COMM. MATH. SCI.
Vol. 5, No. 4, pp. 981–998

EMPIRICAL EVALUATION OF A SUB-LINEAR TIME SPARSE DFT ALGORITHM*

M.A. IWEN†, A. GILBERT‡, AND M. STRAUSS§

Abstract. In this paper we empirically evaluate a recently proposed Fast Approximate Discrete Fourier Transform (FADFT) algorithm, FADFT-2, for the first time. FADFT-2 returns approximate Fourier representations for frequency-sparse signals and works by random sampling. Its implementation is benchmarked against two competing methods. The first is the popular exact FFT implementation FFTW Version 3.1. The second is an implementation of FADFT-2's ancestor, FADFT-1. Experiments verify the theoretical runtimes of both FADFT-1 and FADFT-2. In doing so it is shown that FADFT-2 not only generally outperforms FADFT-1 on all but the sparsest signals, but is also significantly faster than FFTW 3.1 on large sparse signals. Furthermore, it is demonstrated that FADFT-2 is indistinguishable from FADFT-1 in terms of noise tolerance despite FADFT-2's better execution time.

Key words. sub-linear time algorithms, Fourier transforms, compressive sensing

AMS subject classifications. 65T50, 68W40, 68W25, 65T40

1. Introduction

The Discrete Fourier Transform (DFT) for real/complex valued signals is utilized in myriad applications, as is the Fast Fourier Transform (FFT) [5], a model divide-and-conquer algorithm used to quickly compute a signal's DFT. The FFT reduces the time required to compute a length N signal's DFT from $O(N^2)$ to $O(N\log(N))$. Although an impressive achievement, for huge signals (i.e., N large) the FFT can still be computationally infeasible. This is especially true when the FFT is repeatedly utilized as a subroutine by more complex algorithms for large signals.

In some signal processing applications [14, 13] and numerical methods for multiscale problems [6, 12], only the top few most energetic terms of a very large signal/solution's DFT may be of interest. In such applications the FFT, which computes all DFT terms, is computationally wasteful. This was the motivation behind the development of FADFT-2 [11] and its predecessor FADFT-1 [10]. Given a length N signal and a user-provided number m, both of the FADFT algorithms output high fidelity estimates of the signal's m most energetic DFT terms. Furthermore, both FADFT algorithms have a runtime which is primarily dependent on m (largely independent of the signal size N). FADFT-1 and FADFT-2 allow any large frequency-sparse (e.g., smooth, or C^∞) signal's DFT to be approximated with little dependence on the signal's mode distribution and relative frequency sizes.

Related work to FADFT-1/2 includes sparse signal (including Fourier) reconstruction methods via Basis Pursuit and Orthogonal Matching Pursuit [4, 19]. These methods, referred to as "compressive sensing" methods, require a small number of measurements (i.e., $O(m \text{ polylog } N)$ samples [18]) from an N-length m-frequency

*Received: June 6, 2007; accepted (in revised version): September 28, 2007. Communicated by Olof Runborg.

†University of Michigan Department of Mathematics, East Hall, 530 Church Street, Ann Arbor, MI 48109-1043 (markiwen@umich.edu). Supported in part by NSF DMS-0510203.

‡University of Michigan Department of Mathematics, East Hall, 530 Church Street, Ann Arbor, MI 48109-1043 (annacg@umich.edu). Supported in part by NSF DMS-0354600 and DARPA/ONR N66001-06-1-2011.

§University of Michigan Department of Mathematics, East Hall, 530 Church Street, Ann Arbor, MI 48109-1043 (martinjs@umich.edu). Supported in part by NSF DMS-0510203.

Algorithm	Implementation	Output for length N signal	Run Time
FFT [5]	FFTW 3.1 [9]	Full DFT of length N signal	$O(N\log(N))$
FADFT-1* [20]	RAℓSFA [20]	m most energetic DFT terms	$O(m^2 \cdot \mathrm{polylog}(N))$
FADFT-1 [10]	AAFFT 0.5	m most energetic DFT terms	$O(m^2 \cdot \mathrm{polylog}(N))$
FADFT-2 [11]	AAFFT 0.9	m most energetic DFT terms	$O(m \cdot \mathrm{polylog}(N))$

Table 1.1. *Algorithms and implementations*

sparse signal in order to calculate its DFT with high probability. Hence, compressive sensing is potentially useful in applications such as MRI imaging where sampling costs are high [16, 17]. However, despite the small number of required samples, current compressive sensing DFTs are more computationally expensive than FFTs such as FFTW 3.1 [9] for all signal sizes and nontrivial sparsity levels. To the best of our knowledge FADFT-1 and 2 are alone in being competitive with FFT algorithms in terms of frequency-sparse DFT run times.

A variant of the FADFT-1 algorithm, FADFT-1*, has been implemented and empirically evaluated [20]. However, no such evaluation has yet been performed for FADFT-2. In this paper FADFT-2 is empirically evaluated against both FADFT-1 and FFTW 3.1 [9]. During the course of the evaluation it is demonstrated that FADFT-2 is faster than FADFT-1 while otherwise maintaining essentially identical behavior in terms of noise tolerance and approximation error. Furthermore, it's shown that both FADFT-1 and 2 can outperform FFTW 3.1 at finding a small number of a large signal's top-magnitude DFT terms. See Table 1.1 for descriptions/comparisons of all the algorithms mentioned in this paper.

The main contributions of this paper are:

1. We introduce the first publicly available implementation of FADFT-2, the Ann Arbor Fast Fourier Transform (AAFFT) 0.9, as well as AAFFT 0.5, the first publicly available implementation of FADFT-1.
2. Using AAFFT 0.9 we perform the first empirical evaluation of FADFT-2. The evaluation demonstrates that FADFT-2 is generally superior to FADFT-1 in terms of runtime while maintaining similar noise tolerance and approximation error characteristics. Furthermore, we see that both FADFT algorithms outperform FFTW 3.1 on large sparse signals.
3. In the course of benchmarking FADFT-2 we perform a more thorough evaluation of the one dimensional FADFT-1 algorithm than previously completed.

The remainder of this paper is organized as follows. First, in Section 2, we introduce relevant background material and present a short introduction to both FADFT-1 and FADFT-2. Then, in Section 3, we present an empirical evaluation of our new FADFT implementations, AAFFT 0.5/0.9. During the course of our Section 3.1 evaluation we investigate how AAFFT's runtime varies with signal size and degree of sparsity. Furthermore, we present results on AAFFT's accuracy-versus-runtime trade-off. Next, in Section 3.2, we study AAFFT's noise tolerance and its dependence on signal size, the signal to noise ratio, and the number of signal samples used. Finally, we conclude with a short discussion in Section 4.

2. Preliminaries

Throughout the remainder of this paper we will be interested in complex-valued signals (or arrays) of length N. We shall denote such signals by $\mathbf{A}$, where $\mathbf{A}(j) \in \mathbb{C}$

is the signal's j^{th} complex value for all $j \in [0, N-1] \subset \mathbb{N}$. Hereafter we will refer to the process of either calculating, measuring, or retrieving from machine memory any $\mathbf{A}(j) \in \mathbb{C}$ as *sampling* from $\mathbf{A}$. Given a signal $\mathbf{A}$ we define its discrete L^2-norm, or Euclidean norm, to be

$$\|\mathbf{A}\|_2 = \sqrt{\sum_{j=0}^{N-1} |\mathbf{A}(j)|^2}.$$

We will also refer to $\|\mathbf{A}\|_2^2$ as $\mathbf{A}$'s energy.

For any signal, $\mathbf{A}$, its Discrete Fourier Transform (DFT), denoted $\widehat{\mathbf{A}}$, is another signal of length N defined as follows:

$$\widehat{\mathbf{A}}(\omega) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{-2\pi i \omega j}{N}} \mathbf{A}(j), \quad \forall \omega \in [0, N-1].$$

Furthermore, we may recover $\mathbf{A}$ from its DFT via the Inverse Discrete Fourier Transform (IDFT) as follows:

$$\mathbf{A}(j) = \widehat{\widehat{\mathbf{A}}}^{-1}(j) = \frac{1}{\sqrt{N}} \sum_{\omega=0}^{N-1} e^{\frac{2\pi i \omega j}{N}} \widehat{\mathbf{A}}(\omega), \quad \forall j \in [0, N-1].$$

We will refer to any index, ω, of $\widehat{\mathbf{A}}$ as a frequency. Furthermore, we will refer to $\widehat{\mathbf{A}}(\omega)$ as frequency ω's coefficient for each $\omega \in [0, N-1]$. Parseval's equality tells us that $\|\widehat{\mathbf{A}}\|_2 = \|\mathbf{A}\|_2$ for any signal. In other words, the DFT preserves Euclidean norm and energy. Note that any non-zero coefficient frequency will contribute to $\widehat{\mathbf{A}}$'s energy. Hence, we will also refer to $|\widehat{\mathbf{A}}(\omega)|^2$ as frequency ω's energy. If $|\widehat{\mathbf{A}}(\omega)|$ is relatively large we wil say that ω is energetic.

We will also refer to three other common discrete signal quantities besides the Euclidean norm throughout the remainder of this paper. The first is the L^1, or taxi-cab, norm. The L^1-norm of a signal $\mathbf{A}$ is defined to be

$$\|\mathbf{A}\|_1 = \sum_{j=0}^{N-1} |\mathbf{A}(j)|.$$

The second discrete quantity is the L^∞ value of a signal. The L^∞ value of a signal $\mathbf{A}$ is defined to be

$$\|\mathbf{A}\|_\infty = \max\{|\mathbf{A}(j)|, j \in [0, N-1]\}.$$

Finally, the third common discrete signal quantity is the signal-to-noise ratio, or SNR, of a signal. In some situations it is beneficial to view a signal, $\mathbf{A}$, as consisting of two parts: a meaningful signal, $\tilde{\mathbf{A}}$, with added noise, $\mathbf{G}$. In these situations, when we have $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{G}$, we define $\mathbf{A}$'s signal-to-noise ratio, or SNR, to be

$$\mathrm{SNR}(\mathbf{A}) = 20 \cdot \log_{10}\left(\frac{\|\tilde{\mathbf{A}}\|_2}{\|\mathbf{G}\|_2}\right).$$

Both FADFT algorithms produce output of the form $(\omega_1, C_1), \ldots, (\omega_m, C_m)$, where each $(\omega_j, C_j) \in [0, N-1] \times \mathbb{C}$. We will refer to any such set of $m < N$ tuples

$$\{(\omega_j, C_j) \in [0, N-1] \times \mathbb{C} \text{ s.t. } 1 \le j \le m\}$$

as a **sparse Fourier representation** and denote it with a superscript ‘s’. Note that if we are given a sparse Fourier representation, $\widehat{\mathbf{R}}^{s}$, we may consider $\widehat{\mathbf{R}}^{s}$ to be a length-N signal. We simply view $\widehat{\mathbf{R}}^{s}$ as the N length signal

$$\widehat{\mathbf{R}}(j) = \begin{cases} C_j & \text{if } (j, C_j) \in \widehat{\mathbf{R}}^{s} \\ 0 & \text{otherwise} \end{cases}$$

for all $j \in [0, N-1]$. Using this idea we may, for example, compute $\mathbf{R}$ from $\widehat{\mathbf{R}}^{s}$ via the IDFT.

We continue with one final definition: An m-term/tuple sparse Fourier representation is m-optimal for a signal $\mathbf{A}$ if it contains the m most energetic frequencies of $\widehat{\mathbf{A}}$ along with their coefficients. More precisely, we’ll say that a sparse Fourier representation

$$\widehat{\mathbf{R}}^{s} = \{(\omega_j, C_j) \in [0, N-1] \times \mathbb{C} \text{ s.t. } 1 \leq j \leq m\}$$

is m**-optimal** for $\mathbf{A}$ if there exists a valid ordering of $\widehat{\mathbf{A}}$’s coefficients by magnitude

$$|\widehat{\mathbf{A}}(k_1)| \geq |\widehat{\mathbf{A}}(k_2)| \geq \cdots \geq |\widehat{\mathbf{A}}(k_j)| \geq \cdots \geq |\widehat{\mathbf{A}}(k_N)|$$

so that $(k_l, \widehat{\mathbf{A}}(k_l)) \in \widehat{\mathbf{R}}^{s}$ for all $l \in [1, m]$. Note that a signal may have several m-optimal Fourier representations if its frequency coefficient magnitudes are non-unique. For example, there are two 1-optimal sparse Fourier representations for the signal

$$\mathbf{A}(j) = 2e^{\frac{2\pi i j}{N}} + 2e^{\frac{4\pi i j}{N}}, \; N > 2.$$

However, all m-optimal $\widehat{\mathbf{R}}^{s}$ for any signal $\mathbf{A}$ will always have both the same unique $\|\mathbf{R}\|_2$ and $\|\mathbf{A} - \mathbf{R}\|_2$ values.

Given an input signal, $\mathbf{A}$, the purpose of both FADFT-1 and FADFT-2 is to identify the m most energetic frequencies, $\omega_1 \leq \cdots \leq \omega_m$, from $\widehat{\mathbf{A}}$ and approximate their coefficients. Put another way, the goals of both FADFT-1 and FADFT-2 are as follows: Given an input signal, $\mathbf{A}$, both FADFT-1 and FADFT-2 are designed to output an approximate m-optimal sparse Fourier representation for $\mathbf{A}$.

2.1. FADFT-1 algorithm. The main result of [10] is an algorithm, FADFT-1, with the following properties: Denote an m-optimal Fourier representation of a one dimensional signal $\mathbf{A}$ of length N by $\widehat{\mathbf{R}}^{s}_{\text{opt}}$, and assume that, for some M, we have

$$\frac{1}{M} \leq \| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2 \leq \| \mathbf{A} \|_2 \leq M.$$

Then, the FADFT-1 algorithm uses time and space $m^2 \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$ to return a sparse Fourier representation $\widehat{\mathbf{R}}^{s}$ such that

$$\| \mathbf{A} - \mathbf{R} \|_2^2 \leq (1 + \epsilon) \| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2^2$$

with probability at least $1 - \delta$.

Note that for $m \ll N$ the FADFT-1 algorithm is sub-linear time. Also note that the ϵ and δ parameters allow the user to manage approximation error and failure probability, respectively. For a pseudo-code outline of FADFT-1/2 see Algorithm 1.

FADFT-1 is a randomized greedy pursuit algorithm which, in this case, means that it iteratively produces approximations to an $\widehat{\mathbf{R}}^{s}_{\text{opt}}$ which get better with high

Algorithm 1 FADFT-1/2 Algorithm

1: **Input:** Signal $\mathbf{A}$, Number of most energetic frequencies m, Approximation error ϵ, Failure Probability δ
2: **Output:** An approximate m-optimal sparse Fourier representation for $\mathbf{A}$
3: Set sparse Fourier representation, $\widehat{\mathbf{R}}^{\text{s}}$, to $\emptyset$.
4: Set energetic frequencies, I, to $\emptyset$.
5: **for all** $i \leftarrow 0$ **to** $O\left(\frac{\log(\frac{M}{\epsilon})}{\epsilon^2}\right)$ **do**
6: Find a list, L, of energetic frequencies ω with $|(\widehat{\mathbf{A}}-\widehat{\mathbf{R}})(\omega)|^2 \geq O\left(\frac{\epsilon^2}{m}\right) \cdot \|\mathbf{A}-\mathbf{R}\|_2^2$.
7: Set $I = I \cup L$.
8: Update $\widehat{\mathbf{R}}^{\text{s}}$ by estimating coef.s $\forall \omega \in I$ s.t. $|(\widehat{\mathbf{A}}-\widehat{\mathbf{R}})(\omega)|^2 \leq O\left(\frac{\epsilon^2}{|I|+m}\right) \cdot \|\mathbf{A}-\mathbf{A}_I\|_2^2$.
9: **end for**
10: Output top m terms of $\widehat{\mathbf{R}}^{\text{s}}$.

probability as time goes on (i.e., i increases in Algorithm 1). Intuitively, Algorithm 1's step 6 will discover frequencies in $\mathbf{A}-\mathbf{R}$ with large magnitudes relative to $\|\mathbf{A}-\mathbf{R}\|_2^2$ with high probability (the larger the frequency's magnitude, the better chance it will be found). Hence, as long as step 8 continues to estimate frequency coefficients to high enough precision, important frequencies which haven't been detected yet will become increasingly overwhelming in $\mathbf{A}-\mathbf{R}$ as $\|\mathbf{A}-\mathbf{R}\|_2^2$ shrinks (i.e., as $\widehat{\mathbf{R}}^{\text{s}} \rightarrow$ an $\widehat{\mathbf{R}}^{\text{s}}_{\text{opt}}$). The end result is that it becomes increasingly difficult for the top m frequencies in $\mathbf{A}$ to evade detection as time goes on. If the search continues long enough they will all be found with high probability.

2.2. FADFT-2 algorithm. The FADFT-2 Algorithm [11] is identical to the FADFT-1 algorithm with two main exceptions. First, FADFT-2 utilizes a faster method of coefficient estimation (Algorithm 1's step 8) than FADFT-1 does. Second, FADFT-2 also samples from intermediate sparse representations via a faster algorithm than the naive method used by FADFT-1. In order to better understand the differences between FADFT-1 and FADFT-2, we next compare how both algorithms perform coefficient estimation. We refer the reader to [10, 11] for more detailed descriptions of each algorithm's energetic frequency isolation and identification (i.e., Algorithm 1's step 6) methods.

2.2.1. FADFT-1 coefficient estimation. As before, let $\mathbf{A}$ be a given input signal of length N. Furthermore, suppose that we've identified an energetic frequency, ω_{big}, whose value we wish to estimate. Independently choose two uniformly random integers $c, l \in [0, N-1]$, making sure that l is invertible *mod* N. We can now estimate ω_{big}'s coefficient by computing the following sum:

$$\widehat{\mathbf{A}}'(\omega_{\text{big}}) = \frac{\sqrt{N}}{K} \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega_{\text{big}}(c+l\cdot k)}{N}} A(c+l\cdot k) \tag{2.1}$$

where $K \ll N$ will be specified later. Here we have $\mathbf{E}[\widehat{\mathbf{A}}'(\omega_{\text{big}})] = \widehat{\mathbf{A}}(\omega_{\text{big}})$ and $\text{Var}[\widehat{\mathbf{A}}'(\omega_{\text{big}})]$ is $O\left(\frac{\|\mathbf{A}\|_2^2}{K}\right)$. Hence, if we let K be $O(\frac{1}{\nu})$ we'll have

$$|\widehat{\mathbf{A}}'(\omega_{\text{big}}) - \widehat{\mathbf{A}}(\omega_{\text{big}})|^2 < \nu \|\mathbf{A}\|_2^2$$

with constant probability by the Markov inequality. If we next approximate $\widehat{\mathbf{A}}(\omega_{\text{big}})$ by taking the median of $E = O(\log(\frac{1}{\delta}))$ copies of independent and identically distributed $\widehat{\mathbf{A}}'(\omega_{\text{big}})$'s, the Chernoff inequality tells us we will achieve precision $\nu \| \mathbf{A} \|_2^2$ with probability $\geq 1-\delta$. See [10] for details.

Note that K is proportional to the desired number of most energetic frequencies, m (FADFT-1 step 8 requires that frequency coefficients are estimated with accuracy $O\left(\frac{\epsilon^2}{|I|+m}\right) \cdot \| \mathbf{A} - \mathbf{A}_I \|_2^2$). Furthermore, we will need to estimate coefficients for at least m frequencies. Hence, the time to find an m-term Fourier representation using this coefficient estimation method as stated will be proportional to m^2. Fortunately there are also $O(m \cdot \text{polylog}(m))$-time methods for calculating m coefficients to within the same precision.

2.2.2. FADFT-2 coefficient estimation. The main difference between FADFT-1 and FADFT-2 is that FADFT-2 utilizes Unequally Spaced Fast Fourier Transform (USFFT) techniques [2, 7, 8, 15] both to sample from sparse representations and to perform coefficient estimation (i.e., compute Equation 2.1 for m frequencies) in $O(m \cdot \text{polylog}(m))$-time. In this way FADFT-2 is able to avoid all FADFT-1's O(m^2)-time Fourier matrix multiplications. A brief explanation of how FADFT-2 utilizes an USFFT along the lines of [2] to perform coefficient estimation follows. An analogous method allows FADFT-2 to sample m values from the inverse transform of an m-term Fourier representation in time proportional to $m \cdot \text{polylog}(m)$.

Suppose we want to estimate the coefficients of m frequencies $\omega_1, \ldots, \omega_m$ in an input signal $\mathbf{A}$ of length N. Independently choose two uniformly random integers $c, l \in [0, N-1]$, making sure that l is invertible *mod* N. In order to estimate the m frequencies' coefficients we need to calculate Equation 2.1 for $j = 1, \ldots, m$:

$$\widehat{\mathbf{A}}'(\omega_j) = \frac{\sqrt{N}}{K} \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega_j (c+l\cdot k)}{N}} A(c+l\cdot k) \, = \, \frac{\sqrt{N}}{K} e^{\frac{-2\pi i \omega_j c}{N}} f(\omega_j') \text{ for } j = 1, \ldots, m,$$

where we let $f(\omega) = \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega k}{N}} A(c+l\cdot k)$ and $\omega_j' = \omega_j \cdot l$. Now, let $R = 8\cdot K$ and define r_ω to be the integer $r \in [0, R]$ that minimizes $|\omega - \frac{r\cdot N}{R}|$. Expanding f in a Taylor series we see that

$$f(\omega_j) = f(r_{\omega_j} N/R) + f'(r_{\omega_j} N/R) \cdot \Delta_{\omega_j} + f''(r_{\omega_j} N/R) \cdot (\Delta_{\omega_j}^2/2) + \cdots,$$

where

$$\Delta_{\omega_j} = \omega_j - r_{\omega_j} \frac{N}{R} \text{ for each j.}$$

Calculating the derivatives and setting $a_k = A(c+l\cdot k)$ for $k \in [0, K-1]$ (0 otherwise), we get that

$$f(\omega_j) = \left(\sum_{k=0}^{R-1} a_k e^{-2\pi i r_{\omega_j} k/R} \right) + \frac{-2\pi i \Delta_{\omega_j}}{N} \cdot \left(\sum_{k=0}^{R-1} a_k k e^{-2\pi i r_{\omega_j} k/R} \right) +$$

$$2 \left(\frac{-\pi i \Delta_{\omega_j}}{N} \right)^2 \cdot \left(\sum_{k=0}^{R-1} a_k k^2 e^{-2\pi i r_{\omega_j} k/R} \right) + \cdots.$$

Each sum in the expression above may be calculated for all r_{ω_j} simultaneously in time $O(K \log(K))$ via the FFT. And, since $|\frac{-2\pi i \Delta_{\omega_j}}{N}| \cdot K < \frac{1}{2}$, we only need $O(\log(\frac{1}{\nu}))$

such sums to get $\pm\nu \| \mathbf{A} \|_2^2$ precision. The upshot is that we only need time $O\big(m \cdot \text{polylog}(m) \log(\frac{1}{\nu})\big)$ in order to estimate the coefficients of ω_1 through ω_m.

It is important to note that in Equation 2.1 $\mathbf{A}$ is sampled along an arithmetic progression. The k^{th} sample is at location $c+l\cdot k$. It is exactly sampling $\mathbf{A}$ in this fashion that allows USFFT techniques to be utilized. To the best of our knowledge all known USFFT methods require either frequencies or sample positions to be represented as an arithmetic progression. Depending on the user's ability to dictate what samples are used, this may or may not be a weakness of FADFT-2.

2.2.3. FADFT-2 result. The main result of [11] is that FADFT-2 has the following properties: Denote an m-optimal Fourier representation of a one dimensional signal $\mathbf{A}$ of length N by $\widehat{\mathbf{R}}^{\text{s}}_{\text{opt}}$ and assume that, for some M, we have

$$\frac{1}{M} \le \| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2 \le \| \mathbf{A} \|_2 \le M.$$

Then, the FADFT-2 algorithm uses time and space $m \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$ to return a Fourier representation $\widehat{\mathbf{R}}^{\text{s}}$ such that

$$\| \mathbf{A} - \mathbf{R} \|_2^2 \le (1+\epsilon) \| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2^2$$

with probability at least $1-\delta$. When working to double (i.e., 64-bit) precision it should be safe to assume

$$M \approx \max(10^{16}, \| \mathbf{A} \|_2).$$

In other words, even if $\mathbf{A}$ is an exact superposition (e.g., a sinusoid), machine noise (i.e., roundoff errors) will generally limit the accuracy of our m-optimal Fourier representation $\widehat{\mathbf{R}}^{\text{s}}_{\text{opt}}$.

Note that this result indicates FADFT-2 is essentially *linear* in m as opposed to FADFT-1 which is *quadratic*. Second, it is important to note that FADFT-2 is designed to quickly output a high fidelity approximation to FADFT-1's output for any given input signal without having to utilize any extra information (e.g., signal samples). Hence, if given good parameter settings and a frequency-sparse input signal, both versions of FADFT should yield approximately the same output.

3. FADFT implementation and evaluation

Both FADFT-1 and FADFT-2 were implemented in C++ utilizing the Standard Template Library (for readability). Hereafter these implementations will be referred to as different versions of the Ann Arbor Fast Fourier Transform (AAFFT). Version 0.5 of AAFFT is the straightforward implementation of the FADFT-1 algorithm with quadratic time in m, the desired number of largest Fourier terms. Version 0.9 of AAFFT is an implementation of FADFT-2. All AAFFT source code and documentation is available at [1].

Calculating the optimal m-term Fourier representation for a length-N signal may be done naively by computing the entire DFT and then reporting its largest m Fourier terms. This naive approach requires time $O(N\log(N))$ using an FFT implementation. Given the absence of other fast competitors, below we benchmark AAFFT 0.5 and AAFFT 0.9 against this naive approach with FFTW Version 3.1 [9] serving as the FFT implementation. All experiments were carried out on a dual 3.6 GHz processor multi-threaded Dell desktop with 3G of memory. Below FFTW will always refer to FFTW Version 3.1 using an FFTW_ESTIMATE In_Place_Transform_Data plan. In order to help us remain as unbiased as possible we don't include any sorting or non-zero coefficient search time in FFTW's reported run times below. All reported signal sizes are powers of 2.

It is important to note that both AAFFT implementations rely on 20 different user-provided parameter settings that influence approximation error, runtime, the number of signal samples utilized, memory usage, etc.. For the sake of readability we only mention individual parameters in subsequent sections when absolutely necessary. Instead, we will report observable consequences of various parameter settings (e.g., runtime, approximation error, etc.) without providing detailed descriptions of what parameter settings produced them. For a detailed discussion of all AAFFT parameters, along with example parameter settings used for various experiments below, we invite the reader to go to http://www-personal.umich.edu/∼markiwen/respub.htm and download AAFFT.tgz (compressed tar file). Besides the AAFFT source code, AAFFT.tgz contains a file called README.pdf which contains detailed parameter information.

3.1. Empirical evaluation: run time and accuracy.

Run Time: In Figure 3.1 we report how AAFFT's run time changes with input signal size. The 10 reported signal sizes for each implementation are $2^{17}, 2^{18}, \ldots, 2^{26}$. The run time reported at each signal size for each implementation is the average of 1000 test signal DFT times. It is important to remember that AAFFT is randomized and approximate, so the run time depends on how much error the user is willing to tolerate. Parameters for both AAFFT implementations were chosen so that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations was between 10^{-5} and 10^{-7} at each signal size.

The test signals were randomly generated 60-frequency exact superpositions. Hence, m was fixed to 60 for all the AAFFT runs used to create Figure 3.1. The magnitude of each non-zero frequency was 1 so that all frequencies were of the same importance. This is the most difficult type of sparse signal for AAFFT since the energetic frequency isolation and identification portion of the FADFT algorithm works best at finding single frequencies larger than all others. For each of the 1000 test superpositions we generated 60 integers $\omega_1, \ldots, \omega_{60} \in [0, N-1]$ and 60 phases $p_1, \ldots, p_{60} \in [0, 2\pi]$ uniformly at random. We then set the test signal, $\mathbf{A}$, to be

$$\mathbf{A}(x) = \frac{1}{\sqrt{N}} \sum_{j=1}^{60} e^{2\pi i p_j} e^{\frac{2\pi i \omega_j x}{N}} \;\; \forall x \in [0, N-1].$$

In Figure 3.1 below we graph the maximum, minimum, and mean run times for FFTW 3.1, AAFFT 0.5, and AAFFT 0.9 over the 1000 test signals at each signal size. At each data point the top and bottom of the point's vertical line gives the associated implementation's maximum and minimum run times, respectively. The data point itself is located at the associated implementation's mean run time. Note in Figure 3.1 below that both AAFFT 0.9 and AAFFT 0.5 have relatively constant run times despite being randomized.

Recall, AAFFT 0.9's theoretical run time is $m \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$, where m is the number of desired output representation terms, $1-\delta$ is the probability of achieving multiplicative error bound ϵ, M is a bound for the signal's energy, and N is the signal size. Similarly, AAFFT 0.5's theoretical run time is $m^2 \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$. Figure 3.1's run times were generated from sparse

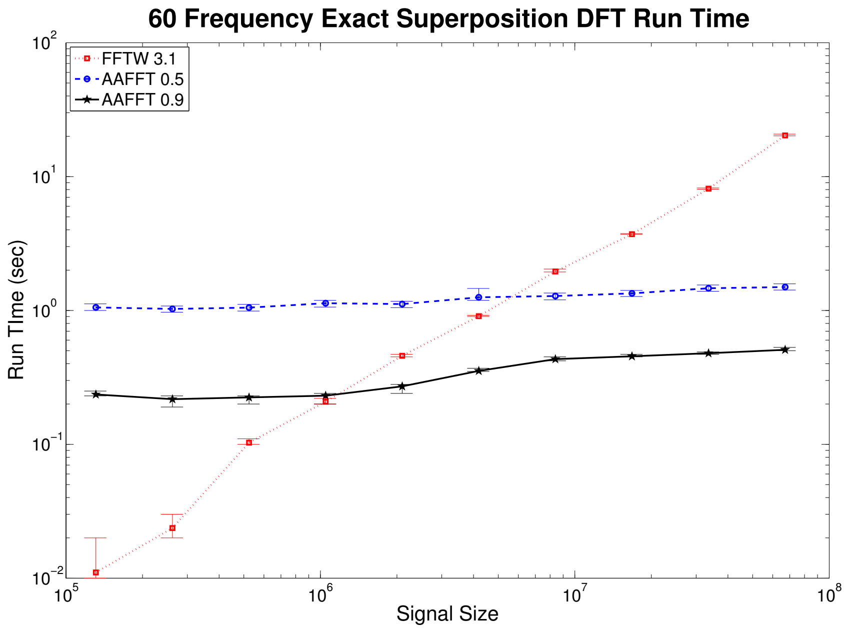


FIG. 3.1. *AAFFT run time Vs. signal size*

exact 60 superpositions with all terms magnitude 1 so that m and M remained fixed for all experiments. Furthermore, requiring that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations be between 10^{-5} and 10^{-7} at each signal size kept δ and ϵ fairly stable. Hence, we expect the run times of AAFFT 0.5 and AAFFT 0.9 to increase with signal size like polylog(N). Our expectation does appear to be realized in Figure 3.1, where we see the run times of both AAFFT 0.9 and AAFFT 0.5 gently increase with N. Note that AAFFT 0.9 is faster than AAFFT 0.5 for all signal sizes when $m = 60$. Figure 3.1 also contains a graph of FFTW 3.1's run times, which appear to increase something like the expected $O(N \log N)$. Note that for signal sizes $> 2^{20}(i.e., 1,048,576)$ AAFFT 0.9 is faster at recovering an exact 60 frequency superposition than FFTW 3.1. Similarly, AAFFT 0.5 begins to beat FFTW 3.1 at signal sizes $\geq 2^{23}(i.e., 8,388,608)$.

In the group of tests used to produce Figure 3.2 below we held the signal size N constant at $2^{22} = 4,194,304$ and varied m. As before, at each reported number of superposition frequencies we graph the maximum, minimum, and mean run times for FFTW 3.1, AAFFT 0.5, and AAFFT 0.9 over the 1000 tests. Each test run was performed on a randomly-generated test m-superposition similar to above. For a fixed m we create each test signal by generating m integers $\omega_1, \ldots, \omega_m \in [0, N-1]$ and m random phases $p_1, \ldots, p_m \in [0, 2\pi]$. We then set the the test signal, $\mathbf{A}$, to be $\mathbf{A}(x) = \frac{1}{2048}\sum_{j=1}^{m} e^{2\pi i p_j} e^{\frac{2\pi i \omega_j x}{N}}$ $\forall x \in [0, 4194303]$. Again, as above, we required that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations was between 10^{-5} and 10^{-7} at each superposition size m. The end result is that we expect little dependence on M, N, ϵ, and δ in our AAFFT runtime results.

As expected, AAFFT 0.5 displays quadratic run time in m while AAFFT 0.9's

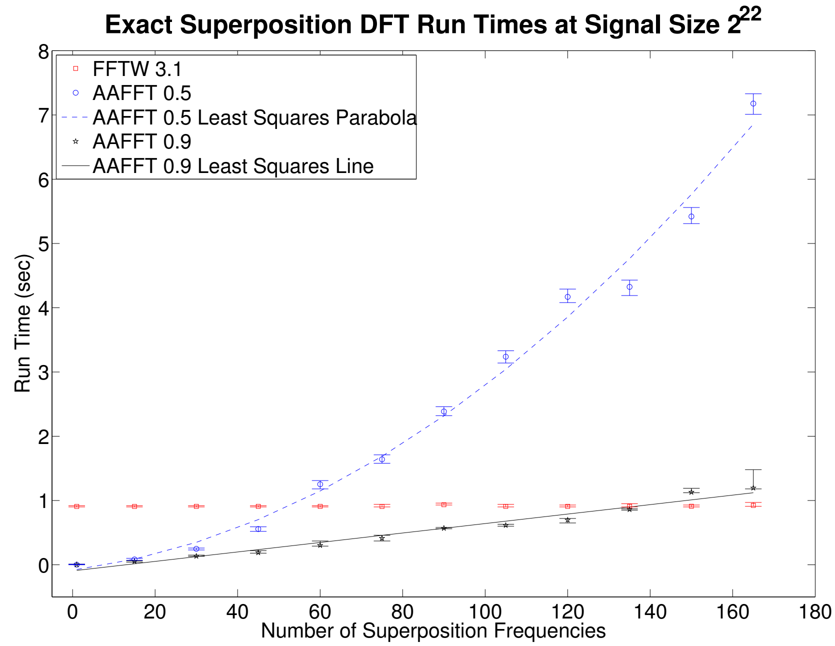


FIG. 3.2. *AAFFT run time vs. superposition size*

run time looks linear. Also, not surprisingly, FFTW 3.1's run time is essentially constant. Note that AAFFT 0.9 can recover superpositions with ≤ 135 frequencies more quickly than FFTW at signal size 2^{22}. Meanwhile, AAFFT 0.5 is only capable of computing ≤ 45-sparse signals more quickly then FFTW. Also notice that AAFFT 0.9 is competitive with AAFFT 0.5 for all values of m. AAFFT 0.5 is, on average, slightly faster than AAFFT 0.9 for small frequency (e.g., $m=1$) superpositions. This is due to AAFFT 0.5's naive $\tilde{O}(m^2)$-time coefficient estimation and sparse Fourier representation sampling methods having a smaller constant runtime factor than the USFFT techniques that AAFFT 0.9 employs. However, for all $m \geq 15$, AAFFT 0.9's $O(m \cdot \text{polylog}(m))$-time USFFT techniques outperform AAFFT 0.5's straightforward (I)DFT methods. Hence, AAFFT 0.9 is generally faster than AAFFT 0.5 for all values of $m \geq 15$.

Approximation Error: When using AAFFT for numerical analysis applications one may desire greater average accuracy than the 5 or 6 digits per term guaranteed above. Hence, we next present some results concerning AAFFT's accuracy versus run time trade-offs. As before, every Figure 3.3 data point results from 1000 runs on randomly generated 60-superpositions whose frequencies each have magnitude 1 with random phase. Furthermore, the signal sizes, N, are once again fixed to 2^{22} for every trial run.

Recall that AAFFT 0.9 frequency coefficient estimation (as well as representation sampling) is carried out by using truncated Taylor series with T terms in order to calculate multiple frequencies' coefficient estimates at once (see Section 2.2.2). Also recall that for each identified frequency, ω_{big}, the median of E such coefficient estimates becomes ω_{big}'s coefficient update for each round of the program (see Section 2.2.1).

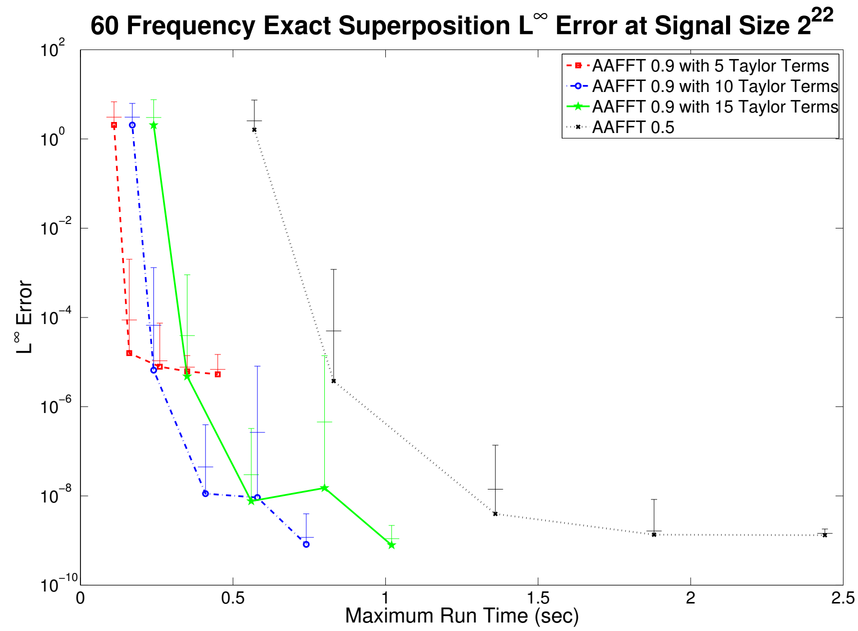


FIG. 3.3. *AAFFT error vs. parameters*

In general, the larger T and E are the more accurate and reliable the final frequency coefficient estimates should be. Note that AAFFT 0.5 works in the same way except that Taylor series are not used. Thus, AAFFT 0.5 does not depend on T.

In Figure 3.3 we investigated the effect of varying E and T on AAFFT 0.5/0.9's accuracy. All other parameters were held fixed. To create Figure 3.3 we varied E for AAFFT 0.5 and three different T-valued AAFFT 0.9 variants (with $T = 5$, 10, and 15). The mean, mean + 1 standard deviation, and maximum L^∞ approximation error values over each of five 1000 run trials (with $E = 1$, 3, 5, 7, and 9) were graphed for all 4 AAFFT versions. In order to give a better idea of AAFFT's approximation error versus run time trade offs, the L^∞ values were graphed against their associated trial's maximum run time for each data point.

As expected, the runtime (and, generally, accuracy) of all 4 AAFFT variants increased monotonically with E. Hence, for each of the 4 curves in Figure 3.3 the upper-left-most data point corresponds to $E=1$, the second-most-upper-left data point to $E=3$, etc. Also as expected, we can see that both AAFFT 0.9's accuracy and runtime tend to increase with T. The 5 Taylor term variant of AAFFT 0.9 is only accurate to about 10^{-5} despite the number of medians used. On the other hand, the 10-Taylor-term AAFFT 0.9 variant is comparable in accuracy to both AAFFT 0.5 and the 15-Taylor-term AAFFT 0.9 variant for each E value. Furthermore, we can see that AAFFT 0.9 with 10 Taylor terms appears to be faster than both AAFFT 0.5 and AAFFT 0.9 with 15 Taylor terms.

Both AAFFT 0.5/0.9 and FFTW 3.1 utilize double precision (i.e., 64-bit) arithmetic/variables. Hence, for Figure 3.3's experiments FFTW 3.1 always reported frequency coefficients that were accurate to within 10^{-15}. Looking at Figure 3.3 above it appears as if AAFFT 0.5/0.9's average worst-case frequency coefficient estimates are only accurate to within about 10^{-9} at best. However, we expect to get better accuracy by increasing AAFFT's K parameter (see Section 2.2.1's Equation 2.1) which was fixed at 128 during these experiments [1]. For example, in the extreme case where K is increased to N, we can expect that AAFFT 0.5/0.9 will calculate each energetic frequency's coefficient to within about 10^{-12} or better. More generally, as K is increased toward N we expect AAFFT's accuracy (and run time) to increase also. However, testing the limits of AAFFT's accuracy is left as future work.

3.2. Empirical evaluation: noise tolerance and sampling complexity.

Noise Tolerance: Our next series of experiments report on the noise tolerance of both the AAFFT 0.9 and AAFFT 0.5 implementations. In order to determine each implementation's level of noise tolerance we will work with signals consisting of a single non-zero coefficient frequency buried in complex Gaussian noise. Given such signals we will try to determine how the noise level influences AAFFT's ability to recover the hidden frequency. In essence, we wish to investigate AAFFT's utility as a denoising tool.

Below we work exclusively with signals consisting of a single non-zero frequency signal, $\tilde{\mathbf{A}}$, in Gaussian noise. Let N be our signal size. Then,

$$\tilde{\mathbf{A}}(x) = Ce^{2\pi i p} \cdot e^{\frac{2\pi i \omega x}{N}} \quad \forall x \in [0, N-1],$$

where $C \in \mathcal{R}^+$ is chosen to control the signal to noise ratio, p is a uniformly random phase in $[0, 2\pi]$, and ω is a uniformly random frequency in $[0, N-1]$. As above, we generate a new $\tilde{\mathbf{A}}$ for every AAFFT trial run.

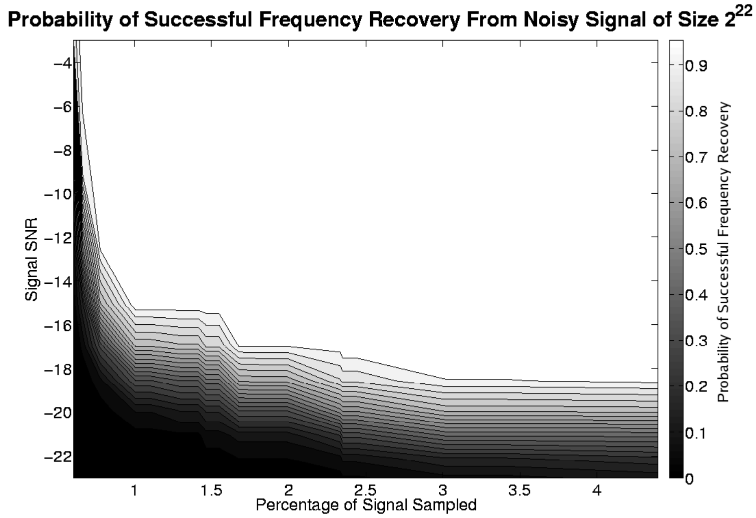
Furthermore, in all subsequent experiments each trial run's Gaussian noise is (re)generated each run by adding standard (i.e., mean 0, variance 1) normally distributed values independently to both the real and imaginary components of every element of the complex hidden signal $\tilde{\mathbf{A}}$. All normally distributed values are generated by the Polar Box-Muller method [3]. For the remainder of this paper we'll denote the noise added to $\tilde{\mathbf{A}}(x)$ by $\mathbf{G}(x)$ $\forall x \in [0, N-1]$. Hence, every trial run's input signal, $\mathbf{A}$, is of the form $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{G}$. The signal to noise ratio, or SNR, of $\mathbf{A}$ is $20 \cdot \log_{10}\left(\frac{\|\tilde{\mathbf{A}}\|_2}{\|\mathbf{G}\|_2}\right)$. Furthermore, for fixed $\tilde{\mathbf{A}}$, note that

$$\min\left\{k \in [0, N-1] \text{ s.t. } |\widehat{\mathbf{A}}(k)| = \|\widehat{\mathbf{A}}\|_\infty\right\}$$

is $\tilde{\mathbf{A}}$'s single nonzero frequency with high probability (depending on the SNR).

For a fixed m, increasing $\mathbf{A}$'s SNR will tend to increase $\| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2^2$. Hence, looking back at Sections 2.1/2.2.3, we will have weaker accuracy guarantees for the m-term Fourier representations returned by AAFFT 0.5/0.9 as SNR increases. If the accuracy guarantees become weak enough we won't even be able to expect AAFFT to correctly discover which $\widehat{\mathbf{A}}$ frequencies are most energetic. Thus, increasing $\mathbf{A}$'s SNR generally requires us to both increase m and/or decrease ϵ in order to properly determine, and then estimate the coefficients of, $\mathbf{A}$'s most energetic DFT modes. Therefore, the higher the SNR, the more samples and run time AAFFT will need in order to recover our $\widehat{\tilde{\mathbf{A}}}$ frequency with high probability.

Figure 3.4 investigates the probability of AAFFT 0.9 and 0.5 successfully recovering an input signal $\mathbf{A}$'s smallest DFT frequency of largest coefficient magnitude. Each Figure 3.4 graph was generated using 200 three-dimensional (i.e., # AAFFT



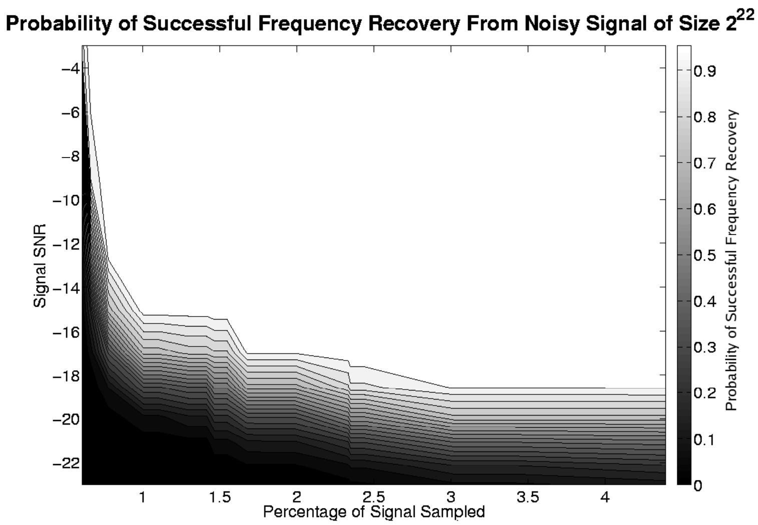


FIG. 3.4. *Probability of hidden signal recovery for AAFFT 0.5 (top) and AAFFT 0.9 (bottom)*

sample points $\times$ average $\mathbf{A}$ SNR $\times$ success probability) data points. Each data point was generated via 1000 AAFFT trial runs. The signal size, N, of all data points' trial signals was fixed at 2^{22}.

Every Figure 3.4 data point had its 1000 runs' input signals' (i.e., $\mathbf{A}$'s) SNR values controlled through the use of a uniform magnitude value, C, over its 1000 randomly generated single-frequency $\tilde{\mathbf{A}}$s. Though new Gaussian noise was generated with every run, each data point's 1000 input signal SNRs were tightly grouped around the mean SNR (standard deviation from each of the 200 data point's reported mean SNRs was < 0.0025). Each data point plots the mean SNR value of its 1000 associated runs against each Figure 3.4 plot's vertical axis.

Note that the sub-linear run times of AAFFT 0.9 and AAFFT 0.5 necessitate that

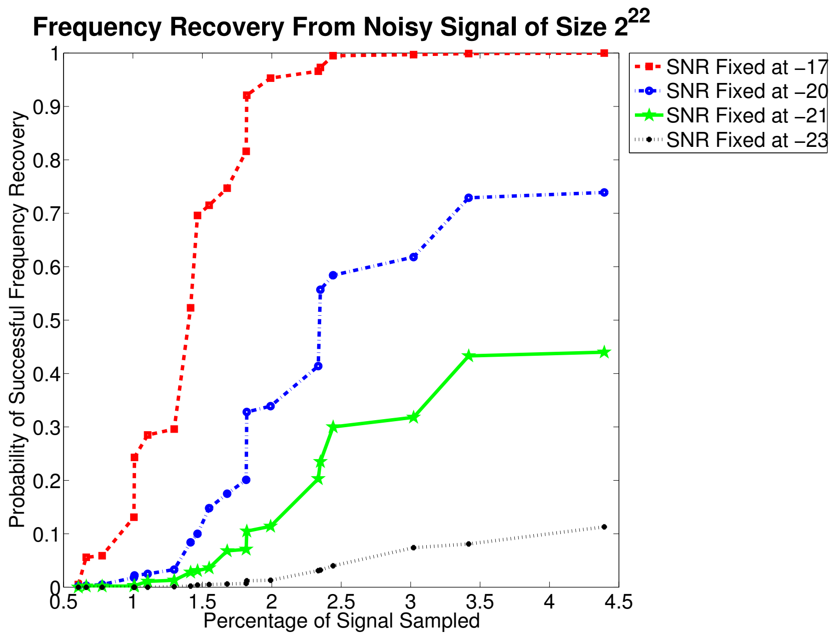


FIG. 3.5. *AAFFT 0.9's probability of hidden signal recovery from signals with noise*

neither method can read the entire input signal $\mathbf{A}$. During Figure 3.4's experiments the number of samples used by both AAFFT 0.9 and 0.5 depended deterministically on a subset of parameter settings common to both implementations. For each of the 200 data points making up Figure 3.4 a uniform set of parameters were used across each point's 1000 trial runs. The number of signal samples resulting from each data point's parameters (listed as a percentage of N) is plotted against Figure 3.4's horizontal axis.

Each Figure 3.4 plot's color/shade at any (percent sampled x, average SNR y) pair indicates the probability of AAFFT 0.9/0.5 successfully determining the smallest frequency, k, so that $|\widehat{\mathbf{A}}(k)| = \|\widehat{\mathbf{A}}\|_\infty$ for a trial signal $\mathbf{A}$ with SNR y if AAFFT 0.9/0.5 is only allowed to use $\frac{x \cdot N}{100}$ samples from $\mathbf{A}$. For each data point the probability of success was calculated from its 1000 trial runs by counting the number of times AAFFT 0.9/0.5 returned the same minimum largest-magnitude frequency as FFTW 3.1, divided by 1000. Figure 3.4's color bars indicate how the gray-scale values in each graph correspond to success probabilities. Lighter values indicate high success probabilities while darker values indicated lower success probabilities.

Looking at Figure 3.4 we can see that there is no significant difference between the performance of AAFFT 0.5 and 0.9 on noisy signals. This is unsurprising given that AAFFT 0.9 was, in essence, designed to quickly return a high fidelity approximation to AAFFT 0.5's output for any given input signal without using additional samples. Thus, we'll concentrate on AAFFT 0.9's noise tolerance results for the remainder of this section.

Figure 3.4's AAFFT 0.9 graph (bottom graph) behaves as expected. If we fix any SNR value we can see that increasing the number of samples that AAFFT 0.9 uses

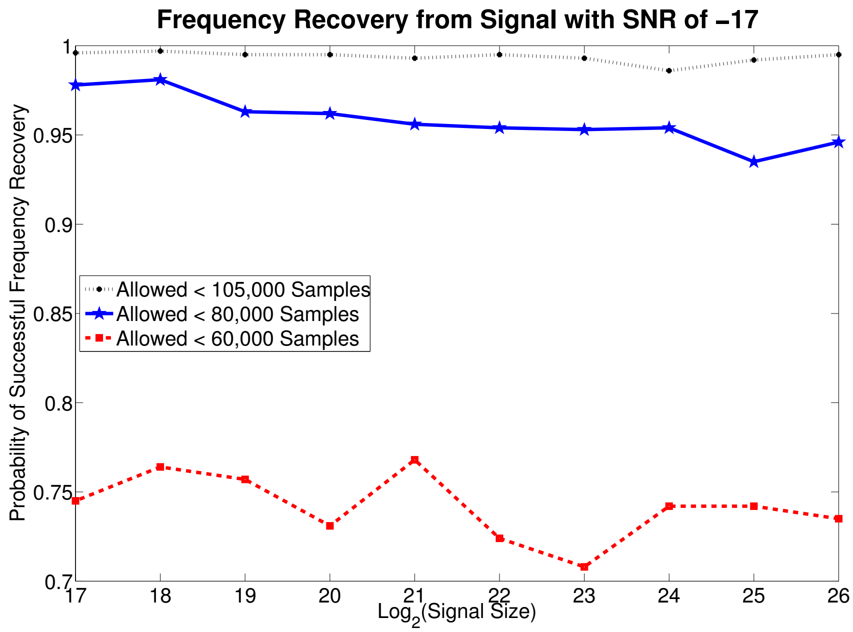


FIG. 3.6. *Signal size vs. probability of hidden signal recovery for AAFFT 0.9*

allows an increase in success probability. In effect, the shading lightens from left to right along any SNR line. Similarly, if we fix the number of AAFFT 0.9 samples and increase the SNR the shading lightens (i.e., success probability increases). In general, Figure 3.4 indicates that AAFFT tolerates small amounts of noise (SNR > -15) well as long as it is allowed to use $> 42{,}000$ samples ($> 1\%$ of 2^{22}).

In order to more clearly see AAFFT 0.9's noise tolerance results for lower SNR values, we present Figure 3.5. Figure 3.5 shows four fixed SNR success probability curves from Figure 3.4's AAFFT 0.9 graph. Again, as expected, Figure 3.5 demonstrates that AAFFT 0.9 is more tolerant of smaller levels of noise than larger levels (i.e., larger-SNR-value curves are higher than smaller-SNR-value curves). Furthermore, each SNR curve increases with increasing AAFFT sample usage. Looking at the -17 SNR curve, it appears as if AAFFT 0.9 will always successfully locate the smallest high energy frequency when it is allowed to use $> 100{,}000$ samples.

Figure 3.6 investigates how signal size influences success probability. Every data point in Figure 3.6 is generated by 1000 trial runs on randomly generated input signals $\mathbf{A}$. The 1000 signals used for each data point vary in size from 2^{17} through 2^{26}. All sizes are powers of two. Otherwise each trial signal $\mathbf{A}$ is created just as before (i.e., consists of a randomly generated single frequency signal, $\tilde{\mathbf{A}}$, with added Gaussian noise, $\mathbf{G}$). The standard complex Gaussian noise is regenerated for each trial run via the Polar Box-Muller method. For every Figure 3.6 data point the magnitude of $\tilde{\mathbf{A}}$ is chosen so that the mean SNR of all the data point's trial signals is tightly grouped around -17 (SNR standard deviations for all data points are < 0.013). Probabilities of successfully calculating the minimum frequency of maximum energy are also calculated just as before.

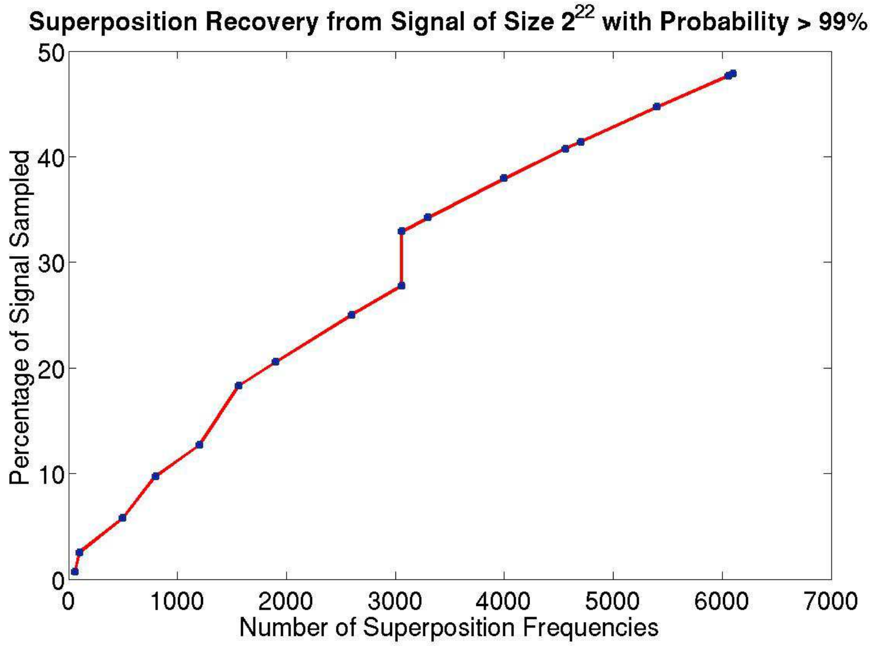


FIG. 3.7. *Signal samples for sparse superposition recovery via AAFFT 0.9*

Figure 3.6 presents the variation of success probability with signal size for AAFFT 0.9 with three different numbers of sample cutoffs. Again, the number of samples was determined by AAFFT's parameter settings. The data points to use for each cutoff curve were selected as follows: For each signal size, 6 data points were created, each using a different number of samples $< 105{,}000$. The data point for the y-sample cutoff curve at signal size x is the x-size data point using $< y$ samples with the highest success probability.

Looking at Figure 3.6, we can see that the achievable success probability appears to vary little with signal size. Each cutoff curve is essentially constant. Also, we see the expected increase of success probability with the number of allowed samples. Based on these results it seems safe to conclude that $\sim 10^5$ samples should be sufficient to achieve near perfect hidden frequency identification for any signal with SNR ≥ -17 that is storable in current computer memory.

Sparse Recovery: In our final experiment we investigated the number of signal positions we must read in order to recover all the frequencies of a sparse superposition. Figure 3.7 contains the results. As before, a sparse superposition was created for each individual trial run by selecting m frequencies uniformly at random from $[0, N)$ and then giving each selected frequency a magnitude 1 coefficient with uniformly random phase. Also, as before, each Figure 3.7 data point is the result of 1000 such trial runs. The probability of successful superposition frequency recovery was calculated by counting the number of trial runs for which AAFFT 0.9's L^∞ error was $< \frac{1}{2}$, divided by 1000. However, for each Figure 3.7 data point, AAFFT 0.9's mean L^∞ error was < 0.02 (i.e., better than $\frac{1}{2}$).

We know from Section 2.2.3 that AAFFT 0.9's runtime should (given a fixed signal size N, failure probability δ, and desired accuracy ϵ) scale linearly in the input signal's sparsity level m. Therefore, assuming good parameter settings, the worst case number of samples AAFFT 0.9 requires to recover a signal must also scale linearly in the sparsity level. Looking at Figure 3.7 we can see that the number of samples required to recover a sparse superposition with high probability does indeed appear to scale linearly with superposition sparsity level (the number of non-zero coefficient frequencies m). Figure 3.7 also indicates that, with high probability, AAFFT 0.9 can approximate the DFT of any roughly 6000-term superposition of length $N = 2^{22}$ using less than half of the superposition's samples.

To date, L^1-minimization-based sparse Fourier methods [4] have not been shown to allow exact reconstruction of an m-term/N-length superposition's DFT with high uniform probability unless at least $O(m \log^4 N)$ signal samples are used [18]. Hence, we can see that the number of samples that AAFFT 0.9 requires to approximate a superposition's Fourier transform with high probability is at worst a polylog(N) multiple of the number of samples required to calculate (to machine precision) a superposition's Fourier transform with high uniform probability via L^1-minimization. This is a potentially promising result given that L^1-minimization based methods have higher theoretical run time complexity than AAFFT 0.9.

4. Conclusion

In this paper we empirically demonstrated that FADFT-2 [11] retains all the advantages of FADFT-1 [10, 20] while also being more computationally efficient. To accomplish this task a C++ implementation, AAFFT 0.9, of FADFT-2 was compared against a C++ implementation, AAFFT 0.5, of FADFT-1. Both implementations were bench-marked against FFTW 3.1 [9].

In Section 3.1 the runtime and approximation error of AAFFT 0.9 and 0.5 were compared for sparse superpositions (i.e., signals with a small number of non-zero frequencies). Section 3.1's comparisons demonstrated that AAFFT 0.9 is generally faster than AAFFT 0.5 while retaining similar accuracy. Furthermore, it was demonstrated that both AAFFT 0.9 and AAFFT 0.5 outperform FFTW 3.1 for large sparse superpositions.

In Section 3.2 we saw that AAFFT 0.9 and AAFFT 0.5 are essentially indistinguishable in terms of noise tolerance. Furthermore, we saw that AAFFT 0.9's noise tolerance is relatively independent of signal size. Based on Section 3.2's results we may safely conclude that both AAFFT 0.9 and 0.5 are highly tolerant to small amounts of noise (e.g., SNR > -10) as long as AAFFT 0.9/0.5 may use a few tens of thousands of samples from signals of size $\sim 10^6$. Finally, we saw that AAFFT 0.9 is capable of approximating the output of higher time complexity L^1-minimization methods using, at worst, polylog(N) times L^1-minimization's required number of samples. As future work we plan to perform a more careful empirical comparison between AAFFT and L^1-minimization based sparse Fourier methods in order to more accurately determine their runtime/sampling complexity tradeoffs.

Acknowledgement. We would like to thank Michael Lieberman for reading, and commenting on, an earlier version of this paper. We would also like to thank Wendy Grus for supportive ideas and helpful comments.

REFERENCES

[1] http://www-personal.umich.edu/~markiwen/.

[2] C. Anderson and M.D. Dahleh, *Rapid computation of the discrete Fourier transform*, SIAM J. Sci. Comput., 17, 913–919, 1996.
[3] G. Box and M. Muller, *A note on the generation of random normal deviates*, Ann. Math. Stat., 29, 610–611, 1958.
[4] E. Candes, J. Romberg and T. Tao, *Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information*, IEEE Trans. Inform. Theory, 52(2), 489–509, 2006.
[5] J. Cooley and J. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput., 19, 297–301, 1965.
[6] I. Daubechies, O. Runborg and J. Zou, *A Sparse spectral method for homogenization multiscale problems*, Multiscale Model. Sim., 2007.
[7] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14, 1368–1383, 1993.
[8] J.A. Fessler and B.P. Sutton, *Nonuniform fast Fourier transforms using min-max interpolation*, IEEE Trans. Signal Proc., 51, 560–574, 2003.
[9] M. Frigo and S. Johnson, *The design and implementation of FFTW3*, Proceedings of IEEE, 93(2), 216–231, 2005.
[10] A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan and M. Strauss, *Near-optimal sparse Fourier estimation via sampling*, ACM STOC, 152–161, 2002.
[11] A. Gilbert, S. Muthukrishnan and M. Strauss, *Improved Time Bounds for Near-Optimal Sparse Fourier Representations*, SPIE, 2005.
[12] M. Iwen, unpublished results. http://www-personal.umich.edu/∼markiwen/respub.htm
[13] S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud and R. Baraniuk, *Analog-to-information conversion via random demodulation*, Proc. IEEE Dallas Circuits and Systems Conference, 2006.
[14] J. Laska, S. Kirolos, Y. Massoud, R. Baraniuk, A. Gilbert, M. Iwen and M. Strauss, *Random sampling for analog-to-information conversion of wideband signals*, Proc. IEEE Dallas Circuits and Systems Conference, 2006.
[15] J.Y. Lee and L. Greengard, *The type 3 nonuniform FFT and its applications*, J. Comput. Phys., 206(1), 1–5, 2005.
[16] M. Lustig, D. Donoho and J. Pauly, *Sparse MRI: The application of compressed sensing for rapid MR imaging*, submitted, 2007.
[17] R. Maleh, A.C. Gilbert and M.J. Strauss, *Sparse gradient image reconstruction done faster*, IEEE International Conference on Image Processing, 2007.
[18] M. Rudelson and R. Vershynin, *Sparse reconstruction by convex relaxation: Fourier and Gaussian measurements*, Conference on Information Sciences and Systems, 2006.
[19] J. Tropp and A. Gilbert, *Signal recovery from partial information via orthogonal matching pursuit*, submitted, 2005.
[20] J. Zou, A. Gilbert, M. Strauss and I. Daubechies, *Theoretical and experimental analysis of a randomized algorithm for sparse Fourier transform analysis*, J. Comput. Phys., 211(2), 572–595, 2006.