Fast and accurate computation of the Fourier transform of an image

Gregory Beylkin

University of Colorado at Boulder, Program in Applied Mathematics
Boulder, CO 80309-0526

ABSTRACT

We use the Battle-Lemarié scaling function in an algorithm for fast computation of the Fourier transform of a piecewise smooth function f. Namely, we compute for $-N \le m,n \le N$ (with a given accuracy ϵ) the integrals

$$\hat{f}(m,n) = \int_0^1 \int_0^1 f(x,y)\, e^{-2\pi i m x} e^{-2\pi i n y} dx\, dy \tag{0.1}$$

in $O(N_D)+O(N^2 \log N)$ operations, where N_D is the number of subdomains where the function f is smooth.

We consider an application of this algorithm to image processing. Notwithstanding that it might be advantageous to consider an image as a piecewise smooth function f, it is a common practice in image processing to simply take the FFT of the pixel values of the image in order to evaluate the Fourier transform. Due to the jump discontinuities of the function f, the accuracy of such a computation is poor.

We propose our algorithm as a tool for the accurate computation of the Fourier transform of an image since the direct evaluation of (0.1) is very costly.

1. IMAGE AS A PIECEWISE SMOOTH FUNCTION

It is natural and useful to consider an image as a piecewise smooth function. For example, the goal of segmentation algorithms is to find the boundaries of smooth subdomains of an image. Furthermore, at the level of pixels, one may consider an image as a collection of tiny squares with different values so that the total image is a linear combination of characteristic functions of elementary squares as in the example in Figure 3.

Yet, computing the Fourier transform of a piecewise smooth function is not an entirely trivial matter, especially if the number of discontinuities of f is large or the subdomains are complicated. If we use the FFT of the pixel values of an $N \times N$ image (which is equivalent to using the trapezoidal rule in (0.1)) then, in some directions, the error will decay only as $1/N$. In other words, instead of a piecewise smooth function, we work with an oscillatory function as in the example in Figure 1.

The cost of the direct evaluation of (0.1) for images is prohibitive. The direct algorithm for evaluating the Fourier transform of a linear combination of characteristic functions of elementary squares (or rectangles) computes $\hat{f}(m,n) = \sum_l \hat{f}_l(m,n)$, for $-N \le m,n \le N$, as a sum of contributions from each rectangle $[a_l, b_l] \times [c_l, d_l]$,

$$\hat{f}_l(m,n) = z_l \left(\frac{e^{-2\pi i m b_l} - e^{-2\pi i m a_l}}{-2\pi i m} \right) \left(\frac{e^{-2\pi i n d_l} - e^{-2\pi i n c_l}}{-2\pi i n} \right), \tag{1.2}$$

where z_l are constants, $l = 1, \ldots, N_D$ and N_D is the number of rectangles. Though accurate, such evaluation of the Fourier transform requires $O(N^2 \cdot N_D)$ operations and since typically for images $N_D \approx N^2$, the direct approach is not practical. Thus, there is a need for a fast and accurate algorithm to evaluate (0.1).

0-8194-1601-0/94/$6.00

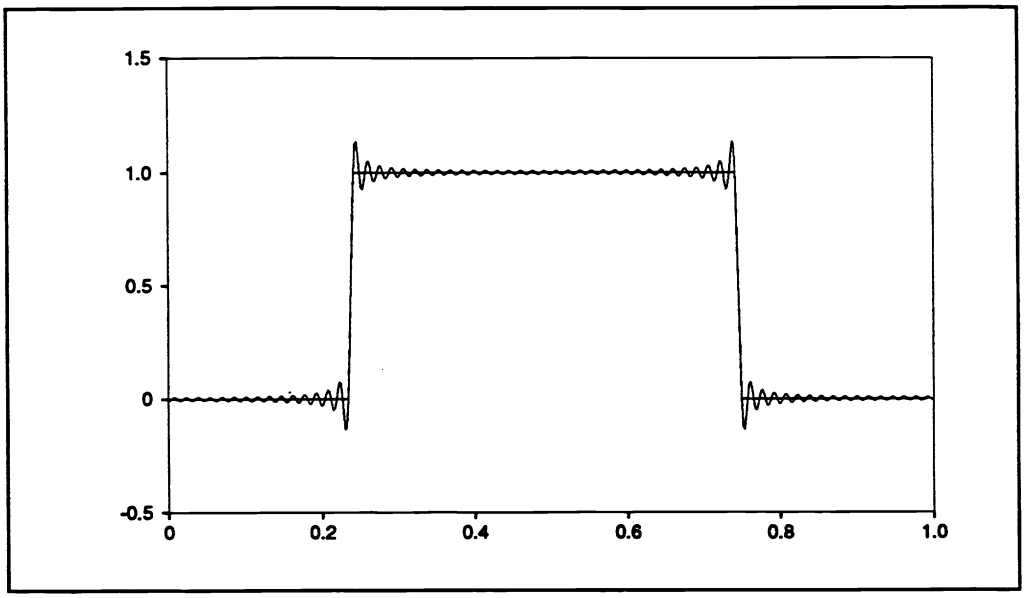

Figure 1: By taking the Fourier transform of pixel values we effectively replace the characteristic function of an interval by an oscillatory function with the same pixel values.

2. FAST ALGORITHM

The problem of computing (0.1) was first considered in [7] where it was motivated by the needs of VLSI design. The algorithm of [7] uses Gauss-Legendre quadratures to evaluate portions of (0.1) and Lagrange interpolation to redistribute the resulting values to an equally spaced grid. The result is then obtained by using the FFT.

In [2] the problem of fast computing of (0.1) is addressed by projecting f on a subspace of a Multiresolution Analysis (MRA) effectively bandlimiting the function f. The algorithm appears to be more efficient than that in [7]. In fact [2] considers a more general problem of computing the Fourier transform of generalized functions with singularities of the type $x_+^\lambda/\Gamma(\lambda+1)$, where Γ is the gamma function and x_+^λ is defined as x^λ for $x > 0$ and zero for $x \leq 0$. For example, $\lambda = 0$ yields the jump discontinuity, whereas $\lambda = -1$ corresponds to the δ-function. The algorithm in [2] allows us to evaluate the unequally spaced fast Fourier transform (see also [4]) as well as to compute integrals in (0.1).

We choose the MRA associated with spaces of polynomial splines. We take advantage of the properties of the Battle-Lemarié scaling function while computing integrals only with the B-splines. Such an approach is critical for the efficiency of the algorithm.

We note that though the analysis of splines is a well-established subject (see e.g. [6]), several families of bases were constructed only recently. The "spline family" of wavelets includes those constructed by Stromberg [8], Battle [1] and Lemarié [5], as well as non-orthogonal families (see [3] and [9]).

Let us start by introducing the integrals of f with the central B-splines $\beta^{(m)}(x)$ of odd order m,

$$f_k = \int_{-\infty}^{\infty} f(x)\, \beta_{kj}^{(m)}(x)\, \mathrm{d}x, \tag{2.3}$$

where $\beta_{kj}^{(m)}(x) = 2^{-j/2}\beta^{(m)}(2^{-j}x - k)$. Let $\hat{\beta}^{(m)}$ be the Fourier transform of $\beta^{(m)}$,

$$\hat{\beta}^{(m)}(\xi) = \left(\frac{\sin \pi\xi}{\pi\xi}\right)^{m+1}, \tag{2.4}$$

and consider the periodic function $a^{(m)}$,

$$a^{(m)}(\xi) = \sum_{l=-\infty}^{l=\infty} |\hat{\beta}^{(m)}(\xi + l)|^2 = \sum_{l=-m}^{l=m} \beta^{(2m+1)}(l)\ \mathrm{e}^{2\pi i l \xi}. \tag{2.5}$$

where $\beta^{(2m+1)}(l)$ are values of the central B-spline of order $2m+1$. We also need the Fourier transform of the Battle-Lemarié scaling function [5], [1],

$$\hat{\varphi}^{(m)}(\xi) = \frac{\hat{\beta}^{(m)}(\xi)}{\sqrt{a^{(m)}(\xi)}}. \tag{2.6}$$

Our approach is based the following

Theorem 1 *Let E_∞ be the error of the approximation of the Fourier transform $\hat{f}$ of the generalized function f by the periodic function $2^{j/2}F(\xi)/\sqrt{a^{(m)}(\xi)}$,*

$$E_\infty = \sup_{|\xi|\le\alpha} |2^{j/2}\frac{F(\xi)}{\sqrt{a^{(m)}(\xi)}} - \hat{f}(2^{-j}\xi)| \ / \ \sup_{|\xi|\le\alpha} |\hat{f}(2^{-j}\xi)|, \quad j < 0, \tag{2.7}$$

where α is a parameter, $a^{(m)}$ is given in (2.5) and F is the Fourier series,

$$F(\xi) = \sum_{k\in\mathbf{Z}} f_k \mathrm{e}^{-2\pi i k \xi}, \tag{2.8}$$

with coefficients f_k given in (2.3).

1. *If*

$$|\hat{f}(\xi)| \le C(1+|\xi|)^\sigma, \quad \sigma < m,$$

then we have

$$E_\infty \le \frac{1}{2\hat{\varphi}(\alpha) - 1}\left[1 - \hat{\varphi}(\alpha) + \frac{1}{C_{\hat{f}}(0,\alpha)} \sum_{l=\pm1,\pm2\ldots} C_{\hat{f}}(l,\alpha)\ \left(\frac{\alpha}{|l|-\alpha}\right)^{m+1}\right], \tag{2.9}$$

where

$$C_{\hat{f}}(l,\alpha) = \sup_{|\xi|\le\alpha} |\hat{f}(2^{-j}(\xi + l))|. \tag{2.10}$$

2. *For any $\epsilon > 0$ we may choose the order m of the central B-spline and $\alpha > 0$ so that for $|\xi| \le \alpha$*

$$E_\infty \le \epsilon. \tag{2.11}$$

The proof of Theorem may be found in [2] where it is also shown that

$$2^{j/2}\frac{F(\xi)}{\sqrt{a^{(m)}(\xi)}} = \sum_{l\in\mathbf{Z}} \hat{f}(2^{-j}(\xi+l))\,\hat{\varphi}^{(m)}(\xi+l) = \hat{f}(2^{-j}\xi)\,\hat{\varphi}^{(m)}(\xi) + \sum_{l=\pm1,\pm2,\ldots} \hat{f}(2^{-j}(\xi+l))\,\hat{\varphi}^{(m)}(\xi+l) \tag{2.12}$$

As an illustration of why the left hand side of (2.12) is a good approximation of $\hat{f}(2^{-j}\xi)$ for $|\xi| \le \alpha$, we plot $\hat{\varphi}^{(m)}(\xi)$, $\hat{\varphi}^{(m)}(\xi+1)$ and $\hat{\varphi}^{(m)}(\xi-1)$ for $m = 23$ in Figure 2. We note that for $\alpha = 1/4$ and $|\xi| \le \alpha$ the

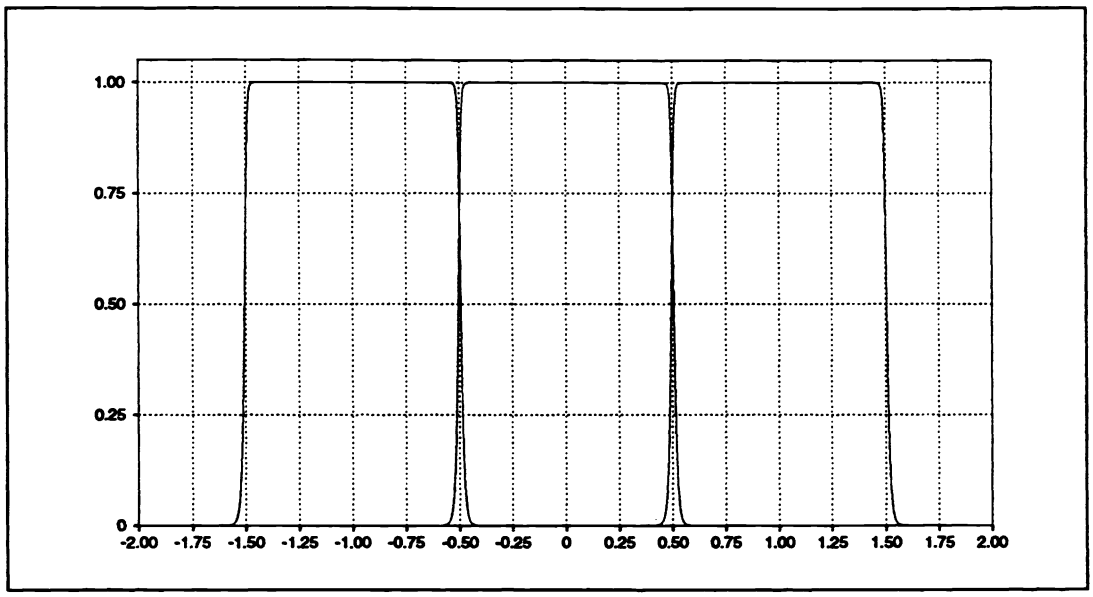

Figure 2: The Fourier transform of Battle-Lemarié scaling function of order $m = 23$. Shown are functions $\hat{\varphi}^{(m)}(\xi)$, $\hat{\varphi}^{(m)}(\xi + 1)$ and $\hat{\varphi}^{(m)}(\xi - 1)$.

values of $\hat{\varphi}^{(m)}(\xi)$ are equal to one (with double precision accuracy), whereas $\hat{\varphi}^{(m)}(\xi + 1)$ and $\hat{\varphi}^{(m)}(\xi - 1)$ are equal to zero with the same precision.

Setting $\alpha = 1/4$ and using Theorem 1, we have (with accuracy ϵ)

$$\hat{f}(l) = \frac{1}{L^{1/2}\sqrt{a^{(m)}(l/L)}} \sum_{k \in \mathbf{Z}} f_k \mathrm{e}^{-2\pi i k l/L}, \tag{2.13}$$

for $-L/4 \le l \le L/4$, where $L = 2^{-j}$. We may always arrange $f_k = 0$ for $k < 0$ and $k \ge L$. Thus, we replace the series in (2.13) by a finite sum and obtain

$$\hat{f}(l) = \frac{1}{L^{1/2}\sqrt{a^{(m)}(l/L)}} \sum_{k=0}^{L-1} f_k \mathrm{e}^{-2\pi i k l/L}, \quad -L/4 \le l \le L/4, \tag{2.14}$$

which may be evaluated using the FFT.

For computing the Fourier transform of an image, we have (setting $\alpha = 1/4$)

$$\hat{f}(n, n') = \frac{1}{N\sqrt{a^{(m)}(n/N)\, a^{(m)}(n'/N)}} \sum_{k=0}^{N-1} \sum_{k'=0}^{N-1} f_{kk'}\ \mathrm{e}^{-2\pi i k n/N} \mathrm{e}^{-2\pi i k' n'/N}, \tag{2.15}$$

for $-N/4 \le n, n' \le N/4$, $N = 2^{-j}$, and

$$f_{kk'} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, \beta_{kj}^{(m)}(x)\, \beta_{k'j}^{(m)}(y)\, \mathrm{d}x \mathrm{d}y. \tag{2.16}$$

The accuracy is controled by an appropriate choice of the order of the spline m.

As a result we have a simple algorithm based on the approximation in Theorem 1 which consists of three steps

1. Computing integrals (2.16). The cost of generating the matrix $f_{kk'}$ is proportional to $m^2 N_D$. The spline order m is usually chosen proportional to $-\log \epsilon$, the number of accurate digits.

2. Computing the FFT of the matrix $f_{kk'}$ in order to compute the sum in (2.15). This step requires $O(\frac{1}{\alpha^2} N^2 \log N)$ operations.

3. Multiplication by the pre-computed factor $\sqrt{a^{(m)}(n/N)\, a^{(m)}(n'/N)}$ in (2.15) to obtain the result. At this step we effectively generate a representation involving the Battle-Lemarié scaling function. The third step requires $O(N^2)$ operations and its cost is negligible if compared to the first two steps.

For given accuracy, m is proportional to α and the total cost may be estimated as

$$C_1 m^2 N_D + C_2 \frac{1}{m^2} N^2 \log N + C_3 N^2,$$

where C_i, $i = 1, 2, 3$ are constants which depend on implementation. The choice of spline order m and the parameter α may be used to optimize the performance of the algorithm.

In order to compute integrals in (2.16) we take advantage of the properties of the B-splines. As piecewise polynomials, the central B-splines may be written as

$$\beta^{(m)}(x) = \sum_{l=0}^{m+1} (-1)^l \begin{pmatrix} m+1 \\ l \end{pmatrix} \frac{(x + \frac{m+1}{2} - l)_+^m}{m!}, \tag{2.17}$$

where x_+^m is defined as x^m for $x > 0$ and zero for $x \leq 0$. Using (2.17) for computing (2.16) reduces the problem to that of evaluating integrals of f with polynomials.

Alternatively, for several important functions (e.g., the characteristic function of a rectangle) the integrals with the B-splines may be computed analytically and expressed in terms of the values of B-splines (of higher order). The values of the B-splines may be obtained using recursion over the spline order,

$$\beta^{(m)}(x) = \frac{(m+1)/2 + x}{m}\, \beta^{(m-1)}(x + 1/2) + \frac{(m+1)/2 - x}{m}\, \beta^{(m-1)}(x - 1/2), \tag{2.18}$$

where $m = 1, 2, \ldots$ and $\beta^{(0)}(x)$ is the characteristic function of the interval $[-1/2, 1/2]$. In our implementation we used (2.18).

Remark. We may use Theorem 1 in a region $|\xi| \leq \alpha$, where $\alpha < 1/4$. Choosing a smaller α permits us to choose a lower order B-spline to achieve given precision and, thus, decreases the number of operations necessary to project f. On the other hand, it increases the number of operations necessary to compute (2.14) due to the larger oversampling factor. The choice of $\alpha = 1/4$ results in the oversampling factor of 2 which we use in our numerical experiments.

3. NUMERICAL EXPERIMENTS

The algorithm has been implemented in FORTRAN 77 and numerical experiments have been carried out on SPARC-10 workstation. All computations were performed in double precision.

For the purposes of image processing the run times presented below should be considered only as preliminary, demonstrating the proper complexity of the algorithm. We note that in order to achieve accuracy sufficient in image processing it is enough to use lower order splines, e.g. $m = 3, \ldots, 9$ whereas in the examples we use $m = 23$. Very significant simplifications such as computing in single precision, tabulating spline

N	T_p	T_{FFT}	T_m	Error E_∞	T_{tot}	T_{dir}
64	0.06	0.50	0.02	4.4e-15	0.58	0.47
128	0.17	2.88	0.07	2.4e-15	3.12	1.89
256	0.60	12.25	0.31	1.3e-15	13.16	7.58
512	2.28	54.32	1.23	1.0e-15	57.83	30.41

Table 1: Timing and accuracy for Example III.1 on SPARC-10

values instead of computing them, etc., will undoubtedly further improve the performance of the algorithm. These issues will be addresed elsewhere.

In what follows we denote by T_p the run time for the projection step of the algorithm where we compute the integrals in (2.16). T_{FFT} denotes the run time for the FFT step, and T_m for the third (multiplication) step of the algorithm. The error E_∞ is the maximal absolute error among all computed frequencies obtained by comparing the output of the algorithm with that of the direct evaluation. T_{tot} denotes the total run time, $T_{tot} = T_p + T_{FFT} + T_m$. For comparison T_{SOR} denotes the run time from [7] for a similar experiment. T_{dir} denotes the run time required for the direct computation.

Example III.1. We compute the Fourier transform of the function f which is a constant and has a rectangle of area ≈ 0.64 as its support. We compare the run time of our algorithm with that of the direct evaluation for $N = 2^n$, $n = 6, 7, 8, 9$ and report the results in Table 1. We observe that the direct algorithm for a single rectangle is faster only by a factor of 2.

Example III.2. In this example we consider f to be a linear combination of characteristic functions of a pseudo-random combination of 1225 rectangles of the total area of ≈ 0.64 and the perimeter ≈ 112. Each rectangle was projected separately. The results are shown in Table 2. For $N = 2^n$, $n = 6, 7$ we compare the run time and accuracy with that of the direct algorithm. We note that in this example our algorithm is dominated by the FFT step. The speed up factor compared with the direct evaluation is ≈ 500.

We also compare the run times with the algorithm in [7]. The run times of the algorithm in [7] are modified by a factor 0.21 which was obtained by comparing with the run times of the direct evaluation. We observe that the speed up is at least by a factor of 10. Our algorithm also requires 4 times less memory. Comparison with the direct evaluation is for illustration purposes only. [1]

Example III.3. In this example f is a linear combination of characteristic functions of a pseudo-random combination of $40,000$ rectangles of the total area of ≈ 0.64 and the perimeter ≈ 640. The results are shown in Table 3. As individual rectangles become smaller, we start to observe that the first step of the algorithm practically does not depend on the number of frequencies (i.e., size of the matrix).

Example III.4. In this example f is a linear combination of characteristic functions of a pseudo-random combination of $160,000$ rectangles of the total area of ≈ 0.64 and the perimeter ≈ 1280. The results are shown in Table 4. Again we observe that T_p practically does not change as N changes from 64 to 512.

[1] In [7] integration by parts is used to reduce the integral over the domain to that over the boundary. A similar approach may be taken here as well with some reduction in the number of operations in the projection step of the algorithm.

N	T_p	T_{FFT}	T_m	Error E_∞	T_{tot}	$T_{SOR} * 0.21$	T_{dir}
64	2.00	0.50	0.02	4.0e-15	2.51	11.55	581.75
128	2.46	2.85	0.07	2.2e-15	5.38	44.73	2,331.8
256	3.30	12.47	0.31		16.08	210.84	9,324 (est.)
512	6.17	53.84	1.24		61.25	NA	37,252 (est.)

Table 2: Timing and accuracy for Example III.2

N	T_p	T_{FFT}	T_m	$T_{tot}(sec)$	Estimate of T_{dir} in hours
64	60.76	0.49	0.02	61.27	5.2 h
128	62.45	2.87	0.07	65.39	21 h
256	64.88	12.23	0.31	77.42	84.2 h
512	72.19	54.33	1.26	127.78	337.9 h

Table 3: Timing for Example III.3 (40,000 rectangles)

N	T_p	T_{FFT}	T_m	T_{tot} (sec)	Estimate of T_{dir} in hours
64	242.58	0.51	0.02	243.11	20.9 h
128	248.57	2.92	0.08	251.57	84 h
256	250.80	12.53	0.32	263.65	336.9 h
512	261.52	55.93	1.22	318.67	1,351.6 h

Table 4: Timing for Example III.4 (160,000 rectangles)

SPIE

Figure 3: Letters SPIE as an image constructed from elementary squares.

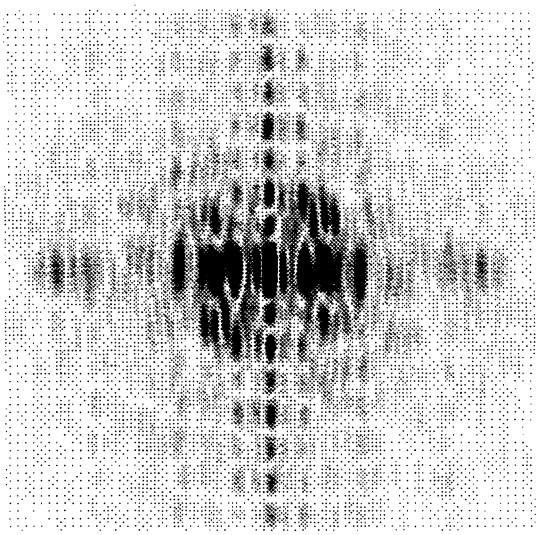

Figure 4: The absolute value of the Fourier transform of the image in Figure 3

Example III.5. For illustration purposes, we compute the Fourier Transform of the image contaning the abbreviation "SPIE" as it is shown in Figure 3. The result is displayed in Figure 3. This set of letters consists of 376 elementary squares which were projected individually.

4. ACKNOWLEDGMENTS

This research was partially supported by ARPA grant F49620-93-1-0474 and ONR grant N00014-91-J4037.

5. REFERENCES

[1] G. Battle. A block spin construction of ondelettes. Part i: Lemarié functions. *Comm. Math. Phys.*, 110:601–615, 1987.

[2] G. Beylkin. On fast Fourier transform of functions with singularities. *submitted to Applied and Computational Harmonic Analysis*, 1994

[3] C. Chui. *An Introduction to Wavelets.* Academic Press, 1992.

[4] A. Dutt and V. Rokhlin. Fast Fourier Transform for Nonequispaced Data. *SIAM J. Sci. Stat. Comp.*, 1993. to appear.

[5] P.G. Lemarié. Ondelettes à localisation exponentielles. *J. Math. Pures et Appl.*, 67:227–236, 1988.

[6] I.J. Schoenberg. *Cardinal Spline Interpolation.* SIAM, 1973. CBMS-NSF Series in Applied Math. #12.

[7] E. Sorets. Fast Fourier Transform of Piecewise Constant Functions. 1993. Yale University Research Report, YALEU/DCS/RR-986.

[8] J. O. Stromberg. A Modified Franklin System and Higher-Order Spline Systems on $\mathbf{R}^n$ as Unconditional Bases for Hardy Spaces. In *Conference in harmonic analysis in honor of Antoni Zygmund, Wadworth math. series*, pages 475–493, 1983.

[9] M. Unser and A. Aldroubi. Polynomial splines and wavelets – a signal processing perspective. In C. Chui, editor, *Wavelets: A Tutorial in Theory and Applications*, pages 91–122. Academic Press, 1992.