```
% gda11_01
% Radon transform example.  This version uses
% an image containing smooth features
% supports Section 11.6
% Note: The inversion of the Radom Transform
% requires working with 2D Fourier transforms,
% which is nitty-gritty stuff and requires
% extensive knowledge of their properties

% image mtrue(i,j) is N by N, in (x=i,y=j) space,
% range of both x, y is +/- 1
N = 256;

% independent variable x
Nx = N;
xmin = -1;
xmax = 1;
Dx = (xmax-xmin)/Nx;
x = xmin + Dx*[1:Nx]';

% independent variable y
Ny = N;
ymin = -1;
ymax = 1;
Dy = (ymax-ymin)/Ny;
y = ymin + Dy*[1:Ny]';

% true image m(x,y)
% test image is a sum of several Normal curves
xbar1=0.2;
ybar1=0;
s1=0.1;
mtrue = exp( -( ((x-xbar1).^2)*ones(1,Ny) + ones(Nx,1)*((y'-ybar1).^2))/(2*s1*s1));
xbar2=-0.3;
ybar2=0;
s2=0.05;
mtrue = mtrue+exp( -( ((x-xbar2).^2)*ones(1,Ny) + ones(Nx,1)*((y'-ybar2).^2))/(2*s2*s2));
xbar3=0;
ybar3=0.2;
s3=0.025;
mtrue = mtrue+exp( -( ((x-xbar3).^2)*ones(1,Ny) + ones(Nx,1)*((y'-ybar3).^2))/(2*s3*s3));

% some statistics, to compare with reconstructed image
fprintf('m has minimum value %f \n', min(min(mtrue)) );
```

```
m has minimum value 0.000000
```

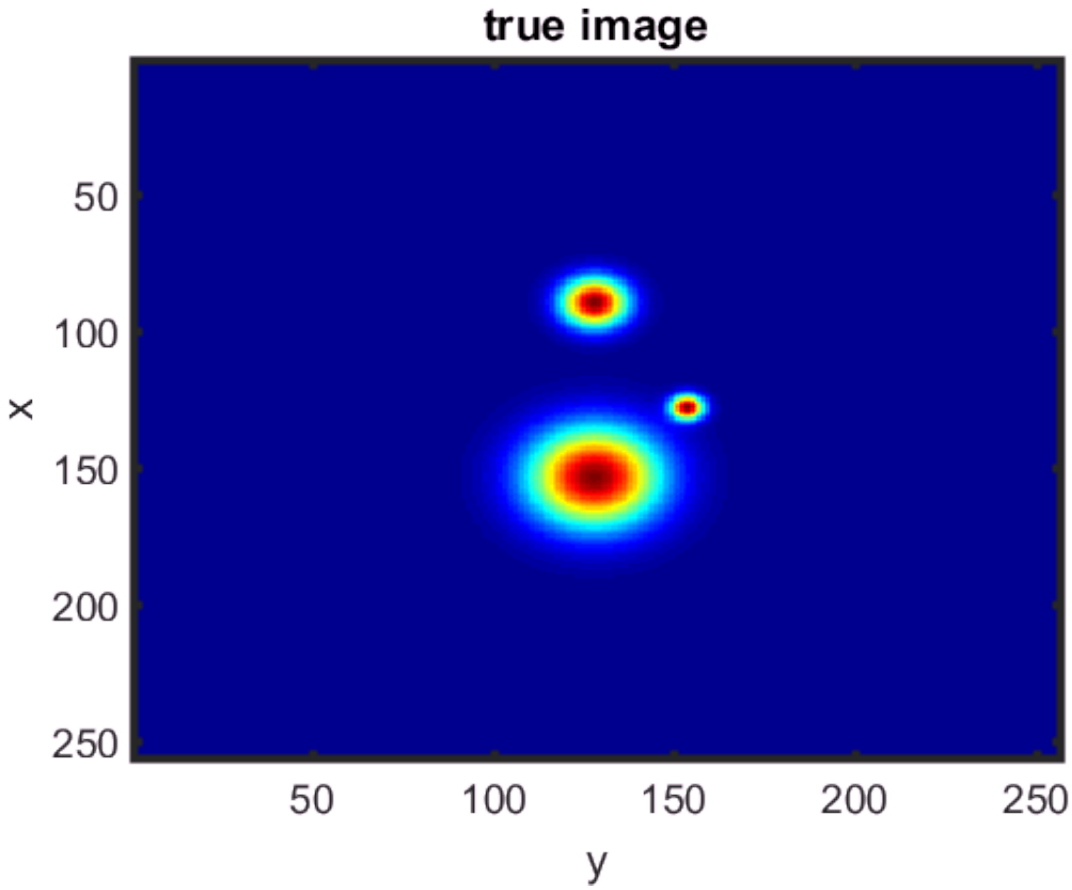
```
fprintf('m has maximum value %f \n', max(max(mtrue)) );
```

```
m has maximum value 1.009415
```

```
fprintf('\n');
```

```
% plot true image m(x,y)
figure(1)
clf;
```

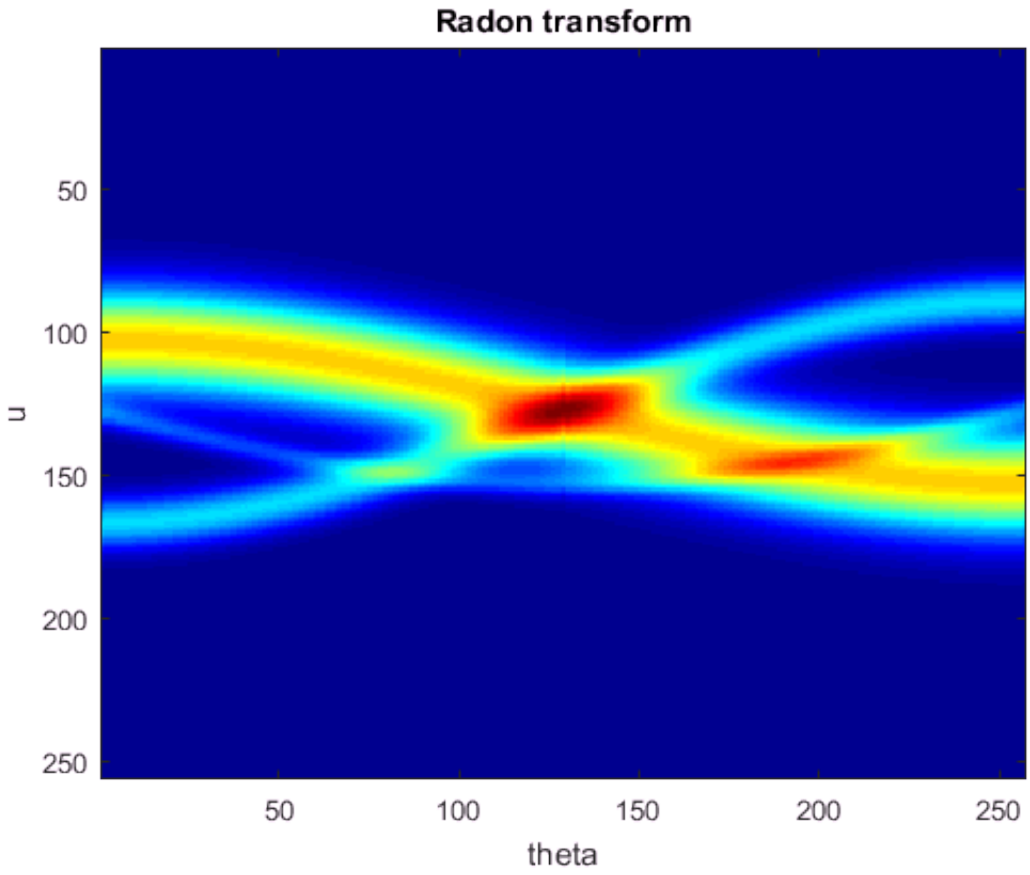
```
imagesc(mtrue);
set(gca,'LineWidth',3);
set(gca,'FontSize',14);
colormap('jet');
title('true image');
xlabel('y');
ylabel('x');
```




```
% Figure. Test image consisting of three circular features

% radon transform R(i,j) is in (u=i, q=theta=j) space
% range of u is +/- 1 (note corners of box cur off)
% range of q is 0 to pi

% independent variable u
Nu = N;
umin = -1;
umax = 1;
Du = (umax-umin)/Nu;
u = umin + Du*[1:Nu]';

% independent variable q
Nq = N+1;
qmin = -pi/2;
qmax = pi/2;
Dq = (qmax-qmin)/(Nq-1); % note includes both -pi/2 and pi/2
q = qmin + Dq*[0:Nq-1]'; % for ease of interpolation

% independent variable s
% integration over s, range +/- 1
```

```
Ns = 4*N;
smin = -1;
smax = 1;
Ds =(smax-smin)/Ns;
s = smin + Ds*[1:Ns]';

% brute force Radon transform via ray sums
R = zeros(Nu,Nq);
% loop over theta
for i = [1:Nq]

    % object here is top build Nu by Ns array of image, m,
    % evaluated at proper values of (u, s)

    % (x,y) from (u,s); these are Nu by Ns matrices
    xp = u*sin(q(i))*ones(1, Ns) - ones(Nu,1)*s'*cos(q(i))';
    yp = u*cos(q(i))*ones(1, Ns) + ones(Nu,1)*s'*sin(q(i));

    % indices corresponding to (x, y)
    ixp = floor((xp-xmin)/Dx) + 1;
    iyp = floor((yp-ymin)/Dy) + 1;

    % handle out of range x indices
    jxp1 = find(ixp<1);
    ixp(jxp1)=1;
    jxp2 = find(ixp>Nx);
    ixp(jxp2)=Nx;

    % handle out of range y indices
    jyp1 = find(iyp<1);
    iyp(jyp1)=1;
    jyp2 = find(iyp>Ny);
    iyp(jyp2)=Ny;

    % nasty array manipulation to sample image, mtrue,
    % at a series of (u, s) points, putting the values
    % in an Nu by Ns matrix, S

    % Step 1: convert Nu by Ns arrays to vectors
    Lixp = ixp( [1:Nu*Ns]' );
    Liyp = iyp( [1:Nu*Ns]' );
    % Step 2: generate linear index into image
    L = sub2ind( [Nx, Ny], Lixp, Liyp);
    % Step 3: sample image, mtrue, and rearrange it back into Nu by Ns
    % array
    S = reshape(mtrue(L),Nu,Ns);

    % now handle points that were off the edge of (x,y) space by
    % setting values to zero
    S( jxp1 ) = 0;
    S( jxp2 ) = 0;
    S( jyp1 ) = 0;
    S( jyp2 ) = 0;

    % integrate (sum) S(u,s) over s to get R(u) for fixed q
    R(:,i) = Ds*sum(S,2);
end

% plot the radon transform
figure(2)
clf;
```

```
set(gca,'LineWidth',3);
set(gca,'FontSize',14);
colormap('jet');
imagesc(R);
title('Radon transform');
xlabel('theta');
ylabel('u');
```




```
% Figure. Radon transform of the test image.

% statistics of the Radon transform
fprintf('R has minimum value %f \n', min(min(R)) );
```

```
R has minimum value 0.000000
```

```
fprintf('R has maximum value %f \n', max(max(R)) );
```

```
R has maximum value 0.375994
```

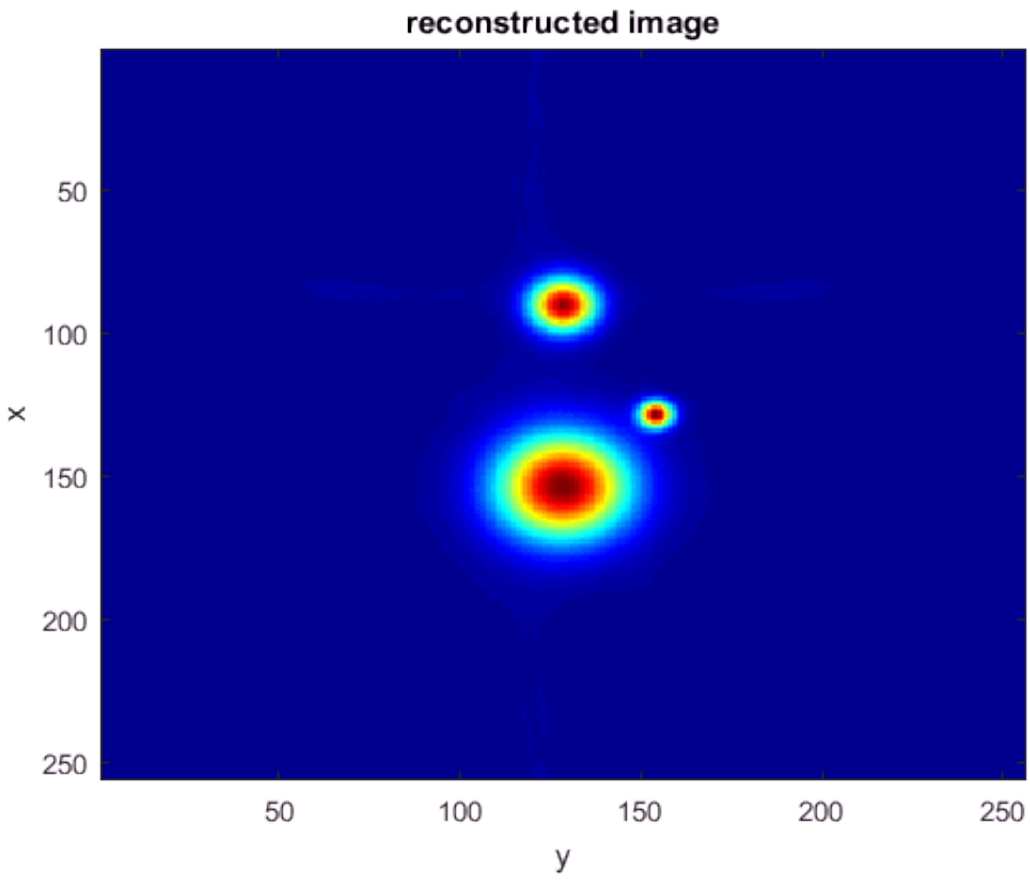
```
fprintf('\n');
```

```
% Inverse Radon Transform via Fourier transforms

% reorder array so that s=0, currenty at row N/2, is at row 1.
R = circshift(R, [-Nx/2+1,0] );
% now take 1D fourier transform over columns of R
```

```
Rt = fft(R);

% invert Radon transform via central slice theorem

% Fourier transform of image
mtt=zeros(Nx,Ny);

% fourier transform u -> ku
kumin=0;
kumax = 1/(2*Du);
Dku=kumax/(Nu/2);
ku=Dku*[0:Nu/2,-Nu/2+1:-1]';
Nku2 = Nu/2+1; % non-negative frequuencies

% fourier transform x -> kx
kxmin=0;
kxmax = 1/(2*Dx);
Dkx=kxmax/(Nx/2);
kx=Dkx*[0:Nx/2,-Nx/2+1:-1]';
Nkx2 = Nx/2+1; % non-negative frequuencies

% fourier transform y -> ky
kymin=0;
kymax = 1/(2*Dx);
Dky=kymax/(Ny/2);
ky=Dky*[0:Ny/2,-Ny/2+1:-1]';
Nky2 = Ny/2+1; % non-negative frequuencies

% Now use the Fourier Slice theorem. All the
% work is in sampling the radon transform at
% a series of points that correspond to wavenumber
% pairs on a rectangular grid.  That requires
% the Radon tranform to be interpolated.  I
% provide two ways of performing the interpolation.
% Note: Method 1 seems to work better.  I think
% that's because my complex interpolator works
% better than using MatLab's TriScattered Interp
% on the real and imaginary parts, separately

if( 1 ) % Method 1: element by element

for i = [1:Nx]
for j = [1:Nky2] % only need left side, will rebuild right using symmetry

        % theta calculation
        theta = atan2( kx(i), ky(j) );
        if( theta > (pi/2) ) % map large thetas onto negative thetas
            theta=pi-theta;
            thflag=1;
        else
            thflag=0;
        end

        % radial wavenumber
        kr = sqrt(kx(i)^2 + ky(j)^2);
        % index into randon transform array
        if( kr > kumax )
            continue; % ignore points that are out of bounds
        end

        % treat indices are real numbers
```

```
        krindex = (kr-kumin)/Dku+1;
        thindex = (theta-qmin)/Dq+1;
        if( thindex<1 )
            thindex=1;
        elseif( thindex>Nq )
            thindex=Nq;
        end

        % bracket real number indices with integers

        % radial index
        A = floor(krindex);
        B = floor(krindex+1);
        if( B >= Nku2 )
            A = Nku2-1;
            B= Nku2;
        end

        % theta index
        C = floor(thindex);
        D = floor(thindex+1);
        if( D >= Nq )
            C=Nq-1;
            D=Nq;
        end

        % Lagrangian polynomial interpoaltion
        % note that the indices are separated by unity
        FC  = (D-thindex);
        CKR = ((B-krindex)*Rt(A,C)+(krindex-A)*Rt(B,C));
        FD = (thindex-C);
        DKR = ((B-krindex)*Rt(A,D)+(krindex-A)*Rt(B,D));
        tmp = (FC*CKR+FD*DKR);
        if (thflag==1)
             mtt(i,j)=conj(tmp);
        else
             mtt(i,j)=tmp;
        end

end
end

else % Method 2: all at once

    % Rt is defined art these ku's and q's
    kkuu = ku*ones(1,Nq);
    qq = ones(Nu,1)*q';

    % create an interpolant
    % Rtinterpr = TriScatteredInterp( kkuu(:), qq(:), real(Rt(:)), 'natural' );
    % Rtinterpi = TriScatteredInterp( kkuu(:), qq(:), imag(Rt(:)), 'natural' );

    % MatLab's TriScatteredInterp() interpolator can't handle complex
    % numbers, so interpolate real and imaginary parts separately.  The
    % Delaunay triangulation needs to be performed only once, so do it
    % separately.
    triangles = DelaunayTri( kkuu(:), qq(:) );
    Rtinterpr = TriScatteredInterp( triangles, real(Rt(:)), 'natural' );
    Rtinterpi = TriScatteredInterp( triangles, imag(Rt(:)), 'natural' );

    % need all these elements of of mtt(i,j)
```

```
    %     ii = [1:Nx]'*ones(1,Nky2);
    %     jj = ones(Nx,1)*[1:Nky2];
    % these indices have kr and theta
    kkrr = sqrt( (kx(1:Nx)*ones(1,Nky2)).^2 + (ones(Nx,1)*ky(1:Nky2)').^2 );
    qq = atan2( kx(1:Nx)*ones(1,Nky2), ones(Nx,1)*ky(1:Nky2)' );

    % interpolate, set NaN's to zero, and populate mtt
    tmp = complex( Rtinterpr( kkrr(:), qq(:) ), Rtinterpi( kkrr(:), qq(:) ) );
    tmp(find(isnan(tmp)))=0;
    mtt(1:Nx, 1:Nky2) = reshape( tmp, Nx, Nky2 );

end

% Fourier transform of image needs to be phase shifted
% else origin is in the wrong place
xshift = (xmax-xmin)/2;
yshift = (ymax-ymin)/2;
phase = 2*pi*(kx*ones(1,Ny)*xshift + ones(Nx,1)*(ky')*yshift);
mtt = mtt .* complex( cos(phase), sin(phase) );

% impose all necessary symmetries required for a real image
% these four elements must be real
mtt(1,1)=real(mtt(1,1));
mtt(Nx/2+1,1)=real(mtt(Nx/2+1,1));
mtt(1,Ny/2+1)=real(mtt(1,Ny/2+1,1));
mtt(Nx/2+1,Ny/2+1)=real(mtt(Nx/2+1,Ny/2+1));
% bottoms of these two columns must be conjugates of top
mtt(Nx:-1:Nx/2+2,1)=conj(mtt(2:Nx/2,1));
mtt(Nx:-1:Nx/2+2,Ny/2+1)=conj(mtt(2:Nx/2,Ny/2+1));
% right hand side flipped conjugate of left hand side
for m = [2:Ny/2]
    mp=Ny-m+2;
    mtt(1,mp) = conj(mtt(1,m));
    mtt(2:Nx,mp) = flipud(conj(mtt(2:Nx,m)));
end

% now take inverse fourier transform
% Note: normalization not necessarily correct for non-square images
mest = ifft2(mtt)*(N/2);
mest = real(mest); % throw away imaginary part (which should be zero)

% plot reconstructed image
figure(3)
clf;
set(gca,'LineWidth',3);
set(gca,'FontSize',14);
colormap('jet');
imagesc(mest);
title('reconstructed image');
xlabel('y');
ylabel('x');
```




```
% Figure. Reconstructed test image.  Note faint  "+" shaped artifacts.

% compare amplitudes of reconstructed image to original,
% to be sure that the normalization is correct
fprintf('mest has minimum value %f \n', min(min(mest)) );
```

```
mest has minimum value -0.008493
```

```
fprintf('mest has maximum value %f \n', max(max(mest)) );
```

```
mest has maximum value 0.996005
```

```
fprintf('\n')
```