Gridding

Nicholas Dwork

Abstract

This document attempts to explain the Gridding and Inverse Gridding algorithms. This document is heavily based on notes for Stanford's EE 369C class by Dr. John Pauly [1] and [2] by Beatty et al.

1 Introduction

This document is concerned with the following question: given a set of coordinates and corresponding values in the source domain how can we approximate the values at a set of coordinates in the destination domain? If the coordinates in both domains lie on the vertices of a cartesian grid then the answer is the DFT (or the DFT^{-1}, depending on which domain one starts in). But what do we do when the coordinates of one domain (or both) don't lie on a uniform grid?

The general problem, where there are no restrictions on the coordinates of either domain, is called the Non-uniform DFT. One might think that this problem is easily solved by interpolating in the source domain, performing a transform, and then interpolating in the destination domain. Unfortunately, this does not work out well.

We will look at two special cases of the Non-uniform DFT problem: *Gridding* and *Inverse Gridding*.

2 Background

Here we make relevant definitions and cover the preliminaries required for this document.

2.1 Fourier Transforms

For this document, we will use the following definitions for the Fourier Transform and inverse Fourier Transform:

$$F(k) = \mathcal{F}\{f\}(k) = \int_{-\infty}^{\infty} f(x)\, \exp(-i\, 2\pi\, k\, x)\, dx$$
$$f(x) = \mathcal{F}^{-1}\{F\}(x) = \int_{-\infty}^{\infty} F(k) \exp(i\, 2\pi\, k\, x)\, dk$$

where $i = \sqrt{-1}$.

The Discrete Fourier Transform (DFT) and the inverse Discrete Fourier Transform (DFT^{-1}) are linear operators; they can be derived as specific Riemann sums that approximate the Fourier Transforms where each element of the partition is equal in size, and the element in the summation (x_n or k_m) is the midpoint of the partition element. They are denoted as follows:

$$V[m] = \mathsf{DFT}(v)[m] = \sum_{n=0}^{N-1} v[n] \exp\left(-i\, 2\pi\, \frac{m\, n}{N}\right)$$
$$v[n] = \mathsf{DFT}^{-1}(V)[n] = \frac{1}{N} \sum_{m=0}^{N-1} V[m] \exp\left(i\, 2\pi\, \frac{m\, n}{N}\right).$$

The DFT and DFT^{-1} can be calculated very quickly using the Fast Fourier Transform (FFT) algorithm.

2.2 Riemann Sum Approximation

Approximating the Fourier Transform as a Riemann sum yields the following expression:

$$F(k) \approx \sum_j f(x_j) \exp\left(-i\, 2\pi\, k\, x_j\right) \Delta x_j .$$

Let us assume that the samples $\{f(x_j)\}$ were sampled over a finite domain with a uniform sampling rate of 1 (i.e. $\Delta x_j = 1$); this is a very common convention. Then

$$F(k) \approx \sum_{n=0}^{N-1} f(x_n) \exp\left(-i\, 2\pi\, k\, x_n\right).$$

Let $\boldsymbol{f}$ be the N-tuple $(f(x_0), \cdots, f(x_{N-1}))$. Then

$$F(k) \approx \sum_{n=0}^{N-1} \boldsymbol{f}[n] \exp\left(-i\, 2\pi\, k\, n\right).$$

Suppose we sample the Fourier domain at frequencies $\boldsymbol{k} = \left(0, \frac{1}{N}, \cdots, \frac{N-1}{N}\right)$. Then

$$F(\boldsymbol{k}_m) \approx \sum_{n=0}^{N-1} \boldsymbol{f}[n] \exp\left(-i\, 2\pi\, \frac{m\, n}{N}\right), \qquad \text{for } m \in \{0, 1, \cdots, N-1\}.$$

But the expression on the right is just the DFT! That is, $F(\boldsymbol{k}) \approx \text{DFT}(\boldsymbol{f})$. This shows that we can approximate the Fourier Transform using the DFT.

2.3 Convolution Theorems

The convolution theorems will be instrumental to the algorithms discussed. We will prepare by reviewing them here. Recall the definition of convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(s)\, g(x-s)\, ds.$$

We seek to relate multiplication in one domain to convolution in the other. Let us first consider the Fourier transform of convolution.

$$\begin{aligned}
\mathcal{F}\{f * g\}(k) &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(s) g(x-s)\, ds\right] \exp\left(-i2\pi\, k\, x\right) dx \\
&= \int_{-\infty}^{\infty} f(s) \left[\int_{-\infty}^{\infty} g(x-s) \exp\left(-i2\pi\, k\, x\right) dx\right] ds \qquad (1) \\
&= \int_{-\infty}^{\infty} f(s) G(k) \exp\left(-i2\pi\, k\, s\right) dx \qquad (2) \\
&= F(k)\, G(k).
\end{aligned}$$

In the above, (1) switched the order of the integrations and (2) used the Fourier shift theorem.

A very nice relationship is revealed; the Fourier transform of the convolution of two functions is the product of the Fourier transforms of those functions. Let us now consider the inverse Fourier transform of the convolution.

$$\begin{aligned}
\mathcal{F}^{-1}\{F * G\}(x) &= \int_{-\infty}^{\infty} (F * G)(k)\, \exp\left(i2\pi\, k\, x\right) dk = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} F(s)\, G(k-s)\, ds\right] \exp\left(i\, 2\pi\, k\, x\right) dx \\
&= \int_{-\infty}^{\infty} F(s) \left[\int_{-\infty}^{\infty} G(k-s) \exp\left(i\, 2\pi\, k\, x\right)\right] ds = \int_{-\infty}^{\infty} F(s)\, g(x) \exp\left(i\, 2\pi\, k\, s\right) dk = f(x)\, g(x).
\end{aligned}$$

This shows that the Fourier transform of multiplication is convolution.

The two results together are the convolution theorem and its dual for the continuous Fourier transform.

$$\boxed{\mathcal{F}\left\{f * g\right\} = F\,G \qquad \mathcal{F}\left\{f\,g\right\} = F * G}$$

We will now go through an analogous procedure to see the convolution theorem of the Discrete Fourier Transform. However, the relationship will hold for circular convolution, defined as

$$(f \circledast g)\,[m] = \sum_{n=0}^{N-1} f[n]\,g_p[m-n],$$

where $f, g \in \mathbb{C}^N$. The function $g_p : \mathbb{Z} \to \mathcal{C}$ is the periodic extension of g, defined as $g_p[m] = g[m \pmod{n}]$.

Let us consider the DFT of the circular convolution of two vectors $f, g \in \mathbb{C}^N$.

$$\begin{aligned}\mathsf{DFT}\{f \circledast g\}[k] &= \sum_{n=0}^{N-1} \left[(f \circledast g)\,[n]\right] \exp\left(-i\,2\pi\,\frac{kn}{N}\right) = \sum_{n=0}^{N-1}\left[\sum_{m=0}^{N-1} f[m]\,g_p[n-m]\right] \exp\left(-i\,2\pi\,\frac{kn}{N}\right)\\ &= \sum_{m=0}^{N-1} f[m] \sum_{n=0}^{N-1} g_p[n-m] \exp\left(-i\,2\pi\,\frac{kn}{N}\right) = \sum_{m=0}^{N-1} f[m] G[k] \exp\left(-i\,2\pi\,\frac{km}{N}\right)\\ &= \sum_{m=0}^{N-1} F[k] G[k].\end{aligned}$$

We see another beautiful relationship: the Discrete Fourier Transform of the circular convolution of two vectors is the Hadamard (or point-wise) product of their respective Discrete Fourier Transforms.

We can view this result in a different form. Recall that circular convolution is a linear transformation and can be represented by a circulant matrix. Let C_g be the circulant matrix such that $C_g\,f = f \circledast g$ for any vector f. Further, let $\boldsymbol{F}$ be the matrix representation of the Discrete Fourier Transform. Then for any vector f

$$\boldsymbol{F}\,C_g\,f = \mathsf{diag}(\boldsymbol{F}g)\,\boldsymbol{F}f.$$

And since this is true for any vector f,

$$\boldsymbol{F}\,C_g = \mathsf{diag}(\boldsymbol{F}g)\,\boldsymbol{F} \quad \Leftrightarrow \quad \boldsymbol{F}\,C_g\,\boldsymbol{F}^{-1} = \mathsf{diag}(G).$$

This shows that the Discrete Fourier Transform diagonalizes circulant matrices.

Now let us consider the Inverse Discrete Fourier Transform of circular convolution.

$$\begin{aligned}\mathsf{DFT}^{-1}\{F \circledast G\}[k] &= \frac{1}{N} \sum_{n=0}^{N-1} \left[\sum_{v=0}^{N-1} F[v]\,G_p[k-v]\right] \exp\left(i\,2\pi\,\frac{kn}{N}\right)\\ &= \frac{1}{N} \sum_{v=0}^{N-1} F[v] \sum_{n=0}^{N-1} G_p[k-v] \exp\left(i\,2\pi\,\frac{kn}{N}\right) = g[n] \sum_{v=0}^{N-1} F[v] \exp\left(i\,2\pi\,\frac{kv}{N}\right) = N\,f[n]\,g[n].\end{aligned}$$

This shows that the Discrete Fourier Transform of pointwise multiplication is circular convolution divided by the number of elements.

The two results together are the convolution theorem and its dual for the Discrete Fourier Transform. For vectors f and g with Discrete Fourier Transforms F and G, respectively:

$$\boxed{\mathsf{DFT}(f \circledast g) = F \cdot G \qquad \mathsf{DFT}\,(f \cdot g) = (1/N) F \circledast G}.$$

3 Gridding

Armed with the preliminaries discussed, we are now ready to discuss the Gridding and Inverse Gridding non-uniform Fast Fourier Transform algorithms. We'll start with Gridding.

Gridding is an algorithm that approximates the Inverse Fourier Transform when the source coordinates $(k_1, \ldots, k_M)$ do not lie on a uniform grid but the destination coordinates $(x_1, \ldots, x_N)$ do. The question becomes, given the coordinates $(k_1, \ldots, k_M)$ and values $(F(k_1), \ldots, F(k_M))$, how can we approximate the destination domain values $(f(x_1), \ldots, f(x_N))$? Importantly, to answer this question we want to take advantage of the computational efficiency of the Inverse Discrete Fourier Transform; that is, Gridding should call the DFT^{-1}.

Define S as the *sampling function*, $S(k) = \sum_j \delta(k - k_j)$. It is a distribution that specifies which frequencies k have known Fourier values. It is assumed that the number of samples is finite, and therefore S has compact support. The "sampled data" is the distribution FS.

The trick of gridding will be to *push not pull*. That is, rather than interpolating from nearby values to determine the values of frequencies that lie on a Cartesian grid, we are going to *push* information from all known frequencies onto the nearby grid points. This is done by performing a convolution in the Fourier domain with a kernel C. Using this idea, the approximations to the Fourier values in the source domain are

$$\tilde{F} = (F\,S) * C, \tag{3}$$

where $*$ denotes convolution. Note that convolution is an integration, so (by the sifting property) convolving with C converts the distribution FS to finite (possibly complex) values at all points in the source domain.

Now that we have an approximation to the function F, we can approximate the values in the destination domain by using the Inverse Fourier Transform

$$\mathcal{F}^{-1}\{F\} \approx \mathcal{F}^{-1}\{F\,S * C\}. \tag{4}$$

This operation can't be performed on a computer. So we sample the source domain data on a uniform grid, modeled as multiplication with a comb function as follows[1]:

$$[F\,S * C]\;\Delta k\,\text{Ш}_{\Delta k}, \tag{5}$$

where Ш is the comb (or Shah) function defined as $\text{Ш}_{\Delta k}(k) = \sum_{m=-\infty}^{\infty} \delta(k - m\,\Delta k)$.

Define $\mathcal{T}$ as a mapping such that $\mathcal{T}\left([F\,S * C]\,\Delta k\,\text{Ш}_{\Delta k}\right)[m]$ equals the weight of the m^{th} nonzero delta. Then $\mathcal{T}$ is an isomorphism that maps the set of functions of the form in (5) (together with pointwise addition and scalar multiplication) onto $\mathbb{C}^M$.

We can approximate the values of (4) by performing an DFT^{-1} on this vector of sampled data; this yields N equally spaced approximations in the destination domain. For any $n \in \{0, 1, \ldots, N-1\}$,

$$\begin{aligned} \hat{f}(n) &= \text{DFT}^{-1}\left\{\mathcal{T}\left[(F\,S * C)\,\Delta k\,\text{Ш}_{\Delta k}\right]\right\}[n] \qquad (6)\\ &\approx \left(\left[(f * s)\cdot c\right] * \text{Ш}_{\text{FOV}}\right)(n), \qquad (7) \end{aligned}$$

where s is the inverse Fourier Transform of S, and FOV $= 1/\Delta k$ where FOV stands for "Field of View". Note that $[(f * s)\cdot c] * \text{Ш}_{\text{FOV}}$ is periodic with period FOV; we will estimate samples from one period, hence the name "Field of View." The function $c = \mathcal{F}^{-1}\{C\}$ is called the "apodization function." Note that (7) is only approximately equal to (6) since pointwise multiplication with the apodization function c in (7) equals, according to the Convolution theorem[2], circular convolution in (6) scaled by N.

Circular convolution with C is a linear transformation from $\mathbb{C}^M \to \mathbb{C}^N$, thus it can be represented by a matrix $\mathcal{C}[\boldsymbol{k}, \boldsymbol{k}'] \in \mathbb{R}^{N \times M}$, where $\boldsymbol{k}$ is a vector of the sampled source domain coordinates and $\boldsymbol{k}'$ is a vector of the source domain coordinates on a Cartesian grid. If the support of C is small then this matrix is sparse. The value $\mathcal{C}[\boldsymbol{k}, \boldsymbol{k}']_{nm}$ is the value of C evaluated at $\|k'_n - k_m\|_2$.

$$\underbrace{\begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} & \cdots & \mathcal{C}_{1M} \\ \mathcal{C}_{21} & \mathcal{C}_{22} & \cdots & \mathcal{C}_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{C}_{N1} & \mathcal{C}_{N2} & \cdots & \mathcal{C}_{NM} \end{bmatrix}}_{\mathcal{C}[\boldsymbol{k}, \boldsymbol{k}']} \begin{bmatrix} \hat{F}_1 \\ \hat{F}_2 \\ \vdots \\ \hat{F}_M \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_N \end{bmatrix}.$$

[1]One might think that we also need to multiply by a rect function in (5). But, since S has compact support, $FS * C$ has compact support. So multiplication with a rect would be superfluous.

[2]The convolution theorem is discussed in section 2.

One can form the matrix $\mathcal{C}[\boldsymbol{k},\boldsymbol{k}']$. However, for images or volumes that matrix can consume a great deal of memory (even when it's stored as a sparse matrix). It may benefit us, then, to identify an algorithm to implement left multiplication by $\mathcal{C}[\boldsymbol{k},\boldsymbol{k}']$. A natural implementation of left multiplication by $\mathcal{C}[\boldsymbol{k},\boldsymbol{k}']$ is one that iterates over the Fourier estimates $\hat{F}$, multiplies each column of $\mathcal{C}[\boldsymbol{k},\boldsymbol{k}']$ by the corresponding estimate of $\hat{F}$, and sums all the results together. Note that this algorithm is embarrasingly parallelizable.

It is important to choose a good kernel function C. One might think to convolve with a rect function, which would be the simplest thing to do (it equates to nearest neighbor interpolation). This leads to multiplication by a sinc function in the destination domain. Since a sinc has significant values over a large segment of the domain (i.e. it dies out slowly), this leads to large aliasing artifacts. Additionally, the sinc function may be 0 in the Field of View which would make deapodization impossible. The theoretically ideal choice would be to convolve with a sinc in the source domain, which leads to multiplication by a rect in the destination domain. But this is extremely computationally intensive, and can't be accomplished theoretically since we won't retain an infinite number of values. A great choice of convolution kernel is the Kaiser Bessel function [2, 3]. A multidimensional kernel can be constructed as a separable product of 1D kernels.

Equation (6) represents the simplest form of Gridding: sample in the Fourier domain, perform a convolution with C, sample on a uniform grid, and perform an DFT^{-1}. Notice, though, that in equation (7) the reconstruction $\hat{f}$ does not equal f; the differences are all corruptions (visualized in Figure 1). These corruptions include convolution by s, apodization (or multiplication) by c, and aliasing due to convolution with $\text{III}_{\mathsf{FOV}}$; we'll now alter the gridding algorithm to address each of these corruptions.

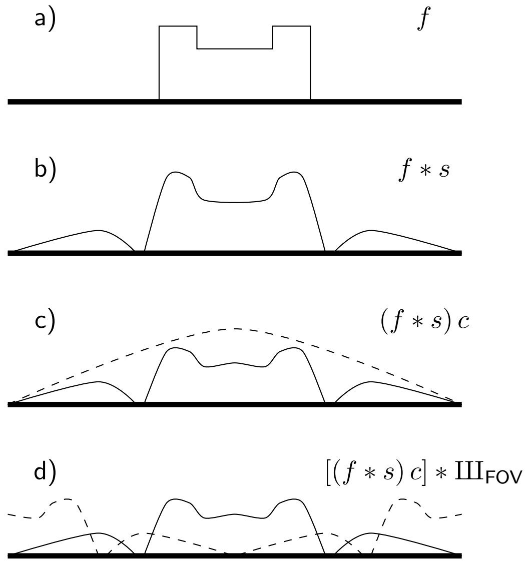

Figure 1: (a) The true function f. (b) f after convolution with s. The sharp edges become rounded and sidelobe artifacts appear. (c) Multiplication by the apodization function c. (d) Aliasing due to convolution with $\text{III}_{\mathsf{FOV}}$. The final function would be the sum of the dashed and solid lines of subfigure (d).

3.1 Reducing Aliasing Artifacts

In (6), we chose to sample on a grid with spacing Δk, where this spacing leads to the Field of View in the destination domain that we desired. But this was not required. We could choose to reconstruct on a grid with smaller spacing, leading to a larger than desired Field of View, and then crop out the center region corresponding to our desired Field of View. This is called "oversampling".

Suppose that, with oversampling, the number of samples is $\tilde{N} > N$; then the oversampling factor is $\alpha = \tilde{N}/N$. The gridding algorithm becomes

$$\begin{aligned}\hat{f}(n) &= \mathsf{DFT}^{-1}\left\{\mathcal{T}\left[(F\,S * C)\,(\Delta k/\alpha)\,\text{III}_{\Delta k/\alpha}\right]\right\}[n] \\ &\approx \left(\left[(f * s)\cdot c\right] * \text{III}_{\alpha\mathsf{FOV}}\right)(n).\end{aligned}$$

The benefit can be seen in figure 2. The aliased sidelobes are pushed farther out and so they have less impact in the central region. After cropping, most of the aliasing artifacts are eliminated. Note that with oversampling, the circular convolution operator has been changed; it now maps $\mathbb{C}^M \to \mathbb{C}^{\alpha N}$ and can be denoted $\mathcal{C}[\boldsymbol{k}, \boldsymbol{k}'']$, where $\boldsymbol{k}''$ is a vector of the k-space coordinates on the oversampled Cartesian grid (which is a larger vector than $\boldsymbol{k}'$).

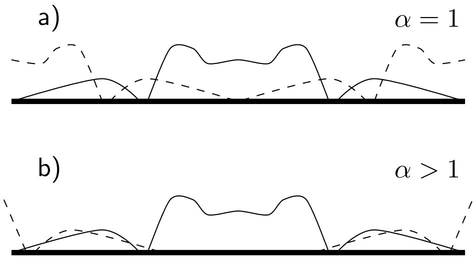

Figure 2: (a) Aliasing artifacts shown with oversampling factor $\alpha = 1$. (b) Aliasing artifacts moved farther away by increasing α.

3.2 Deapodization

We will now address the corruption of apodization by c. After reviewing (7), one would hope that simply dividing $\hat{f}(n)$ by $c(n)/(\alpha N)$ for all n would resolve the issue. This is a deconvolution in the source domain[3], and it's close to what we want, but not quite. Let $\boldsymbol{c} = (c(0), c(1), \ldots, c(\alpha N - 1))$; then

$$\frac{\alpha N}{c(n)} \left([(f * s) \cdot c] * \text{Ш}_{\alpha \mathsf{FOV}} \right)(n) = \mathsf{DFT}^{-1} \left\{ \mathsf{DFT} \left\{ \frac{1}{\boldsymbol{c}} \right\} \circledast \mathcal{T} \left[(F\, S * C)(\Delta k/\alpha)\, \text{Ш}_{\Delta k/\alpha} \right] \right\} [n],$$

where $1/\boldsymbol{c}$ is a vector of the values of $\boldsymbol{c}$ inverted. From here, the problem becomes apparent: we are performing a convolution with C but a circular deconvolution. This works ok, but can be significantly improved. Instead of convolving by C, we alter the gridding algorithm to perform a circular convolution by C. That is, whenever convolution by C extends beyond the bounds $[-0.5, 0.5)$, we *wrap around*[4]. Thus, the gridding algorithm becomes

$$\begin{aligned} \hat{f}(n) &= \frac{\alpha N}{c(n)} \mathsf{DFT}^{-1} \left\{ \mathcal{T} \left[(F\, S \circledast C)(\Delta k/\alpha)\, \text{Ш}_{\Delta k/\alpha} \right] \right\} [n] \\ &\approx \left([f * s] * \text{Ш}_{\alpha \mathsf{FOV}} \right)(n). \end{aligned} \tag{8}$$

Dividing by c is called "deapodization"; it boosts the intensities that were apodized by the convolution with C. The division by c and the multiplication by c approximately cancel each other out. However, any aliasing artifacts that remain and noise that is present will also get boosted up; hopefully these are small.

Since $\Delta k = 1/N$, we can simplify equation (8) by cancelling out the two αNs as follows:

$$\hat{f}(n) = \frac{1}{c(n)} \mathsf{DFT}^{-1} \left\{ \mathcal{T} \left[(F\, S \circledast C) \text{Ш}_{\Delta k/\alpha} \right] \right\} [n].$$

3.3 Density Compensation

We will now address the corruption due to the sampling function. The problem arises when the density of sample points in the source domain is not uniform; thus, the name of the algorithm to address this corruption is *Density compensation*. The key idea is to alter the sampling function; instead of making the sampling function a sum of impulses, we make the sampling function a sum of weighted impulses[5]: $S_w(k) = \sum_i w_i\, \delta\,(k - k_j)$. The question then becomes, how do we determine $w \in \mathbb{R}^M$?

Here, we will present the method of Pipe et al. [4]. The trick is to realize that if $f = \delta$ then $F = 1$ (i.e. F is the constant function 1). In this case, after convolution with C, we would like a value of 1 everywhere in

[3]Recall the convolution theorems: $\mathsf{DFT}(a \circledast b) = \mathsf{diag}(\mathsf{DFT}(a))\ \mathsf{DFT}(b)$ and $\mathsf{DFT}(\mathsf{diag}(a)\, b) = (1/N)\mathsf{DFT}(a) \circledast \mathsf{DFT}(b)$, where $a, b \in \mathbb{C}^N$.

[4]This wrap around can be defined rigorously as $(a \circledast C)(k) = (a * C)(((k + 0.5) \mod 1) - 0.5)$. But this definition is not insightful; just realize that when you want a value off the edge of the domain, you wrap around to the other side.

[5]This is called pre-density compensation. There also exists a post-density compensation which won't be discussed in this document. Refer to [1] for more details.

the source domain. Since we don't have samples everywhere, we desire instead that $S_w * C \approx 1$. With this goal in mind, Pipe et al. designed the following iteration:

$$S_w^{(i+1)} = \frac{S_w^{(i)}}{S_w^{(i)} * C} \tag{9}$$

where $S_w^{(i+1)} = 0$ whenever the denominator equals 0. Let $\mathcal{T}_S \left[S_w^{(i+1)} \right] = w^{(i+1)}$; i.e. it is a vector of the weights at the non-zero locations of the distribution. Note that $\mathcal{T}_S$ is an isomorphism from the set of all possible S_w to $\mathbb{R}^M$ (where we are assuming pointwise addition and scalar multiplication). If $S_w^{(i)} * C \approx 1$ at all nonzero values of $S_w^{(i)}$ then $S_w^{(i+1)} \approx S_w^{(i)}$; that is, the iteration has converged. Pipe et al. specify that this iteration should be run 10 times. Then, the final weight vector is $w = \mathcal{T}_S \left[S_w^{(10)} \right]$.

Hopefully, after density compensation, $s_w = \mathcal{F}^{-1}\{S_w\}$ approximates the Dirac delta δ.

3.4 Final Gridding Algorithm

The final gridding algorithm is shown in equation (10) and summarized in Algorithm 1.

$$\hat{f}(n) = \frac{1}{c(n)} \mathsf{DFT}^{-1} \left\{ \mathcal{T} \left[(F\, S_w \circledast C) Ш_{\Delta k} \right] \right\} [n]. \tag{10}$$

Note that the computational complexity of Gridding can be reduced by performing the convolution with C using a look-up table. When doing so, an interpolation is required; Beatty et al. show that a linear interpolation suffices [2].

Algorithm 1: Gridding

```
**Inputs:** $(k_1, \ldots, k_M))$, $(F(k_1), \ldots, F(k_M))$, $\alpha > 1$
**Step 1:** Determine the weighting vector $w$ using iteration (9).
**Step 2:** Determine the result of convolving the set $FS_w$ with the convolution kernel $C$ and sampling
  on the grid points. Note that the grid points are the vertices of the oversampled grid.
**Step 3:** Perform an Inverse Discrete Fourier Transform.
**Step 4:** Crop out the center region, which corresponds to the desired Field of View.
**Step 5:** Deapodize by $\boldsymbol{c}$.
```

Gridding is a linear transformation $\mathcal{G}$, which can be expressed as

$$\mathcal{G} = \mathsf{diag}\,(\boldsymbol{c})^{-1}\ \mathcal{K}\ \mathsf{DFT}^{-1}\ \mathcal{C}[\boldsymbol{k}, \boldsymbol{k}'']\ \mathsf{diag}(w),$$

where $\mathcal{K} : \mathbb{C}^{\alpha N} \rightarrow \mathbb{C}^N$ is the cropping transformation. Note that the cropping and deapodization could happen in the reverse order, but a slight improvement in efficiency is had by cropping first and then deapodizing (there are fewer divisions).

4 Inverse Gridding

Inverse Gridding is an algorithm that approximates the Fourier Transform when the source coordinates $(x_1, \ldots, x_N)$ lie on a uniform grid but the destination coordinates $(k_1, \ldots, k_M)$ do not[6]. The question becomes, given the coordinates $(x_1, \ldots, x_N)$ and values $(f(x_1), \ldots, f(x_N))$, how can we approximate the destination domain values $(F(k_1), \ldots, F(k_M))$?

The sampled data is $f\, Ш\, \Pi_D$ where the samples are contained within $(-D/2, D/2)$, where Π_D is a rect function of width D. Define t as the mapping such that $t\,(f\, Ш\, \Pi_D)\,[n]$ is the weight of the nth delta. Then after sampling, the Discrete Fourier Transform yields

$$\mathcal{T}^{-1}\, \mathsf{DFT}\{t\,(f\, Ш\, \Pi_D)\} = (F * Ш * b)\, Ш_{\Delta k}\, \Pi$$

[6]It's important to note that Inverse Gridding is not the inverse of Gridding. Instead, Inverse Gridding is an algorithm that approximates the inverse of Gridding.

where $b(k) = D\,\text{sinc}(D\,k)$ and $\Delta k = 1/N$. The convolution of F with III is aliasing and the convolution with the sinc is a blur.

We assume that f has compact support and that the support is a subset of $(-D/2, D/2)$. With this assumption, $F * b = F$. Thus, we can eliminate b from the above equation:

$$\mathcal{T}^{-1}\,\text{DFT}\{t\,(f\,\text{III}\,\Pi_D)\} = (F * \text{III})\,\text{III}_{\Delta k}\,\Pi.$$

At this point we have uniformly spaced samples in the destination domain, but what we want is samples at a specific set of destination domain coordinates $\{k_1, \ldots, k_M\} \subset [-1/2, 1/2)$. Again, we use the *push not pull* idea and circularly convolve with the kernel C:

$$\mathcal{T}^{-1}\,\text{DFT}\{t\,(f\,\text{III}\,\Pi_D)\} \circledast C = (F * \text{III})\,\text{III}_{\Delta k}\,\Pi \circledast C.$$

Finally, we sample in the destination domain at the coordinates of interest:

$$\left[\mathcal{T}^{-1}\,\text{DFT}\{t\,(f\,\text{III}\,\Pi_D)\} \circledast C\right] S = \left[(F * \text{III})\,\text{III}_{\Delta k}\,\Pi \circledast C\right] S.$$

Note that for some function h, hS is a distribution with M nonzero values at locations in the set $\{k_j : j = 1, \ldots, M\}$.

We have arrived at the simplest form of inverse gridding:

$$\hat{F}(k_m) = \mathcal{T}_S\left(\left[\mathcal{T}^{-1}\,\text{DFT}\{t\,(f\,\text{III}\,\Pi_D)\} \circledast C\right] S\right)[m].$$

The procedure is to compute the DFT of the data, circularly convolve with the kernel C, and sample on the desired coordinates.

Again, the estimates $\hat{F}$ are not equal to F, and the differences are corruptions: aliasing due to the convolution with III, and blurring due to the circular convolution with C. Unfortunately, there's nothing we can do about the aliasing. In the next subsection, we will discuss how to address the corruption of the circular convolution with C.

Note that density compensation is not needed with inverse gridding; intuitively this makes sense since the samples that are convolved with C are evenly spread out in the destination domain.

4.1 Zero Padding

The convolution kernel C may be used to interpolate distances that are rather far apart from each other in the case where Δk is large. One can first pad with zeros in the source domain prior to the DFT, which corresponds to sinc interpolation in the destination domain, which is a very accurate form of interpolation[7]. After padding by an amount α, the distances between grid points in the destination domain is $\Delta k/\alpha$. And the distances between the final sample locations and the nearest grid points are not so far away. Zero padding is incorporated into inverse gridding as follows:

$$\hat{F}(k_m) = \mathcal{T}_S\left(\left[\mathcal{T}^{-1}\,\text{DFT}\{\mathcal{K}^T t\,(f\,\text{III}\,\Pi_D)\} \circledast C\right] S\right)[m],$$

where we have used the fact that padding is the adjoint of cropping.

4.2 Pre-emphasis

Like deapodization in Gridding, we wish to perform a deconvolution by C in the destination domain. Do do so, we will perform an appropriate point-wise multiplication in the source domain. In accordance with the convolution theorem, we will divide by c in the source domain; this is called *pre-emphasis*. Let $x = \mathcal{K}^T t\,(f\,\text{III}\,\Pi_D)$; that is, we are estimating Fourier values using the sample vector x. If we perform a Hadamard multiplication and a Hadamard division of x by $\boldsymbol{c}$, since $\boldsymbol{c}$ does not equal 0 anywhere, x remains unchanged. Let us compute the DFT of this expression:

$$\text{DFT}\{x\} = \text{DFT}\left\{\frac{x}{\boldsymbol{c}} \cdot \boldsymbol{c}\right\} = \frac{1}{N}\text{DFT}\left\{\frac{x}{\boldsymbol{c}}\right\} \circledast C.$$

[7]See `https://nicholasdwork.com/tutorials/nyquistInterp.pdf` for details.

This shows us that to compensate for the errors introduced by the circular convolution with C, we must pre-emphasize with $\alpha\, N\, \boldsymbol{c}$, as follows:

$$\hat{F}(k_m) = \mathcal{T}_S\left(\left[\mathcal{T}^{-1}\,\mathsf{DFT}\left\{\mathcal{K}^T\ \mathsf{diag}\left(\alpha\ N\ \boldsymbol{c}\right)^{-1}\ t\left(f\,Ш\,\Pi_D\right)\right\}\circledast C\right]S\right)[m],$$

This algorithm is called pre-emphasis since the effect is to change the emphases (relative magnitudes) of different samples of f.

4.3 Final Inverse Gridding Algorithm

Putting it all together, we arrive at the final Inverse Gridding algorithm, summarized in Algorithm 2:

$$\hat{F}(k_m) = \mathcal{T}_S\left(\left[\mathcal{T}^{-1}\,\mathsf{DFT}\left\{\mathcal{K}^T\ \mathsf{diag}\left(\alpha\ N\ \boldsymbol{c}\right)^{-1}\ t\left(f\,Ш\,\Pi_D\right)\right\}\circledast C\right]S\right)[m].$$

As with Gridding, the computational complexity of Inverse Gridding can be reduced by performing the convolution with C using a look-up table and linear interpolation.

```
Algorithm 2: Inverse Gridding
  Inputs: $(x_1, \ldots, x_N)$, $(f(x_1), \ldots, f(x_N))$
  Step 1: Pre-emphasize by $\alpha\ N\ \boldsymbol{c}$.
  Step 2: Pad the array $x$ with zeros to be a final size of $\alpha N$.
  Step 3: Compute the DFT.
  Step 4: Perform a circular convolution with $C$ and sample on the points of interest.
```

Let the input be represented by $x = t\left(f\,Ш\,\Pi_D\right) \in \mathbb{C}^N$. Then the Inverse Gridding transformation can be written as

$$\mathcal{T}\hat{F} = \mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]\ \mathsf{DFT}\ \mathcal{K}^T\ \mathsf{diag}\left(\alpha\ N\ \boldsymbol{c}\right)^{-1}\ x,$$

where $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}] \in \mathbb{R}^{M\times N}$ is the circular convolution with the kernel C. The value of $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]_{mn}$ is the kernel C evaluated at $\|k_m - k''_n\|_2$. Inverse Gridding is a linear transformation $\mathcal{G}_I$, which can be written as

$$\mathcal{G}_I = \mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]\ \mathsf{DFT}\ \mathcal{K}^T\ \mathsf{diag}\left(\alpha\ N\ \boldsymbol{c}\right)^{-1}. \tag{11}$$

4.4 Adjoint of Inverse Gridding

Note that $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}] = \mathcal{C}[\boldsymbol{k}, \boldsymbol{k}'']^T$. Therefore, from equation (11), it immediately follows that the adjoint of Inverse Gridding is

$$\mathcal{G}_I^* = \mathsf{diag}\left(\boldsymbol{c}\right)^{-1}\ \mathcal{K}\ \mathsf{DFT}^{-1}\ \mathcal{C}[\boldsymbol{k}, \boldsymbol{k}''],$$

where we have used the fact that $\mathsf{DFT}^* = \alpha\,N\,\mathsf{DFT}^{-1}$.

Here we see another beautiful relationship; the adjoint of Inverse Gridding is a subset of the Gridding transformation:

$$\mathcal{G} = \mathsf{diag}\left(\boldsymbol{c}\right)^{-1}\ \mathcal{K}\ \mathsf{DFT}^{-1}\ \mathcal{C}[\boldsymbol{k}, \boldsymbol{k}'']\ \mathsf{diag}(w) = \mathcal{G}_I^*\ \mathsf{diag}(w).$$

Gridding is density compensation weighting followed by the adjoint of Inverse Gridding. The algorithm for the adjoint of Inverse Gridding is shown in Algorithm 3.

```
Algorithm 3: Adjoint of Inverse Gridding
  Inputs: $(y_1, \ldots, y_M)$
  Step 1: Compute the circular convolution with the kernel $C$ and sample on grid points.
  Step 2: Compute the $\mathsf{DFT}^{-1}$.
  Step 3: Crop the result to size $N$.
  Step 4: Deapodize by $\boldsymbol{c}$.
```

Consider left multiplication by $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]^T$ as shown in equation (12). A natural implementation of this is to iterate over the Fourier coefficients, and compute the inner product between the rows of $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]^T$ (or the columns of $\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]$) and the Fourier coefficients γ. Note that this algorithm is embarrassingly parallelizable.

$$\begin{bmatrix} \hat{F}_1 \\ \hat{F}_2 \\ \vdots \\ \hat{F}_M \end{bmatrix} = \underbrace{\begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} & \cdots & \mathcal{C}_{1N} \\ \mathcal{C}_{21} & \mathcal{C}_{22} & \cdots & \mathcal{C}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{C}_{M1} & \mathcal{C}_{M2} & \cdots & \mathcal{C}_{MN} \end{bmatrix}}_{\mathcal{C}[\boldsymbol{k}'', \boldsymbol{k}]} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_N \end{bmatrix}. \tag{12}$$

References

[1] John M. Pauly. Medical image reconstruction. `http://web.stanford.edu/class/ee369c/`. Accessed: 2016-02-15.

[2] Philip J. Beatty, Dwight G. Nishimura, and John M. Pauly. s. *Medical Imaging, IEEE Transactions on*, 24(6):799–808, 2005.

[3] John I. Jackson, Craig H. Meyer, Dwight G. Nishimura, and Albert Macovski. Selection of a convolution function for Fourier inversion using gridding [computerised tomography application]. *Medical Imaging, IEEE Transactions on*, 10(3):473–478, 1991.

[4] James G. Pipe and Padmanabhan Menon. Sampling density compensation in MRI: rationale and an iterative numerical solution. *Magnetic Resonance in Medicine*, 41(1):179–186, 1999.