AN EFFICIENT ALGORITHM FOR THE DISCRETE GABOR TRANSFORM USING FULL LENGTH WINDOWS

PETER L. SØNDERGAARD

Abstract. This paper extends the efficient factorization of the Gabor frame operator developed by Strohmer in [17] to the Gabor analysis/synthesis operator. The factorization provides a fast method for computing the discrete Gabor transform (DGT) and several algorithms associated with it. The factorization algorithm should be used when the involved window and signal have the same length. An optimized implementation of the algorithm is freely available for download.

1. Introduction

The finite, discrete Gabor transform (DGT) of a signal f of length L is given by

$$c(m,n,w) = \sum_{l=0}^{L-1} f(l,w)\overline{g(l-an)}e^{-2\pi iml/M}. \tag{1}$$

Here g is a window (filter prototype) that localizes the signal in time and in frequency. The DGT is equivalent to a Fourier modulated filter bank with M channels and decimation in time a, [2].

Efficient computation of a DGT can be done by several methods: If the window g has short support (consists of relatively few filter taps), a filter bank based approach can be used. We shall instead focus on the case when g and f are equally long. The main advantage of the algorithm presented is its ease of use: The running time is guaranteed to be small even for long windows. This allows for the practical use of non-compactly supported windows like the Gaussian and its tight and dual windows without truncating them.

In the case when the window and signal have the same length, a factorization of the frame operator matrix was found by Zibulski and Zeevi in [19]. The method was initially developed in the $L^2(\mathbb{R})$ setting, and was adapted for the finite, discrete setting by Bastiaans and Geilen in [1]. They extended it to also cover the analysis/synthesis operator. A simple, but not so efficient, method was developed for the Gabor analysis/synthesis operator by Prinz in [15]. Strohmer [17] improved the method and obtained the lowest known computational complexity for computing the Gabor frame operator. This paper extends Strohmer's method to also cover the Gabor analysis and synthesis operators.

The advantage of the method developed in this paper as compared to the one developed in [1], is that it works with FFTs of shorter length, and does not require multiplication by complex exponentials caused by the quasi-periodicity of the Zak transform. The two methods have the same asymptotic complexity, $O(NM \log M)$, where M is the number of channels and N is the number of time steps. A more accurate flop count is presented later in the paper.

We shall study the DGT applied to multiple signals at once. This is for instance a common subroutine in computing a multidimensional DGT. The DGT defined by (1) works on a multi-signal $f \in \mathbb{C}^{L\times W}$, where $W \in \mathbb{N}$ is the number of signals.

2. Definitions

We shall denote the set of integers between zero and some number L by

$$\langle L \rangle = 0, \ldots, L-1. \tag{2}$$

The Discrete Fourier Transform (DFT) of a signal $f \in \mathbb{C}^L$ is defined by

$$\left(\mathcal{F}_L f\right)(k) = \frac{1}{\sqrt{L}} \sum_{l=0}^{L-1} f(l) e^{-2\pi i k l/L}. \tag{3}$$

We shall use the $\cdot$ notation in conjunction with the DFT to denote the variable over which the transform is to be applied. To denote all elements indexed by a variable we shall use the : notation. As an example, if $C \in \mathbb{C}^{M \times N}$ then $C_{:,1}$ is a $M \times 1$ column vector, $C_{1,:}$ is a $1 \times N$ row vector and $C_{:,:}$ is the full matrix. This notation is commonly used in Matlab and FORTRAN programming and also in some prominent textbooks, [8].

The convolution $f * g$ of two functions $f, g \in \mathbb{C}^L$ and the involution f^* is given by

$$\left(f * g\right)(l) = \sum_{k=0}^{L-1} f(k)\, g(l-k), \quad l \in \langle L \rangle \tag{4}$$

$$f^*(l) = \overline{f(-l)}, \quad l \in \langle L \rangle. \tag{5}$$

It is well known how convolution can be computed efficiently using the discrete Fourier transform. We shall use a variant of this result

$$\left(f * g^*\right)(l) = \sqrt{L} \mathcal{F}_L^{-1}\left(\left(\mathcal{F}_L f\right)(\cdot)\, \overline{\left(\mathcal{F}_L g\right)(\cdot)}\right)(l). \tag{6}$$

The Poisson summation formula in the finite, discrete setting is given by

$$\mathcal{F}_M\left(\sum_{k=0}^{b-1} g(\cdot + kM)\right)(m) = \sqrt{b}\left(\mathcal{F}_L g\right)(mb), \tag{7}$$

where $g \in \mathbb{C}^L$, $L = Mb$ with $b, M \in \mathbb{N}$.

A family of vectors e_j, $j \in \langle J \rangle$ of length L is called a *frame* if constants $0 < A \leq B$ exist such that

$$A \|f\|^2 \leq \sum_{j=0}^{J-1} |\langle f, e_j \rangle|^2 \leq B \|f\|^2, \quad \forall f \in \mathbb{C}^L. \tag{8}$$

The constants A and B are called lower and upper frame bounds. If $A = B$, the frame is called *tight*. If $J > L$, the frame is redundant (oversampled). Finite- and infinite dimensional frames are described in [4].

A finite, discrete *Gabor system* (g, a, M) is a family of vectors $g_{m,n} \in \mathbb{C}^L$ of the following form

$$g_{m,n}(l) = e^{2\pi i l m/M} g(l - na), \quad l \in \langle L \rangle \tag{9}$$

for $m \in \langle M \rangle$ and $n \in \langle N \rangle$ where $L = aN$ and $M/L \in \mathbb{N}$. A Gabor system that is also a frame is called a *Gabor frame*. The analysis operator $C_g : \mathbb{C}^L \mapsto \mathbb{C}^{M \times N}$ associated to a Gabor system (g, a, M) is the DGT given by given by (1). The Gabor synthesis operator $D_\gamma : \mathbb{C}^{M \times N} \mapsto \mathbb{C}^L$ associated to a Gabor system (γ, a, M) is given by

$$f(l) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} c(m,n)\, e^{2\pi i m l/M} \gamma(l - an). \tag{10}$$

Algorithm 1 Window factorization

WFAC(g, a, M)

(1) **for** $r = \langle c \rangle$ $k = \langle p \rangle$, $l = \langle q \rangle$
(2) **for** $s = \langle d \rangle$
(3) $tmp\,(s) \leftarrow$
$g\,(r + c \cdot (k \cdot q - l \cdot p + s \cdot p \cdot q \ \mathbf{mod}\ d \cdot p \cdot q))$
(4) **end for**
(5) $Phi\,(r, k, l, :) \leftarrow$DFT$(tmp)$
(6) **end for**
(7) **return** Phi

In (1), (9) and (10) it must hold that $L = Na = Mb$ for some $M, N \in \mathbb{N}$. Additionally, we define $c, d, p, q \in \mathbb{N}$ by

$$c = \gcd(a, M) \quad , \quad d = \gcd(b, N), \tag{11}$$

$$p = \frac{a}{c} = \frac{b}{d} \quad , \quad q = \frac{M}{c} = \frac{N}{d}, \tag{12}$$

where GCD denotes the greatest common divisor of two natural numbers. With these numbers, the *redundancy* of the transform can be written as $L/(ab) = q/p$, where q/p is an irreducible fraction. It holds that $L = cdpq$. The Gabor *frame operator* $S_g : \mathbb{C}^L \mapsto \mathbb{C}^L$ of a Gabor frame (g, a, M) is given by the composition of the analysis and synthesis operators $S_g = D_g C_g$. The Gabor frame operator is important because it can be used to find the *canonical dual window* $g^d = S_g^{-1} g$ and the *canonical tight window* $g^t = S_g^{-1/2} g$ of a Gabor frame. The canonical dual window is important because D_{g^d} is a left inverse of C_g. This gives an easy way to construct an inverse transform of the DGT. Similarly, then D_{g^t} is a left inverse of C_{g^t}. For more information on Gabor systems and properties of the operators C, D and S see [9, 6, 7].

3. The algorithm

We wish to make an efficient calculation of all the coefficients of the DGT. Using (1) literally to compute all coefficients $c\,(m, n, w)$ would require $8MNLW$ flops.

To derive a faster DGT, one approach is to consider the analysis operator C_g as a matrix, and derive a faster algorithm through unitary matrix factorizations of this matrix. This is the approach taken by [17, 16]. Unfortunately, this approach tends to introduce many permutation matrices and Kronecker product matrices. Another approach is the one taken in [1] where the Zak transform is used. This approach has the downside that values outside the fundamental domain of the Zak transform require an additional step to compute. In this paper we have chosen to derive the algorithm by directly manipulating the sums in the definition of the DGT.

To find a more efficient algorithm than (1), the first step is to recognize that the summation and the modulation term in (1) can be expressed as a DFT:

$$c\,(m, n, w) = \sqrt{L} \mathcal{F}_L \left(f(\cdot, w) \overline{g\,(\cdot - an)} \right) (mb). \tag{13}$$

We can improve on this because we do not need all the coefficients computed by the Fourier transform appearing in (13), only every b'th coefficient. Therefore, we

Algorithm 2 Discrete Gabor transform

$\text{DGT}(f, g, a, M)$

(1) $Phi = \text{WFAC}(g, a, M)$
(2) **for** $r = \langle c \rangle$
(3) **for** $k = \langle p \rangle$, $l = \langle q \rangle$, $w = \langle W \rangle$
(4) **for** $s = \langle d \rangle$
(5) $tmp\,(s) \leftarrow$ $f\,(r + (k \cdot M + s \cdot p \cdot M - l \cdot h_a \cdot a \bmod \mathrm{L})\,, w)$
(6) **end for**
(7) $Psitmp\,(k, l + w \cdot q, \cdot) \leftarrow \text{DFT}(tmp)$
(8) **end for**
(9) **for** $s = \langle d \rangle$
(10) $G \leftarrow Phi\,(:, :, r, s)$
(11) $F \leftarrow Psitmp\,(:, :, s)$
(12) $Ctmp\,(:, :, s) \leftarrow G^T \cdot F$
(13) **end for**
(14) **for** $u = \langle q \rangle$, $l = \langle q \rangle$, $w = \langle W \rangle$
(15) $tmp \leftarrow \text{IDFT}(Ctmp\,(u, l + w \cdot q, :))$
(16) **for** $s = \langle d \rangle$
(17) $coef\,(r + l \cdot c, u + s \cdot q - l \cdot h_a \bmod N, w)$ $\leftarrow tmp\,(s)$
(18) **end for**
(19) **end for**
(20) **end for**
(21) **for** $n = \langle N \rangle$,$w = \langle W \rangle$
(22) $coef\,(:, n, w) \leftarrow \text{DFT}(coef\,(:, n, w))$
(23) **end for**
(24) **return** $coef$

can rewrite by the Poisson summation formula (7):

$$\begin{aligned} & c\,(m, n, w) \\ = \; & \sqrt{M} \mathcal{F}_M \left(\sum_{\tilde{m}=0}^{b-1} f(\cdot + \tilde{m}M, w) \overline{g\,(\cdot + \tilde{m}M - an)} \right)(m) \\ = \; & \left(\mathcal{F}_M K\,(\cdot, n, w)\right)(m)\,, \end{aligned} \tag{14}$$

where

$$K\,(j, n, w) = \sqrt{M} \sum_{\tilde{m}=0}^{b-1} f\,(j + \tilde{m}M, w)\, \overline{g}\,(j + \tilde{m}M - na)\,, \tag{15}$$

for $j \in \langle M \rangle$ and $n \in \langle N \rangle$. From (14) it can be seen that computing the DGT of a signal f can be done by computing K followed by DFTs along the first dimension of K.

To further lower the complexity of the algorithm, we wish to express the summation in (15) as a convolution.

We split j as $j = r + lc$ with $r \in \langle c \rangle$, $l \in \langle q \rangle$ and introduce $h_a, h_M \in \mathbb{Z}$ such that the following is satisfied:

$$c = h_M M - h_a a. \tag{16}$$

The two integers h_a, h_M can be found by the extended Euclid algorithm for computing the GCD of a and M.

Using (16) and the splitting of j we can express (15) as

$$\begin{aligned} & K\left(r+lc,n,w\right) \\ = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\tilde{m}M,w\right)\times \\ & \times\overline{g}\left(r+l\left(h_{M}M-h_{a}a\right)+\tilde{m}M-na\right) \end{aligned} \tag{17}$$

$$\begin{aligned} = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\tilde{m}M,w\right)\times \\ & \times\overline{g}\left(r+\left(\tilde{m}+lh_{M}\right)M-\left(n+lh_{a}\right)a\right) \end{aligned} \tag{18}$$

We substitute $\tilde{m}+lh_M$ by $\tilde{m}$ and $n+lh_a$ by n and get

$$\begin{aligned} & K\left(r+lc,n-lh_{a},w\right) \\ = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\left(\tilde{m}-lh_{M}\right)M,w\right)\times \\ & \times\overline{g}\left(r+\tilde{m}M-na\right) \end{aligned} \tag{19}$$

$$\begin{aligned} = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+\tilde{m}M+l\left(c-h_{M}M\right),w\right)\times \\ & \times\overline{g}\left(r+\tilde{m}M-na\right) \end{aligned} \tag{20}$$

We split $\tilde{m}=k+\tilde{s}p$ with $k\in\langle p\rangle$ and $\tilde{s}\in\langle d\rangle$ and $n=u+sq$ with $u\in\langle q\rangle$ and $s\in\langle d\rangle$ and use that $M=cq$, $a=cp$ and $c-h_{M}M=-h_{a}a$:

$$\begin{aligned} & K\left(r+lc,u+sq-lh_{a},w\right) \\ = & \sqrt{M}\sum_{k=0}^{p-1}\sum_{\tilde{s}=0}^{d-1} f\left(r+kM+\tilde{s}pM-lh_{a}a,w\right)\times \\ & \times\overline{g}\left(r+kM-ua+\left(\tilde{s}-s\right)pM\right) \end{aligned} \tag{21}$$

After having expressed the variables j, $\tilde{m}$, n using the variables r, s, $\tilde{s}$, k, l, u we have now indexed f using $\tilde{s}$ and g using $(\tilde{s}-s)$. This means that we can view the summation over $\tilde{s}$ as a convolution, which can be efficiently computed using a discrete Fourier transform. Define

$$\Psi_{r,s}^{f}\left(k,l+wq\right)=\mathcal{F}_{d}f\left(r+kM+\cdot pM-lh_{a}a,w\right), \tag{22}$$

$$\Phi_{r,s}^{g}\left(k,u\right)=\sqrt{M}\mathcal{F}_{d}g\left(r+kM+\cdot pM-ua\right), \tag{23}$$

Using (6) we can now write (21) as

$$\begin{aligned} & K\left(r+lc,u+\tilde{s}q-lh_{a},w\right) \\ = & \sqrt{d}\sum_{k=0}^{p-1}\mathcal{F}_{d}^{-1}\left(\Psi_{r,\cdot}^{f}\left(k,l+wq\right)\overline{\Phi_{r,\cdot}^{g}\left(k,u\right)}\right)\left(\tilde{s}\right) \end{aligned} \tag{24}$$

$$\begin{aligned} = & \sqrt{d}\mathcal{F}_{d}^{-1}\left(\sum_{k=0}^{p-1}\Psi_{r,\cdot}^{f}\left(k,l+wq\right)\overline{\Phi_{r,\cdot}^{g}\left(k,u\right)}\right)\left(\tilde{s}\right) \end{aligned} \tag{25}$$

If we consider $\Psi_{r,s}^{f}$ and $\Phi_{r,s}^{g}$ as matrices for each r and s, the sum over k in the last line can be written as matrix products. Algorithm 2 follows from this.

4. Running time

When computing the flop count of the algorithm, we will assume that a complex FFT of length M can be computed using $4M\log_2 M$ flops. A nice review of flop counts for FFT algorithms is presented in [14]. Table 1 shows the flop count for

TABLE 1. Flop counts

Alg.:	Flop count
Eq. (1)	$8MNL$
Eq. (14)	$8L\frac{L_g}{a} + 4NM\log_2(M)$
[1]	$L\left(8q+1+\frac{q}{p}\right) + 4L\left(1+\frac{q}{p}\right)\log_2 N + 4MN\log_2(M)$
Alg. 2	$L(8q) + 4L\left(1+\frac{q}{p}\right)\log_2 d + 4MN\log_2(M)$

Flop counts for 4 different way of computing the DGT: By the linear algebra definition (1), by the method based on Poisson summation (14), by the method of Bastiaans and Geilen from [1] and by Algorithm 2. The term L_g denotes the length of the window used so L_g/a is the overlapping factor of the window. Note for comparison that $\log_2 N = \log_2 d + \log_2 q$

Algorithm 2 and compares it with the definition of the DGT (1), with the algorithm for short windows using Poisson summation (14) and with the algorithm published in [1]. The algorithm by Prinz presented in [15] has the same computational complexity as the Poisson summation algorithm. For simplicity we assume that both the window and signal are complex valued. In the common case when both f and g are real-valued, all the algorithms will see a 2 to 4 times speedup.

The flop count for definition (1) is that of a complex matrix multiplication. All the other algorithms share the $4MN\log_2 M$ term coming from the application of an FFT to each 'block' of coefficients and only differ in how the application of the window is performed. The Poisson summation algorithm is very fast for a small overlapping factor L_g/a, but turns into an $\mathcal{O}\left(L^2\right)$ algorithm for a full length window. In this case the other algorithms have an advantage. The term $L\left(8q+1+\frac{q}{p}\right)$ in the [1] algorithm comes from calculation of the needed Zak-transforms, and the $4L\left(1+\frac{q}{p}\right)\log_2 N$ term comes from the transform to and from the Zak-domain. Compared to (22) and (23) this transformation uses longer FFTs. Algorithm 2 does away with the multiplication with complex exponentials in the [1] algorithm, and so the first term reduces to $L(8q)$.

Both the Poisson summation based algorithm and Algorithm 2 can do a DGT with $L \approx 2000000$ in less than 1 second on a standard PC at the time of writing. We have not created an efficient implementation of the algorithm from [1] in C so therefore we cannot reliably time it.

5. EXTENSIONS

The algorithm just developed can also be used to calculate the synthesis operator D_γ. This is done by applying Algorithm 2 in the reverse order and inverting each line. The only lines that are not trivially invertible are lines 10-12, which becomes

10) $\Gamma \leftarrow Phi^d(:,:,r,s)$
(11) $C \leftarrow Ctmp(:,:,s)$
(12) $Psitmp(:,:,s) \leftarrow \Gamma \cdot C$

where the matrices $Phi^d(:,:,r,s)$ should be left inverses of the matrices $Phi(:,:,r,s)$ for each r and s.

The matrices $Phi^d(:,:,r,s)$ can be computed by Algorithm 1 applied to a dual Gabor window γ of the Gabor frame (g,a,M). It also holds that all dual Gabor windows γ of a Gabor frame (g,a,M) must satisfy that $Phi^d(:,:,r,s)$ are left inverses of the matrices $Phi(:,:,r,s)$. This criterion was reported in [11, 12].

Algorithm 3 Canonical Gabor dual window

GABDUAL(g, a, M)

(1) $Phi =$WFAC(g, a, M)
(2) **for** $r = \langle c \rangle$, $s = \langle d \rangle$
(3) $\quad G \leftarrow Phi\,(:, :, r, s)$
(4) $\quad Phi^d\,(:, :, r, s) \leftarrow \left(G \cdot G^T\right)^{-1} \cdot G$
(5) **end for**
(6) $g^d =$IWFAC$\left(Phi^d, a, M\right)$
(7) **return** g^d

A special left-inverse in the *Moore-Penrose pseudo-inverse*. Taking the pseudo-inverses of $Phi\,(:, :, r, s)$ yields the factorization associated with the canonical dual window of (g, a, M), [3]. This is Algorithm 3. Taking the polar decomposition of each matrix in $\Phi^g_{r,s}$ yields a factorization of the canonical tight window (g, a, M). For more information on these methods, as well as iterative methods for computing the canonical dual/tight windows, see [13].

6. Special cases

We shall consider two special cases of the algorithm:

The first case is integer oversampling. When the redundancy is an integer then $p = 1$. Because of this we see that $c = a$ and $d = b$. This gives (16) the appearance

$$a = h_M q a - h_a a, \tag{26}$$

indicating that $h_M = 0$ and $h_a = -1$ solves the equation for all a and q. The algorithm simplifies accordingly, and reduces to the well known Zak-transform algorithm for this case, [10].

The second case is the short time Fourier transform. In this case $a = b = 1$, $M = N = L$, $c = d = 1$, $p = 1$, $q = L$ and as in the previous special case $h_M = 0$ and $h_a = -1$. In this case the algorithm reduces to the very simple and well known algorithm for computing the STFT.

7. Implementation

The reason for defining the algorithm on multi-signals, is that the multiple signals can be handled at once in the matrix product in line 12 of Algorithm 2. This is a matrix product of two matrices size $q \times p$ and $p \times qW$, so the second matrix grows when multiple signals are involved. Doing it this way reuses the $\Phi^g_{r,s}$ matrices as much as possible, and this is an advantage on standard, general purpose computers with a deep memory hierarchy, see [5, 18].

The benefit of expressing Algorithm 2 in terms of loops (as opposed to using the Zak transform or matrix factorizations) is that they are easy to reorder. The presented Algorithm 2 is just one among many possible algorithms depending on in which order the r, s, k and l loops are executed. For a given platform, it is difficult a priory to estimate which ordering of the loops will turn out to be the fastest. The ordering of the loops presented in Algorithm 2 is the variant that uses the least amount of extra memory.

Implementations of the algorithms described in this paper can be found in the Linear Time Frequency Toolbox (LTFAT) available from `http://ltfat.sourceforge.net`. The implementations are done in both the Matlab/Octave scripting language and in C. A range of different variants of Algorithm 2 has been implemented and tested, and the one found to be the fastest on a small range of computers is included in the toolbox.

References

[1] M. J. Bastiaans and M. C. Geilen. On the discrete Gabor transform and the discrete Zak transform. *Signal Process.*, 49(3):151–166, 1996.

[2] H. Bölcskei, F. Hlawatsch, and H. G. Feichtinger. Equivalence of DFT filter banks and Gabor expansions. In *SPIE 95, Wavelet Applications in Signal and Image Processing III*, volume 2569, part I, San Diego, july 1995.

[3] O. Christensen. Frames and pseudo-inverses. *J. Math. Anal. Appl.*, 195:401–414, 1995.

[4] O. Christensen. *An Introduction to Frames and Riesz Bases*. Birkhäuser, 2003.

[5] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16(1):1–17, 1990.

[6] H. G. Feichtinger and T. Strohmer, editors. *Gabor Analysis and Algorithms*. Birkhäuser, Boston, 1998.

[7] H. G. Feichtinger and T. Strohmer, editors. *Advances in Gabor Analysis*. Birkhäuser, 2003.

[8] G. H. Golub and C. F. van Loan. *Matrix computations, third edition*. John Hopkins University Press, 1996.

[9] K. Gröchenig. *Foundations of Time-Frequency Analysis*. Birkhäuser, 2001.

[10] A. J. E. M. Janssen. The Zak transform: a signal transform for sampled time-continuous signals. *Philips Journal of Research*, 43(1):23–69, 1988.

[11] A. J. E. M. Janssen. On rationally oversampled Weyl-Heisenberg frames. *Signal Process.*, pages 239–245, 1995.

[12] A. J. E. M. Janssen. The duality condition for Weyl-Heisenberg frames. In Feichtinger and Strohmer [6], chapter 1, pages 33–84.

[13] A. J. E. M. Janssen and P. L. Søndergaard. Iterative algorithms to approximate canonical Gabor windows: Computational aspects. *J. Fourier Anal. Appl.*, 13(2):211–241, 2007.

[14] S. Johnson and M. Frigo. A Modified Split-Radix FFT With Fewer Arithmetic Operations. *IEEE Trans. Signal Process.*, 55(1):111, 2007.

[15] P. Prinz. Calculating the dual Gabor window for general sampling sets. *IEEE Trans. Signal Process.*, 44(8):2078–2082, 1996.

[16] S. Qiu. Discrete Gabor transforms: The Gabor-gram matrix approach. *J. Fourier Anal. Appl.*, 4(1):1–17, 1998.

[17] T. Strohmer. Numerical algorithms for discrete Gabor expansions. In Feichtinger and Strohmer [6], chapter 8, pages 267–294.

[18] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimization of software and the ATLAS project. Technical Report UT-CS-00-448, University of Tennessee, Knoxville, TN, Sept. 2000.

[19] Y. Y. Zeevi and M. Zibulski. Oversampling in the Gabor scheme. *IEEE Trans. Signal Process.*, 41(8):2679–2687, 1993.