Abstract

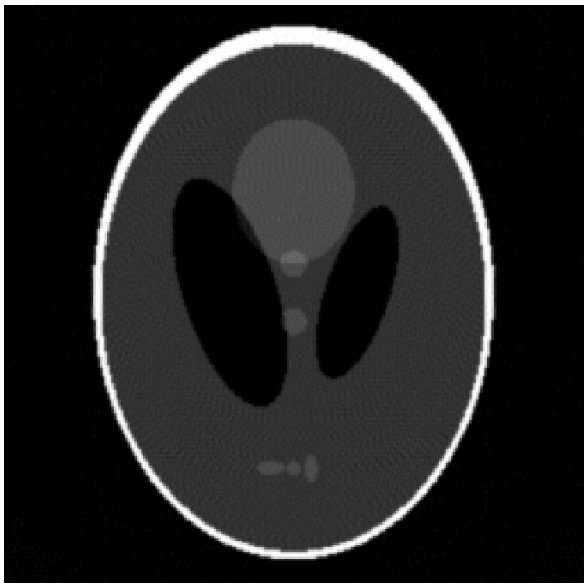
This report corresponds to the evaluation of the obtained results in the fourth laboratory of Applied Harmonic Analysis.

1 The Radon Transform, the filtered backprojection

The purpose of this delivery is to implement the function *iradon*, i.e., the inverse Radon transform. The program should take as an input a matrix with columns corresponding to the Radon transform along the angles given in the vector that you pass a second parameter. The output of the program should be a matrix that correspond to an approximation to the original image.

To begin with, we utilized the *phantom* command to acquire a matrix containing real numbers, which effectively depicts an image. By employing readily available Octave functions, we can effectively execute the Radon transform on the image using the *radon* function and subsequently reconstruct it using the *iradon* function, see Figure 1.

(a) Original images.



(b) Reconstructed image via Octave functions.

Figure 1: Original and reconstructed image via Octave functions.

As said before, the goal of the delivery is to create our *iradon* function. In order to implement it, I have followed the proposed scheme:

- **Step 1**. For every direction $\theta_j, j = 1, \ldots, p$ take the discrete convolution

$$h_{j,k} = \Delta s \sum_{l=-q}^{q} v_\Omega(s_k - s_l) g_{j,l}, \quad (k = -q, \ldots, q)$$

  where v_Ω is, for instance, the Ram-Lak filter.

- **Step 2**. For each x, compute the discrete backprojection using a linear interpolation of the values obtained in Step 1:

$$f_A(x) = \frac{2\pi}{p} \sum_{j=0}^{p-1} (1 - \eta) h_{j,k} + \mu h_{j,k+1},$$

where $k = k(j,x) = \lfloor \frac{x\cdot\theta_j}{\Delta s} \rfloor$, $\eta = \eta(j,x) = \frac{x\cdot\theta_j}{\Delta s} - k$.

In the script, we have made a function named *Ram_Lak* that computes the Ram-Lak filter function for a given value. Recall that the Ram-Lak filter has the following form:

$$v_\Omega(s) = \Omega^2 u(2\pi\Omega s), \qquad \text{where} \qquad u(s) = \text{sinc}(s) - \frac{1}{2}\left(\text{sinc}\left(\frac{s}{2}\right)\right)^2$$

To compute the first step of the algorithm, we need the discretization of s, which is given in the *phantom* function as a name *xp*. This vector contains the grid $(\Delta s\mathbb{Z})$ with $\Delta s \leqslant 1/2\Omega$[1]. We are ready to perform the first step by applying a triple loop (in j, in k and in l) and compute the matrix $h_{j,k}$[2].

Caution: The indices of the matrix are turned, to implemented in Octave, the matrices should be $g_{l,j}$ instead of $g_{j,l}$ and $h_{k,j}$ instead of $h_{j,k}$.[3]

Finally, we are ready to compute the second step, the discrete backprojection using a linear interpolation of the values obtained in the first step. To perform this step, we create an empty matrix in which we are going to add the value of each pixel. Due to the structure of our problem, we need to set the origin in the middle of the matrix. Now, considering points (x,y) in the plane and taking into account the mentioned structure, we can compute $f_A(\mathbf{x})$, where $\mathbf{x} = (x,y)$. Notice that, to compute $f_A(\mathbf{x})$, we need the values $k = k(j,\mathbf{x}) = \lfloor \frac{\mathbf{x}\cdot\theta_j}{\Delta s} \rfloor$ and $\eta = \eta(j,\mathbf{x}) = \frac{\mathbf{x}\cdot\theta_j}{\Delta s} - k$. This requires evaluating a scalar product between $\mathbf{x}$ and $\theta_j = e^{\frac{ij\pi}{180}}$. In order to perform this scalar product, we simply take the real and imaginary part of θ_j and compute the standard scalar product. With these considerations[4], we easily compute k and η and all that remains is to perform the previous loop in order to sum all the values.

Just like before, by utilizing the *imshow* command, we can witness that our *iradon* function produces remarkable outcomes, akin to those of Octave, as depicted in the Figure 2. However, there is a noticeable discrepancy in terms of computation time. While the Octave function completes its task in less than 10 seconds, ours requires approximately 1102 seconds (equivalent to around 18 minutes).

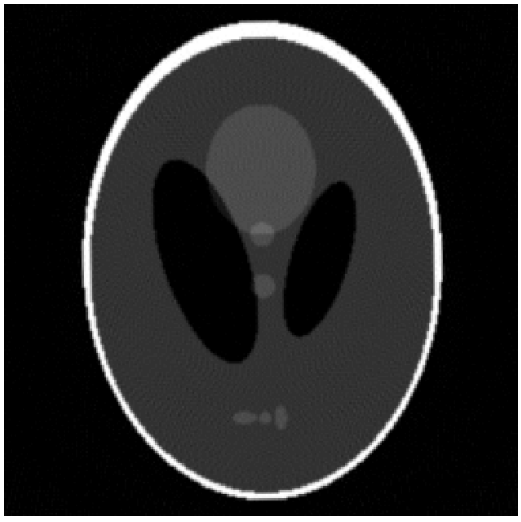


Figure 2: Reconstructed image via our function.

[1] Ω is set to 0.5.

[2] The matrix $g_{j,l}$ of the first step is given by the Radon transform.

[3] To observe this, I encourage the reader to check the dimensions of the matrices in the script.

[4] An important consideration is needed here in order to not get out of bounds on the matrix h. Remember that $k = -q, \ldots, q$. Hence, we need to sum $q+1$ to k so that k goes from 1 to $2q+1$.