Example scribed lecture notes

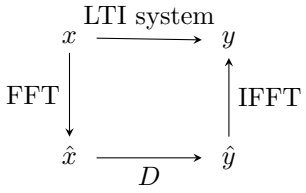
Three authors

1 IFFT from FFT

Last time, we learned about rapidly computing the discrete Fourier transform (DFT),

$$\hat{x}(k) = \sum_{t=0}^{N-1} x(t) e^{-i2\pi kt/N}, \quad k = 0, 1, \ldots, N-1,$$

when N was not prime using the fast Fourier transform (FFT) algorithm. The $O(N \log N)$ computational complexity of the FFT and inverse FFT (IFFT) give us a fast way of evaluating the response of a linear, time-invariant (LTI) system to a signal. This is illustrated in the diagram below, where D stands for a diagonal operator.



Because the FFT is so important, there has been significant research effort to implement the algorithm in hardware [2, 6, 13]. However, we need the IFFT to complete the above diagram. Does this mean we need FFT *and* IFFT hardware? Of course not. The trick is time reversal:

$$\begin{aligned} x(t) &= \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}(k) e^{i2\pi kt/N} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}(N-1-k) e^{i2\pi(N-1-k)t/N} \\ &= \frac{e^{-i2\pi t/N}}{N} \sum_{k=0}^{N-1} \hat{x}(N-1-k) e^{-i2\pi kt/N} \end{aligned}$$

The summation in the last line is the FFT of the time-reversed signal $\hat{x}(N-1-k)$. Thus, the IFFT consists of three steps: (1) time reverse, (2) FFT, and (3) post-multiply.

2 Zero-padding

An important consideration is the effect of zero padding a signal in time before computing a DFT. Suppose our signal is of length N, which we will denote $x_N(t)$. The zero-padded signal of length $2N$, $x_{2N}(t)$, is defined as

$$x_{2N}(t) = \begin{cases} x_N(t), & t = 0, \ldots, N-1 \\ 0, & t = N, \ldots, 2N-1 \end{cases}$$

The DFT of $x_{2N}(t)$ is then

$$\begin{aligned} \hat{x}_{2N}(k) &= \sum_{t=0}^{2N-1} x_{2N}(t) e^{-2\pi itk/(2N)} \\ &= \sum_{t=0}^{N-1} x_{2N}(t) e^{-2\pi itk/(2N)} \end{aligned}$$

This is almost the DFT of x_N. The difference is the $2N$ in the exponential and the fact that k ranges from 0 to $2N-1$. We seek a physical interpretation of this DFT. To do this, we view the DFT as a trigonometric polynomial:

$$\hat{x}_N(k) = \sum_{t=0}^{N-1} x_N(t) e^{-i2\pi kt/N} \rightarrow F(w) = \sum_{t=0}^{N-1} x_N(t) e^{-itw}$$

The DFT of the signal x_N is evaluating the polynomial $F(w)$ at $w = \frac{2\pi k}{N}$, $k = 0, \ldots, N-1$, the FFT provides a fast algorithm for this evaluation. We can view this as sampling at the Nyquist rate in the frequency domain, see Figure 1.

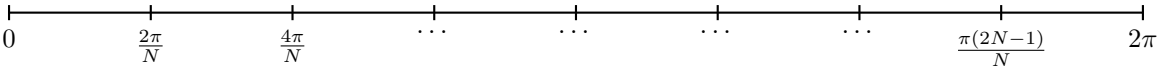


Figure 1: Samples in the frequency domain with no zero-padding (Nyquist rate).

Through this lens, the DFT of $x_{2N}(t)$ is evaluating the same polynomial $F(w)$ at additional points, namely, $w = \frac{\pi k}{N}, k = 0, \ldots, 2N-1$. This is illustrated in Figure 2. We can similarly evaluate the DFT of $x_{dN}(t)$, and we say that d is the *oversampling factor*.

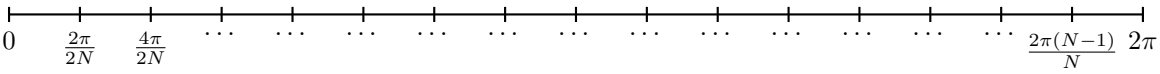


Figure 2: Samples in the frequency domain with an oversampling factor of $d = 2$.

3 Going off the grid

So far, we have assumed that the frequencies w in evaluating $F(w)$ were equispaced and that the sampling times t for $x(t)$ were equispaced. However, in many applications the evaluation points in frequency and/or time are not equispaced yet we would still like to be able to evaluate the trigonometric polynomial at all of the desired points in $O(N \log N)$ time.

3.1 Applications

Let us consider the case where the data, $y(k)$, consists of measurements of $F(\omega_k)$ at unequispaced frequencies ω_k and we want to recover the signal $x(t)$ at equispaced t. By definition these signals satisfy the following relationship:

$$y(k) = \sum_{t=0}^{N-1} x(t) e^{-i\omega_k t}, \quad k = 0, \ldots, M-1. \tag{1}$$

One specific problem of this form is magnetic resonance imaging (MRI). The frequency samples $y(k)$ are collected by the machine and we want to recover the brain image $x(t)$.

Another example is computed axial tomography (CAT) scans. In this case, after suitable preprocessing we are given frequency samples $y(k)$ that lie on a polar grid in $2D$. Note that in this special case there are actually specialized algorithms for computing this so called polar DFT rapidly, see, *e.g.* [1], [4] and [10].

Another application is finding numerical solutions to partial differential equations such as heat equation (the differential equation $u_t = u_{xx}$), see, *e.g.*, [12]. In this case, the problem is solved in the frequency domain at unequispaced points, as shown in Figure 3.

3.2 Least squares formulation

In matrix form, we can write (1) as $y = Ax$, where $A_{kt} = e^{-i\omega_k t}$, $k = 0, \ldots, M-1$, $t = 0, \ldots, N-1$. Typically, $M \geq N$, $y \approx Ax$, and one approach is to solve the following least squares problem:

$$\min_{x \in \mathbb{C}^n} f(x) = ||y - Ax||_2^2$$

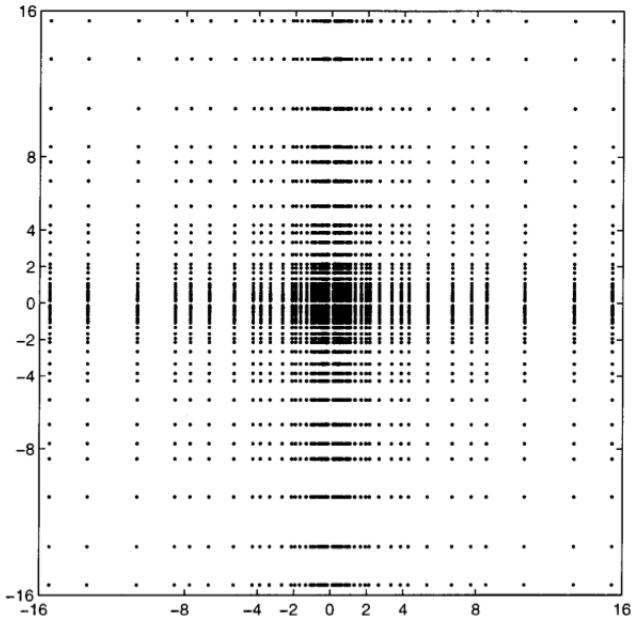


Figure 3: Unequispaced frequency samples for computing a numerical solution of the heat equation. Image from [12].

In practice A is too large for a direct method such as QR, so we use an iterative method to solve this optimization problem. An example iterative method is gradient descent, which requires being able to evaluate the gradient of the objective function f:

$$\nabla f(x) = A^*(Ax - y)$$

Thus, we are concerned with applying the matrices A and A^* quickly. Let's examine how these matrices operate on a vector via

- $(Ax)_k = \sum_{t=0}^{N-1} x(t)e^{-it\omega_k}$ (equispaced $\to$ unequispaced),
- $(A^*x)_k = \sum_{n=0}^{M-1} x(t_n)e^{2\pi i t_n k}$ (unequispaced $\to$ equispaced),

where we have switched the role of time and space for the adjoint operator. Thus, A^*A takes an equispaced vector to an equispaced vector. In the language of Greengard and Lee [11], applying the operator A constitutes a *nonuniform FFT (NUFFT) of Type II* and applying the operator A^* constitutes a *NUFFT of Type I*. For completeness, we also have the *NUFFT of Type III*, which is used to evaluate $\sum_{n=0}^{N-1} x(t_n)e^{-it\omega_k}$ (unequispaced $\to$ unequispaced).

Naïvely, the computational complexity of applying A and A^* is $O(MN)$. In Section 4, we will examine how to rapidly apply A and A^* to a vector. We note that, given the ω_k, we have an analytic representation of A_{kt}, namely $e^{-i\omega_k t}$. Thus, A and A^* require only $O(N)$ memory.

4 NUFFTs for Dummies

The original Dutt & Rokhlin approach to the NUFFT in [9] is "brilliant" (Candès), but requires a lot of background knowledge in signal processing and spectrum analysis. We will review the details in Section 5, but first we look here at a conceptually straight-forward approach by Candès, Demanet, Donoho, & Ying [5]. We will start with the Type II NUFFT, as it is simpler.

4.1 Algorithm for Type II NUFFT

Recall that for the Type II NUFFT we wish to evaluate the trigonometric polynomial $F(\omega) = \sum x(t)\exp(-it\omega)$ on an arbitrary grid in frequency space. In the notation of Section 2, suppose we oversample our N-point signal $X_N(t)$ by a factor of d by appending $(d-1)N$ trailing zeros to $X_N(t)$, i.e.,

$$X_{dN}(t) = \begin{cases} X_N(t) & : t \in \{t_0, \ldots, t_{N-1}\}, \\ 0 & : t \in \{t_N, \ldots, t_{dN-1}\}. \end{cases}$$

As we saw before, this corresponds to evaluating $F(\omega)$ at the regular grid points $\tilde{\omega}_k = \frac{2\pi k}{dN}$ for $k = 0, \ldots, dN-1$.

The key idea for computing the Type II NUFFT will be Taylor approximation. In the first step, on the fine grid we evaluate the trigonometric polynomial and its derivatives, $F^{(l)}(\omega)$ for $l = 0, \ldots, L-1$. In the second step, for each point ω_k in the set of evaluation points we perform a Taylor expansion of order $L-1$ around its nearest neighbor in the fine grid.

4.1.1 Step 1

We wish to determine the values $F^{(l)}(\tilde{\omega}_k)$ for $\tilde{\omega}_k$ in the fine grid and $l = 0, \ldots, L-1$. For $l = 0$, we see that this is just the dN-point DFT of $x(t)$, so we can use the Cooley-Tukey FFT algorithm [7] for regular grids. For $l \neq 0$, we are dealing with some derivative of $F(w)$, but from the definition we see that

$$\begin{aligned} F^{(l)}(\omega) &= \frac{\partial^{(l)}}{\partial \omega^{(l)}} \left[\sum x(t) \exp(-it\omega) \right] \\ &= \sum x(t)(-it)^l \exp(-it\omega) \\ &\equiv \sum x_l(t) \exp(-it\omega), \end{aligned}$$

where we observe that the last quantity is just the DFT of $x_l(t) = (-it)^l x(t)$, which can be computed via pointwise multiplication (to obtain $x_l(t)$) followed by the Cooley-Tukey algorithm.

Thus, we see that the dominant work in this step is L FFTs, each of length dN, for a cost of $O(LdN \log(dN)) = O(LdN \log N)$ (assuming $d \ll N$).

4.1.2 Step 2

At the end of Step 1 we have the values of $F(\omega)$ and its derivatives on the fine grid. The next step is to perform a Taylor expansion for each evaluation point. Suppose ω_k is a point in the non-uniform grid at which we want to evaluate $F(\omega)$. Then, choosing $\tilde{\omega}_k$ to be the closest fine grid point to ω_k, we can Taylor expand around $\tilde{\omega}_k$ to obtain

$$\begin{aligned} F(\omega_k) &\approx F(\tilde{\omega}_k) + F'(\tilde{\omega}_k)(\omega_k - \tilde{\omega}_k) + \ldots + \frac{F^{(L-1)}(\tilde{\omega}_k)(\omega_k - \tilde{\omega}_k)^{(L-1)}}{(L-1)!} \\ &\equiv [P_{l,\tilde{\omega}_k} f](\omega_k). \end{aligned}$$

It is evident that, since we have already computed all the necessary function evaluations, this is $\mathcal{O}(LdN)$.

4.1.3 Error Analysis

The only approximation made in the above algorithm occurs in the Taylor expansions of Step 2. Luckily, the error term here is not too difficult to deal with. Using the Lagrange form of the remainder term for the Taylor expansion, we see that, for any given evaluation point ω_k the bound below holds:

$$|F(\omega_k) - [P_{l,\tilde{\omega}_k} f](\omega_k)| \leq ||F^{(L)}||_\infty \frac{|\omega_k - \tilde{\omega}_k|^L}{L!}. \tag{2}$$

To get a better handle on this bound, we employ the following fact about trigonometric polynomials.

Theorem 1 (*Bernstein's Inequality* [15])**.** *Let $p(z)$ be a trigonometric polynomial of degree n with frequencies ranging from $-n/2$ to $n/2$. Then we have that*

$$||p'||_\infty \leq \frac{n}{2} ||p||_\infty.$$

By using Theorem 1 L times on (2) we obtain the bound

$$|F(\omega_k) - [P_{l,\tilde{\omega}_k} f](\omega_k)| \leq \left(\frac{N}{2}\right)^L \frac{|\omega_k - \tilde{\omega}_k|^L}{L!} ||F||_\infty.$$

Further, noting that the distance between an evaluation point and its nearest neighbor in the fine grid can never be greater than $\frac{\pi}{dN}$, half the spacing of the fine grid, we obtain the final relative error bound

$$\max_{\omega_k} |F(\omega_k) - [P_{l,\tilde{\omega}_k} f](\omega_k)| \leq \left(\frac{\pi}{2d}\right)^L \frac{1}{L!} ||F||_\infty.$$

For $L = 10$ and $d = 4$, the relative error is roughly 2.4×10^{-11}, guaranteeing 11 digits of accuracy. For most applications, noise in the initial data means this level of accuracy is meaningless anyway, so a more reasonable set of parameters might be $L = 4$ and $d = 8$, giving a relative error of about 6.2×10^{-5}, guaranteeing 5 digits of accuracy.

4.2 Example implementation

The MATLAB code below provides an example implementation of the Type II NUFFT algorithm from [5]. Note that some tests will cause the error bound to appear to fail, but this is because we do not know the true value of $||F||_\infty$, so we cannot calculate the true relative error, only an approximation.

```
N = 2^10; L = 4; d = 8;

%Create x, keep track of approximate infinity norm of dN-point FFT of x
t = (0:N-1)';
td = (0:d*N-1)';
x = cos(5*pi*t/N) + 2*cos(20*pi*t/N);
approx_norm = 2*d*N;

%zero-pad, then pointwise multiply to get x_l for each l
xL = [x; zeros((d-1)*N,1)];
for l = 1:L-1
    xL(:,l+1) = (-i.*td).*xL(:,l);
end
F = fft(xL);

%evaluation points
wk = 2*pi*sort(rand(N,1));
wk_tilde = 2*pi*(0:d*N-1)'/d/N;
%This is not a good way to find the nearest neighbor in practice
dist = @(w) find(abs(w-wk_tilde) == min(abs(w - wk_tilde)));
neighbors_idx = arrayfun(dist,wk);
neighbors = wk_tilde(neighbors_idx);
diffs = wk - neighbors;

%true solution at scattered points
ft = @(w) sum(x.*exp(-i*w*t));
Fwk = arrayfun(ft,wk);

%approximate solution from NUFFT Taylor approximation
l = (0:L-1);
taylor = @(neighbor,diff) sum(F(neighbor,:).*diff.^l./factorial(l));

Fwk_approx = arrayfun(taylor,neighbors_idx,diffs);
err = norm(Fwk - Fwk_approx,'inf') /approx_norm;
bnd = abs(pi/2/d)^L / factorial(L);

disp('Maximum error relative to approximate infinity norm of F');
disp(err);
disp('Bound on relative error (depends on true infinity norm of F)');
disp(bnd);
```

4.3 The adjoint operator: Algorithm for Type I NUFFT

Given that the Type I NUFFT is simply the adjoint of the Type II NUFFT, (i.e., $Y(k) = \sum_{t_n} x(t_n) \exp(ikt_n)$), it should be unsurprising that we can use the adjoint operators in each of the above steps to obtain a Type I algorithm.

As a brief diversion, let's look first at how we can write the Type I algorithm in matrix-vector form. Let $x \in \mathbb{C}^N$ be the vector of time-domain samples. We introduce the zero-padding operator Z, such that

$$x_d = Zx \equiv \begin{bmatrix} I \\ 0 \end{bmatrix} x,$$

where the top block is an identity of size $N \times N$ and the bottom block is a block of zeros of size $(d-1)N \times N$. Next, we introduce the operator $M \in \mathbb{C}^{dNL \times dN}$, which performs pointwise multiplication to obtain a vector of the stacked

signals $x_l(t)$, i.e.,

$$\begin{bmatrix} x \\ x_1 \\ \vdots \\ x_{L-1} \end{bmatrix} = Mx_d \equiv \begin{bmatrix} I \\ D \\ D^2 \\ \vdots \\ D^{L-1} \end{bmatrix} x_d,$$

where $D \in \mathbb{C}^{dN \times dN}$ is given by $D = \text{diag}(-it_n)$.

With W as the DFT matrix, we obtain

$$\begin{bmatrix} \tilde{F} \\ \tilde{F}' \\ \vdots \\ \tilde{F}^{(L-1)} \end{bmatrix} = \begin{bmatrix} W & 0 & \dots & 0 \\ 0 & W & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & W \end{bmatrix} \begin{bmatrix} x \\ x_1 \\ \vdots \\ x_{L-1} \end{bmatrix},$$

where each $\tilde{F}^{(l)}$ is a dN-vector of the values of $\tilde{F}^{(l)}(\omega)$ on the fine grid. Finally, defining $T_l \in \mathbb{C}^{dN \times dN}$ to be the matrix that introduces the necessary distance weighting for the Taylor polynomial, i.e.,

$$(T_l)_{ij} = \begin{cases} \frac{(\omega_i - \tilde{\omega}_j)^l}{l!} & \tilde{\omega}_j \text{ is the closest fine-grid point to } \omega_i, \\ 0 & \text{else,} \end{cases}$$

we obtain

$$F = \begin{bmatrix} T_0 & T_1 & \dots & T_{L-1} \end{bmatrix} \begin{bmatrix} \tilde{F} \\ \tilde{F}' \\ \vdots \\ \tilde{F}^{(L-1)} \end{bmatrix},$$

where F is the vector of $F(w)$ evaluated at the scattered points in the frequency domain ω_k. Chaining all the operators together, we can write

$$F = \begin{bmatrix} T_0 & T_1 & \dots & T_{L-1} \end{bmatrix} \begin{bmatrix} W & 0 & \dots & 0 \\ 0 & W & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & W \end{bmatrix} \begin{bmatrix} I \\ D \\ D^2 \\ \vdots \\ D^{L-1} \end{bmatrix} \begin{bmatrix} I \\ 0 \end{bmatrix} x,$$

and from here it is evident that to perform the Type I transform, the adjoint of the Type II, we can simply take the adjoint of each of these individual operators, which yields

$$Y = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} I & D^* & (D^*)^2 & \dots & (D^*)^{L-1} \end{bmatrix} \begin{bmatrix} W^* & 0 & \dots & 0 \\ 0 & W^* & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & W^* \end{bmatrix} \begin{bmatrix} T_0^* \\ T_1^* \\ \vdots \\ T_{L-1}^* \end{bmatrix} x. \tag{3}$$

From an algorithmic standpoint, we are done, but let's try to gain some mathematical insight into what this adjoint transformation actually is doing as far as approximations go.

4.3.1 Mathematical intuition

Recall that the adjoint transform is

$$Y(k) = \sum_{t_n} x(t_n) \exp(it_n k).$$

By Taylor's Theorem, note that for t sufficiently close to t_n, we have that

$$\exp(it_nk) = \exp(itk)\exp(i(t_n - t)k) \approx \exp(itk)\left[1 + i(t_n - t)k + \cdots + \frac{(i(t_n - t)k)^{L-1}}{L!}\right].$$

Let $\tilde{t}_m$ be the uniform fine-grid points obtained via oversampling. Then, using some careful reordering,

$$Y(k) \approx \sum_{l=0}^{L} (ik)^l \sum_{\tilde{t}_m} \exp(i\tilde{t}_m k) \sum_{t_n \in \mathcal{N}(\tilde{t}_m)} \frac{(t_n - \tilde{t}_m)^l}{l!} x(t_n),$$

where we note that it is possible for one $\tilde{t}_m$ to have multiple t_n for which it is the nearest neighbor in the fine grid, and thus the third sum aggregates these terms. This suggests the following algorithm [5]:

1. Compute the dN-vector $x_l(\tilde{t}_m) = \sum_{t_n \in \mathcal{N}(\tilde{t}_m)} \frac{(t_n - \tilde{t}_m)^l}{l!} x(t_n)$ for all m for $l = 0, \ldots, L-1$.
2. Compute the dN-vector $\hat{x}_l(k)$ by taking the IFFT of x_l for each l.
3. Compute the dN vector $\tilde{Y}(k) = \sum_{l=0}^{L} (ik)^l \hat{x}_l(k)$.
4. Extract the first N entries of $\tilde{Y}(k)$.

This is exactly the process described in Eqn. (3). The error and cost analysis are the same as in the forward case.

4.4 Type III NUFFTs

Note that there is nothing of major interest to say about the Type III NUFFT (mapping between nonuniform time and nonuniform frequency) as it is obtained by the same ideas as exposited above: Taylor expand to map to a uniform grid, use the FFT algorithm, then Taylor expand to map to the new nonuniform grid.

5 Other Non-uniform Fast Fourier Transform algorithms

The preceding section presented a method for computing a DFT on nonuniformly spaced time and frequency samples via the so-called NUFFT given in [5]. These algorithms are also sometimes referred to as unequally spaced fast Fourier transform (USFFT) algorithms. However, this is by no means the only method for such a DFT. The late 20th century saw the advent of a number of different USFFT algorithms. Many of these algorithms are similar to the preceding algorithm in that they combine interpolation schemes with oversampling. Here we will provide a brief overview of a few of the key papers and USFFT algorithms. Some references not discussed in detail here that discuss algorithms for computing a NUFFT include [8], [11] and [14]. The notation of this section follows the original papers and thus may differ from our earlier discussion.

5.1 Interpolation with Gaussian Bells

Dutt and Rokhlin [9] presented an algorithm to take a set of N complex numbers h_k and compute sums of the form

$$\hat{h}_j = \sum_{k=0}^{N} h_k e^{ix_j w_k}, \quad j = 0, \ldots, N \tag{4}$$

where $w_n \in [-N/2, N/2]$ and $x_l \in [-\pi, \pi]$ are not necessarily uniformly spaced points. The paper by Dutt and Rokhlin actually contains a collection of algorithms for computing sums of this form, and their inverses, in various circumstances. However, here we are only going to present the ideas behind the algorithms and thus the specifics of the given problems are omitted. Please see [9] for full descriptions of all the algorithms. The presentation here is also in line with the presentation of the paper, which speaks in terms of interpolation schemes. However, as discussed in class there are alternatives to this presentation of the method.

Just as the NUFFT presented in the preceding section uses a combination of interpolation and oversampling, so does the algorithm presented by Dutt and Rokhlin. In particular, a specialized interpolation scheme using Gaussian bells is developed and used to move between nonequispaced points and equispaced points. To achieve this, Rokhlin and Dutt use the fact, Theorem 2.10 in [9], that functions of the form e^{icx} may be approximated on some interval of

the real line using a small collection of terms of the form $e^{bx^2}e^{ikx}$ with integer k. The number of terms, q, needed is independent of c. To illustrate how these representations may be used, consider writing

$$e^{icx} \approx e^{bx^2} \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx}, \tag{5}$$

for some ρ_k, see, *e.g.*, Corollary 2.9 and Theorem 2.10 in [9]. Notationally, $[c]$ represents the integer closest to c and the accuracy of such a representation and the interval on which it is valid may be controlled by choosing q and an oversampling factor denoted m.

These representations allow for the transition between nonequispaced and equispaced grids. Specifically, if the x_j are equispaced and the w_k are not then this representation formula converts (4) into a standard DFT of length mN which may be computed via the FFT. Conversely, if the x_j are not equispaced but the w_k are a formula of the form (5) may be used to interpolate the result of an oversampled DFT, once again computed by the FFT, onto the unequally spaced x_j. These steps analogous to those in the Type I and Type II NUFFT algorithms presented previously.

These observations mean that a NUFFT using this interpolation scheme has two key components, there is an interpolation step and the computation of an FFT of length mN. In particular in [9] the authors show that the interpolation step only requires $O(Nq)$ operations to achieve a desired accuracy. Thus, based on the $O(mN \log N)$ run time for the FFT the authors show that the operation count when either the w_k are equispaced or the x_j are equispaced is $O(mN \log N + Nq)$, in the case where both are not uniformly spaced this grows to $O(m^2 N \log N + Nq)$. The authors assume that $m^2 \ll N$, and in fact it is often sufficient to take m as a small integer. Finally, the authors use the fact that $q \sim \log(1/\epsilon)$ where ϵ is the desired accuracy to get an operation count in the first case of $O(mN \log N + N \log(1/\epsilon))$.

5.2 Projecting functions onto a subspace

Beylkin in [3] presents an alternative USFFT algorithm to the one previously presented. Furthermore, the paper presents a different perspective on the structure of the algorithm. Specifically, instead of considering an interpolation scheme the paper presents a means of projecting a function onto a specific space that allows for the computation of an unequally spaced DFT via a projection, an oversampled DFT, and a final correction step. A brief overview of some of the key ideas in the algorithm are given here, for details please see [3]. Beylkin also discusses the multidimensional case, however, here we will restrict our discussion to the one dimensional case.

Similar to before, let us consider a set of N_p complex numbers h_k for which we wish to compute sums of the form

$$\hat{h}_j = \sum_{k=0}^{N_P-1} h_k e^{-2\pi i x_k \xi_j}, \quad j = 0, 1 \ldots, N \tag{6}$$

where $\xi_j \in [-N, N]$ and $x_k \in [0, 1]$ are not necessarily uniformly spaced points. To address the problem of rapidly computing sums of this form Beylkin introduces the generalized function

$$h(x) = \sum_{k=0}^{N_P-1} h_k \delta(x - x_k). \tag{7}$$

Using this the evaluation of (6) may be thought of as computing

$$\hat{h}(\xi) = \int h(x) e^{-2\pi i x \xi} dx, \tag{8}$$

at the set of points ξ_j, $j = 0, 1, \ldots, N$ where $|\xi_j| \leq N$. This converts the problem into one where we wish to accurately and rapidly compute the Fourier transform of a generalized function in the region $|\xi| \leq N$.

The first step of the algorithm for computing the necessary Fourier transforms is to project the generalized function $h(x)$ onto the j^{th} level of a set of subspaces each spanned by translations and scaling of a scaling function $\phi(x)$, denoted

$$\phi_{k,j}(x) = 2^{-j/2} \phi(2^{-j} x - k),$$

see [3] for details. Considering the projections of $h(x)$ onto $\phi_{k,j}$ yields coefficients

$$g_k = \int h(x) \phi_{k,j}(x) dx.$$

Computation of these coefficients may be interpreted as a blurring of the function, and this interpretation is similar to the role of the interpolation in the algorithm by Dutt and Rokhlin. The next step of the algorithm is to consider the Fourier series,

$$H(x) = \sum_k g_k e^{-2\pi i \xi k}. \tag{9}$$

It is important to note that specifically for the USFFT the compact support of $h(x)$ means that for an appropriate selection of $\phi(x)$ (*e.g.*, compact support) the g_k will only be nonzero for a finite range of k, which depends on j and the support of ϕ. This Fourier series, after an appropriate multiplicative correction in ξ space is an accurate approximation for $\hat{h}(\xi)$ on a fixed interval, see Theorem III.1 in [3]. Under an appropriate selection of $\phi(x)$ and an oversampling factor ν any desired accuracy may be achieved over the region of interest.

Beylkin motivates choosing ϕ to be the central B-spline of m^{th} order, denoted $\beta^{(m)}(x)$. Essentially, the compact support of the splines in the spatial domain coupled with their decay in the Fourier domain motivates the choice.

To outline one version of the algorithm we consider the case where the ξ_j are equally spaced and the x_k are not. As before, the algorithms for the other cases are similar in structure. To address the case where the ξ_j are not equally spaced an interpolation scheme using B-splines is used, see [3] for the details. The compact support in space coupled with the nature of the function $h(x)$ means that g_k may be computed as

$$g_k = 2^{-j/2} \sum_{l=0}^{N_p-1} h_l \beta^{(m)}(2^{-j} x_l - k).$$

Only a finite number of these coefficients will be nonzero and thus the Fourier series

$$H(\xi) = \sum_k g_k e^{-2\pi i k \xi}$$

may be evaluated at the desired equispaced points via the FFT. Multiplying by a corrective factor in ξ, which may be computed cheaply yields an approximation for $\hat{h}_j$. The projection onto the B-splines costs mN_p operations and the FFT is used on a νN length signal. Furthermore, since $m \sim \log(1/\epsilon)$ where ϵ is the desired accuracy the final computational cost of the algorithm is $O(\log(1/\epsilon)N_p + \nu N \log N)$. As before, we observe that these USFFT algorithms all tend to require both an interpolation, or projection, step and the computation of an oversampled DFT via the FFT, hence the broadly similar computational costs.

6 Additional resources

- A collection of non-uniform FFT software and applications is at `http://www.cims.nyu.edu/cmcl/nufft/nufft.html`.
- An image reconstruction toolbox for Matlab that includes NUFFT software is at `http://web.eecs.umich.edu/~fessler/code/index.html`

References

[1] A. Averbuch, R. Coifman, D. Donoho, M. Elad, and M. Israeli, *Fast and accurate polar Fourier transform*, Applied and Computational Harmonic Analysis, 21 (2006), pp. 145 – 167.

[2] B. M. Baas, *A low-power, high-performance, 1024-point FFT processor*, Solid-State Circuits, IEEE Journal of, 34 (1999), pp. 380–387.

[3] G. Beylkin, *On the Fast Fourier Transform of functions with singularities*, Applied and Computational Harmonic Analysis, 2 (1995), pp. 363 – 381.

[4] G. Beylkin, C. Kurcz, and L. Monzón, *Grids and transforms for band-limited functions in a disk*, Inverse Problems, 23 (2007), p. 2059.

[5] E. Candès, L. Demanet, D. Donoho, and L. Ying, *Fast discrete curvelet transforms*, 2005.

[6] E. Cetin, R. C. Morling, and I. Kale, *An integrated 256-point complex FFT processor for real-time spectrum analysis and measurement*, in Instrumentation and Measurement Technology Conference, 1997. IMTC/97. Proceedings. Sensing, Processing, Networking., IEEE, vol. 1, IEEE, 1997, pp. 96–101.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex fourier series*, Mathematics of Computation, 19 (1965), pp. pp. 297–301.

[8] A. Duijndam and M. Schonewille, *Nonuniform fast Fourier transform*, Geophysics, 64 (1999), pp. 539–551.

[9] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, SIAM J. Sci. Comput., 14 (1993), pp. 1368–1393.

[10] M. Fenn, S. Kunis, and D. Potts, *On the computation of the polar FFT*, Applied and Computational Harmonic Analysis, 22 (2007), pp. 257 – 263.

[11] L. Greengard and J. Lee, *Accelerating the Nonuniform Fast Fourier Transform*, SIAM Review, 46 (2004), pp. 443–454.

[12] L. Greengard and P. Lin, *Spectral approximation of the free-space heat kernel*, Applied and Computational Harmonic Analysis, 9 (2000), pp. 83–97.

[13] S. He and M. Torkelson, *Design and implementation of a 1024-point pipeline FFT processor*, in Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998, IEEE, 1998, pp. 131–134.

[14] D. Potts, G. Steidl, and M. Tasche, *Fast Fourier transforms for nonequispaced data: A tutorial*, 2000.

[15] A. Zygmund, *Trigonometric Series*, no. v. 1 in Cambridge Mathematical Library, Cambridge University Press, 2002.