TSBB09 Laboratory Assignment: Tomographic image reconstruction

Developed by Henrik Turbell.
Updated by Maria Magnusson and Klas Nordberg.

Computer Vision Laboratory, Linköping University, Sweden

September 2000 - November 2014

Contents

1 Introduction

Computed tomography is since long an important tool for both medical and industrial applications. In this computer session we will look at how an object can be reconstructed from a set of X-ray projections. Your task is to extend and improve an already working MATLAB reconstruction program.

1.1 Preparations

Before coming to the computer session it is necessary to read through the course material on image reconstruction together with this document. Here you will also find a number of home exercises to be answered before the session. They are all clearly marked with a pointing finger.

Extra

Implementation exercises maked with an extra sign in the margin may be skipped and possibly completed when the other exercises are finished.

Examples of MATLAB commands frequently used are `find`, `imagesc`, `conv2`, `meshgrid`, `fft`, and `fftshift`. If you are unfamiliar with any of these commands, please use the MATLAB `help` function for more information.

1.2 The sinogram

The files you need are located in `/site/edu/bb/Bildsensorer/F-tomography` Copy them to a directory of your own. Start MATLAB and go to the directory where the file `fbp.m` is placed. Begin the reconstruction by typing `fbp` in the MATLAB prompt. After a few seconds, a window with four images should appear.

At the top left is our object. By calculating line integrals through the object, a *sinogram* of projection data is constructed. Study the sinogram in the top right image. How wide is the detector used in this experiment? Over what angular interval are the projections taken?

Which parts of the object correspond to which parts of the sinogram?

Home exercise The X-ray detector in a real tomograph does not detect the line integrals of the object density, but something related. What? How is this measurement related to the line integrals?

1.3 Filtered backprojection

We will use the *filtered backprojection* algorithm to reconstruct the object from the projection data in the sinogram. As the name suggests, this algorithm consists of two steps. First, the sinogram data are filtered column by column with a rampfilter. The filtered data are then smeared back, *backprojected*, along the rays that generated the unfiltered data.

At the lower left corner in the MATLAB window we see the rampfiltered sinogram and to the right we see the reconstructed image. Use the `caxis` command followed by `colorbar` to change the grayscale window of the reconstructed image. Inspect the reconstruction quality for values around 0.0 and around 1.0. It can be helpful to vary between the colormaps `gray` and `jet`. Approximately how big are the reconstruction errors?

Open the file `fbp.m` in an editor and remove the rampfiltering by replacing the line `q = rampfilter(...` with `q = p`. Also comment out the `caxis` command for the image down-right so that you get automatic scaling of the axes. Execute `fbp`. What has happened with the reconstruction result? Why? What is the purpose of the rampfilter?

Open the file `fbp.m` again and restore the rampfilter and the `caxis` command! Have you restored the ramp-filter and the `caxis` command?

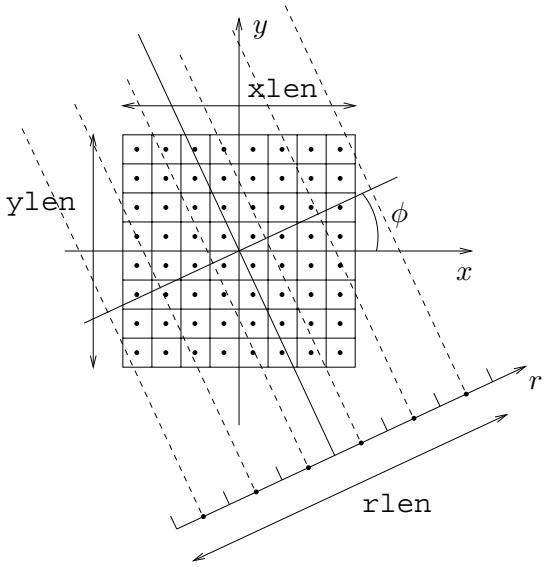

Figure 1: The geometry. In this example `Nx=Ny=8` and `Nr=6`.

Figure 1 shows the geometry of the detector and the image. Study the first part of `fbp.m`. Here, the sampling patterns of the image, detector and sinogram are defined. In order to fully cover the object, the detector needs to be $\sqrt{2}$ larger than the side of the square circumscribing the object. We make it slightly larger in order to avoid interpolation problem at the detector edge. We choose to sample the rays at the center of each detector element. The length of the detector, measured between the edges of the outer elements, is `rLen`. The distance between the two outermost rays is slightly smaller.

Now, we will make some experiments with the parameters.

What happens if we reduce the angular projection interval? Why?

What happens if we change the number of projection angles to a larger number?

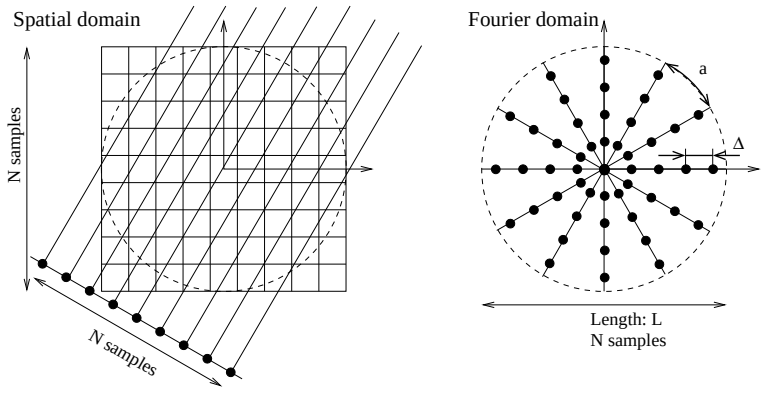

Figure 2: Help to derive the needed number of projection angles.

What happens if we change the number of projection angles to a smaller number?

What happens if we change the number of projection angles to a very small number?

Home exercise There is a rule on how many projection angles M that are needed for an image of diameter N samples and a detector resolution equal to the image resolution, i.e. the number of sample points in the detector is N, see Figure 2. Then, after FFT, the number of sample points along the radial direction is also N.

It is desired that the arc length a is smaller or equal to Δ. Set up an expression for a and then an expression for Δ. Then set $a \leq \Delta$, simplify and get the rule for the relation between M and N.

Use your findings in the last three tasks to answer the following question: What happens with the image quality if the rule is not fulfilled, i.e. $a > \Delta$.

---

2 Phantoms

When developing reconstruction algorithms it is often useful to use synthetically generated projection data instead of measurements from a real tomograph. The object, often called a *phantom*, is then defined using a set of geometrical primitives, such as circles, ellipses, and triangles. The main advantage is that the true object densities are known and can therefore easily be compared to the reconstructed result.

The Mickey phantom used above is solely constructed from circles. More realistic phantoms can be achieved if we use ellipses instead. We will now extend the program to handle ellipses.

2.1 Pixelization

In order to evaluate the reconstructed result it is essential to know the density values of the phantom at the pixel locations of the reconstructed image. The function `pixelize` returns such a density image.

```
1  function phm = pixelize(E, xVec, yVec);
2
3  [xGrid, yGrid] = meshgrid(xVec, yVec);
4  phm = zeros(size(xGrid));
5
6  for i = 1 : size(E, 1)
7     x0 = E(i, 1);          % x offset
8     y0 = E(i, 2);          % y offset
9     a = E(i, 3);           % radius
10    dens = E(i, 4);        % density
11    x = xGrid - x0;
12    y = yGrid - y0;
13    idx = find(x.^2 + y.^2 < a^2);
14    phm(idx) = phm(idx) + dens;
15 end
```

The matrix `E` contains a description of the phantom. Each line of the matrix defines one circle. The columns contain the x- and y-coordinates of the center, the radius, and the density.

Perform `help meshgrid` if you are unfamiliar with the MATLAB command on line 3. It takes two vectors and returns two matrices such that each position in the first matrix indicates the x-coordinate of each matrix element and the second matrix the y-coordinate. The first matrix is therefore constant along the columns and the second along the rows. The two matrices `xGrid` and `yGrid` enable so called *code vectorization* i.e. all pixels

in a matrix can be updated with a single statement. This makes the function run much faster than if `for`-loops would have been used.

The condition $(x-x_0)^2+(y-y_0)^2 < a^2$ is trivially given by the definition of a circle. Note the period before the circumflex on line 13 which indicates the use of an *element wise* square operator instead of a *matrix* square operator.

The `for`-loop on line 6 iterates over the circles of the phantom. The final result is an accumulation of all circles.

Home exercise What is the condition for the pixel (x, y) to be inside the ellipse with radii a and b, centered at (x_0, y_0), and rotation α as drawn in Figure 4?

Implementation exercise Extend `pixelization` to handle ellipses. Generate an image of the Shepp-Logan phantom defined by the six first columns of Table 1. Note that MATLAB requires angles to be given in radians.

x_0	y_0	a	b	α	realistic density	high contrast density
0	0	0.69	0.92	0°	2.00	1.0
0	-0.0184	0.6624	0.874	0°	-0.98	-0.6
0.22	0	0.11	0.31	-18°	-0.02	-0.2
-0.22	0	0.16	0.41	18°	-0.02	-0.2
0	0.35	0.21	0.25	0°	0.01	0.1
0	0.1	0.046	0.046	0°	0.01	0.1
0	-0.1	0.046	0.046	0°	0.01	0.1
-0.08	-0.605	0.046	0.023	0°	0.01	0.1
0	-0.606	0.023	0.023	0°	0.01	0.1
0.06	-0.605	0.023	0.046	0°	0.01	0.1

Table 1: Parameters for the Shepp-Logan phantom. The six first columns define the ordinary realistic brain phantom. The density values in the 7:th column give a high contrast version suitable for the movie in section 3.3.

First test your Shepp-Logan phantom for high contrast. Let the grid be 128×128 pixels. The Shepp-Logan phantom is a simple model of a head. The soft parts of the brain are surrounded by dense bone. The ellipses with densities in Table 1 are added to give the resulting phantom. Study the image and the table to answer: Which values are given to the soft tissues?

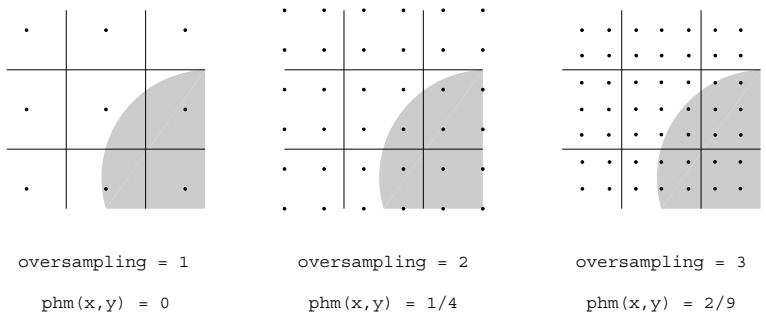

Figure 3: By oversampling the pixel a more acqurate value is obtained.

It is suitable to view the soft parts of the high contrast Shepp-Logan phantom in the grayscale window `caxis([0.2 0.6])`. Which interval is appropriate for studying the soft parts of the low contrast phantom?

In the following, use the low contrast phantom if nothing else is said. The ellipse description is a continuous model of the phantom, whereas the pixelized image is a discrete model of the phantom.

The ellipses produced have "jagged" edges. Instead of sampling the pixels in their center, we can take several samples spread out over the pixel and use the average sample value as the pixel value. See Figure 3. This will result in smoother and more correct values for pixels on an object edge. Similar techniques are often used in computer graphics whenever vector graphics, composed of curves and lines, need to be converted into pixels for presentation on a screen. They are known as anti-aliasing, over- or supersampling.

Implementation exercise The following code adds a fourth parameter, `oversampling`, to `pixelize`.

Extra

```
1   function phm = pixelize(E, xVec, yVec, oversampling);
2
3   Nx = length(xVec) * oversampling;
4   deltaX = (xVec(2) - xVec(1)) / oversampling;
5   xLen = deltaX * Nx;
6   Ny = length(yVec) * oversampling;
7   deltaY = (yVec(2) - yVec(1)) / oversampling;
8   yLen = deltaY * Ny;
9
10  xVec = (0.5 : Nx - 0.5) * xLen / Nx - xLen / 2;
11  yVec = (0.5 : Ny - 0.5) * yLen / Ny - yLen / 2;
```

```
12
13  [xGrid, yGrid] = meshgrid(xVec, yVec);
14  phm = zeros(size(xGrid));
15
16  for i = 1 : size(E, 1)
17     x0 = E(i, 1);          % x offset
18     y0 = E(i, 2);          % y offset
19     a = E(i, 3);           % radius
20     dens = E(i, 4);        % density
21     x = xGrid - x0;
22     y = yGrid - y0;
23     idx = find(x.^2 + y.^2 < a^2);
24     phm(idx) = phm(idx) + dens;
25  end
26
27  phm = conv2(phm, ones(oversampling) / oversampling^2, 'same');
28
29  phm = phm(round(oversampling/2):oversampling:end, ...
30    round(oversampling/2):oversampling:end);
```

This parameter is an integer specifying the oversampling rate such that `oversampling * oversampling` samples are taken for each pixel. The size of the returned image is still specified by `xVec` and `yVec`. It has been possible to avoid extra `for`-loops by using the function `conv2` in the averaging process. Describe what is happening to the phantom image when you change `oversampling` from 1 to 3. You might need to zoom in.

2.2 Projection generation

Home exercise The maximal sampling distance is normally given by the Nyquist theorem. What is the bandwidth and the corresponding sampling distance of the continuous Shepp-Logan phantom? Clue: The Shepp-Logan phantom consists of a number of ellipses - what is the Fourier transform of an ellipse?

Home exercise Guided by the previous answer, what can we expect of the image quality if we take discrete projections of the continuous Shepp-Logan phantom and apply any reconstruction algorithm?

The projection generation described here simulates the measurements in the tomograph. It is *not* part of the reconstruction algorithm.
Each position in the sinogram corresponds to one line integral in the object image. The line integral through a circle is proportional to the length of intersection between the line and the circle. For a circle of radius a, centered at (x_0, y_0), the intersection length becomes

$$l(r,\phi) = \begin{cases} 2\sqrt{a^2 - (r - x_0\cos\phi - y_0\sin\phi)^2} & \text{if } |r - x_0\cos\phi - y_0\sin\phi| < a \\ 0 & \text{otherwise} \end{cases}$$

where (r, ϕ) defines the line as in Figure 1.

The function `generateProj` takes a phantom description matrix `E` and generates the sinogram `p` sampled at the grid points given in the vectors `rVec` and `phiVec`.

```
1  function p = generateProj(E, rVec, phiVec, oversampling)
2
3  Nr = length(rVec) * oversampling;
4  rLen = (rVec(2) - rVec(1)) / oversampling * Nr;
5  rVec = (0.5 : Nr - 0.5) * rLen / Nr - rLen / 2;
6
7  x0 = E(:, 1);
8  y0 = E(:, 2);
9  a = E(:, 3);
10 dens = E(:, 4);
11
12 p = zeros(length(rVec), length(phiVec));
13
14 for phiIx = 1:length(phiVec)
15   phi = phiVec(phiIx);
16   for i = 1:length(x0)
17     aSqr = a.^2;
18     r1Sqr = (rVec' - x0(i) * cos(phi) - y0(i) * sin(phi)).^2;
19     ind = find(aSqr(i) > r1Sqr);
20     p(ind, phiIx) = p(ind, phiIx) + 2*dens(i)*sqrt(aSqr(i) - r1Sqr(ind));
21   end
22 end
23
24 p = conv2(p, ones(oversampling, 1) / oversampling, 'same');
25 p = p(round(oversampling/2):oversampling:end, :);
```

The loops on lines 14 and 16 iterate over the projections and circles respectively. Note the code vectorization on line 20. Here, all lines that intersect the current circle get updated in a single statement.

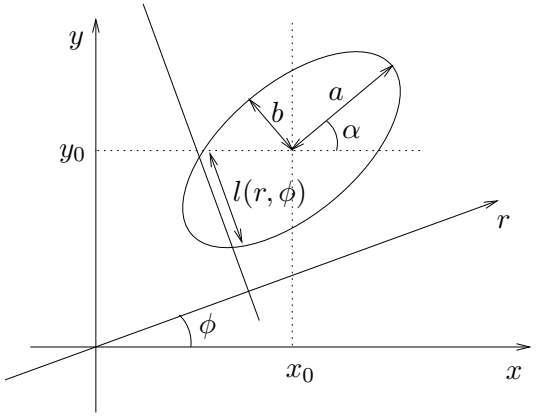

Figure 4: Intersection between a line and an ellipse.

Figure 4 shows the length of intersection between a line and an ellipse. It is straight forward to show that the length $l(r,\phi)$ is given by

$$l(r,\phi) = \begin{cases} 2ab\frac{\sqrt{r_0^2-(r-x_0\cos\phi-y_0\sin\phi)^2}}{r_0^2} & \text{if } |r-x_0\cos\phi-y_0\sin\phi| < r_0 \\ 0 & \text{otherwise} \end{cases}$$

where

$$r_0^2 = a^2\cos^2(\phi-\alpha) + b^2\sin^2(\phi-\alpha)$$

Home exercise Study the code in `generateProj` and extend it so that it works for phantoms defined by ellipses. Reconstruct the Shepp-Logan phantom on a grid of 128×128 pixels, 192 detector elements and `rLen = xLen * 1.5`. Use 128 projection angles.

Implementation exercise Test your new projection generation. Use the low contrast phantom and reconstruct as before. Is the reconstructed image quality similar to the phantom? **Show the image to the teacher!**

If `oversampling = 1`, one ray is cast onto the center of each detector element. A better model of a real X-ray detector is to cast several rays through the Shepp-Logan phantom onto each detector element and then use the average value of these line integrals. It is the same anti-aliasing technique as

we used in `pixelize` above. This technique is also frequently used in ray-tracing software packages that render photo realistic images of 3D scenes.

Up to now we have used `oversampling = 5`. Change to `oversampling = 1`. Show the result for `oversampling = 1` and `oversampling = 5` beside eachother, because they differ only slightly. How is the image quality affected? Try to explain why.

Change back to `oversampling = 5` again.

3 Backprojection

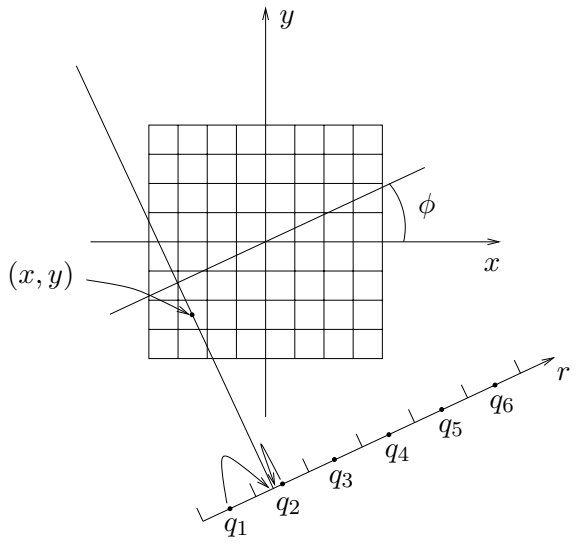

Figure 5: Backprojection. A linear interplotion gives the filtered projection value to be delivered to the pixel (x, y).

The filtered result is backprojected into the image. Each pixel should then get a contribution from each projection. When deciding which detector element that should be used for a certain projection, the center of the pixel is projected on the detector. See Figure 5. This projection does not normally coincide with a sampling point on the detector, which calls for a one-dimensional interpolation. We have implemented nearest neighbour interpolation and you will implement linear interpolation.

3.1 Nearest neighbour interpolation

The backprojection is performed by the function `backproject`:

```
1  function f = backproject(q, rVec, phiVec, xVec, yVec, intpol)
2
3  [xGrid, yGrid] = meshgrid(xVec, yVec);
4  f = zeros(size(xGrid));
5  Nr = length(rVec);
6  Nphi = length(phiVec);
7  deltaR = rVec(2) - rVec(1);
8
9  switch intpol
10   case 'nearest'
11     for phiIx = 1:Nphi
12       phi = phiVec(phiIx);
13       proj = q(:, phiIx);
14       r = xGrid * cos(phi) + yGrid * sin(phi);
15       rIx = round(r / deltaR + (Nr + 1) / 2);
16       f = f + proj(rIx);
17     end
18   case 'linear'
19     % your code here
20   otherwise
21     error('Unknown interpolation.');
22 end
23
24 f = f * pi / (Nphi*deltaR);
```

Again, we make use of code vectorization. The detector position r is calculated for all pixels (x, y) on line 14. Each element in the array r is then scaled, translated and rounded to an integer vector index on line 15. The contributions to each pixel are finally accumulated on line 16.

The final scaling with $\frac{\pi}{N_\phi}$ on line 24 is due to the fact that we try to estimate the integral $\int_0^\pi \ldots d\phi$ with a sum. The infinitesimal element $d\phi$ then translates to $\frac{\pi}{N_\phi}$. The scaling with $\frac{1}{deltaR}$ is actually belonging to the rampfilter.

Home exercise Motivate the equation on line 15 by completing Figure 6. Indicate clearly which values of `r` that are mapped to which values of `rIx`.

3.2 Linear interpolation

The nearest neighbour interpolation works satifactory if the sampling distance on the detector is significantally smaller than the sampling distance of the pixels.

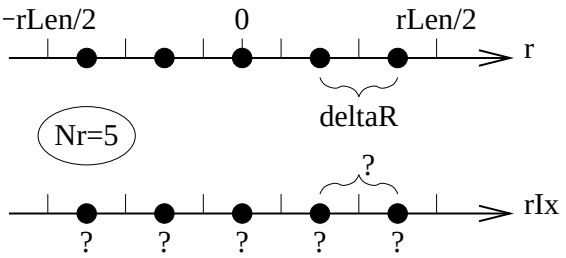

Figure 6: Mapping between the continuous `r`-axis to the discrete `rIx`-axis.

Home exercise In linear interpolation each pixel should recieve a weighted sum of the two detector values closest to the actual detector position `r`. Give expressions of the indexes of the two sample points closest to `r`. The indexes must naturally be integers between 1 and `Nr`. Also give expressions of the two interpolation weights.

Implementation exercise Implement backprojection with linear interpolation. Reconstruct the Shepp-Logan phantom using linear interpolation and compare the result when using nearest neighbour interpolation. Comments? **Save both images and show them to the teacher!**

Extra

Image quality comparison We have now good program code for pixelization, projection generation, and backprojection. The image quality can be evaluted in many ways, by simply looking at the images, plotting a line in the image as

```
plot(f(:,64));
axis([1 128 1.00 1.04]);
```

or by some mathematical measurement, for example the following

```
sum(sum((f-phm).^2))/(size(f,1)*size(f,2))
```

which are not optimal if we are especially interested in the soft tissue, however.

3.3 Animation

We will now create an animated movie of the accumulation process in the backprojection. The statement

```
M(phiIx) = getframe;
```

takes a snaphot of the current plot window and stores it in the 3D array `M`. All snapshots can then be played as a movie using the function

```
movie(M);
```

Implementation exercise Produce an image of `f` after the accumulation from each backprojection. Take snaphots of the images. Instead of returning `f` from `backproject`, return `[f M]`. When `ftp` is ready, the movie can be played with `movie(M)`. Use the *high contrast* Shepp-Logan phantom defined by the alternative density values in the last column of Table 1. Did you manage? **Show the film to the teacher!**

---

4 Filtering

We design the rampfilter $h(r)$ in the signal domain. The filtering is then performed as the convolution $q = f * h$ for each projection angle. The rampfilter kernel is very long. Projection values all over the detector collaborate to yield a single filtered value. Rampfiltering is in this sense a *non-local* operation. The long filter slows down the convolution and we will therefore implement the filtering more efficiently as a multiplication in the Fourier domain. But first we look at the signal domain implementation.

4.1 Filtering in the signal domain

The bandlimited rampfilter can in the Fourier domain be written as the difference between a rectangle function $\square$ and a triangle function $\triangle$. The width of the ramp is $\frac{1}{\Delta r}$, where Δr is the sampling distance. The rectangle and triangle functions have well-known inverse Fourier transforms, giving

$$h(r) = \mathcal{F}^{-1}\{\square - \triangle\} = \frac{1}{2(\Delta r)^2}\mathrm{sinc}(\frac{r}{\Delta r}) - \frac{1}{4(\Delta r)^2}\mathrm{sinc}^2(\frac{r}{2\Delta r})$$

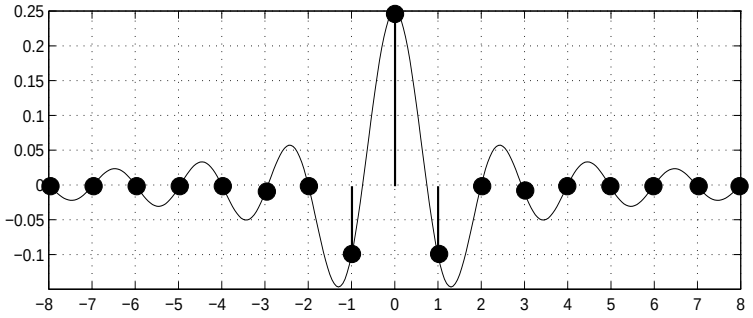

Figure 7: The continous rampfilter $h(r)$ and the sample points at $h(i\Delta r),\ -8 \leq i \leq 8$. $\Delta r = 1$ in the figure.

This function is plotted in Figure 7 for $\Delta r = 1$. The values in the sample points become

$$h(i\Delta r) = \begin{cases} \frac{1}{(2\Delta r)^2}, & i = 0 \\ 0, & i \,\text{even} \\ -\frac{1}{(i\pi\Delta r)^2}, & i \,\text{odd} \end{cases}$$

The rampfiltering is performed by the following function:

```
1  function q = rampfilter(p, rVec, domain)
2
3  Nr = length(rVec);
4  rDelta = rVec(2) - rVec(1);
5  rVecDouble = (-Nr:Nr-1) * rDelta;
6  Nphi = size(p,2);
7  odd = rem(Nr,2);
8
9  h = zeros(2*Nr, 1);
10 h(2-odd:2:end) = -1 ./ (pi^2 * rVecDouble(2-odd:2:end).^2);
11 h(Nr+1) = 1 / (4 * rDelta^2);
12 h = h * rDelta^2;
13 %subplot(2,2,1); plot(rVecDouble,h);
14
15 switch domain
16   case 'signal'
17     q = conv2(p, h, 'same');
18   case 'fourier1'
19     % code for extra task here
20   case 'fourier2'
21     Hvec = 0.5*(-Nr:Nr-1)/Nr;
22     H = fftshift(fft(ifftshift(h)));
23 %    subplot(2,2,2); plot(Hvec,real(H));
24     Hmat = H  * ones(1,Nphi);
25     pp = [zeros((Nr+odd)/2, Nphi); p; zeros((Nr-odd)/2, Nphi)];
```

```
26      P = fftshift(fft(ifftshift(pp), [], 1));
27      Q = P .* Hmat;
28      q = real(fftshift(ifft(ifftshift(Q), [], 1)));
29      q = q((Nr+odd)/2+1:(Nr+odd)/2+Nr, :);
30    otherwise
31      error('Unknown domain.');
32  end
```

Lines 3–5 construct a vector used to define the ramp filter on lines 9–12. Compare the length of the ramp filter with the length of one projection. How and why are they different?

The following little program `ramptest.m` calls the `rampfilter` function.

```
1  Nr = 192;                                % number of detector elements
2  rLen = xLen * 1.5;                       % length of detector
3
4  rVec = (0.5:Nr-0.5) * rLen/Nr - rLen/2; % r-axis
5  p = ones(Nr);
6  domain = 'signal';
7  q = rampfilter(p, rVec, domain);
```

Take away the %-character in front of the first `plot`-command in rampfilter.m and run ramptest.m. Study how the plot corresponds to Figure 7. You might need to change the axis in the plot by using `axis([-0.1 0.1 -0.2 0.25])`. The shape of the filters should be the same, but the sampling distance differ. What sampling distance Δr is used in the program?

The convolution is made by the function `conv2` on line 17. This function is normally used for two-dimensional convolution but we use it for the one-dimensional convolution. Since the filter kernel is a one-dimensional column vector, all columns of the sinogram will be filtered by this single function call. What does the argument `'same'` stand for?

Extra

Comment: We now know that `conv2` assumes the values outside the sinogram to be zero. The result from the filtering can therefore only be valid if the detector is wide enough to cover the complete object from all projection angles. The projections must in other words not not be *truncated*. In medical applications the radiologist may only be interested in a small region of a slice though the body. The projections must nevertheless cover the full width of the patient so that untruncated projections are measured. The patient is then exposed to seemingly unnecessary radiation. Research efforts that try to find algorithms with more local filters, allowing for truncated projections, go under the name *local tomography*.

4.2 Filtering in the Fourier domain

Before we start transforming vectors into the Fourier domain, we must better remind ourselves about some nitty gritty MATLAB details. The function `fft` assumes the origin of the signal to be at the first position of the input vector. The `ifftshift` shifts the origin in the signal domain from the middle position to the first position as shown in Figure 8. The result from `fft` has the Fourier origin, that is the DC-component, in the first position of the output vector. The `fftshift` command shifts it to the middle position where we expect to find it when we plot the function.

Now we are ready to transform the rampfilter into the Fourier domain by

```
H = fftshift(fft(ifftshift(h)));
```

Try to plot it. The filter designed on lines 9–12 is a real even signal. What do we expect of the Fourier transform of such a signal in theory? Why is this not the case on the computer? How can we make a plot of it?

Implementation exercise Try the rampfilter implemented in the fourier domain by calling the rampfilter.m function with 'fourier2'. What is the effect of the following code?

```
pp = [zeros((Nr+odd)/2, Nphi); p; zeros((Nr-odd)/2, Nphi)];
```

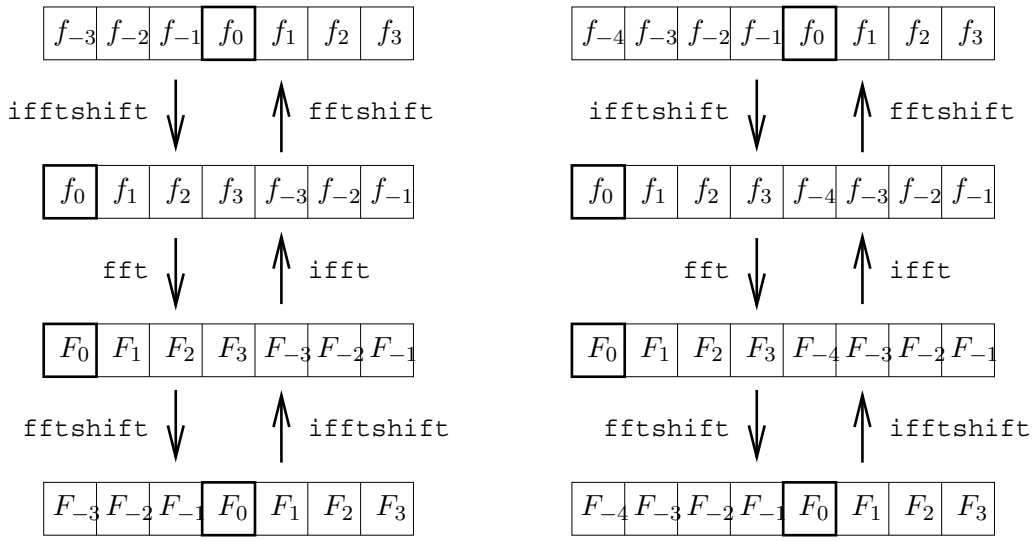

Figure 8: The functions `fftshift` and `ifftshift` are used to place the origin at the correct positions when transforming vectors. Left: odd number of samples. Right: even number of samples.

Compare the reconstruction results from the two ways of performing the filtering. How large are the differences?

We will now try to reduce the computations by designing the rampfilter directly in the Fourier domain. We try to avoid the zero-padding above by making the filter as long as the projections.

Implementation exercise Design a ramp filter of length `Nr` in the Fourier domain by adding the following code to rampfilter.m.

```
1     case 'fourier1'
2       Hvec = (-(Nr+odd)/2:(Nr+odd)/2-1)/(Nr+odd);
3       H = abs(Hvec');
4       Hmat = H  * ones(1,Nphi);
5       P = fftshift(fft(p, [], 1));
6       Q = P .* Hmat;
7       q = real(ifft(ifftshift(Q), [], 1));
```

Plot this filter in the Fourier domain and compare it with the filter called by 'fourier2'. Can you see any differences? Zoom around 0!

Use this shorter filter and inspect the reconstruction result. What has happened? You will need to adjust the `caxis([1 1.04])` command. Show this image to the teacher!

Replace this short incorrect filter with the longer one derived in the signal domain.

Extra

The `fft` function in MATLAB accepts vectors of arbitrary length. But as noted in the `help` text, the performance depends on the length of the vector. The basic condition for the transform to be fast is that the size should be a power of 2. Recently, however, there has been a developement of fast fft-algorithms so that it is no longer nescessary to have a power of 2 size. Still some sizes give a slow fft.

5 Computational complexity

See Figure 5, which illustrates backprojection. Suppose that there are $N_x \cdot N_y$ pixels in the image plane. During backprojection of the projection at angle ϕ, each pixel should get a constribution, i.e. the complexity is $\mathcal{O}(N_x \cdot N_y)$. The total number of projection angles is N_ϕ, and consequently the complextity for the whole backprojection procedure is: .....

See Figure 9, which illustrates filtering in the spatial domain using convolution. In the figure, p is a projection of size $N_r = 8$, h is a rampfilter of size $\geq 2N_r - 1$, and q is a filtered projection of size $N_r = 8$.
To calculate 1 value in q demands: N_r mult, $N_r - 1$ add $\Rightarrow \mathcal{O}(N_r)$
To calculate all values in q demands: .....
To calculate all values in q for all proj. angles N_ϕ demands: .....

See Figure 10, which illustrates filtering in the Fourier domain. In the fig-

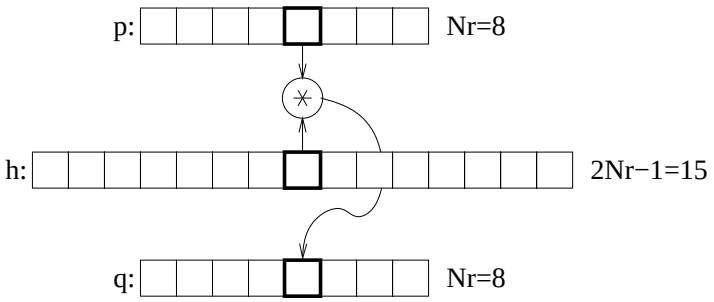

Figure 9: Ramp filtering in the spatial domain.

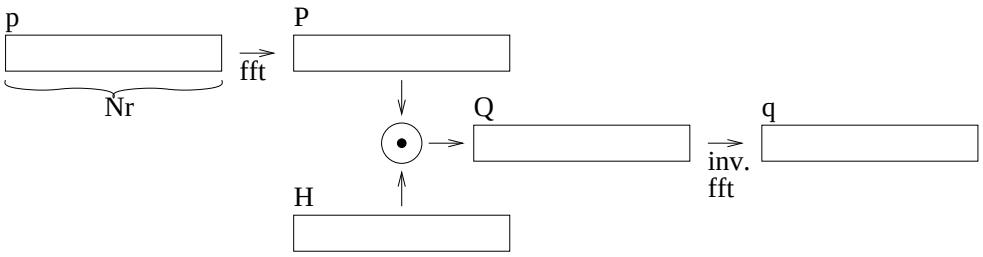

Figure 10: Ramp filtering in the Fourier domain.

ure, p is a projection of size N_r. The fft-calculation of p demands $\mathcal{O}(N_r \log N_r)$ operations and the multiplication of P and H demands $\mathcal{O}(N_r)$ operations. Therefore, to calculate q demands: .....
The calculation should be performed for all angles N_ϕ, which gives: .....

Home exercise After considering the content in this section, you should be able to complete Table 2.

Operation	Complexity as a function of N_x, N_y, N_r, and N_ϕ	Complexity as a function of N when $N = N_x = N_y = N_r = N_\phi$
Backprojection		
Filtering (in the signal domain)		
Filtering (in the Fourier domain)		

Table 2: Complexity of the different algorithms.

Consequently, which part of the reconstruction is the most time consuming?

To accelerate the speed of the reconstruction on real CT-scanners, this part is normally implemented in hardware, i.e. ASIC or FPGA.

Acknowledgements

The author wants to thank Prof. Doris Hinestoza for her elegant projection generation code which ignited the development of this lab. Comments and suggestions from Maria Magnusson-Seger, Katarina Flood, and Prof. Per-Erik Danielsson have also been of great value.

A Complete MATLAB code

A.1 fbp.m

```
%%%
% GEOMETRY SETUP
%%%

Nx = 64;                                % number of grid points in x
Ny = 64;                                % number of grid points in y
Nr = 100;                               % number of detector elements
Nphi = 100;                             % number of projection angles

xLen = 2.0;                             % side of reconstructed rectangle
yLen = 2.0;                             % side of reconstructed rectangle
phiLen = pi;                            % angular interval
rLen = xLen * 1.5;                      % length of detector

phiVec = (0 : Nphi - 1) * phiLen / Nphi;
rVec = (0.5 : Nr - 0.5) * rLen / Nr - rLen / 2;
xVec = (0.5 : Nx - 0.5) * xLen / Nx - xLen / 2;
yVec = (0.5 : Ny - 0.5) * yLen / Ny - yLen / 2;

%%%
% PHANTOM DEFINITION
%%%

E = [ 0.0  -0.2  0.6  1.0; ...
      0.5  0.55  0.3  1.0; ...
```

```
      -0.5  0.55  0.3  1.0];

phm = pixelize(E, xVec, yVec);
subplot(2,2,1); imagesc(yVec, xVec, phm); title('Phantom');
axis xy; axis image; colorbar; colormap gray; pause(0);

%%%
% SINOGRAM GENERATION
%%%

p = generateProj(E, rVec, phiVec, 5);
subplot(2,2,2); imagesc(phiVec, rVec, p); title('Sinogram');
xlabel('\phi'); ylabel('r'); axis xy; colorbar; pause(0);

%%%
% RECONSTRUCTION
%%%

q = rampfilter(p, rVec, 'signal');
subplot(2,2,3); imagesc(phiVec, rVec, q);
axis xy; colorbar; title('Rampfiltered sinogram'); pause(0);

f = backproject(q, rVec, phiVec, xVec, yVec, 'nearest');
subplot(2,2,4); imagesc(yVec, xVec, f);
axis xy; axis image;  colorbar; title('Reconstructed image'); pause(0);
```

A.2 pixelize.m

```
function phm = pixelize(E, xVec, yVec);

[xGrid, yGrid] = meshgrid(xVec, yVec);
phm = zeros(size(xGrid));

for i = 1 : size(E, 1)
   x0 = E(i, 1);          % x offset
   y0 = E(i, 2);          % y offset
   a = E(i, 3);           % radius
   dens = E(i, 4);        % density
   x = xGrid - x0;
   y = yGrid - y0;
   idx = find(x.^2 + y.^2 < a^2);
   phm(idx) = phm(idx) + dens;
end
```

A.3 generateProj.m

```
function p = generateProj(E, rVec, phiVec, oversampling)

Nr = length(rVec) * oversampling;
rLen = (rVec(2) - rVec(1)) / oversampling * Nr;
```

```
5   rVec = (0.5 : Nr - 0.5) * rLen / Nr - rLen / 2;
6
7   x0 = E(:, 1);
8   y0 = E(:, 2);
9   a = E(:, 3);
10  dens = E(:, 4);
11
12  p = zeros(length(rVec), length(phiVec));
13
14  for phiIx = 1:length(phiVec)
15    phi = phiVec(phiIx);
16    for i = 1:length(x0)
17      aSqr = a.^2;
18      r1Sqr = (rVec' - x0(i) * cos(phi) - y0(i) * sin(phi)).^2;
19      ind = find(aSqr(i) > r1Sqr);
20      p(ind, phiIx) = p(ind, phiIx) + 2*dens(i)*sqrt(aSqr(i) - r1Sqr(ind));
21    end
22  end
23
24  p = conv2(p, ones(oversampling, 1) / oversampling, 'same');
25  p = p(round(oversampling/2):oversampling:end, :);
```

A.4 rampfilter.m

```
1   function q = rampfilter(p, rVec, domain)
2
3   Nr = length(rVec);
4   rDelta = rVec(2) - rVec(1);
5   rVecDouble = (-Nr:Nr-1) * rDelta;
6   Nphi = size(p,2);
7   odd = rem(Nr,2);
8
9   h = zeros(2*Nr, 1);
10  h(2-odd:2:end) = -1 ./ (pi^2 * rVecDouble(2-odd:2:end).^2);
11  h(Nr+1) = 1 / (4 * rDelta^2);
12  h = h * rDelta^2;
13  %subplot(2,2,1); plot(rVecDouble,h);
14
15  switch domain
16    case 'signal'
17      q = conv2(p, h, 'same');
18    case 'fourier1'
19      % code for extra task here
20    case 'fourier2'
21      Hvec = 0.5*(-Nr:Nr-1)/Nr;
22      H = fftshift(fft(ifftshift(h)));
23  %    subplot(2,2,2); plot(Hvec,real(H));
24      Hmat = H  * ones(1,Nphi);
25      pp = [zeros((Nr+odd)/2, Nphi); p; zeros((Nr-odd)/2, Nphi)];
26      P = fftshift(fft(ifftshift(pp), [], 1));
27      Q = P .* Hmat;
28      q = real(fftshift(ifft(ifftshift(Q), [], 1)));
```

```
29      q = q((Nr+odd)/2+1:(Nr+odd)/2+Nr, :);
30    otherwise
31      error('Unknown domain.');
32  end
```

A.5 backproject.m

```
1   function f = backproject(q, rVec, phiVec, xVec, yVec, intpol)
2
3   [xGrid, yGrid] = meshgrid(xVec, yVec);
4   f = zeros(size(xGrid));
5   Nr = length(rVec);
6   Nphi = length(phiVec);
7   deltaR = rVec(2) - rVec(1);
8
9   switch intpol
10    case 'nearest'
11      for phiIx = 1:Nphi
12        phi = phiVec(phiIx);
13        proj = q(:, phiIx);
14        r = xGrid * cos(phi) + yGrid * sin(phi);
15        rIx = round(r / deltaR + (Nr + 1) / 2);
16        f = f + proj(rIx);
17      end
18    case 'linear'
19      % your code here
20    otherwise
21      error('Unknown interpolation.');
22  end
23
24  f = f * pi / (Nphi*deltaR);
```