In conclusion, inspection of the right singular vectors gives a lot of information about the reconstruction. The singular vectors in the null space represent structures that we cannot reconstruct, and the singular vectors outside the null space represent the structures that we can reconstruct.

Exercises

If you use the MATLAB `svd` function in these exercises, then please be careful not to create matrices that are too large, since the SVD computations may be very time-consuming. Start with small image dimensions, say, 32×32, and determine the computing time; then scale the problem, if desired, keeping in mind that the `svd` computing time is $O(mn^2)$ for an $m \times$n system matrix $\boldsymbol{A}$. You will need to use `svd` in Exercises 10.5 to 10.7.

Alternatively, you may use the MATLAB `svds` function, which can be used to compute a partial SVD, i.e., a smaller number of singular values and vectors. This is often sufficient to get an understanding of the behavior of the SVD. You will use `svds` in the remaining exercises.

Please keep in mind that for small image dimensions the computed singular vectors—especially the right singular vectors $\boldsymbol{v}_i$—will not appear as nice and smooth as the ones in the figures of this chapter. This is not due to errors in the discretization (or rounding errors on the computer); it is due to the unavoidable approximation errors associated with creating a discretized problem.

10.1. **SVD Analysis of the Radon Transform**
The goal of this exercise is to demonstrate how we can use the SVD to numerically compute approximations to the singular values and singular functions of the Radon transform. As explained in Section 10.4, we need to work with a disk domain instead of the square image domain that is typical in CT. To aid this, we provide three MATLAB functions:

- `MatrixToDiskDomain` removes columns of the matrix `A` corresponding to pixels outside the disk.
- `VectorToDiskDomain` removes elements of the "long" vector `x` corresponding to pixels outside the disk.
- `VectorToRectDomain` transforms a "short" vector `x` (in the disk domain) to a square image that can be displayed; elements outside the disk are set to `NaN`.

We suggest the following code, where the function `paralleltomo` is from AIR Tools II:

```
N = 100;                          % Image dimension.
theta = 4:4:360;                  % Projection angles.
```

```
A = paralleltomo(N,theta,N);     % N detector pixels cover
                                 %    the disk domain.
A = MatrixToDiskDomain(A);       % Remove columns.
[U,S,V] = svds(A,100);           % Compute partial SVD.
```

We suggest that you use `svds(A,100)` to compute the first 100 SVD components. Then plot the first 100 singular values σ_i by means of `semilogy(diag(S))`. These singular values should behave like the analytic singular values for the Radon transform.

Show some of the singular vectors as images, e.g., by means of the following code:

```
imagesc( VectorToRectDomain(V(:,i),N) )
imagesc( reshape(U(:,i),p,ntheta) )
```

You should see a resemblance with the analytic singular functions from Chapter 7.

Depending on your curiosity and patience, you may want to try larger image sizes, more projection angles, and more detector elements.

10.2. **SVD Analysis with a Square Domain**
Repeat the above experiments, but this time use a square domain—i.e., skip the transformation to the disk domain using the `MatrixToDiskDomain` function. Do you see any significant change in the singular values and vectors?

10.3. **SVD Analysis of a Fan-Beam CT Problem**
You should also compute the partial SVD of the system matrix for a CT problem with fan-beam geometry and a linear detector. For example, you can use the following code:

```
N = 100;
theta = 4:4:360;
A = fanlineartomo(N,theta);
A = MatrixToDiskDomain(A);
```

Here the function `fanlineartomo` is from AIR Tools II. You should see that the singular values and vectors of this matrix are not so different from those for the parallel-beam problem.

10.4. **SVD Analysis of Limited-Angle Problems**
As explained in Chapter 8, there are certain applications where we cannot measure projections for all angles; in this case we face a limited-angle problem. An SVD analysis of such problems gives important insight into the properties of the reconstructions that we can expect, thus complementing the analytic results.

Repeat the experiment from Exercise 10.1, but this time with a limited angular range $\theta \in [0^\circ, 90^\circ] \cup [180^\circ, 270^\circ]$ generated as follows:

```
theta = 2:2:90;
theta = [theta,theta+180];
```

You should see a striking difference in the right singular vectors for this problem. Describe what you see, and also describe how this will influence the reconstructions for this problem.

What will happen to the singular vectors if you shift the angular range, e.g., by setting `theta = theta + 45`? Try to predict the change before actually computing the corresponding singular vectors.

10.5. **SVD Coefficients and TSVD Solutions**
Although we do not encourage the use of the SVD for practical solution of CT problems, due to their large size, it is still instructive to perform an SVD analysis of small systems $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$ in order to get a feeling for the problem. We return to the test problems from Exercise 10.1 and/or 10.2 (it's your choice), and we will now use the SVD to analyze the reconstruction problem for two different phantoms:

- the Shepp–Logan phantom (see Section 5.4.2), which can be generated by `phantom(N)` or `phantomgallery('shepplogan',N)`, the former from MATLAB and the latter from AIR Tools II, and
- a smooth phantom, generated by `phantomgallery('smooth',N)`.

In both cases you must use `VectorToDiskDomain` to convert the square phantom images to the disk domain, if you decide to work on that domain. Then compute the corresponding data (the sinogram) `b = A*x`.

First, use the function `svd(full(A))` to compute the complete SVD of the system matrix—this will take several minutes, but we need many SVD coefficients in this exercise. Since you need to compute the complete SVD, we suggest you use a small image size, say, 32×32.

For both phantoms, compute the SVD coefficients of the image $\boldsymbol{x}$ and the data $\boldsymbol{b}$. You can compute the vector of all SVD coefficients of the right-hand side by means of `beta = U'*b`. Plot their absolute values by means of `semilogy`; we do not need to see all the coefficients—the first 1000 are enough (if the SVD computing time is too large, then just compute fewer components). Note that the coefficients level off due to discretization effects. It is possible (but beyond the scope of this book) to show that the coefficients for the smooth phantom decay faster, initially, than those for the Shepp–Logan phantom; can you confirm this with your experiment?

Now try to compute some TSVD reconstructions $\boldsymbol{x}_k$ for different truncation parameters k. We expect that we will need fewer SVD components for the smooth phantom than for the Shepp–Logan phantom. For each phantom, how many SVD components are needed to compute a reasonable reconstruction?

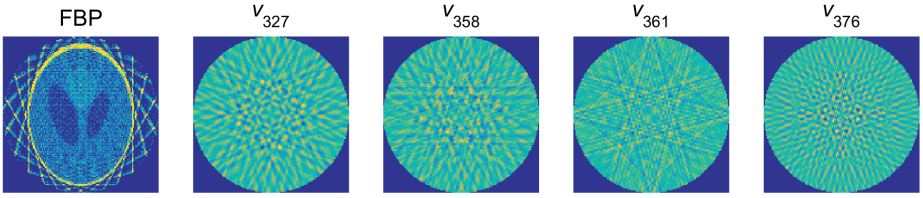

Figure 10.18. *Left: The FBP reconstruction for a problem with only* 24 *projection angles* $15^\circ, 30^\circ, 45^\circ, \ldots, 360^\circ$ *showing distinct line artifacts. Right: Four selected singular vectors* $\boldsymbol{v}_i$ *in which we see the same line structures.*

10.6. **TSVD Solutions to the Limited-Angle Problem**
To continue the previous exercise, here we compute TSVD solutions for the limited-angle problem in Exercise 10.4, using the same two phantoms as before. Again you need to compute the complete SVD here by means of `svd(full(A))`. Try with both

```
theta = 2:2:90; theta = [theta,theta+180];
```

and

```
theta = 92:2:180; theta = [theta,theta+180]; ,
```

and comment on the appearance of the reconstructions.

10.7. **SVD Analysis of a Problem with Very Few Projections**
In this exercise we consider a problem where the projection angles cover the full range from 0° to 360°, but there are very few projection angles. For example, you can use these parameters:

```
N = 140; theta = 15:15:360; ,
```

and once again you should use `svd(full(A))` to compute the complete SVD. Reconstructions from so few projection data points have very distinct line artifacts, as shown in Figure 10.18. Your goal is to confirm that similar line structures are present in some of the right singular vectors $\boldsymbol{v}_i$; see the four examples in Figure 10.18. Not all vectors have this structure, so you must carefully and patiently monitor a number of these vectors.

10.8. **The Significance of the Null Space**
Figure 10.19 shows a very small problem with a 3×3 image and eight rays that are pairwise parallel (you can think of this situation as being recorded by two detector pixels and the object being rotated 0°, 45°, 90°, and 135°). The corresponding

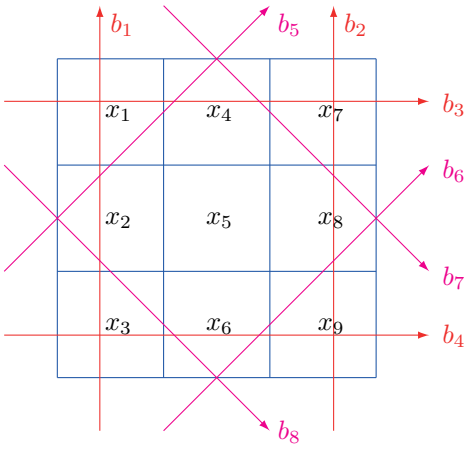

Figure 10.19. *A very small problem with a* 3×3 *image and eight rays.*

system matrix is

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ s & s & 0 & s & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & s & 0 & s & s \\ 0 & 0 & 0 & s & 0 & 0 & s & s & 0 \\ 0 & s & s & 0 & 0 & s & 0 & 0 & 0 \end{pmatrix}, \qquad s = 1/\sqrt{2}. \tag{10.30}$$

Consider an example where the ground truth image, represented as a vector, is

$$\overline{\boldsymbol{x}} = (1\ 2\ 3\ 2\ 3\ 4\ 3\ 4\ 5)^T. \tag{10.31}$$

Then compute the corresponding projection data $\boldsymbol{b} = \boldsymbol{A}\,\overline{\boldsymbol{x}}$.

Now compute the minimum-norm solution $\boldsymbol{x}^0_{\mathrm{LS}}$. For such a small problem, the easiest way to do this in MATLAB is to write `pinv(A)*b`, but don't do that for large problems. Can you reconstruct $\overline{\boldsymbol{x}}$ in (10.31)?

To explain the difficulty with reconstructing this $\overline{\boldsymbol{x}}$, consider the following questions: what is the rank of $\boldsymbol{A}$ and what is the null space $\mathrm{Null}(\boldsymbol{A})$? You can use the MATLAB functions `rank` and `null`. Can you propose a different $\overline{\boldsymbol{x}}$ that can be perfectly reconstructed?