A Flexible Implementation of Matching Pursuit for Finite Gabor Sequences

S. E. Ferrando, L. A. Kolasa and N. Kovačević

Research partially supported by NSERC

The matching pursuit algorithm of Mallat et. al. is discussed in the context of discretized Gabor functions on an interval. Results from frame theory are used to introduce corresponding finite dictionaries. We then proceed to describe two software implementations based on these dictionaries. One implementation allows for users to have great flexibility in the Gabor dictionary to be used. This is a useful improvement over other implementations which only allow for a fixed dictionary. The other implementation takes advantage of the FFT algorithm and is faster. These implementations are written in C++, and can be used in many practical situations given its flexibility and generality.

Categories and Subject Descriptors: G.1.2 [**Numerical Analysis**]: Approximations; G.4 [**Mathematics of Computing**]: Mathematical Software; I.4.5 [**Image Processing**]: Reconstruction.

General Terms: Algorithms

Additional Key Words and Phrases: Adaptive signal processing. Gabor analysis.

1. INTRODUCTION

In digital signal processing it is often advantageous to analyize a given signal using an adaptive method (see [Mallat 1998] for example), where the signal is approximated or represented as a superposition of “basic” waveforms which are chosen from a *dictionary* of such waveforms so as to best match the signal. The goal is to obtain expansions which provide good synthesis for applications such as *denoising* and *super resolution* ([Chen et al. 1995]). The dictionary is an *overcomplete* collection of vectors thereby providing a number of bases from which to choose when representing a given signal. There are, of course, many measures of what constitutes a good dictionary relative to a given collection of signals; once one has been chosen the central problem of adaptive signal analysis is to search for and find the best

Name: S. E. Ferrando
Address: Department of Mathematics, Physics and Computer Science, Ryerson Polytechnic University, Toronto, Ontario M5B 2K3, Canada.
email: ferrando@acs.ryerson.ca
Name: L. A. Kolasa
Address: Department of Mathematics, Physics and Computer Science, Ryerson Polytechnic University, Toronto, Ontario M5B 2K3, Canada.
email: lkolasa@acs.ryerson.ca
Name: N. Kovačević
Address: Sunnybrook Health Science Centre, Toronto, Ontario M4N 3M5, Canada.
email: nkovacev@sten.sunnybrook.utoronto.ca

or ideal representation of a signal in that dictionary. In general this is a difficult problem and a number of techniques have been developed [Chen and Donoho 1995], [Chen et al. 1995], [Coifman and Wickerhauser 1992] and [Mallat and Zhang 1993]. In [Mallat and Zhang 1993] the matching pursuit (MP) algorithm was introduced. In matching pursuit the single dictionary element which best matches the signal is removed from the signal, and the search process for the best match is repeated with the signal residue from the previous step until a stopping rule is satisfied. If by "best dictionary element" we mean the one with maximum inner product with the signal, the residue obtained at each step has squared norm as small as possible for that step. An algorithm that operates in this way with minimal "look ahead" is known as a *greedy algorithm*. The greedy MP algorithm is simple, fast, and general with many interesting applications [Neff and Zakhor 1997], [Jaggi 1998], [Phillips 1998] however, it does not claim any global optimization property. Its generality is due to the fact that it requires minimal assumptions on the dictionary vectors, which must only belong to a Hilbert space. Rates of convergence and other mathematical questions related to MP are investigated in [DeVore and Temlyakov 1996] and [Mallat and Zhang 1993].

In general, greedy algorithms offer a tradeoff between global optimization properties and speed. The fact that the MP algorithm may offer speed advantages over other adaptive algorithms is very much related to the special dictionary in which the algorithm is being implemented. For small dictionaries, as for example wavelet packets ([Mallat 1998]), there are no delicate issues related to the structure of the dictionary as a data type and the MP algorithm. On the other hand, for larger dictionaries the definition of a *sub-dictionary* with which to implement the MP search efficiently becomes an essential question from an implementational point of view. The goal of this paper and its associated software is to provide two well crafted implementations of MP with the *Gabor dictionary*. Currently, there are implementations of MP on this dictionary (see [Mallat 1998] for information regarding MP software); our implementation differs essentially from these implementations and complements them.

We now briefly explain the novelty of our approach in contrast to previous implementations. The Gabor dictionary is built from a gaussian window function $g(\cdot)$ that is scaled, translated and modulated as follows:

$$g_{(s,u,v,w)}(t) = \frac{K_{(s,u,v,w)}}{\sqrt{s}} g\left(\frac{t-u}{s}\right) \cos(vt+w), \tag{1}$$

where $K_{(s,u,v,w)}$ is a normalizing constant. In a digital implementation of the MP algorithm decisions must be made as how to treat the continuous variable t and the continuous parameters s, u, v, w. In [Mallat and Zhang 1993], [Mallat 1998] the following implementation is considered. A signal $f = (f[0], f[1], \ldots, f[N-1])$ is considered as being periodic of period N, where $f[i]$ is the original signal sampled at time $t = i$, $i = 0, 1, \ldots N-1$, and each of the above Gabor dictionary functions is also sampled at time $t = i$, $i = 0, 1, \ldots N-1$ and periodized. The parameters are then discretized as follows. The scale paramter, s is first chosen as a power of 2, then

$$(s, u, v) = (2^j, p2^j \Delta u, k2^{-j} \Delta v), \tag{2}$$

where $\Delta u = \frac{1}{2},\ \Delta v = \pi,\ 0 < j < \log_2(N),\ 0 \leq p < N2^{-j+1}$, and $0 \leq k < 2^{j+1}$. A value for the phase parameter, w, is found using complex arithmetic.

The advantage with this particular implementation is that the order of complexity is low—$\mathcal{O}(N \log_2(N))$. The two main disadvantages with this implementation are that one typically does not have periodic data, and that one may wish to use a finer dictionary—i.e., to make a finer partition of the parameters. A third minor drawback is the introduction of complex arithmetic needed to find the phase parameter, w. Our approach is not to treat signals as a periodic sequence, nor as an infinite, compactly supported supported sequence (aperiodic). We assume that the input signal and dictionary signals are finite in extent and do not exist beyond the boundaries of an interval. This approach eliminates well known undesirable boundary effects ([Taswell and McGill 1994]) that are so often present in the other two mentioned cases. We will also have more freedom in choosing the dictionary of Gabor functions that we shall use. In particular we may choose scales $s = a^j$, where a is not necessarily 2. The price to be paid, of course, is that the order of complexity increases. We have been very careful, however, to write fast executing code; signals of large size are easily handled; thus for typical signals our version of MP is applicable.

In [Mallat and Zhang 1993] analytical formulas are developed which allow for a low complexity algorithm; this depends heavily on the periodicity assumption. We do not assume periodicity so we must find other devices to speed up our calculations. One of the main points of our implementation revolves around our optimizations which make it viable. These are described in Section 4 where we introduce two alogrithms. The first is a flexible one whereby the user has great leeway in choosing the dictionary. The second is a fast algorithm which uses practically the same dictionary as in [Mallat and Zhang 1993] and has complexity $\mathcal{O}(N(log_2(N))^2)$. Finally, in either case, we show how to optimize away the phase parameter using basic calculus. Previously this fact seems to have passed unnoticed.

Our algorithms are part of a wavelet software package, *Wave++* ([Ferrando et al. 2000]) which has documentation and demos. The remainder of the paper is organized as follows. Section 2 and subsections describe briefly the MP algorithm for Gabor dictionaries. In Section 3 we review results from frame theory which are relevant to the MP algorithm for Gabor dictionaries. Section 4 is the core of the paper and describes the novelty and technical details of our implementations.

2. THE MP ALGORITHM

In this section we review the essential aspects of the matching pursuit algorithm as discussed in [Mallat and Zhang 1993]. Let $\mathbf{H}$ be a Hilbert space. We define a dictionary as a family $\mathcal{D} = \{g_\gamma : \gamma \in \mathbf{\Gamma}\}$ of vectors in $\mathbf{H}$ such that $\|g_\gamma\| = 1$. Let $\mathbf{V}$ be the closed linear span of the dictionary vectors. We say that the dictionary is complete if $\mathbf{V} = \mathbf{H}$ and overcomplete if $\mathcal{D}$ is complete and is a linearly dependent set.

We want to compute a linear expansion of $f \in \mathbf{H}$ over a set of vectors selected from $\mathcal{D}$ in a way that describes f as simply as possible in terms of vectors from $\mathcal{D}$. In MP this is done by successive approximations of f by orthogonal projections on elements of $\mathcal{D}$; i.e., given $g_{\gamma_0} \in \mathcal{D}$ the vector f can be written as

$$f = \langle f, g_{\gamma_0} \rangle g_{\gamma_0} + Rf,$$

where Rf is the residual vector left after approximating f in the direction of g_{γ_0}. Clearly g_{γ_0} is orthogonal to Rf, so

$$\|f\|^2 = |\langle f, g_{\gamma_0} \rangle|^2 + \|Rf\|^2.$$

To minimize $\|Rf\|$ we must maximize $|\langle f, g_{\gamma_0} \rangle|$ over $g_{\gamma_0} \in \mathcal{D}$. In general, it is only computationally feasible to find an "almost optimal" vector g_{γ_0} in the sense that

$$|\langle f, g_{\gamma_0} \rangle| = \max_{\gamma \in \mathbf{\Gamma}_\alpha} |\langle f, g_\gamma \rangle| \geq \alpha \sup_{\gamma \in \mathbf{\Gamma}} |\langle f, g_\gamma \rangle|, \tag{3}$$

where $\mathbf{\Gamma}_\alpha \subseteq \mathbf{\Gamma}$ and α is an optimality factor which satisfies $0 < \alpha \leq 1$. The construction of $\mathbf{\Gamma}_\alpha$ depends on the dictionary; typically, if the dictionary is indexed by a set of continuous parameters Γ, then Γ_α will be a discrete grid of some sort in Γ. We call the collection vectors $\{g_\gamma : \gamma \in \mathbf{\Gamma}_\alpha\}$ a *sub-dictionary.*

We continue the matching pursuit by induction. Let $R^0 f = f$. Suppose that we have computed $R^n f$, the residue of order n, for some $n \geq 0$. We then choose an element $g_{\gamma_n} \in \mathcal{D}_\alpha = \{g_\gamma : \gamma \in \mathbf{\Gamma}_\alpha\}$ which closely matches the residue $R^n f$:

$$|\langle R^n f, g_{\gamma_n} \rangle| \geq \alpha \sup_{\gamma \in \mathbf{\Gamma}} |\langle R^n f, g_\gamma \rangle|.$$

The residue $R^n f$ is decomposed as

$$R^n f = \langle R^n f, g_{\gamma_n} \rangle g_{\gamma_n} + R^{n+1} f, \tag{4}$$

which defines $R^{n+1} f$, the residue of order $n+1$. Since $R^{n+1} f$ is orthogonal to g_{γ_n},

$$\|R^n f\|^2 = |\langle R^n f, g_{\gamma_n} \rangle|^2 + \|R^{n+1} f\|^2.$$

Let us repeat this decomposition m times. Writing f in terms of the residues $R^n f$, $n = 0, 1, \ldots, m$ and applying (4) yields

$$f = \sum_{n=0}^{m-1} \langle R^n f, g_{\gamma_n} \rangle g_{\gamma_n} + R^m f.$$

The following theorem is fundamental to the MP algorithm [Mallat and Zhang 1993].

Theorem 1. *If $\mathcal{D}$ is a complete dictionary and if $f \in \mathbf{H}$ then*

$$f = \sum_{k=0}^{\infty} \langle R^k f, g_{\gamma_k} \rangle g_{\gamma_k}$$

and

$$\|f\|^2 = \sum_{k=0}^{\infty} \left| \langle R^k f, g_{\gamma_k} \rangle \right|^2.$$

2.1 Gabor Dictionaries

Below we introduce a specific family $\mathcal{D} \subset \mathbf{H} = L^2(\mathbf{R})$ consisting of Gabor functions, "windowed" trigonometric functions with infinite exponentially decreasing tails. We start by introducing continuous Gabor functions, having in mind that for the actual implementation all functions must be discretized.

As the window function $g(t)$ we use a Gaussian given by

$$g(t) = 2^{1/4} e^{-\pi t^2}. \tag{5}$$

notice that this is a bit different from (8), the reason for this is just to keep some compatibility with the notation and the discrete dictionary used in [Mallat and Zhang 1993]. For any $\gamma = (s, u, v) \in \mathbf{R}^+ \times \mathbf{R}^2 = \Gamma$, let the Gabor function g_γ be given by

$$g_\gamma(t) = \frac{1}{\sqrt{s}} g\left(\frac{t-u}{s}\right) e^{ivt}.$$

The factor $1/\sqrt{s}$ normalizes $g_\gamma(t)$. Here $s > 0$ is called the *scale* of the function, u its *translation* and v its *frequency modulation*, $g_\gamma(t)$ is centered at the abscissa u and its energy is mostly concentrated in a neighborhood of u of size proportional to s. Finite linear expansions of Gabor functions are dense in $L^2(\mathbf{R})$, hence this dictionary is complete.

In order to obtain a decomposition with real expansion coefficients when the signal $f(t)$ is real, we will use dictionaries of real time-frequency functions. For any $\gamma = (s, u, v)$ with $s > 0$ and for any phase $w \in [0, 2\pi)$, define

$$g_{(\gamma,w)}(t) = \frac{K_{(\gamma,w)}}{2}(e^{iw} g_{(s,u,v)}(t) + e^{-iw} g_{(s,u,-v)}(t)) = \frac{K_{(\gamma,w)}}{\sqrt{s}} g\left(\frac{t-u}{s}\right) \cos(vt+w),$$

where the positive constant $K_{(\gamma,w)}$ is determined by the condition $\|g_{(\gamma,w)}\|_2 = 1$. The phase w which was hidden in the complex expansion coefficients now appears explicitly as a parameter of the real Gabor vectors. The dictionary of real time-frequency vectors is defined by $\mathcal{D}_R = \{g_{(\gamma,w)} : (\gamma, w) \in \Lambda = \Gamma \times [0, 2\pi)\}$. For convenience we use the notation $\beta = (\gamma, w)$. Matching pursuit performed with this dictionary decomposes any real signal $f(t)$ into the sum

$$f(t) = \sum_{n=0}^{\infty} \langle R^n f, g_{\beta_n} \rangle g_{\beta_n}(t)$$

where the indices $\beta_n = (s_n, u_n, v_n, w_n)$ are chosen by maximizing $|\langle R^n f, g_{\beta_n} \rangle|$ over Λ. In practice this maximization is not feasible and an approximation scheme as indicated in equation (3) should be used. This is done in the next section by discretizing the dictionary as suggested by frame theory.

3. FRAME THEORY

In this section we follow [Mallat 1998]. Frame theory can be used to discretize continuous transforms while retaining a complete and stable representation. For example, the windowed Fourier transform of $f \in \mathbf{L^2}(\mathbf{R})$ is defined by

$$Sf(u, \xi) = \langle f, g_{u,\xi} \rangle,$$

with

$$g_{u,\xi}(t) = g(t-u)\, e^{i\xi t}.$$

Setting $||g|| = 1$ implies that $||g_{u,\xi}|| = 1$. A discrete windowed Fourier transform representation

$$\{Sf(u_n, \xi_k) = \langle f, g_{u_n,\xi_k}\rangle\}_{(n,k)\in \mathbf{Z}^2}$$

is complete and stable if $\{g_{u_n,\xi_k}\}_{(n,k)\in\mathbf{Z}^2}$ is a *frame* of $\mathbf{L^2}(\mathbf{R})$, i.e.,

$$A||f||_2 \leq \sum_{(n,k)\in\mathbf{Z}^2} |\langle f, g_{u_n,\xi_k}\rangle|^2 \leq B||f||_2, \tag{6}$$

for every $f \in \mathbf{L^2}(\mathbf{R})$.

The time and frequency parameters (u,ξ) are discretized over a rectangular grid with time and frequency intervals of size u_0 and ξ_0. Let us denote

$$g_{n,k}(t) = g(t-nu_0)\, e^{ik\xi_0 t}.$$

The sampling intervals (u_0, ξ_0) must be adjusted to the time-frequency spread of g.

Window scaling Suppose that $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$ is a frame of $\mathbf{L^2}(\mathbf{R})$ with frame bounds A and B, as in (6). Let us dilate the window $g_s(t) = \frac{1}{\sqrt{s}} g(\frac{t}{s})$. It increases by a factor of s the time width of the Heisenberg box of g and reduces by a factor of $\frac{1}{s}$ its frequency width. We thus obtain the same cover of the time-frequency plane by increasing u_0 by s and reducing ξ_0 by $\frac{1}{s}$. Let

$$g_{s,n,k}(t) = g_s(t - nsu_0)\, e^{ik\frac{\xi_0}{s}t}.$$

It follows that $\{g_{s,n,k}\}_{(n,k)\in\mathbf{Z}^2}$ satisfies the same frame inequalities as $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$, with the same frame bounds A and B as can be seen by a change of variable $t' = ts$ in the inner product integrals.

Necessary conditions Daubechies [Daubechies 1992] proved several necessary conditions on g, u_0 and ξ_0 to guarantee that $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$ is a frame of $\mathbf{L^2}(\mathbf{R})$. We summarize the main results:

Theorem 2 (Daubechies). *The windowed Fourier family $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$ is a frame only if*

$$\frac{2\pi}{u_0\,\xi_0} \geq 1. \tag{7}$$

The frame bounds A and B necessarily satisfy

$$A \leq \frac{2\pi}{u_0\,\xi_0} \leq B,$$

$$\forall t \in \mathbf{R} \quad , \quad A \leq \frac{2\pi}{\xi_0} \sum_{n=-\infty}^{+\infty} |g(t-nu_0)|^2 \leq B,$$

$$\forall \omega \in \mathbf{R} \quad , \quad A \leq \frac{1}{u_0} \sum_{k=-\infty}^{+\infty} |\hat{g}(\omega - k\xi_0)|^2 \leq B.$$

The ratio $\frac{2\pi}{u_0\xi_0}$ measures the density of windowed Fourier atoms in the time-frequency plane.

Gaussian window The Gaussian window

$$g(t) = e^{-\frac{t^2}{2}} \tag{8}$$

has a Fourier transform $\hat{g}$ that is a Gaussian with the same variance. The time and frequency spreads of this window are identical. It is best to choose equal sampling intervals in time and frequency: $u_0 = \xi_0$ given that for the same product $u_0\xi_0$ other choices would degrade the frame bounds. If g is dilated by s then the time and frequency sampling intervals must become su_0 and $\frac{\xi_0}{s}$. If the time-frequency sampling density is above the critical value: $\frac{2\pi}{u_0\xi_0} > 1$, then Daubechies [Daubechies 1990] proves that $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$ is a frame. When $u_0\xi_0$ tends to 2π, the frame bound A tends to 0. For $u_0\xi_0 = 2\pi$, the family $\{g_{n,k}\}_{(n,k)\in\mathbf{Z}^2}$ is complete in $\mathbf{L^2(R)}$, which means that any $f \in \mathbf{L^2(R)}$ is entirely characterized by the inner products $\{\langle f, g_{n,k}\rangle\}_{(n,k)\in\mathbf{Z}^2}$. These are the results from frame theory which motivate the choice of discrete dictionary given in (2), it is easy to check that for this dictionary condition (7) is satisfied for each scale (notice our use of window (5) instead of (8)). Obviously other choices for discretizing the scale parameter are also possible. Thus, frame theory can be used to obtain useful discrete dictionaries to implement MP.

It is important to realize that the multiscale dictionaries obtained above can in turn be considered as a frame [Zibulski and Zeevi 1995] with this perspective analysis is performed by means of the frame algorithm. The expansion given by the frame algorithm actually achieves the *singular value decomposition*, i.e., solves a least squares problem for an overcomplete system of linear equations under the constrain that the sum of the squares of the coefficients is a minimum. This type of solution does not always produce good compression of the signal and, therefore, precludes the possibility of useful applications to denoising and super resolution (sparsity). This fact has been documented in [Chen and Donoho 1995], [Chen et al. 1995] and [Donoho and Huo]. Attempts to remedy this problem can be found in [Daubechies et al. 1995]. Multi windows (or multi scale) approaches to this problem are discussed from different points of views in [Janssen 1998] and [Qian and Chen 1996].

This section has assumed continuous-time Gabor functions. Without further details we move to discrete signals in the next section. Formal treatments of discrete Gabor schemes can be found in [Zibulski and Zeevi 1994] and [Wexler and Raz 1990].

4. IMPLEMENTING MP WITH DISCRETIZED GABOR DICTIONARIES

4.1 Discrete Gabor Functions on an Interval

We turn now We assume that the input signal $f(t)$ has been sampled at times $t = 0, 1, \ldots, N-1$, where N is the number of equally spaced samples. Then

$$f = (f[0], f[1], \ldots, f[N-1])$$

is the input *vector* for the MP algorithm. We do the same for the dictionary elements in (1) by sampling them on the integers $t \in \{0, 1, \ldots, N-1\}$. Once a discrete partition of the parameters (s, u, v, w) is chosen, the MP algorithm takes its setting in Euclidean space: we are approximating f by elements of an overcomplete basis of $\mathbf{R}^N$, where the distance is measured by the standard l_2 norm.

Recall that the dictionary elements must be normalized in the l_2 sense. The normalization constants used for continuous Gabor functions in (1) can be dropped from the formulae since they must be calculated anyway. In other words, we will say that a real Gabor function $g_{(s,u,v,w)}$ is defined by

$$g_{(s,u,v,w)}(t) = \frac{g\left(\frac{t-u}{s}\right)\cos(vt+w)}{||g\left(\frac{\cdot-u}{s}\right)\cos(v.+w)||}.$$

As before let $\gamma = (s,u,v)$ and g is given by (5). We define the following functions,

$$P_\gamma(t) = g\left(\frac{t-u}{s}\right)\cos(vt), \qquad Q_\gamma(t) = g\left(\frac{t-u}{s}\right)\sin(vt).$$

Then we find that

$$g_{(\gamma,w)} = \frac{P_\gamma \cos w - Q_\gamma \sin w}{||P_\gamma \cos w - Q_\gamma \sin w||},$$

and therefore inner products will be given by

$$\langle f, g_{(\gamma,w)}\rangle = \frac{\langle f, P_\gamma\rangle \cos w + \langle f, Q_\gamma\rangle \sin w}{||P_\gamma \cos w - Q_\gamma \sin w||}.$$

Recall that at each step of the MP algorithm we seek $\max_{\gamma,w} |\langle R^n f, g_{(\gamma,w)}\rangle|$. The following proposition explains how to find the optimal w for a given $R^n f$ and γ. As a result, the fourth parameter w is uniquely determined, and the size of the dictionary depends on partitions in the first three parameters s, u and v only.

Proposition 1. *Given* $f = (f[0], \ldots, f[N-1])$ *and* $\gamma = (s,u,v)$ *denote* $P = P_\gamma$, $Q = Q_\gamma$ *and*

$$a = \langle f, P\rangle \qquad b = \langle f, Q\rangle a_1 = a||Q||^2 - b\langle P, Q\rangle \quad b_1 = b||P||^2 - a\langle P, Q\rangle. \tag{9}$$

Then an optimal w_0, *i.e., one for which* $\max_{w\in[0,2\pi)} |\langle f, g_{(\gamma,w)}\rangle|$ *is attained is given by:*
(i) If $v = 0$, *then*

$$w_0 = 0, \ \textit{and} \ \langle f, g_{(\gamma,w_0)}\rangle = \frac{a}{||P||}. \tag{10}$$

(ii) If $v \neq 0$ *and* $a_1 = 0$, *then*

$$w_0 = \frac{\pi}{2}, \ \textit{and} \ \langle f, g_{(\gamma,w_0)}\rangle = -\frac{b}{||Q||}. \tag{11}$$

(iii) If $v \neq 0$ and $a_1 \neq 0$, then

$$tan\, w_0 = -\frac{b_1}{a_1} \quad \textit{and} \quad \langle f, g_{(\gamma,w_0)}\rangle = \frac{aa_1 + bb_1}{||Pa_1 + Qb_1||}. \tag{12}$$

Proof. If $v = 0$, then $Q_\gamma \equiv 0$ and $\langle f, g_{(\gamma,w)}\rangle$ is independent of w, and so we *choose* $w_0 = 0$ as in (10). Assuming $v \neq 0$, we let $x = \tan(w)$. The proof now reduces to maximazing the function of one variable $h(x) = \langle f, g_{(\gamma,w)}\rangle^2$; we see that

$$h(x) = \frac{(a-bx)^2}{||P-Qx||^2} = \frac{(a-bx)^2}{||P||^2 + ||Q||^2x^2 - 2\langle P, Q\rangle x}.$$

Taking derivatives we find

$$h'(x) = \frac{-2(a-bx)}{(\|P-Qx\|)^4}(a_1x+b_1).$$

When $a_1 = 0$, $h(x)$ has no global maximum, except asymptotically at $|x| = \infty$ which is equivalent to choosing $w_0 = \pi/2$ as in (11). Otherwise, it is clear that $x = -b_1/a_1$ where $h(x)$ takes its global maximum. Choosing this value for $\tan(w_0)$ gives (12)

$\square$

We present two implementations of MP based on discretized Gabor functions.

—Implementation A, offers flexibility in choosing partitions. It also has the advantage that it works with arbitrary dimensions of the input signal (e.g., N does not have to be a power of 2).

—Implementation B, which requires that N is a power of 2 and is restricted to the dictionary given by (2). The algorithm is based on the Fast Fourier Transform and is faster.

4.2 Data Structures

The algorithms have been written in C++. We find this language convenient because it allows for the memory management associated with data structures to be implemented "behind the scenes", whereby the user treats data structures like native data types. Otherwise we make little use of the *Object Oriented* features of C++.

We use two type definitionss:

—`typedef real double;`

—`typedef integer long;`

Naturally, users can change these definitions according to their concerns for memory usage and precision requirements.

The three data structures, or classes, which we use are `Interval, RealGabor` and `Partition`. We use the class `Interval` from *Wave++* ([Ferrando et al. 2000]). This class is basically an array designed to represent vectors of real numbers, with indices being any set of consecutive integers. One typical use of `Interval` is for storing an input signal f. In our implementations of the MP algorithm, input signals are indexed from 0 to $N-1$. This reflects the idea that we can think of an input signal f as a sample of some function evaluated at times $t = 0, t = 1, \ldots, t = N-1$. `RealGabor` is a class which is designed to encode real Gabor functions. Its data members s, u, v and w have obvious interpretations as scale, center, frequency and phase. The member function `evaluate(real t)` returns $g_{(s,u,v,w)}(t) = g\left(\frac{t-u}{s}\right)\cos(vt+w)$. The data member `Sample` of `RealGabor` contains the sample values of $g_{(s,u,v,w)}(t)$ at times $t = 0, t = 1, \ldots, t = N-1$. When needed, the data member `Sample` is set by calling the member function `createSample(Interval I)` which first samples $g_{(s,u,v,w)}$ using `evaluate` on elements of `I`, and then normalizes the sample to unity in the l_2 norm. `Partition` is explained below.

4.3 Implementation A

One main feature of this implementation is the flexibility afforded in choosing the fineness of the parameter partition. Typically one does not make an arbitrary choice for the parameters s, u, v, but given the considerations in Section 3, one chooses the desires scales $s[j]$ first (often $s[j] = a^j$), then one chooses the translations $u[j]$ and the frequencies $v[j]$ with guidance from Theorem 2. Therefore this implementation can accomodate any partition in s, u, v satisfying the following conditions:

—s can take any desired values, say $s[j]$ for $1 \leq j \leq n$, provided that $0 \leq s[1] < s[2] < \ldots s[n]$. The $s[j]$'s need not be integers.

—Once we have decided the leftmost and rightmost values of u, denoted `lmu` and `rmu` respectively, then for any fixed $s[j]$ the partition in u is defined by a constant increment $du = du[j]$, which depends on $s[j]$ only. More precisely, $u \in \{\texttt{lmu} + p du \,|\, p \in \mathbf{Z},\, 0 \leq p,\, \texttt{lmu} + p\,du \leq \texttt{rmu}\}$.

—The leftmost value of v is set to be 0 always. Once we have decided on what is the rightmost value of v, denoted `rmv`, then for a fixed $s[j]$ the partition in v is defined by a constant increment $dv = dv[j]$, which depends on $s[j]$ only. More precisely, $v \in \{kdv \,|\, k \in \mathbf{Z},\; 0 \leq k,\, k\,dv < \texttt{rmv}\}$.

We encode all this information in the class named `Partition`. Its only constructor `Partition(integer N, real a)` is designed to create the following partition suitable for signals of dimension N:

—$s[j] = a^j$, for $1 \leq j \leq n$, where n is the largest integer power of a such that $a^n \leq N$

—$du[j] = s[j]/2$ and $dv[j] = \pi/s[j]$

—$\texttt{lmu} = 0$, $\texttt{rmu} = N - 1$ and $\texttt{rmv} = 2\pi$.

For example, the partition given by (2) is obtained by using this constructor with $a = 2$. Users can decide to define a completely different partition as long as it complies with rules stated above. In this case, though, they would have to set all data members with a user defined function.

We implement the MP algorithm using two functions: `getOptimalGabor` and `RunGaborMP`. The first one is the core function: given $R^n f$ it searches the entire dictionary for a Gabor function which has a maximal inner product with $R^n f$. The second function is a wrapper function which runs the MP algorithm by repeated calls to `getOptimalShiftGabor`.

The main expenses in terms of the complexity of the MP algorithm are the number of arithmetic operations performed when calculating inner products and the number of function calls made. In this implementation we have made great efforts to reduce the number of function calls, so that in spite of the complexity, the algorithm executes quickly for $N = 2^{12}$, an applicable size.

For illustrative purposes let us look at the level of the inner most loop where we need to evaluate, for chosen values of s, u and v,

$$P_{(s,u,v)}[t] = g\left(\frac{t-u}{s}\right)\cos(vt), \text{ where } t_i = 0, 1, \ldots, N-1. \tag{13}$$

The logic of the `Partition` structure causes the outermost loop of the `getOptimalShiftGabor` algorithm to be indexed by the scale variable, s, say $s = s_j$.

Once s is chosen the increments for u and v, respectively du_j and dv_j, are fixed. At this point we must choose which loop is next, and we choose the next loop to be indexed by the frequency variable, v. This allows us to minimize function calls in the following way.

We see from (13) that the cosine evaluations are independent of the exponential. For a fixed of $v = v_k = k\,dv_j$ we store the values of $\cos(v_k t_i),\ \sin(v_k t_i)\quad i = 0, 1, \ldots, N-1$ in two arrays [1]. Clearly, as we progress through the innermost loop we need not recalculate the sine and cosine values again. But even more savings may be realized by an updating scheme which makes use of trigonometric identities. Apparently

$$\begin{aligned}\cos(v_k t_i) = \cos(k\,dv_j t_i) = {} & \cos((k-1)\,dv_j t_i)\cos(dv_j t_i) - \sin((k-1)\,dv_j t_i)\sin(dv_j t_i)\\ & = \cos(v_{k-1} t_i)\cos(dv_j t_i) - \sin(v_{k-1} t_i)\sin(dv_j t_i).\end{aligned} \tag{14}$$

Also,

$$\cos(dv_j t_i) = \cos(dv_j t_{i-1})\cos(dv_j) - \sin(dv_j t_{i-1})\sin(dv_j). \tag{15}$$

Thus we may update the values in the arrays $\cos(v_k t_i),\ \sin(v_k t_i)\quad i = 0, 1, \ldots, N-1$ by using the previously stored values $\cos(v_k t_i), \sin(v_k t_i)$, applying (14), while the "increments" $\cos(dv_j t_i),\ \sin(dv_j t_i)$ are updated as in (15) following only two initial function evaluations $\cos(dv_j),\ \sin(dv_j)$.

Before we enter the innermost loop we make use of one last savings device. It is not necessary to calculate the value of the exponentials for *every* value of the translation parameter u. We can simply offset or shift index values by making use of the following identities:

$$\begin{aligned}P_{(s,c+\delta,v)}[t] &= P_{(s,c,v)}[t-\delta]\cos(v\delta) - Q_{(s,c,v)}[t-\delta]\sin(v\delta)\\ Q_{(s,c+\delta,v)}[t] &= P_{(s,c,v)}[t-\delta]\sin(v\delta) + Q_{(s,c,v)}[t-\delta]\cos(v\delta)\end{aligned} \tag{16}$$

Therefore, for fixed s and v, we need only evaluate *one* P and Q, say $P_{(s,c,v)}$ and $Q_{(s,c,v)}$, where c is some fixed value (we took $c = N/2$). Then for arbitrary u, set the *shift*, δ [2], by $\delta = u - c$, and then

$$\begin{aligned}\langle f, P_{(s,u,v)}\rangle &= \cos(v\delta)\textstyle\sum_t f[t]P_{(s,c,v)}[t-\delta] - \sin(v\delta)\sum_t f[t]Q_{(s,c,v)}[t-\delta],\\ \langle f, Q_{(s,u,v)}\rangle &= \sin(v\delta)\textstyle\sum_t f[t]P_{(s,c,v)}[t-\delta] + \cos(v\delta)\sum_t f[t]Q_{(s,c,v)}[t-\delta].\end{aligned} \tag{17}$$

The input parameters of `getOptimalShiftGabor` are the signal f and the partition `Part`. Its output parameters are the real Gabor function `G` and the scalar `coef`. On output `G` receives the optimal Gabor function from the dictionary defined by *Part* and `coef` $= \langle G, f\rangle$. The algorithm in its most basic outline performs the search in the following way:

```
coef = 0
for $j = 1;\ j \le n;\ j++$
```

[1]The actual implementation stores something else, but the effect is the same. See the code for `getOptimalShiftGabor` for the technical details.

[2]By (16) the δ's must be integers and so we need the $du[j]$'s to be integers, even though they are defined as reals in the class `Partition`. So `getOptimalGabor` will use integer casting on $du[j]$'s (which is essentialy flooring).

```
    $s = s[j]$
    for $v = 0$; $v \leq$ rmv; $v+ = dv[j]$
        update sines and cosines according to (14) and (15)
        Calculate $P_{(s,c,v)}$ and $Q_{(s,c,v)}$, where $c = N/2$
        for $u =$ lmu; $u \leq$ rmu; $u+ = du[j]$
            set $P = P_{(s,u,v)}$ and $Q = Q_{(s,u,v)}$ according to (16)
            calculate $a$, $b$, $a_1$, $b_1$ according to (9)
            find optimal phase w using Proposition
            calculate $product = \langle f, g_{(s,u,v,w)} \rangle$
            if $|product| >$ coef
            set coef $= product$ and G $=$ G$(s,u,v,w)$
        end
    end
end
```

Now let us examine the wrapper function RunGaborMP. Its input parameters are: a partition Part, an input signal f, a maximal number of iterations max_iter and a precision ϵ. Its output parameters are: a vector of real Gabor functions G, a vector of scalars Gcoef corresponding to elements of G, an output signal f_{approx} and the residual Rf. RunGaborMP will keep calling the core function getOptimalGabor and after each such call, G is appended with one more Gabor function and Gcoef is appended with the corresponding coefficient. On the output we will have

$$f_{approx} = \sum_{i=0}^{n} \texttt{Gcoef[i]} * \texttt{G[i]}, Rf = f - f_{approx},$$

where n denotes the number of iterrations performed. If ϵ-precision is achived in $n < max\_iter$ steps, i.e., $||Rf|| < \epsilon$, then the function returns early, otherwise $n = max\_iter$. An outline of the code is

```
$f_{approx}$ = 0
$Rf = f$
for $i = 0$; $i \leq max\_iter$; $i++$
    getOptimalShiftGabor(Rf, Part) return G[i], Gcoef[i]
    $f_{approx} = f_{approx} + Gcoef[i] * G[i]$
    $Rf = Rf - Gcoef[i] * G[i]$
    error$^2 =$ error$^2 + ||Rf||^2$
    if error < epsilon, stop
end
```

It should be noted that the Gabor functions comprising G have their samples set after they come fresh from RunShiftGabor. So if users want to watch how MP progresses from iterration to iterration, they can plot intermediate steps. For example, using Interval arithmetics we can find j-th approximation of f as $\sum_{i=0}^{j} G[i].Sample * Gcoef[i]$. A simple analysis of the cost tell us that for N data points the above algorithm takes order of magnitude N^2 calculations, when a partition given by the constructor for the Partition class is used.

4.4 Implementation B

We may also implement a fast version of the Matching Pursuit Algorithm on an interval by taking advantage of the Fast Fourier Transform (FFT) implementation of the Discrete Fourier Transform (DFT). This fast implementation takes $N \log^2(N)$ calculations. The price to be paid, however, is that we loose much of the flexibility in choosing the dictionary.

The structure of this algorithm is the same as for the previous algorithm. There is a wrapper function, `RunFFTGaborMP`, which makes repeated calls to the function, `getOptimalFFTGabor`. This latter function finds, at any given iteration, that element of the Gabor dictionary defined for this implementation which best matches the current residue. The dictionary we use is practically the same as in (2), but for technical reasons is not identically the same.

The function `RunFFTGaborMP` has the same parameter list as `RunShiftGaborMP`, save for the partition variable which is unnecessary in the former function. Otherwise `RunFFTGaborMP` behaves exaxtly the same as its counterpart explained above. The novelty of implementation B is in the way that the optimal Gabor function is found. We may save vastly on the number of arithmetic operations used by using the FFT algorithm; to do so we must set things up in the proper way. This fixes the partition that we use.

In order to be take advantage of the FFT algorithm we must first have that N is a power of 2, $N = 2^J$ for some J. Then the only allowable values for the scale parameter s are $s = 2^j,\ j = 1, 2, \ldots, J$. For a given value of s the allowable values of the frequency parameter, v are $v = 2\pi k/M$, where $M = 8s$ when $8s \leq N,\ M = N$ otherwise, and $k = 0, 1, \ldots M-1$. The values of the translation parameter, u may be chosen as desired; the fast implementation uses the same convention as the more general implementation: $u = ps/2$, where $0 \leq p \leq 2N/s$.

The algorithm begins by first choosing $s = 2^i$ and then selecting an allowable u. Having done this define

$$P_k(j) = g\left(\frac{j-u}{s}\right)\cos\left(\frac{2\pi jk}{N}\right),$$

$$Q_k(j) = g\left(\frac{j-u}{s}\right)\sin\left(\frac{2\pi jk}{N}\right).$$

This is just $P_\gamma(t)$ and $Q_\gamma(t)$ of equation (4.1) with $t = j$, $\gamma = (2^i, u, 2\pi k/N)$. For each value $k = 0, 1, \ldots N-1$ we may use the FFT to calculate

$$\langle f, P_k\rangle,\ \langle f, Q_k\rangle,\ \langle P_k, P_k\rangle,\ \langle Q_k, P_k\rangle,\ \langle Q_k, Q_k\rangle, \qquad (18)$$

for all values of k. These are precisely the quantities needed to find the parameters, (9), of Proposition 1 and, ultimately, $\langle f, g_{(s,u,v,w)}\rangle$.

The main idea is that the above quantities are just the real and imaginary parts of the DFT of carefully chosen inputs. With a little finesse we may take advantage in the gain in speed offered by the FFT algorithm. There are, however some technicalities to be dispensed with before we may actually use the FFT.

Given the effective support of $g(t/s)$, i.e., that $g \approx 0$ when $|t/s| > 4$, we must distinguish two cases, *Case I* when $8s > N$ and *Case II* when $8s \leq N$. We must also take into account the shift parameter u and its relation to the right and left hand end points of the interval. This leads to *Case II* having three subcases: (a) $0 \le u \le 4s$, (b) $4s < u \le N - 4s$ and (c) $u > N - 4s$.

Before considering the cases above we recall the formula for the DFT of M data points $\{x_j\}_{j=0}^{M-1}$. For $k = 0, 1, \ldots, M-1$ define X_k by

$$X_k = \sum_{j=0}^{M-1} x_j \exp(-2\pi i jk/M) = \sum_{j=0}^{M-1} x_j \cos(2\pi jk/M) - i \sum_{j=0}^{M-1} x_j \sin(2\pi jk/M).$$

Let us make the notation: Output.Re$[k] = Re(X_k)$, Output.Im$[k] = Im(X_k)$.

Case I is computationally the simplest. The assumption that $8s > N$ means that the support of g is effectively the whole interval. If we take as our input $x_j = f(j)g(\frac{j-u}{s})$ and have $M = N$, then for $k = 0, 1, \ldots M-1$,

$$\begin{aligned} \langle f, P_k \rangle, &= \text{Output.Re}[k], \\ \langle f, P_k \rangle, &= -\text{Output.Re}[k]. \end{aligned}$$

By taking the input to the FFT to be $x_j = g(\frac{j-u}{s})^2$ and using the double angle formulas for sines and cosines we have, with $C = $ Input.Re[0],

$$\begin{aligned} \langle P_k, P_k \rangle &= \frac{1}{2}\left(C + \text{Output.Re}[2k]\right), \\ \langle Q_k, Q_k \rangle &= \frac{1}{2}\left(C - \text{Output.Re}[2k]\right), \\ \langle Q_k, P_k \rangle &= -\frac{1}{2}\text{Output.Im}[2k], \end{aligned}$$

for $k = 0, 1, \ldots, \frac{N}{2} - 1$, and

$$\begin{aligned} \langle P_k, P_k \rangle &= \langle P_{k-\frac{N}{2}}, P_{k-\frac{N}{2}} \rangle, \\ \langle Q_k, Q_k \rangle &= \langle Q_{k-\frac{N}{2}}, Q_{k-\frac{N}{2}} \rangle, \\ \langle Q_k, P_k \rangle &= \langle Q_{k-\frac{N}{2}}, P_{k-\frac{N}{2}} \rangle, \end{aligned}$$

when $k = \frac{N}{2}, \ldots, N-1$.

In Case II $8s \le N$, and the effective support of the Gaussian $g(t/s)$ is smaller than the interval. We take advantage of the fewer number of data points and let $M = 8s$ in the FFT algorithm. The three subcases account for the center of the Gaussian, which is the main technicality in this algorithm.

Subcase (a), $0 \le u \le 4s$, is exactly as the previous case; we do everything as before with N replaced by $M = 8s$. In this case the support of $g(\frac{t-u}{s})$ still contains the left hand endpoint of the interval and so the FFT algorithm may be used exactly as before.

In subcase (b), the lefthand endpoint of the interval is no longer in the support of the Gaussian, and in order to use the DFT we must make a translation of indices; let the input to the FFT be $x_j = f(j + u - 4s)g(\frac{j-4s}{s})$. A modulation in the frequency domain takes place; let $\alpha_k = 2\pi(u - 4s)k/M$. Then,

$$\begin{aligned} \langle f, P_k \rangle, &= \cos(\alpha_k)\text{Output.Re}[k] + \sin(\alpha_k)\text{Output.Im}[k], \\ \langle f, P_k \rangle, &= \sin(\alpha_k)\text{Output.Re}[k] - \cos(\alpha_k)\text{Output.Im}[k]. \end{aligned}$$

If we now let the input be $x_j = g(\frac{j-4s}{s})^2$, and take C as before, with $k = 0, 1, \ldots, \frac{M}{2} - 1$,

$$\begin{aligned}
\langle P_k, P_k \rangle &= \frac{1}{2}\left(C + \cos(\alpha_{2k})\text{Output.Re}[2k] + \sin(\alpha_{2k})\text{Output.Im}[2k]\right), \\
\langle Q_k, Q_k \rangle &= \frac{1}{2}\left(C - \cos(\alpha_{2k})\text{Output.Re}[2k] - \sin(\alpha_{2k})\text{Output.Im}[2k]\right), \\
\langle Q_k, P_k \rangle &= \frac{1}{2}\left(- \cos(\alpha_{2k})\text{Output.Im}[2k] + \sin(\alpha_{2k})\text{Output.Re}[2k]\right).
\end{aligned}$$

And for $k = \frac{M}{2}, \ldots, M-1$ we use the above formulas with k replaced by $k - \frac{M}{2}$ as before.

Subcase (c) sees the support of the Gaussian going past the right hand endpoint of the interval. To make up for this we only have to pad with zeros. For $j = 0, 1, \ldots, N - 1 - u + 4s$ we let x_j be defined as before. Otherwise $x_j = 0$.

The logic of the `getOptimalFFTGabor` is as follows. The input signal f, which has N samples where $N = 2^J$ for some $J > 0$, is passed to `getOptimalFFTGabor`. The scales, s, are traversed in decreasing order: we start with $s = N$ and decrease s by a factor of $\frac{1}{2}$ in each successive iteration of the outermost loop. The first three passes of the s loop involve Case I, while the others involve the three subcases of Case II.

In any case, once s is fixed the next loop is indexed by the translation variable, u. The increment du is set at $\frac{s}{2}$ and the u variable is traversed paying special attention to which case and subcase is to be considered. For a fixed value of u the FFT algorithm is employed in the manner discussed above and the optimal Gabor vector, $g_(s,u,v,w)$, and the corresponding coefficient are found. Once this is accomplished the functon returns to `RunFFTGaborMP`, the rsidual is updated and the process may repeat itself as desired.

Here is pseudocode for `getOptimalFFTGabor`.

```
$s = N$
coef$= 0$
while $s > \frac{N}{8}$
        $du = \frac{s}{2}$
        for $u = 0$;  $u < N$;  $u+ = du$
            for $j = 0$;  $j < N$;  $j++$
                set $X_j = f(j)g(\frac{j-u}{s})$
                set $\tilde{X}_j = g(\frac{j-u}{s})^2$
            end
            FFT($X$), FFT($\tilde{X}$)
            for $k = 0$;  $k < N$;  $k++$
                calculate the quantities in (18)
                calculate $a$, $b$, $a_1$, $b_1$ according to (9)
                calculate $product = \langle f, g_{(s,u,v,w)} \rangle$
                if $|product| >$ coef
                set coef $= product$ and G $=$ G$(s,u,v,w)$
            end
        end
```

```
      $s = \frac{s}{2}$
end
while $s > 1$
      $du = \frac{s}{2}$
      for $u = 0;\ u \leq 4s;\ u+ = du$
          for $j = 0;\ j < 8s;\ j++$
              set $X_j = f(j + u - 4s)g(\frac{j-4s}{s})$
              set $\tilde{X}_j = g(\frac{j-42}{s})^2$
          end
          FFT($X$), FFT($\tilde{X}$)
          for $k = 0;\ k < 8s;\ k++$
              calculate the quantities in (18)
              calculate $a$, $b$, $a_1$, $b_1$ according to (9)
              calculate $product = \langle f, g_{(s,u,v,w)} \rangle$
              if $|product| >$ coef
              set coef $= product$ and G $=$ G$(s, u, v, w)$
          end
      end
      for $u = 4s;\ u \leq N - 4s;\ u+ = du$
          for $j = 0;\ j < 8s;\ j++$
              if $j \leq N - 1 - u - 4s$
                 set $X_j = f(j + u - 4s)g(\frac{j-4s}{s})$
                 set $\tilde{X}_j = g(\frac{j-42}{s})^2$
              else
                 set $X_j = \tilde{X}_j = 0$
              fi
          end
          FFT($X$), FFT($\tilde{X}$)
          for $k = 0;\ k < 8s;\ k++$
              calculate the quantities in (18)
              calculate $a$, $b$, $a_1$, $b_1$ according to (9)
              calculate $product = \langle f, g_{(s,u,v,w)} \rangle$
              if $|product| >$ coef
              set coef $= product$ and G $=$ G$(s, u, v, w)$
          end
      end
      $s = \frac{s}{2}$
end
```

Given the $N \log(N)$ complexity of the FFT algorithm, the complexity of implementation B is seen to be $N(\log(N))^2$. This compares favorably with the algorithm of Mallat in [Mallat and Zhang 1993] where they claim that their algorithm is of $N \log(N)$ complexity.

REFERENCES

Chen, S. and Donoho, D. 1995. Atomic decomposition by basis pursuit. In *SPIE International Conference on Wavelets* (July 1995). SPIE.

Chen, S., Donoho, D., and Saunders, M. 1995. Atomic decomposition by basis pursuit. *Technical Report* 479.

Coifman, R. and Wickerhauser, V. 1992. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory 38*, 712–718.

Daubechies, I. 1990. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. on Info. Theory 36*, 5 (September), 961–1005.

Daubechies, I. 1992. *Ten Lectures on Wavelets.* SIAM, Philadelphia, PA.

Daubechies, I., Landau, H., and Landau, Z. 1995. Gabor time-frequency lattices and the wexler-raz identity. *The Journal of Fourirer Analysis and Applications 1*, 4, 961–1005.

DeVore, R. and Temlyakov, V. 1996. Some remarks on greedy algorithms. *Advances in Computational Mathematics 5*, 173–187.

Donoho, D. and Huo, X. Uncertainty principles and ideal atomic decompositions. *Technical Report*.

Ferrando, S., Kolasa, L., and Kovačević, N. 2000. C++wavelets: A user's guide. *Included in distribution of Wave++,* `http://www.scs.ryerson.ca/~lkolasa/CppWavelets.html`.

Jaggi, S. 1998. High resolution pursuit for feature extraction. *J. of Appl. and Comput. Harmonic Analysis 5*, 9, 428–439.

Janssen, A. E. J. M. 1998. The duality condition for weyl-heisenberg frames. In *Gabor Analysis and Algorithms* (1998). Birkhauser.

Mallat, S. 1998. *A Wavelet Tour of Signal Processing.* Academic Press, Boston, MA.

Mallat, S. and Zhang, Z. 1993. Matching pursuit with time-frequency dictionaries. *IEEE Transactions on Signal Processing 41*, 12, 3397–3415.

Neff, R. and Zakhor, A. 1997. Very low bit-rate video coding based on matching pursuit. *IEEE Trans. on Circuit Syst. for Video Tech. 7*, 1 (February), 158–171.

Phillips, P. J. 1998. Matching pursuit filters applied to face identification. *IEEE Trans. Image Proc. 7*, 8 (August), 1150–1164.

Qian, S. and Chen, D. 1996. *Joint Time-Frequncy Analysis: Method and Application.* Prentice Hall, Englewood Cliffs, NJ.

Taswell, C. and McGill, K. C. 1994. Algorithm 735: Wavelet transform algorithms for finite-duration discrete-time signals. *ACM Transcations on Mathematical Software 20*, 3 (September), 398–412.

Wexler, J. and Raz, S. 1990. Discrete gabor expansions. *Signal Processing 21*, 207–220.

Zibulski, M. and Zeevi, Y. 1994. Frame analysis of the discrete Gabor-scheme analysis. *IEEE Transactions on Signal Processing 42*, 942–945.

Zibulski, M. and Zeevi, Y. 1995. Multi-window gabor-type transform for signal representation and analysis. In *SPIE Proc. Wavelet Applications in Signal and Image Processing I* (July 1995), pp. 116–127.