Second Order Solution of Fritz John's Ultrahyperbolic PDE for Volumetric Computed Tomography

Jicun Hu [*] Chris Ingrassia [†] Svenja Lowitzsch [‡] Jang Park [§]
Angel Pineda [¶] Daniel Reynolds [||] Nicholas Valdivia [**]

July 28, 2000

Industry Mentor: Dr. S.K. Patch, GE Medical Systems [††]

[*]Marquette University
[†]New York University
[‡]Texas A&M University
[§]Northwestern University
[¶]University of Arizona
[||]Rice Univeristy
[**]Wichita State University
[††]contact email: sarah.patch@med.ge.com

1 Introduction

Volumetric computed tomography (VCT) is a possible method for taking CT scans in less time and with less wear to the physical components. It is possible that by finding a feasible method for dealing with this system, we could greatly speed up the process necessary to provide information of this type.

We compute unmeasured volumetric computed tomography views from a set of measured views. This set of measured data comes from moving our focal spot around an object in a helical pattern. At a pre-determined set of points on this helix, we take a conical set of line integrals measuring an object's linear attenuation coefficient. This effectively provides us with a four-dimensional set of data, parameterized by the six variables ξ_i, and η_i, for $i = 1, 2, 3$. Thankfully, the VCT data must satisfy Fritz-John's ultrahyperbolic PDE [1]

$$\left(\frac{\partial^2}{\partial \eta_i \partial \xi_j} - \frac{\partial^2}{\partial \eta_j \partial \xi_i} \right) u(\xi; \eta) = 0 \text{ for } i, j = 1, 2, 3. \tag{1}$$

Thus by solving a characteristic boundary value problem for these three equations, unmeasured views corresponding to neighboring helices interleaved with our source trajectory may be computed. Initial work on this problem can be found at [2]

2 Phantom Generation

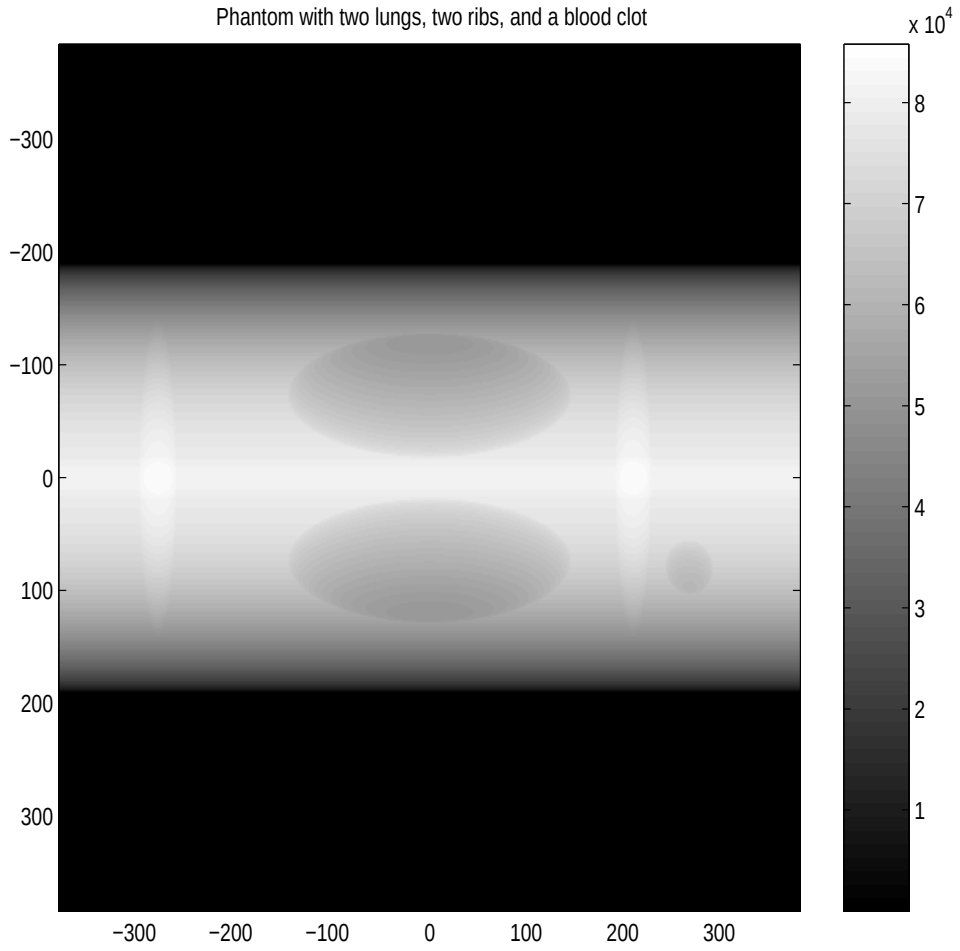
Since 3rd generation machines do not yet exist, we created a synthetic, noise-free phantom as initial data. It consists of two lungs, two ribs, and a blood clot. The phantom is a 512x512 equidistant grid of pixels, where the intensity of each pixel represents the X-ray attenuation through the patient at one point. For this particular phantom, the focal spot is at $\theta = 0$, $z = 0$. Figure (2) shows the phantom used.

3 Solution of Fritz John's Ultrahyperbolic PDE

The system we are dealing with is a 3^{rd} Generation VCT system. Here we may undergo a change of coordinates since in this system the radii of rotation are fixed. Hence we only measure our function u for varying θ, z, α_1 and α_2, where (θ, z) parameterize our focal spot positions and (α_1, α_2) give pixel location on our detector. Therefore, all three of the conditions in 1 reduce down to the single constraint:

$$\frac{\partial^2 u}{\partial \alpha_2 \partial \theta} - \rho \frac{\partial^2 u}{\partial z \partial \alpha_1} = \frac{-1}{(\rho + d)} \left[2\alpha_1 \frac{\partial u}{\partial \alpha_2} + \alpha_1 \alpha_2 \frac{\partial^2 u}{\partial \alpha_2^2} + \left((\rho + d)^2 + \alpha_1^2 \right) \frac{\partial^2 u}{\partial \alpha_2 \partial \alpha_1} \right] \tag{2}$$

We may now transform this PDE into a coupled system of ODE's. Since u has compact support in α_1, α_2 its Fourier transform with respect to these variables is smooth. By taking the Fourier transform of (2) with respect to α_1, α_2, having dual variables κ_1, κ_2, the left hand side becomes a directional derivative. The right hand side then becomes a function of the Fourier transform $\hat{u}$ and its derivatives with respect to κ_1, κ_2.


Phantom with two lungs, two ribs, and a blood clot
x 10^4
8
7
6
5
4
3
2
1
−300
−200
−100
0
100
200
300
−300
−200
−100
0
100
200
300

$$(-R\kappa_1, \kappa_2) \cdot (\hat{u}_z, \hat{u}_\theta) = \frac{\kappa_2}{i\pi(R+D)} \left[2\hat{u}_{\kappa_1} - \kappa_2 \hat{u}_{\kappa_1,\kappa_2} + \kappa_1 \left(\pi^2 (R+D)^2 \hat{u} - \hat{u}_{\kappa_1,\kappa_1} \right) \right] \tag{3}$$

Or more simply written, given a position (κ_1, κ_2) in our Fourier Domain, we need only solve the ODE

$$\frac{d\hat{u}}{ds} = f(\hat{u}, \hat{u}_{\kappa_2}, \hat{u}_{\kappa_1,\kappa_2}, \hat{u}_{\kappa_2,\kappa_2}) \tag{4}$$

along the direction $(-R\kappa_1, \kappa_2)$ in our Θ, z plane. In this solve, we will end up moving all of the information on the line given by (κ_2, κ_1) in the Fourier domain of their original position to their corresponding points in the Fourier domain of the position we wish to compute. This leads to a choice of the directions in which we want to propagate our data.

We choose to solve this ODE using a standard second-order Runge-Kutta formula

$$\begin{aligned} \hat{u}^{n+1/2} &= \hat{u}^n + \tfrac{h}{2} f(\hat{u}^n) \\ \hat{u}^{n+1} &= \hat{u}^n + h f(\hat{u}^{n+1/2}). \end{aligned} \tag{5}$$

Note that in order to achieve this higher-order approximation, we need to not only compute the right-hand side at our initial point in this direction, but also at any intermediate points in the same direction. Because of this, we need to choose these directions $(-R\kappa_1, \kappa_2)$ wisely. Specifically with this in mind, we choose to solve our ODE in directions that would take us from one gridpoint on the current helix to a gridpoint on the next helix. The direction in our Fourier space corresponding to the direction $(-R\kappa_1, \kappa_2)$ in the Θ, z plane is (κ_2, κ_1).

These directions, though necessary for the ODE solution, provide for much of the difficulty with this method. To solve the ode along these lines, we need to have information about our function $\hat{u}$ and its derivatives on these radial lines, yet the initial measurements provide function data on a cartesian grid. We initially transfer this cartesian data to our polar lines through interpolation, which will be discussed in further sections. After we have our information transferred onto these radial lines, we keep the information on this same set of lines throughout the entire solution process.

The next difficulty encountered is that the forcing term of our ODE requires derivatives at these points is in the directions κ_1 and κ_2, though we now have our data stored radially. Thus to calculate these derivatives, we will need to have enough of these lines in the κ_1, κ_2 plane to approximate our derivatives well. These derivative approximations will also be discussed later.

Because we can only work with Fourier space data on radial lines, our other consideration in choosing solving directions is that we must achieve enough information about the Fourier space to both retain image data and compute the right-hand side. By increasing the number of directions in which we solve the ODE, we can then increase the sampling of points in the Fourier domain. Following are two figures which relate our sampling in the κ_1, κ_2 plane to the directions we solve along in the Θ, z plane (check system parameters in the Appendix).

A difficulty in choosing these lines is that given the system parameters of the scanner, we found that there was a tradeoff between the stepsize for our ODE solver and our sampling in the Fourier domain. By looking at the distribution of lines in the κ_1, κ_2 plane, it is easily seen that some areas of the space are not sampled as well as others. It is in these regions that the

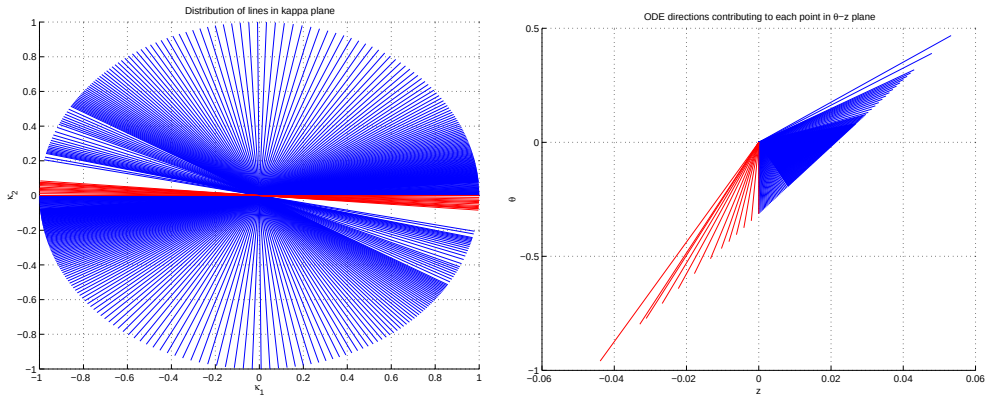


Figure 1: Lines sampling κ_1, κ_2 plane, and corresponding ODE directions in Θ, z plane.

tradeoff occurrs. Given the parameters at hand, the distribution shown achieved our best idea of this balance.

One final note: we must use twice the normal number of points on our half-step helix than required on a full helix. This is because for neighboring views to contribute to the same view that we wish to compute, we must generate views at half the spacing. Though this seems like a doubling of the information and storage needed on the temporary helix, in fact each of these half-step views will only contain half of the full set of lines. Thus our temporary (half step) helix will require twice the number of views, but the same storage as our full step helices.

3.1 Derivative Estimation for Forcing Term

After we obtain the matrix that contain the information of the radial lines then we can use the Jacobian tranformation

$$\frac{\partial}{\partial \kappa_1} = -\frac{sin(\theta)}{r}\frac{\partial}{\partial \theta} + \cos(\theta)\frac{\partial}{\partial r}$$

$$\frac{\partial^2}{\partial \kappa_1^2} = \frac{sin^2(\theta)}{r^2}\frac{\partial^2}{\partial \theta^2} - \frac{sin(2\theta)}{r}\frac{\partial^2}{\partial r \partial \theta} + cos^2(\theta)\frac{\partial^2}{\partial r^2} + \frac{sin(2\theta)}{r^2}\frac{\partial}{\partial \theta} + \frac{sin^2(\theta)}{r}\frac{\partial}{\partial r}$$

$$\frac{\partial^2}{\partial \kappa_1 \partial \kappa_2} = -\frac{sin(2\theta)}{2r^2}\frac{\partial^2}{\partial \theta^2} + \frac{cos(2\theta)}{r}\frac{\partial^2}{\partial r \partial \theta} + \frac{sin(2\theta)}{2}\frac{\partial^2}{\partial r^2} - \frac{cos(2\theta)}{r^2}\frac{\partial}{\partial \theta} - \frac{sin(2\theta)}{2r}\frac{\partial}{\partial r}$$

These formulas are machine accurate when we have the correct information of the radial derivatives $\partial/\partial\theta$,$\partial/\partial r$, $\partial^2/\partial\theta\partial r$, $\partial^2/\partial\theta^2$ and $\partial^2/\partial r^2$. Once we obtain partial derivatives $\partial/\partial\kappa_1$, $\partial^2/\partial\kappa_1\partial\kappa_2$ and $\partial^2/\partial\kappa_1^2$, we can calculate the forcing term.

The forcing term will be needed for the ODE solver to calculate from one helix to the next one. Its formula is

$$F\left(\kappa_1, \kappa_2, \widehat{U}, \widehat{U}_{\kappa_1}, \widehat{U}_{\kappa_1\kappa_2}, \widehat{U}_{\kappa_1\kappa_1}\right) = \frac{\kappa_2}{i\pi\left(R+D\right)\tau}\left(2\widehat{U}_{\kappa_1} - \kappa_2\widehat{U}_{\kappa_1\kappa_2} + \pi^2\left(R+D\right)^2\kappa_1\widehat{U} - \kappa_1\widehat{U}_{\kappa_1\kappa_1}\right)$$

In calculating the polar derivatives, we made use of Matlab's spline toolbox. In this way, we first got the piecewise polynomial representation of the 2-D grid of data in the radial and angular directions. From there, we calculate the derivatives from the spline representation. "Not a knot" conditions were used in the treatment of boundary data. Computations were checked using the function, $\hat{u} = \frac{sin(\kappa_1)}{\kappa_1}\frac{sin(\kappa_2)}{\kappa_2}$, where $\kappa_1 = rcos(\phi)$, and $\kappa_2 = rsin(\phi)$. This function was chosen because it is not radially symmetric and the ϕ derivatives would provide a good test. The derivatives were computed using the piecewise polinomial form and then checked for accuracy.

The phi values were non-uniformly distributed from about $\phi = 0$ to $\phi = \pi$. The radial axis was distributed evenly with 257 points which means that r_{max} decreases linearly with dr, the radial spacing. We start from $r = 0$ with grid spacing $dr = 1.297$, originally, and later this parameter, dr, was varied. These parameters were chosen to reflect actual values that will be used in the computation. Here are some relative L_2 errors and another corresponding error estimate, E_2:

$L_2 = \frac{||U_{comp} - U_{true}||}{||U_{true}||}$,

$E_2 = sup(min(|U_{comp} - U_{true}|, \frac{|U_{comp} - U_{true}|}{|U_{true}|}))$.

derivative	dr	=1.297	dr	=1.297/5	dr	=1.297/10	dr	=1.297/20
	L_2	E_2	L_2	E_2	L_2	E_2	L_2	E_2
u_ϕ	0.4022	.4320	0.0855	0.2190	7.26E-3	2.47E-2	4.02E-4	0.0015
u_r	0.0606	0.0361	5.85E-4	8.05E-4	5.29E-5	1.03E-4	4.70E-6	1.30E-5
$u_{r\phi}$	0.4254	0.4311	9.02E-2	2.36E-1	5.93E-3	1.69E-2	3.53E-4	1.03E-3
$u_{\phi\phi}$	0.5709	38.0000	1.14E-1	4.4500	5.61E-2	0.8610	1.62E-2	1.95E-1
u_{rr}	0.2657	0.1254	1.21E-2	1.22E-2	2.32E-3	3.13E-3	4.53E-4	7.88E-4

The error for the radial derivatives are much smaller than the angular derivatives. We find that the error is concentrated in a cone near $\phi = \frac{\pi}{2}$, and grows with increasing r. As we step down dr and r_{max}, we find that we get much better agreement with the actual values. We interpret this as meaning that low frequency components of the derivatives in both the ϕ and r directions can be better approximated by decreasing dr.

This next table was constructed by decreasing dr and increasing the number of points along the radial direction so that r_{max} is fixed. For this calculation, the point corresponding to $r = 0$ was taken out before finding the errors. This is done because the derivatives in the κ_1 and κ_2 directions need not be computed for this point.

derivative	dr	=1.297	dr	=1.297/2	dr	=1.297/4
	L_2	E_2	L_2	E_2	L_2	E_2
u_ϕ	0.4020	0.4320	0.4020	0.4330	0.4022	0.4330
u_r	0.0139	0.0069	0.0032	2.57E-3	3.36E-4	4.07E-4
$u_{r\phi}$	0.4250	0.4310	0.4260	0.4330	0.4260	0.4330
$u_{\phi\phi}$	0.5710	38.0000	0.5710	41.2000	0.5710	41.8000
u_{rr}	0.1056	0.0267	0.0289	7.22E-3	7.04E-3	3.02E-3

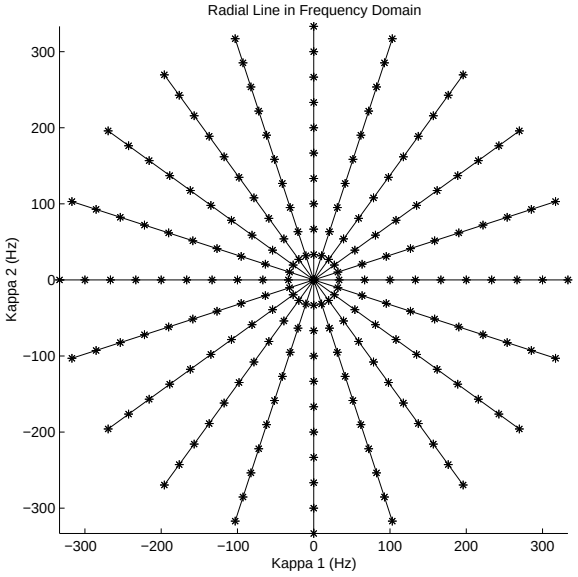
Here we see that decreasing the grid spacing while keeping the actual length, r_{max} fixed, results in very little change in the angular derivatives. The error for the angular derivatives comes from the fact that our radial lines in kappa space are not sufficient to interpolate our test function when the domain is fixed and large. This means that high frequency values did not

improve by lowering the spacing in r. However, low frequency components do improve with a significant decrease in dr. Due to parameter contraints, no corresponding computations were done with a denser set of angluar values.

4 Interpolation

4.1 Introduction

Two interpotion steps are involved in our solution. The first interpolation happened when we want to take the Fourier component along radial lines, (Fig.1.), which is not necessarily on the grid point in the Cartesian coordinates.



Another interpolation is involved when we want to recover the unmeasured projections in Cartesian coordiates while the fourier component is only available along radial lines. In this project,we used different approaches to do the interpolation. The methods used were: Simple interpolation based on neighboring points, interpolation based on Radial basis functions,interpolation based on Sinc basis functions and also interpolation based on Fourier bases.

In order to check the interpolation schemes, we apply it to the test function

$$f(x,y) = sinc(x)sinc(y) * sinc(x/10)sinc(y/10), \tag{6}$$

with $\vec{x} = (x,y)$. The sinc function behaves very similar to a Fourier tranform due to its oscillations. Therefore it seems reasonable to use it as a test function for our purposes. The locations of the data points are the same as the points we use for the interpolation of the phantom, i.e. the cartesian grid based on the interval $[-1000/3, 1000/3]$, with a stepsize of $\Delta k = \frac{2000/3}{512}$. We interpolate along the polar lines we also use for the phantom, and compare the interpolated data, f_{int}, with the measured data f_{meas} along these polar lines. The criteria of measuring the error are the following two errors:

$$E1 = \frac{\|f_{int} - f_{meas}\|_2}{\|f_{meas}\|_2} \tag{7}$$

$$E2 = \|min(B1, B2)\|_\infty, \tag{8}$$

where $B1 = abs(f_{meas} - f_{int}t)$ and $B2 = abs((f_int - f_meas)./f_meas)$.

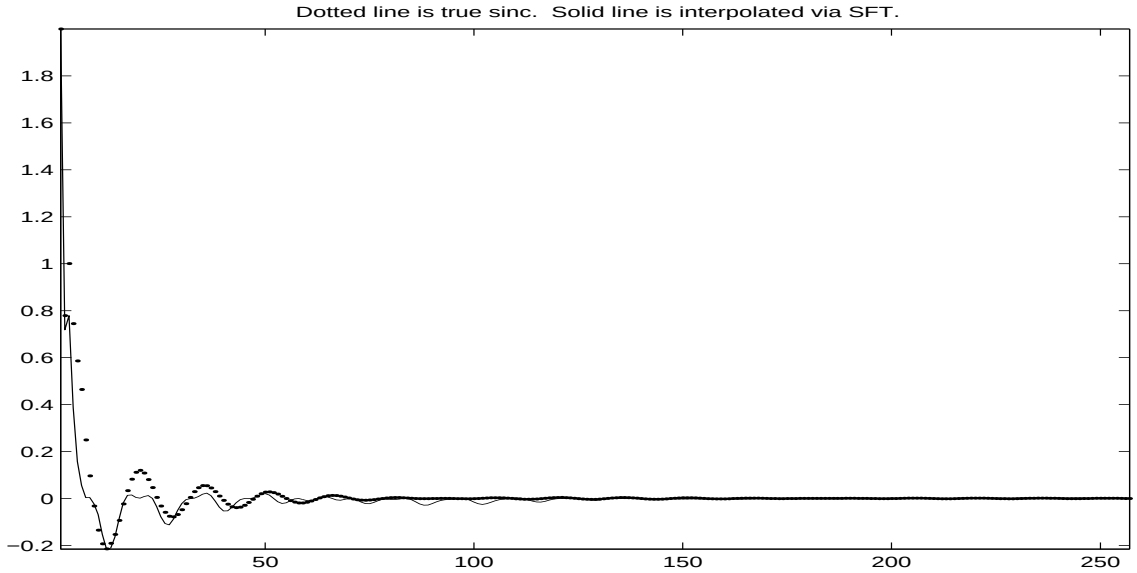
4.2 Slow Fourier Transform

Another interpolation option is the Slow Fourier Transform (SFT). The idea is to determine $\hat{u}$ along the radial lines from u data on the cartesian grid. This is perhaps the most natural routine since it incorporates a Fourier basis. For any prescribed radial direction, $\vec{\kappa} = [\kappa_1 \ \kappa_2]$,

$$\hat{u}(\vec{\kappa}) = \sum_{\vec{\alpha}} u(\vec{\alpha}) e^{-2\pi i(\vec{\kappa}\cdot\vec{\alpha})/N} \tag{9}$$

where $u(\vec{\alpha})$ is the phantom data at the cartesian point indicated by $\vec{\alpha}$. It is clear that SFT sums over the entire cartesian grid to evaluate $\hat{u}$ for a single point on a radial line. Assuming a grid of size 512x512, we typically consider 512 points for each polar line. It follows that SFT is $\mathcal{O}(512^3 * Nlines)$. In MATLAB, most of the runtime is spent traversing the grid with two loops. We are able to vectorize (9) by replacing the loops with matrix operations, which MATLAB performs very efficiently. Despite attempts to optimize the code, the SFT is still the slowest procedure because it is the only global interpolation technique.

To verify this routine, it is important to realize that SFT simultaneously performs interpolation and a transformation into the Fourier domain. Hence, we cannot directly compare the algorithm with MATLAB's FFT function, since it does not perform interpolation. Furthermore, it is difficult to relate the other interpolation schemes to SFT because the others do not implement a basis transform. To work around this dilemma, we applied SFT to the Inverse FFT of the sinc, instead of the sinc itself, to obtain the interpolated data on the radial lines.



4.3 Interpolation Using a Basis of Sinc Functions

The rationale for choosing the sinc basis relies on the projection data u being well represented by a Fourier series where the $\hat{u}_{\kappa 1,\kappa 2}$ elements represent the coefficients:

Method	RunTime	Rel. Err	E2
RBF	12 min	0.0073	0.0986
SFT	58 min	0.4805	1.1266
Sinc	1 min	0.0200	0.1100
Splines	6 min	0.0091	0.0449

Table 1: Interpolation Comparison. Each method is listed with the associated runtime, relative error, and E2 norm.

$$u(x,y) \approx \sum_{\kappa 1}^{N}\sum_{\kappa 2}^{N} \hat{u}_{\kappa 1 \kappa 2}\, e^{2\pi i(x\,\kappa 1 + y\,\kappa 2)} \tag{10}$$

If the function u is band limited below the Nyquist frequency then this Fourier series representation is exact. Given any set of Fourier coefficients $\{\hat{u}_{\kappa 1,\kappa 2}\}$ we can estimate any other set of coefficients $\{\hat{u}_{\kappa 1_{int},\kappa 2_{int}}\}$ by using the Fourier series representation of $u(x,y)$:

$$\begin{aligned}
\hat{u}_{\kappa 1_{int}\kappa 2_{int}} &\approx (u, e^{2\pi i(x\,\kappa 1 + y\,\kappa 2)})_{L^2(\Omega)} && (11)\\
&= \sum_{\kappa 1}^{N}\sum_{\kappa 2}^{N} \hat{u}_{\kappa 1 \kappa 2} \int_{-\frac{L}{2}}^{\frac{L}{2}}\int_{-\frac{L}{2}}^{\frac{L}{2}} e^{2\pi i(x\,(\kappa 1 - \kappa 1_{int}) + y\,(\kappa 2 - \kappa 2_{int}))} dxdy && (12)\\
&= \sum_{\kappa 1}^{N}\sum_{\kappa 2}^{N} \hat{u}_{\kappa 1 \kappa 2} \frac{sin(\pi(\kappa 1 - \kappa 1_{int}))}{\pi(\kappa 1 - \kappa 1_{int})} \frac{sin(\pi(\kappa 2 - \kappa 2_{int}))}{\pi(\kappa 2 - \kappa 2_{int})} && (13)\\
&= \sum_{\kappa 1}^{N}\sum_{\kappa 2}^{N} \hat{u}_{\kappa 1 \kappa 2} sinc(\kappa 1 - \kappa 1_{int}) sinc(\kappa 2 - \kappa 2_{int}) && (14)
\end{aligned}$$

Even though the motivation and derivation uses the Fourier series, the interpolation occurs strictly in the frequency domain and can be localized by only using frequencies in the Fourier series whose frequencies are close to the desired interpolated frequency. In order to pick a set of Fourier components to use we look at the interpolating function (Figure2). A 16x16 subgrid of Fourier coefficients near the desired value were used for the interpolation.

4.4 Interpolation by Radial Basis Functions (RBF's)

In this section we will describe the procedure of interpolation with radial basis functions. More explicitly, we use the Gaussian function, i.e. $f(x) = e^{-t\,\|x\|^2}$, $t > 0$, as basis functions. Our data $\{\hat{u}_j\}_{j=1}^N$ is the Fourier transform of measured X-ray attenuation $\{u_j\}_{j=1}^N$ at a given set of two-dimensional points $K = \{\kappa_j\}_{j=1}^N$. Since u is a compactly supported function, the $\hat{u}$ is an analytic function and therefore C^∞. Hence, we would like the interpolation function to also be smooth. The Gaussian function is C^∞, which gives the motivation for using this specific interpolation scheme for our problem.

The interpolation scheme is as follows: given complex-valued data $\{\hat{u}_j\}_{j=1}^N$ at distinct points $K = \{\kappa_j\}_{j=1}^N$, which are contained in a compact subset D of $\mathbb{R}^2$, solve the system of equations

$$s_{\hat{u}}(\kappa_j) = \hat{u}_j, \quad 1 \le j \le N, \tag{15}$$

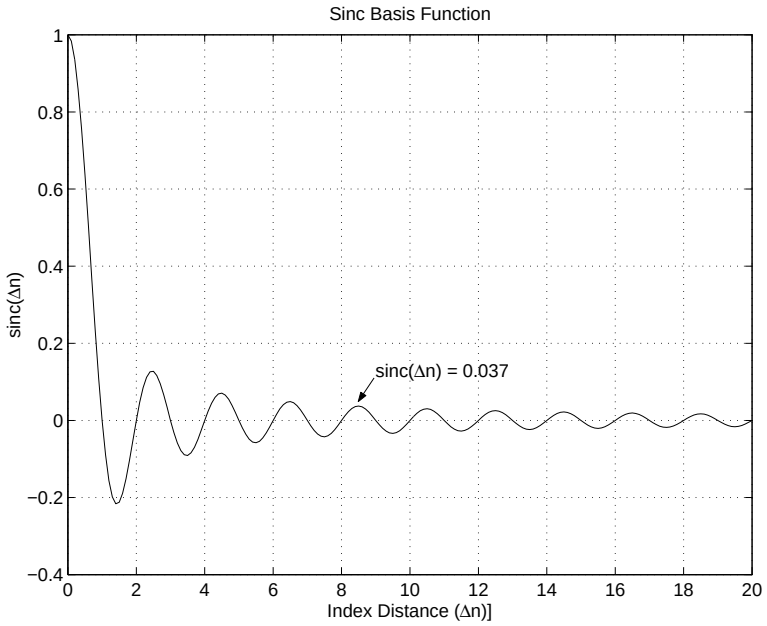


Figure 2: Sinc basis function

where $s_{\hat{u}}$ is an interpolant of the form

$$s_{\hat{u}}(\kappa) = \sum_{j=1}^{N} \alpha_j e^{-t\,\|\kappa_j - \kappa\|^2},$$

where α is a vector in $\mathbb{C}^N$ and $t > 0$ is the scaling factor (or squeezing coefficient) of the Gaussian. If we define $A = \{e^{-t\|\kappa_j - \kappa_l\|^2}\}_{1\le j,l\le N}$, $\alpha = (\alpha_j)_{1\le j\le N}$, and $\hat{u} = (\hat{u}(\kappa_j))_{1\le j\le N}$, then we can rewrite (15) in the form

$$A\alpha = \hat{u}. \tag{16}$$

Note that the Gaussian is a positive definite function, which implies that A is a positive definite symmetric matrix, and hence the interpolation problem (15) has a unique solution. The interpolation is of order e^{-t/h^2}, where

$$h(\kappa) = \max_{\|\tilde{\kappa}-\kappa\|\le\rho} (\min_{1\le j\le N} \|\tilde{\kappa} - \kappa_j\|_2)$$

is the mesh-norm, which only depends on K and ρ. For reference, see e.g. Schaback [3].

In our problem, the number of interpolation points is $N = 262144$. Let $\hat{u}_0$ be the value that we want to interpolate. Instead of doing a global interpolation, which would have lead to a matrix A of dimension 262144x262144, we use a local interpolation scheme. This scheme consists only of 4 grid points in each direction (in two dimensions), which gives a total of $N_{loc} = 64$ local interpolation points. The advantage of this local interpolation is that solving a local system of the form (15) is very fast in comparison with the full system, but the disadvantage is that our interpolant is only reflecting a very small neighborhood of the point $\hat{u}_0$, and hence it might have a larger error than the global scheme. Since the total number of computations is very large

$\mathcal{O}(984 * 262144)$, i.e. $\mathcal{O}(984 * 512^2)$, and our time rather limited, we preferred to use the local interpolation approach.

We obtain the values of the weights α by solving the system (16),which we need to do for every local interpolation. As a result, this method is more expensive in time than the interpolation methods that calculate the weights α_j, $1 \leq j \leq N$, directly, as for example the Slow Fourier Transform. In order to avoid the problem of interpolating at the boundaries of the grid, we used zero-padding around the boundary.

This RBF-scheme can interpolate the value $\hat{u}_0$, for any arbitrary point $\hat{u}_0 \in D$, given any set of distinct points $K = \{\kappa_j\}_{j=1}^N \in D$. Hence, this interpolation scheme can be used to interpolate data along polar lines, given data on a cartesian grid, and vice versa.

In order to test the RBF-interpolation scheme, we interpolate data along the polar lines for the function (6), as described in the first part of this section. The results are given below.

4.5 Interpolation Based on Neighboring Pixels

Bilinear, bicubic and bispline ([4]) are popular local interpolation approaches. Bilinear is the simplest approach but with the worst quality. Bicubic and bispline interpolation can obtain better quality with continuous derivatives on the grid but need more computational time. We did 3 simple but very efficient interpolation methods using a MATLAB routine.

5 Computational Issues

As may be expected, this is a very computationally intensive method. On the positive side, since the data is the two-dimensional Fourier transform of our initial pictures, it satisfies Hermitian symmetry. Thus we can store only half of the actual data for our function $\hat{u}$ at each point, since the other half is merely the complex conjugate of our stored data. This effectively cuts our data storage and workload in half.

This half is still quite large. Given some coordinate (θ, z), our views (using the Hermitian symmetry) contain $Nlines * 256$ complex data points (Nlines corresponds to the number of radial lines in the κ_1, κ_2 plane). Each helix then contains 984 views, and we wish to compute approximately 20 helices in each direction. As far as memory, we must hold one full helix and one temporary (half) helix, amounting to over 2 Gb of RAM at any given time.

Computationally, this method is quite intensive. For testing purposes, we propagated one helix of data through to its neighboring helix in MATLAB, and when run on a dual-processor 500 Mhz Alpha PC with 3.5 Gb RAM, it took approximately 10 hours. This does not include the initial processing to take the views u on a cartesian grid to the function $\hat{u}$ on our radial grid.

Considering the amount of work that can be done autonomously, this seems like an ideal task for parallel processing, though the high memory cost is still a troublesome matter.

6 Future Work

Given that this workshop lasted only ten days, we found that there were other things that we would have liked to try to make our method better. The first of these was to use one of our interpolation schemes for our derivative estimation. This would have eliminated the need to compute our radial derivatives completely. This way we could analytically find the derivatives

in the directions κ_1 and κ_2 of our basis functions, and merely multiply them by the coefficients used in interpolation.

Additionally, we thought that there might be other interpolation methods that could not only be more accurate, but also faster to calculate. These could be used to both interpolate from our cartesian grid to our radial grid, and vice versa.

Higher order ODE methods are also quite feasible, but for each temporary helix we would need a full set of storage and work. Thus higher-order methods are still possible, though need to be implemented carefully.

Acknowledgements

The authors would like to acknowledge the support from the National Security Agency, the Institute of Mathematics and its Applications and General Electric.

Appendix

Symbol	**Meaning**
R	Focal point to isocenter distance
D	Isocenter to detector distance
Nviews	Number of projections in $2pi$ rotation
$\Delta\theta_h$	Angular distance
Pitch	Relates the transverse length to a revolution
α_1	Detector coordinate
α_2	Detector coordinate
$\Delta\alpha$	Distance between detector pixels
u	Line integral of attenuation from source to detector
$\hat{u}$	Fourier transform of u
ξ	Detector coordinate in 3-space
η	Source position in 3-space
κ_1	Frequency
κ_2	Frequency
$\Delta\kappa$	Step size along radial line
$Nlines$	Number of radial lines required to sample Fourier domain

Table 2: Table of parameters

References

[1] F. John, *The Ultrahyperbolic Differential Equation with Four Independent Variables*, Duke Math Journal 4, 1938, pgs. 300-322.

[2] RD 27317 filed with the US Patent Office Oct 1999.

[3] R. Schaback, *Error Estimates and Condition Numbers for Radial Basis Function Interpolation*, Advances in Computational Mathematics 3, 1995, pgs. 251-264.

[4] W. Press et. al *Numerical Recipes in C*, Cambridge, 1992.