Homework 3 Solutions

Some notations have been altered from the assignment for ease of presentation. When considering implementations of discrete Fourier transforms, there are painfully many conventions one can take. One's specific choice is generally immaterial provided consistency is observed. These solutions follow one set of conventions according to the preference of the author. Should confusion arise, please consult the CA.

All code displayed is written in MATLAB.

1. (a) $\mathcal{P}$ stands for polar!

   (b) The projection-slice theorem

$$\int \mathcal{R}f(t,\theta)e^{-irt}\,dt = \hat{f}(r\boldsymbol{\theta}), \quad \boldsymbol{\theta} = (\cos\theta, r\sin\theta)$$

restricts us to computing the Fourier transform along the lines of $\mathbb{R}^2$ making angles

$$\theta_\ell = \frac{\ell - 1}{N_\theta - 1}, \quad \ell = 1, 2, \ldots, N_\theta$$

with the positive x-axis. And for each such angle, there are N_t samples of $\mathcal{R}f(\,\cdot\,,\theta)$, namely at

$$t_k = -\frac{1}{2} + \frac{k-1}{N_t - 1}, \quad k = 1, 2, \ldots, N_t.$$

The sampling rate is thus $N_t - 1$ samples per unit time, suggesting the maximum frequency we can detect is $\pi(N_t - 1)$. By modulating the signal and zero-padding the DFT, we can in theory estimate the value of $\hat{f}(r\boldsymbol{\theta}_\ell)$ for any $|r| \le \pi(N_t - 1)$. For our purposes, though, it will be sufficient to use a standard FFT to determine the N_t-point spectrum of the discrete signal `Y0[l, :]`.
Using the fact that $\mathcal{R}(t,\theta)$ vanishes for $|t| > \frac{1}{2}$ and applying a simple left-hand quadrature rule, we have

$$\begin{aligned}
\int_{\mathbb{R}} \mathcal{R}f(t,\theta_\ell)e^{-irt}\,dt &= \int_{-\frac{1}{2}}^{\frac{1}{2}+\frac{1}{N_t-1}} \mathcal{R}f(t,\theta_\ell)e^{-irt}\,dt \\
&\approx \frac{1}{N_t - 1}\sum_{k=1}^{N_t} \mathcal{R}f(t_k,\theta_\ell)e^{-ir\left(-\frac{1}{2}+\frac{k-1}{N_t-1}\right)} \\
&= \frac{e^{\frac{ir}{2}}}{N_t - 1}\sum_{k=1}^{N_t} \mathcal{R}f(t_k,\theta_l)e^{-ir\frac{k-1}{N_t-1}}. \qquad (1)
\end{aligned}$$

MATLAB's implementation of the FFT computes, for a discrete signal x of (even) length M, the quantities

$$y[j] = \sum_{k=1}^{M} x[k]e^{-i2\pi(j-1)(k-1)/M}, \quad j = 1, 2, \ldots, M. \qquad (2)$$

In particular, it assumes the signal is sampled at times $0, \frac{1}{M}, \ldots, \frac{M-1}{M}$ and generates Fourier samples at frequencies $0, 2\pi, \ldots, 2\pi(M-1)$. If one applies an `fftshift` *before* computing the DFT, the time samples are then assumed to occur at $-\frac{1}{2}, -\frac{1}{2}+\frac{1}{M}, \ldots, \frac{1}{2}-\frac{1}{M}$. If one applies an `fftshift` *after* computing the DFT, then the frequencies sampled are

$$-2\pi\frac{M}{2}, -2\pi\left(\frac{M}{2}-1\right), \ldots, 2\pi\left(\frac{M}{2}-1\right),$$

To match (1) to the form of (2), we choose

$$r_j = 2\pi\frac{N_t - 1}{N_t}(j-1), \quad j = 1, 2, \ldots, N_t.$$

Then

$$\int \mathcal{R}f(t,\theta)e^{-ir_j t}\,dt \approx \frac{e^{\frac{ir_j}{2}}}{N_t-1}\underbrace{\sum_{k=1}^{N_t}\mathcal{R}f(t_k,\theta_l)e^{-i2\pi(j-1)(k-1)/N_t}}_{\text{computed with an FFT}}.$$

We do not require a shift before applying an FFT, but we make one after in order to obtain Fourier samples at the centered frequencies

$$r_j = 2\pi\left(-\frac{N_t-1}{2}+(j-1)\frac{N_t-1}{N_t}\right), \quad j=1,2,\ldots,N_t.$$

Note that $\hat{f}(\mathbf{0})$ is computed for each angle, as $r_{\frac{N_t}{2}+1}=0$, and so we can remove repeated instances. In addition, for $\theta_{N_\theta}=\pi$, all points are accounted for with $\theta_0=0$ except for the point at distance $r_1=-\pi(N_t-1)$ (so the frequency $(\pi(N_t-1),0)$ in the positive x-axis).

```
%% Parameters

N = 128;
N_t = 256;
N_theta = 64;
D = 8;
L = 4;

% maximum number of iterations in conjugate gradient
max_iter = 10;

% column vector of angles
theta = linspace(0, 1, N_theta)' * pi;

% column vector of radii
r = 2 * pi * (N_t - 1)/N_t * (-N_t/2 : N_t/2-1)';

% list of frequencies on polar grid
P = [0 0;
     r(1) * cos(theta(N_theta)) r(1) * sin(theta(N_theta));
     kron(r(1 : N_t/2), [cos(theta(1 : N_theta-1)) sin(theta(1 : N_theta-1))]);
     kron(r(N_t/2+2 : N_t), [cos(theta(1 : N_theta-1)) sin(theta(1 : N_theta-1))])];

%% From Radon to Fourier via projection slice

Y1 = fftshift(fft(Y0, N_t, 2), 2) .* exp(1i * repmat(r.', [N_theta 1]) / 2) / (N_t-1);

% value at origin
Y_origin = Y1(1, N_t/2 + 1);

% value at the one radius at angle pi not reached from angle 0
Y_pi = Y1(N_theta, 1);

% all other values
Y_else = Y1(1 : N_theta-1, [(1 : N_t/2) (N_t/2+2 : N_t)]);

% put values together, matching the order of P
Y1 = [Y_origin; Y_pi; Y_else(:)];
```

(c) See section 4 of the notes from Lecture 12 for the one-dimensional version of the NUFFT algorithms used in this problem. First, the calculation

$$\boldsymbol{X} \mapsto \sum_{\boldsymbol{x}\in\mathcal{C}} X_{\boldsymbol{x}} e^{-i\langle \boldsymbol{\omega},\,\boldsymbol{x}\rangle}, \quad \boldsymbol{\omega}\in\mathcal{P} \tag{3}$$

requires a type II NUFTT. Here $\mathcal{C}$ is the uniform, Cartesian grid in the spatial domain, with points of the form

$$\boldsymbol{x}_{n_1,n_2} = \left(\frac{n_1+\frac{1}{2}}{N}, \frac{n_2+\frac{1}{2}}{N}\right), \quad n_1, n_2 = -\frac{N}{2}, -\frac{N}{2}+1, \ldots, \frac{N}{2}-1.$$

It will be easier to consider the finer grid $\widetilde{\mathcal{C}}$ with points

$$\widetilde{\boldsymbol{x}}_{n_1,n_2} = \left(\frac{n_1}{2N}, \frac{n_2}{2N}\right), \quad n_1, n_2 = -N, -N+1, \ldots, N-1,$$

and then subsample at the end. This choice is motivated by the problem's hypothesis that N_t is twice of N, meaning we have additional frequency data to use. We will consider a dense frequency grid $\mathcal{D}$, consisting of the lattice

$$\boldsymbol{\tau}_{j_1,j_2} = (\tau_{j_1}, \tau_{j_2}), \qquad \tau_j = 2\pi\left(-N + \frac{j-1}{D}\right), \quad j = 1, 2, \ldots, 2DN.$$

Notice that $2\pi N$ is the maximum frequency we can compute from data on $\widetilde{\mathcal{C}}$, and is one we need since $r_1 = -\pi(N_t - 1) \approx -2\pi N$. We have

$$\hat{f}(\boldsymbol{\omega}) \approx \frac{1}{(2N)^2} \sum_{\boldsymbol{x}\in\widetilde{\mathcal{C}}} f(\boldsymbol{x}) e^{-i\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle} = Y(\boldsymbol{\omega}).$$

Taking partial derivatives of the trigonometric polynomial Y, we obtain

$$\frac{\partial^{\ell_1+\ell_2}}{\partial_1^{\ell_1}\partial_2^{\ell_2}} Y(\boldsymbol{\omega}) = \frac{1}{(2N)^2} \sum_{\boldsymbol{x}\in\widetilde{\mathcal{C}}} (-ix)^{\ell_1}(-iy)^{\ell_2} f(\boldsymbol{x}) e^{-i\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle},$$

where $\boldsymbol{x} = (x, y)$. When $\boldsymbol{\omega} = \boldsymbol{\tau}_{j_1,j_2}$, the exponent can be written

$$\begin{aligned} -i\langle \boldsymbol{\tau}_{j_1,j_2}, \boldsymbol{x}\rangle &= -i2\pi\left[x\left(-N + \frac{j_1-1}{D}\right) + y\left(-N + \frac{j_2-1}{D}\right)\right] \\ &= i2\pi(x+y)N - i2\pi\left[x\frac{j_1-1}{D} + y\frac{j_2-1}{D}\right]. \end{aligned}$$

Now

$$\frac{\partial^{\ell_1+\ell_2} Y}{\partial_1^{\ell_1}\partial_2^{\ell_2}}(\boldsymbol{\tau}_{j_1,j_2}) = \frac{1}{(2N)^2} \sum_{\boldsymbol{x}\in\widetilde{\mathcal{C}}} (-ix)^{\ell_1}(-iy)^{\ell_2} e^{i2\pi(x+y)N} f(\boldsymbol{x}) e^{-i2\pi(x(j_1-1)+y(j_2-1))/D}$$

If x and y are taken from $0, \frac{1}{2N}, \ldots, \frac{2N-1}{2N}$, these values are exactly the entries of the $2DN$-point DFT ($2N$-point DFT with zero-padding by a factor of D) of

$$\frac{1}{(2N)^2}(-ix)^{\ell_1}(-iy)^{\ell_2} e^{i2\pi(x+y)N} f(\boldsymbol{x}),$$

where j_1 and j_2 are allowed to vary between $1, 2, \ldots, 2DN$, as desired. To enforce that x and y are instead taken from $-\frac{1}{2}, -\frac{1}{2}+\frac{1}{2N}, \ldots, \frac{1}{2}-\frac{1}{2N}$, we must apply an `fftshift` prior to computing the FFT.
If D and L are taken sufficiently large, then for any $\boldsymbol{\omega} = (\omega_1, \omega_2) \in \mathcal{P}$, we can find $\boldsymbol{\tau} = (\tau_1, \tau_2) \in \mathcal{D}$ close to $\boldsymbol{\omega}$. In this case, we use the approximation

$$Y(\boldsymbol{\omega}) \approx \sum_{\ell_1,\ell_2=0}^{L-1} \frac{(\omega_1-\tau_1)^{\ell_1}(\omega_2-\tau_2)^{\ell_2}}{\ell_1!\,\ell_2!} \frac{\partial^{\ell_1+\ell_2} Y}{\partial_1^{\ell_1}\partial_2^{\ell_2}}(\boldsymbol{\tau}).$$

```
%% NUFFT setup

% determine the j's corresponding to nearest neighbors
J = round(D * (P / (2*pi) + N) + 1);

% ensure 2*D*N + 1 is not one of those indices
J = min(J, 2*D*N);
```

```
% column of error vectors
distances         = P - 2*pi * (-N + (J - 1) / D);

% compute indices when Fourier matrix on dense grid is reshaped to a column vector
J = 2*D*N * (J(:, 1) - 1) + (2*D*N - J(:, 2) + 1);

% each column (row) of x (y) is constant
vec = -1/2 + (0 : 2*N-1) / (2*N);
[x, y] = meshgrid(vec, vec);

% flip so that y coordinates ascend from bottom to top
y = flipud(y);

% padded versions of x and y, used in NUFFT_II
xx = padarray(x, [N*(D-1) N*(D-1)]);
yy = padarray(y, [N*(D-1) N*(D-1)]);

%% Type II

function Y = NUFFT_II(X)

 XX = reshape(X, [2*N 2*N]);

 % pad X so that nonzero entries are in the center
 XX = padarray(XX, [N*(D-1) N*(D-1)]);

 % initialize the column vector Y
 Y = zeros(size(P, 1), 1);

 for l1 = 0 : L-1
  for l2 = 0 : L-1
   T = fft2(fftshift((-1i*xx).^l1 .* (-1i*yy).^l2 .* exp(1i * 2*pi * (xx+yy) * N) .* XX));

   % flip upside down because y-coordinate is ascending from bottom to top
   T = flipud(T);

   % turn matrix into a column vector
   T = reshape(T, [(2*D*N)^2 1]);

   Y = Y + distances(:,1).^l1 .* distances(:,2).^l2 /factorial(l1)/factorial(l2) .* T(J);
  end
 end

 Y = Y / (2*N)^2;

end
```

On the other hand,

$$\boldsymbol{Y} \mapsto \sum_{\boldsymbol{\omega} \in \mathcal{P}} Y_{\boldsymbol{\omega}} e^{i\langle \boldsymbol{\omega}, \boldsymbol{x} \rangle}, \quad \boldsymbol{x} \in \mathcal{C} \tag{4}$$

is a type I calculation. We adopt the process that is adjoint to the one just described. For each $\boldsymbol{\tau} \in \mathcal{D}$, let

$$\mathcal{N}(\boldsymbol{\tau}) = \{\boldsymbol{\omega} \in \mathcal{P} \ : \ \boldsymbol{\tau} \text{ is the closest point in } \mathcal{D} \text{ to } \boldsymbol{\omega}\}.$$

Write $\boldsymbol{x} = (x, y)$. When $\boldsymbol{\omega} = (\omega_1, \omega_2)$ is close to $\boldsymbol{\tau} = (\tau_1, \tau_2)$, we can use the approximation

$$\begin{aligned} e^{i\langle \boldsymbol{\omega},\, \boldsymbol{x}\rangle} &= e^{i\langle \boldsymbol{\tau},\, \boldsymbol{x}\rangle} e^{i\langle \boldsymbol{\omega}-\boldsymbol{\tau},\, \boldsymbol{x}\rangle} \\ &\approx e^{i\langle \boldsymbol{\tau},\, \boldsymbol{x}\rangle} \sum_{\ell_1,\ell_2=0}^{L-1} \frac{(ix(\omega_1-\tau_1))^{\ell_1}(iy(\omega_2-\tau_2))^{\ell_2}}{\ell_1!\;\ell_2!} \end{aligned}$$

Hence

$$\begin{aligned} \sum_{\boldsymbol{\omega}\in\mathcal{P}} \hat{f}(\boldsymbol{\omega}) e^{i\langle \boldsymbol{\omega},\, \boldsymbol{x}\rangle} &= \sum_{\boldsymbol{\tau}\in\mathcal{D}} \sum_{\boldsymbol{\omega}\in\mathcal{N}(\boldsymbol{\tau})} \hat{f}(\boldsymbol{\omega}) e^{i\langle \boldsymbol{\omega},\, \boldsymbol{x}\rangle} \\ &\approx \sum_{\ell_1,\ell_2=0}^{L-1} \frac{(ix)^{\ell_1}(iy)^{\ell_2}}{\ell_1!\;\ell_2!} \sum_{\tau\in\mathcal{D}} \left(\sum_{\omega\in\mathcal{N}(\boldsymbol{\tau})} (\omega_1-\tau_1)^{\ell_1}(\omega_2-\tau_2)^{\ell_2} \hat{f}(\boldsymbol{\omega}) \right) e^{i\langle \boldsymbol{\tau},\, \boldsymbol{x}\rangle}. \end{aligned}$$

The sum over $\mathcal{D}$ is the discrete inverse Fourier transform of the function X_{ℓ_1,ℓ_2} given by

$$X_{\ell_1,\ell_2}(\boldsymbol{\tau}) = \sum_{\omega\in\mathcal{N}(\boldsymbol{\tau})} (\omega_1-\tau_1)^{\ell_1}(\omega_2-\tau_2)^{\ell_2} \hat{f}(\boldsymbol{\omega}).$$

We also observe that for $\boldsymbol{\tau} = \boldsymbol{\tau}_{j_1,j_2} = (\tau_{j_1}, \tau_{j_2})$,

$$\begin{aligned} i\langle \boldsymbol{\tau},\, \boldsymbol{x}\rangle &= i2\pi \left[x\left(-N + \frac{j_1-1}{D}\right) + y\left(-N + \frac{j_2-1}{D}\right)\right] \\ &= -i2\pi(x+y)N + i2\pi\left[x\frac{j_1-1}{D} + y\frac{j_2-1}{D}\right]. \end{aligned}$$

Therefore, in order to evaluate the discrete IFT of X_{ℓ_1,ℓ_2} onto $\widetilde{\mathcal{C}}$, we apply a $2DN$-point IFFT, followed by an `ifftshift`, and then subsample. In total,

$$\sum_{\boldsymbol{\omega}\in\mathcal{P}} \hat{f}(\boldsymbol{\omega}) e^{i\langle \boldsymbol{\omega},\, \boldsymbol{x}\rangle} \approx e^{-i2\pi(x+y)N} \sum_{\ell_1,\ell_2=0}^{L-1} \frac{(ix)^{\ell_1}(iy)^{\ell_2}}{\ell_1!\;\ell_2!} \sum_{\boldsymbol{\tau}\in\mathcal{D}} X_{\ell_1,\ell_2}(\boldsymbol{\tau}) e^{i2\pi(x(j_1-1)+y(j_2-1))/D}$$

.

```
%% Type I

function X = NUFFT_I(Y)

    % initialize the matrix X
    X = zeros(2*N, 2*N);

    for l1 = 0 : L-1
        for l2 = 0 : L-1
            % contributions to X_{l1,l2} from nearest neighbors
            S = sparse(J, 1, distances(:,1).^l1 .* distances(:,2).^l2 .* Y, (2*D*N)^2, 1);
            S = reshape(S, [2*D*N 2*D*N]);

            % flip upside down because y-coordinate is ascending from bottom to top
            S = ifftshift(ifft2(full(flipud(S))));

             % subsample back in space (middle square)
            S = S(end/2 - N + 1 : end/2 + N, end/2 - N + 1 : end/2 + N);

            X = X + (1i * x).^l1 .* (1i * y).^l2 /factorial(l1)/factorial(l2) .* S;
        end
    end

    X = X .* exp(-1i * 2*pi * (x+y) * N);
```

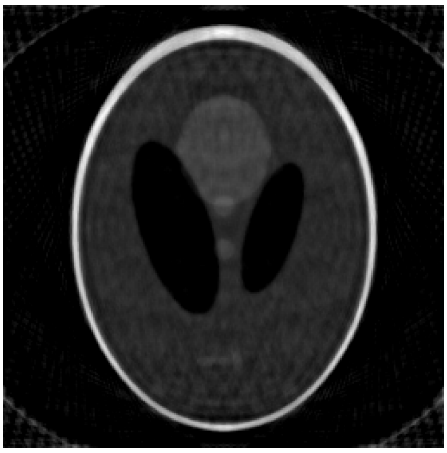
```
    X = reshape(X, [(2*N)^2 1]);

end
```

(d) Our NUFFTs allow us to approximately evaluate (3) and (4) for any signals $\boldsymbol{X}$ and $\boldsymbol{Y}$. Call the unknown signal f, and denote the forward transform (3) by A, and the adjoint transform (4) by A^*. Since these are non-uniform transforms, we should not expect that $A^*Af = f$. Rather, we may say $g = A^*\hat{f}$, and then try to solve $A^*Af = g$ for f. Since A^*A is symmetric and positive semidefinite, we can apply conjugate gradient iteration to do so. We may think of g as our naive reconstruction, stored as `X1`, and $\hat{f}$ (on the polar grid $\mathcal{P}$) is stored as `Y1`.

```
%% Perform one iteration of type I

X1 = NUFFT_I(Y1);
X1 = reshape(X1, [(2*N)^2 1]);

%% Run pcg

f = pcg(@(x) NUFFT_I(NUFFT_II(x)), X1, [], max_iter);
f = reshape(f, [2*N 2*N]);
X1 = reshape(X1, [2*N 2*N]);

%% Display results of problem 1

figure
imshow(X1 / max(max(X1)));

figure
imshow(f / max(max(f)));
```

The results shown in Fig. 1 are reconstructions on the 256×256 grid $\widetilde{\mathcal{C}}$, and they can obviously be subsampled to produce results on the 128×128 grid $\mathcal{C}$. Notice that the conjugate gradient method sharpens edges, even if introducing some artifacts. This allows us to better identify fine features. It is reasonable that the direct "inversion" from polar data would be blurred and smoothed, since a polar arrangement of samples in the Fourier domain eliminates high frequency data contained in corners of a cartesian arrangement.

(a) Naive reconstruction, `X1`



(b) Solution via conjugate gradient, `f`

Figure 1: Fourier reconstructions with no noise. Here $D = 8$ and $L = 4$, and 10 iterations were used in conjugate gradient.

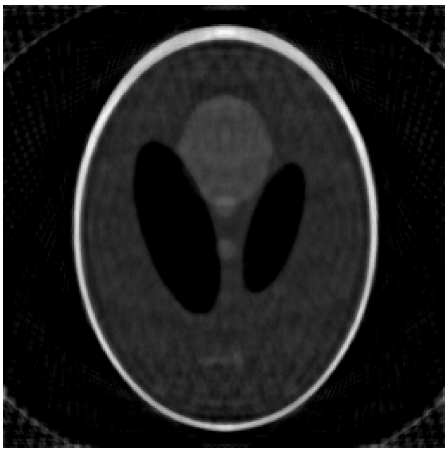
(f) We apply the following code, and then run everything already discussed.

```
%% Introduce Poisson noise
```

```
lambda = 10^3 / max(max(Y0));
Y0 = poissrnd(lambda * Y0);
```

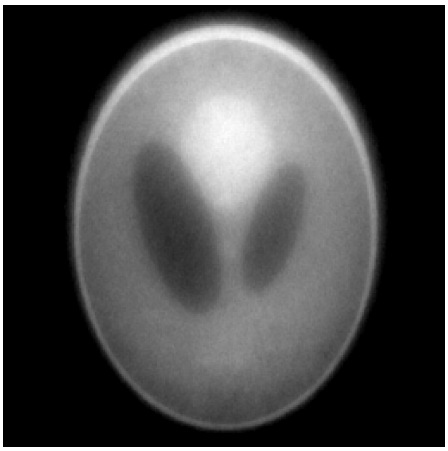
The results are shown in Fig. 2 and Fig. 3. Obviously the Poisson noise introduces artifacts, tending to obscure fine details of the image. The effects are not very dramatic when $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^6$, but seriously pollute the final image when $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^3$. This makes sense because of the mean of a Poisson(λ) distribution is λ, while the standard deviation is $\sqrt{\lambda}$. Hence the signal-to-noise ratio (defined as the inverse of the coefficient of variation) grows like $\sqrt{\lambda}$. Observe that the noise is more apparent in the solution obtained via conjugate gradient, due to the fact that edges and small perturbations are preserved in this reconstruction. On the other hand, the direct inversion largely washes out the noise, as it is a blurred and smoothed version of the true signal.

(a) Naive reconstruction, `X1`

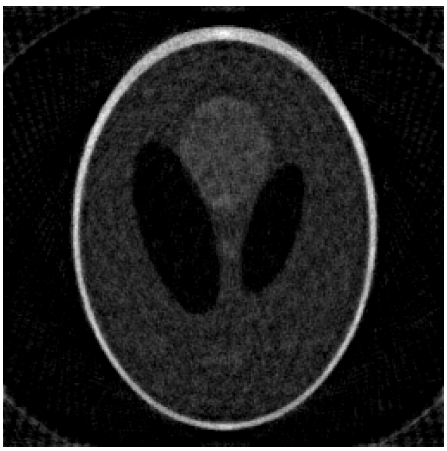


(b) Solution via conjugate gradient, `f`

Figure 2: Fourier reconstructions with $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^6$. Here $D = 8$ and $L = 4$, and 10 iterations were used in conjugate gradient.



(a) Naive reconstruction, `X1`



(b) Solution via conjugate gradient, `f`

Figure 3: Fourier reconstructions with $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^3$. Here $D = 8$ and $L = 4$, and 10 iterations were used in conjugate gradient.

2. (a) The backprojection formula (when properly normalized) computes, for each $\boldsymbol{x} \in \mathbb{R}^2$, the average value of the integrals over lines passing through $\boldsymbol{x}$. Indeed, $\boldsymbol{x}$ is on the line associated with the angle θ and distance t from the origin, if and only if $\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle = t$. If we want to compute the backprojection

$$\mathcal{R}^* g(\boldsymbol{x}) = \int_0^\pi g(\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle, \theta)\, d\theta,$$

we obviously must interpolate with values of θ for which we actually have data. The resulting quadrature takes

the general form

$$\mathcal{R}^* g(\boldsymbol{x}) \approx \sum_{\ell=1}^{N_\theta} g(\langle \boldsymbol{x}, \boldsymbol{\theta}_\ell \rangle, \theta_\ell)\ \alpha_\ell, \tag{5}$$

where the weights α_ℓ satisfy

$$\sum_{\ell=1}^{N_\theta} \alpha_\ell = \pi.$$

A simple choice is trapezoidal weights: $\alpha_\ell = \frac{\pi}{N_\theta - 1}$ for all ℓ, but divided by 2 at the endpoints.
Next realize that because we have fixed points of interest $\boldsymbol{x}_{n_1,n_2}$, we should not expect to know the value of g *exactly* at $(\langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle, \theta_\ell)$. That is, it is unlikely that $\langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle = t_k$ for some $k = 1, 2, \ldots, N_t$. Hence, for each angle θ_ℓ, we need to approximate $g(\langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle, \theta_\ell)$ by interpolating between nearby points. Since our samples in t are reasonably dense, a good strategy is to interpolate in t. Indeed, if t_k is close to $\langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle, \theta_\ell$, then $\boldsymbol{x}_{n_1,n_2}$ is close to the line associated with θ_l and t_k.
Written most generally, the interpolation is expressed

$$g(\langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle, \theta_\ell) \approx \sum_{k=1}^{N_t} g(t_k, \theta_\ell)\ \beta_k^{(\ell)}, \tag{6}$$

where now the weights $\beta_k^{(\ell)}$ satisfy

$$\sum_{\ell=1}^{N_\theta} \beta_k^{(\ell)} = 1.$$

In this case, we will employ a very simple interpolation kernel: If

$$t_m \le t = \langle \boldsymbol{x}_{n_1,n_2}, \boldsymbol{\theta}_\ell \rangle \le t_{m+1}, \tag{7}$$

then we will set

$$\begin{aligned} \beta_k^{(\ell)} &= \begin{cases} (t - t_m)/(t_{m+1} - t_m) & k = m \\ (t_{m+1} - t)/(t_{m+1} - t_m) & k = m+1 \\ 0 & \text{else.} \end{cases} \\ &= \begin{cases} (t - t_m)(N_t - 1) & k = m \\ (t_{m+1} - t)(N_t - 1) & k = m+1 \\ 0 & \text{else.} \end{cases} \end{aligned}$$

This is just linear interpolation between the two nearest neighbors. Notice that there will always exist a $1 \le p \le N_\theta$ so that (7) holds, since $t_1 = 0$ and $t_{N_\theta} = \pi$. Putting (5) and (6) together, we have

$$\mathcal{R}^* g(\boldsymbol{x}) \approx \sum_{\ell=1}^{N_\theta} \sum_{k=1}^{N_t} g(t_k, \theta_\ell)\ \beta_k^{(\ell)}\ \alpha_\ell. \tag{8}$$

(b) Let us implement the interpolation kernel proposed in (a). That is, α_ℓ is a uniform weight, and $\beta_k^{(\ell)}$ is a nearest neighbor linear weight. We can rewrite (8) as

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{N_t} \end{pmatrix} \underbrace{\begin{pmatrix} & \vdots & \\ \cdots & \beta_k^{(\ell)} g(t_k, \theta_\ell) & \cdots \\ & \vdots & \end{pmatrix}}_{\boldsymbol{G}} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

where the entry in the ℓth row and kth column of $\boldsymbol{G}$ is $\beta_k^{(\ell)} g(t_k, \theta_\ell)$. The only matter left unresolved is to how to identify the integer m so that (7) holds. We simply recall the expression for t_m and solve for m:

$$\begin{aligned} -\frac{1}{2} + \frac{m-1}{N_t - 1} \le t \le -\frac{1}{2} + \frac{m}{N_t - 1} \\ \Rightarrow m \le (N_t - 1)\left(t + \frac{1}{2}\right) + 1 \le m + 1. \end{aligned}$$

```
%% Parameters

N = 256;
N_t = 256;
N_theta = 64;

% column vector of angles
theta = linspace(0, 1, N_theta)' * pi;
COS = cos(theta);
SIN = sin(theta);

% column vector of radii
r = 2 * pi * (N_t - 1)/N_t * (-N_t/2 : N_t/2-1)';

% column vector of grid points along one dimension
vec = ((-N/2 : N/2-1) + 1/2)' / N;

% row vector [1 1 2 2 ... N_theta N_theta]
vec_angle = [1 : N_theta; 1 : N_theta]; vec_angle = vec_angle(:);

%% Backprojection

f = backproj(Y0);

%% Function for backprojection

function f = backproj(g)

        f = zeros(N, N);

        Alpha = pi / (N_theta-1) * ones(1, N_theta);
        Alpha([1 N_theta]) = Alpha([1 N_theta])/2;

        for ix = 1 : N
                for iy = 1 : N
                        dot_products = vec(ix) * COS + vec(iy) * SIN;

                        % determine left neighbor (i.e. determine m)
                        K = floor((N_t-1) * (dot_products + 1/2) + 1);

                        % weights for left neighbor
                        distances = (dot_products + 1/2 - (K-1) / (N_t-1)) * (N_t-1);

                        % include right neighbor
                        K = [K.'; K.'+1]; K = K(:); K = min(K, N_t); K = max(K, 1);

                        % include weights for right neighbor
                        distances = [distances.'; 1 - distances.']; distances = distances(:);

                        % matrix of weights for linear interpolation
                        Beta = sparse(vec_angle, K, distances, N_theta, N_t);

                        % row (column) number determines the y (x) coordinate;
                        f(iy, ix) = Alpha * (Beta .* g) * ones(N_t, 1);
                end
        end

end
```

(c) Convolution in the spatial domain is equivalent to multiplication in the Fourier domain (and vice versa). So our strategy will be to take an FFT of g, apply the filter in the Fourier domain by multiplying by $h(r) = |r|$, and then compute the IFFT. From there we can just apply standard backprojection (modulo a factor of $(2\pi)^2$). We have

$$\widehat{(g(\,\cdot\,,\theta_\ell) * h)}(r) = \widehat{g(\,\cdot\,,\theta_\ell)}(r)\hat{h}(r),$$

and we saw in Problem 1 that we can compute, from Radon data, Fourier samples at choice values of r:

$$\hat{g}_\ell(r) := \widehat{(g(\,\cdot\,,\theta_\ell)}(r_j) \approx \frac{e^{\frac{ir_j}{2}}}{N_t - 1} \sum_{k=1}^{N_t} g(t_k, \theta_l) e^{-i2\pi j(k-1)/N_t}, \tag{9}$$
$$r_j = 2\pi \frac{N_t - 1}{N_t} j, \quad j = -\frac{N_t}{2}, -\frac{N_t}{2} + 1, \ldots, \frac{N_t}{2} - 1.$$

After we multiply by the filter, an inverse Fourier transform gives

$$\begin{aligned}
(g(\,\cdot\,,\theta_\ell) * h)(t_m) &\approx 2\pi \frac{N_t - 1}{N_t} \sum_{j=-\frac{N_t}{2}}^{\frac{N_t}{2}-1} |r_j| \hat{g}_\ell(r_j) e^{ir_j t_m} \\
&= 2\pi \frac{N_t - 1}{N_t} \sum_{j=-\frac{N_t}{2}}^{\frac{N_t}{2}-1} |r_j| \hat{g}_\ell(r_j) e^{ir_j(-\frac{1}{2} + \frac{m-1}{N_t - 1})} \\
&= 2\pi \frac{N_t - 1}{N_t} \sum_{j=-\frac{N_t}{2}}^{\frac{N_t}{2}-1} |r_j| \hat{g}_\ell(r_j) e^{-ir_j/2} e^{i2\pi j(m-1)/N_t}
\end{aligned} \tag{10}$$

Using (9) in (10), we arrive at

$$(g(\,\cdot\,,\theta_\ell) * h)(t_m) = \frac{1}{N_t} \sum_{j=-\frac{N_t}{2}}^{\frac{N_t}{2}-1} |r_j| \left(\sum_{k=1}^{N_t} g(t_k, \theta_l) e^{-i2\pi(j-1)(k-1)/N_t} \right) e^{i2\pi j(m-1)/N_t}$$

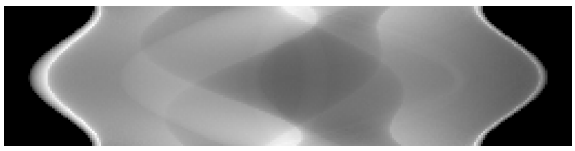
This is a standard FFT, followed by an `fftshift` to center the radii r_j, followed by multiplication by $|r_j|$, followed by an `ifftshift` to ensure the inverse DFT receives an input on $[-\frac{1}{2}, \frac{1}{2}]$, and then finally an IFFT multiplied by 2π. The two shifts compose to the identity, however, and so we can equivalently shift only the $|r_j|$. It is now easy to apply the filtered backprojection

$$\frac{1}{(2\pi)^2} \int_0^\pi (g(\,\cdot\,,\theta) * h)(\langle \boldsymbol{x}, \boldsymbol{n}_\theta \rangle)\, d\theta.$$

```
%% Function for filter

function gg = my_filter(g)

        gg = fft(g, N_t, 2) .* repmat(abs(fftshift(r).'), [N_theta 1]);
        gg = ifft(gg, N_t, 2) / (2*pi);

end
```

When this code is applied with `g` equal to `Y0`, we obtain filtered Radon data. The result is compared with the original Radon data in Fig. 4. Since the visualizations display relative magnitude, we see that the high frequency content (i.e. large t values) has now been amplified along each θ-slice. Indeed, this is clear from the fact that $\hat{h}(\omega) = |\omega|$, meaning the filter dampens low frequencies and strengthens high frequencies.

(d) We run the code presented in parts (b) and (c):



(a) Original Radon data, `Y0`

(b) Filtered Radon data, `my_filter(Y0)`

Figure 4: Effect of filtering by h

```
%% Backprojection

ff = backproj(Y0);

%% Filtered backprojection

f = backproj(my_filter(Y0));

%% Display results of problem 2

figure
imshow(ff / max(max(ff)));

figure
imshow(f / max(max(f)));
```

Results are displayed in Fig. 5. As expected, the backprojection provides a blurred version of the true image, while the filtered backprojection enhances contrast and edges, bringing out the fine details. This is enabled by the filter's amplification of high frequency content, since discontinuities (i.e. edges in the image) are created by a superposition of high frequency signals. On the other hand, low frequency content creates the larger, more general features of the image.

(a) Backprojection, `ff`

(b) Filtered backprojection, `f`

Figure 5: Radon reconstructions with no noise

(e) We introduce Poisson noise as before and display the results. See Fig. 6 and Fig. 7. The same comments as in problem 1, part (f) apply. It seems the noise is more effectual in these experiments, suggesting that the Fourier reconstructions may be less sensitive to noise than the Radon reconstructions.

(a) Backprojection, `ff`

(b) Filtered backprojection, `f`

Figure 6: Radon reconstructions with $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^6$

(a) Backprojection, `ff`

(b) Filtered backprojection, `f`

Figure 7: Radon reconstructions with $\lambda\|\boldsymbol{Y}_0\|_\infty = 10^3$