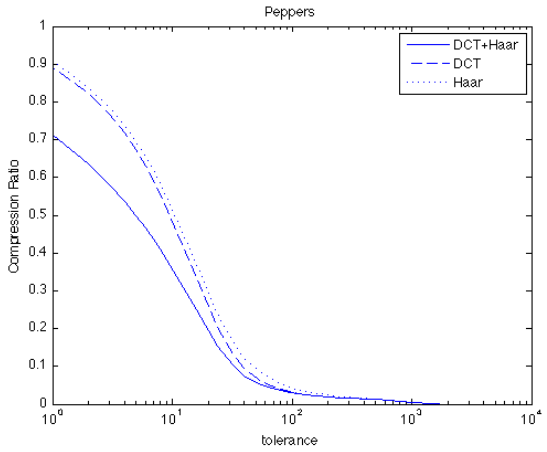
(b) Boat



(c) Elaine



(d) Lena



(e) Peppers

Figure 9: Compression ratio vs tolerance. We can observe that for all images the best compression ratio for a given tolerance is obtained for matrix $\mathbf{A}$ = [DCT Haar] which combines both bases. Also, $\mathbf{B}$ = [DCT] outperforms $\mathbf{C}$ = [Haar].

From the results shown in figure 9, we can see that combining both the DCT and Haar bases results in better compression than if either basis is used alone. This is very encouraging. However, what can be said of the quality of the reconstructed images? For what range of the tolerances tested is the image quality acceptable? What does an "acceptable" image quality mean? To answer these and related questions, we need to address the issues of image reconstruction and error estimation.

6.2.4 Image reconstruction and error estimation

Given a 512×512 image $\mathcal{I}$ in our database, we can proceed to compress it using the methodology described in section 6.2.2. If $\mathcal{I}$ is broken down in 8×8 non-overlapping sub-images $\{\mathbf{Y}_l\}_{l=1,\ldots,4096}$, we can obtain for each of them a corresponding reconstructed sub-image $\widetilde{\mathbf{Y}}_l = c2^{-1}(\tilde{\mathbf{y}}_l)$, and from those reconstruct an approximation $\widetilde{\mathcal{I}}$ to $\mathcal{I}$. Here, $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$, $\mathbf{x}_l = \texttt{OMP}(\mathbf{A}, \mathbf{y}_l, \epsilon_0)$, and $\mathbf{y}_l = c2(\mathbf{Y}_l)$, as before. The compression would come from storing wisely $\{\mathbf{x}_l\}$. We can summarize this procedure with the following notation: $\widetilde{\mathcal{I}} = rec(\mathcal{I}, \mathbf{A}, \epsilon_0)$.

How do we assess the quality of $\widetilde{\mathcal{I}}$ when compared to the original image $\mathcal{I}$? We introduce two error estimators—three in actuality, but two of them are related.

Traditionally, the signal processing community has relied on the *peak signal-to-noise ratio*, or *PSNR* [11].

Definition 3 (PSNR)**.** *The Peak Signal-to-Noise Ratio between two images $\widetilde{\mathcal{I}}$ and $\mathcal{I}$ is the quantity, measured in dB,*

$$\mathrm{PSNR}(\widetilde{\mathcal{I}}, \mathcal{I}) = 20 \log_{10} \left(\frac{\max_{\mathcal{I}}}{\sqrt{\mathrm{mse}(\widetilde{\mathcal{I}}, \mathcal{I})}} \right),$$

where $\max_{\mathcal{I}}$ is the maximum possible value for any given pixel in $\mathcal{I}$, $\max_{\mathcal{I}} = 255$ in this case; and $\mathrm{mse}(\widetilde{\mathcal{I}}, \mathcal{I}) = \frac{1}{nm} \sum_{i,j} \left(\widetilde{\mathcal{I}}(i,j) - \mathcal{I}(i,j) \right)^2$ is the mean square error between both images. Here n and m represent the dimensions of $\mathcal{I}$, $n = m = 512$ in our case, and $\mathcal{I}(i,j)$ represents the value of the pixel at coordinates (i,j) in image $\mathcal{I}$. Similarly for $\widetilde{\mathcal{I}}(i,j)$.

PSNR has the advantage that it is easy to compute and has widespread use, but it has been criticized for poorly correlating with perceived image quality [9, 10], but in recent years extensive work on other error estimators that take into account the human visual system have arisen. In particular, we present and define the *structural similarity* and *mean structural similarity* indices [9].

Definition 4 (SSIM)**.** *Let $\widetilde{\mathcal{I}}$ and $\mathcal{I}$ be two images that have been decomposed in $L \times L$ non-overlapping sub-images $\{\widetilde{\mathbf{Y}}_l\}$ and $\{\mathbf{Y}_l\}$, respectively. Then the Structural Similarity index for two corresponding sub-image linearizations, say $\tilde{\mathbf{y}}_l = c2(\widetilde{\mathbf{Y}}_l)$ and $\mathbf{y}_l = c2(\mathbf{Y}_l)$, is defined as follows*

$$\mathrm{SSIM}(\tilde{\mathbf{y}}_l, \mathbf{y}_l) = \frac{(2\mu_{\tilde{\mathbf{y}}_l}\mu_{\mathbf{y}_l} + C_1)(2\sigma_{\tilde{\mathbf{y}}_l\mathbf{y}_l} + C_2)}{(\mu^2_{\tilde{\mathbf{y}}_l} + \mu^2_{\mathbf{y}_l} + C_1)(\sigma^2_{\tilde{\mathbf{y}}_l} + \sigma^2_{\mathbf{y}_l} + C_2)},$$

where $\mu_{\mathbf{y}_l}$ and $\sigma_{\mathbf{y}_l}$ represent the mean and standard deviation of $\mathbf{y}_l$, respectively; and similarly for $\widetilde{\mathbf{y}}_l$. The term $\sigma_{\tilde{\mathbf{y}}_l\mathbf{y}_l}$ is the correlation between $\tilde{\mathbf{y}}_l$ and $\mathbf{y}_l$. The values C_1 and C_2 are two small constants.

For our purposes, we used the default values of $L = 11$, $C_1 = 0.01$, and $C_2 = 0.03$ used in [9] when assessing the SSIM of an image in our database and its reconstruction. We used a value of $L = 4$ when we modified OMP to use internally the SSIM as a stopping criteria. More on this later.

From the above definition, we can see that the SSIM index is a localized quality measure that can be represented on an plane that maps its values. It can takes values from 0 to 1 and when it takes the value of 1 the two images are identical. In practice, we usually require a single overall quality of measure for the entire image. In that case we use the mean SSIM index to evaluate the overall image quality.

Definition 5 (MSSIM)**.** *Let $\widetilde{\mathcal{I}}$ and $\mathcal{I}$ be two images, where the former is the approximation and the later is the original. Then the Mean Structural Similarity index is*

$$\mathrm{MSSIM}(\widetilde{\mathcal{I}}, \mathcal{I}) = \frac{1}{M} \sum_{l=1}^{M} \mathrm{SSIM}(\tilde{\mathbf{y}}_l, \mathbf{y}_l),$$

where $\tilde{\mathbf{y}}_l$ and $\mathbf{y}_l$ are the image contents at the l-th local sub-image; and M is the number of local sub-images in the image.

Finally, we take a look at the relationship between the size of the sub-image and the tolerance, and how this affects the quality of the approximation. We analyze the idealized error distribution in which all pixels of the approximation are c units apart from the original. Consider an $L \times L$ sub-image that has been linearized to a vector $\mathbf{y}$ of length L^2. Assume that the OMP approximation within ϵ has distributed the error evenly, that is, if $\mathbf{x} = \texttt{OMP}(\mathbf{A}, \mathbf{y}, \epsilon)$ and $\tilde{\mathbf{y}} = \mathbf{A}\mathbf{x}$, then

$$\begin{aligned} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 < \epsilon &\Leftrightarrow \|\tilde{\mathbf{y}} - \mathbf{y}\|_2^2 < \epsilon^2, \\ &\Leftrightarrow \sum_{j=1}^{L^2} \big(\tilde{\mathbf{y}}(j) - \mathbf{y}(j)\big)^2 < \epsilon^2, \\ &\Leftrightarrow L^2 c^2 < \epsilon^2, \\ &\Leftrightarrow c < \frac{\epsilon}{L}. \end{aligned} \tag{6}$$

That is, if we want to be within c units from each pixel, we have to choose a tolerance ϵ such that $c = \epsilon/L$.

We note that the least-squares approximation at the core of OMP approximates the idealized error distribution. This can be seen in figure 10 where the black dashed line represents this idealized error approximation. For tolerances $\epsilon > 40$, we can see that the PSNR for all images considered are above this idealized error distribution. This can be explained by noting that, for example, for $\epsilon = 2048$, we would have from equation (6) that $c = 2048/8 = 256$, but the maximum pixel value is only 255. Therefore, unless the original image $\mathcal{I}$ is just a white patch, the initial value of the OMP approximation being an all black image, there are matching pixels in the original and the approximation image $\widetilde{\mathcal{I}} = rec(\mathcal{I}, \mathbf{A}, 2048)$ that are less than 256 units apart. This would necessarily, by definition 3, imply $\mathrm{PSNR}(\widetilde{\mathcal{I}}, \mathcal{I}) > 0$, a value above the value of the PSNR for the idealized error distribution when $\epsilon = 2048$, which is a small negative value.

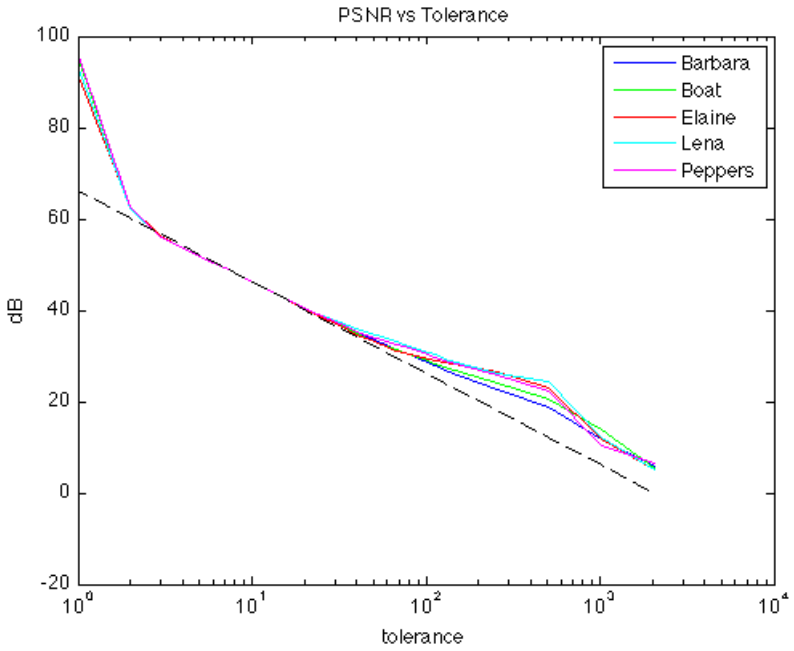


Figure 10: Peak Signal-to-Noise Ratio vs tolerance. We observe three typical behaviors for all images. For large values of the tolerance, about $\epsilon > 40$, the PSNR of all images is above the PSNR value for the idealized error distribution marked by the black dashed line. This behavior is also observed for very small values of the tolerance, about $\epsilon < 3$. Then for values between these two extreme behaviors, all images conform very closely to the idealized error distribution, fact that is expected from the least-squares approximation at the core of the OMP algorithm.

On the other hand, for really small tolerances, about $\epsilon < 3$, we observe that the PSNR value for all images jumps again above the PSNR for the idealized error model. This is a happy case when roundoff error actually helps. What happens is that for such small tolerances, the roundoff to the closest integer for all entries in $\tilde{\mathbf{y}}_l = \mathbf{A}\mathbf{x}_l$ when we form the sub-image approximation $\widetilde{\mathbf{Y}}_l = c2^{-1}(\tilde{\mathbf{y}}_l)$, coincides with the true value of the pixels in the original sub-image $\mathbf{Y}_l$. Again, by definition 3, this increases the value of PSNR$(\widetilde{\mathcal{I}}, \mathcal{I})$ compared to the case where roundoff would not have taken place.

6.2.5 PSNR and MSSIM comparison

Now that we have some tools to asses the quality of a reconstruction, how do they compare?

In figure 11 we have plotted the compression ratio versus both error indices MSSIM and PSNR. The first thing that we observe is that the sensitivity for PSNR varies more dramatically than the sensitivity for MSSIM over the range of tolerances chosen.

From figure 12 we can observe that for the range of 20 to 40 dB in PSNR, the MSSIM index ranges from about 0.4 to 0.97. Since a value of 1 in MSSIM corresponds to two identical images, we can focus on values of PSNR no greater than 40 dB in our analysis. Also in figure 12 we corroborate the criticism that has been addressed to PSNR as a measure

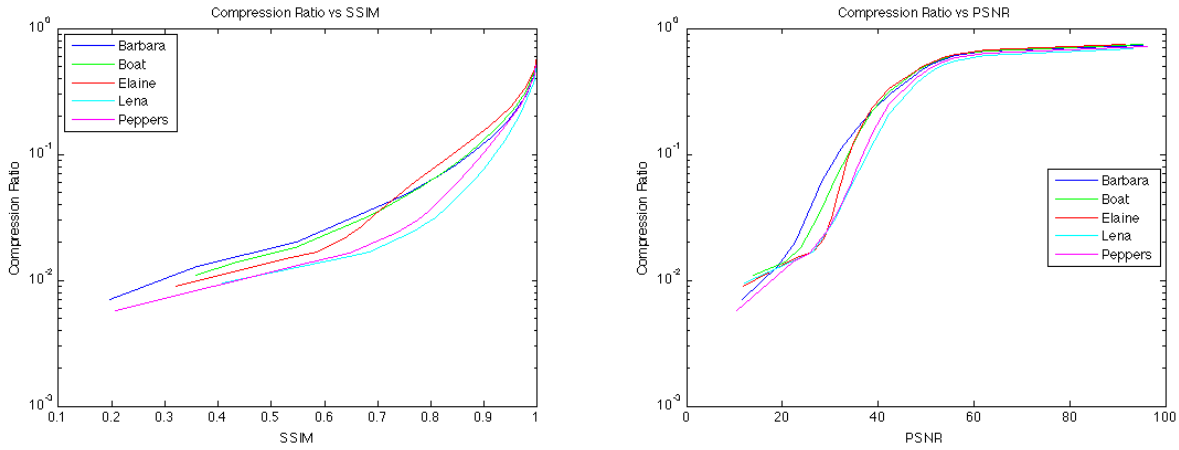


Figure 11: Compression ratio vs MSSIM, PSNR

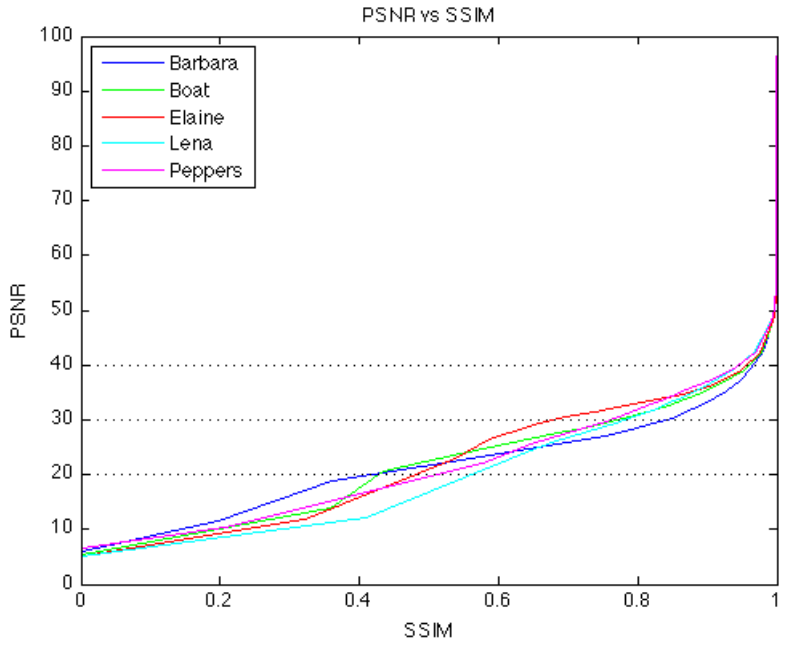


Figure 12: Peak Signal-to-Noise Ratio vs Mean Structural Similarity

of image quality. For example, for image *Barbara* at 20 dB we have an MSSIM value of 0.4, whereas for image *Lena* we have an MSSIM value of 0.55. A similar wide range between 0.69 (*Elaine*) and 0.84 (*Barbara*) for MSSIM is observed for 30 dB in PSNR. It is not until 40 dB that we have a much smaller range of MSSIM values, 0.95 (*Peppers*) to 0.97 (*Barbara*). Therefore, if SSIM/MSSIM capture more accurately the human visual system perception of image quality, then the PSNR index is shown to be not so good at it until after values larger than or equal to 40 dB.

We therefore drop from the rest of our analysis the PSNR index, other than for an occasional reference point, and focus on the SSIM and MSSIM indices. We retake the questions at the end of section 6.2.3 and answer them with figure 13. From it, if we were to consider desirable values of MSSIM to be above or equal to 0.9, we would see that this would correspond to a tolerance of less than 32 for the *Peppers* and of 48 for *Barbara*, all other tolerances for the rest of the images falling in between these two values.

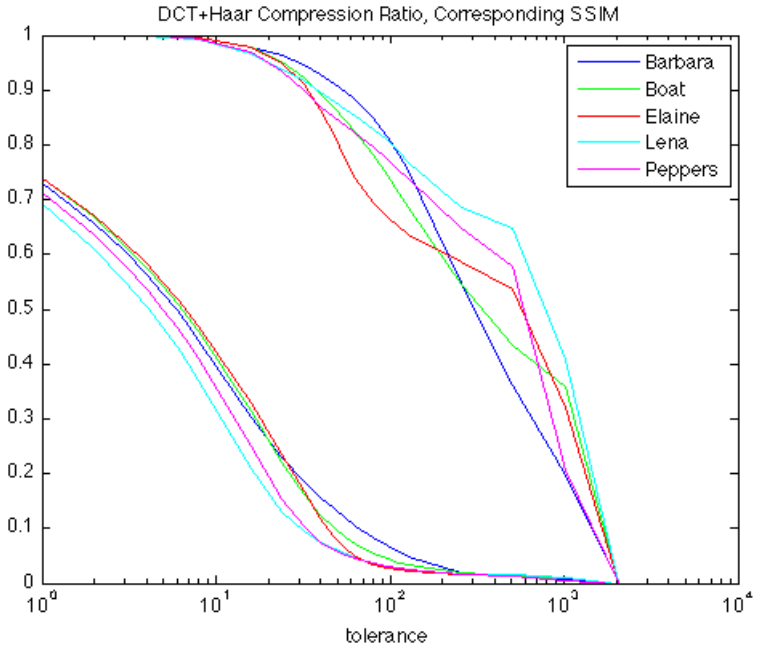


Figure 13: Compression ratio and its corresponding MSSIM vs tolerance. In this graph we have plotted together the best compression ratio obtained by combining the DCT and Haar bases, and the corresponding value of the MSSIM index for a given tolerance. The compression ratio graphs are on the bottom left, and the MSSIM index values are above these. This is done for all images.

This means that if we wanted all images to have an MSSIM index of 0.9 or better, we would have to pick a tolerance no larger than $\epsilon = 32$. This tolerance corresponds, according to equation (6), to an average value of at most $32/8 = 4$ units away per pixel. Under these circumstances we would achieve a compression ratio of 0.1 to 0.18, that is, 10:1 to 5.5:1 compression. But how good would images with MSSIM greater than or equal to 0.9 actually look? Moreover, what if we could modify OMP as to guarantee a certain MSSIM quality

level? It turns out that this modification is possible.

Consider the following change in the termination condition for the OMP algorithm from $\|\mathbf{Ax} - \mathbf{b}\|_2 < \epsilon_0$ to $\|\mathbf{Ax} - \mathbf{b}\|_{SSIM} \equiv \text{SSIM}(c_2^{-1}(\mathbf{Ax}), c_2^{-1}(\mathbf{b})) > \delta_0$, where δ_0 is a desired minimum MSSIM index value to achieve in each individual sub-image of the reconstruction of $\mathcal{I}$. When we make this change, and recompute the compression ratio vs MSSIM graphs, we obtain figure 14. In this figure, we observe that changing the termination condition for OMP leads to an improvement in the compression ratio without sacrificing image quality. Or, view from the opposite perspective, given a compression ratio, we can achieve a better image quality index MSSIM with the new stopping criteria. As we shall see in the pictures below, this change redistributes the work that OMP performs more evenly across the image.

Finally, to address the question of how good images actually look, we let the reader be the judge. See figures 15 through 20. All figures consist of two images, the reconstruction from the original, to the left; and the SSIM index map to the right. The SSIM map represents the localized quality of the image reconstruction. Lighter values are values closer to 1 (white = 1), whereas darker values are values closer to 0 (black = 0). Figures 15 through 18 show the *Boat* with different tolerances and stopping criteria. Figures 19 and 20 show *Barbara*.

7 Conclusions, future work, and acknowledgements

We achieved all goals of the project, and had extra time to explore other aspects that were not in the initial project proposal.

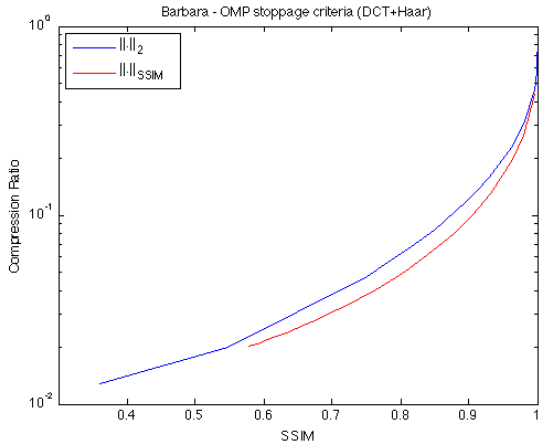
Our implementations of the *orthogonal matching pursuit* algorithm were validated against our validation protocol and a published public implementation found in [4]. Through a series of modifications to our initial algorithm implementation we were able to improve its performance two fold for the general matrix version.

During the testing phase of the project we reproduced successfully published results in [1], and our implementation was slightly more robust in finding sparse solutions than the implementation benchmark used above.

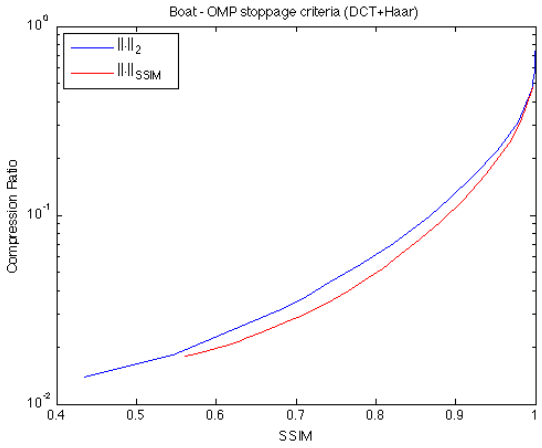
Also, we studied the compression properties of $\mathbf{A} = [\text{DCT Haar}]$ as part of our test suite. We established that this matrix has better compression properties in the context of image processing than if the DCT or Haar bases are used by themselves. We used the PSNR, SSIM, and MSSIM indices to assess the quality of image reconstructions. This gave us the clue to modify the stopping criteria of OMP and obtain that way better compression ratios for equivalent perceived image quality reconstructions.

There are, however, some aspects to this work that would require more study. In the future we would like to address the very important issue of how to actually store the sparse representations obtained by solving problem (P_0^ϵ) expressed in equation (3). Also, it would be of relevance to study the properties of $\mathbf{A} = [\text{DCT Haar}]$, and other possible matrices $\mathbf{A}$, from the perspective of frame theory. In that sense, can we do better than Haar? What is the role of the *uncertainty principle* in all of this?

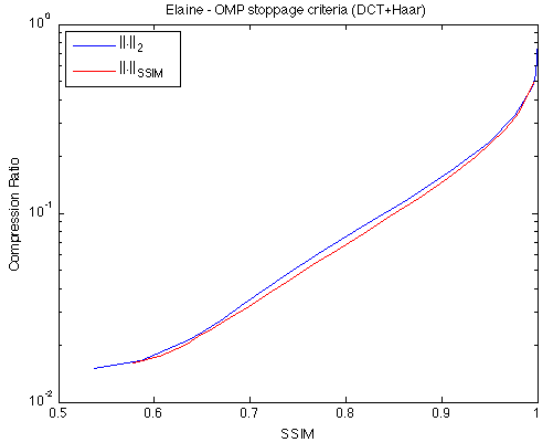
I would like to thank Radu V. Balan and Manuel Tiglio for their support and constructive feedback in the preparation and execution of this AMSC 663/664 class project. I would like to particularly thank John J. Benedetto, my advisor, for his unwavering support and valuable insights. Thank you John!



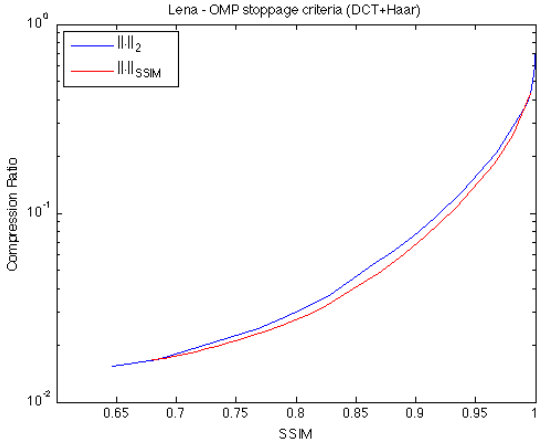
(a) Barbara



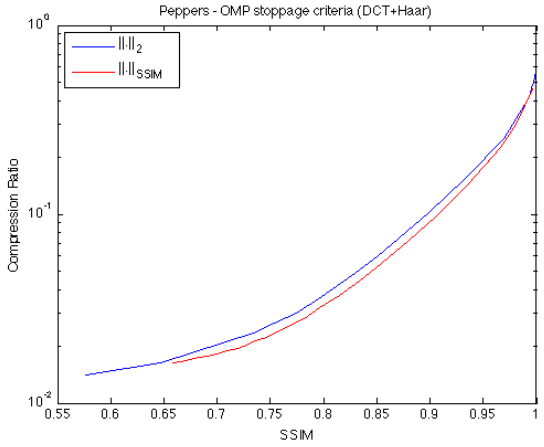
(b) Boat



(c) Elaine

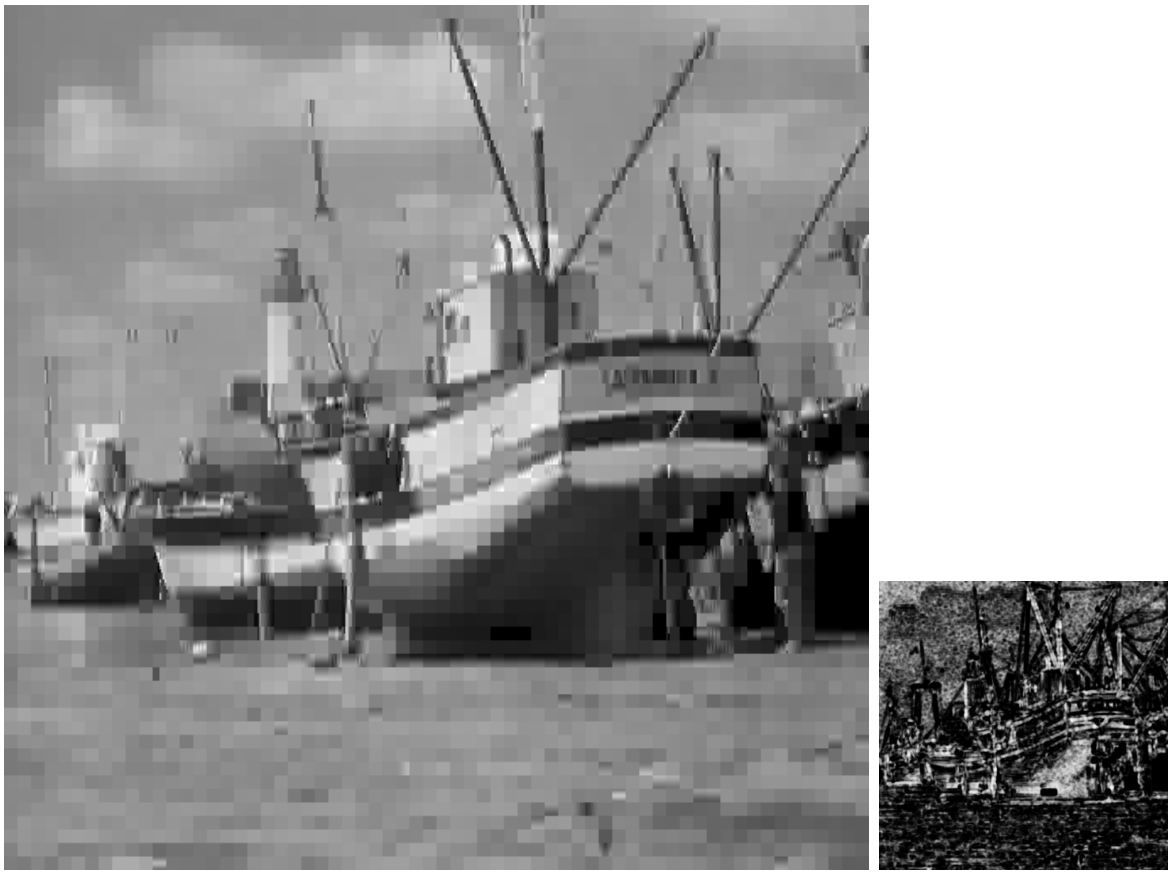


(d) Lena



(e) Peppers

Figure 14: Compression ratio vs MSSIM. Comparison of different termination criteria for OMP.



(a) Boat (b) MSSIM

Figure 15: Boat: $\epsilon_0 = 200$, PSNR = 25.2711, MSSIM = 0.6006, compression = 46.08 : 1, termination criteria: $\|\cdot\|_2$

(a) Boat (b) MSSIM

Figure 16: Boat: $\epsilon_0 = 64$, PSNR = 31.7332, MSSIM = 0.8222, compression = 14.08 : 1, termination criteria: $\|\cdot\|_2$

(a) Boat (b) MSSIM

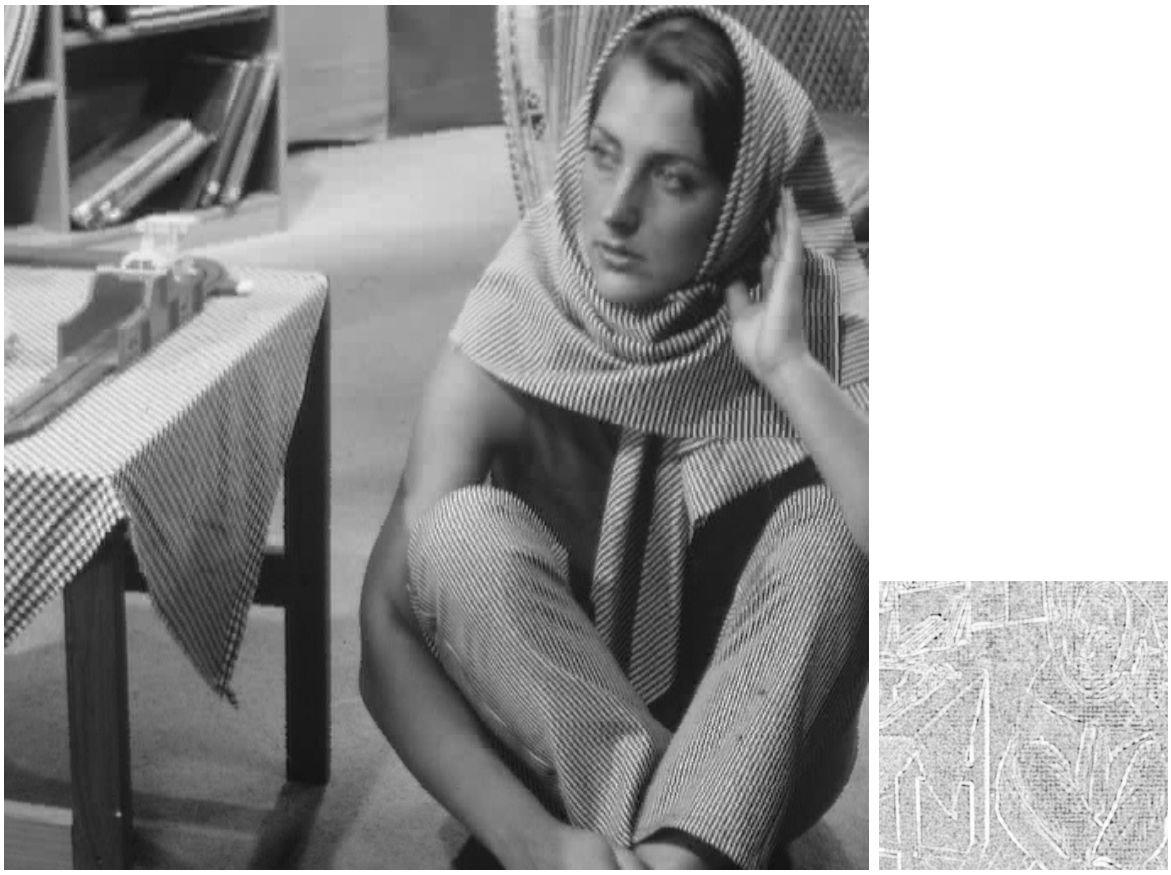
Figure 17: Boat: $\epsilon_0 = 32$, PSNR $= 36.6020$, MSSIM $= 0.9214$, compression $= 6.22 : 1$, termination criteria: $\|\cdot\|_2$

(a) Boat (b) MSSIM

Figure 18: Boat: $\delta_0 = 0.92$, PSNR = 34.1405, MSSIM = 0.9355, compression = 6.27 : 1, termination criteria: $\|\cdot\|_{SSIM}$

(a) Barbara (b) MSSIM

Figure 19: Barbara: $\epsilon_0 = 32$, PSNR = 36.9952, MSSIM = 0.9447, compression = 5.36 : 1, termination criteria: $\|\cdot\|_2$



(a) Barbara (b) MSSIM

Figure 20: Barbara: $\delta_0 = 0.94$, PSNR = 32.1482, MSSIM = 0.9466, compression = 6.49 : 1, termination criteria: $\|\cdot\|_{SSIM}$

References

[1] A. M. Bruckstein, D. L. Donoho, and M. Elad, *From sparse solutions of systems of equations to sparse modeling of signals and images*, SIAM Review, 51 (2009), pp. 34–81.

[2] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 1998.

[3] B. K. Natarajan, *Sparse approximate solutions to linear systems*, SIAM Journal on Computing, 24 (1995), pp. 227–234.

[4] SparseLab. http://sparselab.stanford.edu/.

[5] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, 1973.

[6] T. Strohmer and R. W. Heath, *Grassmanian frames with applications to coding and communication*, Appl. Comput. Harmon. Anal., 14 (2003), pp. 257–275.

[7] D. S. Taubman and M. W. Mercellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2001.

[8] G. K. Wallace, *The JPEG still picture compression standard*, Communications of the ACM, 34 (1991), pp. 30–44.

[9] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, *Image quality assessment: From error measurement to structural similarity*, IEEE Transactions on Image Processing, 13 (2004), pp. 1–14.

[10] A. B. Watson, ed., *Digital Images and Human Vision*, The MIT Press, 1993.

[11] Wikipedia. http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.