SNARK14: A PROGRAMMING SYSTEM FOR THE RECONSTRUCTION OF 2D IMAGES FROM 1D PROJECTIONS

RAN DAVIDI, EDGAR GARDUÑO, GABOR T. HERMAN, OLIVER LANGTHALER, STUART W. ROWLAND, SAUMYA SARDANA AND ZE YE

October 29, 2017

Contents

List of Figures

Chapter 1

INTRODUCTION

"Just the place for a Snark!" the Bellman cried,
As he landed his crew with care;
Supporting each man on the top of the tide
By a finger entwined in his hair.

"Just the place for a Snark! I have said it twice:
That alone should encourage the crew.
Just the place for a Snark! I have said it thrice:
What I tell you three times is true."

Fit the First - The Landing
The Hunting of the Snark (an agony in eight fits)
by Lewis Carroll

In the area of image reconstruction, researchers often desire to compare two or more reconstruction techniques and assess their relative merits. SNARK14 provides a uniform framework in which the algorithms are implemented and their performance is evaluated. SNARK14 has been designed to treat both parallel and divergent projection geometries and can create test data for use by reconstruction algorithms. A number of frequently-used reconstruction algorithms are incorporated.

SNARK14 has been designed to be usable in many application areas and to be transportable to a wide variety of computers, in places at the expense of efficiency. (For example, SNARK14, as it is now distributed in a VirtualBox version -see, Subsection 1.2.1- will most likely run slightly slower than if it were installed specifically for a host system, but it will return the same outputs irrespective of the host computer, be it a pc, or a mac, or whatever.) While SNARK14 can be used to reconstruct repeatedly from data collected by a particular device, it is likely that one can design a more efficient special purpose program just for that device.

This manual of SNARK14 is written for the practitioner in the field of reconstruction. The novice will find it necessary to get acquainted first with one of the many tutorial works in the field, such as [29] or [48].

1.1 Statement of purpose and history

The reconstruction problem has arisen in a large number of scientific fields (including electron microscopy, radiology, radio astronomy and holography) and many different methods (algorithms) have been suggested

for its solution. The reader is referred to [29] for a survey of such algorithms and to [26, 44] for discussions on some of the application areas.

SNARK14 is a programming system designed to help researchers interested in developing and evaluating reconstruction algorithms. It is a descendant of earlier releases of SNARK, the first one of which was written by Richard Gordon in 1970.

SNARK77 [42] and SNARK89 [37] followed the original SNARK and were specifically designed to help with the problem of reconstructing cross-sections of the x-ray absorption coefficient distribution inside the body from x-ray projections. SNARK93 [10] extended this capability to include positron emission tomography, PET (where the problem is that of reconstructing cross-sections of isotope concentrations inside the body from γ-ray projections). A related programming system, specifically designed for emission studies, is RECLBL produced by Huesman et al. [45]. More recent relevant software packages are briefly surveyed in [48, Section 1.2]

The SNARK93 programming system was implemented in FORTRAN77. It was designed to

1. be capable of dealing with many modes of data collection (different geometrical arrangements of x-ray source and detectors, different x-ray spectra, etc.);
2. contain many of the published reconstruction algorithms;
3. be capable of generating mathematically described phantoms that realistically represent various cross-sections of the human body, together with mathematically simulated projection data of these cross-sections reflecting the characteristics (including noise) of various possible tomography devices;
4. contain subroutines to carry out work which appears to be common to many reconstruction algorithms, so as to ease the incorporation of additional (user-defined) algorithms;
5. be capable of a variety of display modes;
6. contain routines for the statistical evaluation of reconstruction algorithms;
7. provide a methodology for testing for statistically significant differences between reconstruction algorithms.

The successor of SNARK93 was SNARK05. The following were the major advances that were incorporated into the SNARK05 package:

1. SNARK05 was implemented in C++;
2. the file structures for holding projection data, phantoms and reconstructions had been redesigned to match the capabilities of the typical computer environment at the beginning of the third millennium (specifically, XML headers are used in data files);
3. all iterative algorithms were capable of handling image representations that use "blobs" [54, 56], as well as those that use "pixels";
4. the efficient data access ordering proposed in [38] was a standard feature of SNARK05;
5. all artificial restrictions on sizes were removed - the only remaining restrictions were those imposed by the hardware, compiler and operating system.

SNARK09 is an updated version of SNARK05. The following are the major advances that are incorporated into the SNARK09 package:

1. capability of generating mathematically described phantoms that include inhomogeneity;
2. capability of applying beam hardening correction to polychromatic x-ray projection data;
3. inclusion of the *imagewise-region-of-interest* figure of merit (FOM) [59];
4. handles up to 10 user-implemented algorithms, at most five using blobs;

5. the user can implement up to five user-defined FOMs.

SNARK14 is an updated version of SNARK09 and includes the following major updates:

1. inclusion of the superiorization methodology described in [21], see Section 5.10;
2. new stopping criteria: MLEM-STOP, Kullback-Leibler distance, residual, weighted squared distance, see Section 5.9;
3. new FOMs for SNARK EXPERIMENTER: Kullback-Leibler distance and weighted squared distance, see Subsections 8.2.6.5 and 8.2.6.6;
4. addition of the SART algorithm, see Section 7.7.

In addition to changes to the ability of the software package (as enumerated above), a major change has been made to its transportability. While SNARK09 was designed to run as a Linux/Unix program, the complete SNARK14 can be downloaded as an independent package onto any computer system that has a VirtualBox capability. This makes SNARK14 essentially universally available. The next section is devoted to describing how SNARK14 can be installed on just about any current computer system.

1.2 Installation of SNARK14

1.2.1 Downloading instructions

Instructions for acquiring SNARK14 can be found at

http://turing.iimas.unam.mx/SNARK14M/

This distribution includes the complete source code for SNARK14.

We are distributing a VirtualBox version of SNARK14. To download the VirtualBox platform package that is appropriate for your operating system, go to

https://www.virtualbox.org/wiki/Downloads

To use SNARK14, you need to download and save the file CentOSwithsnark.zip. You can initiate this process by visting

http://turing.iimas.unam.mx/SNARK14M/CentOSwithsnark.zip (Location 1 - North America)
http://snark.cappatec.com/CentOSwithsnark.zip (Location 2 - Europe)

After you downloaded the zip file, you need to unzip it. (Note that some unzipping tools may not be compatible with the provided zip file; in case of problems, a different tool should be used for the extraction.) After this you should have in the directory in which you unzipped CentOSwithsnark.zip access to two additional files: **CentOS.vmdk** and **CentOS.vbox**.

Clicking on **CentOS.vbox** should result in generating a Virtual Machine named **CentOS+SNARK14**.

You should now initiate this (for example, by clicking on its name in the Oracle VM VirtualBox Manager). After this you may receive some warning/error message that should be ignored (for example, by clicking on OK whenever that option is available). The same instruction is valid regarding warning/error messages that occur as a result of doing steps that are recommended below in this section.

This should result in a window (we refer to it below as the virtual machine) with a header such as:

`CentOS+SNARK14(Running)-Oracle VM VirtualBox.`

The window was configured to occupy a large region of the screen; however, you may wish to adjust the size of the display panel in the virtual machine by clicking on System, Preferences, Display and selecting a new Resolution.

You should now be able to run SNARK14 (by opening a terminal in the virtual machine and typing snark14 in that terminal) and also its Graphical User Interface utilities (snark14Input and snark14Display); for more details on these, see Sections 1.4.1 and 1.4.2.

1.2.2 Validation of downloaded software

If SNARK14 is correctly installed, then there should be a snark14Examples folder in the Desktop directory.

To check that SNARK14 is installed properly, you need to open a terminal in the virtual machine and type the following:

```
cd ~/Desktop/snark14Examples
./run_all
./regression
```

For a proper installation, SNARK14 should report "No differences found" for each of the examples b1 to b11.

1.2.3 Sharing files between the host system and the virtual machine

A directory in the virtual machine can be shared with the host operating system. The method for setting up such a shared directory is as follows.

1. Select “Shared Folders” under the menu Devices of the virtual machine.
2. A window called **CentOS+SNARK14 - Settings** should pop up with the header Shared Folders.
3. There is an option of “Add Shared Folder” (on some systems this is indicated by a green plus sign) in the above window. Clicking on the icon for that results in the opening of a new window with the header “Add Share.”
4. Now you have to specify the Folder Path and the Folder Name. The Folder Path is that to the to-be-shared directory on the host computer and the Folder Name refers to the name to be assigned to the shared folder in the virtual machine.
5. Also, there are options present for Read-only, Auto-mount, and Make Permanent. We suggest that you enable the options for Auto-Mount and Make Permanent, so that this process need not be repeated every time the VirtualBox is started.
6. Restart the system by rebooting the Virtual Machine. You can do this by executing the `reboot` command in a terminal.
7. Now, to check whether or not the shared directory is created, open the “Computer” directory present on desktop of virtual machine and open “media.” The shared folder must be present in this directory with the name as: sf_name, where name is that of the shared folder specified above in virtual machine.
8. A link of this shared directory can be created on the desktop of virtual machine. This can be done by using a terminal. Open a terminal, and type the following command:

   ```
   ln -s /media/sf_name ~/Desktop/Name
   ```

Here, `Name` is the name of the folder that will be created on the desktop of virtual machine that is linked with the shared folder present in media.

1.3 Running SNARK14

If SNARK14 was downloaded according to the instructions provided in Section 1.2.1, a SNARK14 run can be initiated by opening a terminal in the virtual machine, going to a desired directory and typing the following at the command prompt:

```
snark14 input.in >output.out
```

As a result of this, SNARK14 will read the INPUT from the file input.in in the current directory and will save the OUTPUT in the file output.out in the current directory.

If `input.in` is missing, then SNARK14 will expect the INPUT to be entered as the standard input, which typically means entering the file INPUT in the terminal line-by-line. Each command will be executed line-by-line after it has been completely entered. As soon as the command END is entered, the SNARK14 run will be terminated.

If `>output.out` is missing, then SNARK14 will use the standard output for OUTPUT, which typically means that the output will appear in the terminal.

All other files that will be created during a typical SNARK14 run (such as file11, prjfil, recfil, punch and eval) are saved in the current directory. If a file with the same name already exists in that directory, then it will be overwritten. It is therefore important to note that, if these files are to be saved, either rename them and/or move them into another directory prior to the next SNARK14 run.

1.4 Interactive interfaces

To ease the use of SNARK14, interactive interfaces have been designed for making and running a SNARK14 INPUT file and for observing the results of a SNARK14 run as images and/or as graphs. We now provide pointers to these.

1.4.1 Graphical interface for creating and running a SNARK14 INPUT

This interactive graphical interface, called snark14Input, can be used to produce and/or run an INPUT file for a SNARK14 run. It can be initiated by opening a terminal in the virtual machine, going to a desired directory and typing in the command prompt:

snark14Input

Alternatively, snark14Input can be initiated by clicking on its icon on the Desktop of the virtual machine.

1.4.2 Graphical interface for displaying the results of a SNARK14 run

This interactive graphical interface, called snark14Display, allows the user to display the results of a SNARK14 run. It can be initiated by opening a terminal in the virtual machine, going to a desired directory and typing in the command prompt:

snark14Display

snark14Display has two primary functions. One is to read images from prjfil and recfil files and display them as gray-level intensity maps, and plot their row/column profiles. The other is to plot results stored in an eval file.

Alternatively, snark14Display can be initiated by clicking on its icon on the Desktop of the virtual machine.

1.5 Outline

This manual has eight chapters, whose topics are as follows:

1. this introduction;
2. a discussion of the physical meaning of the concepts used in SNARK14;
3. an introduction to the information flow and the files used by SNARK14;

4. a description of the notational and format conventions;
5. a description of the SNARK14 language;
6. a description of the method of implementing user-defined algorithms and the support routines available;
7. a description of the use of the SNARK14 built-in reconstruction algorithms;
8. a description of SNARK experimenter, a facility for testing for statistically significant differences between reconstruction algorithms.

In addition, there are four appendices: the first provides detailed examples of the use of SNARK14, the second provides information on how a user may enter the projection data to be used by the reconstruction algorithms, the third describes the structure of two essential files produced by SNARK14 (one stores the projection data and the other stores a phantom and/or reconstructions), and the fourth describes the location of the source code in the distribution and the usage of utility scripts that are part of the SNARK14 package.

1.6 Acknowledgments

In addition to the authors of this manual, a major contributor to SNARK09 was Joanna Klukowska [48] and a major contributor to the transition from SNARK09 to SNARK14 was Bernhard Prommegger.

SNARK09 was an adaptation from and improvement upon SNARK05 that was a product of the Discrete Imaging and Graphics Group in the Department of Computer Science, The Graduate Center of the City University of New York, written by Bruno Carvalho, Wei Chen, Joel Dubowy, Gabor T. Herman, Miroslaw Kalinowski, Hstau Y. Liao, Lajos Rodek, László Ruskó, Stuart W. Rowland and Eilat Vardi-Gonen. In producing SNARK05, we had the help and advice of our colleagues in the Ph.D. Program in Computer Science at the Graduate Center, CUNY, especially Edgar Garduño, Xingjian Li, Roberto Marabini, Deniz Sarioz, and Gary Weng.

SNARK05 was an adaptation from SNARK93 that was a product of the Medical Image Processing Group in the Department of Radiology, University of Pennsylvania, written by Jolyon A. Browne, Gabor T. Herman, and Dewey Odhner. Its production was helped by many coworkers, including Lyle Berkowitz, Alvaro De Pierro, Paul Edholm, Robert M. Lewitt, David Roberts and Jingsheng Zheng.

The previously released SNARK77 (written by Gabor T. Herman and Stuart W. Rowland) and SNARK89 (written by Gabor T. Herman, Robert M. Lewitt, Dewey Odhner, and Stuart W. Rowland) were also products of the Medical Image Processing Group, originally in the Department of Computer Science at the State University of New York at Buffalo and then in the Department of Radiology, University of Pennsylvania. Contributors to their development include Martin Altschuler, Ehud Artzy, Yair Censor, Tao Chang, Paul Eggermont, Gideon Frieder, James Hinds, Julieann Kostyo, Yuan-hern Kuo, A.V. Lakshminarayanan, Arnold Lent, Hsun-kao Liu, Shirley Long, Hsi-ping Lung, Peter Lutz, Roger Peretti, Seshagiri Sadananda, James Turner, Mann-may Yau, and Daniel Yeung.

The production of SNARK77 was supported mainly by NCI contract CB 53860, but also by NIH grants HL 18968 and HL 4664 and NSF grant MCS 75-22347. Interaction with the Principal Investigators of related NCI contracts (Richard Robb of the Mayo Clinic and Thomas Budinger of the University of California, Berkeley) was also helpful. The adaptations resulting in SNARK89 and SNARK93 were supported by NIH grant HL 28438. The research work of the authors of SNARK05 and SNARK09 were supported by NIH grant HL 70472.

There are many packages available for reconstruction from projections, some of them are briefly surveyed in Section 1.2 of [48]. Here we single out for mention jSNARK [1] that incorporates a large subset of the capabilities of SNARK14 for reconstructing 2D objects from their 1D projections, but it also extends those capabilities to reconstructing 3D objects from their 2D projections. It is written in Java allowing for greater platform flexibility and for user-defined routines written as plugins.

Chapter 2

THE SNARK14 FRAMEWORK

This chapter is written to acquaint the user with the physical meaning of the concepts used in SNARK14.

2.1 Units and constants in SNARK14

SNARK14 deals with physically measurable quantities, such as the distance between the x-ray source and detector or the linear attenuation (per unit length) at a point inside an object. The actual values assigned to these quantities are dependent on the assumed *unit of length*. While the unit of length does not have to be specified for SNARK14 (in fact, there is no facility for its specification) consistency must be maintained throughout a single SNARK14 run.

In particular, we shall assume a fixed two-dimensional coordinate system. The point (x, y) in this system represents the physical location whose abscissa is x and ordinate is y, as measured in the assumed unit of length.

There are two particular constants, called ZERO and INFIN, which are defined by SNARK14 and whose meanings need to be clearly understood (especially since their names are potentially misleading). ZERO is a very small positive floating point number (10^{-20} to be exact) and INFIN is a very large positive floating point number (10^{20} to be exact). These values are referred to below (e.g., SNARK14 requires that some input items must have a value greater than ZERO, rather than just be positive). Also, they are available to those users who wish to create their own routines; they can, in particular, be used for checking for underflow and overflow conditions. Other constants that are available to the user of SNARK14 to a high accuracy by referring to them by their names are PI (π), TWOPI (2π), PISQ (π^2), PID2 ($\pi/2$), SQRT3 ($\sqrt{3}$) and ISQRT3 ($1/\sqrt{3}$). The actual naming of these constants in user-written code is explained in Subsection 6.3.2.

2.2 Pictures and digitization

SNARK14 deals with "pictures" defined over the 2D plane of points (x, y) in some assumed fixed coordinate system. To be exact, a *picture* has two components:

1. the *picture region*, which is a square whose center is at the origin of the coordinate system and whose sides are parallel to its axes;

2. a *function* of two variables whose value is zero outside the picture region.

Sometimes, when this is not confusing, we shall call the function as the "picture." However, identical functions may give rise to different pictures if the picture regions are different.

We shall often refer to the value $f(x, y)$ of the picture f at the point (x, y) as the *density* of f at (x, y). Within SNARK14, $f(x, y)$ is often specified by a *grid* G (which is a finite set $\{(g_1, h_1), \cdots, (g_J, h_J)\}$ of points in the plane), a *basic basis function* b (which is just a function of two variables), and a *coefficient* c_j associated with each grid point (g_j, h_j) as follows. For $1 \leq j \leq J$, each of the functions

$$b_j(x, y) = b(x - g_j,\, y - h_j) \tag{2.1}$$

is called a *basis function* and f is defined by

$$f(x,y) = \sum_{j=1}^{J} c_j b_j(x,y); \tag{2.2}$$

i.e., as an expansion into the basis functions b_j with coefficients c_j. SNARK14 allows the use of two different kinds of basis functions: pixels and blobs, each with its own type of grid.

The exact definition of a *pixel basis function* depends on a variable called PIXSIZ that is specified by the input to SNARK14. (How PIXSIZ, and many other variables, are specified by the input is explained in Section 5.3.) The *basic pixel basis function* is then defined to have the value 1 at points strictly inside the square that is centered at the origin and that has edges of length PIXSIZ parallel to the coordinate axes, and to have the value 0 at points strictly outside this square. (While this does not make any difference in any possible application of SNARK14, for completeness we can define the basic pixel basis function to have the value 0.5 on the edges and the value 0.25 at the corners of the square.) The associated grid G is defined, by an additional input-specified variable called NELEM (which is restricted to be an odd positive integer), to be the set

$$\{(m \times \text{PIXSIZ},\, n \times \text{PIXSIZ}) \,|\, m \text{ and } n \text{ integers},\ \max\{|m|, |n|\} \le \text{NELEM}/2\}, \tag{2.3}$$

where $|\cdot|$ denotes the absolute value. This approach subdivides the picture region into NELEM^2 equal squares. Each of these smaller squares is called a *pixel* (short for picture element). In the interior of a pixel, the density of the function, as defined by (2.2), is uniform. An arbitrary picture can be approximated by such an expansion by simply assigning to each c_j the average density of the picture in the corresponding pixel; such an approximation is referred to as the NELEM^2 *digitization* of the picture. In SNARK14, the picture region (which is sometimes referred to as the *reconstruction region*) is determined by the program (based on the input-specified variables PIXSIZ and NELEM) as the square whose corners have coordinates (c,c), $(-c,c)$, $(-c,-c)$, $(c,-c)$, where

$$c = \frac{\text{PIXSIZ} \times \text{NELEM}}{2}. \tag{2.4}$$

A *blob basis function* also depends on some input-specified variables. A *basic blob basis function* is a generalization of a well-known class of window functions in digital signal processing called *Kaiser-Bessel* [54]; it is circularly symmetric, has nonzero values only in a circular disk around the origin, and smoothly decreases from a positive value at the origin to zero at the edge of the disk. The exact definition depends on three input-specified variables (SUPPORT and DELTA that have to be greater than ZERO, and SHAPE that has to be nonnegative) as follows:

$$b(x,y) = \begin{cases} C \times I_2\left(\text{SHAPE} \times \sqrt{1-r^2}\right) \times \left(1-r^2\right), & \text{if } 0 \le r \le 1, \\ 0, & \text{otherwise,} \end{cases} \tag{2.5}$$

where I_i denotes the so-called modified Bessel function of the first kind of order i, $r = \frac{\sqrt{x^2+y^2}}{\text{SUPPORT}}$ and $C = \frac{\sqrt{3} \times \text{DELTA}^2 \times \text{SHAPE}}{4\pi \times \text{SUPPORT}^2 \times I_3(\text{SHAPE})}$. The grid G that determines the blob basis functions using (2.1) is hexagonal [28]; it is defined as the set of all those points in the set

$$\left\{\left(\frac{m}{2} \times \text{DELTA}, \frac{\sqrt{3}n}{2} \times \text{DELTA}\right) \,|\, m \text{ and } n \text{ integers and } m+n \text{ even}\right\} \tag{2.6}$$

that are also inside the reconstruction region specified by (2.4).

It is not unusual that one desires to reconstruct only part of a picture. This can be achieved in SNARK14 by specifying a reconstruction region which is smaller than the region of the picture from which data have been collected. Some reconstruction methods can handle such a situation, others will give erroneous results. It is the user's responsibility to choose the correct reconstruction algorithm.

2.3 Rays and ray sums

There are two kinds of *rays* in SNARK14:

1. a LINE ray, which is a straight line, and
2. a STRIP ray, which is a region of the plane between a pair of parallel straight lines.

Given a picture and a ray, the *real ray sum* is the integral of the picture along the ray. (This may be a line integral or a strip integral.) In this terminology, the *reconstruction problem* that is treated by SNARK14may roughly be stated as follows: **given** approximations (based on physical measurements) of the real ray sums of a picture for a number of rays, **estimate** the NELEM^2 digitization of the picture.

SNARK14 also deals with *pseudo ray sums*; these are defined only for expansions of the form (2.2). In the LINE case, the pseudo ray sum is the real ray sum of the picture defined by the function f of (2.2) and a picture region large enough to contain all points at which the value of f is not zero. (Since G is finite, it is always possible to find such a picture region.) In the STRIP case, the pseudo ray sum is defined as

$$\left(\sum_{j\in S} c_j\right) \times \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} b(x,y)dxdy, \tag{2.7}$$

where S contains exactly those j $(1 \le j \le J)$ for which (g_j, h_j) is in the strip. Note that the integral in the above equation depends only on the basic basis function; it is PIXSIZ^2 in the pixel case.

2.4 Geometry of data collection

SNARK14 is not capable of handling an arbitrary arrangement of rays, but it can handle a number of arrangements of rays that are typical of what one might come across in practice.

First of all, the set of all rays along which data are collected is divided into a number of subsets, called *projections*, each one of which contains the same number of rays. (PRJNUM denotes the number of projections and USRAYS denotes the number of rays along which data have been collected in any one projection. In SNARK14, USRAYS must be an odd number.) There are two basically different modes of data collection: DIVERGENT and PARALLEL.

In DIVERGENT geometry (Figure 2.1), a projection consists of a set of LINE rays that go through a common point (the *source position*). In all the projections, the source is at a fixed distance (RADIUS) from the origin. The angle between the line from origin to source and the x-axis (marked THETA in Figure 2.1) is called the *projection angle*. The rays in one projection connect the source to points (*detectors*) which lie either on an arc of a circle whose center is at the source or on a straight line. In either case, one of the detectors (marked C in Figure 2.1) lies on the line connecting the source to the origin, at a distance STOD from the source. The other detectors are spaced symmetrically at equal intervals on the two sides of C, either on the ARC whose center is the source or on the TANGENT line to this ARC at C. The spacing between detectors (the length of the ARC or that of the TANGENT line between two neighboring detectors) is denoted by PINC (specified, as usual, by the input to SNARK14).

In PARALLEL geometry (Figure 2.2), a projection consists of a set of parallel LINE or STRIP rays. The angle these rays make with the x-axis (denoted by THETA in Figure 2.2) is called the projection angle. If we consider the LINE case, one of the rays goes through the origin and if the STRIP case is considered, the origin is equidistant to the two lines bounding one of the strips. The other rays are spaced symmetrically at equal intervals on the two sides of this ray. In the STRIP case, the rays are abutting. Let d denote the distance between the rays in the LINE case or the width of the rays in the STRIP case; it is determined as follows. The input specifies the variable PINC and also whether the *ray spacing* is to be UNIFORM or VARIABLE. In the UNIFORM case, d = PINC, for all projections. In the VARIABLE case it depends on the projection angle THETA: d = PINC $\times$ max $\{|\sin \text{THETA}|, |\cos \text{THETA}|\}$. (A consequence of this definition is that the distance between two consecutive intercepts with either the x- or the y-axis will be PINC.)

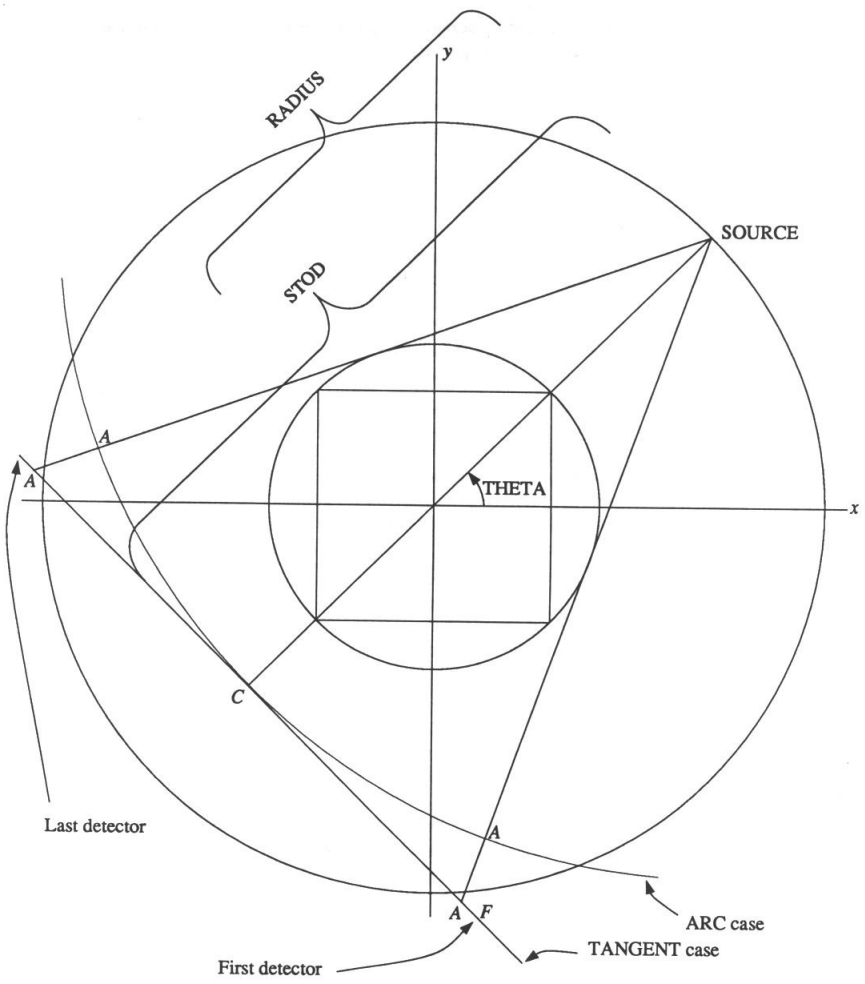

Figure 2.1: SNARK DIVERGENT geometry schematic.

2.5 Number of rays in a projection

As has been discussed in the last section, in SNARK14 the variable USRAYS denotes the number of rays along which data have been collected in any one projection. Since the value of USRAYS and the values of NELEM and PIXSIZ (see Section 2.2) are independently defined, it is possible that many of the rays along which data have been collected miss the reconstruction region, or that only a small part of the reconstruction region is covered by the rays along which data have been collected. We introduce the variable SNRAYS to denote the minimum number of rays needed to cover the reconstruction region in all projections. SNRAYS is calculated by SNARK14 based on the input-supplied values of NELEM, PIXSIZ, PINC, etc., according to the following method.

Let D be the distance along the detector strip between the center point of the detector strip (marked C in Figures 2.1 and 2.2) and the points on the detector strip where the tangents from the source to the smallest circle that contains the reconstruction region meet the detector strip (marked A in Figures 2.1 and 2.2). Let e be PINC, except in the PARALLEL VARIABLE case in which $e = \text{PINC}/\sqrt{2}$. Let $\lceil x \rceil$ denote the smallest integer that is not smaller than the real number x. Then $\text{SNRAYS} = 2 \times \lceil \frac{D}{e} \rceil + 1$.

Just like USRAYS, SNRAYS is an odd integer. Either of these two variables may have a value greater than the other: we use the variable NRAYS to denote their maximum.

In SNARK14 reconstructions the rays are referred to by a double index (NP, NR) where $0 \leq \text{NP} <$ PRJNUM and $0 \leq \text{NR} <$ NRAYS. The value of NP is the *projection number*, it is determined by the order

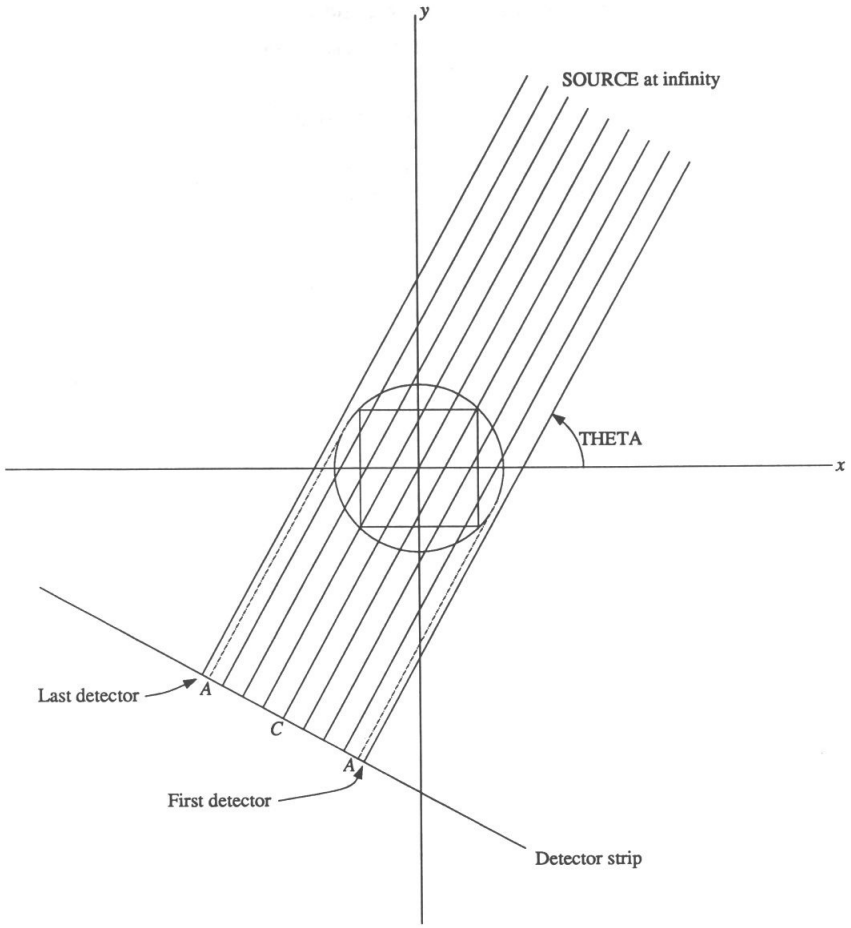

Figure 2.2: SNARK14 PARALLEL geometry schematic.

in which the projection angles have been specified. (There is no restriction on the projection angles having to be specified in increasing, decreasing or any other order.) The value of NR is the *ray number*. The rays are assumed to be numbered clockwise on the detector strip relative to the origin (see Figures 2.1 and 2.2). The ray numbered (NRAYS−1)/2 goes through the origin.

If USRAYS < SNRAYS = NRAYS, then SNARK14 assumes that the value of the real ray sum for a ray along which data have not been collected is zero. In order to give the user the ability to access only the central USRAYS or SNRAYS, SNARK14 evaluates four further variables: FUSRAY, LUSRAY, FSNRAY and LSNRAY, defined as follows:

$$\text{FUSRAY} = \frac{\text{NRAYS} - \text{USRAYS}}{2}, \tag{2.8}$$

$$\text{LUSRAY} = \frac{\text{NRAYS} + \text{USRAYS}}{2} - 1, \tag{2.9}$$

$$\text{FSNRAY} = \frac{\text{NRAYS} - \text{SNRAYS}}{2}, \tag{2.10}$$

$$\text{LSNRAY} = \frac{\text{NRAYS} + \text{SNRAYS}}{2} - 1. \tag{2.11}$$

2.6 SNARK14 input/output

The following summarizes the input requirements of SNARK14 that we have discussed and some additional input and output possibilities. An exhaustive description is given in the following chapters.

Typical inputs to SNARK14:

- Geometry information
  - Number of elements in a row or column of the pixel grid (NELEM)
  - Pixel size (PIXSIZ)
  - Detector spacing (PINC)
  - PARALLEL or DIVERGENT geometry
  - If PARALLEL, then the input also specifies STRIP or LINE integrals, number of projections (PRJNUM), and UNIFORM or VARIABLE detector spacing
  - If DIVERGENT, then the input also specifies TANGENT or ARC, source-to-origin distance (RADIUS) and source-to-detector distance (STOD)
  - Environment for data collection: e.g., noise (quantum, scatter, etc.)
  - Number of projections (PRJNUM)
  - List of projection angles
- Test picture (optional)
  - This is either taken from a previous SNARK14 run or is specified by a description in terms of geometrical figures
- Projection data
  - Gathered experimentally or generated by SNARK14 (from a test picture)

Typical outputs from SNARK14:

- Comparison of reconstructions with the test picture with respect to average density, relative error, variance, standard deviation, and residual
- A file containing decimal values of the test picture, if present, and selected reconstructions
- A comparison of the exact values of a test picture and a reconstruction for every row of the object in selected columns
- The CPU time used for each command and each iteration of an algorithm

The following chapters describe in detail the required structure of the input/output commands, their associated interpretation and expected results. The reader is also referred to Appendix A, where examples of input files and the corresponding outputs are presented.

Chapter 3

SNARK14 INFORMATION FLOW AND FILES

A SNARK14 run consists of up to three phases:

1. data generation phase;
2. initialization and reconstruction phase;
3. analysis phase.

Each of these phases requires some input files and produces some output files. Some of the output files are used as the input files of a later (or even the same) phase. Provided that the appropriate input files are available, a single SNARK14 run may consist of just one of the three phases, or either of two consecutive phases, or all three phases.

We now proceed with a description of each of the three phases together with the input files required and output files produced by the respective phases . The reader should consult Figure 3.1 for an overview. The following file descriptions are rather brief. With the exception of the INPUT file, all files are written (and subsequently accessed) by SNARK14, and the user need to be concerned only with their formats if involved in the process of extending the capabilities of SNARK14 by additional code.

Warning: The SNARK14 manual refers to files named INPUT and OUTPUT. (These names, as many other names in the manual, are for descriptive purposes only. Within the code these files are referred to by names that are more appropriate for the C++ programming language in which the SNARK14 program is written.) The three files referred to in Figure 3.1 and elsewhere in the manual as Data Generation Input file, Initialization and Reconstruction Input file, and Analysis Input file are in fact not three separate files, but rather three consecutive parts of the single file INPUT. Similarly, the Data Generation Output file, the Initialization and Reconstruction Output file, and the Analysis Output file are three consecutive sections of the single file OUTPUT. Distinctions between the different sections of the INPUT and OUTPUT files are made to simplify the description of what is read from INPUT and what is written to OUTPUT during the three phases of a SNARK14 run. INPUT is the standard input, OUTPUT is the standard output. They are usually redirected to disk files when the SNARK14 program is executed.

The last section of this chapter points out the availability (outside SNARK14) of some interactive applications that have been designed to ease the creation of the SNARK14 INPUT file and to observe the results of a SNARK14 run as images and/or as graphs.

3.1 Data generation phase

During this phase, SNARK14 generates a *test picture* (a *phantom*) and/or projection data of it. The *projection data* consist of real ray sums of the phantom, possibly contaminated by the different types of noise that one may come across in a device used for collecting data for reconstruction.

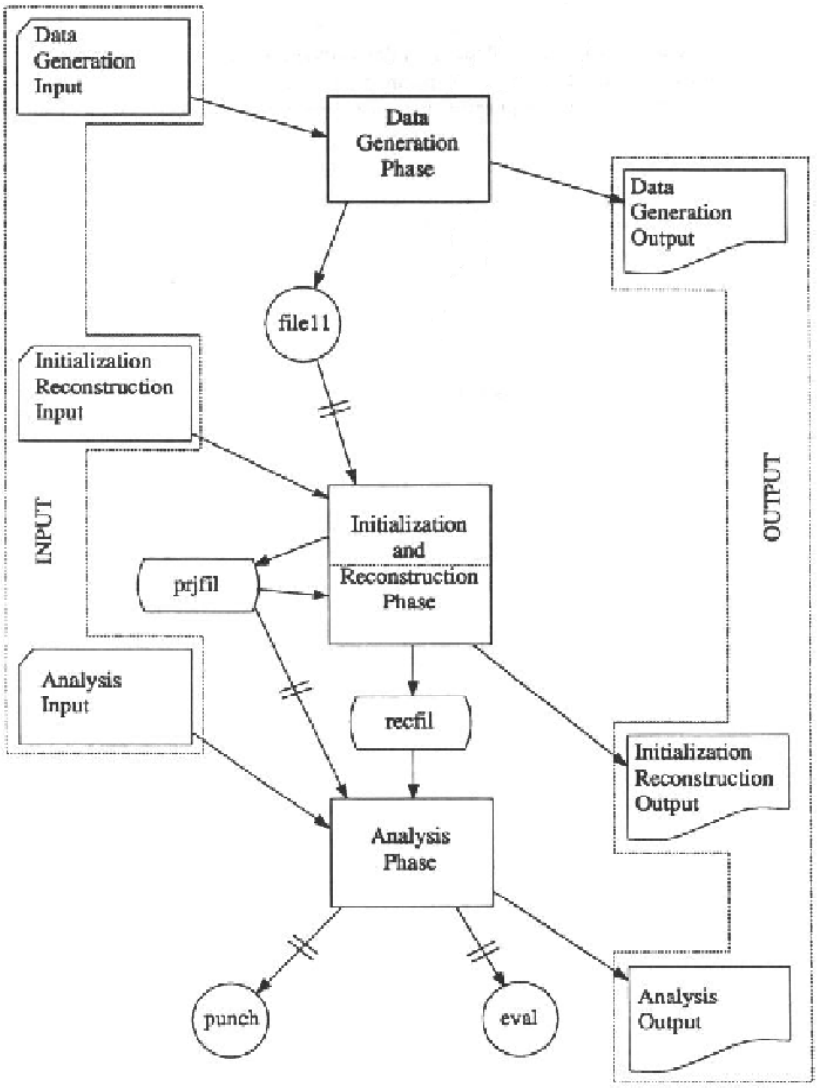

Figure 3.1: Information flow within SNARK14. Optional paths are marked by ||.

The *Data Generation Input file* is the section of the file INPUT that contains the commands that cause the generation of the test picture and the projection data. The SNARK14 commands permitted to appear in this section are TRACE, MODE, CREATE, and END (see Chapter 5).

file11 is a text file that contains the description of the phantom and/or its projection data. This description contains information on how the phantom and/or data have been specified, as well as the actual density values assigned to the pixels in the phantom and/or to the ray sums in the projection data. See Appendix B for instructions on how to create a file11 containing user supplied data.

The *Data Generation Output file* is the section of the file OUTPUT that echoes the data generation input file and reports on the work carried out by the data generation phase.

3.2 Initialization and reconstruction phase

The *Initialization and Reconstruction Input file* is the section of the file INPUT that contains the commands specifying where the initialization information is coming from and what reconstructions are to be carried out. The SNARK14 commands that are permitted to appear in this section are TRACE, MODE, PICTURE,

PROJECTION, SELECT, BASIS, STOP, EXECUTE, and END (see Chapter 5). The *Initialization and Reconstruction Output file* is the section of the file OUTPUT that echoes the initialization and reconstruction input file and reports on the work carried out by the initialization and reconstruction phase.

This phase can be broken up into two subphases: the initialization subphase and the reconstruction subphase. The latter cannot be performed without the former having been performed in the same run.

3.2.1 Initialization subphase

During this subphase the nature of the grid for the reconstruction region (NELEM and PIXSIZ) as well as the assumed geometry of data collection are determined. If there is a test picture, it will be read from file11 and put on recfil (see below). If the projection data set is not available on the file prjfil (see below), it is transferred onto this file.

prjfil is a file that contains a description of the geometry of data collection and, for every projection, the projection angle and the estimates of the ray sums in that projection from rays FUSRAY to LUSRAY (see Section 2.5). The structure of prjfil is discussed in Appendix C.

recfil is a file that contains a description of the phantom and/or the reconstructions performed during the reconstruction subphase. The structure of recfil is discussed in Appendix C.

3.2.2 Reconstruction subphase

During this subphase, the reconstruction algorithms specified by the initialization and reconstruction input file are carried out and the results are put on recfil.

If additional files are required for any user-written subroutine, they should be explicitly defined and opened by that subroutine (and closed when the work of the subroutine is done).

3.3 Analysis phase

During this phase various displays of the phantom and/or the reconstructions are generated based on the information in recfil. Statistical analysis of the results as well as comparisons of the reconstructions to the phantom may also be carried out.

The *Analysis Input file* is the section of the file INPUT that contains the commands specifying the displays, analysis, and comparisons to be generated. The SNARK14 commands permitted to appear in this section are TRACE, MODE, EVALUATE, DISPLAY, PUNCH, LINES, SKUNK, and END (see Chapter 5).

punch is a text file that contains descriptions of specified reconstructions in the format in which a phantom is described in file11. These may be used as a file11 in a later run.

eval is a text file that contains the evaluation of reconstructed images with respect to the phantom. This file can be read by a plotting program outside SNARK14 (see Subsection 1.4.2).

The *Analysis Output file* is the section of the file OUTPUT that echoes the analysis input file and reports on work carried out by the analysis phase, including details of the specified comparisons, analyzes, and displays.

Chapter 4

NOTATIONAL AND FORMAT CONVENTIONS

In this chapter we discuss the notational and format conventions for the file INPUT (that contains some, maybe all, of the data generation input file, initialization and reconstruction input file, and the analysis input file). The file INPUT is a sequence of lines. These are read one-by-one and, typically, they result in some action by SNARK14 before the next line is read. The INPUT file is logically divided into SNARK14 commands, each command occupying one or more lines in the input file.

4.1 Comment lines in the input file

Any line that begins with an * (an asterisk) in column 1 will be treated as a *comment*. It will be echoed in the OUTPUT, but it will otherwise be ignored by SNARK14. The only purpose of the comment lines is to make the files INPUT and OUTPUT more informative to humans who read them. Comments may appear almost anywhere in the SNARK14 INPUT file. The exception is where the input calls for a "name-of-..." command line (see 5.3.3 for an example), a line starting with an asterisk is treated as the name, not as a comment.

4.2 Commands and their levels

The part of the file INPUT that determines what gets computed during a SNARK14 run consists of the SNARK14 *commands*; each one of which occupies one or more (consecutive) lines in the input file.

A command starts with a command line in which the first continuous sequence of four letters identifies the type of command that is to be carried out. The exact details of what is to be done are specified by putting additional information on the same command line and, usually, on the lines immediately following it. Chapter 5 describes, for each of the fifteen commands, the precise contents and format in which the information has to be provided.

There are fifteen commands in SNARK14, which are classified into six levels as follows:

1. General (may appear in any one of the three input files mentioned above): TRACE, MODE.
2. Data generation command: CREATE.
3. Initialization commands: PICTURE, PROJECTION.
4. Reconstruction commands: SELECT, BASIS, SUPERIORIZE, STOP, EXECUTE.
5. Analysis commands: EVALUATE, DISPLAY, PUNCH, LINES, SKUNK.
6. Termination command: END.

There are some rules that must be observed regarding the order in which commands may appear in the INPUT file of a single SNARK14 run. Violation of these rules will result in termination of the run. To explain these rules we make use of the variable CL (the current level of the run) in addition to the variable L (the level of a command). The following are the rules:

- At the beginning of a run, $CL = 1$.
- Level 1 commands can occur (repeatedly) anywhere in the command sequence.
- If the level L of a command is greater than 1, then it must be the case that $CL \leq L$ and, after the execution of the command we will have $CL = L$.
- The CREATE, PICTURE, and PROJECTION commands may appear only once.
- An EXECUTE command must be preceded by a PROJECTION command and a PROJECTION command must be preceded by a PICTURE command.
- If the level of the first command in the sequence is 3, 4, or, 5, then the file specified below must be present in the current directory:
  - for 3, file11 is required if either PICTURE TEST (see Section 5.4) or PROJECTION REAL (see Section 5.5) is specified,
  - for 4, prjfil must be present,
  - for 5, recfil must be present and prjfil is required if residuals are computed.
- An END command must occur in the sequence.

With the exception of CREATE, PICTURE, and PROJECTION, all SNARK14 commands may occur repeatedly in the INPUT file of a SNARK14 run. The first occurrence of END will cause the termination of the run and further lines on the INPUT file will not be read. The SNARK14run will also be terminated if any rule regarding the contents of INPUT that is specified above or in the following chapters is violated.

4.3 Notations used in syntax

In describing the contents of the command lines which make up the commands we shall assume the following conventions.

SNARK14 *commands* will be identified by a greater than (>) symbol at the left margin followed by about one inch of white space.

Key words. If a capital word appears in the syntax, the first four letters of this word, or the whole word if its length is less than four, must appear together (without any blanks between them) at the corresponding position in the command. For words of length greater than four, the first four letters may optionally be followed by other symbols; all symbols up to the first blank following these letters will be ignored by SNARK14. In other words, in place of a long key word, the programmer may use any contiguous sequence of non-blank characters, provided only that the first four characters of this sequence coincide with the first four letters of the keyword. The keyword may be written on the INPUT file in either upper case or lower case or a mixture of both.

Lower-case words. Lower case words represent generic terms, values of which must be supplied by the user in the corresponding position in the command.

Braces. When words or phrases written underneath each other are enclosed in braces { }, a choice of one of the enclosed words or phrases must be made.

Brackets. When a word or phrase is enclosed in brackets [ ], the word or phrase is an optional feature of the instruction. Users may include the feature by coding according to the syntax enclosed in the brackets or they may omit the feature completely. When one of a number of optional features is permitted at a particular position, these are written underneath each other and are enclosed in brackets.

4.4 Control command line format

Some of the command lines in the INPUT file will be identified as *control command lines*. These are command lines which contain information in a free format, subject to some restrictions which we now describe.

Control command lines contain a sequence of three types of *modifiers*: word, integer and floating point. A modifier must be followed by a blank, unless it is the last modifier on the line. Additional blanks preceding or following the modifier are ignored.

A *word modifier* is any contiguous sequence of non-blank characters. Examples: CREATE, ART2, SMOOTH. In a word modifier all characters after the first four are ignored by SNARK14.

When SNARK14 is looking for a word modifier in a control command line, it ignores all characters and blanks until it comes across the first four characters of the sought-after word modifier or the whole of the word modifier followed by a blank or the end of the line. If it finds a word that is not one of the possible expected word modifiers, it skips that word and tries again. If a required word modifier is not found in the line, then the SNARK14 run is terminated.

In this manual, all word modifiers are written in upper case. Processing of the INPUT file, however, is case insensitive. Word modifiers may be in either upper or lower case or a mixture of them.

An *integer modifier* is a contiguous nonempty sequence of digits (0-9) of length between one and nine, possibly preceded by a minus sign (−). Examples −123456789, 003.

A *floating point modifier* is a string satisfying the regular expression:

$$['\text{-}'] \ \{\text{digit+} \ ['.' \ \text{digit*}] \ , \ '.' \ \text{digit+}\} \ [\ \{'\text{E}' \ , \ '\text{e}'\} \ ['+' \ , \ '\text{-}'] \ \text{digit} \ [\text{digit}] \ ]$$

where digit is one of the characters 0..9, 'x' is the literal character x, an asterisk (*) is zero or more repetitions of the preceding item, a plus (+) is one or more repetitions of the preceding item, brackets ([]) indicate optional items, braces ({}) indicate required items, and a comma (,) separates choices. In words, a floating point modifier is either any contiguous nonempty sequence of digits, possibly preceded by a minus sign, and possibly followed by a decimal point (.) followed by another sequence of digits, or it is a decimal point, possibly preceded by a minus sign followed by one or more digits. These may further be followed by the letter E or e, possibly a plus (+) or a minus (-) sign and one or two digits. Examples: 4, 0, 35.6E4, -25., 35e-64, .7071, 12345678912.12345678912.

When SNARK14 is looking for a number modifier in a control command line, it ignores all alphabetic characters (a to z and A to Z) as well as the equal sign (=) preceding the number modifier. This can be used to make the input file more readable, as will be illustrated later (e.g., in Section 5.2). If a required number modifier is not found in the line, then the SNARK14 run is terminated.

Every command in SNARK14 starts with a control command line, the first modifier of which is the word modifier indicating the command type.

4.5 Iteration-flag-line format

Some of the command lines in the INPUT file will be identified as an

> iteration-flag-line

Such a command line contains a string of digits 0 to 9 (a blank is assumed to be 0) in its first 51 columns. A command contains at most one iteration-flag-line. When describing the effect of the contents of the iteration-flag-line on the execution of the command, the following notation is used.

fl(0) is the digit in the first column of the iteration-flag-line.

For any positive integer q, fl(q) is the digit in column $t + 1$ of the iteration-flag-line, where $1 \leq t \leq 50$ and $q \equiv t \pmod{50}$. For example, fl(357) is the digit in column 8 of the iteration-flag-line.

An important use of the iteration-flag-line occurs in conjunction with the EVALUATE command (Section 5.11), it controls what gets written on the eval file (see Section 3.3) about the images that are produced after q iterations; for details, see Section 5.11.

4.6 Names used in the manual vs. names used in the code

This manual has been written with the desire (not always achieved) to be readable. For this reason, names are given to things in the manual that are not identical to the names for the same things in the code. (The names in the code reflect the structure of the program as it is written in C++; such names tend to be long and "funny looking" to ordinary mortals.) In the manual, variables are given names that usually consist of capital letters. (Many examples of this have appeared already: ZERO, PIXSIZ, NELEM, etc.) For those users who wish to write their own subroutines to be used in conjunction with SNARK14, precise instructions are given below at the appropriate places as to the names of the corresponding variables in the code.

Chapter 5

SNARK14 COMMANDS

In this chapter we describe the fifteen SNARK14 commands in detail.

5.1 TRACE

> TRACE [trace-level]

Because SNARK14 was primarily designed as a test-bed for algorithm development, for debugging purposes it is essential to have a built-in method of examining key parameter values during program execution. The TRACE command provides such a method. Because it is a general command, TRACE may appear anywhere in the SNARK14 INPUT file. Effective placement of this command should eliminate useless trace information.

The TRACE keyword is followed (optionally) by an integer that specifies the desired trace level. If an integer is not specified, the level is set to 0. The trace levels are defined by the following. (For a detailed description of the terms used below, the reader is referred to Chapter 6). If the trace-level is not positive, then the TRACE is off. Other trace-levels are interpreted as follow:

1. Provided so that the user can insert trace statements in a user specified algorithm and turn these debugging aids on and off. These statements should have the form
   if (trace $>$ 0) output . . .
2. Not used.
3. Not used.
4. Not used.
5. Prints projections (NP) and rays (NR) selected by PICK (6.4.1).
6. Prints the real ray sum (or pseudo ray sum) as well as NP and NR for the NP-th projection and NR-th ray used by the function PRDTA (6.3.6.2) (or PSEUDO (6.4.4) or BPSEUDO (6.3.4.5)).
7. Prints the number of basis functions intersected (NUMB), the sum of the squares of the integrals of the intersected basis functions over the ray (SNORM), NP, and NR, for the NP-th projection and NR-th ray used in RAY (6.4.3), WRAY (6.4.2), BRAY (6.3.4.4), or BWRAY (6.3.4.3).
8. Prints the equation of the current ray consisting of a point on the ray (AX,AY), and the direction of the ray (MX,MY) as produced by POSIT (6.4.7).
9. Not used.

10. Prints the indices of the basis functions intersected by the NPth projection and NR-th ray as produced by RAY, WRAY, BRAY, or BWRAY. Additionally, for WRAY and BWRAY, the integrals of the intersected basis functions over the ray will be printed in the same order as their indices are listed (see Sections 6.3.4.3, 6.3.4.4, 6.4.2, and 6.4.3).

TRACE is an inclusive command, i.e., the integer specified produces the trace described above, as well as all traces for integers less than the specified integer. For example, TRACE 4 produces also a trace of algorithm execution (TRACE 1).

5.2 MODE

> MODE [LOWER lower-value [UPPER upper-value] / UPPER upper-value [LOWER lower-value]]

MODE enables the user to set two logical flags named LOFL and UPFL, and to give values to two floating point variables named LOWER and UPPER. If there is no modifier following MODE, the values of LOFL and UPFL are set to FALSE, and the values of LOWER and UPPER remain unchanged (SNARK14 initializes LOFL and UPFL to have the value FALSE, and LOWER and UPPER to have the values 0.0 and 1.0 respectively). If the modifier LOWER follows MODE, LOFL is set to TRUE and the variable LOWER is set to lower-value. (Note that, since alphabetic characters and the equal sign between the modifiers LOWER and lower-value are ignored by SNARK14, the commands

MODE LOWER = −1.0

MODE LOWER CONSTRAINT IS SET AT −1.0

are both legitimate and have the same effect.) If the modifier UPPER follows MODE, UPFL is set to TRUE and the variable UPPER is set to upper-value. If only one of the two modifiers LOWER and UPPER follow MODE, the logical flag associated with the other one is set to FALSE. *Warning:* If both LOFL and UPFL are set to TRUE, then lower-value must not be greater than upper-value.

This instruction has two purposes. Primarily, the values of LOFL, UPFL, LOWER and UPPER are available to all algorithms and may be used by them. (They may be especially useful in reconstruction algorithms, see Chapters 6 and 7.) Secondly, they tell SNARK14 how to interpret the reconstruction when performing an evaluation or display of a reconstruction. If both LOFL and UPFL are FALSE, then the picture values stored on recfil (see Section 3.2) are used unchanged; if LOFL is TRUE, all densities less than LOWER are considered to be equal to LOWER; if UPFL is TRUE, all densities greater than UPPER are considered to be equal to UPPER. While performing the evaluation or display of a reconstruction (which at such time would be residing on recfil) the stored values in the phantom and the reconstruction are not changed, but they are interpreted according to the rules described above for the purpose of evaluation or display.

Example. If on a particular scale the picture values associated with air, brain, and skull are about 0.0, 1.0, and 2.0 respectively, then we can evaluate how well the brain is reconstructed by setting LOWER to 0.5 and UPPER to 1.5 and LOFL and UPFL set to TRUE. This will cause the regions of the picture that are associated with air and skull to have values 0.5 and 1.5, respectively, in both the original and the reconstruction (unless the reconstruction is grossly inaccurate). Thus the error statistics reported on by SNARK14 will be influenced only by the accuracy of reconstruction of the brain matter.

When a reconstruction is being performed in BLOB mode (see Sections 2.2 and 5.7), the values of LOWER and UPPER are applied to the blob coefficients. Because blobs overlap, when such a reconstruction is converted to pixels the pixel values may exceed these bounds.

5.3 CREATE

> CREATE

This command enables the user to create test phantoms and their projection data. The data so generated are then written on file11 in the format required for use by the PICTURE and PROJECTION commands

(see Sections 5.4 and 5.5). See Appendix B for instructions on how to create a file11 containing user supplied data.

Under the CREATE command the user can make use of any of the following features:

1. The objects giving rise to absorption of incident rays (phantom) are approximated by superimposing a number of elemental objects, which may be one of the following: ellipse (circle), rectangle (square), isosceles triangle, sector or segment of a circle.

2. Data corresponding to a monochromatic or a discrete polychromatic input energy spectrum (of up to 7 different energy levels) may be approximated.

3. Projection data for either divergent or parallel ray geometries may be created.

4. Accuracy in calculating the phantom densities and projection data may be improved by sampling finer than finally desired and taking an average.

5. An aperture function may be specified, which would weight the above average so as to simulate an uneven dispersion of intensities of a detected ray across the face of a detector when the detectors are not assumed to be points but of finite spatial extent.

6. In order to simulate non-ideal situations like noise due to photon counting statistics (quantum noise) and scattering from neighboring rays, the user may specify photon distribution and scattering characteristics. In addition, mathematically convenient additive and multiplicative noise may be introduced in the projection data.

7. A background absorption at each energy level may be specified.

Since this is by far the most complex SNARK14 command, both in the length and complexity of its follow up input sequence and in what it has to accomplish, its description is broken up into subsections.

5.3.1 Creation of the phantom

A phantom is created when PHANTOM AVERAGE is specified as part of the CREATE command and the user provides all the necessary information (see Section 5.3.3). The phantom is based on a test picture region that is a square with center at the origin and is divided into equal non-overlapping smaller squares (pixels). This region is defined by the number of pixels to a side (NELEM) and the length of a side of the pixel (PIXSIZ). NELEM must be an odd integer. The origin is assumed to lie at the center of the central pixel.

The phantom is put together by superimposing a number of elemental objects, placed at desired positions, at desired orientations, and of desired size and (absorption) density. The density of the elemental objects may be negative. The density (picture value) at any point is then defined as the sum of the densities associated with all the elemental objects within which the point lies.

To obtain a measure of the average density within a pixel, the user specifies the variable NAVE1, which has the effect that the density within a pixel is determined by averaging the value of the density at NAVE1×NAVE1 uniformly spaced points within the pixel. NAVE1 must be odd.

The user specifies the absorption $d_{j,i}$ at each energy level i and for each elemental object j. From this information we obtain, for each pixel and for each energy level i, an *average density* defined by

$$\frac{1}{(\text{NAVE1})^2} \sum_{k=1}^{(\text{NAVE1})^2} \sum_{j=1}^{\text{NOBJ}} \delta_{k,j} d_{j,i}, \tag{5.1}$$

(where NOBJ is the number of elemental objects in the phantom and $\delta_{k,j}$ is 1 if the kth point of the pixel is inside the jth elemental object and is 0 otherwise) and an *inhomogeneity* that is obtained by multiplying the average density with a random sample from a zero mean Gaussian distribution with standard deviation sd (see Section 5.3.3, Set 3). What we call the *phantom* in SNARK14 is obtained by summing, at each pixel, the average density and the inhomogeneity for that pixel at the *first* energy level.

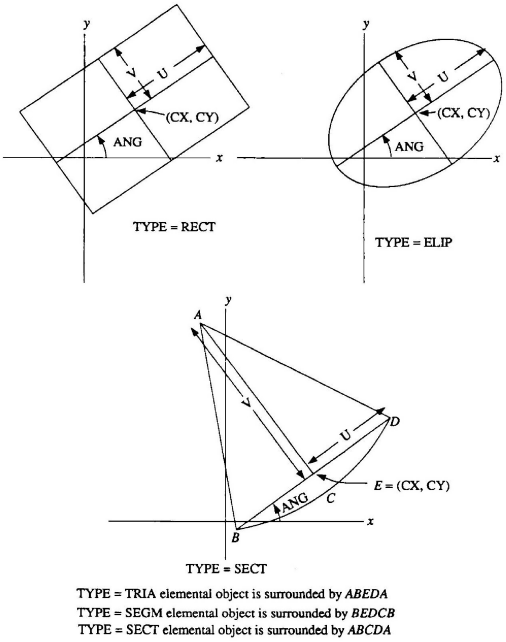

Figure 5.1: Elemental objects

The type and location of an elemental object is described by the six variables TYPE, CX, CY, U, V, and ANG. TYPE can assume the values ELIPSE (note misspelling), RECTANGLE, TRIANGLE, SEGMENT, and SECTOR. For each one of these cases, the explanation of the rest of the variables is given by Figure 5.1. The boundary of an elemental object is considered to be part of the object.

5.3.2 Creation of raysums

Raysums (projection data) are created for a number of rays and a number of projections when RAYSUMS AVERAGE is specified as part of the CREATE command and the user provides all the necessary geometry and measurement information (see Section 5.3.3). Projection data calculated by CREATE are approximations to the real ray sums in the divergent ray geometry (Figure 2.1) or the parallel ray geometry (Figure 2.2). A real ray sum is defined as the integral of the density along a ray. As we have seen in the last section, SNARK14 allows multiple densities associated with each point (one for each energy level), which makes the definition of real ray sums inapplicable in the polychromatic case.

The mathematically simplest way of defining the real raysums in the presence of inhomogeneities is to think of each pixel as an additional square-shaped elemental objects whose density at energy level i is the inhomogeneity (as defined in the last subsection) at that pixel and energy level. Then, for each energy level i, the real ray sum R_i for a LINE ray (the STRIP case will be discussed below) is the sum

$$R_i = \sum_{j=1}^{\text{NOBJ}+(\text{NELEM})^2} l_j d_{j,i}, \tag{5.2}$$

where l_j is the length of intersection of the ray with the jth elemental object (l_j may be 0), and $d_{j,i}$ is the density of the jth elemental object at energy level i.

Projection data that are experimentally collected can only approximate this sum. CREATE contains a number of features that enable it to generate data that approximate the idealized projection data in the same way as projection data obtained from x-rays would. This is explained in the following paragraphs. The explanation assumes a minimal amount of knowledge of how x-rays interact with matter. Such knowledge can be obtained, for instance, from [24, 29].

The approximation to the ray sum produced by CREATE is a function of p, where

$$p = -\ln \frac{A}{C}, \tag{5.3}$$

where C is the calibration measurement (proportional to the number of photons counted by the detector before the insertion of the object to be reconstructed), A is the actual measurement (proportional to the number of photons counted by the detector with the object of interest in place), and ln denotes the natural logarithm. In the following paragraphs we discuss how the number of photons reaching the detector is estimated in increasingly complicated situations.

It is important that the user has a clear understanding of the intended interpretation of the density value $d_{j,i}$ assigned to the elemental object j at energy level i. SNARK14 allows two different interpretations: $d_{j,i}$ is an absolute attenuation coefficient or $d_{j,i}$ is an attenuation coefficient relative to some standard (background) material, such as water. In the latter case, it is assumed that calibration is done with the background material between the source and the detector, and during the actual measurement the background material is (partially) replaced by the object to be reconstructed. It is also assumed that, for each energy level i, the attenuation by the background material is constant, B_i, for all rays. A special case of this is obtained by assuming that the background material is vacuum and all B_is are zero. This special case is the same as using absolute attenuation coefficients.

The calibration measurement C is estimated in a way that depends on the arrangement of the x-ray source and detector(s). In certain scanners, the value of C is constant for all rays of one projection, but varies from projection to projection. In other scanners, the value of C is constant for all projections for a particular ray (say, the fifth ray), but varies from ray to ray. Yet in other cases, C may be different for each ray in each projection. We use the variable QUANIN to indicate which of these three cases applies. The value of QUANIN is 1 in the first case, 2 in the second, and 3 in the third. (The value of QUANIN is determined by the CREATE follow up command lines.) QUANIN may also assume the value 0. This indicates that data are to be generated without any noise due to photon statistics (quantum noise), in which case C need not be calculated at all, as we shall see below.

In order to take care of the variability in the strength of the x-ray source a reference x-ray beam is used. Thus C itself is calculated as a ratio

$$C = \frac{C_0}{C_r}, \tag{5.4}$$

where C_0 is the number of photons counted by the detector under consideration and C_r is the number of photons counted during the same period by a reference detector during the calibration measurement.

The expected values of C_0 and C_r are the same number; C_0 and C_r differ from each other only because of the statistical nature of the counting process, which results in the so-called *quantum noise*. Hence, if QUANIN = 0, we may assume that $C = 1$.

Since we assume that, for each energy level i, the attenuation by the background material is a constant, B_i, for all rays, we get that (see [24]), if λ is the expected number of photons that would be detected by the detector when the ray went through vacuum, then the expected number of photons that are detected when the ray goes through the background material is

$$\lambda \sum_{i=1}^{\text{NERGY}} p_i e^{-B_i}, \tag{5.5}$$

where NERGY is the number of energy levels and p_i is the portion of the energy spectrum at level i ($\sum_{i=1}^{\text{NERGY}} p_i = 1$).

In case the follow up command lines of CREATE indicate that quantum noise is to be introduced in the projection data (QUANIN > 0), C is calculated as the ratio of two instances from the normal distribution whose mean is given by Equation (5.5) and whose standard deviation is the square root of its mean. The λ in this case is defined by the product QUANMN×QUANCM, where the values of QUANMN and QUANCM are supplied by the follow up command lines of CREATE.

The actual measurement is also estimated as a ratio

$$A = \frac{A_0}{A_r}, \tag{5.6}$$

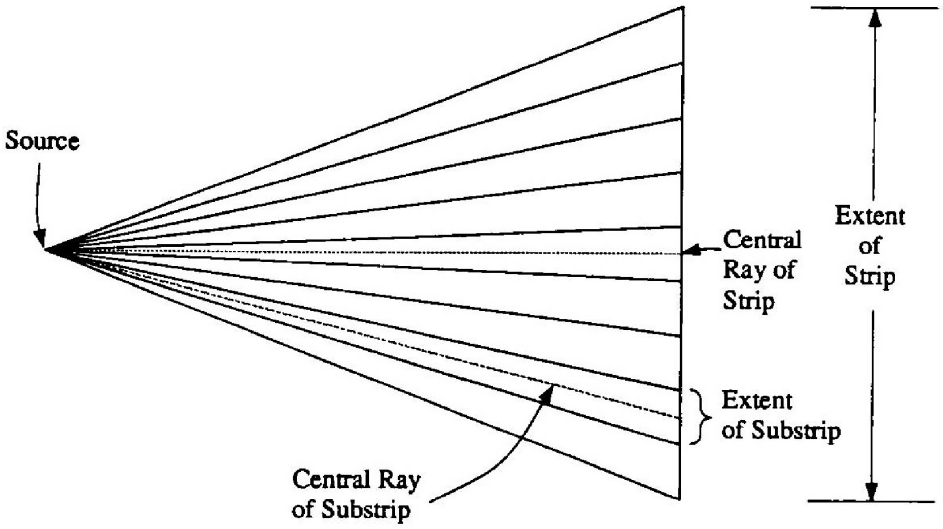

Figure 5.2: Substrip interpretation for DIVERGENT-TANGENT geometry. (It is assumed that the source is a mathematical point, even though in practice it would have some extent.)

where A_0 is the number of photons counted by the detector under consideration and A_r is the number of photons counted by the reference detector during the actual measurement.

If quantum noise is to be introduced in the projection data, the value of A_r is an instance from the normal distribution whose mean is given by Equation (5.5) and whose standard deviation is the square root of its mean. The λ in this case is defined to be QUANMN. (Thus QUANCM is the ratio of the number of photons used for the calibration measurements to the number of photons used in the actual measurements.)

The calculation of A_0 is more complex, since the total attenuation along different lines from the source to the detector can no longer be assumed to be the same. For the purpose of estimating A_0, we assume that photons are collected from within a strip that, in the DIVERGENT-TANGENT and in the PARALLEL geometries, meets the line of detectors in a line segment of width PINC (or PINC $\times \max(|\sin\theta|, |\cos\theta|)$ in the parallel case if VARIABLE detector spacing is specified) centered around the mathematical position of the detector (see Figure 5.2). In the DIVERGENT-ARC geometry the definition is similar, but now the strip is on an arc (see Figure 2.1).

To simulate what happens in practice, the user specifies a variable NAVE2 and an array NAPER(1), ..., NAPER(NAVE2). NAVE2, a positive integer, determines the number of smaller strips into which the projection strip is divided for the purpose of calculating A_0 (see Figure 5.2). The number of photons that is assumed to enter the nth substrip is

$$r_n = \text{NAPER}(n) / \sum_{k=1}^{\text{NAVE2}} \text{NAPER}(k) \tag{5.7}$$

times the total number of photons entering the strip. Thus, NAPER defines an aperture function.

Let $R_{i,n}$ be the real sum for the ith energy level for the central ray of the nth substrip (defined by Equation (5.2)). Then the expected number of photons that is detected by the detector is estimated by

$$\text{QUANMN} \sum_{n=1}^{\text{NAVE2}} r_n \sum_{i=1}^{\text{NERGY}} p_i e^{-(R_{i,n}+B_i)}. \tag{5.8}$$

If quantum noise is to be introduced in the projection data, the actual value of A_0 is an instance of a random variable. Let Q be the value given by Equation (5.8). If Q is less than 100.0, then A_0 is an instance of a Poisson distribution whose mean is Q. Otherwise, A_0 is an instance of a normal distribution whose mean is Q and whose standard deviation is the square root of Q. If quantum noise is not to be introduced

in the projection data, A_0 is given by Equation (5.8), A_r is given by Equation (5.5) with λ = QUANMN, and so the value of QUANMN is irrelevant in estimating A by Equation (5.6).

Warning: The way quantum noise is dealt in SNARK14is based on the assumptions that the means are reasonably large. If the means are small it may be incorrect [29].

Whether or not quantum noise is to be introduced in the projection data, the value of A_0 may be further modified before substitution into Equation (5.6). The purpose of this modification is to simulate the change in A_0 due to photons scattered from nearby rays and cross-talk between neighboring detectors. The model used to simulate these effects is such that it would not change the expected values of C_0 and we assume that such effects do not enter into the working of the reference detectors. Hence, when *scatter noise* is specified by the CREATE follow up sequence, only A_0 is modified from the value given to it above.

In order to explain the scatter model precisely, let $\overline{A}_i$ and A_i denote the value of A_0 for the ith detector (FUSRAY $\leq i \leq$ LUSRAY) before and after scatter has been introduced.

$$A_i = \left[\sum_{j=\text{FUSRAY}}^{\text{LUSRAY}} v_{i-j} \overline{A}_j \right] / V_i \tag{5.9}$$

where v_k and V_i are calculated as follows. The user provides in the CREATE follow up command lines two variables: SCTNPK (scattering peak) and SCTNWD (scattering width). For all integers k, let

$$v_k = \begin{cases} 0, & \text{if } |k \times \text{PINC}| > \text{SCTNWD}, \\ \text{SCTNPK}(1 - \frac{|k \times \text{PINC}|}{\text{SCTNWD}}), & \text{if } k \neq 0 \text{ and } |k \times \text{PINC}| \leq \text{SCTNWD}, \\ 1 + \text{SCTNPK}, & \text{if } k = 0, \end{cases} \tag{5.10}$$

and let

$$V_i = \sum_{j=\text{FUSRAY}}^{\text{LUSRAY}} v_{i-j}. \tag{5.11}$$

In the PARALLEL VARIABLE case, every occurrence of PINC in these formulas must be replaced by PINC $\times \max(|\sin\theta|, |\cos\theta|)$ where θ is the projection angle. (Thus, scattering is modeled by convolution with a triangle shaped function, such that the total number of photons detected by all the detectors remain constant, except for edge effects.) Strictly speaking, scattering should increase the total number of photons detected, but the ratio of increase would be the same during the calibration measurement, and the two increases cancel out.

To summarize, real ray sum in the LINE case is estimated by Equation (5.3), where C and A are given by Equations (5.4) and (5.6), respectively. This estimation may incorporate the effects of calibration with a background material, polychromaticity, wide detector with a nonuniform detecting efficiency across its face, quantum noise and scatter. In addition, the user may specify additive and multiplicative noise to further degrade the projection data. If the follow up command lines so indicate, CREATE will first multiply p by an instance from a normal distribution and then add to the resulting value an instance from another normal distribution. The means and standard deviations of these distributions are provided by the user in the CREATE follow up command lines. It is this finally altered value of p that is stored by CREATE on file11.

In the PARALLEL geometry situation, we allow for STRIP integrals by multiplying the value obtained as described above by PINC, if UNIFORM detector spacing is specified, or by PINC $\times \max(|\sin\theta|, |\cos\theta|)$ if VARIABLE detector spacing is specified (θ is the projection angle, denoted as THETA in Figure 2.2).

5.3.2.1 Simulating PET with SNARK14

In positron emission tomography (PET) we are interested in the uptake of positron-emitting isotopes by various parts of the human body. When a positron is emitted it annihilates with a nearby electron and produces two γ-ray photons of identical energy traveling in approximately opposite directions. The two photons are detected in near coincidence by a pair of opposite detectors. The annihilation, and thus the positron emission, is known to take place somewhere along the line joining the detector pair. We count such coincidences for a number of detector pairs around the body. From these measured counts our aim is to estimate the concentration of the positron emitter at various points in the body cross-section.

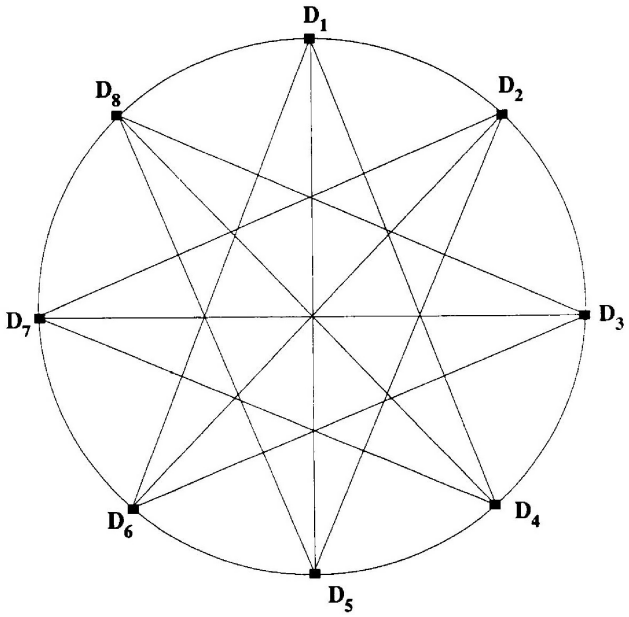

Figure 5.3: Simplified geometry of eight-detector PET system

Figure 5.3 shows a simplified PET geometry consisting of a ring of 8 detectors. For simplicity, we assume that each detector is coupled with 3 opposite detectors to detect (near) coincidence arrivals of photons. Thus the lines sampled by each detector form a divergent pattern as shown in Figure 5.3. By analogy with x-ray CT, we refer to the collections of such (divergent) lines as a *view* and the lines themselves as *rays* in the view [50]. Thus in Figure 5.3 we have 8 views with 3 rays per view and 12 rays in total $\left(\frac{8\times 3}{2}\right)$ in all the views.

To simulate measurements by a PET system, SNARK14 utilizes many of the existing routines previously written to simulate x-ray CT measurements. In generating the PET data, SNARK14 attempts to simulate the general characteristics of data that would be collected by a PET scanner, rather than attempt to simulate a particular PET device. To see how SNARK14 uses its x-ray CT organization to simulate data collection for the simplified PET geometry in Figure 5.3, consider Figure 5.4 that shows a schematic of a SNARK14 DIVERGENT x-ray CT geometry with the detectors located on an arc (Section 2.4). The raysums (defined by Equation (5.2) in Section 5.3.2) are measured along lines joining the source and three opposite detectors located on an arc of a circle whose center is at the source. Figure 5.4 illustrates the situation when the source is at location D_2 and the detectors are on the arc $D_7'D_6'D_5'$. The locations $D_1 - D_8$ in Figure 5.4 correspond to the detector positions in Figure 5.3. It is clear from Figure 5.4 that the detector locations (except for the central detector) are not the same as those in the PET geometry of Figure 5.3. In Figure 5.4 a full scan is made by measuring the raysums as the source rotates through locations D_1-D_8. The PET data simulation is completed by generating, for each raysum, a Poisson random variable whose mean is given by the value of the raysum. Thus, it is easy to see that while the CT geometry of Figure 5.4 does not precisely simulate the PET geometry of Figure 5.3, the measurements obtained by the CT system possess similar characteristics of the data that would be collected by the PET system. Note that as a consequence of using its x-ray CT organization to simulate the PET system of Figure 5.3, SNARK14 generates a total of 24 raysums (instead of the 12 measurements that would be obtained in Figure 5.3), since two separate measurements (raysums) are obtained for each pair of PET detectors.

In SNARK14, PET data are simulated by using a value of 4 for the variable QUANIN. In this case, the values of QUANMN and QUANCM are ignored, except for the sign of QUANMN. If QUANMN ≥ 0.0 then each raysum is fed to a Poisson random number generator. If QUANMN < 0.0, then the raysum is not fed to the Poisson random number generator (i.e. the data are noise-free). The "noise" in the data is adjusted by scaling the phantom values to achieve the desired total number of photon counts. This scaling is achieved by means of the LAST command line of the OBJECTS command (Section 5.3.3, Set 3), and the

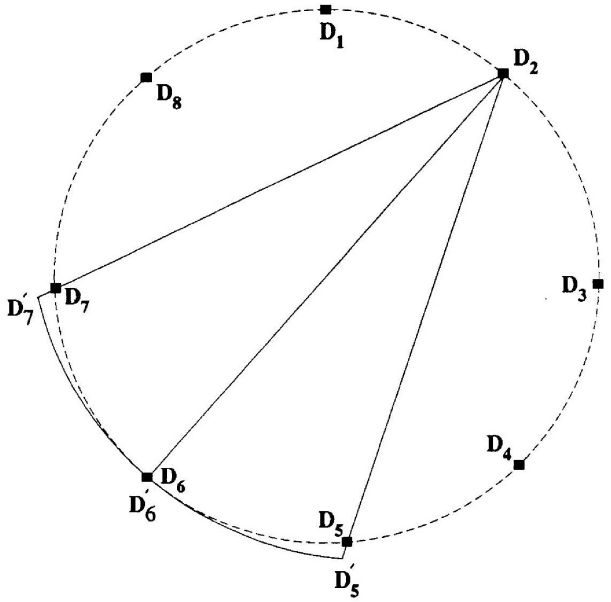

Figure 5.4: SNARK14 divergent x-ray CT geometry used to simulate the PET geometry shown in Figure 5.3.

total count is equal to the value of the parameter TOTDEN written to the file OUTPUT generated after a SNARK14 PET run. The procedure for achieving a total count of, say, $T2$ is the following. Let $T1$ be the total count with unit scaling (i.e scale=1 in the LAST command line of Section 5.3.3, Set 3). Then scale=$\frac{T_2}{T_1}$ will produce the count $T2$. (An example of a SNARK14 run that simulates PET data collection is given in Appendix A, Section A.7).

Note: When simulating PET data collection SNARK14 ignores attenuation effects.

5.3.3 Description of the CREATE input sequence

The CREATE command line is followed by a certain input in a specific form and order in the *Data Generation Input file*. This input sets out all the information necessary for the generation of phantom and/or projection data. The input has been separated into eight sets of logically divided segments for convenience in presentation, implementation, and use. It has been designed to enable the user to identify each input quantity easily (and as such may appear redundant in places). This is done by making generous use of descriptive words. The following provides a description of each set of records (command lines); the entire command is the totality of these command lines taken in order.

Certain restrictions are imposed on input quantities so that they are computationally meaningful or physically acceptable. These are also indicated. Errors are trapped during input and error messages are issued. Errors stop the input processing and the program aborts. Examples of input sequences are given in Appendix A.

SET 1: IDENTIFYING HEADER

1.1 FORMAT: string of up to 80 characters

> name-of-the-pattern

This name is used for identification of the phantom and projection data. A command line beginning with an asterisk is not treated as a comment here. It is the name of the pattern.

SET 2: ENERGY SPECTRUM

2.1 FORMAT: control command line format (see Section 4.4)

> SPECTRUM { MONOCHROMATIC energy / POLYCHROMATIC nergy }

Both energy and nergy are integer modifiers.

The value of energy is used for echo checking purposes only. For example, the value 70 may be assigned to energy to indicate that we intend to simulate a monochromatic x-ray beam at 70 keV. The output produced on file11 by CREATE will carry this information in its headers for both the phantom (see Section 5.4) and the projection data (see Section 5.5), but it will not be influenced in any other way by the value given to energy.

The value of nergy is the number of energy levels and is assigned to the variable NERGY. *Restriction*: 1 ≤ nergy ≤ 7.

2.2 FORMAT: control command line format

> energy(1) percent(1) ... energy(NERGY) percent(NERGY)

energy(i), percent(i) (1 ≤ i ≤ NERGY) are integer modifiers.

Restriction: This command line is absent in the MONOCHROMATIC case. The value of energy(i) (1 ≤ i ≤ NERGY) is used for the ith energy level in the polychromatic case exactly the same way as energy is used in the monochromatic case (see command line 2.1 above).

Restriction: The values of percent(1), . . . , percent (NERGY) must be nonnegative integers and add up to 100. The portion p_i of the energy spectrum at level i (used in Section 5.3.2) is defined by $p_i = \text{percent}(i)/100$.

SET 3: ELEMENTAL OBJECTS

3.1 FORMAT: control command line format

> OBJECTS

3.(2n) (1 ≤ n ≤ NOBJ) FORMAT: control command line format

> { ELIPSE / RECTANGLE / TRIANGLE / SEGMENT / SECTOR } $c_x\ c_y\ u\ v$ ang den(1)

c_x, c_y, u, v, ang, and den(1) are all floating point modifiers.

This command line describes an elemental object. The values of c_x, c_y, u, v, and ang are assigned to the variables CX, CY, U, V, and ANG, which (together with the shape name) define the elemental object as described in Section 5.3.1, especially in Figure 5.1. The value of den(1) is the density of this elemental object at energy level 1, in the polychromatic case, or at the only energy level in the monochromatic case. *Restriction*: $u >$ ZERO and $v >$ ZERO.

This command line appears NOBJ (number of elementary objects) times. In the polychromatic case, if NERGY > 1, it is always followed by:

3.(2n+1) (1 ≤ n ≤ NOBJ) FORMAT: control command line format

> DENSITY den(2) . . . den(NERGY)

den(2) . . . den(NERGY) are floating point modifiers.

Restriction: This command line is absent in the monochromatic case or if NERGY = 1. The value of den(i) (2 ≤ i ≤ NERGY) is the density of the elemental object defined by the previous command line at the ith energy level.

3.(2 NOBJ + 2) FORMAT: control command line format

> LAST scale [seed sd]

scale is a floating point modifier, seed is an integer modifier and sd is a floating point modifier. If seed and sd are not specified, their values are assumed to be 0.

This command line indicates the end of the part of the input stream describing the elemental objects. Note that NOBJ is not explicitly read in, but is defined by the number of command lines between command line OBJECTS and the command line LAST. All previously defined densities are changed by multiplication by the value of scale. It is not necessary that there be any objects between the OBJECTS and LAST control lines; i.e. NOBJ can be 0.

Restrictions: scale > ZERO and sd ≥ 0.

If seed and sd are specified, then to each pixel density d (see Equation (5.1)) is altered by adding a random sample from a zero-mean normal distribution with standard deviation sd$\times d$, where seed controls the random number generator for producing the instances from the normal distribution (see Section 5.3.1) as follows:

If seed < 0, the current time (using the Linux system call "time") is used as the seed of the random number generator.

If seed $= 0$, a default sequence of random numbers is produced.

If seed > 0, a sequence of random numbers is produced that is dependent on the value of seed.

SET 4: PHANTOM GENERATION

4.1 FORMAT: control command line format

> PHANTOM [AVERAGE nave1]

Restriction: The value of the integer modifier nave1 must be positive and odd. In case AVERAGE is specified, a test phantom is generated and put on file11. The value of nave1 is assigned to the variable NAVE1, whose purpose is described in Section 5.3.1.

In case AVERAGE is not specified, a test phantom is not generated. With our conventions the user may in this case use the command line

PHANTOM IS NOT GENERATED

4.2 FORMAT: control command line format

> nelem pixel-size

Restrictions: This command line is absent if AVERAGE is not specified on the previous command line. nelem > 0 and odd, pixel-size > ZERO. The value of nelem is assigned to the variable NELEM and the value of pixel-size is assigned to PIXSIZ, the purpose of which is explained in Section 2.2.

SET 5: RAYSUM GENERATION

5.1 FORMAT: control command line format

> RAYSUM [AVERAGE nave2]

Restriction: The value of the integer modifier nave2 must be positive, odd and not greater than 13.

In case AVERAGE is specified, projection data are generated and put on file11 (following the test phantom data if such have been generated due to command line 4.1). The value of nave2 is assigned to the variable NAVE2 whose purpose is described in Section 5.3.2.

In case AVERAGE is not specified, projection data are not generated.

Restriction: If AVERAGE is not specified there must be no further follow up command lines in the CREATE command.

5.2 FORMAT: control command line format

> naper(1) ... naper(NAVE2)

Restriction: The value of each of the integer modifiers naper must be nonnegative and their sum must be positive.

The value of naper(i) is assigned to the variable NAPER(i) (for $1 \leq i \leq$ NAPER) whose purpose is described in Section 5.3.2.

SET 6: PROJECTION GEOMETRY SPECIFICATION

6.1 FORMAT: control command line format

> GEOMETRY

This command line introduces the projection geometry.

6.2 FORMAT: control command line format

$$> \left\{ \begin{array}{l} \text{PARALLEL} \left\{ \begin{array}{l} \text{UNIFORM} \\ \text{VARIABLE} \end{array} \right\} \left\{ \begin{array}{l} \text{STRIP} \\ \text{LINE} \end{array} \right\} \\ \text{DIVERGENT} \left\{ \begin{array}{l} \text{ARC} \\ \text{TANGENT} \end{array} \right\} \text{source-to-origin source-to-detector} \\ \text{LINOGRAM} \end{array} \right\}$$

The first word on this command line determines whether rays in a single projection are parallel (as in Figure 2.2) or divergent (as in Figure 2.1).

In the PARALLEL case the spacing between the detectors may be the same in all projections (UNIFORM) or it may vary from projection to projection (VARIABLE). The actual size of detector spacing is determined by information read on the next command line. The ray sum in the STRIP case is the ray sum in the LINE case multiplied by the detector spacing (see Section 5.3.2).

In the DIVERGENT case, the detector strip may be a circular ARC with its center at the source or a TANGENT to this arc (see Figure 2.1). The value of the floating point modifier source-to-origin is the RADIUS of the circle on which the source is located, the value of the floating point modifier source-to-detector is the distance (STOD) of the source to the detector strip (see Figure 2.1). *Restrictions*: source-to-origin > ZERO and source-to-detector > ZERO.

If LINOGRAM is specified, PHANTOM AVERAGE nave1 and its follow-up line (see create command Set 4 on the preceding page) are required and therefore a phantom will be generated on file11.

In the LINOGRAM case, the following relationship between the object size, the number of projections and the number of rays per projection holds. Let $2N+1$ be the number of pixels along a side of the digitized picture of the object. Then there are a total of $2(4N+3)$ projections with $4N+3$ rays per projection. Moreover, the $2(4N+3)$ projections are generated over two sets of angles as follows:

$$\begin{aligned} \theta \quad \in \quad & \left\{ \arctan\left(\frac{2i}{4N+3}\right) + \frac{\pi}{2} \mid -(2N+1) \leq i \leq (2N+1) \right\} \cup \\ & \left\{ \arctan\left(\frac{2i}{4N+3}\right) + \pi \mid -(2N+1) \leq i \leq (2N+1) \right\}. \end{aligned}$$

Thus for the first set of angles we have $\frac{\pi}{4} < \theta < \frac{3\pi}{4}$ and for the second set of angles we have $\frac{3\pi}{4} < \theta < \frac{5\pi}{4}$. Further, LINOGRAM implies PARALLEL VARIABLE ray geometry.

Warning: Due to the assumptions in the routines ray in Section 6.4.3 and wray in Section 6.4.2, reconstructions from STRIP data may produce poor reconstructions unless VARIABLE is also specified and PINC is an integer multiple of PIXSIZ.

6.3 FORMAT: control command line format

$$> \quad \text{RAYS} \left\{ \begin{array}{l} \text{USER user-rays} \\ \text{PROGRAM nelem pixel-size} \end{array} \right\} \text{detector-spacing}$$

If USER is specified, the real ray sum is estimated in each projection for as many rays as is the value of the integer modifier user-rays. The value of user-rays is assigned to the variable USRAYS. *Restrictions*: The value of user-rays must be a positive odd integer. The central one of these rays goes through the origin, the others are symmetrically distributed around the central one (see Section 2.4). In the DIVERGENT ARC case (user-rays $-$ 1) $\times$ detector-spacing $<$ STOD $\times$ π.

If PROGRAM is specified, the number of rays is calculated based on the values of nelem, pixel-size and detector-spacing. The value of USRAYS is calculated by the formula given for SNRAYS in Section 2.5, with the values of NELEM, PIXSIZ and PINC determined by nelem, pixel-size and detector-spacing. The values of nelem and pixel-size on command line 6.3 are given for the sole purpose of calculating SNRAYS; they do not have to be the same as the values of the corresponding modifiers on command line 4.2. *Restriction*: nelem $>$ 0 and odd, and pixel-size $>$ ZERO. Also nelem $\times$ pixel-size/$\sqrt{2}$ $<$ RADIUS.

The value of detector-spacing is the distance (PINC) between the centers of the detectors in all but the PARALLEL VARIABLE case. In this special case the distance between the centers of the detectors in a projection is the value of detector-spacing multiplied by the maximum of $|\sin(\text{THETA})|$ and $|\cos(\text{THETA})|$, where the angle THETA is defined as in Figure 2.2.

This line is absent in the LINOGRAM case.

Restriction: detector-spacing $>$ ZERO.

6.4 FORMAT: control command line format

$>$ ANGLES prjnum [EQUAL SPACING]

The value of the integer modifier prjnum is the number of projections for which projection data are to be calculated. It is assigned to the variable PRJNUM.

This line is absent in the LINOGRAM case.

Restriction: prjnum $>$ 0 and if EQUAL SPACING is specified then prjnum $>$ 1.

6.5 FORMAT: control command line format

$>$ { angle(0) angle(PRJNUM-1) } or { angle(0) angle(1) ... angle(PRJNUM-1) }

angle(0), angle(1), . . . , angle(PRJNUM-1) are floating point modifiers.

The value of angle(i) is used as the value in degrees of the angle THETA for the ith projection, $0 \le i <$ PRJNUM (see Figures 2.1 and 2.2). In case EQUAL SPACING is specified on the previous command line, the first option indicated above is used and angle is calculated by

$$\text{angle}(i) = \text{angle}(0) + \frac{\text{angle(PRJNUM-1)} - \text{angle}(0)}{\text{PRJNUM} - 1} i.$$

If EQUAL SPACING is not specified, the values of angle(0), angle(1), ... , angle(PRJNUM-1) are read in this order from command line 6.5 and lines following 6.5 until PRJNUM floating point values are read.

The list of angles is missing in the LINOGRAM case.

SET 7: MEASUREMENT STATISTICS

7.1 FORMAT: control command line format

$>$ MEASUREMENT { PERFECT or NOISY }

If the option PERFECT is specified, then in the calculation of raysums it is assumed that measurements are unaffected by quantum noise, scatter, additive, or multiplicative noise (see Section 5.3.2). In this case command lines 7.2 and 7.3 are to be omitted. If the option NOISY is specified, at least one of the four sources of noise mentioned above is assumed to be present. The various sources of noise are described on (possibly multiple appearances of) command line 7.2. If neither PERFECT or NOISY is specified, PERFECT is assumed.

7.2 FORMAT: control command line format

$$> \left\{ \begin{array}{l} \text{QUANTUM quanmn quancm CALIBRATION quanin} \\ \text{SCATTER sctnpk sctnwd} \\ \text{ADDITIVE addnmn addnsd} \\ \text{MULTIPLICATIVE multnmn multnsd} \end{array} \right\}$$

Restriction: Command line 7.2 is omitted if PERFECT is specified on command line 7.1.

The first key-word of this command line indicates what type of noise is described by the command line. Multiple versions of this command line may appear in the input sequence at this point, one for each possible type of noise. If there is no command line 7.2 for a particular type of noise, then that type of noise will not be used in the calculation of the ray sums.

The values of the floating-point modifiers quanmn, quancm, sctnpk and sctnwd are assigned to the variables QUANMN, QUANCM, SCTNPK, and SCTNWD, whose purpose has been explained in Section 5.3.2. The value of the integer modifier quanin is assigned to the variable QUANIN, which controls the nature of the assumed calibration methods (for x-ray CT data simulation); as discussed in Section 5.3.2.1, when simulating PET data QUANIN should be set to 4. *Restrictions*: quanmn > ZERO, quancm > ZERO (these restrictions on quanmn and quancm do not apply when PET is simulated; see Section 5.3.2.1 for details), sctnpk > ZERO and sctnwd $\geq$ PINC, $1 \leq$ quanin ≤ 4.

The values of the floating point modifiers addnmn and addnsd are the mean, ADDNMN, and standard deviation, ADDNSD, of the normal distribution an instance from which is used as additive noise (see Section 5.3.2), and the values of the floating point modifiers multnmn and multnsd are the mean, ULTNMN, and standard deviation, ULTNSD, of the normal distribution an instance from which is used as multiplicative noise. *Restriction*: |multnmn| > ZERO. In mathematical notation, in the presence of these types of noise SNARK14provides an estimate P_{new} of the ray sum based on the estimate P_{old} of the ray sum without additive or multiplicative noise by the formula

$$P_{new} = P_{old} \times gauss(\text{ULTNMN}, \text{ULTNSD}) + gauss(\text{ADDNMN}, \text{ADDNSD}),$$

see Section 6.4.18 for definition of *gauss*.

In case there is more than one command line 7.2 with a particular type of noise in the input sequence, only the last of these command lines is effective. If there is no command line 7.2 for a particular type of noise, default values are assigned to the variables mentioned above. These default values are all zeros. For SCATTER, ADDITIVE, and MULTIPLICATIVE noise there are logical variables, (SCTNFL, ADDNFL, and ULTNFL, respectively) the value of each of which is set to TRUE if and only if a command line 7.2 for that particular noise is present. The presence of QUANTUM noise is indicated by the value of QUANIN>0.

7.3 FORMAT: control command line format

> SEED [seed]

Restriction: This command line appears in the input sequence only when it is preceded by a command line of the form 7.2 with one of the key-words QUANTUM, ADDITIVE or MULTIPLICATIVE. It controls the random number generator that is used for producing instances from normal distributions for the various types of noise.

The value of the integer modifier seed controls the random number sequence as follows. If no seed value is specified, 0 is assumed.

If seed < 0, the current time (using the Linux system call "time") is used as the seed to the random number generator.

If seed = 0, a default sequence of random numbers is produced.

If seed > 0, a sequence of random numbers is produced that is dependent on the value of seed.

7.4 FORMAT: control command line format

> BACKGROUND backgr(1) . . . backgr(NERGY)

The value of the floating-point modifier backgr(i) is the value of B_i, the assumed attenuation by the background material at energy level i (see Section 5.3.2).

SET 8: RUN COMMAND

8.1 FORMAT: control command line format

> RUN

5.4 PICTURE

> PICTURE { RECONSTRUCTION nelem pixel-size / TEST }

This command specifies the values of the variables NELEM and PIXSIZ for the picture to be reconstructed (see Section 2.2).

If RECONSTRUCTION is specified, these values are provided by the integer modifier nelem and floating-point modifier pixel-size respectively. *Restrictions*: nelem > 0 and odd, and pixel-size > ZERO.

If TEST is specified then it is assumed that file11 is present and it contains the following command lines:

I. Command lines that are in the order and format of Sets 1, 2, 3, and 4 of the CREATE follow up sequence. Apart from specifying NELEM and PIXSIZ, these command lines have the purpose of keeping a complete record of how the test-phantom has been created.

II. These are followed by the values given to the pixels in the test-phantom. The order of pixels is row-by-row (top row first) and, within rows, left-to-right. Each new row starts on a new command line.

This instruction results in the beginning of the creation of recfil, a binary file that is used during the analysis phase (see Section 3.3). NELEM and PIXSIZ are stored on this file, as well as the name-of-the-pattern (from command line 1.1 of the CREATE follow up sequence) and the pixel values of the phantom, if TEST is specified. The name of the pattern appears in an identifying header of any phantom-dependent output produced during the analysis phase.

5.5 PROJECTION

> PROJECTION { OLD / REAL [BEAM HARDENING CORRECTION] / PSEUDO }

The purpose of this command is to create the binary file prjfil (see Section 3.2.1), which contains information on the projection data both for the reconstruction subphase and the analysis phase. In addition, the PROJECTION command causes the determination of the values of all variables (other than NELEM and PIXSIZ) that are needed by the reconstruction subphase (e.g., PINC, PRJNUM, NRAYS, etc.). Of particular importance is the assignment to the variable AVEDEN, the estimated value based on the projection data, of the average density of the picture to be reconstructed. The exact method by which AVEDEN is calculated is explained in Section 6.3.3. *Restriction*: A PROJECTION command must be preceded by a PICTURE command in the *Initialization and Reconstruction Input file*.

If OLD is specified, it is assumed that prjfil is already present. In this case, values of the variables that are needed in the reconstruction subphase are read in from prjfil, but these values are not echo-checked on the *Initialization and Reconstruction Output file*. For this reason the use of this option is recommended only if the creation of prjfil by the appropriate alternative option (REAL or PSEUDO) would be time consuming.

If REAL is specified, it is assumed that file11 is present and is positioned at the beginning of the following sequence of command lines:

I. Command lines that are of the order and format of Sets 1, 2, 3, 5, 6, and 7 of the CREATE follow up sequence. If command line 6.2 specifies LINOGRAM, then the projection angles are computed to a high degree of accuracy. The projection angles in the projection data (see below) are expected to match these computed angles. *Restrictions*: On command line 6.3 only the USER option may be used. In the DIVERGENT case detector-spacing $\times$ user-rays $< \pi \times$ source-to-detector and $\sqrt{2} \times$ source-to-origin $\geq$ nelem $\times$

pixel-size. (The values of nelem and pixel-size are determined by the PICTURE command that precedes the PROJECTION command.)

II. These are followed by command lines containing the projection data. These are arranged projection-by-projection. For each projection, the first command line contains the projection angle THETA of Figures 2.1 and 2.2 in radians and the same angle in degrees (*restriction*: these must match with the angles as specified in degrees by command line 6.5), and the following command lines contain the USRAYS ray sums, where the value of USRAYS is determined by the integer modifier user-rays on command line 6.3. See Appendix B for instructions on how to create a file11 containing user supplied data.

If REAL is specified, the geometry of data collection described on file11 is also the geometry of data collection described on prjfil.

If BEAM HARDENING CORRECTION is specified, the projection data stored on file11 are corrected for beam hardening that occurs when a polychromatic x-ray source is used. The corrected projections are estimates of monochromatic projections at the first energy level. This option is ignored if called in a run that uses monochromatic projections. The option requires the follow-up sequence described below.

> niters ndegree

The integer modifier niters is the number of iterative steps in the correction. The integer modifier ndegree is the degree of the polynomial used to approximate monochromatic projection values based on the polychromatic projection data. The coefficients of this polynomial are provided on the next line of the input:

> a(0) ... a(ndegree)

The floating-point modifiers a(0) ... a(ndegree) are the ndegree+1 coefficients of the polynomial.

If the value of niters is 0, the rest of the followup sequence is not read and can be omitted.

> nergy

The integer modifier nergy must have the same value as NERGY, i.e., the number of energy levels as specified in Set 2 of the CREATE followup sequence (it is here just as means of verification).

The beam hardening correction uses iterative data refinement in order to improve on the rough correction provided by the above polynomial [25]. The attenuation values at energy levels 2 to energy(nergy) are approximated based on the piecewise linear function provided by the user on the following lines of the input:

> npoints

The integer modifier npoints is the number of points used in the piecewise linear approximation of the pixel values at energy levels 2 ... nergy based on the value at the first energy level.

> x(2)(1) y(2)(1)

> ...

> x(2)(npoints) y(2)(npoints)

> ...

> x(nergy)(1) y(nergy)(1)

> ...

> x(nergy)(npoints) y(nergy)(npoints

The interpretation of these sequence of floating point modifiers is the following.

It is assumed that if the density is 0 at the first energy level, then it is also 0 at all other energy levels. Further, for $2 \leq i \leq$nergy and $1 \leq j \leq$npoints, it is assumed that if the density is x$(i)(j)$ at the first energy level, then it is y$(i)(j)$ at the ith energy level.

Restrictions: If BEAM HARDENING CORRECTION is specified, the projection angles need to be specified in increasing order, i.e. angle(i) < angle(i+1) on control command line 6.5.

If PSEUDO is specified, the PROJECTION command line must be followed by command lines that are of the order and format of Sets 1, 6, and 7 of the CREATE follow up sequence. *Restriction*: PSEUDO can only be specified if the PICTURE command specified TEST. In this case, prjfil will contain the pseudo ray sums of the NELEM × NELEM digitized picture that is the test phantom (see Sections 2.2 and 2.3) for all rays of the geometry specified in Set 6 of the CREATE follow up sequence. These ray sums will be contaminated by noise according to the specifications of Set 7 of the CREATE follow up sequence. It will be assumed that the number of energy levels (NERGY) is 1.

Warning: In all cases, the ray sums stored on prjfil will be corrected for bias. Thus, if REAL is specified, then the ray sum on file11 is first reduced by ADDNMN and then is divided by ULTNMN prior to being stored on prjfil. If the user needs the value of a ray sum, the function `prdta` (see Section 6.3.6.2) returns the unbiased value on prjfil. The values of ULTNMN and ADDNMN are available to the user, so that the original biased value may be recalculated.

The prjfil will contain an 80 character long name-of-the-projection. If OLD is specified, this is of course the already existing name-of-the-projection on the file. If REAL is specified, the name-of-the projection is what is read in from command line 1.1 of the CREATE follow up sequence on file11. If PSEUDO is specified, the name-of-the-projection is provided by the line immediately following the PROJECTION command line.

In any output produced during the analysis phase reporting on reconstructions based on prjfil the name-of-the-projection appears in an identifying header.

Warning: The file punch records only the first 30 characters of the name-of-the-projection (see Section 5.13).

5.6 SELECT

$$> \quad \text{SELECT} \left\{ \begin{array}{l} \text{USER} \\ \text{SNARK} \end{array} \right\} \left\{ \begin{array}{l} [\text{type1 } [\text{n1 type2 } [\text{n2}]]] \\ \text{EFFICIENT} \end{array} \right\}$$

As will be discussed in Section 6.4.1, a function is available to the user for selecting the next ray to be considered in a reconstruction by SNARK14. This function is called PICK(NP,NR) and it returns a projection number NP ($0 \leq$ NP $<$ PRJNUM) and a ray number NR (if USER is specified FUSRAY $\leq$ NR $\leq$ LUSRAY, and if SNARK is specified FSNRAY $\leq$ NR $\leq$ LSNRAY, see Section 2.5).

The word modifiers type1 and type2 may have values RAYSEQ, PROJSEQ, and RANDOM. *Restriction*: n1 > 0 and n2 > 0.

When all four modifiers type1, n1, type2, n2 are present, then the command is interpreted to mean that repeatedly n1 actions of type1 are followed by n2 actions of type2. If n2 is absent, SNARK14 behaves as if n2 were infinity (i.e., n1 actions of type 1 are followed by as many actions of type2 as there are calls of PICK(NP, NR)). If only type1 is present, SNARK14 behaves as if n1 were infinity. *Restriction*: If both type1 and type2 are present, then exactly one of them must be RANDOM.

In RANDOM mode, PICK(NP,NR) returns a randomly selected ray in a randomly selected projection.

In order to understand how the other two modes operate, we need to specify a fairly complicated ordering mechanism. Let A and B be integers in the ranges [AF, AL] and [BF, BL], respectively, and let AI and BI be positive integers. The following C++-like code describes a way of writing all pairs $\{(A, B) \,|\text{AF} \leq A \leq \text{AL}, \text{BF} \leq B \leq \text{BL}\}$ in a particular order without any repeats:

```
for (int i = AF; i < min(AF+AI, AL+1); ++i) {
   for (int j = BF; j < min(BF+BI, BL+1), ++j) {
      for (int b = j; b < BL+1; b += BI) {
         for (int a = i; a < AL+1; a += AI) {
            printf("%d %d\n", a, b);
         }
```

```
        }
    }
}
```

For example, if AI = 1 and BI = 1, the sequence will be (AF,BF), (AF+1,BF), . . . , (AL,BF), (AF,BF+1), . . . , (AL,BF+1), . . . , (AL,BL). If AF = 5, AL = 7, AI = 2, BF = 3, BL = 7, BI = 3, the sequence will be (5,3), (7,3), (5,6), (7,6), (5,4), (7,4), (5,7), (7,7), (5,5), (7,5), (6,3), (6,6), (6,4), (6,7), (6,5).

We use the notation SEQ(AF,AL,AI,BF,BL,BI) to denote the infinite sequence of pairs (A, B) produced by repetition of the sequence described above.

If type1 is RAYSEQ or PROJSEQ or type2 is RAYSEQ or PROJSEQ, the SELECT command line must be followed by a command line in control command line format whose content is

```
>                STEP mod1 mod2
```

The values of the integer modifiers, mod1 and mod2, must be positive.

In order to describe precisely the sequence in which NP and NR are picked by RAYSEQ and PROJSEQ, we define

$$\mathrm{F} = \begin{cases} \text{FUSRAY}, & \text{if USER is specified,} \\ \text{FSNRAY}, & \text{if SNARK is specified,} \end{cases}$$

$$\mathrm{L} = \begin{cases} \text{LUSRAY}, & \text{if USER is specified,} \\ \text{LSNRAY}, & \text{if SNARK is specified.} \end{cases}$$

The function PICK(NP,NR) is only supposed to be used while performing a reconstruction due to an EXECUTE command (see Section 5.8). If the first call of PICK(NP,NR) under control of an EXECUTE command is in RAYSEQ or PROJSEQ mode, then PICK returns (1,F). In other words, each EXECUTE command in the Initialization and Reconstruction Input file reinitializes the sequence of (NP,NR)s returned by PICK.

Suppose now that PICK(NP,NR) is called and (NP′,NR′) are the values that have been returned by PICK the last time it was called. Then PICK returns the pair (NP,NR) determined as follows. If RAYSEQ is specified, (NP,NR) is the pair that follows (NP′,NR′) in the sequence SEQ(F,L,mod2,1,PRJNUM,mod1). If PROJSEQ is specified, (NP,NR) is the pair that follows (NP′,NR′) in the sequence SEQ(1,PRJNUM,mod1,F,L ,mod2).

The EFFICIENT option orders the projections and rays as described in [38]. The principle behind this ordering is to order the projections and rays within the projections such that each action is as independent as possible from the previous action. Let P be some number and let

$$P = p_U \times \ldots \times p_2 \times p_1 \tag{5.12}$$

be the unique decomposition of P into its prime factors with $p_U \geq \ldots \geq p_2 \geq p_1$. Let T be the set of U-dimensional vectors t, whose uth component t_u is an integer that satisfies

$$0 \leq t_u < p_u, \tag{5.13}$$

for $1 \leq u \leq U$. With any such vector t, we associate an integer

$$\begin{aligned} v(t) \quad = \quad & (p_U \times p_{U-1} \times \ldots \times p_2 \times t_1) \\ & +(p_U \times p_{U-1} \times \ldots \times p_3 \times t_2) + \ldots \\ & +(p_U \times t_{U-1}) + t_U. \end{aligned} \tag{5.14}$$

It is easy to prove that v is a one-to-one mapping of T onto the range $[0, P)$ of integers.

We now define, inductively, another one-to-one mapping τ of the range $[0, P)$ of integers onto T. First, $\tau(0)$ is the vector of all 0's. Suppose now that $\tau(p-1) = (t_1, t_2, \ldots, t_U)$. Let v be the smallest integer such that $t_v \neq p_v - 1$. Then $\tau(p)$ is the vector whose uth component is 0 for $1 \leq u < v$, is $t_u + 1$ for $u = v$, and is t_u for $v < u < U$. Finally, our permutation π of the integers in the range $[0, P)$ is defined by $\pi(p) = v(\tau(p))$.

When EFFICIENT is selected, the above permutation is applied to the projections (P = NPROJ) and within that to the rays in the projection (P =SNRAYS if SNARK is specified or P =USRAYS if USER is

specified). In order to achieve efficiency, it is important that the P be decomposable into many (preferably small) prime factors.

If this command is omitted, the default behavior of PICK is as if

SELECT USER EFFICIENT

had been executed.

5.7 BASIS

> BASIS { PIXEL | BLOB [support shape delta] }

The BASIS command controls the basis functions for the algebraic algorithms. They may either be PIXELs or BLOBs (see Section 2.2). The command has no effect on transform methods of reconstruction, but it changes the behavior of the series expansion methods [29]. If BLOB is selected, it may optionally be followed by three floating point parameters that control the shape of the blob. If they are omitted, they are set to the following default values:

$$\begin{aligned} \text{DELTA} &= 2 \times \text{pixel-size}/\sqrt{3}, \\ \text{SHAPE} &= 11.2828631556, \text{ and} \\ \text{SUPPORT} &= 1.7865601396 \times \text{delta}. \end{aligned}$$

If this command is omitted,

BASIS PIXEL

is assumed.

5.8 EXECUTE

"To seek it with thimbles, to seek it with care;
To pursue it with forks and hope;
To threaten its life with a railway share;
To charm it with smiles and soap!

"For the Snark's a peculiar creature, that won't
Be caught in a common place way.
Do all that you know, and try all that you don't:
Not a chance must be wasted today!"

Fit the Fourth - The Hunting
The Hunting of the Snark (an agony in eight fits)
by Lewis Carroll

> EXECUTE [ZERO | AVERAGE | CONTINUE | PHANTOM] algname [CONTOUR | SMOOTH]

Restriction: This command must be preceded by a PICTURE and a PROJECTION command in the Initialization and Reconstruction Input file.

The EXECUTE command line must always be followed by a command line containing

> name-of-the-execution

Only the first 80 characters of this command line are significant. Characters in excess of 80 are discarded. The name-of-the-execution appears on all outputs produced during the analysis phase that report on reconstructions produced by this EXECUTE command line. A command line beginning with an asterisk is not treated as a comment here. It is the name of the execution.

Warning: The file punch records only the first 30 characters of the name-of-the-execution (see Section 5.13).

The EXECUTE command produces reconstructions from the projection data on prjfil and stores them on the recfil. If an iterative algorithm is used, the reconstruction after each iteration (together with the iteration number) is stored on recfil. The total number of iterations caused by the EXECUTE command is determined by the most recently executed STOP command (see Sections 5.9 and 6.1). When BASIS BLOB is in effect, blob reconstructions are converted from blobs to pixels before storing them on recfil.

The options ZERO, AVERAGE, CONTINUE, and PHANTOM determine how the reconstruction region is initialized. (If these are absent, ZERO is assumed.) If ZERO is specified, then the elements of the array that is used to store the reconstructed picture are set to 0.0 at the very beginning of the reconstruction process. If AVERAGE is specified, then the elements of the array that is used to store the reconstructed pictures are set to AVEDEN at the beginning of the reconstruction process. (See Section 6.3.3 for definition of AVEDEN.) If CONTINUE is specified, then the content of the reconstruction region (i.e., the reconstructed picture due to the last EXECUTE command) is used unaltered at the beginning of the reconstruction algorithm. This enables us, for example, to use the result of an algorithm as the starting point (0'th iterate) of a different algorithm. If CONTINUE is specified by the first EXECUTE command in the initialization and reconstruction input file, it is treated as if ZERO had been specified. If the previous algorithm produced a pixel reconstruction and the current algorithm is a blob based algorithm, the reconstruction is converted from pixels to blobs before the algorithm is invoked. Similarly, if the previous reconstruction was a blob reconstruction and the current algorithm is a pixel algorithm, the reconstruction is converted from blobs to pixels before the algorithm is invoked. If the previous algorithm produced a blob reconstruction and the current algorithm is also a blob algorithm, then the blob parameters must be the same for both executions. If the blob parameters have changed, the algorithm is not executed. If PHANTOM is specified the reconstruction region is initialized to the phantom that was read in (see Section 5.4).

The word modifier algname may assume the following values: BACKPROJECTION, CONVOLUTION, RFL, FOURIER, DCONV, ART, MART, QUADRATIC, SIRT, EMAP, LINO, ALP1, ALP2, ALP3, ALP4, ALP5, ALB1, ALB2, ALB3, ALB4, and ALB5. (The algorithms ART, MART, QUADRATIC, SIRT, ALB1, ALB2, ALB3, ALB4, and ALB5 produce either pixel or blob images depending on the BASIS -see Section 5.7- that is in effect. The other algorithms always produce pixel images.) All but ALP1, ALP2, ALP3, ALP4, ALP5, ALB1, ALB2, ALB3, ALB4, and ALB5 refer to built-in SNARK14algorithms that can be called to do a reconstruction. They are described in detail in Chapter 7. If any of ALP1, ALP2, ALP3, ALP4, ALP5, ALB1, ALB2, ALB3, ALB4, and ALB5 is specified, the reconstruction algorithm is defined by the user in a C++ class, whose name is alp1, alp2, alp3, alp4, alp5, alb1, alb2, alb3, alb4 or alb5, respectively (see Section 6.1). This class must be compiled and bound with the rest of SNARK14prior to execution. The default implementations of these user defined routines are do-nothing reconstruction algorithms. That is, they leave the reconstruction region unchanged.

If either CONTOUR or SMOOTH is specified, the name-of-the-execution command line is followed by a command line in control command line format

> threshold weight-1 weight-2 weight-3

and an

> iteration-flag-line

(see Section 4.5) *Restriction*: These two command lines are absent if neither CONTOUR nor SMOOTH is specified on the EXECUTE control command line.

The effect of CONTOUR and SMOOTH is that some of the reconstructions are modified prior to being stored on recfil. If BASIS BLOB is in effect, the modifications are performed after the blob image is converted to a pixel image. These modifications have the following forms.

If CONTOUR is specified, and weight-3 $\leq$ ZERO, all reconstructed densities that are less than or equal to threshold are set to weight-1, and all reconstructed densities that are greater than threshold are set to weight-2. If weight-3 $>$ ZERO, essentially the same process is carried out except that the user-defined value of threshold is replaced by another value defined as follows. While executing the PROJECTION command, the average density of the picture to be reconstructed is estimated based on the projection data and is stored in a variable called AVEDEN (see Section 6.3.3 for a precise definition of AVEDEN). If weight-3 $>$ ZERO, threshold is chosen so that the average density of the contoured picture is as near to AVEDEN as is possible for the given weight-1 and weight-2, unless AVEDEN $<$ weight-1 or AVEDEN $>$ weight-2 or if the reconstruction (prior to contouring) has the same value in all pixels or blobs. (If any of these conditions exist, the reconstruction is not changed and a warning message to that effect is printed.)

If SMOOTH is specified, let $V_1 \ldots V_9$ denote the reconstructed densities in a pixel and its neighbors before smoothing, as indicated by the diagram below.

V_6	V_2	V_7
V_3	V_1	V_4
V_8	V_5	V_9

The value of the center pixel that is stored on recfil will be

$$\frac{W_1 \times V_1 + W_2 \times \sum_{i=2}^{5} f_i V_i + W_3 \times \sum_{i=6}^{9} f_i V_i}{W_1 + W_2 \times \sum_{i=2}^{5} f_i + W_3 \times \sum_{i=6}^{9} f_i},$$

where

$$f_i = \begin{cases} 1, & \text{if } |V_i - V_1| \leq \text{threshold}, \\ 0, & \text{if } |V_i - V_1| > \text{threshold}, \end{cases}$$

and W_i denotes the value of weight-i from the command line, for $1 \leq i \leq 3$. If the pixel labeled by 1 is on an edge, and therefore doesn't have a neighbor labeled by i, then f_i is assumed to be 0.

If the fraction defined above has a value greater than INFIN (an overflow condition, see Section 2.1), SNARK14 stores INFIN for the corresponding pixel and gives a warning message on the Initialization and Reconstruction Output file.

The iteration-flag-line determines which of the reconstructions produced by EXECUTE are altered in this fashion prior to being stored on recfil. If fl(q) > 0, then the operation (of contouring or smoothing) will be performed on the reconstruction after q iterations (unless the number of iterations is exactly q) if $q > 0$ and on the final reconstruction of the EXECUTE command if fl(0) > 0. (For definition of "reconstruction after q iterations", see Section 5.9.) The following sequence can be used to save both the original and smoothed reconstructions from the convolution algorithm on recfil:

```
STOP ITERATION 2
EXECUTE CONVOLUTION SMOOTH
Both original and smoothed convolution reconstructions
0.5 4.0 2.0 1.0
1
... rest of convolution parameters
```

These command lines may have to be followed with other command lines, containing input required by the algorithm to be executed. The formats and contents of these command lines for the internal SNARK14 algorithms are described in Chapter 7.

5.9 STOP

$$
> \quad \text{STOP} \left\{ \begin{array}{l} \text{ITERATION iter} \\ \\ \text{TERMINATION} \left\{ \begin{array}{l} \text{VARIANCE epsilon} \\ \text{MLST [RPRT [n]]} \\ \text{KLDS epsilon [RPRT [n]]} \\ \text{RESI epsilon [RPRT [n]]} \\ \text{WSQD epsilon [RPRT [n]]} \\ \text{TRM1} \\ \text{TRM2} \end{array} \right. \end{array} \right.
$$

The purpose of this command is to control the number of iterations used by an algorithm. It terminates an iterative algorithm when the specified stopping criterion is satisfied. This is achieved as follows (see Section 6.1 for additional details).

The EXECUTE command (Section 5.8) causes SNARK14 to execute the instructions of a reconstruction algorithm. Some algorithms are iterative, i.e., the same set of instructions need to be executed repeatedly, so as to improve the result. One complete execution of a set of such instructions is called an iteration.

SNARK14 updates a NELEM×NELEM array in each iteration. The elements of this array are perceived as estimates of the pixel densities of the picture to be reconstructed. For this reason, the digitized picture whose values are based on the values in this array after q iterations is called the reconstruction after q iterations. In the blob case, the NELEM×NELEM array is the result of converting the blob reconstruction to a pixel reconstruction using blob2pix (see Section 6.3.4.1).

A single EXECUTE command causes as many iterations of an iterative algorithm to be carried out as is specified by the most recently executed STOP command. If there are no STOP commands executed prior to the EXECUTE command, the behavior of the EXECUTE is the same as if

STOP ITERATION = 1

had been executed.

If ITERATION is specified, an EXECUTE command results in iter iterations. *Restriction*: iter > 0.

If TERMINATION is specified, then at the end of each iteration the Run method of an instance of the termtest_class, specified by the word modifier VARIANCE, MLST, KLDS, RESI, TRM1, or TRM2, is invoked. This method tests whether certain conditions are satisfied and SNARK14 decides whether to carry out another iteration based on the outcome of these tests.

If VARIANCE is specified, the iterative process is terminated as follows. Let ρ_i^q be the density assigned to the ith pixel of the reconstruction after q iterations and before any SMOOTHING or CONTOURING that may have been specified (see Section 5.8). Let AREA=NELEM^2 be the number of pixels in the reconstruction. Let $\overline{\rho}^q$ be the average density of the reconstruction after q iterations. We define

$$v^q = \frac{1}{\text{AREA}} \sum_{i=0}^{\text{AREA-1}} \left(\rho_i^q - \overline{\rho}^q\right)^2 . \tag{5.15}$$

The iterative process stops after the qth iteration if $q > 1$ and $|v^q - v^{q-1}| <$ epsilon$\times v^{q-1}$. *Restriction*: epsilon > ZERO.

If MLST is specified, the execution of the algorithm is stopped as soon as the indicator function $D(x)$, which is advocated to be used for MLEM-STOP in [5], reaches a value which is less than or equal to 1. It is defined as:

$$D(x) = \frac{\sum_{i=1}^{I} \left(b_i - \sum_{j=1}^{J} a_{ij} x_j\right)^2}{\sum_{i=1}^{I} \sum_{j=1}^{J} \left(a_{ij} x_j\right)}, \tag{5.16}$$

where x denotes the reconstructed image, I denotes the number of lines of response (LOR), J denotes the number of pixels for which the activity needs to be determined by the reconstruction process, b_i is defined as the count of events in the ith LOR and a_{ij} is defined as length of intersection of the ith LOR with the jth pixel, which is equivalent to the probability of counting an event generated in the jth pixel for the ith

LOR. MLST supports the output of the current value of $D(x)$ by setting the RPRT flag and will report to the command line as well as to the file RPRTmlst. [n] indicates the nth iteration of the iterative algorithm; specifying the value of n will cause STOP to report only on the first, last and every n-th iteration.

If KLDS is specified, the iterative process is terminated if the value of the Kullback-Leibler distance (KLDS) of the reconstructed image is less than or equal to the specified value of epsilon. The Kullback-Leibler distance is a proximity value and it is defined as:

$$KL\left(x\right)=\sum_{i=1}^{I}\left(b_i\ln\frac{b_i}{\sum_{j=1}^{J}a_{ij}x_j}+\sum_{j=1}^{J}a_{ij}x_j-b_i\right), \tag{5.17}$$

where x denotes the reconstructed image, I denotes the number of lines of response (LOR) and J denotes the number of pixels for which the activity needs to be determined by the reconstruction process, b_i is defined as the count of events in the ith LOR and a_{ij} is defined as the probability of counting an event generated in the jth pixel for the ith LOR. KLDS supports the output of the current value of $Pr\left(x\right)$ by setting the RPRT flag and will report to the command line as well as to the file RPRTklds. Specifying the value of n will cause STOP to report only on the first, last and every n-th iteration.

If RESI is specified, the reconstruction process is stopped once the residual reaches a value less than or equal to the specified epsilon. Calculation of the residual is performed similarly to its calculation for the eval file; see (5.31). RESI supports the output of its current value by setting the RPRT flag and will report to the command line as well as to the file RPRTresi. Specifying the value of n will cause STOP to report only on the first, last and every n-th iteration.

If WSQD is specified, the reconstruction is terminated if the value of the weighted squared distance of the reconstructed image is less than or equal to the specified value of epsilon. The weighted squared distance is a proximity value [46] and it is defined as:

$$WS(x)=\sum_{i=1}^{I}\frac{\left(b_i-\sum_{j=1}^{J}a_{i,j}x_j\right)^2}{\sum_{j=1}^{J}a_{i,j}}, \tag{5.18}$$

where the meaning of the various variables is the same as for the KLDS proximity value above. WSQD supports the output of its current value by setting the RPRT flag and will report to the command line as well as to the file RPRTwsqd. Specifying the value of n will cause STOP to report only on the first, last and every n-th iteration.

If TRM1 or TRM2 are specified, the termination test is defined by the user in a C++ class whose name is `trm1_class` or `trm2_class`, respectively (see Section 6.1). This class must be compiled and bound with the rest of SNARK14 prior to execution. The `Init` method of this class is invoked during the STOP command processing so that it may read its run time parameters, if any, using the methods of the `InputFile_class` (see Section 6.3.8).

5.10 SUPERIORIZE

$$> \quad \text{SUPERIORIZE N a b} \left\{\begin{array}{l}\text{TVAR}\\ \text{SMOO}\\ \text{SCR3}\\ \text{SCR4}\\ \text{SCR5}\end{array}\right\}[\text{POSI}]\left[\begin{array}{l}\text{ATL1}\\ \text{ATL2}\end{array}\right][\text{RPRT [n]}]$$

SUPERIORIZE is an implementation of the Superiorization Methodology [21]. If this command is inserted into the INPUT file, then any EXECUTE command calling an iterative algorithm that follows it in the INPUT file will be automatically superiorized according to a specified secondary optimization criterion, whose name is one of TVAR, SMOO, SCR3, SCR4 or SCR5. (The first two of these are built into SNARK14, the other three are user defined; see Section: 5.10.1.) We explain the roles of the various modifiers in the

SUPERIORIZE control command line using a pseudocode specification (Algorithm 5.1), in which the name of the secondary criterion is denoted by ϕ. (*Warning*: This command can be applied only with PIXEL basis functions.)

Suppose that the iterative algorithm that is to be superiorized is specified by the algorithmic operator $\mathbf{P}$, such that if the AREA-dimensional vector x specifies the contents of the array that is used to store the reconstructed picture at the beginning of an iterative step, then $\mathbf{P}x$ specifies the contents of the same array at the end of the iterative step. The array in question is referred to by the name `recon` in Sections 6.1 and 6.2. The pseudocode below defines how the superiorized version of the iterative algorithm behaves; the next few lines explain some of the terminology in the pseudocode.

The stopping criterion is provided by the command STOP of Section 5.9.

The notation $random(k, \ell)$ denotes a randomly selected integer not smaller than ℓ and not larger than k. The code to generate this integer makes use of the function Rand; see Section 6.4.17.

We use $x^{(k)}$ to denote the AREA-dimensional column vector whose jth component, $x_j^{(k)}$ is `recon[j-1]` after the kth step. In particular, $x^{(0)}$ represents the contents of the `recon` array at the beginning of the reconstruction process, as determined by the ZERO, AVERAGE, or CONTINUE option of the EXECUTE command (see Section 5.8).

The choice of a "nonascending vector for ϕ at $x^{(k,n)}$" is discussed in Section 5.10.2.

The set Δ is the set of all AREA-dimensional vectors unless POSI is specified, in which case an AREA-dimensional vector is in Δ only if all its components are nonnegative. (*Warning*: Specifying POSI may result in an endless loop if combined with a reconstruction algorithm that allows negative values in the reconstruction, such as ART without MODE LOWER 0.)

Algorithm 5.1 Superiorized Version of an Iterative Algorithm

set $k = 0$

set $\ell = 0$

while stopping criterion is not satisfied

    if ATL1 has been specified **set** $\ell = k$

    if ATL2 has been specified **set** $\ell = random(k, \ell)$

    set $n = 0$

    set $x^{(k,n)} = x^{(k)}$

    while $n <$ N

        set $v^{(k,n)}$ to be a nonascending vector for ϕ at $x^{(k,n)}$

        set *loop=true*

        while *loop*

            $\beta = \mathrm{b} \times \mathrm{a}^{\ell}$

            set $\ell = \ell + 1$

            set $z = x^{(k,n)} + \beta v^{(k,n)}$

            if $z \in \Delta$ **and** $\phi(z) \leq \phi\left(x^{(k)}\right)$ **then**

                set $n = n + 1$

                set $x^{(k,n)} = z$

                set *loop* = *false*

19. **set** $x^{(k+1)} = \mathbf{P}x^{(k,N)}$

20. **set** $k = k + 1$

21. **return** $x^{(k)}$

SUPERIORIZE supports the output of the current values of ℓ and ϕ by setting the RPRT flag and will report to the command line as well as to the file RPRTsuperiorization. Specifying the value of n will cause SUPERIORIZE to report only on the first, last and every n-th iteration.

5.10.1 Secondary optimization criteria

The secondary optimization criterion ϕ indicates how well a vector $x \in \Delta$ satisfies a desired condition. A vector $x_1 \in \Delta$ is considered superior to the vector $x_2 \in \Delta$ if $\phi(x_1) < \phi(x_2)$. Currently, there are two built-in secondary optimization criteria in SNARK14: TVAR and SMOO.

TVAR calculates the total variation (TV) of a reconstruction x and is defined as:

$$TV\,(x) = \sum_{c \in C} \sqrt{\left(x_c - x_{\rho(c)}\right)^2 + \left(x_c - x_{\xi(c)}\right)^2}, \tag{5.19}$$

where C is the set of all indices of pixels that are not in the rightmost column or the bottom row of the pixel array. For any pixel with index c in C, $\rho(c)$ and $\xi\,(c)$ are the indices of the pixels to its right and below it.

SMOO calculates a measure of smoothness. It is an implementation of the penalty function ψ recommended in [53] and is defined as:

$$\psi(x) = \sum_{r \in M} \left(x_r - \frac{1}{8} \sum_{l \in M_r} x_l \right)^2, \tag{5.20}$$

where x is a reconstructed image, M is the set of indices r such that the rth pixel is not on the border of the image region and M_r, for $r \in M$, is the set of indices associated with the eight pixels neighboring the rth pixel.

If SCR3, SCR4 or SCR5 are specified, the secondary criterion must be defined by the user in a C++ class whose name is `scr3_class, scr4_class` or `scr5_class`, respectively (see Section 6.1). This class must be compiled and bound with the rest of SNARK14 prior to execution. The `Init` method of this class is invoked during the SUPERIORIZE command processing so that it may read its run time parameters, if any, using the methods of the `InputFile_class` (see Section 6.3.8).

5.10.2 Nonascending vectors

For the Superiorization Methodology to work as presented in [21], it is necessary to obtain a nonascending vector for the secondary optimization criterion ϕ at the point $\boldsymbol{x}^{(k,n)}$, see line 9 of Algorithm 5.1. We define $\boldsymbol{d}$ to be of a *nonascending vector* for ϕ at a point $\boldsymbol{x}$ if the following condition is satisfied:

$$\begin{aligned} &\text{there is a } \delta > 0 \text{ such that for all } \lambda \in [0, \delta]\,, \\ &\quad (x + \lambda d) \in \Delta \text{ and } \phi\,(x + \lambda d) \leq \phi\,(x)\,. \end{aligned} \tag{5.21}$$

The set Δ is defined just above Algorithm 5.1.

One way to obtain a nonascending AREA-dimensional vector d is specified by Theorem 2 of [21], as follows. Consider any AREA-dimensional vector g that satisfies the property: For $1 \leq j \leq J$, if the jth component g_j of g is not zero, then the partial derivative $\frac{\partial \phi}{\partial x_j}(x)$ of ϕ at x exists and its value is g_j. Define d to be the zero vector if $\|g\| = 0$ and to be $-g/\|g\|$ otherwise. Then d is a nonascending vector for ϕ at $\boldsymbol{x}$. There exist other methods to obtain nonascending vectors, but the one just described is a powerful one; for example, it is guaranteed to work if ϕ is a convex function.

Currently, there are only two built-in procedures in SNARK14 to compute nonascending vectors. They compute a nonascending vector for the cases when $\phi\,(x)$ is the $TV\,(x)$ of (5.19) or the $\psi\,(x)$ of (5.20), respectively, and they are invoked appropriately when the keywords TVAR and SMOO are used. (For example,

when the keyword TVAR is used, then SNARK14 will employ a procedure to compute a nonascending vector for the secondary optimization criterion $TV(x)$).

If SCR3, SCR4 or SCR5 are specified, the procedure to compute the nonascending vector for the corresponding secondary criteria must be defined by the user in a C++ class whose name is `nav3_class`, `nav4_class` or `nav5_class`, respectively (see the previous subsection and Section 6.1). This class must be compiled and bound with the rest of SNARK14 prior to execution (see Appendix D3).

5.11 EVALUATE

"My poor client's fate now depends on your votes."
Here the speaker sat down in his place,
And directed the Judge to refer to his notes
And briefly to sum up the case.

But the Judge said he had never summed up before;
So the Snark undertook it instead,
And summed it so well that it came to far more
Than the witness ever had said!

Fit the Sixth - The Barrister's Dream
The Hunting of the Snark (An agony in eight fits)
by Lewis Carroll

> EVALUATE [RESOLUTION | POINT | BOTH] [LINE]

This command causes the evaluation of some quantitative measures of the overall difference between a test picture and reconstructions. (*Restriction*: The test picture must be present on recfil.) For the purpose of comparison only, the pixel values in both the test picture and the reconstruction are interpreted according to the current values of LOFL, UPFL, LOWER and UPPER (see Section 5.2). The measures of difference are reported in a file called eval, see Section 3.3.

The EVALUATE command line must always be followed by a command line containing

> name-of-the-evaluation

This name of the evaluation as it appears in the first line of the eval file.

The name-of-the-evaluation command line is followed by a region selection command line:

> { WHOLEPIC [lowden highden] | SELECTIVE }

If WHOLEPIC is selected, all the pixels in the entire square picture are used. Within this picture, the user can set a density window by specifying two optional parameters, lowden and highden, to exclude pixels that lie outside the specified density range. The density selection is with respect to the phantom density. The default density window is set with lowden = -INFIN and highden = +INFIN.

Note that the effect on evaluation with the setting of a density window is very different from that of the flags: LOFL, and UPFL, and density thresholds: LOWER and UPPER as described in Section 5.2. The latter has the effect of clipping both the phantom and reconstruction densities to the specified density range, the pixels with clipped density values are then used in the evaluation process. (See also the remarks on the computation of residual given at the end of this section.)

If SELECTIVE is chosen, it has to be followed by specifications of the regions. Each region is defined by a region specification line:

> shape c_x c_y u v ang [lowden highden]

The input of region specifications is terminated by the line

> LAST

In each region specification line, the keyword "shape" describes the type of object used to define the region. This keyword can either be ELIPSE or RECTANGLE. The next five parameters have the same meanings as in object definitions for the phantom (see command Set 3 on page 35) that specify the center coordinates, the sizes of the principal axes and the orientation of the object. If the specified region is not entirely within the picture region, it is clipped at the boundaries and a warning message is issued. The last two parameters lowden and highden are optional for setting the density window within each region. The usage is the same as with the WHOLEPIC command line described above. If more than one region is specified, the measures are computed separately for each region.

The EVALUATE command input is completed with an

> iteration-flag-line

(see Section 4.5). The first digit controls which measures will be reported on the eval file for the last iteration. The other digits on the iteration-flag-line can be set either to 0 or to the same as the first digit. The eval file does not report the measures for an iteration q if fl(q) is 0.

We now define the measures provided by EVALUATE. We use $\rho^q_{i,j}$ to denote the density assigned to the pixel in the ith row and jth column of the reconstruction after q iterations (see Section 5.9 for definition of terminology) if $q \geq 1$, and the density in the corresponding pixel in the phantom if $q = 0$. Let S denote the set of pairs (i, j) where the pixel in the ith row and jth column is in the region of interest, i.e., it is within the shape and lowden $\leq \rho^0_{i,j} \leq$ highden.

$$\alpha = \sum_{(i,j)\in S} 1. \tag{5.22}$$

$$\overline{\rho}^q = \frac{1}{\alpha} \sum_{(i,j)\in S} \rho^q_{i,j}. \tag{5.23}$$

$$\upsilon^q = \frac{1}{\alpha} \sum_{(i,j)\in S} \left(\rho^q_{i,j} - \overline{\rho}^q\right)^2. \tag{5.24}$$

$$\sigma^q = \sqrt{\upsilon^q}. \tag{5.25}$$

$$\delta^q = \begin{cases} \frac{1}{\sigma^0}\sqrt{\frac{1}{\alpha}\sum_{(i,j)\in S}\left(\rho^q_{i,j} - \rho^0_{i,j}\right)^2}, & \text{if } \sigma^0 > \text{ZERO}, \\ \sqrt{\sum_{(i,j)\in S}\left(\rho^q_{i,j} - \rho^0_{i,j}\right)^2}, & \text{if } \sigma^0 \leq \text{ZERO}. \end{cases} \tag{5.26}$$

$$m = \sum_{(i,j)\in S} |\rho^0_{i,j}|. \tag{5.27}$$

$$R^q = \begin{cases} \left(\sum_{(i,j)\in S} |\rho^q_{i,j} - \rho^0_{i,j}|\right)/m, & \text{if } m > \text{ZERO}, \\ \sum_{(i,j)\in S} |\rho^q_{i,j} - \rho^0_{i,j}|, & \text{if } m \leq \text{ZERO}. \end{cases} \tag{5.28}$$

The following measures are defined only if all pixels in the full picture are included in the evaluation, and will not be computed if a "region of interest" is specified either through density windowing or region selection (or both).

For the definition of the next set of measures, we need to introduce some preliminary concepts. For any $n \times n$ digitized picture p, we use $p(k)$ to denote the $\lfloor \frac{n}{2^k} \rfloor \times \lfloor \frac{n}{2^k} \rfloor$ (where $\lfloor x \rfloor$ is the greatest integer $\leq x$) digitized picture obtained from p by averaging the grayness of $2^k \times 2^k$ pixels of p into one pixel of $p(k)$. If 2^k does not divide n, the right-most columns and the bottom rows of p are discarded in the calculation of $p(k)$. For any two $n \times n$ digitized pictures p and p', we define

$$E(p, p') = \max_{0\leq i<n, 0\leq j<n} |p_{i,j} - p'_{i,j}|, \tag{5.29}$$

where $p_{i,j}$ and $p'_{i,j}$ denote the values assigned to the pixel in the ith row and jth column of p and p', respectively. Using these concepts we define, for any integer k, such that $1 \leq 2^k \leq$ NELEM,

$$E(2^k)^q = E(\rho^q(k), \rho^0(k)). \tag{5.30}$$

Let $R_{p,r}$ denote the estimate of the real ray sum, as stored on prjfil, of the rth ray of the pth projection. Let $R^q_{p,r}$ and $R^0_{p,r}$ denote the pseudo ray sum of the rth ray of the pth projection of the reconstructed picture and test phantom, respectively, except if LINE is specified on the EVALUATE control command line and data generation was in STRIP mode. In this latter case, $R^q_{p,r}$ (respectively, $R^0_{p,r}$) denotes PINC times the real ray sum of the reconstructed picture (respectively, test phantom) along the central line of the rth ray in the pth projection.

$$r^q = \sqrt{\sum_{p=0}^{\text{PRJNUM-1}} \sum_{r=\text{FUSRAY}}^{\text{LUSRAY}} (R^q_{p,r} - R_{p,r})^2}. \tag{5.31}$$

If RESOLUTION is specified, SNARK14 puts the values of α (AREA, see (5.22)), $\overline{\rho}^0$ (TEST AVERAGE, see (5.23)), υ^0 (TEST VARIANCE, see (5.24)), σ^0 (TEST STD DEV, see (5.25)) are put on the eval file. If $\text{fl}(0) = 2$ (see Section 4.5), r^0 (TEST RESIDUAL, see (5.31)) is also put on the eval file. If $\text{fl}(0) = 3$ (see Section 4.5), $KL\left(\rho^0\right)$ (TEST KULLBACK-LEIBLER DISTANCE, see (5.17)) and $WS\left(\rho^0\right)$ (TEST WEIGHTED SQUARE DISTANCE, see (5.18)) are also put on the eval file. (*Note:* The Kullback -Leibler distance is derived based on the assumption that the b_i in (5.17) are all nonnegative. If this condition is violated, then the value put on the eval file will be "nan.") Unless $\text{fl}(0) = 0$, corresponding values are reported also for each reconstruction after q iterations for which $\text{fl}(q) = \text{fl}(0)$.

If POINT is specified, SNARK14 reports on the values of $E(2^k)^q$ (POINTWISE DISTANCE, see (5.30)), for $1 \leq 2^k \leq$ NELEM, for each reconstruction after q iterations for which $\text{fl}(q) \geq 1$.

In both cases, the reconstructions after the final iteration of an EXECUTE command are always reported (in the same fashion) provided that $\text{fl}(0) \geq 1$.

If BOTH is specified, SNARK14 puts both the reports specified by RESOLUTION as well as POINT on the eval file. This also happens if none of RESOLUTION, POINT, or BOTH is specified.

Remarks: There are two remarks concerning the use of the SELECTIVE command. The first is that the "residual" r^q (even if requested) cannot and will not be computed as the projection data for individual regions are not available from the projection data file prjfil; nor will $E(2^k)^q$ be computed. The second remark concerns the warning message about the specified region not residing entirely within the picture. As explained below, such a warning message could be a false alarm in the case of an elliptical object oriented at an angle to the coordinate axes.

To minimize the search area within the picture for pixels that are within the region, a bounding rectangle that encloses the region is estimated as follows. The smallest bounding rectangle with the same orientation as the elliptical region is first formed (see Figure 5.5). A second bounding rectangle, whose sides are parallel to the picture, that just encloses the first rectangle is then formed. All pixels that are within the given object must therefore fall inside this rectangular boundary and only pixels within this boundary need the rigorous "within region test". If the estimated rectangular boundary is not entirely inside the picture, it is clipped at the picture boundary and a warning message is issued. As illustrated in the figure, because of the estimation, this could happen even if the original elliptical object is entirely within the picture. In general, one can use the reported area (number of pixels within the region) as an index or conduct a simple experiment to verify whether the specified object is entirely within the picture or not.

5.12 DISPLAY

> DISPLAY [SCALE scale][PHANTOM]

This command causes some of the digitized pictures on recfil to be listed on the *Analysis Output file*. The density values of the pixels are interpreted according to the current values of LOFL, UPFL, LOWER and UPPER (see Section 5.2) and, if SCALE is specified, then these are multiplied by the value of the floating point modifier scale prior to printing. *Restriction*: scale > ZERO.

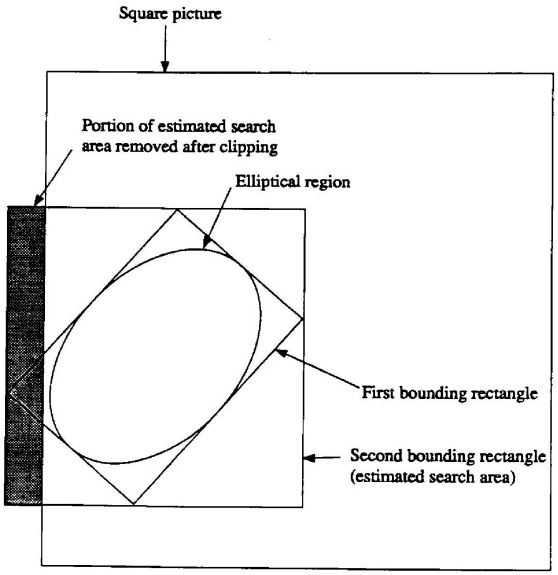

Figure 5.5: Estimation of search area for "within object test".

If PHANTOM is specified, the test phantom is listed, if there is such a phantom on recfil.
The DISPLAY control command line is followed by an

> iteration-flag-line

(see Section 4.5). SNARK14 puts on the Analysis Output file a listing of each reconstruction after q iterations (unless the number of iterations is exactly q, see Section 5.9 for definition) for which $\mathrm{fl}(q) \geq 1$ and puts listings of all final reconstructions of EXECUTE command if $\mathrm{fl}(0) \geq 1$.

5.13 PUNCH

> PUNCH [PHANTOM]

This command causes some of the digitized pictures on recfil to be copied onto the file punch (see Section 3.3). This file is of the same format as the initial part of file11 that is read by PICTURE TEST (see Section 5.4). This command name is an anachronism. The command was originally used to output reconstructions on 80 column Hollerith punched cards. It is now used to output the reconstructions in ASCII in a limited width format.
The PUNCH control command line is followed by an

> iteration-flag-line

(see Section 4.5).

The precise content of the punch file for each picture is determined as follows (consult Sections 5.4 and 5.3 if needed):

- The name-of-the-pattern is that on the recfil if the test phantom is being put on the punch file. If a reconstruction is being put on the punch file, the 80 columns of the name-of-the-pattern on the punch file are put together as:

Column	Contents
1-30	first 30 characters of the name-of-the-execution (see Section 5.8)
31	space
32-34	"alg"
35	space
36-39	value algname from the EXECUTE line (see Section 5.8)
40	space
41-44	"iter"
45	space
46-49	iteration number (see Section 5.8)
50	space
51-80	first 30 characters of the name-of-the-projection (see Section 5.5)

- The command lines of Set 2 of the CREATE follow up sequence are determined by the information on prjfil.
- Set 3 of the CREATE follow up sequence consists of the LAST command line with scale set to 1.0.
- In Set 4 of the CREATE follow up sequence AVERAGE is specified and nave1 is set to 1. The values of nelem and pixel-size are determined by the information on recfil.
- This is followed by the values of the densities in the pixels in the specified format. Prior to being put on the punch file, these densities are interpreted according to the current values of LOFL, UPFL, LOWER, and UPPER (see Section 5.2).

If PHANTOM is specified, the test phantom is put on the punch file. A reconstruction after q iterations (unless the number of iterations is exactly q, see Section 5.9 for definition) is put on the punch file if $\mathrm{fl}(q) \geq 1$, and all final reconstructions of EXECUTE commands are put on the punch file if $\mathrm{fl}(0) \geq 1$ (see Section 4.5).

5.14 LINES

> LINES [SCALE scale][COLUMNS column1 [column2 [column3 [column4]]]]

This command results in a comparison of the exact densities of all the pixels in selected columns of the test phantom and reconstructions. *Restriction*: Test phantom must be present on recfil. The density values of the pixels are first interpreted according to the current values of LOFL, UPFL, LOWER, and UPPER (see Section 5.2) and are then multiplied by the floating point modifier scale prior to the comparisons being made. *Restriction*: scale > ZERO. If SCALE is not specified, scale = 1.

The columns for which the comparisons are to be carried out are specified by the integer modifiers column1, column2, column3, and column4. (*Warning*: If the value of any of these modifiers is outside the range [0, NELEM), then that modifier is ignored.)

The LINES command is followed by an

> iteration-flag-line

(see Section 4.5). Comparisons with the test phantom are made for reconstructions after q iterations (unless the number of iterations is exactly q, see Section 5.9 for definition) whenever $\mathrm{fl}(q) \geq 1$, and comparisons with the test phantom are made for all final reconstructions of the EXECUTE command if $\mathrm{fl}(0) \geq 1$.

The values of the pixels in the specified columns in the test phantom and the reconstruction, as well as their differences, are reported on the Analysis Output file.

5.15 SKUNK

> SKUNK [PHANTOM / DIFFERENCE] [MINIMUM min] [MAXIMUM max] [AMPLITUDE / INTENSITY]

This command causes some of the digitized pictures on recfil to be written to the current directory in PGM format. The densities in the pixels are interpreted according to the current values of LOFL, UPFL, LOWER and UPPER (see Section 5.2). This command name is an anachronism. It was originally used to create pseudo half-tone images on a line printer by means of overprinting. It is now used to create images in the PGM format. The phantom PGM image is named with the first 40 characters of the phantom name with leading and trailing spaces removed as given in the CREATE or PICTURE command. Reconstructions are named as PROJ_REC_ALG_ITER_TYPE_INTAMP.pgm where PROJ is the first 12 characters of the projection data as given in the CREATE or PROJECTION command, REC is the first 12 characters of the name of the reconstruction as given in the EXECUTE command, ALG is the 4 character algorithm name, ITER is the 4 digit iteration number, TYPE is either "r" or "d" for reconstruction or difference, respectively, and INTAMP is either "i" or "a" for intensity or amplitude, respectively. In both cases, embedded spaces in the name are replaced with the underscore character.

If PHANTOM is specified, the test phantom is written, if there is such a phantom on the recfil. If PHANTOM is not specified, only the reconstructions are written.

If DIFFERENCE is specified, reconstructions are reported on by creating the picture whose value in a pixel is the absolute value of the difference between the values in the corresponding pixel of the test phantom and the reconstruction.

If DIFFERENCE is not specified, the reconstructions themselves are written by the method described below.

Restriction: If DIFFERENCE is specified, recfil must contain a test phantom.

The SKUNK control command line is followed by an

> iteration-flag-line

(see Section 4.5). Reconstructions after q iterations (unless the number of iterations is exactly q, see Section 5.9 for definition) are reported on if $\mathrm{fl}(q) \geq 1$, and final reconstructions of EXECUTE commands are reported on if $\mathrm{fl}(0) \geq 1$.

Thus, a SKUNK command results in half-tone displays of a sequence of pictures. We now describe the display method.

Let LO be the value of the floating point modifier min if MINIMUM is specified, or the smallest pixel density in the first picture to be displayed otherwise.

Let HI be the value of the floating point modifier max if MAXIMUM is specified, or the largest pixel density in the first picture to be displayed otherwise.

Restriction: HI - LO > ZERO.

We define the function d by

$$d(x) = \min\left(1, \max\left(0, \frac{x - \mathrm{LO}}{\mathrm{HI} - \mathrm{LO}}\right)\right).$$

For each picture to be displayed we form a new picture by replacing a pixel density x by $d(x)$ if AMPLITUDE is specified, or by $\sqrt{d(x)}$ if INTENSITY is specified. (If neither is specified, AMPLITUDE is assumed.)

5.16 END

> END

This command causes the SNARK14 run to terminate. If the file INPUT does not have this command at the end of it, improper termination of SNARK14 will take place (see Section 4.4).

Chapter 6

IMPLEMENTING USER-DEFINED ALGORITHMS

He had bought a large map representing the sea,
  Without the least vestige of land;
And the crew were much pleased when they found it to be
  A map they all could understand.

"What's the good of Mercator's North Poles and Equators,
  Tropics, Zones, and Meridian Lines?"
So the Bellman would cry: and the crew would reply
  "They are merely conventional signs!

"Other maps are such shapes, with their islands and capes!
  But we've got our brave Captain to thank"
(So the crew would protest) "that he's bought us the best-
  A perfect and absolute blank!"

  Fit the Second - The Bellman's Speech
  The Hunting of the Snark (an agony in eight fits)
  by Lewis Carroll

6.1 Introduction

SNARK14 provides the option for users to define their own reconstruction algorithms (ALP1-ALP5 and ALB1-ALB5, see Section 5.8), termination tests (TRM1 and TRM2, see Section 5.9) and Superiorization-related functions (SCR3-SCR5, NAV3-NAV5, see Section 5.10). In this introduction we describe how SNARK14 makes use of these algorithms. Suppose that SNARK14 encounters an EXECUTE command (Section 5.8) on the INPUT file. This will cause a subroutine (named `exalg`) to be called, which controls the execution of the reconstruction algorithm. We now describe those actions of `exalg` that are relevant to the user-defined algorithms. Some of these actions are influenced by the last STOP command (Section 5.9) on the INPUT file prior to the EXECUTE command. We shall refer to this as the active STOP command. If there is no STOP command prior to the EXECUTE command under consideration,

STOP ITERATION = 1

is assumed.

`exalg` has a list of the algorithms that can perform blob reconstructions. ALB1-ALB5 are in the list and ALP1-ALP5 are not. If BASIS BLOB is in effect and an algorithm that is in the list is invoked, a blob reconstruction is performed. In all other cases a pixel reconstruction is performed.

All SNARK14 algorithms are descendants of the C++ class `alg_class`:

```
class alg_class
{
private:
public:
  virtual INTEGER Init();
  virtual BOOLEAN Run(REAL* recon, INTEGER* list, REAL* weight, INTEGER iter);
  virtual INTEGER Reset();
};
```

Default (do nothing) implementations of these methods are provided by `alg_class`.

All SNARK14 termination tests are descendants of the C++ class `termtest_class`:

```
class termtest_class
{
public:
  virtual void Init();
  virtual BOOLEAN Run(REAL* recon, INTEGER* list, REAL* weight, INTEGER iter);
};
```

Default (do nothing) implementations of these methods are provided by `termtest_class`. The `Init` method is invoked by the STOP command (see Section 5.9) for the purpose of collecting user-defined parameters for the termination test and the `Run` method is invoked during algorithm execution as described below.

Executing a reconstruction algorithm involves the following steps:

Step (i). If this is the first EXECUTE command in the INPUT file, or if CONTINUE is not specified on the EXECUTE control command line, an array of size `GeoPar.area` (see Section 6.3.3), which is used to hold the reconstruction, is initialized (see Section 5.8). (Otherwise, this array will contain the final reconstruction of the last EXECUTE command). If this is the first blob reconstruction (see Section 5.7), a second array of size `Blob.area` (see Section 6.3.4) is initialized. If both the previous and current algorithms use blobs and CONTINUE is not specified, the Blob.recon array is deleted and reallocated. Also, an integer variable that counts the number of iterations is initialized to 1.

If CONTINUE is specified and the last algorithm returned a blob reconstruction and the current algorithm is performing a pixel reconstruction, the blob reconstruction is converted to a pixel reconstruction. Conversely, if the last algorithm returned a pixel reconstruction and the current algorithm is performing a blob reconstruction, the pixel reconstruction is converted to a blob reconstruction. If the last algorithm returned a blob reconstruction and the current algorithm is performing a blob reconstruction, the blob parameters must be the same for both executions.

Step (ii). `exalg` invokes the `Init` method of the instance of the algorithm class associated with the algorithm name on the EXECUTE command. The `Init` method can be used to parse any parameters needed by the algorithm, allocate any resources that may be needed by the algorithm and must persist from one iteration to the next. The most common resources are dynamic memory and files used to store temporary data. It is also a good place to initialize the dynamic memory if the values do not change from iteration to iteration. The `Init` method is expected to return 0 on success and a non-zero value if the initialization failed.

Step (iii). `exalg` calls the `Run` method of the instance of the algorithm class. `exalg` passes to `Run` the array of values of the basis functions in the reconstruction (the name of this array is `recon`) and the iteration number (the name of this variable is `iter`). `recon` is either the pixel reconstruction array or the blob reconstruction array, depending on whether or not a blob reconstruction is being performed. In addition,

`exalg` passes two arrays, `list` and `weight`, which are large enough to hold the longest sequences that can be returned by `wray`, `ray`, `bwray`, or `bray` (whichever is appropriate for the current situation, see Section 6.4). The `exalg` subroutine can invoke extra commands when the SUPERIORIZE command is appropriately used in the input file, see 5.10, and the way the extra calls take place is explained below.

Step (iv). `exalg` checks whether any of the following is the case: (a) `Run` returned the value TRUE (this is used to indicate that the algorithm is not applicable to the specified situation and so the values in `recon` are not the values of a reconstruction), (b) the active STOP command has ITERATION specified and the iteration number is the same as the value of the integer modifier `iter` on the active STOP command, (c) the active STOP command has TERMINATION specified, and when the `Run` method of the class whose name is the value of test-name on the active STOP command is called, the value TRUE is returned (this function is provided with the array of pixel or blob values in the reconstruction and the iteration number).

If any of the three conditions (a), (b), or (c) is satisfied, `exalg` calls the `Reset` method and returns control to the main program of SNARK14, which moves on to the next command on the INPUT file. If none of the three conditions (a), (b), or (c) is satisfied, the iteration number is increased by 1 and Step (iii) is repeated. The `Reset` method is where the resources allocated by the `Init` method are released. The `Reset` method is expected to return 0 for success and a non-zero value for failure.

After the initialization in Step (i), the only changes to the array of basis function values in the reconstructions are done in Step (iii) by `Run`. If the iteration number is q when `Run` is called, `Run` returns via `recon` the reconstruction after q iterations (see Section 5.8).

Information is passed to the user-defined algorithms both via the parameters in the formal parameter list and via classes accessed through global variables. Typically such an algorithm begins with some subset of the following statements. (Classes that are not used in the algorithm can of course be omitted.)

```
#include "anglist.h"
#include "blkdta.h"
#include "blob.h"
#include "consts.h"
#include "geom.h"
#include "inputfile.h"
#include "modefl.h"
#include "noise.h"
#include "objects.h"
#include "projfile.h"
#include "spctrm.h
#include "uiod.h"
```

The execution of an iterative reconstruction algorithm using PIXEL basis functions may be affected by the command SUPERIORIZE. If such a command appears in the INPUT prior to an EXECUTE command that calls for an the iterative algorithm, then that iterative algorithm is superiorized as explained in Section 5.10. The SUPERIORIZE command invokes a secondary optimization criterion, which may be one of the built-in choices (TVAR or SMOO) or may be user defined (SCR3, SCR4, or SCR5). The user-defined secondary optimization criteria are descendants of the C++ class `sec_cri_class`:class sec_cri_class

```
{
private:
public:
  virtual void Init();
  virtual REAL Run(REAL* image);
};
```

Default (do nothing and return the value 0.0) implementations of the user-defined secondary criteria are provided by `sec_cri_class` when using SCR3, SCR4, and SCR5.

The method `Init` of any descendant of the `sec_cri_class` is also executed in *Step (ii)* and the implementation of this method can be used in a similar fashion as the `Init` method of class `alg_class`.

When the SUPERIORIZE command is used, the execution of the `Run` command of the class `alg_class` is preceded by a superiorization loop, see lines 8 - 18 of Algorithm 5.1. The `Run` method of the instance of

the class `sec_cri_class` is invoked several times, at least $N+1$ times, by `exalg` during this superiorization loop, see Section 5.10, in the following manner: *a*) just before starting the superiorization loop, the `exalg` subroutine invokes the `Run` method and passes the array of values of the pixel basis functions in the reconstruction and *b*) in the nth iteration of the superiorization loop `exalg` calls the `Run` method and passes the array of values of the pixel basis functions in the perturbed reconstruction; in both cases the array passed to the `Run` method is `image`. It is important to emphasize that any implementation of the sec_cri_class should provide a `Run` method that returns an appropriate and useful `REAL` value.

The superiorization process requires both a secondary optimization criterion function and a procedure to obtain nonascending vectors for it; see Section 5.10. When invoking the SUPERIORIZE command together with a built-in (TVAR or SMOO) secondary optimization criterion, the procedure to obtain a nonascending vector is already defined. When invoking the SUPERIORIZE command together with a user-defined (SCR3, SCR4, or SCR5) secondary optimization criterion, the procedure to obtain a nonascending vector must be defined by the user as well. The user should use the implementations in NAV3, NAV4 and NAV5 corresponding to the secondary optimization criteria SCR3, SCR4 and SCR5, respectively. In SNARK14, all the procedures to obtain the appropriate nonascending vector are descendant of the C++ class `nav_class:`

```
class nav_class
{
private:
public:
  virtual void Init();
  virtual void Run(REAL* xkn, REAL* vkn);
};
```

Default (return a nonascending vector whose elements are all zero) implementations of these methods are provided by `nav_class` when using NAV3, NAV4, and NAV5.

The method `Init` of any descendant of the `nav_class` is also executed in *Step (ii)* and the implementation of this method can be used in a similar fashion as the `Init` method of class `alg_class`.

The `Run` method of the instance of the class `nav_class` is invoked N times by line 9 of Algorithm 5.1. In the nth iteration of the superiorization loop `exalg` calls the `Run` method and passes the array of values of the pixel basis functions in the superiorized reconstruction (the name of this array is `xkn`). Before returning control to the `exalg` subroutine, the `Run` method must return an array representing the nonascending vector for the user-defined secondary optimization criterion (the name of this array if `vkn`).

6.2 Formal parameter lists of the user-defined algorithms

The formal parameter list for the `Run` method for a reconstruction algorithm class or a termination test class is:

`(REAL* recon, INTEGER* list, REAL* weight, INTEGER iter)`

where the types `REAL` and `INTEGER` are defined in Section 6.3.1.

We now explain the nature and purpose of these parameters. `recon` is a real array. The size of `recon` is either `GeoPar.area` or `Blob.area` depending on whether this is a pixel reconstruction or not. `recon[0]` is the first element of this array. Each time the user algorithm executes, the reconstructed picture must be stored in the `recon` array. It is the user's responsibility to map the reconstructed picture into the linearized `recon` array according to the following numbering scheme.

For a pixel reconstruction, if $\rho_{i,j}$ is the density of a pixel in the ith row and jth column of the reconstructed picture, then $\rho_{i,j}$ will map into the `recon` array element `recon[`$i\times$NELEM+j`]`. The coordinates of the center of the pixel are $((j-c)\times \text{PIXSIZ}, (c-i)\times \text{PIXSIZ})$, where c is defined in Equation (2.4) on page 14.

For a blob reconstruction, the coordinates of the centers of a blob are given in Equation (2.6) on page 14. Let $M_2 = \lfloor \text{NELEM}\times\text{PIXSIZ}/\text{DELTA}\rfloor$, $H_2 = \lfloor \text{NELEM}\times\text{PIXSIZ}/(\sqrt{3}\times\text{DELTA})\rfloor$, $M = 2\times M_2+1$, and $H = 2\times H_2+1$. ($\lfloor x\rfloor$ is the greatest integer not greater than x.) Then the dimension of the `recon` array is $M\times H$ and the (m,n) blob of Equation (2.6) is the $((n+H_2)\times M+m+M_2)$th element of the `recon` array. Note that the restriction that $m+n$ is even results in recon being twice as big as it needs to be. Those elements corresponding to $m+n$ odd are unused.

At the end of each execution of the user-defined `Run` method the `recon` array is automatically written to recfil by SNARK14, unless the `Run` method returns the value TRUE (see Section 6.1). `list` is an integer array (whose size is computed internally by SNARK14) used in `wray`, `ray`, `pseudo`, `bwray`, `bray`, and `bpseudo` (see Section 6.4) to store the indices of the basis functions intersected by a specified ray. `weight` is a real array (whose size is equal to the size of the `list` array) used in `wray`, `ray`, `pseudo`, `bwray`, `bray` and `bpseudo` (see Section 6.4) to store the integrals of the ray across the intersected basis functions in the order given in the `list` array. The values stored in `list` and `weight` are not used by `exalg` (see Section 6.1), they are purely internal to the user-defined algorithm. They are in the formal parameter list to remove the burden from the user of having to allocate space in the `Init` method for `list` and `weight` when `wray`, `ray`, `pseudo`, `bwray`, `bray`, or `bpseudo` is used by this algorithm. `iter` is an integer variable whose value is the iteration number (see Section 6.1).

Warning: The values of `recon` should not be altered by a user-defined termination test. Alteration of these values by the termination test may produce results quite different from what the user intended.

6.3 C++ classes in user-defined algorithms

We now explain the contents of each of the C++ classes that may be used in a user-defined algorithm to pass information from the rest of the program to the algorithm. Each of these classes is a singleton [16], although they do not follow the singleton pattern. (A singleton is a class that has at most one instance.) Access to the single instance of each class is provided through a global variable. *Warning*: The variables in these classes should not be altered in any way by the user-defined algorithm. Alterations of these variables by the algorithm may produce results quite different from what the user intended. Creating new instances of the classes is not recommended as they will have been properly initialized (inheriting the initialization of the variables from the first instance of the class), which again may produce results quite different from what the user intended.

The C++ code listed for each class is not exactly what is contained in the actual header file for the class. Attributes and methods that are for internal SNARK14 use have been deleted. The order may also be changed for a clearer description.

6.3.1 blkdta.h

```
#define REAL double
#define INTEGER int
#define BOOLEAN bool
#define CHAR char
#define TRUE true
#define FALSE false
#define CHAR2INT(CHAR1, CHAR2, CHAR3, CHAR4) \
        ( (((INTEGER)(CHAR1))<<24) | (((INTEGER)(CHAR2))<<16) | \
        (((INTEGER)(CHAR3))<<8) | (((INTEGER)(CHAR4))) )
#define SQR(NUM1) ((NUM1)*(NUM1))
#define CA(ARRAY, ELEMENT, INDEX) ARRAY[(ELEMENT) + 2 * (INDEX)]
#define MIN0(ARG1, ARG2) (((ARG1) > (ARG2)) ? (ARG2) : (ARG1))
#define MAX0(ARG1, ARG2) (((ARG1) < (ARG2)) ? (ARG2) : (ARG1))
#define round(X) (floor((X) + 0.5))
extern INTEGER trace;
```

`blkdta.h` defines a number of macros that are used throughout the SNARK14 program. The macros `REAL`, `INTEGER`, `BOOLEAN`, `CHAR`, `TRUE`, and `FALSE` are clear from their names. `CHAR2INT` packs a sequence of four characters in an integer to simplify comparisons with the four character keywords used in SNARK14. `SQR` computes the square of its argument. `CA` is used to access the real or complex components (`ELEMENT`) of a complex `ARRAY`. `MIN0` and `MAX0` return the minimum and maximum of the two arguments, respectively. `round` returns the nearest integer to its argument. The reference to the global `trace` variable is defined here.

It contains the value contained on the last TRACE command (see Section 5.1). A typical user algorithm or termination test trace statement follows the pattern:

```
#include "uiod.h"

if (trace > 0) fprintf(output, "format", args...);
```

where the file `output` is defined in `uiod.h`, *`format`* is replaced by the appropriate C++ format string and *`args...`* by the arguments needed by the format string. The convention in SNARK14 is to put the newline character (`\n`) at the beginning of the format string.

6.3.2 consts.h

```
extern class Consts_class
{
public:
  static const REAL    zero;
  static const REAL    infin;
  static const REAL    pi;
  static const REAL    twopi;
  static const REAL    pisq;
  static const REAL    pid2;
  static const REAL    sqrt3;
  static const REAL    isqrt3;
} Consts;
```

`zero` and `infin` are the smallest and largest positive real numbers that can be used (see Section 2.1). In other sections of this manual, they have been referred to with the names ZERO and INFIN. The other constants `pi`, `twopi`, `pisq`, `pid2`, `sqrt3`, and `isqrt3` are floating point approximations to the mathematical constants π, $2 \times \pi$, π^2, $\pi/2$, $\sqrt{3}$, and $1/\sqrt{3}$, respectively; in Section 2.1 we used upper-case letters for naming these.

6.3.3 geom.h

```
extern class Geometry_class
{
public:
  INTEGER nelem;
  INTEGER area;
  INTEGER midpix;
  REAL pixsiz;
  INTEGER prjnum;
  REAL pinc;
  INTEGER usrays;
  INTEGER snrays;
  INTEGER nrays;
  INTEGER midray;
  INTEGER fsnray;
  INTEGER lsnray;
  INTEGER fusray;
  INTEGER lusray;
  BOOLEAN div;
  BOOLEAN tang;
  BOOLEAN arc;
  REAL radius;
  REAL stod;
```

```
    BOOLEAN par;
    BOOLEAN uni;
    BOOLEAN vri;
    BOOLEAN strip;
    BOOLEAN line;
    INTEGER nave2;
    INTEGER naper[13];
    REAL aveden;
    INTEGER numray(INTEGER melen, REAL zisxip);
} GeoPar;
```

This class describes the image and projection geometries.

`nelem` is the number of elements in the reconstruction grid (see Section 2.2). In other sections of this manual, this has been referred to as NELEM. The value of `nelem` is determined by the PICTURE command (see Section 5.4). `area` is `nelem*nelem` and `midpix` is (`nelem`-1)/2.

`pixsiz` is the length of the side of a pixel (see Section 2.2). In other sections of this manual, it is referred to as PIXSIZ. The value of `pixsiz` is determined by the PICTURE command (see Section 5.4). `prjnum` is the number of projections and `pinc` is (except in the VARIABLE case) the spacing between the detectors (see Section 2.4). In other sections of this manual, they are referred to as PRJNUM and PINC. The values of `prjnum` and `pinc` are determined by the PROJECTION command (see Section 5.5).

`usrays` is the number of rays along which data have been collected (see Section 2.5). `usrays` is determined by the PROJECTION command (see Section 5.5). `snrays` is the number of rays needed to cover the reconstruction region. This is calculated by SNARK14 based on the values of `nelem`, `pixsiz` and on the data collection geometry (see Section 2.5). `nrays` is the number of rays in a projection. It is the maximum of `usrays` and `snrays`. `midray` is (`nrays`-1)/2. In the numbering of the rays in the projection from 0 to `nrays`-1, `fsnray` and `lsnray` are respectively the numbers of the first and last ray needed to cover the reconstruction region, and `fusray`, and `lusray` are respectively the numbers of the first and last ray along which data have been collected. The values of these variables are determined by SNARK14 according to formulas given in Section 2.5. In other sections of the manual, `usrays`, `snrays`, `nrays`, `fsnray`, `lsnray`, `fusray`, and `lusray` have been referred to with the names USRAYS, SNRAYS, NRAYS, FSNRAY, LSNRAY, FUSRAY, and LUSRAY, respectively.

`div` is a logical variable whose value is TRUE if, and only if, the geometry of data collection is DIVERGENT (see Section 2.4). If `div` is TRUE, the values of the variables `tang`, `arc`, `radius`, and `stod` have the following interpretation. `tang` or `arc` is TRUE if, and only if, the detectors are spaced along a TANGENT line or an ARC of a circle, respectively. `radius` is the radius of the circle on which the source is positioned and `stod` is the source-to-detector distance (see Section 2.4). The values of these variables are determined by the PROJECTION command (see Section 5.5). If `div` is FALSE, `tang`, `arc`, `radius` and `stod` are assigned default values (FALSE and 0). `par` is a logical variable whose value is TRUE if, and only if, the geometry of data collection is PARALLEL. If `par` is TRUE, the values of the logical variables `uni`, `vri`, `strip`, and `line` are TRUE if, and only if, the ray spacing is UNIFORM, the ray spacing is VARIABLE, the rays are STRIPS and the rays are LINES, respectively (see Sections 2.3 and 2.4). The values of these variables are determined by the PROJECTION command (see Section 5.5). If `par` is FALSE, `uni`, `vri`, `strip`, and `line` are assigned the default value FALSE. The variable `nave2` and array `naper[13]` (only the first `nave2` elements are defined) describe the aperture function of the detector (see Section 5.3.3, S 5). Their physical interpretation is explained in Section 5.3.2. Their values are determined by the PROJECTION command (see Section 5.5).

`aveden` is an estimate, based on the projection data, of the average density of the picture to be reconstructed. This estimation is done under the control of the PROJECTION command (see Section 5.5). Let SUMRAY denote the sum of the ray sums (divided by the width of the strips in the STRIP case) that are stored on prjfil. Let SUMLENGTH denote the sum of the lengths of intersections of the reconstruction region with the rays (central lines of the rays in the STRIP case) that contribute to SUMRAY. Then `aveden` = SUMRAY/SUMLENGTH. In other sections of this manual, the variables `radius`, `stod`, `nave2`, `naper`, and `aveden` have been referred to as RADIUS, STOD, NAVE2, NAPER, and AVEDEN respectively. (*Warning*: `aveden` is only an estimate of the average density in the picture. It may be a very bad estimate if it is based

on too few projections, or if the region of the picture from which projection data have been collected is larger than the reconstruction region (see Section 2.2)).

6.3.3.1 numray

```
INTEGER numray(INTEGER melen, REAL zisxip);
```

`numray` returns the number of rays necessary to span a square shaped picture region whose center is the origin, and whose sides, which are parallel to the axes of the coordinate system, are of length `melen`×`zisxip`. The precise rules for calculating this number of rays are given in Section 2.5. (In that discussion `melen` is NELEM, `zisxip` is PIXSIZ and the number of rays is SNRAYS). *Restriction*: `melen` > ZERO and `zisxip` > ZERO.

6.3.4 blob.h

```
extern class Blob_class
{
public:
  BOOLEAN pix_basis;   //true if pixel basis, false if blob basis
  REAL    support;     // support of a blob
  REAL    shape;       //  shape of a blob
  REAL    delta;       // resolution of the (hexagonal) grid
  INTEGER M;           // number of underlying square grid points in the horizontal direction
  INTEGER H;           // number of underlying square grid points in the vertical direction
  INTEGER area;        // M*H
  REAL *recon;         // array of the blob coefficients
  void blob2pix(REAL *blob_array, REAL *pict);
  void pix2blob(REAL* pict, REAL* blob_array);
  void bwray(
     INTEGER np,
     INTEGER nr,
     INTEGER* list,
     REAL* weight,
     INTEGER* numb,
     REAL* snorm
  );
  void bray(
     INTEGER np,
     INTEGER nr,
     INTEGER* list,
     REAL* weight,
     INTEGER* numb,
     REAL* snorm
  );
  REAL bpseudo(
     REAL* pict,
     INTEGER np,
     INTEGER nr,
     INTEGER* list,
     REAL* weight,
     INTEGER* numb,
     REAL* snorm,
     BOOLEAN line,
     BOOLEAN constr
  );
```

```
    void bsmooth(
       REAL* a,
       INTEGER H,
       INTEGER M,
       REAL thresh,
       REAL w1,
       REAL w2
    );
  } Blob;
```

This class describes the blobs that may be used in a reconstruction. The class is only usable when `pix_basis` is FALSE.

`support`, `shape`, and `delta` are described in Section 2.2. In other sections of this manual, they are referred to as SUPPORT, SHAPE, and DELTA. The blob reconstruction region, `recon`, has `H` rows and `M` columns. The size of the blob `recon` array, `area`, is `H`×`M`. The following C++ code illustrates how to access the elements of the blob `recon` array:

```
INTEGER lhf = (INTEGER)(-Blob.H / 2));
INTEGER lhl = (INTEGER)((Blob.H - 1) / 2);
INTEGER lmf = (INTEGER)(-Blob.M / 2);
INTEGER lml = (INTEGER)((Blob.M - 1) / 2);
for (INTEGER h = lhf; h <= lhl; ++h) {
   for (INTEGER m = lmf; m <= lml; ++m) {
      if ((m + h) % 2 == 0) {
         // m and h must have the same parity!!
         ind = (h + Blob.H2) * Blob.M + (m + Blob.M2);
         // do blob processing here
         // ind is the index of the blob in Blob.recon
      }
   }
}
```

Note: The variable `h` in the code above processes the blobs from bottom to top; that is in the same direction as the y-axis. This is the opposite of how the pixel array is stored.

The functions `bwray`, `bray`, and `bpseudo` that are specified below are blob equivalents of the pixel functions `wray`, `ray`, and `pseudo`, respectively (see Sections 6.4.2, 6.4.3, and 6.4.4 below).

6.3.4.1 blob2pix

```
void blob2pix(REAL *blob_array, REAL *pict);
```

`blob2pix` is used to convert a blob image to a pixel image. It is assumed that the blob image, `blob_array`, is created with the blob parameters in the Blob class and that the `pict` array can hold at least NELEM × NELEM pixels.

6.3.4.2 pix2blob

```
void pix2blob(REAL* pict, REAL* blob_array);
```

`pix2blob` is used to convert a pixel image to a blob image. It is assumed that the pixel image, `pict`, contains NELEM × NELEM pixels and that the blob array can hold at least `Blob.area` blobs.

6.3.4.3 bwray

```
void bwray(
     INTEGER np,
```

```
        INTEGER nr,
        INTEGER* list,
        REAL* weight,
        INTEGER* numb,
        REAL* snorm
    );
```

`bwray` is the blob equivalent of `wray` (see Section 6.4.2). For the npth projection and nrth ray, this routine calculates the trace of a LINE ray through the picture region. *Restriction*: $0 \leq np <$ PRJNUM and $0 \leq nr <$ NRAYS. *Warning*: In case of STRIP rays, the subroutine will return values that are appropriate for LINE rays located at the centers of the actual STRIP rays.

`bwray` returns the following information:

`numb` For rays that intersect the picture region, this is the number of blobs that are intersected by the ray. For rays that do not intersect the picture region, the value of numb is set to 0.

`list[0], ..., list[numb-1]` The indices of the blobs intersected by the ray. I.e., if j = `list[`i`]`, then `Blob.recon[`j`]` is the present density in the ith blob intersected by the ray (see Section 6.2).

`weight[0], ..., weight[numb-1]` The LINE integrals of the ray through the blobs in the order given in `list`.

`snorm` The sum of the squares of the above weights.

6.3.4.4 bray

```
    void bray(
        INTEGER np,
        INTEGER nr,
        INTEGER* list,
        REAL* weight,
        INTEGER* numb,
        REAL* snorm
      );
```

`bray` is the blob equivalent of `ray` (see Section 6.4.3). For the npth projection and nrth ray, this routine calculates the trace of a STRIP ray through the picture region and includes those blobs whose centers are in the strip. *Restriction*: $0 \leq np <$ PRJNUM and $0 \leq nr <$ NRAYS. *Warnings:* The assumptions made about strip ray projections are inappropriate for blobs and can result in inaccurate reconstructions. In case of PARALLEL LINE rays, the subroutine will return the values that are appropriate for PARALLEL STRIP rays whose central lines are the actual LINE rays (see Section 2.3). If `bray` is called in the case of DIVERGENT rays, the SNARK14 run will be terminated.

`bray` returns the following information:

`numb` Number of blobs whose centers are in the ray. When `numb` = 0, the ray in question does not contain the center of any blob.

`list[0], ..., list[numb-1]` The indices of the pixels whose centers are in the strip ray. I.e., if j = `list[`i`]`, then `Blob.recon[`j`]` is the present density in the ith blob whose center is in the ray (see Section 6.2).

`weight[0], ..., weight[numb-1]` Are all assigned the value of the integral of a blob.

`snorm` Is assigned the value of integral of a blob squared times `numb`.

6.3.4.5 bpseudo

```
REAL bpseudo(
     REAL* pict,
     INTEGER np,
     INTEGER nr,
     INTEGER* list,
     REAL* weight,
     INTEGER* numb,
     REAL* snorm,
     BOOLEAN line,
     BOOLEAN constr
  );
```

`bpseudo` is the blob equivalent of `pseudo` (see Section 6.4.4). Please refer to that section for the meaning of the arguments.

6.3.4.6 bsmooth

```
void bsmooth(
     REAL* a,
     INTEGER H,
     INTEGER M,
     REAL thresh,
     REAL w1,
     REAL w2
  );
```

`bsmooth` is the blob equivalent of `smooth` (see Section 6.4.10). The blob array is assumed to contain `H` $\times$ `M` blobs. `thresh` acts as described in Section 5.8. The main difference between `bsmooth` and `smooth` is that each blob only has six neighbors. The weight `w1` is applied to the central blob and the weight `w2` is applied to its six neighbors.

6.3.5 projfile.h

```
extern class SnarkPrjFile : private DIGFileSnarkProj
{
public:
        INTEGER ReadProj(INTEGER np, REAL* data, unsigned INTEGER nsize);
} ProjFile;
```

This class provides access to the projection data file, prjfil.

6.3.5.1 ReadProj

```
INTEGER ReadProj(INTEGER np, REAL* data, unsigned INTEGER nsize);
```

Roughly speaking, `ReadProj` copies from prjfil (see Sections 3.2 and 5.5) the ray sums for the npth projection into the array `data`. *Warning*: The size of `data` must be at least `nsize`.

More precisely, the first `nsize` values of `data` are changed. Let prjfil(np, J) denote the$J$th ray sum stored on prjfil for the npth projection. (Due to our numbering convention of rays, this is in fact the ray sum for the (`fusray` + J)th ray in the npth projection; see Section 2.5.) We distinguish between two cases.

Case 1. `nsize` < `usrays` (see Section 2.5). In this case we define

$$\text{iskip} = \lfloor (\text{usrays} - \text{nsize})/2 \rfloor$$

and, for $0 \leq i <$ `nsize`,

$$\text{data}[i] = \text{prjfil}(\text{np}, \text{iskip} + i).$$

Case 2. `nsize` $\geq$ `usrays`. In this case we define

$$\text{iskip} = \lfloor (\text{nsize} - \text{usrays})/2 \rfloor$$

and, for $0 \leq i <$ `nsize`,

$$\text{data}[i] = \begin{cases} \text{prjfil}(np, i - \text{iskip}), & \text{if iskip} \leq i < \text{iskip} + \text{usrays}, \\ 0 & \text{otherwise.} \end{cases}$$

6.3.6 anglst.h

```
extern class anglst_class
{
public:
  void getang(INTEGER np, REAL* theta, REAL* sinth, REAL* costh);
  REAL prdta(INTEGER np, INTEGER nr);
} Anglst;
```

This class provides access to the projection angle data (`getang`) and individual ray measurements (`prdta`).

6.3.6.1 getang

```
void getang(INTEGER np, REAL* theta, REAL* sinth, REAL* costh);
```

`getang` returns the values of `theta` in radians for the *np*th projection and the values of the sine and cosine of `theta` (see Figures 2.1 and 2.2), `sinth` = sin(`theta`) and `costh` = cos(`theta`). In other sections of the manual, `theta` is referred to as THETA.

6.3.6.2 prdta

```
REAL prdta(INTEGER np, INTEGER nr);
```

`prdta` returns as its value the ray sum for the *nr*th ray of the *np*th projection, as it is stored on prjfil (see Sections 3.2.1 and 5.5). The numbering of the rays in a projection is from 0 to NRAYS-1, as is explained in Section 2.5. prjfil only contains ray sums for `fusray` $\leq nr \leq$ `lusray`. If $0 \leq nr <$ `fusray` or `lusray` $< nr <$ `nrays` the function `prdta(`*np*`,` *nr*`)` returns the value 0. *Restrictions*: $0 \leq np <$ PRJNUM and $0 \leq nr <$ NRAYS.

6.3.7 infile.h

```
#include "inputfile.h"
extern class InFile_class: public InputFile_class
{
public:
} InFile;
```

This class is used for processing the SNARK14 INPUT file. The methods of interest to the algorithm developer are contained in the super class, `InputFile_class` (see Section 6.3.8).

6.3.8 inputfile.h

```
class InputFile_class
{
public:
  INTEGER getwrd(BOOLEAN NewLine, BOOLEAN* eol, const INTEGER *expect, int nexpect);
  REAL getnum(BOOLEAN NewLine, BOOLEAN* eol);
  INTEGER getint(BOOLEAN NewLine, BOOLEAN* eol);
  void getnxt(BOOLEAN flag);
};
```

This class maintains a buffer that contains the current command line being parsed and a pointer to the next character in that line to be processed. The `getwrd`, `getnum`, and `getint` methods return the next word modifier, real number, and next integer, respectively. `getnxt` causes the next command line to be read into the buffer.

6.3.8.1 getnxt

```
void getnxt(BOOLEAN flag);
```

This subroutine reads the next command line (up to 80 characters) from the INPUT file into the buffer and initializes the "scan pointer" to the beginning of the buffer. If the command line is longer than 80 characters, the excess characters are discarded. If the value of the logical variable `flag` is FALSE, then the output file will carry an echo of the characters read. If the `flag` is TRUE, then the OUTPUT file will not echo the characters read by this subroutine.

6.3.8.2 getwrd

```
INTEGER getwrd(BOOLEAN NewLine, BOOLEAN* eol, const INTEGER *expect, int nexpect);
```

This function returns the first four characters of a word modifier (see Section 4.4) packed in integer.

If `NewLine` is TRUE, a new command line is read using `getnxt` before continuing. The current line buffer is searched for a word modifier. The `expect` argument is an array of word modifiers packed as integers (see `CHAR2INT` in Section 6.3.1). `nexpect` is the size of the expect array. The current line buffer is searched from the present position of the scan pointer until either a word modifier in the `expect` array is found or the end of the buffer is reached. In the former case, `getwrd` returns the first four characters of the word modifier packed into an integer, sets `eol` to FALSE, and places the scan pointer at the end of the word modifier. In the latter case, `getwrd` returns blanks (a string of four spaces packed into an integer), sets `eol` to TRUE, and places the scan pointer at the end of the buffer. `getwrd` will skip over words that are not in the `expect` array.

6.3.8.3 getnum

```
REAL getnum(BOOLEAN NewLine, BOOLEAN* eol);
```

This function returns the value of a floating point modifier (see Section 4.4). *Restriction*: The floating point modifier must be in the precise format described in Section 4.4.

If `NewLine` is TRUE, a new command line is read using `getnxt` before continuing. The current line buffer is searched from the present position of the scan pointer until either a floating point modifier is found or the end of the buffer is reached. In the former case, `getnum` returns the value of the floating point modifier, sets `eol` to FALSE, and places the scan pointer at the end of the floating point modifier. In the latter case, `getnum` returns 0.0, sets `eol` to TRUE, and places the scan pointer at the end of the buffer.

6.3.8.4 getint

```
INTEGER getint(BOOLEAN NewLine, BOOLEAN* eol);
```

This function returns the value of an integer modifier (see Section 4.4). *Restriction*: The number of digits of the integer modifier must be no more than 9.

If `NewLine` is TRUE, a new command line is read using `getnxt` before continuing. The current line buffer is searched from the present position of the scan pointer until either an integer modifier is found or the end of the buffer is reached. In the former case, `getint` returns the value of the integer modifier, sets `eol` to FALSE, and places the scan pointer at the end of the integer modifier. In the latter case, `getint` returns 0, sets `eol` to TRUE, and places the scan pointer at the end of buffer.

6.3.9 noise.h

```
extern class Noise_class
{
public:
  INTEGER quanin;
  REAL    quanmn;
  REAL    quancm;
  BOOLEAN sctnfl;
  REAL    sctnpk;
  REAL    sctnwd;
  BOOLEAN ultnfl;
  REAL    ultnmn;
  REAL    ultnsd;
  BOOLEAN addnfl;
  REAL    addnmn;
  REAL    addnsd;
} NoisePar;
```

These attributes describe the nature of noise that exists in the projection data. In other sections of this manual `quanin`, `quanmn`, `quancm`, `sctnpk`, `sctnwd`, `ultnmn`, `ultnsd`, `addnmn`, and `addnsd` are referred to as QUANIN, QUANMN, QUANCM, SCTNPK, SCTNWD, ULTNMN, ULTNSD, ADDNMN, and ADDNSD, respectively. Their meanings are given in Section 5.3.2 and in Section 5.3.3 as part of the description of command line 7.2. The values of the variables are determined by the PROJECTION command (see Section 5.5). *Warning*: The values of `ultnmn` and `addnmn` refer to the original projection data; the data stored on prjfil are corrected for bias (see Section 5.5).

6.3.10 spctrm.h

```
extern class spctrm_class
{
public:
  INTEGER nergy;
  INTEGER energy[7];
  REAL    engwt[7];
  REAL    backgr[7];
} Spctrm;
```

These attributes describe the energy distribution in the x-ray beam that was used to collect the data and the nature of the background material that was used for the calibration measurements. This is explained in detail in Section 5.3.2. The values of the variables are determined by the PROJECTION command (see Section 5.5) as follows.

`nergy` is 1 if MONOCHROMATIC is specified on command line 2.1 (see Section 5.3.3) and `nergy` is the value of nergy if POLYCHROMATIC is specified on command line 2.1. In other sections of this manual, it is referred to as NERGY.

In the MONOCHROMATIC case, `energy[0]` has the value of energy on command line 2.1 and `energy[i]` = 0 for $0 < i < 7$. In the POLYCHROMATIC case, `energy[i]` has the value of energy(i+1) on command line 2.2 for $0 \leq i <$ nergy and 0 for nergy $\leq i < 7$.

In the MONOCHROMATIC case, `engwt[0]` = 1 and `engwt[i]` = 0 for $0 < i < 7$. In the POLYCHROMATIC case, `engwt[i]` is percent(i+1)/100 from command line 2.2 for $0 \leq i <$ `nergy`, and 0 for `nergy` $\leq i < 7$.

6.3.11 modefl.h

```
extern class modefl_class
{
public:
  BOOLEAN lofl;
  REAL    lower;
  BOOLEAN upfl;
  REAL    upper;
} Modefl;
```

The values of these attributes are assigned by the most recent MODE command. See Section 5.2 for details. In other sections of this manual, `lofl`, `lower`, `upfl`, and `upper` are referred to as LOFL, LOWER, UPFL, and UPPER, respectively.

6.4 Service routines available to the user

In addition to the methods contained in the various classes described above, there are a number of basic operations that are not associated with any class. These operations are coded in separate subroutines, which are then called by the reconstruction algorithm when needed. Since these subroutines are also likely to make the writing of user-defined algorithms easier, their names, their formal parameter lists and their actions are explained below. These subroutines are also referred to in the explanation of the built-in reconstruction algorithms in Chapter 7.

6.4.1 pick

```
#include "pick.h"

void pick(INTEGER* np, INTEGER* nr);
```

This subroutine assigns to *np* a projection number and to *nr* a ray number. The actual values to be returned are selected according to the method determined by the most recently executed SELECT command (see Section 5.6). If no SELECT command has been executed, *np* and *nr* are selected in the same way as if the most recently executed SELECT command was

SELECT USER EFFICIENT

Warning: Using `pick` in a termination test is not advisable, since this would alter the sequence of pairs (*np*, *nr*) that is returned to the reconstruction algorithm after the first iteration (see Section 6.1).

6.4.2 wray

```
#include "wray.h"

void wray(
     INTEGER np, INTEGER nr,
```

```
        INTEGER* list, REAL* weight,
        INTEGER* numb, REAL* snorm
    );
```

For the npth projection and nrth ray, this routine calculates the trace of a LINE ray through the picture region and the pixels intersected. *Restriction*: $0 \leq np$ < PRJNUM and $0 \leq nr$ < NRAYS. *Warning*: In case of STRIP rays, the subroutine will return values that are appropriate for LINE rays located at the centers of the actual STRIP rays.

`wray` returns the following information:

`numb` Number of pixels intersected. When `numb` = 0, the ray in question does not intersect the picture region.

`list[0], ..., list[numb-1]` The indices of the pixels intersected by the ray. I.e., if j = `list[`i`]` , then `recon[`j`]` is the present density in the ith pixel intersected by the ray (see Section 6.2).

`weight[0], ..., weight[numb-1]` The lengths of the ray segments within the pixels in the order given in `list`.

`snorm` The sum of the squares of the above lengths.

See Section 6.3.4.3 for the blob equivalent of this subroutine.

6.4.3 ray

```
#include "ray.h"

void ray (
        INTEGER np, INTEGER nr,
        INTEGER* list, REAL* weight,
        INTEGER* numb, REAL* snorm
);
```

For the npth projection and nrth ray, this routine calculates the trace of a STRIP ray through the picture region. *Restriction*: $0 \leq np$ < PRJNUM and $0 \leq nr$ < NRAYS. *Warnings*: In case of PARALLEL LINE rays, the subroutine will return the values that are appropriate for PARALLEL STRIP rays whose central lines are the actual LINE rays (see Section 2.3). If `ray` is called in the case of DIVERGENT rays, the SNARK14 run will be terminated.

`ray` returns the following information:

`numb` Number of pixels whose centers are in the ray. When `numb` = 0, the ray in question does not contain the center of any pixel.

`list[0], ..., list[numb-1]` The indices of the pixels whose centers are in the ray. I.e., if j = `list[`i`]`, then `recon[`j`]` is the present density in the ith pixel whose center is in the ray (see Section 6.2).

`weight[0], ..., weight[numb-1]` Are all assigned the value `pixsiz*pixsiz`.

`snorm` is assigned the value `pixsiz`×`pixsiz`×`pixsiz`×`pixsiz`×`numb`.

See Section 6.3.4.4 for the blob equivalent of this subroutine.

6.4.4 pseudo

```
#include "pseudo.h"

REAL
pseudo(
        REAL* pict, INTEGER np, INTEGER nr,
```

```
     INTEGER* list, REAL* weight, INTEGER* numb, REAL* snorm,
     BOOLEAN line, BOOLEAN constr
);
```

Roughly speaking, this function returns the pseudo ray sum of a picture, whose pixel densities are stored in `pict`, for the nrth ray of the npth projection. The exact operation depends on the values of the logical variables `line` and `constr` and is best explained by listing the whole of the rest of the subroutine.

```
#include <cstdio>
#include "blkdta.h"
#include "modefl.h"
#include "uiod.h"
#include "ray.h"
#include "wray.h"
#include "pseudo.h"

REAL
pseudo(
     REAL* pict,
     INTEGER np,
     INTEGER nr,
     INTEGER* list,
     REAL* weight,
     INTEGER* numb,
     REAL* snorm,
     BOOLEAN line,
     BOOLEAN constr
) {
  INTEGER nb, k;
  REAL pictk;
  REAL sum = 0.0;
  if (!line) {
    ray(np, nr, list, weight, numb, snorm);
  }
  else {
    wray(np, nr, list, weight, numb, snorm);
  }
  if (*numb != 0) {
    for (nb = 0; nb < *numb; nb++) {
      k = list[nb];
      pictk = pict[k];
      if (constr) {
        if (Modefl.lofl) pictk = MAX0(pictk, Modefl.lower);
        if (Modefl.upfl) pictk = MIN0(pictk, Modefl.upper);
      }
      sum += pictk * weight[nb];
    }
  }
  if (trace > 5) {
    fprintf(output,
            "\n          pseudo  np = %5i  nr = %5i  pseudo = %9.4f\n",
            np, nr, sum, line);
  }
  return sum;
}
```

Note: By calling the function `pseudo`, the variables `list`, `weight`, `numb`, and `snorm` will have their values determined by `wray` or `ray` (see Sections 6.4.2 and 6.4.3).

See Section 6.3.4.5 for the blob equivalent of this subroutine.

6.4.5 qintp

```
#include "qintp.h"

REAL qintp(REAL pos, REAL* table, INTEGER n, INTEGER interp);
```

This is a one-dimensional interpolation routine. Suppose the array `table` is such that `table[i]` is the value of a function at the integer point `i`, for $0 \leq$ `i` $<$ `n`. `qintp` returns the estimated value of the function at the point `pos`, based on the values in the array table and the interpolation method specified by the integer `interp`.

Warning: When using this function, the value of `pos` must lie within certain limits, which will be specified below for each of the interpolation methods separately. Calling the function with a value of `pos` outside the specified range will result in unpredictable consequences. It is also essential that the size of the array table is at least `n`.

In what follows we use E to denote the largest integer not greater than `pos` and F to denote the largest integer not greater than `pos+0.5` (F is the integer nearest to `pos`.) The restrictions on `pos` are expressed in terms of E or F.

There are eight interpolation methods available in `qintp`. (Restriction: -1 $\leq$ `interp` $\leq$ 6.)

1. Modified cubic spline interpolation proposed in [60]. *Restriction*: $1 \leq E \leq$ `n`-2. If `interp` = -1, `qintp` returns the value at `pos` of a cubic polynomial whose values at E and $E+1$ are `table[`E`]` and `table[`$E+1$`]`, respectively, and whose slopes at E and $E+1$ are `(table[`$E+1$`]-table[`$E-1$`])/2` and `(table[`$E+2$`]-table[`E`])/2`, respectively.

2. Band limiting (sinc) interpolation. *Restriction*: 0$\leq F \leq$ `n-1`. If `interp` = 0, `qintp` returns a value defined as follows.

$$\texttt{qintp} = \begin{cases} \texttt{table}[F], & \text{if } |F - \texttt{pos}| < \text{ZERO}, \\ \left(\sum_{k=1}^{\texttt{n}} (-1)^k \frac{\texttt{table}[k]}{\texttt{pos}-k}\right) \frac{\sin(\pi \times \texttt{pos})}{\pi}, & \text{otherwise.} \end{cases}$$

3. Nearest neighbor interpolation. *Restriction*: $0 \leq F \leq$ `n-1`. If `interp` = 1, `qintp` returns `table[`F`]`.

4. Linear interpolation. *Restriction*: $0 \leq E \leq$ `n-1`. If `interp` = 2, `qintp` returns ($E+1$-`pos`)*`table[`E`]` + (`pos`-E)*`table[`$E+1$`]`.

5. Three point Lagrange interpolation. *Restriction*: $1 \leq F \leq$ `n-2`. If `interp` = 3, `qintp` returns the value at `pos` of a quadratic polynomial whose values at $F-1$, F, and $F+1$ are `table[`$F-1$`]`, `table[`F`]`, and `table[`$F+1$`]`, respectively.

6. Four point Lagrange interpolation. *Restriction*:$1 \leq E \leq$ `n-3`. If `interp` = 4, `qintp` returns the value at `pos` of a cubic polynomial whose values at $E-1$, E, $E+1$, and $E+2$ are `table[`$E-1$`]`, `table[`E`]`, `table[`$E+1$`]`, and `table[`$E+2$`]`, respectively.

7. Five point Lagrange interpolation. *Restriction*: $2 \leq F \leq$ `n-3`. If `interp` = 5, `qintp` returns the value at `pos` of a quartic polynomial whose value at `i` is `table[i]` for $F-2 \leq$ `i` $\leq F+2$.

8. Six point Lagrange interpolation. *Restriction*: $2 \leq E \leq$ `n-4`. If `interp` = 6, qintp returns the value at `pos` of a quintic polynomial whose value at `i` is `table[i]` for $E-2 \leq$ `i` $\leq E+3$.

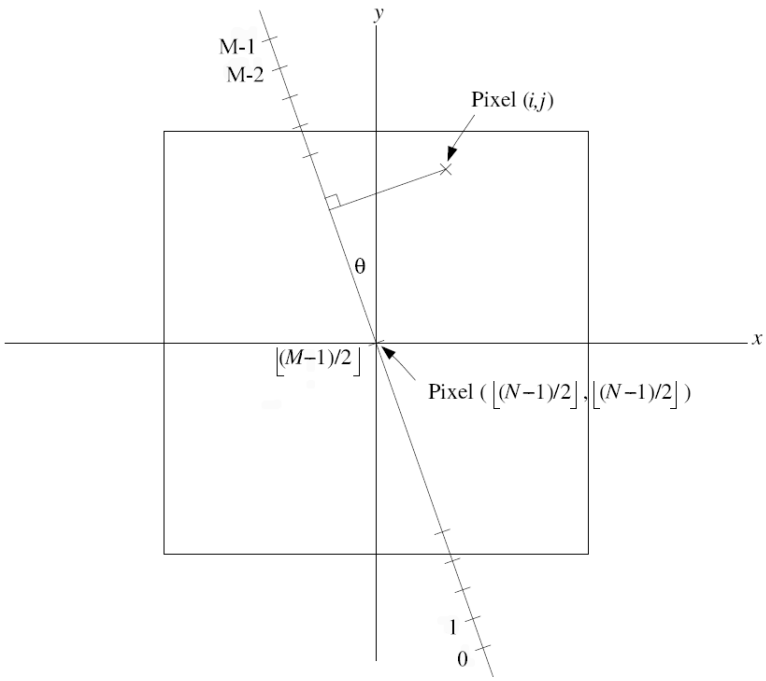

Figure 6.1: The backprojection operation.

6.4.6 bckprj

```
#include "bckprj.h"

void
bckprj(
     REAL* a, INTEGER n, REAL* g, INTEGER m,
     REAL sth, REAL cth, REAL spac, INTEGER interp
);
```

Roughly speaking the purpose of this routine is to back-project ("smear") a function of one variable (determined by the array `g`) in the direction determined by the unit vector (`cth`, `sth`) onto a picture region, and to add the values of the digitized version of the "smear" picture to array `a`.

More precisely, `a` is an array, such that `a[`i`*n+`j`]` represents the density of the jth pixel of the ith row of an `n` × `n` digitized picture in which the length of a side of a pixel is `spac`. (*Warning*: The size of the array must be at least `n` × `n`.) This is the same indexing as is used for `recon` (see Section 6.2). `n` must be a positive integer, but it need not be odd. It is assumed that the origin of the coordinate system is at the center of the ⌊`(n-1)/2`⌋th pixel of the ⌊`(n-1)/2`⌋th row, where $\lfloor x \rfloor$ denotes the largest integer not greater than x.

Consider a line L through the origin that makes an angle θ with the y-axis (see Figure 6.1), where $0 \le \theta \le 2\pi$, `cth` $= \cos\theta$ and `sth` $= \sin\theta$. We measure the position along this line as indicated in Figure 6.1. In particular, the position of the origin is `(m-1)/2`. Then it is easy to calculate that the orthogonal projection of the center of the (i, j)th pixel onto the line that has position

`pos` $= \lfloor$`(m-1)/2`$\rfloor$-{j-$\lfloor$`(n-1)/2`$\rfloor$)×`sth`+(i-$\lfloor$`(n-1)/2`$\rfloor$)×`cth`}×`spac`.

`p` is an array whose first `m` values represent the values of a function `p` defined on the line L. (*Warning*: The dimension of the array `p` must be at least `m`.) The value of `p(x)` at a non-integer `x` is to be defined by interpolation, as will be explained below.

Now we are in position to explain the precise effect of bckprj. The (`i*n+j`)th entry of the array `a` has added to it the value of

```
qintp(pos, p, m, interp)
```

(see Section 6.4.5), where `pos` is defined by the formula given above. *Restrictions*: $-1 \leq$ `interp` ≤ 6 and the value of `m` has to be large enough so that `pos` will lie between the limits specified in Section 6.4.5

Note: In actual applications of this subroutine the value of `spac` will be the ratio of pixel size to detector spacing (see, e.g., Section 7.1).

6.4.7 posit

```
#include "posit.h"

void posit(INTEGER np, INTEGER nr, REAL* ax, REAL* ay, REAL* mx, REAL* my);
```

This subroutine returns, for the nrth ray of the npth projection, the coordinate (`ax`, `ay`) of a point on the ray (in the STRIP case a point on the central line of the ray) and the cosine `mx` and the sine `my` of the angle that the ray makes with the x-axis. *Restrictions*: $0 \leq np <$ PRJNUM and $0 \leq nr <$ NRAYS.

6.4.8 raylen

```
#include "raylen.h"
#include "objects.h"

REAL raylen(
     SNARK_Object_type type,
     REAL ax, REAL ay, REAL mx, REAL my,
     REAL cx, REAL cy, REAL u, REAL v, REAL px, REAL py
);
```

This function returns the length of the part of a straight line (ray) that lies inside an elemental object. The straight line is determined by the coordinates (`ax`, `ay`) of a point on it, and by the cosine `mx` and the sine `my` of the angle that the line makes with the x-axis. The type of elemental object (see Section 5.3.1) is determined by the enumerated variable type. *Restriction*: The value of `type` must be one of `SOT_elip`, `SOT_rect`, `SOT_tria`, `SOT_segm`, `SOT_sect`, which are defined in `objects.h` (see Figure 5.1). The values of `cx`, `cy`, `u`, and `v` are interpreted according to Figure 5.1. The values of `px` and `py` are the cosine and the sine, respectively, of the angle denoted by ANG in Figure 5.1.

6.4.9 contur

```
#include "contur.h"

void contur(REAL* a, INTEGER m, INTEGER n, REAL thresh, REAL w1, REAL w2, REAL w3);
```

This subroutine changes the first `m`×`n` elements of the array `a`, by setting the value to `w1` or `w2`. If w3 $\leq$ ZERO, the value is set depending on whether the original value was $\leq$ or $>$ `thresh`, respectively. If w3 $>$ ZERO, the threshold is chosen such that the average density of the resulting image is as close to AVEDEN as possible. (See Section 5.8.) *Warning*: The size of `a` must be at least `m` $\times$ `n`.

6.4.10 smooth

```
#include "smooth.h"

void smooth(REAL* a, INTEGER m, INTEGER n, REAL thresh, REAL w1, REAL w2, REAL w3);
```

This subroutine carries out the nine point selective smoothing procedure, which is described in detail in Section 5.8. For $0 \leq i <$ `m` and $0 \leq j <$ `n`, the (i×`n`+j)th element of the array `a` is interpreted as the value of the (i, j)th pixel of an `m` $\times$ `n` digitized picture. (We have only defined $N \times N$ digitized pictures in Section

2.2, but the generalized definition is obvious.) *Warning*: The size of `a` must be at least `m`×`n`. The smoothing of this picture is done in the manner specified in Section 5.8, with the values of `thresh`, `w1`, `w2`, and `w3` assigned to the threshold, weight1, weight2, and weight3, respectively.

See Section 6.3.4.6 for the blob equivalent of this subroutine.

6.4.11 snfft

```
#include "snfft.h"

void snfft(REAL* x, INTEGER* n, INTEGER invdir);
```

This subroutine performs complex Fourier transforms on a complex three-dimensional array $f[k,l,m]$. The integer array n is of size 3; n`[i]` denotes the size of the 3-dimensional array $f[k,l,m]$ in the ith dimension. (A one-dimensional array can be treated by this subroutine by setting n`[1]` =n`[2]` = 1.)

As input, the array x contains $f[k,l,m]$ in a vector fashion. For $0 \le k < n$`[0]`, $0 \le l < n$`[1]`, and $0 \le m < n$`[2]`, the real part of $f[k,l,m]$ is assigned to the $(2 \times (k \times n$`[1]` $\times n$`[2]` $+ l \times n$`[2]` $+ m))$th location of `x`, the imaginary part of $f[k,l,m]$ to the immediately following location. *Warning*: The size of `x` must be at least $2 \times n$`[0]` $\times n$`[1]` $\times n$`[2]`. As output, `x` contains (using the same indexing conventions), the complex discrete Fourier transform $\mathcal{F}f$ of f if `invdir` = 1 or the complex inverse discrete Fourier transform $\mathcal{F}^{-1}f$ of f, if `invdir` = -1. (These transforms are defined below.) *Restriction*: the value of `invdir` must be 1 or -1.

$$[\mathcal{F}f](k,l,m) = \sum_{p=0}^{n[0]-1} \sum_{q=0}^{n[1]-1} \sum_{r=0}^{n[2]-1} f(p,q,r) e^{-2\pi i(pk/n[0]+ql/n[1]+rm/n[2])}$$

$$[\mathcal{F}^{-1}f](k,l,m) = \frac{1}{n[0] \times n[1] \times n[2]} \times \sum_{p=0}^{n[0]-1} \sum_{q=0}^{n[1]-1} \sum_{r=0}^{n[2]-1} f(p,q,r) e^{2\pi i(pk/n[0]+ql/n[1]+rm/n[2])}$$

This subroutine is based on [62].

6.4.12 rtfort

```
#include "rtfort.h"

void rtfort(REAL* x, INTEGER* n, INTEGER invdir);
```

This subroutine performs discrete Fourier transforms on real one or two-dimensional arrays. It also has the capability of performing the inverse discrete Fourier transform of complex arrays to real arrays. It is based on the property that (in one dimension) $[\mathcal{F}g](X) = [\mathcal{F}g](-X)^*$, where z^* denotes the conjugate of the complex number z, so that we need only compute half of the Fourier Transform of the original data. Therefore, even though the Fourier Transform is complex valued, it requires essentially no more space than the real array (see [7] Section 10-10). *Note*: Due to the complexity of using this routine, it is recommended that snfft (Section 6.4.11) be used instead.

`x` is a pointer to a real array. In the forward direction, each element of `x` is a real value representing the function to be transformed. As the output of rtfort, each pair of `REAL`s of `x`, `x[2*i]` and `x[2*i+1]`, is a complex value representing the Fourier transform of the function. In the reverse direction, the roles are reversed.

n is a pointer to an integer array of 2 elements that is related to the size of `x`. If `x` is a one dimensional array, then n`[0]` = 1, otherwise, n`[0]` is the number of rows in the two dimensional array. n`[1]` is one-half the number of real elements of `x` in the one dimensional case and one-half the number of columns of `x` in the two dimensional case. It may be necessary to pad the array with extra elements so that this value is an

integer. *Warning*: The size of the array `x` must be at least $2\times n$`[0]`$\times(n$`[1]`$+1)$ to accommodate the extra elements in the Fourier transform.

`invdir` is 1 if the discrete Fourier transform is desired and -1 if the inverse discrete Fourier transform is desired.

If $g(k,l)$ is a real array then, for $0 \le k < n$`[0]` and $0 \le l < 2\times n$`[1]`, $g(k,l)$ is stored in location $(2 \times k\times n$`[1]`$+l)$ of `x`. The subroutine `rtfort` has the property that the C++ statement

```
rtfort(x, n, 1);
```

will place in the array `x` the (complex) discrete Fourier transform of g. For $0 \le k < n$`[0]` and $0 \le l \le n$`[1]`, the real part of $[\mathcal{F}g](k,l)$ is assigned to the $(2 \times (k \times (n$`[1]`$+1) + l))$th location in `x`, and the imaginary part to the immediately following location. Since g is real,

$$[\mathcal{F}g](k,l) = ([\mathcal{F}g](n[0]-k, 2n[1]-l))^*.$$

Hence, the value of $[\mathcal{F}g](k,l)$ for $0 \le k <$ `n[0]`, `n[1]` $< l <$ $2\times$`n[1]` are also easily obtained from the contents of the array `x`.

The subroutine `rtfort` also has the property that if

```
rtfort(x, n, -1);
```

is applied to the array `x` that contains in the just described manner the half of the complex discrete Fourier transform $[\mathcal{F}g]$ of g, then the resulting content of the $(2 \times k\times n$`[1]`$+l)$th element of the array `x` will be the real number $g(k,l)$, for $0 \le k < n$`[0]` and $0 \le l < 2\times n$`[1]`.

This subroutine is based on [61].

6.4.13 qfilt

```
#include "qfilt.h"

REAL qfilt(REAL R, REAL cutoff, INTEGER filter);
```

This function returns the value of a "filter function" F of one variable at the point R. There are four types of filters in SNARK14. Filters of the same type are distinguished from each other by the value of the parameter `cutoff`. The type of the filter is determined by the value of the variable `filter`. (*Restriction*: $0 \le$ `filter` ≤ 3.)

We define

$$\Pi(x) = \begin{cases} 1, & \text{if } x \le \frac{1}{2}, \\ 0, & \text{if } x > \frac{1}{2}, \end{cases}$$

$$\operatorname{sinc}(x) = \begin{cases} \frac{\sin \pi x}{\pi x}, & \text{if } x \ne 0, \\ 1, & \text{if } x = 0. \end{cases}$$

We use C to abbreviate the value of `cutoff`.

(i) *Band limiting filter* (used if FILTER = 0):

$$F(R) = \Pi\left(\frac{R}{2C}\right) \times |R|.$$

(ii) *Sinc filter* (used if FILTER = 1):

$$F(R) = \operatorname{sinc}\left(\frac{R}{2C}\right) \times \Pi\left(\frac{R}{2C}\right) \times |R|.$$

(iii) *Cosine filter* (used if FILTER = 2):

$$F(R) = \cos\left(\frac{\pi R}{2C}\right) \times \Pi\left(\frac{R}{2C}\right) \times |R|.$$

(iv) *Hamming filter* (used if FILTER = 3):

$$F(R) = (C + (1-C)\cos(\pi R)) \times \Pi\left(\frac{R}{2}\right) \times |R|.$$

6.4.14 qinit

```
#include "qinit.h"

void qinit (REAL* table, INTEGER isize, INTEGER filter, REAL cutoff);
```

This subroutine puts in the first `isize` positions of the array `table` a discrete "convolving function". The actual function depends on the "filter" that is indicated by the value of `filter` and `cutoff`. *Restriction*: $0 \leq$ `filter` ≤ 3 (see Section 6.4.13 for the meaning of `filter`). *Warning*: the size of `table` must be at least `isize`.

For any permissible value of `filter` and `cutoff` let the filter function $F(R)$ be defined as in Section 6.4.13. Then for $0 \leq m <$ `isize`, `table[`m`]` is assigned half the value of the mth Fourier coefficient of the function $F(R)$ defined over the interval [-1,1]. In other words, for $0 \leq m <$ `isize`,

$$\text{table}[m] = \frac{1}{4}\int_{-1}^{1} F(R)e^{\pi imR}dR.$$

These entries in table are real, since F is symmetric.

The reasoning behind this definition of the "convolving function" has been given in [64]. Various convolving functions proposed in the literature can be produced by adjusting the values of `filter` and `cutoff`. These include the convolving functions proposed in [6, 12, 63, 66].

6.4.15 sinc

```
#include "sinc.h"

REAL sinc(REAL x);
```

This function returns the value 1 if |`x`| < ZERO and it returns the value $\sin(\pi$`x`$)/\pi$`x` otherwise.

6.4.16 cin

```
#include "cin.h"

REAL cin(REAL x);
```

This function returns the estimated value of $\int_0^x (1-\cos u)/u du$. The estimation is done using rational approximation as described by [20, 69].

6.4.17 Rand

```
#include "DIGRand.h"

void Srand(unsigned int pSeed);
REAL Rand();
```

This function is used for generating random numbers uniformly distributed in the range [0.0, 1.0). `Srand()` is used to set the seed to the random number generator. `Rand()` return the next random number in the sequence.

Warning: The sequence produced by `Rand()` is installation dependent.

6.4.18 Gauss

```
#include "DIGRand/DIGGauss.h"

double Gauss(double mean, double std);
```

This function returns a sample from a normal distribution with mean `mean` and standard deviation `std`. The method used for calculating the sample is the polar method based on the uniform distribution generated by `Rand` (see Section 6.4.17). This method is an implementation of one proposed by Knuth [49].

6.4.19 Poisson

```
#include "DIGRand/DIGPoisson.h"

int Poisson(double pALambda);
```

This subroutine returns a sample from a Poisson distribution with mean `pALambda`. This algorithm is described in [65].

6.4.20 second

```
#include "second.h"

void second(REAL* ctime);
```

This subroutine assigns to the variable `ctime` the CPU time used by the SNARK14 program measured in seconds. It is used to report on the execution times in the SNARK14 output. The accuracy of ctime is system dependent.

Chapter 7

BUILT-IN RECONSTRUCTION ALGORITHMS

This chapter documents the algorithms existing within SNARK14 and provides the user with a description of the follow-up command lines a particular algorithm requires after the EXECUTE control command line (see Section 5.8). The descriptions of the algorithms provided here are by necessity rather sketchy; the user is referred to the cited literature for derivation of the formulas and for discussion of the properties of the algorithms. Our description will depend heavily on the material introduced in Chapter 6.

The sections of this chapter are entitled by the value of the word modifier algname on the EXECUTE control command line that determines the reconstruction algorithm that will be executed. These names are BACKPROJECTION, CONVOLUTION, RFL, FOURIER, DCONV, ART, MART, QUADRATIC, SIRT, EMAP, and LINO. In each case we describe the necessary follow-up command lines in the EXECUTE command and explain the purpose of the modifiers occurring on these command lines. These command lines are all in control command line format and they immediately follow the last line of the EXECUTE sequence (see Section 5.8).

References that are given to the literature for description of the various methods are not necessarily to sources that originated the method in question. They have been chosen for their informative value rather than for their originality.

7.1 BACKPROJECTION

$$> \quad \left\{ \begin{array}{l} \text{CONTINUOUS interp} \\ \text{DISCRETE [ZEROS] [DISTRIBUTED]} \end{array} \right\} \left[\begin{array}{l} \text{ADDITIVE} \\ \text{MULTIPLICATIVE} \end{array} \right]$$

This is a general implementation of the backprojection or summation reconstruction methods (see [29] Chapter 7). It incorporates two fairly different approaches. Both of them first produce a NELEM×NELEM "preliminary picture" in a fashion that we now describe.

If CONTINUOUS is specified, then, roughly speaking, the preliminary picture has a value assigned to a pixel that is the sum of the ray sums for the rays that go through the center of the pixel. (One and only one ray in each projection is assumed to go through the center of a pixel.)

A precise description of how the preliminary picture is obtained in the CONTINUOUS case is best given by C++ like code that makes use of some subroutines described in Section 6.4.

```
Anglst.getang(GeoPar.prjnum -1, &theta, &sinth, &costh);
theta0 = theta - thmod;
for (np = 0; np < GeoPar.prjnum; np++) {
    ProjFile.ReadProj(np, g, nsize);
    Anglst.getang(np, &theta1, &sinth, &costh);
    np2 = np + 1;
    if (np == (GeoPar.prjnum - 1)) np2 = 0;
```

```
        Anglst.getang(np2, &theta2, &sinth2, &costh2);
        if (np == (GeoPar.prjnum - 1)) theta2 += thmod;
        w = (theta2 - theta0) / thfac;
        rinc = GeoPar.pinc;
        if (GeoPar.div) rinc *= GeoPar.radius / GeoPar.stod;
        if (GeoPar.vri) rinc *= (REAL) MAX0(fabs(sinth), fabs(costh));
        if (GeoPar.strip) w /= rinc;
        theta0 = theta1;
        for (nr = 0; nr < nsize; nr++) {
            g[nr] *= w;
        }
        bckprj(recon, GeoPar.nelem, g, nsize, sinth, costh, GeoPar.pixsiz/rinc, interp);
    }
```

The variables that are not in one of the C++ classes (see Section 6.3) and appear in the code above have their values defined as follows. (All these variables are assigned their correct values by SNARK14.)

`nsize`, the size of the array `g`, is an integer large enough so that the restriction on the size of `m` in `bckprj` (see Section 6.4.6) is satisfied.

w(np) depends on the geometry of data collection. Let the variable RANGE have the value π in the PARALLEL case and the value 2π in the DIVERGENT case. Let $\theta(np)$ denote the projection angle in radians of the npth projection for $0 \le np <$ PRJNUM, and let θ(-1) be θ(PRJNUM)-RANGE and θ(PRJNUM) be $\theta(0)$ + RANGE. (*Restriction*: For $0 \le np \le$ PRJNUM, $\theta(np\text{-}1) < \theta(np)$.) Let WIDTH($np$) denote the actual distance between rays in the PARALLEL LINE case and the actual width of the strips in the PARALLEL STRIP case. Then,

$$\mathrm{w}(np) = \begin{cases} \frac{\theta(np+1)-\theta(np-1)}{2}, & \text{in the PARALLEL LINE case,} \\ \frac{\theta(np+1)-\theta(np-1)}{2\times \mathrm{WIDTH}(np)}, & \text{in the PARALLEL STRIP case,} \\ \frac{\theta(np+1)-\theta(np-1)}{4}, & \text{in the DIVERGENT case.} \end{cases}$$

$$\mathrm{SPAC} = \begin{cases} \frac{\mathrm{PIXSIZ}}{\mathrm{WIDTH}(np)}, & \text{in the PARALLEL case,} \\ \frac{\mathrm{PIXSIZ}\times\mathrm{STOD}}{\mathrm{PINC}\times\mathrm{RADIUS}}, & \text{in the DIVERGENT case.} \end{cases}$$

The value assigned to `interp` is the value of the integer modifier interp. *Restriction*: $-1 \le$ interp ≤ 6 (see Section 6.4.5).

Warning: In the CONTINUOUS case the rays of DIVERGENT geometry are treated as if they were PARALLEL with UNIFORM ray spacing PINC×RADIUS/STOD. A large divergence angle will result in unacceptable reconstructions.

If DISCRETE is specified, then roughly speaking the value of a pixel in the preliminary picture is the sum of contributions from every ray that intersects the pixel. The size of the contribution is proportional to the ray sum of the ray and also depends on the size of the intersection with the pixel.

A precise description of how the preliminary picture is obtained in the DISCRETE case is best given by C++ like code that makes use of some of the subroutines in Section 6.4.

```
    for (np = 0; np < GeoPar.prjnum; np++) {
        ProjFile.ReadProj(np, g, GeoPar.nrays);
        for (nr = GeoPar.fsnray; nr <= GeoPar.lsnray; nr++) {
            if (GeoPar.strip) ray(np, nr, list, weight, &numb, &snorm);
            if (GeoPar.line) wray(np, nr, list, weight, &numb, &snorm);
            if (numb != 0) {
                amt = g[nr];
                if (dist) {
                    if (zeros) {
                        snorm = 0.0;
                        for (nb = 0; nb < numb; nb++) {
```

```
                        k = list[nb];
                        if (recon[k] > Consts.neginf) {
                            snorm += weight[nb]*weight[nb];
                        }
                    }
                }
                amt /= snorm;
            }
            for (nb = 0; nb < numb; nb++) {
                k = list[nb];
                recon[k] += amt * weight[nb];
            }
        }
    }
}
```

The variables that are not in C++ classes (see Section 6.3) and appear in the code above have their values defined as follows. (All these variables are assigned their correct values by SNARK14.)

`zeros` is a logical variable whose value is TRUE if the option ZEROS is specified on the follow-up line for this algorithm. If `zeros` is TRUE then prior to the code described above the algorithm loops through all the rays (from FUSRAY to LUSRAY) in all the projections and flags all the pixels that are intersected by rays whose ray sum is 0.

`dist` is a logical variable whose value is TRUE if the option DISTRIBUTED is specified on the follow-up line for this algorithm.

The preliminary picture produced by either the CONTINUOUS or DISCRETE method described above is further altered according to the following rules. The values given to the pixels in the preliminary picture are altered either by addition of (if ADDITIVE is specified) or by multiplication by (if MULTIPLICATIVE is specified) a constant (the same constant for all pixels) so that the average density of the resulting picture is AVEDEN (see Section 6.3.3). If neither MULTIPLICATIVE nor ADDITIVE is specified, the preliminary picture is not normalized and is the final reconstructed picture produced by the algorithm.

If the iteration number `iter` > 1, this algorithm returns control to `exalg` without changing the contents of the `recon` array. Multiple iterations may nevertheless be used for smoothing or contouring the reconstruction (see Sections 5.8 and 6.1).

7.2 CONVOLUTION

$$> \quad \left\{ \begin{array}{l} \text{BANDLIMITING} \\ \text{SINC} \\ \text{COSINE} \\ \text{HAMMING} \end{array} \right\} \text{ cutoff interp}$$

This is a general implementation of the filtered backprojection, also known as the convolution, method for parallel beams (see, e.g., [29] Chapter 8, or [64]). Roughly speaking, the projection data are first "convolved" by a function based on a "filter", and then the convolved projection data are backprojected to form the reconstruction.

More precisely, the convolving function is defined by the filter (BANDLIMITING, SINC, COSINE, or HAMMING) and the floating point modifier cutoff. The convolving function is put into the array `table` by the C++ statement

```
qinit(table, nsize, filter, c);
```

(see Section 6.4.14), where `nsize` is an integer large enough for the "backprojection" to be discussed below to be a valid operation, `filter` is 0, 1, 2, or 3 if the filter is BANDLIMITING, SINC, COSINE, or HAMMING,

respectively, and

$$c = \begin{cases} \text{cutoff}, & \text{if ZERO} < \text{cutoff} \le 1, \\ \min\left(\frac{2\times\text{PRJNUM}\times\text{PINC}}{\pi\times\text{NELEM}\times\text{PIXSIZ}}, 1\right), & \text{if cutoff} < 0, \\ 1, & \text{otherwise.} \end{cases}$$

The reconstruction proceeds in exactly the same way as was used to obtain the preliminary picture in the CONTINUOUS case of the BACKPROJECTION method (Section 7.1), with one important exception. Immediately after the statement

```
ProjFile.ReadProj(np, data, nsize);
```

additional code is inserted, as a result of which the array `data` is altered. The new value of `data[`i`]`, for $0 \le i <$ `nsize`, can be expressed in terms of the old values of the array `data` and the `table` generated by `qinit` above by the convolution formula

$$\frac{1}{\text{WIDTH}(np)} \sum_{j=0}^{\text{nsize}-1} \text{data}[i] \times \text{table}\left[|i-j|\right].$$

Here WIDTH(np) denotes (for the npth projection) the actual distance between the lines in the PARALLEL LINE case, the actual width of the strips in the PARALLEL STRIP case, and PINC×RADIUS/STOD in the DIVERGENT case. The result of the continuous backprojection of the convolved data is (without any normalization) the final reconstructed picture produced by the convolution algorithm.

Restriction: For $0 \le np \le$ PRJNUM, $\theta(np\text{-}1) < \theta(np)$. (see Section 7.1 for definition of $\theta(np)$). $-1 \le$ `interp` ≤ 6 (see Section 6.4.5).

Warning: The rays of DIVERGENT geometry are treated as if they were PARALLEL with UNIFORM ray spacing PINC×RADIUS/STOD. If divergence of rays is so great that this results in unacceptable error, then the use of the DCONV reconstruction algorithm (Section 7.5) is recommended.

If the iteration number `iter` > 1, this algorithm returns control to `exalg` without changing the contents of the `recon` array. Multiple iterations may nevertheless be used for smoothing or contouring the reconstruction (see Sections 5.8 and 6.1).

7.3 RFL

$$> \quad \left\{ \begin{array}{l} \text{BANDLIMITING} \\ \text{SINC} \\ \text{COSINE} \\ \text{HAMMING} \end{array} \right\} \text{ cutoff interp [size]}$$

This algorithm, whose name abbreviates *rho filtered layergram* (see, e.g., [64] or [68]), (i) takes the output of the CONTINUOUS BACKPROJECTION, (ii) performs a two-dimensional Fourier transform, (iii) multiplies the values of the Fourier transform at all points by a filter function $F(\rho, \phi)$, where ρ denotes the distance of the point from the origin, and then (iv) it performs an inverse two-dimensional Fourier transform to get the reconstructed picture.

(i) An array, which we call `lrecon` is set up to hold the densities associated with the pixels of a size×size digitized picture. The same indexing is to be used as for `recon`, see Section 6.2. (*Restriction*: size > NELEM and even.) If size is not specified, its value is taken to be NELEM + 1. It is assumed that the origin of the coordinate system is at the center of the (size/2)th pixel of the (size/2)th row (cf., Section 6.4.6).

A picture is produced in the `lrecon` array by exactly the same method by which BACKPROJECTION produces a picture in the `recon` array if

CONTINUOUS interp ADDITIVE

is specified (see Section 7.1). *Restrictions*: $-1 \le$ `interp` ≤ 6 (see Section 6.4.5). For $0 \le np \le$ PRJNUM, $\theta(np) < \theta(np+1)$ (see Section 7.1 for definition of $\theta(np)$).

(ii) We transform the contents of `lrecon` by

`rtfort(lrecon,` n`, 1);`

(see Section 6.4.12), with n`[1]` = `size`, and n`[2]` = `size`/2. After this transformation `lrecon[i]`, for $1 \leq$ `i` $\leq$ `size` $\times$ `size`, represents the value at some point, whose polar coordinates we denote by (ρ_i, ϕ_i), of the real or imaginary part of the Fourier transform of the picture obtained by backprojection.

(iii) At this point, the subroutine alters `lrecon[i]` by multiplying it by $F(\rho_i, \phi_i)$. (Definition of the function F will be given below).

(iv) The resulting array in `lrecon` is inverse Fourier transformed using

`rtfort(lrecon,` n`, -1);`

(see Section 6.4.12), producing a picture in the array `lrecon`, the central part of which is transferred to `recon` as the reconstructed picture. Finally, the `recon` array is normalized by addition of a constant so that its average density is AVEDEN (see Section 6.3.3) .

The function F is defined as follows, based on the filter (BANDLIMITING, SINC, COSINE, or HAMMING) and floating point modifier cutoff.

Let

$$C = \begin{cases} \text{cutoff}, & \text{if ZERO} < \text{cutoff} \leq 1, \\ \min\left(\frac{2\times\text{PRJNUM}\times\text{PINC}}{\pi\times\text{NELEM}\times\text{PIXSIZ}}, 1\right), & \text{if cutoff} < 0, \\ 1, & \text{otherwise.} \end{cases}$$

and

$$A(\phi) = \begin{cases} 2 \times \text{PINC}, & \text{in the PARALLEL UNIFORM case,} \\ 2 \times \text{PINC} \times \max(|\sin\phi|, |\cos\phi|), & \text{in the PARALLEL VARIABLE case,} \\ \frac{2\times\text{PINC}\times\text{RADIUS}}{\text{STOD}}, & \text{in the DIVERGENT case.} \end{cases}$$

Then,

$$F(\rho, \phi) = \text{qfilt}(A(\phi) \times \rho, C, \text{filter})/A(\phi)$$

where `qfilt` is the function defined in Section 6.4.13 and the value of filter is 1, 2, 3 or 4 if the filter is BANDLIMITING, SINC, COSINE, or HAMMING, respectively. Restrictions stated on the value of C in Section 6.4.13 apply.

Warning: The rays of DIVERGENT geometry are treated as if they were PARALLEL with UNIFORM ray spacing PINC×RADIUS/STOD. A large divergence angle will result in unacceptable reconstructions.

If the iteration number `iter` > 1, this algorithm returns control to `exalg` without changing the contents of the `recon` array. Multiple iterations may nevertheless be used for smoothing or contouring the reconstruction (see Sections 5.8 and 6.1).

7.4 FOURIER

$$> \quad \left\{ \begin{array}{l} \text{BANDLIMITING} \\ \text{SINC} \\ \text{COSINE} \\ \text{HAMMING} \end{array} \right\} \text{ cutoff interp [SAMPLE-SIZE size1] [PICTURE-SIZE size2]}$$

This is a general implementation of the Fourier method (see, e.g., [19, 29, 55, 57]). Roughly speaking, the projection data are first transformed using the one dimensional Fourier transform. This provides us with the values of the two dimensional Fourier transform of the picture on radial lines. From these values the Fourier transform of the picture is estimated at the centers of the pixels of a grid, and the discrete inverse two-dimensional Fourier transform is used to get the reconstructed picture.

More precisely, the method has four stages: (i) Fourier transforming the data; (ii) filtering the Fourier transform; (iii) interpolating the Fourier transform to a square grid; and (iv) taking the inverse Fourier transform. We now explain what happens in each one of these stages.

(i) The variable `nsize` is assigned the value NRAYS + 1 if the integer modifier size1 $<$ USRAYS or is not specified. If size1 $\geq$ USRAYS, `nsize` is either size1 or size1 + 1, whichever is even.

For $0 \leq np <$ PRJNUM, the discrete Fourier transform of the reordered raysums is taken, according to the following C++-like code.

```
ProjFile.ReadProj(np, data, nsize);
for (int i = 0; i < (nsize/2)+1; ++i) {
    x[i] = data[(nsize/2)+i-1] / w(np);
}
for (int i = (nsize/2)+1; i < nsize; ++i) {
    x[i] = data[i-(nsize/2)-1] / w(np);
}
rtfort(x, n, 1);
```

Here `w(np)` is the width of the strips in the npth projection in the PARALLEL STRIP case and is 1 in the PARALLEL LINE or the DIVERGENT case.

At the end of this stage we have that for $0 \le k <$ `nsize`$/2$ the complex number x[$2k$]+ix[$2k$+1] is (i is $\sqrt{-1}$) an estimate of the value of

$$\frac{1}{\text{WIDTH(np)}}[\mathcal{F}f]\left(\frac{k}{n \times \text{WIDTH(np)}}, \theta(\text{np})\right),$$

where $\theta(np)$ is the projection angle of the npth projection; WIDTH(np) is the actual distance between rays in the PARALLEL LINE case, the actual width of the strips in the PARALLEL STRIP case, PINC $\times$ RADIUS / STOD in the DIVERGENT case; and $[\mathcal{F}f](\rho, \phi)$ denotes the Fourier transform of the picture to be reconstructed at the point with polar coordinates.

Restrictions: For $0 \le np \le$ PRJNUM, $\theta(np-1) < \theta(np) \le \theta(np-1) + \alpha$ where α is $\pi/4$ in the PARALLEL VARIABLE case and is $\pi/2$ otherwise (see Section 7.1 for definition of θ(-1) and θ(PRJNUM)).

(ii) Let (ρ, ϕ) be any point for which $[\mathcal{F}f](\rho, \phi)$ has been computed during the previous stage. Filtering the Fourier transform consists of multiplying$[\mathcal{F}f](\rho, \phi)$ by $F(\rho, \phi)/\rho$, where the function is defined exactly the same way it was defined towards the end of Section 7.3. (Restrictions that occurred in the definition of F apply.)

(iii) As a result of the calculations performed in the previous stages we may assume that we know the values of the Fourier transform of the picture, $\mathcal{F}f$, at points with polar coordinates of the form $(k\Delta\rho, \phi)$, where $\phi = \theta(np)$ for some projection np, $0 \le np <$ PRJNUM. For -(nsize/2) $\le k \le$ (nsize/2), the values of $\mathcal{F}f$ are calculated based on the results produced by stage (ii). For $|k| >$ (nsize/2) the value of $[\mathcal{F}f](k\Delta\rho, \phi)$ is assumed to be 0. During stage (iii) we estimate, using interpolation, the values of the Fourier transform of f at points whose Cartesian coordinates are of the form $(m\Delta l, n\Delta l)$, where $\Delta l = 1/(\text{M}\times\text{PIXSIZ})$, -(M/2)+1 $\le m \le$ M/2, $0 \le n \le$ M/2, where M is assigned the value NELEM+1 if the integer modifier size2 $<$ NELEM or is not specified, and is either size2 or size2 + 1 (whichever is even) if size2 $\ge$ NELEM.

In the PARALLEL UNIFORM case or in the DIVERGENT case $\Delta\rho$ is the same for all projections. Hence, for any fixed k, the points $(k\Delta\rho, \phi)$ lie on a circle, as indicated in Figure 7.1. On the other hand, in the PARALLEL VARIABLE case the corresponding points lie on a square, as indicated in Figure 7.2. We use these figures to explain the method of interpolation used to estimate the Fourier transform of f. We shall use the notation $\mathcal{F}(Q)$ to denote our estimate of the Fourier transform of f at the point Q.

Let O denote the origin. We estimate $\mathcal{F}(O)$ to be zero.

Let P be the point for which we wish to find $\mathcal{F}(P)$. We assume that P is not the origin. Let h_1 and h_2 be the two half-lines (starting at the origin) nearest to P on either side such that values of $\mathcal{F}$ are known along h_1 and h_2. We assume that h_2 is in the counter clockwise direction from h_1. Let the points E (respectively F) be defined as the intersection of h_1 (respectively h_2) with the circle whose center is at the origin through P as in Figure 7.1 or with the straight line parallel to the side of the squares through P as in Figure 7.2. Let A and B be the points on h_1 nearest to E on either side at which $\mathcal{F}$ is defined, with A nearer to the origin. Let C and D be the points on h_2 nearest to F on either side at which $\mathcal{F}$ is defined, with C nearest to the origin.

interp determines the interpolation method to be used. (*Restriction*: $1 \le$ interp ≤ 4).

If the value of interp is 1, $\mathcal{F}(P)$is estimated to be $\mathcal{F}(Q)$ where Q is the point nearest to P among A, B, C, and D. If two or more of these points are at an equal distance from P, the order of preference in choosing Q is A, B, C, D.

If the value of interp is 2, $\mathcal{F}(P)$is estimated to be $\mathcal{F}(Q)$ where Q is chosen as follows. We divide the area surrounded by the lines AB and CD and the arcs (or lines) AC and BD into four parts, labelled I, II,

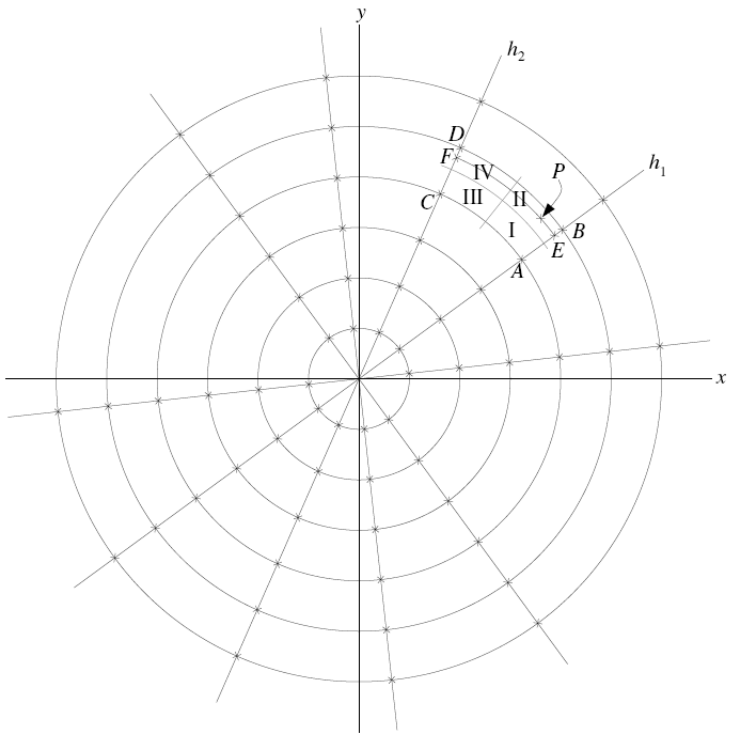

Figure 7.1: Points at which the Fourier transform of a picture can be estimated using the discrete Fourier transform of the projection data in the PARALLEL UNIFORM and DIVERGENT cases.

III and IV, using the arc (line or lines) that connect the half-way points on AB and CD and the half-way points on AC and BD (see Figures 7.1 and 7.2). If P is in I, Q is chosen as A. If P is in II, but not in I, Q is chosen as B. If P is in III, but not in I or II, Q is chosen as C. If P is in IV, but not in I, II, or III, Q is chosen as D.

If the value of interp is 3, $\mathcal{F}(P)$ is estimated as follows. Let, for any two points Q_1 and Q_2, $\overline{Q_1Q_2}$ denote the distance between Q_1 and Q_2. We estimate $\mathcal{F}(E)$ to be $(\overline{BE} \times \mathcal{F}(A) + \overline{AE} \times \mathcal{F}(B))/(\overline{BE} + \overline{AE})$ and $\mathcal{F}(F)$ to be $(\overline{DF} \times \mathcal{F}(C) + \overline{FC} + \mathcal{F}(D))/(\overline{DF} + \overline{FC})$. Let $\overline{\overline{PE}}$ (respectively $\overline{\overline{PF}}$) be the distance along the arc (respectively line or lines) connecting P and E (respectively F) in Figure 7.1 (respectively in Figure 7.2). The is estimated to be $(\overline{\overline{PE}} \times \mathcal{F}(F) + \overline{\overline{PF}} \times \mathcal{F}(E))/(\overline{\overline{PE}} + \overline{\overline{PF}})$.

If the value of interp is 4, $\mathcal{F}(P)$ is estimated as

$$\frac{a \times \mathcal{F}(A) + b \times \mathcal{F}(B) + c \times \mathcal{F}(C) + d \times \mathcal{F}(D)}{a + b + c + d},$$

where and $a = \overline{PB} \times \overline{PC} \times \overline{PD}$, $b = \overline{PA} \times \overline{PC} \times \overline{PD}$, $c = \overline{PA} \times \overline{PB} \times \overline{PD}$, and $d = \overline{PA} \times \overline{PB} \times \overline{PC}$.

(iv) As a result of the last two stages we now have estimates of the (filtered) Fourier transform of the picture at points whose Cartesian coordinates are $(m\Delta l, n\Delta l)$, where -(M/2)+1 $\leq m \leq$ M/2 and $0 \leq n \leq$ M/2. Using `rtfort` we produce from these estimates an M $\times$ M real array, the central NELEM $\times$ NELEM part of which is transferred to `recon` as the reconstructed picture. (The origin is assumed to be at the (M/2,M/2) location of the large array). Finally, the `recon` array is normalized by addition of a constant so that its average density is AVEDEN (see Section 6.3.3).

Warning: The rays of DIVERGENT geometry are treated as if they were PARALLEL with UNIFORM ray spacing PINC $\times$ RADIUS / STOD. A large divergence angle will result in unacceptable reconstructions.

If the iteration number `iter` > 1, this algorithm returns control to `exalg` without changing the contents of the `recon` array. Multiple iterations may nevertheless be used for smoothing or contouring the reconstruction (see Sections 5.8 and 6.1).

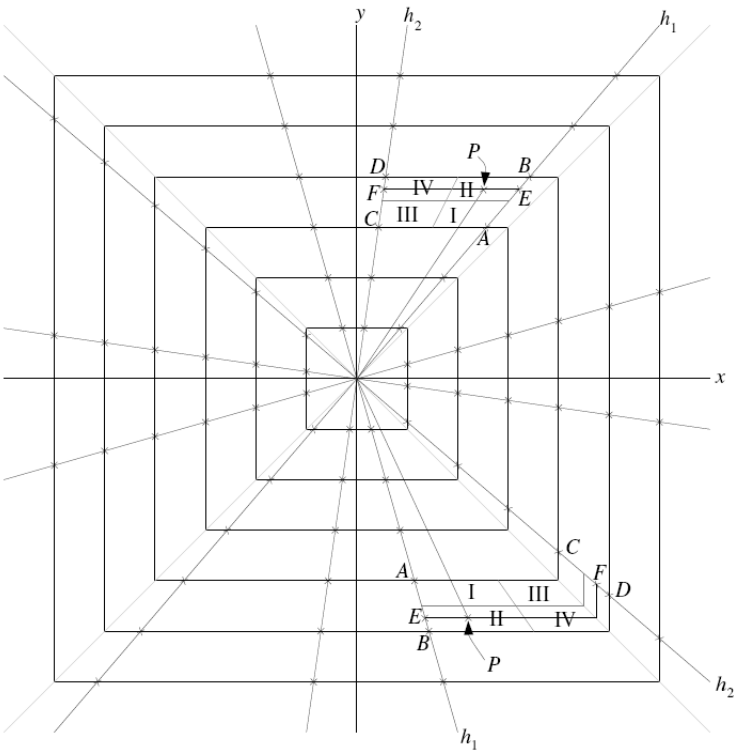

Figure 7.2: Points at which the Fourier transform of a picture can be estimated using the discrete Fourier transform of the projection data in the PARALLEL VARIABLE case.

7.5 DCONV

> interp missp range weight method

This is a general implementation of the filtered backprojection, also known as convolution, method for divergent beam projection data (see [11, 29, 32, 39]). (*Restriction*: This subroutine can only be used for DIVERGENT geometry.) Roughly speaking, the projection data are first convolved with functions based on a "filter", and then the convolved data are weighted and back-projected to form the reconstruction.

More precisely, the reconstruction is calculated projection by projection. Not all the projections on prjfil are necessarily used. The integer modifier missp determines which projections are used: only the first in every missp projections. (*Restriction*: $1 \leq$ missp $<$ PRJNUM.) The procedure for processing data for each projection can be divided into 3 steps, namely, data smoothing, convolution, and back-projection.

Immediately after reading the projection data, `data[i]`, for $0 \leq i <$ NRAYS, each data item is smoothed with its two neighboring items using the parameter specified by the floating point modifier weight. The smoothed data $G(i)$, for $0 \leq i <$ NRAYS, are calculated as follows. (*Restriction*: $0.0 <$ weight ≤ 1.0.)

$$G(i) = \text{weight} \times \text{data}[i] + \frac{1 - \text{weight}}{2} \times (\text{data}[i-1] + \text{data}[i+1]).$$

Here we define both data[-1] and data[NRAYS] to be zero.

The next two steps are best explained separately for the ARC and TANGENT geometries (see Section 2.4). We deal with ARC case first.

In the second step the smoothed data are convolved with two convolving functions $Q1$ and $Q2$ (which will be discussed later). The discrete convolution process can be expressed by the following equation.

$$\text{MODI}(k) = \sum_{i=-N}^{N} [Q1(|i|) \times G(k-i) \times \cos(k-i)\alpha] + \left[\sum_{i=-N}^{N} Q2(|i|) \times G(k-i) \right] \times \cos k\alpha.$$

In the equation above, MODI is the convolved data for $0 \le k <$ NRAYS, α is PINC / STOD and N is the convolution range specified by the integer modifier range by

$$N = \begin{cases} \text{range}, & \text{if } 0 \le \text{range} < \text{NRAYS}, \\ \text{NRAYS-1}, & \text{otherwise.} \end{cases}$$

The values of $G(i)$ for $i < 0$ or $i \ge$ NRAYS are assumed to be zero. (*Restriction*: NRAYS > 1.)

The last step of reconstruction is a weighted backprojection process. The contribution of the npth projection to the reconstructed density of the pixel whose center is at (x, y) is calculated according to the formula

$$B_{np}(x,y) = \frac{[\theta(np\text{+missp}) - \theta(np\text{-missp})] \times \text{RADIUS} \times V}{8\pi^2} \times \text{qintp}(P, \text{MODI}, \text{NRAYS}, \text{INTERP}),$$

where the variables that are not in C++ classes (see Section 6.3) have their values defined as follows. (All these variables are assigned their correct values by SNARK14.)

$\theta(np)$ denotes the projection angle in radians of the npth projection for $0 \le np <$ PRJNUM. For $np \ge$ PRJNUM, $\theta(np)$ is $\theta(0)+2\pi$. For $np < 0$, $\theta(np) = \theta(\text{last-missp})-2\pi$, where last denotes the largest integer not greater than PRJNUM/missp. (*Restriction*: For $-1 \le$ np $\le$ PRJNUM, $\theta(np) < \theta(np+1)$.)

INTERP is assigned the value of the integer modifier interp. (*Restriction*: $-1 \le$ interp ≤ 6.)

`qintp` is the function defined in Section 6.4.5. (*Restriction*: NRAYS must be large enough so that the restrictions stated in Section 6.4.5 are satisfied.)

The definition of V and P depend on the integer modifier method.

$$V = \begin{cases} \frac{1}{((x\sin\theta(np) - y\cos\theta(np))^2 + (\text{RADIUS} - x\cos\theta(np) - y\sin\theta(np))^2)}, & \text{if method} = 0, \\ \frac{1}{\text{RADIUS}^2} + 2\frac{\cos\theta(np)}{\text{RADIUS}^3}x + 2\frac{\sin\theta(np)}{\text{RADIUS}^3}y, & \text{if method} \ne 0, \end{cases}$$

$$P = \begin{cases} \frac{1}{\alpha}\tan^{-1}\frac{-x\sin\theta(np) + y\cos\theta(np)}{\text{RADIUS} - x\cos\theta(np) - y\sin\theta(np)}, & \text{if method} \ge 0, \\ \frac{1}{\alpha}\left[\tan^{-1}\frac{y\cos\theta(np)}{\text{RADIUS} - y\sin\theta(np)} - \frac{\text{RADIUS}\times\sin\theta(np) - y}{\text{RADIUS}^2}\right], & \text{if method} < 0, \end{cases}$$

with $\alpha =$ PINC / STOD. (*Note*: method $= 0$ provides the most accurate but most expensive reconstruction, method < 0 provides the least accurate but least expensive reconstruction, with method > 0 in between; see [39].)

The next line in the INPUT file must be

> { BANDLIMITING | HAMMING | COSINE | PARABOLIC | EXPONENTIAL | SINC | SHEPP-LOGAN | LINE }

The filter is used to specify the convolving functions Q1 and Q2 used in the algorithm.

When a filter other than BANDLIMITING is specified, additional information about the filter has to be supplied by the user. Except for the LINE filter, this information takes the form

> cutoff [alpha]

Restriction: This line is absent if BANDLIMITING is specified and has a different form if LINE is specified.

The values of $Q1(m)$ and $Q2(m)$ for $0 < m \le N$ are determined according to the following equations.

$$Q1(m) = \frac{-m\alpha Q(m\alpha)}{\sin^2(m\alpha)},$$

$$Q2(m) = \frac{Q(m\alpha) + m\alpha Q'(m\alpha)}{\sin(m\alpha)},$$

where

$$Q(u) = 2\pi \int_0^{\frac{C}{2\alpha}} f(\xi) \sin 2\pi\xi u d\xi,$$

and $Q1(0)$ and $Q2(0)$ are defined by their "limit values"; see [39].

The parameter C is specified by the floating point modifier cutoff. (*Restriction*: $0 < C \leq 1$.) When BANDLIMITING or LINE is specified C is automatically set to be 1. With the exception of LINE filter, the "filter" function $f(\xi)$ is determined by the word modifier filter as shown below.

filter	$f(\xi)$
BANDLIMITING	1
HAMMING	alpha + (1 − alpha) cos($2\pi\xi/A$)
COSINE	cos($\pi\xi/A$)
PARABOLIC	$1 - (2\xi/A)^2$
EXPONENTIAL	$(e - e^{2\xi/A})/(e-1)$
SINC	sinc($2\xi/A$)
SHEPP-LOGAN	sinc(ξ/A)

where $A = C\times$STOD/PINC, $\text{sinc}(X) = \sin(\pi x)/(\pi x)$ and the alpha in Hamming filter is specified by the floating point modifier alpha. (*Restriction*: $0 \leq$ alpha ≤ 1.) If alpha is not specified, alpha = 1 is assumed.

In case LINE is specified, users design their own filter function by supplying the following information.

> x_1 $y_1[x_2y_2[x_3y_3[$ ... $[x_ly_l]$ ... $]]]$

Restriction: This line is present only if LINE is specified. *Note*: The information may occupy more than one line. The program will keep reading pairs of floating point modifiers (x_i, y_i) until it comes across a y_l whose value is 0.0. *Restrictions*: $0 \leq x_1 \leq x_2 \leq ... \leq x_l \leq 1$, $1 \geq y_1 \geq y_2 \geq ... \geq y_l = 0$.

This sequence of pairs of numbers defines the filter function in the range [0, $1/2\alpha$] at all points other than 0, $x_1/2\alpha$, $x_2/2\alpha$, ... , $x_l/2\alpha$ as follows. If $\xi > x_l/2\alpha$, $f(\xi) = 0$. Otherwise, let $x_0 = 0$, $y_0 = 1$ and let j be the largest integer such that $\xi > x_j/2\alpha$. Then

$$f(\xi) = \frac{y_j \times (x_{j+1} - 2\xi\alpha) + y_{j+1} \times (2\xi\alpha - x_j)}{x_{j+1} - x_j},$$

and Q, $Q1$ and $Q2$ are defined based on this f by the formulas given above.

The TANGENT geometry is handled somewhat differently. Unless the filter specified is BANDLIMITING, ray sums are estimated, using linear interpolation, for rays that form a DIVERGENT ARC data set with angular spacing $\alpha = \tan^{-1}(\frac{\text{NRAYS-1}}{2} \times \frac{\text{PINC}}{\text{STOD}})/(\frac{\text{NRAYS-1}}{2})$. The algorithm operates on this data set according to the rules described above.

If BANDLIMITING is specified, the convolution step takes the form

$$M(k) = \sum_{i=-N}^{N} \frac{Q1(|i|) \times G(k-i)}{\sqrt{1 + (k-i)^2 \times a^2}},$$

where $a =$ PINC/STOD and all terms have the same meaning as in the ARC case, except for $Q1$, which is defined by

$$Q1(m) = \begin{cases} \frac{\pi^2}{2a^2}, & \text{if } m = 0, \\ -\frac{2}{m^2 \times a^2}, & \text{if } m \text{ is an odd integer}, \\ 0, & \text{otherwise.} \end{cases}$$

The back-projection step is analogous to the back-projection step in the ARC case except for the definition of P, which is

$$P = \begin{cases} \frac{\text{PINC}}{\text{STOD}} \times \frac{-x\sin\theta(np) + y\cos\theta(np)}{\text{RADIUS} - x\cos\theta(np) - y\sin\theta(np)}, & \text{if method} \geq 0, \\ \frac{\text{PINC}}{\text{STOD}} \times \left[\frac{y\cos\theta(np)}{\text{RADIUS} - y\sin\theta(np)} - \frac{\text{RADIUS}\times\sin\theta(np) - y}{\text{RADIUS}^2} x\right], & \text{if method}<0. \end{cases}$$

If the iteration number `iter` > 1, this algorithm returns control to `exalg` without changing the contents of the `recon` array. Multiple iterations may nevertheless be used for smoothing or contouring the reconstruction (see Sections 5.8 and 6.1).

7.6 ART

$$> \left\{ \begin{array}{l} \text{ART3} \\ \text{ART4} \\ \text{BAYESIAN snr} \end{array} \right\} \left[\text{RELAXATION} \left\{ \begin{array}{l} \text{CONSTANT r} \\ \text{VARIABLE} \end{array} \right\} \right] [\text{NORM l}] \left[\text{TOLERANCE} \left\{ \begin{array}{l} \text{FIXED t} \\ \text{VARIABLE} \\ \text{NOISE t} \end{array} \right\} \right]$$

This is a general implementation of the additive Algebraic Reconstruction Techniques (ART). It incorporates the features and options described in [22, 23, 27, 31, 35, 36, 38]. In our description we shall follow, as closely as possible, the notation of [33], which provides a survey of ART-type methods.

The method is an iterative procedure that, starting from an initial estimate of the picture to be reconstructed, updates the estimate through a sequence of steps. A single step is influenced by exactly one ray for which we have an estimate of the ray sum. Only those basis function (pixel or blob) densities are updated that contribute to the associated pseudo ray sum. The updating is done by the addition of a correction term (related to the contribution of the basis function to the pseudo ray sum) to the density in each such basis function, so that after the correction the pseudo ray sum for the ray in question will be nearer to the estimated ray sum.

More precisely, let n be the number of basis functions (either NELEM^2 in the pixel case or Blob.area in the blob case) and m abbreviate PRJNUM $\times$ NRAYS. (In introducing our notation we assume familiarity with Sections 6.2 and 6.3.) We use $x^{(k)}$ to denote the n-dimensional column vector whose jth component, $x_j^{(k)}$ is `recon[j-1]` after the kth step. In particular, $x^{(0)}$ represents the contents of the `recon` array at the beginning of the reconstruction process, as determined by the ZERO, AVERAGE, or CONTINUE option of the EXECUTE command (see Section 5.8). We use p to denote the m-dimensional column vector, whose ith component, p_i, is the value returned by `Anglst.prdta(`np`,`$nr`)` when $i = np\times\text{NRAYS}+nr+1$ with $0 \le np <$ PRJNUM and $0 \le nr <$ NRAYS (see Section 6.3.6.2).

For $1 \le i \le m$, we also define an n-dimensional column vector m_i as follows. Let np and nr be defined as in the last paragraph. Let the integer `numb` and the arrays `list` and `weight` be defined by one of:

```
wray(np, nr, list, weight, &numb, &snorm); // in the pixel LINE case
ray(np, nr, list, weight, &numb, &snorm);  // in the pixel STRIP case
Blob.bwray(np, nr, list, weight, &numb, &snorm); // in the blob LINE case
Blob.bray(np, nr, list, weight, &numb, &snorm); // in the blob STRIP case
```

(see Sections 6.4.2, 6.4.3, 6.3.4.3, and 6.3.4.4). Recall that the blob STRIP case does not result in useful reconstructions (see Section 6.3.4.4). The jth component of m_i, $m_{i,j}$, is defined by

$$m_{i,j} = \left\{ \begin{array}{ll} \text{weight[k]}, & \text{if list[k]} = j-1 \text{ for some k, } 0 \le \text{k} < \text{numb}, \\ 0, & \text{otherwise.} \end{array} \right.$$

Some versions of the ART algorithm also make use of a set of auxiliary variables, one variable for each ray. We let $u^{(k)}$ denote the m-dimensional column vector whose ith component, $u_i^{(k)}$, is the auxiliary variable associated with the nrth ray in the npth projection ($i = np\times\text{NRAYS} + nr + 1$) after the kth step. $u^{(0)}$ is defined to be the vector whose components are all 0. We use e_i to denote the m-dimensional column vector whose only nonzero component is a 1 in the ith row.

At the beginning of the $(k+1)$st step of an ART algorithm $x^{(k)}$ and $u^{(k)}$ are stored in memory. A ray is selected by

```
pick(&np, &nr);
```

(see Section 6.4.1), and i is defined by $np\times\text{NRAYS} + nr + 1$. Then $x^{(k+1)}$ and $u^{(k+1)}$ are defined by

$$\hat{x}^{(k+1)} = x^{(k)} + r^{(k)} c^{(k)} m_i,$$
$$x^{(k+1)} = \Psi(x^{(k)}, \hat{x}^{(k+1)}),$$
$$u^{(k+1)} = u^{(k)} + r^{(k)} d^{(k)} e_i.$$

The function Ψ, which produces the jth component of $x^{(k+1)}$ from the jth components of $x^{(k)}$ and $\hat{x}^{(k+1)}$, will be defined below. The basic difference between the different ART algorithms is the way $c^{(k)}$, $d^{(k)}$, and $r^{(k)}$ are chosen. We first describe this, starting with $r^{(k)}$.

The value of $r^{(k)}$ is determined by what follows the optional word RELAXATION. (If this word is missing,

RELAXATION CONSTANT 1.0

is assumed). If CONSTANT is specified, then the value of $r^{(k)}$ is equal to the value of the floating point modifier r, for all k. If VARIABLE is specified, then the user must replace the default subroutine

```
REAL art_class::rset(
  REAL* recon,
  INTEGER iter,
  INTEGER np,
  INTEGER nr,
  INTEGER iflag,
  REAL raysum,
  INTEGER* list,
  REAL* weight,
  BOOLEAN *flag
);
```

with one that returns the value of $r^{(k)}$. Here `recon`, `list`, `weight`, and `iter` are the formal parameters of the reconstruction algorithm (see Section 6.2); np and nr have been returned by `pick`; the value of the integer variable `iflag` is 1 the first time `rset` is called and it can be changed only by `rset`; `raysum` is the value of p_i; and `flag` is a logical variable whose value is set to TRUE if, and only if, the value of `rset` cannot be calculated for some reason.

The definitions of $c^{(k)}$ and $d^{(k)}$ depend on whether ART3, ART4, or BAYESIAN is specified. They also depend on what follows the optional words NORM and TOLERANCE. We first explain the effect of these words.

The integer modifier following NORM is used to give a definition to the notation $||m||^l$ that we shall use below. (If the word NORM is missing NORM 2 is assumed.)

$$||m_i||^l = \sum_{j=1}^{n} (m_{i,j})^l$$

Restriction: l must be a positive integer.

What follows the optional word TOLERANCE determines the value of the number ϵ that will be used in the formulas for $c^{(k)}$ below. *Restriction*: $\epsilon \geq 0.0$ (If the word TOLERANCE is missing

TOLERANCE FIXED 0.0

is assumed.) If FIXED is specified, ϵ is assigned the value of the floating point modifier t. If VARIABLE is specified, then the user must replace the default subroutine

```
REAL art_class::tset(
  REAL* recon,
  INTEGER iter,
  INTEGER np,
  INTEGER nr,
  INTEGER iflag,
  REAL raysum,
  INTEGER* list,
  REAL* weight,
  BOOLEAN* flag
);
```

with one that returns the value of ϵ. The parameters have the same meaning as for `rset` defined above. If NOISE is specified, the value of ϵ is defined to be t$\times\sqrt{v}$, where t is the floating point modifier following NOISE and

$$v = D \times C \times e^{(p_i/E)} \times E^2 + B \times p_i + A,$$

where

$$A = \begin{cases} (\text{ADDNSD})^2, & \text{if ADDNFL is TRUE,} \\ 0, & \text{otherwise,} \end{cases}$$

$$B = \begin{cases} (\frac{\text{ULTNSD}}{\text{ULTNMN}})^2, & \text{if ULTNFL is TRUE,} \\ 0, & \text{otherwise,} \end{cases}$$

$$C = \begin{cases} \frac{1}{\text{QUANMN} \times \sum_{k=1}^{\text{NERGY}} \text{ENGWT}(k) \times e^{-\text{BACKGR}(k)}}, & \text{if QUANIN>0,} \\ 0, & \text{otherwise,} \end{cases}$$

$$D = \begin{cases} \frac{2.0 \times C}{\text{QUANCM}} + C, & \text{if QUANIN>0,} \\ 0, & \text{otherwise,} \end{cases}$$

$$E = \begin{cases} \text{PINC}, & \text{in the UNIFORM STRIP case,} \\ \text{PINC} \times \max(|\sin\theta_{np}|, |\cos\theta_{np}|), & \text{in the VARIABLE STRIP case,} \\ 1, & \text{in the LINE case.} \end{cases}$$

This υ is an estimate based on the ray sum of the variance of the noise in the ray sum.

The values of $c^{(k)}$ and $d^{(k)}$ depend also on the value of a variable DIFF, which is the difference between the ray sum (as it appears on prjfil) and the pseudo ray sum of a constrained version of the picture in `recon` (represented by $x^{(k)}$). The method of constraining is determined by the contents of the next line in the INPUT file.

> CONSTRAINT { ART2 / BOUND / BART } [CONRELAX { CONSTANT cr / VARIABLE }] [STEPS kount] [NOMLZ]

We define, for $1 \le j \le n$,

$$\overline{x}_j^{(k)} = \begin{cases} \text{LOWER}, & \text{if ART2 is specified, LOFL is TRUE, and } x_j^{(k)} < \text{LOWER}, \\ \text{UPPER}, & \text{if ART2 is specified, UPFL is TRUE, and } x_j^{(k)} > \text{UPPER}, \\ x_j^{(k)}, & \text{otherwise.} \end{cases}$$

DIFF is assigned the value

$$p_i - \sum_{j=1}^{n} m_{i,j} \bar{x}_j^{(k)}.$$

We are now in position to define $c^{(k)}$ and $d^{(k)}$.

Whenever $||m_i||^l \le \text{ZERO}$, $c^{(k)} = d^{(k)} = 0$. Otherwise $c^{(k)}$ and $d^{(k)}$ depend on whether ART3, ART4, or BAYESIAN is specified.

If ART3 is specified,

$$c^{(k)} = \begin{cases} 0, & \text{if } |\text{DIFF}| \le \epsilon, \\ \frac{2 \times (\text{DIFF} - \epsilon)}{||m_i||^l}, & \text{if } \epsilon < \text{DIFF} < 2\epsilon, \\ \frac{2 \times (\text{DIFF} + \epsilon)}{||m_i||^l}, & \text{if } \epsilon < \text{-DIFF} < 2\epsilon, \\ \frac{\text{DIFF}}{||m_i||^l}, & \text{if } 2\epsilon \le |\text{DIFF}|, \end{cases}$$

$$d^{(k)} = 0.$$

If ART4 is specified,

$$c^{(k)} = \text{mid}\left\{ u_i^{(k)}, \frac{\text{DIFF} + \epsilon}{||m_i||^l}, \frac{\text{DIFF} - \epsilon}{||m_i||^l} \right\},$$

$$d^{(k)} = -c^{(k)},$$

where $\text{mid}\{a, b, c\}$ denotes the median of the three numbers a, b, and c.

If BAYESIAN is specified,

$$c^{(k)} = (\text{snr})^2 \frac{\text{DIFF} - u_i^{(k)}}{1 + (\text{snr})^2 ||m_i||^l},$$

$$d^{(k)} = c^{(k)}/(\text{snr})^2.$$

(*Restriction*: $(\text{snr})^2 > \text{ZERO}$.)

To complete the specification of a single step of the ART method we need to describe how, for $1 \le j \le n$, $x_j^{(k+1)}$ is obtained from $\hat{x_j}^{(k+1)}$ and $x_j^{(k)}$.

If ART2 is specified

$$x_j^{(k+1)} = \hat{x}_j^{(k+1)}.$$

If BOUND is specified

$$x_j^{(k+1)} = \begin{cases} \hat{x}_j^{(k+1)} + (\text{cr})^{(k)} \left(\text{LOWER} - \hat{x}_j^{(k+1)}\right), & \text{if LOFL is TRUE and } \hat{x}_j^{(k+1)} < \text{LOWER}, \\ \hat{x}_j^{(k+1)} + (\text{cr})^{(k)} \left(\text{UPPER} - \hat{x}_j^{(k+1)}\right), & \text{if UPFL is TRUE and } \hat{x}_j^{(k+1)} > \text{UPPER}, \\ \hat{x}_j^{(k+1)}, & \text{otherwise}, \end{cases}$$

where $(\text{cr})^{(k)}$ is defined as follows. If CONRELAX is not specified $(\text{cr})^{(k)} = 1$. If CONRELAX is specified, $(\text{cr})^{(k)}$ is obtained by the method we described for $r^{(k)}$ above, except that in the VARIABLE case the user must replace the default subroutine

```
REAL art_class::crset(
  REAL* recon,
  INTEGER iter,
  INTEGER np,
  INTEGER nr,
  INTEGER iflag,
  REAL raysum,
  INTEGER* list,
  REAL* weight,
  BOOLEAN *flag
);
```

with one that returns the value of $(\text{cr})^{(k)}$. The parameters are as described for the function `rset` above.

If BART is specified, the definition of $x_j^{(k)}$ makes use of some auxiliary concepts. Let HALFWAY = (UPPER + LOWER)/2. (*Restriction*: If BART is specified, both LOFL and UPFL must be TRUE.) Let ABOVE = HALFWAY + (UPPER - LOWER)/1000 and BELOW = HALFWAY - (UPPER - LOWER)/1000. In addition, we define two sequences of numbers $\lambda^{(k)}$ and $\mu^{(k)}$, based on the integer modifier kount. (*Restriction*: kount is a positive integer.) If kount is not specified, its value is PRJNUM × SNRAYS if SELECT SNARK is specified or PRJNUM × USRAYS if SELECT USER is specified (see Section 5.6).

$$\lambda^{(0)} = \text{LOWER},$$

$$\lambda^{(k+1)} = \lambda^{(k)} + \frac{\text{HALFWAY} - \text{LOWER}}{\text{kount} \times 2^{\text{iter}}},$$

$$\mu^{(0)} = \text{UPPER},$$

$$\mu^{(k+1)} = \mu^{(k)} - \frac{\text{UPPER} - \text{HALFWAY}}{\text{kount} \times 2^{\text{iter}}}.$$

If BART is specified

$$\tilde{x}_j^{(k+1)} = \begin{cases} \hat{x}_j^{(k+1)} + (\text{cr})^{(k)} \left(\text{LOWER} - \hat{x}_j^{(k+1)}\right), & \text{if } \hat{x}_j^{(k+1)} < \lambda^{(k)}, \\ \hat{x}_j^{(k+1)} + (\text{cr})^{(k)} \left(\text{UPPER} - \hat{x}_j^{(k+1)}\right), & \text{if } \hat{x}_j^{(k+1)} > \mu^{(k)}, \\ \hat{x}_j^{(k+1)}, & \text{otherwise}, \end{cases}$$

where $(\mathrm{cr})^{(k)}$ is defined just as in the case when BOUND is specified.

$$x_j^{(k+1)} = \begin{cases} \text{BELOW}, & \text{if } x_j^{(k)} \le \lambda^{(k)} \text{ and } \tilde{x}_j^{(k+1)} \ge \text{HALFWAY}, \\ \text{ABOVE}, & \text{if } x_j^{(k)} \ge \mu^{(k)} \text{ and } \tilde{x}_j^{(k+1)} \le \text{HALFWAY}, \\ \tilde{x}_j^{(k+1)}, & \text{otherwise.} \end{cases}$$

This completes the description of a single step of an ART algorithm. One SNARK14 iteration of the algorithm (see Section 6.1) consists of kount steps. (*Restriction*: kount > 0.) The contents of the `recon` array at the end of the iteration with iteration number `iter` are described by $x^{(k)}$ with $k =$ `iter` $\times$ kount, unless NOMLZ is specified. If NOMLZ is specified, then prior to returning control to `exalg` the ART subroutine adds to each entry of `recon` the value (AVEDEN $-\frac{1}{n}\sum_{j=1}^{n} x_j^{(k)}$). If another iteration is called, the initial value of $x_j^{(k)}$ for the new iteration will be the content of `recon[`$j-1$`]` at the end of the last iteration.

7.7 SART

> SART [RELAXATION CONSTANT r]

SART is an implementation of the additive Simultaneous Algebraic Reconstruction Technique algorithm according to [2, 46, 47]. (The intended aim of SART is to minimize the weighted squared distance WS as defined in (5.18).) The SART algorithm is based on the ART algorithm of the previous section and so we use the notation from that section to specify the iterative step of SART from $x^{(k)}$ to $x^{(k+1)}$: For $1 \le j \le n$,

$$x_j^{(k+1)} = x_j^{(k)} + \mathrm{r}\,\frac{1}{\sum_{i=1}^{m} m_{i,j}} \sum_{i=1}^{m} \left\{ a_{i,j} \frac{p_i - \sum_{j'=1}^{n} m_{i,j'} x_{j'}^{(k)}}{\sum_{j'=1}^{n} m_{i,j'}} \right\}. \tag{7.1}$$

The value of r is determined by what follows the option RELAXATION CONSTANT. If this option is not chosen, then r is given the default value 1. *Restriction:* $0 < \mathrm{r} \le 2$. If this restriction is violated, then r is given the default value 1.

7.8 MART

> METHOD $\left\{ \begin{array}{l} \text{GBH} \\ \text{LENT} \\ \text{LAK1} \\ \text{LAK2} \end{array} \right\}$ kount relax data-zero [NORMAL] [ENTROPY]

This is a general implementation of the multiplicative Algebraic Reconstruction Techniques (MART. It incorporates features and options described in [18, 19, 52], as well as two versions intended to reduce computer time invented by A.V. Lakshminarayanan (unpublished). In our description we shall follow, as closely as possible, the notation of [33], which provides a survey of ART-type methods.

The method is an iterative procedure that, starting from an initial estimate of the picture to be reconstructed, updates the estimate through a sequence of steps. A single step is influenced by exactly one ray for which we have an estimate of the ray sum. Only those basis function densities that contribute to the associated pseudo ray sum are updated. The updating is done by multiplying by a correction term (related to the contribution of the basis function to the pseudo ray sum) the density in each such basis function, so that after the correction the pseudo ray sum for the ray in question will be nearer to the estimated ray sum.

For the more precise description we adapt the definitions given in Section 7.6 for n, m, $x^{(k)}$, p, m_i, and $m_{i,j}$.

There are four versions of MART. All MART methods assume that ray sums are nonnegative. (*Restriction*: AVEDEN > ZERO.) This is insured by the use of the floating point modifier data-zero. (*Warning*:

If data-zero < 0, the program will operate as if data-zero were specified to be 0.0.) The algorithms use an m-dimensional column vector q, whose ith component, q_i, is defined by

$$q_i = \begin{cases} 0, & \text{if } p_i \le \text{data-zero}, \\ p_i, & \text{if } p_i > \text{data-zero}. \end{cases}$$

The first difference between the methods is the way the initial estimate is defined. (*Warning*: This overrides how `recon` is initialized according to the EXECUTE control command line.)

In Method GBH, $x_j^{(0)} = \text{AVEDEN}$, for $1 \le j \le n$.

Let J be the set of all js such that $1 \le j \le n$, and there exists an i satisfying $1 \le i \le m$, $q_i = 0$, and $m_{i,j} > 0$. (In other words, $j \in J$ if the jth basis function is intersected by a ray whose modified ray sum is 0). If every j between 1 and n is in J, all elements of the `recon` array are set to 0, a warning message is printed, and control is passed back to `exalg` (see Section 6.1). Otherwise, suppose that there are u values of j between 1 and n that are not in J. In Methods LENT, LAK1, and LAK2, we define, for $1 \le j \le n$,

$$x_j^{(0)} = \begin{cases} 0, & \text{if } j \in J, \\ \frac{n}{u} \times \text{AVEDEN}, & \text{if } j \notin J. \end{cases}$$

At the beginning of the $(k+1)$st step of a MART algorithm, $x^{(k)}$ is stored in memory. A ray is selected by

```
pick(&np, &nr);
```

(see Section 6.4.1), and i is defined by

$$i = np \times \text{NRAYS} + nr + 1.$$

We also define s_i to be $\sum_{j=1}^{n} m_{i,j} x_j^{(k)}$. (Note that it will always be the case that $s_i \ge 0$.)

If $q_i = 0$,

$$x_j^{(k+1)} = \begin{cases} 0, & \text{if } m_{i,j} > 0, \\ x_j^{(k)}, & \text{otherwise.} \end{cases}$$

If $q_i > 0$ and $s_i = 0$, $x^{(k+1)} = x^{(k)}$.

If $q_i > 0$ and $s_i > 0$, then the four methods behave differently.

In Methods GBH and LENT,

$$x_j^{(k+1)} = x_j^{(k)} \left(\frac{q_i}{s_i} \right)^{(r m_{i,j})/A_i},$$

where r is the value of the floating point modifier relax if relax > 0, and r is 1 if relax ≤ 0; and $A_i = 1$ in Method GBH and $A_i = \max_{1 \le j \le n}(m_{i,j})$ in Method LENT.

For Methods LAK1 and LAK2, we define, for $1 \le j \le n$,

$$\alpha_j = \frac{r m_{i,j}}{A},$$

where r is the value of the floating point modifier relax if relax > 0, and r is 1 if relax ≤ 0; and $A = \max_{1 \le j \le n, 1 \le i \le m}(m_{i,j})$. In Method LAK1,

$$x_j^{(k+1)} = \begin{cases} x_j^{(k)} \left(1 + \frac{q_i - s_i}{q_i} \alpha_j \right), & \text{if } q_i \ge s_i, \\ x_j^{(k)} \left(1 + \frac{q_i - s_i}{s_i} \alpha_j \right), & \text{if } q_i < s_i. \end{cases}$$

In Method LAK2,

$$x_j^{(k+1)} = \begin{cases} x_j^{(k)} \left[1 + \frac{q_i - s_i}{q_i} \alpha_j \left(1 + \frac{1}{2} \frac{q_i - s_i}{q_i} (\alpha_j + 1) \right) \right], & \text{if } q_i \ge s_i, \\ x_j^{(k)} \left[1 + \frac{q_i - s_i}{s_i} \alpha_j \left(1 + \frac{1}{2} \frac{q_i - s_i}{s_i} (\alpha_j - 1) \right) \right], & \text{if } q_i < s_i. \end{cases}$$

This completes the description of a single step of a MART algorithm. One SNARK14 iteration of the algorithm (see Section 6.1) consists of kount steps. (If kount ≤ 0, it is treated as if its value was PRJNUM $\times$ SNRAYS if SELECT SNARK was specified or PRJNUM $\times$ USRAYS if SELECT USER was specified (see Section 5.6). The contents of the `recon` array at the end of the iteration with iteration number iter are described by $x^{(k)}$, with $k =$ `iter` $\times$ kount, unless NORMAL is specified and $\sum_{j=1}^{n} x_j^{(k)} >$ ZERO. In that case, prior to returning control to exalg the MART subroutine multiplies each entry of `recon` by AVEDEN/$\sum_{j=1}^{n} x_j^{(k)}$. If another iteration is called, the initial value for the new iteration will be the contents of `recon` at the end of the last iteration.

If ENTROPY is specified, SNARK14 will print at the end of an iteration by MART the value of the function

$$-\sum_{j=1}^{n} \frac{\text{recon}[j-1]}{\text{TOTAL}} \times \ln\left(\frac{\text{recon}[j-1]}{\text{TOTAL}}\right),$$

provided that TOTAL$= \sum_{j=1}^{n} \text{recon}[j-1] >$ ZERO.

7.9 QUADRATIC

> algorithm delta-flag d-flag min-eigen period toler delta gamma

Restrictions: See Table 7.1.

This is an implementation of the quadratic optimization techniques as described in [4, 34]. Roughly speaking, the algorithm minimizes a quadratic function on the vector of basis function densities with an iterative process of the type

$$x^{(k+1)} = x^{(k)} + \delta^{(k)} g^{(k)},$$

where $x^{(0)}$ is an initial guess, and $\delta^{(k)}$ and $g^{(k)}$ are a scalar and a vector that depend on the particular quadratic function and minimization algorithm.

For the more precise description we adapt the definitions given in Section 7.6 for n, m, $x^{(k)}$, p, and m_i. We use M to denote the $m \times n$ matrix whose ith row is the transpose of m_i. The reconstruction aims at finding an x^* that minimizes

$$k(x) = a(p - Mx)^T A(p - Mx) + (x - \bar{x})^T \left(bB^l + cC^{-t}\right)(x - \bar{x}), \tag{7.2}$$

and if more than one x minimizes $k(x)$, then x^* is the one among all the ones that minimize $k(x)$ that also minimizes $||D^{-1}x||$. Here A, B, C, and D are matrices, $\bar{x}$ is a vector, a, b, c, and d are scalars, and l is an integer, which will be discussed later. We use Z^T to denote the transpose of the matrix Z. The algorithm minimizes $k(x)$ by attempting to find a solution to an equivalent problem, which can be stated as follows. Find the minimum norm solution y^* of

$$Py = f,$$

where

$$P = \begin{cases} aC^{t/2}M^T AMC^{t/2} + bC^{t/2}B^l C^{t/2} + cC^{t-1}, & \text{if } b + c > 0, \\ DM^T AMD, & \text{if } b + c = 0, \end{cases}$$

$$f = \begin{cases} aC^{t/2}M^T A(p - M\bar{x}), & \text{if } b + c > 0, \\ DM^T Ap, & \text{if } b + c = 0. \end{cases}$$

It has been shown that

$$x^* = \begin{cases} C^{t/2}y^* + \bar{x}, & \text{if } b + c > 0, \\ Dy^*, & \text{if } b + c = 0. \end{cases}$$

Our description of the algorithm will be in terms of a sequence $y^{(0)}$, $y^{(1)}$, $y^{(2)}$, ... This is purely a descriptive device; the contents of `recon` at the end of the kth iteration are the components of $x^{(k)}$, where

$$x^{(k)} = \begin{cases} C^{t/2}y^{(k)} + \bar{x}, & \text{if } b + c > 0, \\ Dy^{(k)}, & \text{if } b + c = 0. \end{cases}$$

The description consists of three parts: the basic method for getting from $y^{(k)}$ to $y^{(k+1)}$ (choosing the algorithm), the way the function $k(x)$ is determined (defining the function), and additional features that can be incorporated into the quadratic optimization reconstruction procedures.

name	type	restrictions	described in	on pages
algorithm	integer	$\geq 1, \leq 4$	7.9.1	98-99
delta-flag	integer	$\geq -2, \leq 3$	7.9.1	98-99
d-flag	integer	$\geq 1, \leq 4$	7.9.2	100
min-eigen	integer	none	7.9.1	98
period	integer	≥ 1	7.9.1	99
toler	floating point	>ZERO	7.9.1	98
delta	floating point	none	7.9.1	98-99
gamma	floating point	none	7.9.1	98-99
order	integer	$\geq 1, \leq 3$	7.9.1	99
error-type	integer	$\geq 1, \leq 3$	7.9.2	100
exp-value	integer	$\geq 1, \leq 5$	7.9.2	100
split	integer	$\geq 0, \leq 2$	7.9.1	100
normalize	integer	$\geq 0, \leq 1$	7.9.3	102
print	integer	$\geq 0, \leq 2$	7.9.3	102
error	floating point	>ZERO	7.9.2	100
minv	floating point	>ZERO	7.9.2	100
interp	integer	$\geq -1, \leq 6$	7.9.2	100

Table 7.1: Summary of the modifiers on the first two follow up lines for the QUADRATIC algorithm. *Warning*: The values of some of these modifiers are sometimes not relevant to defining what happens in the algorithm (e.g., order is used only if delta-flag = 2). Nevertheless, a value in the proper range must be given to all the modifiers.

7.9.1 Choosing the algorithm

The choice of algorithm depends on the values of the integer modifiers algorithm, delta-flag, min-eigen, period and the floating point modifiers toler, delta, and gamma, as well as the integer modifiers order and split, which appear on the next line. Since the seventeen modifiers appearing on those two lines are likely to be confusing, a summary of their use appears in Table 7.1.

> order error-type exp-value split normalize print error minv interp

Restrictions: See Table 7.1.

In defining the sequence of $y^{(k)}$s our descriptions make use of sequences of other vectors and scalars. All the different algorithms use an n-dimensional vector $g^{(k)}$ and a scalar $\delta^{(k)}$. Other vectors and scalars will be introduced as and when they are needed.

If algorithm = 1 or algorithm = 2, the sequences are defined as follows.

$$g^{(0)} = f - Py^{(0)},$$

$$y^{(k+1)} = y^{(k)} + \delta^{(k)} g^{(k)},$$

$$g^{(k+1)} = \begin{cases} f - Py^{(k+1)}, & \text{if algorithm=1,} \\ g^{(k)} - \delta^{(k)} Pg^{(k)}, & \text{if algorithm=2.} \end{cases}$$

The value of $\delta^{(k)}$ is defined as follows. (*Restriction*: In what follows $\delta^{(k)}$ is defined as a fraction. Except in the case delta-flag = 3, all values of $\delta^{(k)}$ that are needed during the iterative process are calculated before the first iteration. The denominator in the definition of all of these $\delta^{(k)}$s must be greater than ZERO.)

If delta-flag = 0, then $\delta^{(k)}$ = delta.

If delta-flag = 1, then $\delta^{(k)} = 2/(\bar{\lambda} + \underline{\lambda})$, where $\bar{\lambda}$ and $\underline{\lambda}$ are supposed to be estimates of the largest and smallest eigenvalues of P. These are estimated by the iterative procedure described in [34]. That procedure gives a sequence of upper bounds and lower bounds for $\bar{\lambda}$. The procedure stops when the difference between these bounds is less than toler times the upper bound. The estimate provided for $\underline{\lambda}$ is replaced by 0 if min-eigen = 0.

If delta-flag = -1, then $\delta^{(k)} = 2/(\bar{\lambda} + \underline{\lambda})$, where $\bar{\lambda}$ = delta and $\underline{\lambda}$ = gamma.

If delta-flag = 2, then

$$\delta^{(k)} = \frac{2}{\bar{\lambda} + \underline{\lambda} - (\bar{\lambda} - \underline{\lambda})\cos((2q(k) - 1) \times \pi/2N)},$$

where $\bar{\lambda}$ and $\underline{\lambda}$ are defined as in the case delta-flag = 1, N is the value of the integer modifier period and q is a function whose definition depends on the integer modifier order. In any case, $q(k) = \sigma(i)$, where $0 \le i < N - 1$ and $k = jN + i$ for some integer j. If order = 1, then $\sigma(i) = i + 1$. If order = 2, the values of $\sigma(i)$ are given on a (sequence of) line(s) in the INPUT file.

> $\sigma(0)$ $\sigma(1)$ ... $\sigma(N-1)$

(*Restrictions*: This line is absent unless algorithm is 1 or 2, delta-flag is -2 or 2 and order is 2. For $0 \le i \le N - 1$, $1 \le \sigma(i) \le N$.) If order = 3, then $\sigma(i)$ is defined by the following C++ like code that generates a permutation of the integers from 1 to 2^U. (This definition of $\sigma(i)$ is advisable to achieve stability, see [3].) *Restriction*: If order = 3, then N must be of the form 2^U, where $U \ge 1$.

```
sigma[0] = 1;
int k = 1;
for (int v = 1; v <= U; ++v) {
    int n = 2*k + 1;
    for (int m = 1; m <= k; ++m) {
        int i = k - m;
        sigma[2*i+1] = n - sigma[i];
        sigma[2*i] = sigma[i];
    }
    k *= 2;
}
```

If delta-flag = -2, then $\delta^{(k)}$ is defined the same way as in the case delta-flag = 2, except that $\bar{\lambda}$ = delta and $\underline{\lambda}$ = gamma.

If delta-flag = 3, then

$$\delta^{(k)} = \frac{\langle g^{(k)}, g^{(k)} \rangle}{\langle g^{(k)}, Pg^{(k)} \rangle},$$

where the inner product $\langle x, y \rangle$ of two n-dimensional vectors x and y (with components x_i and y_i, respectively) is defined to be $\sum_{i=1}^{n} x_i y_i$. If $\langle g^{(k)}, Pg^{(k)} \rangle <$ ZERO, no iterative steps past the kth step will be carried out.

If algorithm = 3 or algorithm = 4, we need an additional n-dimensional vector $r^{(k)}$ and additional scalar $\gamma^{(k)}$. The sequences are defined as follows.

$$g^{(0)} = r^{(0)} = f - Py^{(0)},$$

$$y^{(k+1)} = y^{(k)} + \delta^{(k)} g^{(k)},$$

$$r^{(k+1)} = \begin{cases} f - Py^{(k+1)}, & \text{if algorithm} = 3, \\ r^{(k)} - \delta^{(k)} Pg^{(k)}, & \text{if algorithm} = 4, \end{cases}$$

$$g^{(k+1)} = r^{(k+1)} + \gamma^{(k)} g^{(k)}.$$

The values of $\delta^{(k)}$ and $\gamma^{(k)}$ are defined as follows. (*Restriction*: If algorithm is 3 or 4, delta-flag must be 0 or 3.)

If delta-flag = 0, then $\delta^{(k)}$ = delta and $\gamma^{(k)}$ = gamma.

If delta-flag = 3, then

$$\delta^{(k)} = \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle g^{(k)}, Pg^{(k)} \rangle},$$

$$\gamma^{(k)} = \frac{\langle r^{(k+1)}, r^{(k+1)} \rangle}{\langle r^{(k)}, r^{(k)} \rangle}.$$

If the denominator in the definition of either $\delta^{(k)}$ or $\gamma^{(k)}$ is less than ZERO, no iterative step past the kth step will be carried out.

All these techniques may possibly be accelerated by the method of *splitting* or *scaling*. This means that instead of solving $Py = f$ we solve $P'y' = f'$, where $P' = SPS$, $y' = S^{-1}y$, and $f' = Sf$ and then setting $y = Sy'$. If split = 0, then $S = I$. If split > 0, S is an $m \times m$ diagonal matrix whose jth entry is the jth component of the array SV returned by the user supplied procedure:
if split = 1:

```
void quad_class::seset(REAL* sv, INTEGER* list, REAL* weight, BOOLEAN* alg);
```

if split = 2:

```
void quad_class::sset(REAL* sv, INTEGER* list, REAL* weight, BOOLEAN* alg);
```

where `list` and `weight` are defined in Section 6.2. `alg` is a logical variable whose value is set to TRUE if, and only if, the subroutine fails to calculate the value of `sv` for some reason.

7.9.2 Defining the functional

We now discuss how the matrices A, B, C, D, the vector $\bar{x}$, the scalars a, b, c, and the integers l and t are specified for the definition of the function k, the matrix P and the vector f.

A is an $m \times m$ diagonal matrix whose definition depends on the integer modifier error-type and floating point modifiers error and minv. (*Restrictions*: 1 ≤ error-type ≤ 3, error > ZERO, and minv > ZERO.) If error-type = 1, then every diagonal element of A is equal to error. If error-type = 2, the ith diagonal entry of A is $1/\max\{\text{minv}, v\}$ where v is defined by the formulas on page 92. If error-type = 3, then the value of the ith diagonal element of A is the reciprocal of the value returned by a user supplied procedure:

```
REAL quad::uerror(INTEGER np, INTEGER nr, REAL pd, BOOLEAN* alg);
```

where $i = np \times \text{NRAYS} + nr + 1$, `pd` $= p_i$ and `alg` is a logical variable whose value is set to TRUE if, and only if, the function fails to calculate the value of `uerror` for some reason. *Warning*: It is the user's responsibility to insure that the value returned by this function is greater than ZERO.

D is an $n \times n$ diagonal matrix that is needed only if $b + c = 0$. Its definition depends on the value of the integer modifier d-flag. (*Restriction*: 1 ≤ d-flag ≤ 4.) If d-flag = 1, then the jth diagonal element of D is given by

$$\left(\sum_{i=1}^{m} \frac{m_{i,j}^2}{\sigma_i^2} \right)^{-1/2},$$

where σ_i is the ith diagonal element of the matrix A defined above. (*Restriction*: If d-flag is 1 or 2, then for every j, such that $1 \le j \le n$, there is at least one i, such that $m_{i,j} \neq 0$.)

If d-flag = 2, then the jth diagonal element of D is given by

$$\left(\sum_{l=1}^{n} \left[M^T M \right]_{j,l} \right)^{-1/2},$$

where $[M^T M]_{j,l}$ denotes the (j,l)th element of the matrix $M^T M$.

If d-flag = 3, then the jth diagonal element of D is the jth component of the array `dv` returned by a user supplied procedure:

```
void quad::dset(REAL* dv, INTEGER* list, REAL* weight, BOOLEAN* alg);
```

where `list` and `weight` are defined in Section 6.2, and `alg` is a logical variable whose value is set to TRUE if, and only if, the function fails to calculate the value of `dset` for some reason.

If d-flag = 4, D is the $n \times n$ identity matrix.

$\bar{x}$ is an n-dimensional vector whose definition depends on the integer modifiers exp-value and interp. (*Restrictions*: 1 ≤ exp-value ≤ 5, −1 ≤ interp ≤ 6.)

If exp-value = 1, $\bar{x}_j = 0$, for $1 \le j \le n$.

If exp-value = 2, $\bar{x}_j$ = AVEDEN, for $1 \le j \le n$.

If exp-value = 3, $\bar{x}$ is the picture of size n, which is related to an NELEM×NELEM picture in the following way. In the pixel case, $\bar{x}$ is the picture that would be produced from the given projection data by

EXECUTE BACKPROJECTION
name-of-the-execution
DISCRETE MULTIPLICATIVE

(see Section 7.1), with the usual indexing convention (described in Section 6.2). In the blob case, $\bar{x}$ is the picture returned by pix2blob (see Section 6.3.4.2) when it is given the picture defined above.

If exp-value = 4, $\bar{x}$ is the picture of size n, which is related to an NELEM×NELEM picture in the following way. In the pixel case, $\bar{x}$ is the picture that would be produced from the given projection data by

EXECUTE BACKPROJECTION
name-of-the-execution
CONTINUOUS interp MULTIPLICATIVE

In the blob case, $\bar{x}$ is the picture returned by pix2blob (see Section 6.3.4.2) when it is given the picture defined above.

If exp-value = 5, then $\bar{x} = x^{(0)}$, the contents of the `recon` array prior to the first iteration of the reconstruction procedure (see Section 5.8).

If d-flag is 1, 2 or 3, then $a = 1$ and $b = c = 0$. If d-flag is 4, then the constants a, b, and c are given the values of the floating point modifiers a, b, and c, which appear on the next line in the file INPUT.

> a b c

Restrictions: a $\ge$ 0.0, b $\ge$ 0.0, and c $\ge$ 0.0. This line is absent unless d-flag = 4.

B is an $n \times n$ matrix, determined by the values on the next line in the file INPUT.

> l bw1 bw2 bw3 { SNARK / SAJZ / SAJC / KAMI }

Restriction: This line is absent unless d-flag = 4. The value of the integer modifier l is assigned to the l in the definition (7.2) of the function k (*restriction*: $l > 0$). The matrix B is defined based on the floating point modifiers bw1, bw2, and bw3 and the following word (SNARK, SAJZ, SAJC, or KAMI) so that the vector represents a picture that is a smoothed version of the picture represented by the vector x. Smoothing pixel and blob images are different enough to require separate explanations.

For pixel pictures, we make use of the notation V_1, ... , V_9, W_1, W_2, W_3 introduced in Section 5.8, with W_1, W_2, and W_3 equal to the values of the floating modifiers bw1, bw2 and bw3, respectively. Also, the smoothing operates in such a way as if the threshold were so large that $|V_i - V_1| \le$ threshold is always satisfied. If SNARK is specified, the smoothing performed by B is exactly the smoothing described in Section 5.8. (*Restriction*: If SNARK is specified, then $W_1 >$ ZERO, $W_2 \ge 0.0$ and $W_3 \ge 0.0$.) If SAJZ is specified, the smoothing performed by B is the same as the smoothing described in Section 5.8, except that the denominator in the formula on page 46 is replaced by 1. If SAJC is specified, the smoothing operates as follows. Put an extra layer of pixels all around the digitized picture. Assign to every boundary pixel the density in the nearest interior pixel. Apply the smoothing described in the case SAJZ to the extended picture, and then remove all the boundary pixels. The resulting picture is represented by the vector Bx. If KAMI is specified, the smoothing operates as follows. (*Restriction*: KAMI is not allowed unless, $l = 1$.) Apply the smoothing described in the case SAJZ. Set the density in every boundary pixel to 0.0. Apply again the smoothing described in the case SAJZ.

For blob pictures only the weights bw1 and bw2 are used. If SNARK is specified, the value of each blob is replaced with the weighted average of the central blob and the up to six surrounding blobs, where the weight of the central blob is bw1 and the weight of the surrounding blobs is bw2. If SAJZ is specified, the smoothing is the weighted sum of the central blob and the surrounding blobs. This differs from the SNARK case in that the resulting value is not scaled by the sum of the weights. If SAJC is specified, the smoothing operates as follows. Put an extra layer of blobs all around the digitized picture. Each boundary blob is closest to 1, 2, or 3 image blobs. Assign to each boundary blob the average of the nearest image

blobs. Apply the smoothing described by SAJZ to the extended picture, then remove the boundary pixels. If KAMI is specified, apply the smoothing described in the case SAJZ, set the density in each boundary blob to 0.0, and apply the smoothing described in the case SAJZ again. (*Restriction*: KAMI is not allowed unless, $l = 1$.)

C is an $n \times n$ matrix, determined by the values on the next line in the file INPUT.

> t cw1 cw2 cw3 { SNARK | SAJZ | SAJC | KAMI }

Restriction: This line is absent unless d-flag = 4. If the value of the integer modifier $t = 0$, C is taken to be the identity matrix. Otherwise, the value of the integer modifier t is assigned to the t in the definitions of the function k, the matrix P and the vector f on page 97. (*Restrictions*: $t \geq 0$. If algorithm > 1 or delta $= 3$, t must be even.) If $t > 0$, the matrix C is defined based on the floating point modifiers cw1, cw2, and cw3 and the word modifier in exactly the same way as the matrix B was defined above based on bw1, bw2, bw3 and its word modifier.

7.9.3 Additional features

One SNARK14 iteration of the QUADRATIC algorithm (see Section 6.1) consists of a single step of the kind described in Section 7.9.1. The contents of the `recon` array at the end of the iteration with iteration number k are obtained from $x^{(k)}$ in a way that depends on the value of the integer modifier normalize. (*Restriction*: The value normalize is 0 or 1.) This is best described by the following C++ like code.

```
double total = 0.0;
for (int i = 0; i < n; ++i) {
    double v = recon[i];
    if (lofl) v = max(v, lower);
    if (upfl) v = min(v, upper);
    total += v;
    recon[i] = v;
}
if (normalize != 0) {
    if (abs(total) > Consts.zero) factor = nelem*nelem*aveden/total;
    // factor is not needed if condition is not met.
    for (int i = 0; i < n; ++i) {
        recon[i] = (abs(total) > Consts.zero) ? recon[i]* factor : aveden;
    }
}
```

The user may obtain information about the progress of the iterative procedure by the use of the integer modifier print. (*Restriction*: $0 \leq$ print ≤ 2.) If print ≥ 1, the following information is reported on the file OUTPUT. If delta-flag is 1 or 2, then the estimates of $\underline{\lambda}$ and $\bar{\lambda}$ calculated by the program are reported. At the end of each iteration the following values are also reported: $||f - Py^{(k)}||^2$ (provided algorithm$\neq 1$), `factor` (defined by the code given above, provided it is calculated), $||\delta^{(k-1)}g^{(k-1)}||$, $\delta^{(k-1)}$ and $||g^{(k-1)}||$. If print $= 2$ and $t = 0$, the value of $k(x^{(k)})$ is also reported. (If d-flag$\neq 4$, t is not read in and defaults to the value 0.)

7.10 SIRT

> METHOD { GSIRT | LSIRT type } [RELAX relax | SIGMA sigma] [START [BACKPROJECTED]] [NORMAL]

This is an implementation of four generalized and modified versions of the (additive) Simultaneous Iterative Reconstruction Technique (SIRT) of [17]. The generalizations are presented in [51]. In our description we shall follow, as closely as possible, the notation of [33]. (*Restriction*: This SIRT routine is applicable

only to PARALLEL STRIP projection data. The QUADRATIC routine contains a SIRT-type algorithm for other types of projection data, see Section 7.9.)

SIRT is an iterative procedure that, starting from an initial estimate of the picture to be reconstructed, updates the estimate through a sequence of steps. Roughly speaking, the correction at each update is the discrete backprojection of a set of "projection error data" that consists of the differences between all the estimated ray sums and corresponding pseudo ray sums from the current estimate of the image. In addition, there are some scaling operations involved in the update step that will be discussed in detail, as they are the distinguishing features of the different versions of the SIRT algorithm.

For the more detailed description we adopt the definitions in Section 7.6 for n, m, $x^{(k)}$, p, m_i, and $m_{i,j}$. In addition, for $1 \le j \le n$, we define a set of integers R_j as follows.

The set R_j depends on the most recently executed SELECT command (see Section 5.6). If no SELECT command has been executed,

SELECT EFFICIENT

is assumed.

We define two variables F and L as follows. If USER is specified by SELECT, F = FUSRAY and L = LUSRAY. If SNARK is specified by SELECT, F = FSNRAY and L = LSNRAY (see Section 2.5).

An integer i is in R_j if it satisfies the following conditions: (a) i is of the form $np \times$ NRAYS + nr + 1 with $F \le nr \le L$. (b) $m_{i,j} \neq 0$. (c) $d_i \le c_i - (w_i/1000)$, where d_i denotes the distance of the central line of the nrth ray of the npth projection from the origin, w_i denotes the width of the rays in the npth projection, and c_i is the distance from the origin to a line A that is defined as follows. A is parallel to the rays of the npth projection, it goes through a vertex of the reconstruction region, and among the (up to four) lines that have these properties, A is the one that is furthest from the origin. The purpose of this last condition is to ensure that, for $i \in R_j$, the central line of the nrth ray of the npth projection properly intersects the reconstruction region.

At the beginning of the $(k+1)$st step of a SIRT algorithm, $x^{(k)}$ is stored in the `recon` array. Then $x^{(k+1)}$ is defined to be the n-dimensional column vector whose jth component is $x_j^{(k+1)}$, where

$$x_j^{(k+1)} = \begin{cases} x_j^{(k)} + r\left[b_j - \dfrac{\sum_{i\in R_j} f_i \sum_{t=1}^{n} m_{i,t} x_t^{(k)}}{w \times d_j}\right], & \text{if } d_j \neq 0, \\ x_j^{(k)}, & \text{if } d_j = 0, \end{cases}$$

and

$$w = \begin{cases} (\text{PIXSIZ})^2, & \text{in the pixel case,} \\ \sqrt{3} \times (\text{DELTA})^2/2, & \text{in the blob case.} \end{cases}$$

Here b and d are n-dimensional vectors and f is an m-dimensional vector. The precise definition, to be given below, of the components b_j, d_j, and f_i is what distinguishes the different versions of SIRT. The variable r denotes a relaxation parameter that is defined as follows.

If RELAX is specified, r is assigned the value of the floating point modifier relax. If SIGMA is specified, r is assigned the reciprocal of the value of the floating point modifier sigma. (*Restrictions*: relax > ZERO and sigma > ZERO.) If neither RELAX nor SIGMA is specified, r is assigned the value 1.0.

We now define the vectors b, d, and f. In order to do this we need two auxiliary m-dimensional vectors n and a, whose ith components are defined as follows. Let np and NR be such that $i = np\times$NRAYS + nr+1 with $0 \le nr <$ NRAYS. Then n_i is the value of `numb` returned by either

```
ray(np, nr, list, weight, &numb, &snorm);
```

in the pixel case or

```
Blob.bray(np, nr, list, weight, &numb, &snorm);
```

in the blob case.

The value of a_i approximates the "area" of the nrth ray of the npth projection; it is the width of this ray multiplied by the length of intersection of the central line of the ray with the reconstruction region.

If GSIRT is specified, then

$$b_j = \sum_{i \in R_j} p_i / \sum_{i \in R_j} a_i,$$

$$d_j = \sum_{i \in R_j} n_i,$$

$$f_i = 1.$$

If LSIRT is specified, the exact algorithm depends on the value of the integer modifier type. *Restriction*: $1 \leq \text{type} \leq 3$.

If type = 1, then

$$b_j = \sum_{i \in R_j} (n_i p_i / a_i) / \sum_{i \in R_j} n_i,$$

$$d_j = \sum_{i \in R_j} n_i,$$

$$f_i = 1.$$

If type = 2, then

$$b_j = \sum_{i \in R_j} (p_i / a_i) / d_j,$$

$$d_j = \sum_{i \in R_j} 1,$$

$$f_i = 1/n_i.$$

(Note that f_i need not be defined unless $i \in R_j$, in which case $n_i \geq 1$).

If type = 3, then

$$b_j = \sum_{i \in R_j} (p_i / (n_i \times a_i)) / \sum_{i \in R_j} (1/n_i),$$

$$d_j = \sum_{i \in R_j} (1/n_i),$$

$$f_i = 1/n_i^2.$$

The choice of $x^{(0)}$ is determined by the keywords START or START BACKPROJECTED. If either of these options is chosen, then $x^{(0)}$ is set to the vector b. Otherwise the initialization option on the EXECUTE line (Section 5.8) is in effect.

One SNARK14iteration of the SIRT algorithm (see Section 6.1) consists of a single step. The contents of the `recon` array at the end of the iteration with iteration number k are described by $x^{(k)}$, unless the following conditions are met: NORMAL is specified, AREA$\times$|AVEDEN| > ZERO, and $|\sum_{j=1}^{n} x_j^{(k)}| >$ ZERO. If all three conditions are satisfied, then, prior to returning control to `exalg`, the SIRT routine multiplies each entry of `recon` by AVEDEN$/\sum_{j=1}^{n} x_j^{(k)}$. If another iteration is called, the initial value of $x_j^{(k)}$ for the new iteration will be the contents of `recon[`j`-1]` at end of the last iteration.

7.11 EMAP

> gamma [EVAL]

This is a general implementation of a maximum *a posteriori* probability (MAP) algorithm for PET based on a modified expectation-maximization (EM) algorithm [30, 67].

The MAP approach implemented by SNARK14 is the problem of finding an image vector x that maximizes the log-posterior functional (called the log-likelihood functional if $\gamma = 0$)

$$\ln P(x|y) = \sum_{i=1}^{I} \left[y_i \ln \left(\sum_{j=1}^{J} l_{ij} x_j \right) - \sum_{j=1}^{J} l_{ij} x_j \right] - \frac{\gamma}{2} x^T S x, \tag{7.3}$$

where x_j is the jth basis function of the image vector x $(1 \le j \le J)$, y_i is the ith component of the measurement vector y $(1 \le i \le I)$ and l_{ij} is the length of intersection of the ith measurement line with the jth basis function. *Restriction*: γ must be 0.0 in the blob case.

In view of this restriction the matrix S is defined only in the pixel case. It has the property that

$$x^T S x = \sum_{r \in N} \left(x_r - \frac{1}{8} \sum_{j \in N_r} x_j \right)^2,$$

where N is the set of indices r for that the rth basis function is not on the border of the picture region, N_r $(1 \le r \le N)$ is the set of indices associated with those at most eight basis functions that share a vertex with the rth basis function. If N is the empty set, the sum is 0. The entries of S are defined by

$$S_{uv} = \sum_{r \in N} s_{ru} s_{rv}, \text{ for } 1 \le u, v \le J,$$

where

$$s_{rj} = \begin{cases} 1, & \text{if } j = r, \\ -\frac{1}{8}, & \text{if } j \in N_r, \\ 0, & \text{otherwise.} \end{cases}$$

In case the floating-point modifier gamma in the follow-up command line is strictly greater than 0.0, its value is assigned to γ in (7.3). In this case, application of De Pierro's modified EM algorithm to the MAP problem [30] leads to the following iterative solution. Let $x^{(k)}$ be the image vector whose jth component, $x_j^{(k)}$, is `recon[`$j-1$`]` after the kth iteration of the algorithm. Then

$$x_j^{(k+1)} = \frac{1}{2} \left(-p_j^{(k)} + \sqrt{\left(p_j^{(k)} \right)^2 + 4 q_j^{(k)}} \right),$$

where

$$p_j^{(k)} = \frac{W_{ij}}{\gamma(9 S_{jj})} - x_j^{(k)} + \frac{1}{(9 S_{jj})} \sum_{u=1}^{J} S_{ju} x_u^{(k)},$$

$$q_j^{(k)} = \frac{x_j^{(k)}}{\gamma(9 S_{jj})} \sum_{i=1}^{J} \frac{l_{ij} y_i}{\sum_{n=1}^{J} l_{in} x_n^{(k)}},$$

$$W_{jj} = \sum_{i=1}^{I} l_{ij}.$$

If gamma = 0.0, the MAP EM algorithm reduces to the maximum likelihood (ML) EM algorithm ([67]) and the image vector is computed iteratively as follows (this option is available for blobs as well as for pixels):

$$x_j^{(k+1)} = \frac{x_j^{(k)}}{W_{jj}} \sum_{i=1}^{I} \frac{l_{ij} y_i}{\sum_{n=1}^{J} l_{in} x_n^{(k)}}.$$

If EVAL is specified, the log-likelihood functional and, if gamma > 0.0, the log-posterior functional are evaluated at each iteration and the results are written to a file named "MAPuser1".

7.12 LINO

$$> \quad \left\{ \begin{array}{l} \text{BANDLIMITING} \\ \text{SINC} \\ \text{COSINE} \end{array} \right\} \text{cutoff}$$

This is a general implementation of the linogram method [13, 14]. The reader is referred to [14] for a step-by-step description of the algorithm and its implementation.

The filter name and the floating point modifier cutoff define a filter function used to filter the projection data. (*Restriction*: $0 < \text{cutoff} \leq 1$).

Warning: Note that during collection of sinogram data the detector array is always perpendicular to the ray direction. On the other hand, during collection of linogram data the detector array lies fixed along the x-axis for projection angles $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$, and lies fixed along the y-axis for projection angles $\frac{\pi}{4} < \theta < \frac{3\pi}{4}$. The consequence of this is that to simulate linogram data collection with SNARK14 it is necessary to use the LINOGRAM geometry (Section 5.3.3, Set 6).

In SNARK14 the linogram program assumes the following relationship between the object size, the number of projections and the number of rays per projection. Let $2N+1$ be the (odd) number of pixels along a side of the digitized picture of the object. Then there are a total of $2(4N+3)$ projections with $4N+3$ rays per projection. Moreover, the $2(4N+3)$ projections are generated over two sets of angles as follows:

$$\begin{aligned} \theta \quad \in \quad & \left\{ \arctan\left(\frac{2i}{4N+3}\right) | -(2N+1) \leq i \leq (2N+1) \right\} \cup \\ & \left\{ \arctan\left(\frac{2i}{4N+3}\right) + \frac{\pi}{2} | -(2N+1) \leq i \leq (2N+1) \right\}. \end{aligned}$$

Thus for the first set of angles we have $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$ and for the second set of angles we have $\frac{\pi}{4} < \theta < \frac{3\pi}{4}$. These angles are generated by SNARK14 when the user specifies the LINOGRAM option of the GEOMETRY command (Section 5.3.3, Set 6). *Note*: The relationship between the angle THETA generated by SNARK14 (and defined in Figure 2.2) and the projection angle above is the following: $\text{THETA} = \theta + \frac{\pi}{2}$.

Chapter 8

SNARK EXPERIMENTER

8.1 Introduction

In medical imaging an enormous variety of algorithms have been proposed to reconstruct a cross-section of the human body. It is therefore often desirable to evaluate the relative efficacy of two or more reconstruction methods for a specific medical task in a manner that is statistically sound [8, 15, 40, 41, 43]. Such an evaluation, therefore, must be done using a sample set that is large enough to provide us with a statistically significant result [43].

Performing this evaluation on mathematical phantoms requires a means of running the competing algorithms on projection data obtained from a large number of randomly generated phantoms. Thereafter, various numerical measures of agreement between the reconstructed images and the original phantoms may be used to reach a conclusion that has some statistical substance [43]. A straightforward way of achieving this goal with SNARK14 is to provide a front-end or driver program that contains all the requisite commands that may be fed to SNARK14 to generate as many phantoms as needed (plus their projection data); to implement the desired reconstruction algorithms on these data; and to evaluate the reconstructed images. Such a driver program is provided by SNARK experimenter. The method used in the comparative evaluation of the algorithms consists of the following procedures [15]:

- generation of random samples from a statistically described ensemble of phantoms and their projection data (see Sections 8.2.2-8.2.4);
- reconstruction from the projection data by each of the algorithms to be compared (Section 8.2.5);
- assignment of a figure of merit (FOM) to each reconstructed image. The FOM should be a measure of image quality for solving the specified task (Section 8.2.6);
- calculation of the statistical significance, based on the FOMs for all reconstructions, at which we can reject the hypothesis that the methods are equally helpful for solving the task (Section 8.4).

SNARK experimenter has been designed to be very easy to use once a working knowledge of the SNARK14 commands is attained. This chapter describes the SNARK experimenter commands and provides several examples of the use of SNARK experimenter. The notation used here follows the convention described in Sections 4.3-4.5.

8.2 SNARK experimenter commands

SNARK experimenter reads input from UNIXs standard input or a file. This means that the input lines may be manually entered after the SNARK experimenter executable file is evoked, or they may be stored in a file. For example, if `file.in` is the file containing the SNARK experimenter commands, then SNARK experimenter is executed when the user enters the following line at the UNIX command prompt:

```
snark14 -e file.in
```

If file.in is missing, then SNARK14 experimenter will expect the INPUT to be entered as the standard input, which typically means entering the INPUT on the console line-by-line. If the SNARK14 experimenter makes use of user-defined algorithms and/or criteria, then the user-defined executable (i.e., snark14UserDefined) should be placed inside the current directory. In such a case, the experimenter should be invoked using the command:

```
snark14UserDefined -e file.in
```

(Note: Since SNARK experimenter drives SNARK14, it evokes the SNARK14 executable file. Warning: If the current directory contains the executable file snark14UserDefined, that version of SNARK14 will be used, otherwise, the version found in the PATH environment variable will be used.)

A single execution of SNARK experimenter requires an input sequence consisting of the following seven command lines (the first command is optional):

```
[SEED seed_value ]
ENSEMBLE file-specification
EXPERIMENT nexp exp-flag nelem pixel-size nave1 nphan_1 nrun_1 [nphan_2 nrun_2]
DATA file-specification
RECONSTRUCTION file-specification
ANALYSIS file-specification
END
```

A typical input file for a single execution of SNARK experimenter is:

```
SEED -1
ENSEMBLE b.9.virus.ens
EXPERIMENT 1 0 95 1.6 7 18 30
DATA b.9.projection.ss
RECONSTRUCTION b.9.recon.ss
ANALYSIS b.9.compare.ss
END
```

Multiple executions of SNARK experimenter may be achieved by repeating the first five command lines. The END command signals termination of the SNARK experimenter input stream. In the following sections we describe in detail the first five command lines.

8.2.1 SEED

> SEED seed_value

In this optional command line seed_value is the value used to seed the random number generation in SNARK experimenter. The random number generator is used in three places in SNARK experimenter: 1) to choose the phantom from the ensemble of phantoms (see 8.2.2), 2) to generate the seed used for generation of optional inhomogeneity in the phantom (see description of SCALE in 8.2.2), and 3) to generate the seed used for generation of optional noise in projection data (see description of MEASUREMENT in 8.2.4). If seed_value ≥ 0, that number is used to seed the random number generator (the entire experiment can be repeated with the same results, as long as the same random number generator is used). If seed_value < 0, the current time is used to seed the random number generator (each time the experiment is run, the results will be different).

If this line is not provided, the current time is used to seed the random number generator.

8.2.2 ENSEMBLE

> ENSEMBLE file-specification

In this command line file-specification is the name of an ensemble data file that contains a list of n phantom file names. The format of an ensemble data file is:

> phantom_file_1

> phantom_file_2
> ...
> phantom_file_n

In the example input file listed in Section 8.2 the file `b.9.virus.ens` is a typical ensemble data file that contains a list of 18 phantom file names as follows:

```
virus_24_1_0.96_0.6.atl
virus_24_1_0.96_0.7.atl
virus_24_1_0.96_0.8.atl
virus_24_1_0.97_0.6.atl
virus_24_1_0.97_0.7.atl
virus_24_1_0.97_0.8.atl
virus_26_1_0.96_0.6.atl
virus_26_1_0.96_0.7.atl
virus_26_1_0.96_0.8.atl
virus_26_1_0.97_0.6.atl
virus_26_1_0.97_0.7.atl
virus_26_1_0.97_0.8.atl
virus_28_1_0.96_0.6.atl
virus_28_1_0.96_0.7.atl
virus_28_1_0.96_0.8.atl
virus_28_1_0.97_0.6.atl
virus_28_1_0.97_0.7.atl
virus_28_1_0.97_0.8.atl
```

Each phantom file should start with one or two lines describing the spectrum of the x-ray beam in the manner specified by 5.3.3, Set 2. This is followed by a description of the phantom itself, it contains keywords and a series of follow-up lines that describe the elemental objects that make up the phantom:

> SINGLE
> [shape c_x c_y u v ang den(1)]
> [DENSITY den(2) . . . den(NERGY)]
> ...
> PAIR
> [shape c_x c_y u v ang den1(1) den2(1) prob]
> [DENSITY den1(2) . . . den1(NERGY)]
> [DENSITY den2(2) . . . den2(NERGY)]
> ...
> DUMMY
> [shape c_x c_y u v ang den(1)]
> [DENSITY den(2) . . . den(NERGY)]
> ...
> SCALE
> scale [sd]

Restriction: The lines starting with the keyword DENSITY are absent in the monochromatic case or if NERGY = 1.

The keywords SINGLE, PAIR and DUMMY may be followed by one or more follow-up lines that describe an elemental object, or structure, that makes up the phantom. In these follow-up lines shape is one of the following: ELIPSE, RECTANGLE, TRIANGLE, SEGMENT, or SECTOR (see Section 5.3.3, Set 3). c_x, c_y, u, v, ang, and den(1) ... den(NERGY) are all floating point modifiers that, together with shape, define the elemental object (see Section 5.3.3, Set 3).

For each follow-up line after the SINGLE keyword a single structure of type described by shape is generated.

For each follow-up line after the PAIR keyword a pair of structures of type described by shape is generated. One structure is generated at coordinate (c_x, c_y) and its mirror image is generated at coordinate $(-c_x, c_y)$

with a rotation of -ang. The floating-point modifier prob is the probability of randomly assigning densities den1(1) ... den1(NERGY) to the structure at location (c_x, c_y) and den2(1) ... den2(NERGY) to the structure at location $(-c_x, c_y)$. Equivalently, (1-prob) is the probability of randomly assigning densities den1(1) ... den1(NERGY) to the structure at location $(-c_x, c_y)$ and den2(1) ... den2(NERGY) to the structure at location (c_x, c_y). *Restriction*: den1(i) $\neq$ den2(i) for all i. The current time (using the Linux system call "time") is used as the seed of the random number generator. Multiple runs of SNARK experimenter with the same input files will generate different phantoms.

For each follow-up line after the DUMMY keyword a single structure of type described by shape is generated. However this structure is not used in calculating the figures of merit described in Section 8.2.6.

Note: In case a SINGLE, PAIR or DUMMY structure is not desired the corresponding follow-up line is left blank.

scale is a floating-point modifier that scales all previously defined densities. It may be optionally followed by local inhomogeneity specification using sd. These are identical to the scale and sd that follows the LAST keyword of the SNARK14 CREATE input sequence (see Section 5.3.3, Set 3). *Note*, that the seed value that follows the LAST keyword of the SNARK14 CREATE input sequence is generated automatically by SNARK experimenter; it should not be provided on the line following SCALE.

The contents of a typical phantom file are given below:

```
SPECTRUM MONOCHROMATIC 60
SINGLE
elip 0 0 29.72 29.72 0 0
PAIR
elip 4.14 63.19 3.89 3.89 0 1 0.97 0.5
elip 12.36 62.11 3.89 3.89 0 1 0.97 0.5
elip 20.36 59.97 3.89 3.89 0 1 0.97 0.5
elip 28.01 56.80 3.89 3.89 0 1 0.97 0.5
elip 35.18 52.66 3.89 3.89 0 1 0.97 0.5
elip 41.76 47.61 3.89 3.89 0 1 0.97 0.5
elip 47.61 41.76 3.89 3.89 0 1 0.97 0.5
elip 52.66 35.18 3.89 3.89 0 1 0.97 0.5
elip 56.80 28.01 3.89 3.89 0 1 0.97 0.5
elip 59.97 20.36 3.89 3.89 0 1 0.97 0.5
elip 62.11 12.36 3.89 3.89 0 1 0.97 0.5
elip 63.19 4.14 3.89 3.89 0 1 0.97 0.5
elip 63.19 -4.14 3.89 3.89 0 1 0.97 0.5
elip 62.11 -12.36 3.89 3.89 0 1 0.97 0.5
elip 59.97 -20.36 3.89 3.89 0 1 0.97 0.5
elip 56.80 -28.01 3.89 3.89 0 1 0.97 0.5
elip 52.66 -35.18 3.89 3.89 0 1 0.97 0.5
elip 47.61 -41.76 3.89 3.89 0 1 0.97 0.5
elip 41.76 -47.61 3.89 3.89 0 1 0.97 0.5
elip 35.18 -52.66 3.89 3.89 0 1 0.97 0.5
elip 28.01 -56.80 3.89 3.89 0 1 0.97 0.5
elip 20.36 -59.97 3.89 3.89 0 1 0.97 0.5
elip 12.36 -62.11 3.89 3.89 0 1 0.97 0.5
elip 4.14 -63.19 3.89 3.89 0 1 0.97 0.5
DUMMY
elip 0 0 59.44 59.44 0 0.8
SCALE
1
```

This input file may generate the following SNARK14 command lines that describe the objects to be generated:

```
OBJECTS
elip 0 0 29.72 29.72 0 0
```

```
elip 4.140000 63.189999 3.890000 3.890000 0.000000 1.000000
elip -4.140000 63.189999 3.890000 3.890000 -0.000000 0.970000
elip 12.360000 62.110001 3.890000 3.890000 0.000000 1.000000
elip -12.360000 62.110001 3.890000 3.890000 -0.000000 0.970000
elip 20.360001 59.970001 3.890000 3.890000 0.000000 0.970000
elip -20.360001 59.970001 3.890000 3.890000 -0.000000 1.000000
elip 28.010000 56.799999 3.890000 3.890000 0.000000 1.000000
elip -28.010000 56.799999 3.890000 3.890000 -0.000000 0.970000
elip 35.180000 52.660000 3.890000 3.890000 0.000000 1.000000
elip -35.180000 52.660000 3.890000 3.890000 -0.000000 0.970000
elip 41.759998 47.610001 3.890000 3.890000 0.000000 0.970000
elip -41.759998 47.610001 3.890000 3.890000 -0.000000 1.000000
elip 47.610001 41.759998 3.890000 3.890000 0.000000 0.970000
elip -47.610001 41.759998 3.890000 3.890000 -0.000000 1.000000
elip 52.660000 35.180000 3.890000 3.890000 0.000000 1.000000
elip -52.660000 35.180000 3.890000 3.890000 -0.000000 0.970000
elip 56.799999 28.010000 3.890000 3.890000 0.000000 0.970000
elip -56.799999 28.010000 3.890000 3.890000 -0.000000 1.000000
elip 59.970001 20.360001 3.890000 3.890000 0.000000 1.000000
elip -59.970001 20.360001 3.890000 3.890000 -0.000000 0.970000
elip 62.110001 12.360000 3.890000 3.890000 0.000000 0.970000
elip -62.110001 12.360000 3.890000 3.890000 -0.000000 1.000000
elip 63.189999 4.140000 3.890000 3.890000 0.000000 1.000000
elip -63.189999 4.140000 3.890000 3.890000 -0.000000 0.970000
elip 63.189999 -4.140000 3.890000 3.890000 0.000000 0.970000
elip -63.189999 -4.140000 3.890000 3.890000 -0.000000 1.000000
elip 62.110001 -12.360000 3.890000 3.890000 0.000000 1.000000
elip -62.110001 -12.360000 3.890000 3.890000 -0.000000 0.970000
elip 59.970001 -20.360001 3.890000 3.890000 0.000000 0.970000
elip -59.970001 -20.360001 3.890000 3.890000 -0.000000 1.000000
elip 56.799999 -28.010000 3.890000 3.890000 0.000000 0.970000
elip -56.799999 -28.010000 3.890000 3.890000 -0.000000 1.000000
elip 52.660000 -35.180000 3.890000 3.890000 0.000000 0.970000
elip -52.660000 -35.180000 3.890000 3.890000 -0.000000 1.000000
elip 47.610001 -41.759998 3.890000 3.890000 0.000000 0.970000
elip -47.610001 -41.759998 3.890000 3.890000 -0.000000 1.000000
elip 41.759998 -47.610001 3.890000 3.890000 0.000000 1.000000
elip -41.759998 -47.610001 3.890000 3.890000 -0.000000 0.970000
elip 35.180000 -52.660000 3.890000 3.890000 0.000000 0.970000
elip -35.180000 -52.660000 3.890000 3.890000 -0.000000 1.000000
elip 28.010000 -56.799999 3.890000 3.890000 0.000000 0.970000
elip -28.010000 -56.799999 3.890000 3.890000 -0.000000 1.000000
elip 20.360001 -59.970001 3.890000 3.890000 0.000000 1.000000
elip -20.360001 -59.970001 3.890000 3.890000 -0.000000 0.970000
elip 12.360000 -62.110001 3.890000 3.890000 0.000000 1.000000
elip -12.360000 -62.110001 3.890000 3.890000 -0.000000 0.970000
elip 4.140000 -63.189999 3.890000 3.890000 0.000000 0.970000
elip -4.140000 -63.189999 3.890000 3.890000 -0.000000 1.000000
elip 0 0 59.44 59.44 0 0.8
last 1
```

8.2.3 EXPERIMENT

> EXPERIMENT nexp exp-flag nelem pixel-size nave1 nphan-1 nrun-1 [nphan-2 nrun-2]

nexp is an integer modifier that equals the number of experiments to be performed. *Restriction*: $1 \leq$ nexp ≤ 2. When nexp $= 2$, the experiment described in the reconstruction phase (see Section 8.2.5) is performed twice.

exp-flag is an integer modifier used to control the phantom-file set used in each experiment as follows: if exp-flag $= 0$ the same set of phantom files is available for use in all experiments; if exp-flag $= 1$ then the phantom files used in the first experiment are excluded in the second experiment.

nelem, pixel-size, and nave1 are modifiers that are described by Set 4 of the SNARK14 CREATE input sequence (see Section 5.3.3).

nphan-i is an integer modifier that determines the number of phantom files used for experiment i.

nrun-i is an integer modifier that determines the number of runs for experiment i. *Note*: nphan-2 and nrun-2 are present only if nexp=2.

Restriction: nphan_i > 0; nrun_i > 0.

Note: The current time (using the Linux system call "time") is used as the seed for the random number generator to select which phantoms of the ensemble are used in an experiment and which structure of a pair has the higher density. Multiple runs of snark experimenter will use different sets of phantoms from the ensemble and generate different structure pairs.

8.2.4 DATA

> DATA file-specification

In this command line file-specification is the name of a file that contains information about generation of the projection data. The input lines of this file have the same syntax as Sets 5-8 of the SNARK14 CREATE input sequence (see Section 5.3.3 and Sections 5.4-5.5); i.e.

> RAYSUM (Section 5.3.3, Set 5)
> GEOMETRY (Section 5.3.3, Set 6)
> MEASUREMENT (Section 5.3.3, Set 7)
> RUN (Section 5.3.3, Set 8)
> PICTURE (Section 5.4)
> PROJECTION (Section 5.5)

The file `b.9.projection.ss` that follows the DATA command in Section 8.2 is a typical file containing information about the generation of the projection data as follows:

```
raysum average 1
1
geometry
divergent arc 153 306
rays user 101 detector spacing 3.2
angles 300 equally spaced
0 358.8
measurement noisy
quantum 0.0 0.0 calibr 4
multiplicative 1.0 0.1
seed
background 0.0
run
picture test
projection real
```

Note: If quantum noise is simulated (see Section 5.3.3, Set 7), the seed value is automatically generated by SNARK experimenter; it should not be provided on this line (it is ignored if provided).

8.2.5 RECONSTRUCTION

> RECONSTRUCTION file-specification

In this command line file-specification is the name of the file that contains information about the reconstruction process. The input lines of this file have the same syntax as discussed in Sections 5.6-5.8; i.e.

> [SELECT (Section 5.6)]
> [BASIS (Section 5.7)]
> [STOP (Section 5.9)]
> EXECUTE algorithm-1 (Section 5.8)
> key1: name-of-execution-1
> algorithm-1 parameters
...
> [SELECT (Section 5.6)]
> [BASIS (Section 5.7)]
> [STOP (Section 5.9)]
> EXECUTE algorithm-n (Section 5.8)
> key-n: name-of-execution-n
> algorithm-n parameters

The follow-up string after each EXECUTE command is a string as described in Section 5.8. As explained in that section, SNARK14 merely echos this string on all outputs produced during the analysis phase of a SNARK14 run. The first 4 characters of each follow-up string after an EXECUTE command should be unique since they are used as keywords by SNARK experimenter to identify the two reconstruction algorithms being compared during the SNARK experimenter analysis phase (see Section 8.2.6). The lines described by algorithm-i parameters refer to the input command line(s) required by algorithm-i for execution. The file `b.9.recon.ss` that follows the RECONSTRUCTION command in Section 8.2 is a typical file containing information about the reconstruction process as follows:

```
stop iteration 15
execute average emap
alg1: testing super snark with ML
gamma is 0.0
execute average emap
alg2: testing super snark with MAP
gamma is 10.0
```

Here we note that after, each EXECUTE command, the first 4 characters of the follow-up string form a unique keyword (`e.g., alg1` or `alg2`) that identify each reconstruction algorithm. Section 8.2.6 illustrates how these keywords are used in comparing two algorithms.

8.2.6 ANALYSIS

> ANALYSIS file-specification

In this command line file-specification is the name of the file containing information about the algorithms being compared. The format of this file is:

> fileout
> [MODE highden lowden]
> COMPARE key1 key2 FOM-1 [FOM-2 [FOM-3 [FOM-4]]]
> iteration-number-1 iteration-number-2
> iteration-number-3 iteration-number-4
...
> [MODE highden lowden]
> COMPARE keyn keym FOM-1 [FOM-2 [FOM-3 [FOM-4]]]
> iteration-number-1′ iteration-number-2′
> iteration-number-3′ iteration-number-4′
...
> END

Note that multiple comparisons may be achieved by repeating all but the first and last lines. The END command signals termination of this file.

fileout is the prefix of the file name that stores the significance results based on the FOMs discussed below. These results are stored in a text file whose name is of the form fileout.exp where exp is the experiment number (Section 8.2.3).

The MODE command line is an optional line whose function is discussed in Section 8.2.6.2.

In the first COMPARE command line key1 and key2 are unique keywords that identify the two algorithms being compared. These keys are the same as the first 4 characters of the follow-up strings discussed in Section 8.2.5. FOM-i is the FOM to be computed and may assume the following values: STRU, POIN, HITR, IROI, KLDS, USR1,USR2, USR3, USR4 and USR5. These FOMs are discussed below.

Following the COMPARE command line is a two-column list of iteration numbers to be compared. For example, the reconstructions from the algorithm identified by key-1 at iterations iteration-number-1, iteration-number-3, ... are compared to the reconstructions from the algorithm identified by key-2 at iterations iteration-number-2, iteration-number-4, respectively. *Note*: If an algorithm is noniterative then iteration-number-i should be set to 1. If iteration-number-i is set to 0, then the final iteration of that algorithm, whichever number that happens to be, will be compared.

In the example of Appendix A.9, the file `b.9.compare.ss` that follows the ANALYSIS command in Section 8.2 is a typical file containing the following instructions to compare the algorithms in the sample file `b.9.recon.ss` listed in Section 8.2.5:

```
testem
COMPARE alg1    alg2    HITR USR1
        1       1
        3       3
        6       6
        9       9
        12      12
        15      15
END
```

SNARK experimenter reads the MODE command lines and the list of iteration numbers from this file and combines these with the information contained in the files discussed in Sections 8.2.2-8.2.5 to generate a SNARK14 INPUT file (Chapter 3). SNARK experimenter then initiates a SNARK14 run with this file as input. The listing below illustrates a typical INPUT file generated by SNARK experimenter.

```
CREATE
virus_24_1_0.97_0.8.atl, seed used in phantom generation= 1126459725
SPECTRUM MONOCHROMATIC 511
OBJECTS
elip 0 0 29.72 29.72 0 0
elip 4.140000 63.189999 3.890000 3.890000 0.000000 1.000000
elip -4.140000 63.189999 3.890000 3.890000 -0.000000 0.970000
elip 12.360000 62.110001 3.890000 3.890000 0.000000 1.000000
elip -12.360000 62.110001 3.890000 3.890000 -0.000000 0.970000
elip 20.360001 59.970001 3.890000 3.890000 0.000000 0.970000
elip -20.360001 59.970001 3.890000 3.890000 -0.000000 1.000000
elip 28.010000 56.799999 3.890000 3.890000 0.000000 1.000000
elip -28.010000 56.799999 3.890000 3.890000 -0.000000 0.970000
elip 35.180000 52.660000 3.890000 3.890000 0.000000 1.000000
elip -35.180000 52.660000 3.890000 3.890000 -0.000000 0.970000
elip 41.759998 47.610001 3.890000 3.890000 0.000000 0.970000
elip -41.759998 47.610001 3.890000 3.890000 -0.000000 1.000000
elip 47.610001 41.759998 3.890000 3.890000 0.000000 0.970000
elip -47.610001 41.759998 3.890000 3.890000 -0.000000 1.000000
elip 52.660000 35.180000 3.890000 3.890000 0.000000 1.000000
```

```
elip -52.660000 35.180000 3.890000 3.890000 -0.000000 0.970000
elip 56.799999 28.010000 3.890000 3.890000 0.000000 0.970000
elip -56.799999 28.010000 3.890000 3.890000 -0.000000 1.000000
elip 59.970001 20.360001 3.890000 3.890000 0.000000 1.000000
elip -59.970001 20.360001 3.890000 3.890000 -0.000000 0.970000
elip 62.110001 12.360000 3.890000 3.890000 0.000000 0.970000
elip -62.110001 12.360000 3.890000 3.890000 -0.000000 1.000000
elip 63.189999 4.140000 3.890000 3.890000 0.000000 1.000000
elip -63.189999 4.140000 3.890000 3.890000 -0.000000 0.970000
elip 63.189999 -4.140000 3.890000 3.890000 0.000000 0.970000
elip -63.189999 -4.140000 3.890000 3.890000 -0.000000 1.000000
elip 62.110001 -12.360000 3.890000 3.890000 0.000000 1.000000
elip -62.110001 -12.360000 3.890000 3.890000 -0.000000 0.970000
elip 59.970001 -20.360001 3.890000 3.890000 0.000000 0.970000
elip -59.970001 -20.360001 3.890000 3.890000 -0.000000 1.000000
elip 56.799999 -28.010000 3.890000 3.890000 0.000000 0.970000
elip -56.799999 -28.010000 3.890000 3.890000 -0.000000 1.000000
elip 52.660000 -35.180000 3.890000 3.890000 0.000000 0.970000
elip -52.660000 -35.180000 3.890000 3.890000 -0.000000 1.000000
elip 47.610001 -41.759998 3.890000 3.890000 0.000000 0.970000
elip -47.610001 -41.759998 3.890000 3.890000 -0.000000 1.000000
elip 41.759998 -47.610001 3.890000 3.890000 0.000000 1.000000
elip -41.759998 -47.610001 3.890000 3.890000 -0.000000 0.970000
elip 35.180000 -52.660000 3.890000 3.890000 0.000000 0.970000
elip -35.180000 -52.660000 3.890000 3.890000 -0.000000 1.000000
elip 28.010000 -56.799999 3.890000 3.890000 0.000000 0.970000
elip -28.010000 -56.799999 3.890000 3.890000 -0.000000 1.000000
elip 20.360001 -59.970001 3.890000 3.890000 0.000000 1.000000
elip -20.360001 -59.970001 3.890000 3.890000 -0.000000 0.970000
elip 12.360000 -62.110001 3.890000 3.890000 0.000000 1.000000
elip -12.360000 -62.110001 3.890000 3.890000 -0.000000 0.970000
elip 4.140000 -63.189999 3.890000 3.890000 0.000000 0.970000
elip -4.140000 -63.189999 3.890000 3.890000 -0.000000 1.000000
elip 0 0 59.44 59.44 0 0.8
last 1
phantom average 7
95 1.600000
raysum average 1
1
geometry
divergent arc 153 306
rays user 101 detector spacing 3.2
angles 300 equally spaced
0 358.8
measurement noisy
quantum 0.0 0.0 calibr 4
multiplicative 1.0 0.1
seed 1126459725
background 0.0
run
picture test
projection real
stop iteration 15
execute average emap
```

```
alg1: testing super snark with ML experiment= 1 run= 1
gamma is 0.0
execute average emap
alg2: testing super snark with MAP experiment= 1 run= 1
gamma is 10.0
EVALUATE RESOLUTION
experimenter evaluation 1
SELECTIVE
elip 0.000000 0.000000 29.719999 29.719999 0.000000
elip 4.140000 63.189999 3.890000 3.890000 0.000000
elip -4.140000 63.189999 3.890000 3.890000 -0.000000
elip 12.360000 62.110001 3.890000 3.890000 0.000000
elip -12.360000 62.110001 3.890000 3.890000 -0.000000
elip 20.360001 59.970001 3.890000 3.890000 0.000000
elip -20.360001 59.970001 3.890000 3.890000 -0.000000
elip 28.010000 56.799999 3.890000 3.890000 0.000000
elip -28.010000 56.799999 3.890000 3.890000 -0.000000
elip 35.180000 52.660000 3.890000 3.890000 0.000000
elip -35.180000 52.660000 3.890000 3.890000 -0.000000
elip 41.759998 47.610001 3.890000 3.890000 0.000000
elip -41.759998 47.610001 3.890000 3.890000 -0.000000
elip 47.610001 41.759998 3.890000 3.890000 0.000000
elip -47.610001 41.759998 3.890000 3.890000 -0.000000
elip 52.660000 35.180000 3.890000 3.890000 0.000000
elip -52.660000 35.180000 3.890000 3.890000 -0.000000
elip 56.799999 28.010000 3.890000 3.890000 0.000000
elip -56.799999 28.010000 3.890000 3.890000 -0.000000
elip 59.970001 20.360001 3.890000 3.890000 0.000000
elip -59.970001 20.360001 3.890000 3.890000 -0.000000
elip 62.110001 12.360000 3.890000 3.890000 0.000000
elip -62.110001 12.360000 3.890000 3.890000 -0.000000
elip 63.189999 4.140000 3.890000 3.890000 0.000000
elip -63.189999 4.140000 3.890000 3.890000 -0.000000
elip 63.189999 -4.140000 3.890000 3.890000 0.000000
elip -63.189999 -4.140000 3.890000 3.890000 -0.000000
elip 62.110001 -12.360000 3.890000 3.890000 0.000000
elip -62.110001 -12.360000 3.890000 3.890000 -0.000000
elip 59.970001 -20.360001 3.890000 3.890000 0.000000
elip -59.970001 -20.360001 3.890000 3.890000 -0.000000
elip 56.799999 -28.010000 3.890000 3.890000 0.000000
elip -56.799999 -28.010000 3.890000 3.890000 -0.000000
elip 52.660000 -35.180000 3.890000 3.890000 0.000000
elip -52.660000 -35.180000 3.890000 3.890000 -0.000000
elip 47.610001 -41.759998 3.890000 3.890000 0.000000
elip -47.610001 -41.759998 3.890000 3.890000 -0.000000
elip 41.759998 -47.610001 3.890000 3.890000 0.000000
elip -41.759998 -47.610001 3.890000 3.890000 -0.000000
elip 35.180000 -52.660000 3.890000 3.890000 0.000000
elip -35.180000 -52.660000 3.890000 3.890000 -0.000000
elip 28.010000 -56.799999 3.890000 3.890000 0.000000
elip -28.010000 -56.799999 3.890000 3.890000 -0.000000
elip 20.360001 -59.970001 3.890000 3.890000 0.000000
elip -20.360001 -59.970001 3.890000 3.890000 -0.000000
elip 12.360000 -62.110001 3.890000 3.890000 0.000000
```

```
elip -12.360000 -62.110001 3.890000 3.890000 -0.000000
elip 4.140000 -63.189999 3.890000 3.890000 0.000000
elip -4.140000 -63.189999 3.890000 3.890000 -0.000000
LAST
11010010010010010000000000000000000000000000000000
EVALUATE RESOLUTION
experimenter evaluation 2
WHOLEPIC
11010010010010010000000000000000000000000000000000
END
```

Since this input file contains the SNARK14 EVALUATE command, several quantitative measures of agreement between the phantom and the reconstructions are written to the eval file (Section 5.11). At the end of the SNARK14 run SNARK experimenter accesses the eval file and uses the various quantitative measures stored there to compute the FOMs discussed in Sections 8.2.6.1-8.2.6.7. These FOMs are then stored in a text file whose name is of the form fomfil.exp where exp is the experiment number (Section 8.2.3). This file is then used in the calculation of statistical significance.

8.2.6.1 STRU

If STRU is specified the FOM *structural accuracy* [41] is calculated. This FOM is computed as follows.

Consider a phantom that contains a total of N structures. (Recall from Section 8.2.2 that the SINGLE command generates one structure for each elemental object and the PAIR command generates two structures for each elemental object.) For a reconstruction at iteration q, let $A_k^{(q)}$ be the average pixel value for those pixels whose centers are within structure k. Let $A_k^{(0)}$ be the average pixel value of the corresponding structure in the phantom.

We define the structural accuracy of a reconstruction at iteration q as

$$-\frac{1}{N}\sum_{k=1}^{N}|A_k^{(q)} - A_k^{(0)}|.$$

8.2.6.2 POIN

If POIN is specified the FOM *pointwise accuracy* [41] is calculated. The pointwise accuracy is defined as the negative of the normalized root mean square distance, δ^q (defined in Section 5.11) between a reconstruction and the phantom. It is sometimes desirable to compute the pointwise accuracy when both the phantom and the reconstruction are clipped to a specified density (for CT) or activity (for PET) range. If such clipping is desired then the COMPARE command line should be preceded by a command line of the form

```
>       MODE lowden highden
```

When this command is present, both the phantom and the reconstruction are clipped to a density (for CT) or activity (for PET) range specified by the lowden and highden values of the MODE command (see Section 5.2 for a description of the MODE command).

Notes:

- The clipping caused by the MODE command (if it is present) only affects the calculation of pointwise accuracy. The other figures of merit are not affected.
- If no clipping is desired in the calculation of pointwise accuracy then the MODE command is absent.
- If POIN is specified in more than one COMPARE command line and the MODE command appears at least once, then the clip values set by the most recent preceding MODE command are used in the calculation of the pointwise accuracy.

8.2.6.3 HITR

If HITR is specified the FOM *hit-ratio* [41] is computed. This FOM is calculated only for those phantoms containing structures generated by the PAIR command. (*Note*: such paired structures must have unequal densities as noted in Section 8.2.2.) These structures form symmetric pairs in a phantom. For such pairs a hit occurs if the structure in the pair with the higher average density in the phantom is also the structure in the pair with the higher average density in the reconstruction. The hit-ratio for a reconstruction is the number of hits divided by the total number of pairs.

8.2.6.4 IROI

If IROI is specified, the FOM *imagewise-region-of-interest* [59] is computed. This FOM is calculated only for phantoms that contain structures generated by the PAIR command. Such paired structures must have unequal densities with one of the structures having non-zero density (we refer to it as the *tumor*) and the other having density zero. (*Warning*: As usual, in case of a polychromatic spectrum, density here refers to the value of den(1).) The pairs of structures are numbered from 1 to B. For $1 \le b \le B$, let $\alpha_t^p(b)$ (respectively, $\alpha_n^p(b)$) denote the average density in the phantom of the structure of the bth pair that is (respectively, is not) the tumor. We specify similarly $\alpha_t^r(b)$ (respectively, $\alpha_n^r(b)$) for the reconstruction. The IROI figure of merit is defined by

$$\mathrm{IROI} = \frac{\sum_{b=1}^{B} (\alpha_t^r(b) - \alpha_n^r(b))}{\sqrt{\sum_{b=1}^{B} \left(\alpha_n^r(b) - \frac{1}{B} \sum_{b'=1}^{B} \alpha_n^r(b') \right)^2}} \Big/ \frac{\sum_{b=1}^{B} (\alpha_t^p(b) - \alpha_n^p(b))}{\sqrt{\sum_{b=1}^{B} \left(\alpha_n^p(b) - \frac{1}{B} \sum_{b'=1}^{B} \alpha_n^p(b') \right)^2}}.$$

If the reconstruction is perfect (in the sense of being identical to the phantom), then IROI=1.

8.2.6.5 KLDS

If KLDS is specified, then the *Kullback-Leibler distance* KL, which has been previously specified in Section 5.9 (5.17), is computed as the FOM.

8.2.6.6 WSQD

If WSQD is specified, then the *weighted squared distance* WS, which has been previously specified in Section 5.9 (5.18), is computed as the FOM.

8.2.6.7 USR1, USR2, USR3, USR4, USR5

If one of the five FOMs, USR1, USR2, USR3, USR4 or USR5, is specified, then a user-defined FOM is computed. To calculate this FOM, SNARK experimenter provides the user with the C subroutine, for example user_fom1, which exchanges information with the calling program via the parameters in the parameter list of the subroutine. The subroutine contains the following statements and may be modified as indicated below. (An example illustrating the use of a user-defined FOM is given in Appendix A, Section A.9.)

```
#include <malloc.h>
#include "experimenter.h"
#include "read_eval_phantom1.h"
#include "read_eval_recon3.h"
#include "user_fom1.h"

/* ----------------------------- user_fom_example.c --------------
        This function illustrates the use of a user-defined figure of merit.
INPUTS:
  itr1 - array of length niters containing the iteration numbers for the
```

```
          first algorithm.
  itr2 - array of length niters containing the iteration numbers for the
          second algorithm.
  niters - number of iterations to be compared
  keywrd1 - string containing the keyword that defines the first algorithm.
  keywrd2 - string containing the keyword that defines the second algorithm.
OUTPUTS:
  user1 - array of length niters containing the user defined figure of
          merit for the first algorithm.
  user2 - array of length niters containing the user defined figure of
          merit for the second algorithm.
*/

void user_fom1(int* itr1, int* itr2, int niters,
               char* keywrd1, char* keywrd2,
               double* user1, double* user2)
{
  double *phantom,*recon1,*recon2;
  int *regions,numstr,i,j,*strarea;
  float totarea=0.0,avgarea;

  /* Read in abnormality index and area for phantom structures.
     The program that does this is read_eval_phantom1.c.
     NOTE: the only information to be used is the value
     numstr (which is the number of structures in the phantom) */
  read_eval_phantom1(&regions,&numstr,&phantom,&strarea);

  /* allocate memory to store abnormality indexes and area
     (i.e number of pixels) for structures in the
     reconstructions */
  recon1 = (double *) malloc(numstr*sizeof(double));
  recon2 = (double *) malloc(numstr*sizeof(double));

  for(i=0;i < niters;i++) {
    /* read in variance for structures in
       the reconstructions. Initialize the arrays
       that will store the user FOM */

    read_eval_recon3(itr1[i],keywrd1,numstr,recon1);
    read_eval_recon3(itr2[i],keywrd2,numstr,recon2);
    /* the variance of the SINGLE structure to be
       used is the first in the array.  The FOM value
       is 1-variance */
    user1[i]=1-recon1[0];
    user2[i]=1-recon2[0];
  }

  free(phantom);        /* free memory for next function call */
  free(recon1);
  free(recon2);
  free(strarea);
}
```

8.3 Running in DEBUG mode

It is often useful to be able to look at individual runs of SNARK14 that are part of an experiment. SNARK experimenter saves the SNARK14 input and output files from the very last run in files snark_e.in and snark_e.out respectively. If this is not enough, and the record of input and output files from all the runs is desired, the user can run SNARK experimenter in the DEBUG mode. In order to do this, the environment variable SNARK_EXP should be set to DEBUG, before SNARK experimenter is run. To do so, type on the Linux/Unix command line:

```
export SNARK_EXP=DEBUG
```

(Note, there should be no spaces around equal sign.) and then run SNARK experimenter as usual.

The input files will be saved with names snark_e1.in ... snark_eN.in, the corresponding output files will be saved with names snark_e1.out ... snark_eN.out, and recfil files from each run will be saved with names recfil1 ... recfilN.

Warning: The recfil files may be very large and consume a lot of space on your system.

To disable this mode, type on the Linux/Unix command line:

```
export SNARK_EXP=
```

(nothing follows the equal sign).

8.4 Statistical significance level at which the null hypothesis can be rejected

In Section 8.1 the final step in the four-step evaluation procedure is the calculation of statistical significance. The FOMs defined in Section 8.2.6 indicate how good a particular reconstruction is for a given task. An algorithm with a higher average FOM is judged to be more appropriate for the given task than the one with a lower average FOM [43]. If we find, based on this FOM, that one method is more appropriate than another for the task at hand, we still need to determine if our finding is statistically significant. To determine this, we assign a level of statistical significance to rejecting the null-hypothesis that two reconstruction methods are equally good (from the point of view of the FOM), in favor of the hypothesis that the one with the higher average FOM is better, as follows [9, 15].

Let $\psi_m^{(q)}$ and $\phi_m^{(r)}$ be the FOMs of the reconstructions from the mth of altogether M data sets at iterations q and r, as reconstructed by the two methods, respectively. Then, according to the null-hypothesis, $\psi_m^{(q)} - \phi_m^{(r)}$ is a sample of a zero-mean random variable. It follows, for large enough M ($M \geq 30$), that

$$\sum_{m=1}^{M} (\psi_m^{(q)} - \phi_m^{(r)})$$

is a sample from a normally distributed random variable with zero mean and variance [41]

$$\sum_{m=1}^{M} (\psi_m^{(q)} - \phi_m^{(r)})^2.$$

We can thus use the normal distribution to calculate significance [58].

Listed below is the output file generated by SNARK experimenter that contains the computed significance levels based on the analysis file listed in Section 8.2.6.

Appendix A

EXAMPLES

A.1 Daisy pattern

In this example a test phantom consisting of ellipses, segments and a triangle is generated. Divergent projection data are simulated for 60 projections and the data are reconstructed using DCONV. In addition to the analysis output file both punch and recfil files are produced. From the recfil, images of phantom and reconstruction are produced by the display program snark14Display (see Figure A.1). It is also possible to produce these images from the file punch.

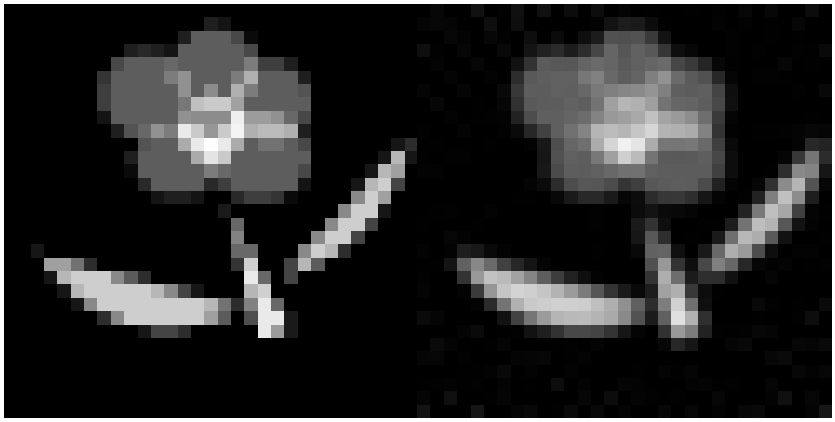

Figure A.1: Half-tone displays from Example A.1.

A.1.1 SNARK14 INPUT file

```
* EXAMPLE 1
* DAISY PATTERN
* CREATION OF PHANTOM AND DIVERGENT PROJECTION DATA
* RECONSTRUCTION WITH DCON
* ANALYSIS
CREATE
EXAMPLE 1 DAISY
SPECTRUM MONOCHROMATIC 75
OBJECTS
TRIA  1.9  -3.7  0.5  4.1  20.0  2.5
SEGM  4.0   0.1  2.9  4.3  47.0  2.2
SEGM -2.1  -2.1  3.1  3.0 -15.0  2.2
ELIP  0.0   2.5  1.0  1.0   0.0  1.5
```

```
ELIP  0.1   4.3  1.3  1.2   0.0  1.0
ELIP -2.0   3.5  1.3  1.3   0.0  1.0
ELIP -1.0   1.5  1.3  1.1   0.0  1.0
ELIP  1.5   1.6  1.5  1.3   0.0  1.0
ELIP  1.8   3.3  1.2  1.2   0.0  1.0
LAST 1.0
PHANTOM AVERAGE 3
31 0.4
RAYSUM AVERAGE 1
1 1
GEOMETRY
DIVERGENT ARC 20.0 10000.0
RAYS USER 65 150.0
ANGLES 60 EQUALLY SPACED
0.0 352.5
MEASUREMENT PERFECT
BACKGROUND 0.0
RUN
*
PICTURE TEST
*
PROJECTION REAL
*
EXECUTE DCON
EXAMPLE 1 DIVERGENT CONVOLUTION FROM 60 VIEWS
-1 1 -1 0.5 0
BANDLIMITING
*
LINES SCALE 100.0 COLUMN 16
1
*
PUNCH PHANTOM
1
*
END
```

A.1.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


     <*>  EXAMPLE 1

     <*>  DAISY PATTERN

     <*>  CREATION OF PHANTOM AND DIVERGENT PROJECTION DATA

     <*>  RECONSTRUCTION WITH DCON

     <*>  ANALYSIS

     <#> CREATE
```

```
        EXAMPLE 1 DAISY


    <#> SPECTRUM MONOCHROMATIC 75
        energy spectrum is monochromatic at energy level    75


    <#> OBJECTS
        description of objects
                                                                        density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 tria   1.9000  -3.7000   0.5000   4.1000  20.0000   2.5000    2.5000

   2 segm   4.0000   0.1000   2.9000   4.3000  47.0000   2.2000    2.2000

   3 segm  -2.1000  -2.1000   3.1000   3.0000 -15.0000   2.2000    2.2000

   4 elip   0.0000   2.5000   1.0000   1.0000   0.0000   1.5000    1.5000

   5 elip   0.1000   4.3000   1.3000   1.2000   0.0000   1.0000    1.0000

   6 elip  -2.0000   3.5000   1.3000   1.3000   0.0000   1.0000    1.0000

   7 elip  -1.0000   1.5000   1.3000   1.1000   0.0000   1.0000    1.0000

   8 elip   1.5000   1.6000   1.5000   1.3000   0.0000   1.0000    1.0000

   9 elip   1.8000   3.3000   1.2000   1.2000   0.0000   1.0000    1.0000

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000

    <#> PHANTOM AVERAGE 3

        this run will generate a phantom
        density in each pixel is obtained as the average of 3 x 3 points


    <#> 31 0.4
        picture size 31 x 31,  pixel size     0.4000


    <#> RAYSUM AVERAGE 1

        this run will generate projection data
        projection data are calculated by dividing each ray interval into 1 substrips

        with aperture (substrip) weights     1


    <#> GEOMETRY
```

```
<#> DIVERGENT ARC 20.0 10000.0
    rays are divergent from point sources
    source to origin distance     20.0000
    the detectors lie on an arc with source to detector distance = 10000.0000


<#> RAYS USER 65 150.0
    number of rays per projection    65
    at detector spacing   150.0000


<#> ANGLES 60 EQUALLY SPACED
    total number of projections    60

    projection angles    0.0000    5.9746   11.9492   17.9237   23.8983   29.8729   35.8475
                        59.7458   65.7203   71.6949   77.6695   83.6441   89.6186   95.5932  1
                       119.4915  125.4661  131.4407  137.4153  143.3898  149.3644  155.3390  1
                       179.2373  185.2119  191.1864  197.1610  203.1356  209.1102  215.0847  2
                       238.9831  244.9576  250.9322  256.9068  262.8814  268.8559  274.8305  2
                       298.7288  304.7034  310.6780  316.6525  322.6271  328.6017  334.5763  3


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND 0.0
                           at levels
                                  75
     background absorption   0.0000


<#> RUN
         0.001 seconds phantom creation
         0.004 seconds projection data creation
         0.005 seconds used for processing command crea


<*>

<#> PICTURE TEST

    EXAMPLE 1 DAISY


<#> spec    mono   75
    energy spectrum is monochromatic at energy level    75


<#> obje
    description of objects
                                                          density at levels
```

```
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 tria   1.9000  -3.7000   0.5000   4.1000  20.0000   2.5000    2.5000

   2 segm   4.0000   0.1000   2.9000   4.3000  47.0000   2.2000    2.2000

   3 segm  -2.1000  -2.1000   3.1000   3.0000 -15.0000   2.2000    2.2000

   4 elip   0.0000   2.5000   1.0000   1.0000   0.0000   1.5000    1.5000

   5 elip   0.1000   4.3000   1.3000   1.2000   0.0000   1.0000    1.0000

   6 elip  -2.0000   3.5000   1.3000   1.3000   0.0000   1.0000    1.0000

   7 elip  -1.0000   1.5000   1.3000   1.1000   0.0000   1.0000    1.0000

   8 elip   1.5000   1.6000   1.5000   1.3000   0.0000   1.0000    1.0000

   9 elip   1.8000   3.3000   1.2000   1.2000   0.0000   1.0000    1.0000

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000

    <#> phan    aver    3

        density in each pixel is obtained as the average of 3 x 3 points


    <#> pixe       31    size        0.4000
        picture size 31 x 31,  pixel size     0.4000

        test picture read
        EXAMPLE 1 DAISY
             0.001 seconds used for processing command pict


    <*>

    <#> PROJECTION REAL

        EXAMPLE 1 DAISY


    <#> spec    mono   75
        energy spectrum is monochromatic at energy level    75


    <#> obje
        description of objects
                                                                 density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75
```

```
1 tria   1.9000  -3.7000   0.5000   4.1000  20.0000   2.5000    2.5000

2 segm   4.0000   0.1000   2.9000   4.3000  47.0000   2.2000    2.2000

3 segm  -2.1000  -2.1000   3.1000   3.0000 -15.0000   2.2000    2.2000

4 elip   0.0000   2.5000   1.0000   1.0000   0.0000   1.5000    1.5000

5 elip   0.1000   4.3000   1.3000   1.2000   0.0000   1.0000    1.0000

6 elip  -2.0000   3.5000   1.3000   1.3000   0.0000   1.0000    1.0000

7 elip  -1.0000   1.5000   1.3000   1.1000   0.0000   1.0000    1.0000

8 elip   1.5000   1.6000   1.5000   1.3000   0.0000   1.0000    1.0000

9 elip   1.8000   3.3000   1.2000   1.2000   0.0000   1.0000    1.0000

     scale factor multiplying object densities     1.0000

     seed set to 0
     inhomogeneity set to     0.0000

 <#> rays    aver    1

     projection data are calculated by dividing each ray interval into 1 substrips

     with aperture (substrip) weights     1


 <#> geom


 <#> dive    arc     source at   20.0000     det dist10000.0000
     rays are divergent from point sources
     source to origin distance     20.0000
     the detectors lie on an arc with source to detector distance = 10000.0000


 <#> rays    user       65    spacing      150.0000
     number of rays per projection    65
     snark computed number of rays    63
     at detector spacing   150.0000


 <#> angl       60
     total number of projections    60


     projection angles    0.0000    5.9746   11.9492   17.9237   23.8983   29.8729   35.8475
                         59.7458   65.7203   71.6949   77.6695   83.6441   89.6186   95.5932  1
                        119.4915  125.4661  131.4407  137.4153  143.3898  149.3644  155.3390  1
                        179.2373  185.2119  191.1864  197.1610  203.1356  209.1102  215.0847  2
                        238.9831  244.9576  250.9322  256.9068  262.8814  268.8559  274.8305  2
```

```
                        298.7288  304.7034  310.6780  316.6525  322.6271  328.6017  334.5763  3

<#> meas    perf
    projection data are noiseless


<#> back      0.0000
                           at levels
                                  75
     background absorption   0.0000

    estimate of totlen =     31273.639204
    estimate of totden =     11129.885164
    estimate of average density =     0.3559
    projection data read
    EXAMPLE 1 DAISY
         0.003 seconds used for processing command proj


<*>

<#> EXECUTE DCON

    EXAMPLE 1 DIVERGENT CONVOLUTION FROM 60 VIEWS

<#> -1 1 -1 0.5 0
     cubic spline interpolation
     one out of every     1 projection(s) used for reconstruction
      64 points used on each side of convolution
     pre-smoothing weight on center ray = 0.5000
     pre-smoothing weight on side rays  = 0.2500
     accurate back-projection

<#> BANDLIMITING
     band-limiting filter
     total time for obtaining smoothed projection data was     0.000
     total time for convoluting projection data was            0.000
     total time for back-projecting was                        0.005
    algorithm executed in iteration    1
         0.006 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.006 seconds for this iteration
         0.010 seconds used for processing command exec


<*>

<#> LINES SCALE 100.0 COLUMN 16

    last iteration

lines display of reconstruction using DCON iter 1
```

```
    phantom name:    EXAMPLE 1 DAISY
    projection data: EXAMPLE 1 DAISY
    execution name:  EXAMPLE 1 DIVERGENT CONVOLUTION FROM 60 VIEWS
    scaling factor =  100.00000

column =              16
row    original   estimate difference
  0     0.00000   -7.80005   -7.80005
  1    22.22222   28.11589    5.89367
  2   100.00000   88.37252  -11.62748
  3   100.00000  104.43894    4.43894
  4   100.00000  102.69494    2.69494
  5   100.00000  107.73658    7.73658
  6   100.00000  147.49863   47.49863
  7   216.66667  188.04080  -28.62586
  8   150.00000  180.51891   30.51891
  9   216.66667  211.83020   -4.83647
 10   250.00000  245.52265   -4.47735
 11   216.66667  199.38366  -17.28301
 12   111.11111  120.98082    9.86971
 13    66.66667   68.05861    1.39194
 14     0.00000   13.57312   13.57312
 15    27.77778   12.48964  -15.28813
 16     0.00000   27.99191   27.99191
 17     0.00000   24.69207   24.69207
 18     0.00000   10.47314   10.47314
 19     0.00000   -0.41472   -0.41472
 20     0.00000    0.60572    0.60572
 21     0.00000   10.05671   10.05671
 22   122.22222  102.66325  -19.55897
 23    97.77778   87.82826   -9.94952
 24     0.00000    5.13094    5.13094
 25     0.00000   -2.85855   -2.85855
 26     0.00000   -3.34366   -3.34366
 27     0.00000    1.38876    1.38876
 28     0.00000    3.44155    3.44155
 29     0.00000    5.01602    5.01602
 30     0.00000   -0.04334   -0.04334


            0.001 seconds used for processing command line


    <*>

    <#> PUNCH PHANTOM

        last iteration

         reconstruction of EXAMPLE 1 DIVERGENT CONVOLUTIO using DCON iter     1,   EXAMPLE 1 DAISY
            0.002 seconds used for processing command punc
```

```
<*>

<#> END
```

A.2 Simple pattern and user subroutines

In this example we illustrate user written subroutines and multiple executions in the same run. A user written function `tset` is provided for an ART reconstruction with variable tolerance. A user written algorithm `alp1` is also provided. This algorithm incorporates an extrapolation feature into an ART-type reconstruction procedure. Parallel projection data are simulated for 30 projections and the data are reconstructed by both algorithms mentioned above. Analysis output is produced using EVALUATE and LINES.

A.2.1 User written tset

```
/*
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                             *
 *                              S N A R K   1 4                                *
 *                                                                             *
 *                     A PICTURE RECONSTRUCTION PROGRAM                        *
 *                                                                             *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 art_tset.cpp,v 1.2 2008/09/06 15:18:05 rdavidi Exp

*/

//    Example 2  user-written tolerance function
//    This function returns RAYSUM/(10*ITER) as tolerance.

#include <stdio.h>

#include "blkdta.h"
#include "uiod.h"

#include "art.h"

REAL art_class::tset(REAL* recon, INTEGER iter, INTEGER np, INTEGER nr,INTEGER i, REAL raysum, INTE
{
  REAL tmp = raysum / ((REAL)(10 * iter));
  *flag = FALSE;
  if(trace >= 1) {
    fprintf(output, "\n          TOLERANCE IS %19.10f \n", tmp);
  }
  return tmp;
}
```

A.2.2 User written alp1

```
/*
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                             *
```

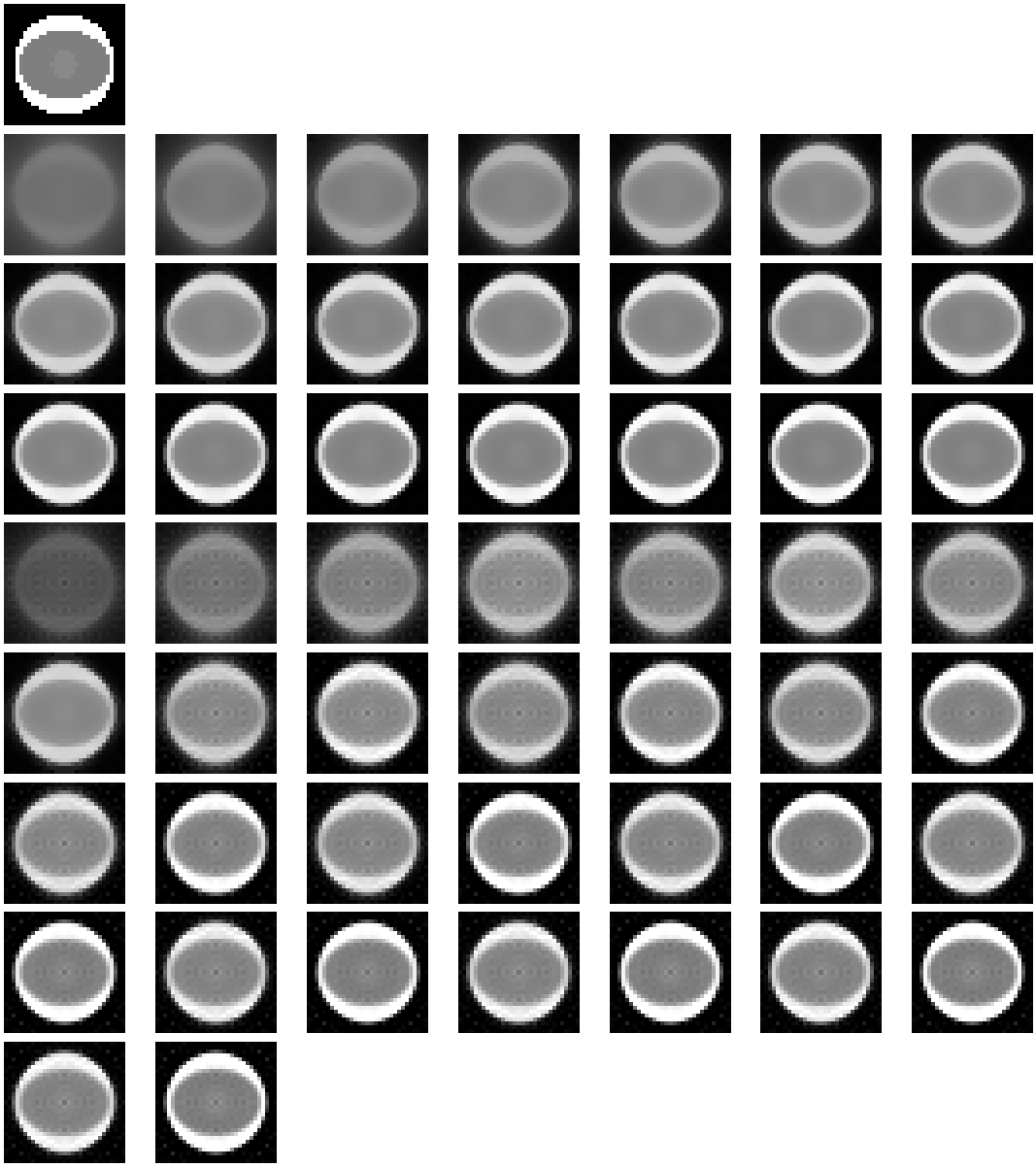

Figure A.2: Half-tone displays from Example A.2. The phantom is on top, the next three rows show the ART reconstruction at each iteration, and the last five rows show the reconstruction with the user defined algorithm at each iteration.

```
 *                              S N A R K   1 4                              *
 *                                                                           *
 *                      A PICTURE RECONSTRUCTION PROGRAM                     *
 *                                                                           *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

 alp1.cpp,v 1.1 2009/06/01 23:26:59 jklukowska Exp

*/
//     Example 2  user-written reconstruction function


#include <stdio.h>
#include <math.h>

#include "blkdta.h"
#include "pseudo.h"
#include "projfile.h"
#include "uiod.h"
#include "consts.h"

#include "alp1.h"

void extrap(REAL* newrec, REAL* oldrec, REAL c, INTEGER area);
void itrun(REAL* recon, REAL* oldrec, REAL* proj, INTEGER* list, REAL* weight);

BOOLEAN alp1_class::Run(REAL* recon, INTEGER* list, REAL* weight, INTEGER iter)
{

  static REAL* oldrec;
  static INTEGER curprj;
  static REAL* proj;
  INTEGER i;
  static REAL newdif;
  static REAL olddif;
  static REAL c;

  if(iter == 1) {

    oldrec = new REAL[GeoPar.area];

    proj = new REAL[GeoPar.nrays];

    for(i = 0; i < GeoPar.area; i++) {
      oldrec[i] = 0;
    }

    // The previous art iteration is in OLDREC
    itrun(recon, oldrec, proj, list, weight);

    for(i = 0; i < GeoPar.area; i++) {
      oldrec[i] = recon[i];
    }
```

```
      curprj = 1;
      return FALSE;
    }
    else if(iter == 2) {
      itrun(recon, oldrec, proj, list, weight);
      newdif = 0.0;
      curprj = 2;

      for(i = 0; i < GeoPar.area; i++) {
        newdif += SQR(recon[i] - oldrec[i]);
        oldrec[i] = recon[i];
      }

      newdif = (REAL) sqrt(newdif);
      return FALSE;
    }
    else if((iter/2)*2 != iter) {
      curprj += 1;
      itrun(recon, oldrec, proj, list, weight);
      olddif = newdif;
      newdif = 0.0;

      for(i = 0; i < GeoPar.area; i++) {
        newdif += SQR(recon[i] - oldrec[i]);
      }

      newdif = (REAL) sqrt(newdif);
      if(olddif <= Consts.zero * newdif) {
        fprintf(output, "\nCONVERGENCE DIFFERENCES %20.12e, %20.12e", olddif, newdif);
        return FALSE;
      }
      c = newdif/olddif;
      fprintf(output, "\n EXTRAPOLATION PARAMETER IN EXTRAP IS %20.12e", c);

      return FALSE;
    }

    extrap(recon, oldrec, c, GeoPar.area);

    return FALSE;
  }

  // NEWREC will contain extrapolated values
  // OLDREC will contain the old values of NEWREC

  void extrap(REAL* newrec, REAL* oldrec, REAL c, INTEGER area)
  {
    REAL d, save;
    INTEGER i;

    if(fabs(1.0 - c) > Consts.zero) {
      fprintf(output, "\n EXTRAPOLATION PARAMETER IN EXTRAP IS %20.12e", c);
      d = c / ((REAL) 1.0 - c);
```

```
    for(i = 0; i < area; i++) {
      save = newrec[i];
      newrec[i] += d * (newrec[i] - oldrec[i]);
      oldrec[i] = save;
    }
    return;
  }
  fprintf(output, "\n EXTRAPOLATION PARAMETER IN EXTRAP IS %20.12e", c);

  fprintf(output, "\n EXTRAPOLATION PARAMETER TOO CLOSE TO 1 ");


  for(i = 0; i < area; i++) {
    oldrec[i] = newrec[i];
  }
  return;
}

void itrun(REAL* recon, REAL* oldrec, REAL* proj, INTEGER* list, REAL* weight)
{
  INTEGER i, nr, np, numb, j;
  REAL snorm, raysum;

  for(i = 0; i < GeoPar.area; i++) {
    recon[i] = oldrec[i];
  }

  for(np = 0; np < GeoPar.prjnum; np++) {

    ProjFile.ReadProj(np, proj, GeoPar.nrays);

    for(nr = 0; nr < GeoPar.nrays; nr++) {
      raysum = pseudo(recon, np, nr, list, weight, &numb, &snorm, TRUE, FALSE);

      for(i = 0; i < numb; i++) {
        j = list[i];
        recon[j] += (proj[nr]-raysum)* weight[i] * (REAL) 0.025 / snorm;
      }
    }
  }

  return;
}
```

A.2.3 SNARK14 INPUT file

```
**** EXAMPLE 2
*    Creation of a small test pattern and projection data with execution of
*    a user-defined reconstruction algorithm and analysis.
*
CREATE
Example 2,  A simple test pattern and projection data
SPECTRUM MONOCHROMATIC with energy at 75 KeV
OBJECTS   cx    cy    u     v     ang   den
```

```
ELIP      0.00  0.00  0.90  0.90  0.00  2.50
ELIP      0.00  0.00  0.80  0.60  0.00 -1.25
ELIP      0.00  0.00  0.21  0.25  0.00  0.10
LAST  1.00
PHANTOM AVERAGE 1 point for estimating pixel density
number of elements is 31  Size of a pixel is 0.07
RAYSUM AVERAGE 1 line for estimation of ray sum with weight
    1
GEOMETRY
PARALLEL UNIFORM LINE
RAYS USER-defined 63 rays per projection at spacing 0.05
ANGLES 30  EQUAL SPACING between the first and last angles which are
    0.0    174.0
MEASUREMENT PERFECT
BACKGROUND absorption is 0.0
RUN
*
PICTURE TEST
*
PROJECTION REAL
*
MODE LOWER constraint set at 0.0  UPPER constraint set at 2.5
*
SELECT USER RAYSEQ
STEP projections by 10 and rays by 1
*
STOP TERMINATION VARIANCE at 0.01
*
EXECUTE AVERAGE ART SMOOTH
Example 2a  underrelaxed art4 with variable tolerance
smoothing threshold is 0.2  Weights are 2.25 1.5 1.0
1111111111111111111111111111111111111111111111111
ART4  RELAXATION CONSTANT = 0.025  TOLERANCE VARIABLE decreasing
CONSTRAINT BOUND CONRELAX CONSTANT = 0.5 NOMLIZE at each iteration
*
STOP ITERATIONS = 30
*
EXECUTE ALP1
Example 2b  underrelaxed art with extrapolation
*
* Reset mode to unconstrained
MODE
*
EVALUATE RESOLUTION
Example 2a  underrelaxed art4
WHOLEPIC
2222222222222222222222222222222222222222
*
LINES COLUMNS 15 and 18 are to be printed
1         1         1         1
*
END
```

A.2.4 SNARK14 OUTPUT file

```
snark14UserDefined.s170710 - A PICTURE RECONSTRUCTION PROGRAM


  <*> *** EXAMPLE 2

  <*>     Creation of a small test pattern and projection data with execution of

  <*>     a user-defined reconstruction algorithm and analysis.

  <*>

  <#> CREATE

      Example 2,  A simple test pattern and projection data


  <#> SPECTRUM MONOCHROMATIC with energy at 75 KeV
      energy spectrum is monochromatic at energy level    75


  <#> OBJECTS   cx    cy    u     v     ang   den
      description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 elip   0.0000   0.0000   0.9000   0.9000   0.0000   2.5000    2.5000

   2 elip   0.0000   0.0000   0.8000   0.6000   0.0000  -1.2500   -1.2500

   3 elip   0.0000   0.0000   0.2100   0.2500   0.0000   0.1000    0.1000

      scale factor multiplying object densities     1.0000

      seed set to 0
      inhomogeneity set to     0.0000

  <#> PHANTOM AVERAGE 1 point for estimating pixel density

      this run will generate a phantom
      density in each pixel is obtained as the average of 1 x 1 points


  <#> number of elements is 31  Size of a pixel is 0.07
      picture size 31 x 31,  pixel size     0.0700


  <#> RAYSUM AVERAGE 1 line for estimation of ray sum with weight

      this run will generate projection data
      projection data are calculated by dividing each ray interval into 1 substrips
```

```
    with aperture (substrip) weights     1


<#> GEOMETRY


<#> PARALLEL UNIFORM LINE
    rays are parallel with uniform spacing between rays
    data collected along lines



<#> RAYS USER-defined 63 rays per projection at spacing 0.05
    number of rays per projection    63
    at detector spacing     0.0500


<#> ANGLES 30  EQUAL SPACING between the first and last angles which are
    total number of projections    30

    projection angles    0.0000    6.0000   12.0000   18.0000   24.0000   30.0000   36.0000
                        60.0000   66.0000   72.0000   78.0000   84.0000   90.0000   96.0000  1
                       120.0000  126.0000  132.0000  138.0000  144.0000  150.0000  156.0000  1


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND absorption is 0.0
                           at levels
                                  75
     background absorption   0.0000


<#> RUN
         0.001 seconds phantom creation
         0.001 seconds projection data creation
         0.002 seconds used for processing command crea


<*>

<#> PICTURE TEST

    Example 2,  A simple test pattern and projection data


<#> spec    mono   75
    energy spectrum is monochromatic at energy level    75


<#> obje
    description of objects
```

```
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 elip   0.0000   0.0000   0.9000   0.9000   0.0000   2.5000    2.5000

   2 elip   0.0000   0.0000   0.8000   0.6000   0.0000  -1.2500   -1.2500

   3 elip   0.0000   0.0000   0.2100   0.2500   0.0000   0.1000    0.1000

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000

    <#> phan    aver    1

        density in each pixel is obtained as the average of 1 x 1 points


    <#> pixe       31    size        0.0700
        picture size 31 x 31,  pixel size     0.0700

        test picture read
        Example 2,  A simple test pattern and projection data
             0.000 seconds used for processing command pict


    <*>

    <#> PROJECTION REAL

        Example 2,  A simple test pattern and projection data


    <#> spec    mono   75
        energy spectrum is monochromatic at energy level    75


    <#> obje
        description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 elip   0.0000   0.0000   0.9000   0.9000   0.0000   2.5000    2.5000

   2 elip   0.0000   0.0000   0.8000   0.6000   0.0000  -1.2500   -1.2500

   3 elip   0.0000   0.0000   0.2100   0.2500   0.0000   0.1000    0.1000

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000
```

```
<#> rays    aver    1

    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> geom


<#> para unif line
    rays are parallel with uniform spacing between rays
    data collected along lines



<#> rays    user       63    spacing        0.0500
    number of rays per projection    63
    snark computed number of rays    63
    at detector spacing     0.0500


<#> angl       30
    total number of projections    30


    projection angles    0.0000    6.0000   12.0000   18.0000   24.0000   30.0000   36.0000
                        60.0000   66.0000   72.0000   78.0000   84.0000   90.0000   96.0000  1
                       120.0000  126.0000  132.0000  138.0000  144.0000  150.0000  156.0000  1


<#> meas    perf
    projection data are noiseless


<#> back      0.0000
                           at levels
                                 75
     background absorption   0.0000

    estimate of totlen =      2823.757863
    estimate of totden =      2677.896607
    estimate of average density =     0.9483
    projection data read
    Example 2,  A simple test pattern and projection data
         0.001 seconds used for processing command proj


<*>

<#> MODE LOWER constraint set at 0.0  UPPER constraint set at 2.5
 lower constraint set to    0.0000
 upper constraint set to    2.5000
         0.000 seconds used for processing command mode
```

```
<*>

<#> SELECT USER RAYSEQ

<#> STEP projections by 10 and rays by 1
    sequential ray selection with ray-step     1 and projection-step    10
         0.000 seconds used for processing command sele


<*>

<#> STOP TERMINATION VARIANCE at 0.01
    termination test vari
    epsilon =   0.010000
         0.000 seconds used for processing command stop


<*>

<#> EXECUTE AVERAGE ART SMOOTH

    Example 2a  underrelaxed art4 with variable tolerance

<#> smoothing threshold is 0.2  Weights are 2.25 1.5 1.0
    reconstruction is smoothed after

    iterations    1   2   3   4   5   6   7   8   9  10
                 11  12  13  14  15  16  17  18  19  20
                 21  22  23  24  25  26  27  28  29  30
                 31  32  33  34  35  36  37  38  39  40
                 41  42  43  44  45  46  47  48  49  50
    last iteration

    threshold =   0.20000      w1 =   2.25000      w2 =   1.50000      w3 =   1.00000

<#> ART4  RELAXATION CONSTANT = 0.025  TOLERANCE VARIABLE decreasing
     art4 method
     relaxation parameter is    0.0250
     tolerance is variable

<#> CONSTRAINT BOUND CONRELAX CONSTANT = 0.5 NOMLIZE at each iteration
     constraint set to boundaries
     relax constraints with cr=    0.5000
     picture is normalized to    0.9483 after each iteration
      1890 rays are used for each iteration
     lower constraint =    0.0000
     upper constraint =    2.5000
    algorithm executed in iteration    1
         0.002 seconds for the execution of the algorithm
    iteration    1 completed
         0.002 seconds for this iteration
    algorithm executed in iteration    2
```

```
     0.002 seconds for the execution of the algorithm
iteration    2 completed
     0.002 seconds for this iteration
algorithm executed in iteration    3
     0.002 seconds for the execution of the algorithm
iteration    3 completed
     0.002 seconds for this iteration
algorithm executed in iteration    4
     0.002 seconds for the execution of the algorithm
iteration    4 completed
     0.002 seconds for this iteration
algorithm executed in iteration    5
     0.002 seconds for the execution of the algorithm
iteration    5 completed
     0.002 seconds for this iteration
algorithm executed in iteration    6
     0.002 seconds for the execution of the algorithm
iteration    6 completed
     0.002 seconds for this iteration
algorithm executed in iteration    7
     0.002 seconds for the execution of the algorithm
iteration    7 completed
     0.002 seconds for this iteration
algorithm executed in iteration    8
     0.002 seconds for the execution of the algorithm
iteration    8 completed
     0.002 seconds for this iteration
algorithm executed in iteration    9
     0.002 seconds for the execution of the algorithm
iteration    9 completed
     0.002 seconds for this iteration
algorithm executed in iteration   10
     0.002 seconds for the execution of the algorithm
iteration   10 completed
     0.002 seconds for this iteration
algorithm executed in iteration   11
     0.002 seconds for the execution of the algorithm
iteration   11 completed
     0.002 seconds for this iteration
algorithm executed in iteration   12
     0.002 seconds for the execution of the algorithm
iteration   12 completed
     0.002 seconds for this iteration
algorithm executed in iteration   13
     0.002 seconds for the execution of the algorithm
iteration   13 completed
     0.002 seconds for this iteration
algorithm executed in iteration   14
     0.002 seconds for the execution of the algorithm
iteration   14 completed
     0.002 seconds for this iteration
algorithm executed in iteration   15
     0.002 seconds for the execution of the algorithm
iteration   15 completed
```

```
             0.002 seconds for this iteration
        algorithm executed in iteration   16
             0.002 seconds for the execution of the algorithm
        iteration   16 completed
             0.002 seconds for this iteration
        algorithm executed in iteration   17
             0.002 seconds for the execution of the algorithm
        iteration   17 completed
             0.002 seconds for this iteration
        algorithm executed in iteration   18
             0.002 seconds for the execution of the algorithm
        iteration   18 completed
             0.002 seconds for this iteration
        algorithm executed in iteration   19
             0.003 seconds for the execution of the algorithm
        iteration   19 completed
             0.003 seconds for this iteration
        algorithm executed in iteration   20
             0.002 seconds for the execution of the algorithm
        iteration   20 completed
             0.002 seconds for this iteration
        algorithm executed in iteration   21
             0.002 seconds for the execution of the algorithm
         iterative process stops at iteration    21
         the change in variance is less than   0.009852 of the variance
        reconstruction completed after iteration   21
             0.002 seconds for this iteration
             0.039 seconds for all iterations
             0.041 seconds used for processing command exec


    <*>

    <#> STOP ITERATIONS = 30
          30 iterations
             0.000 seconds used for processing command stop


    <*>

    <#> EXECUTE ALP1

        Example 2b  underrelaxed art with extrapolation
        algorithm executed in iteration    1
             0.001 seconds for the execution of the algorithm
        iteration    1 completed
             0.001 seconds for this iteration
        algorithm executed in iteration    2
             0.001 seconds for the execution of the algorithm
        iteration    2 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   5.062718796778e-01
        algorithm executed in iteration    3
             0.001 seconds for the execution of the algorithm
```

```
        iteration    3 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   5.062718796778e-01
        algorithm executed in iteration    4
             0.000 seconds for the execution of the algorithm
        iteration    4 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   6.560964044719e-01
        algorithm executed in iteration    5
             0.001 seconds for the execution of the algorithm
        iteration    5 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   6.560964044719e-01
        algorithm executed in iteration    6
             0.000 seconds for the execution of the algorithm
        iteration    6 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   7.480594958675e-01
        algorithm executed in iteration    7
             0.001 seconds for the execution of the algorithm
        iteration    7 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   7.480594958675e-01
        algorithm executed in iteration    8
             0.000 seconds for the execution of the algorithm
        iteration    8 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   7.951514183812e-01
        algorithm executed in iteration    9
             0.001 seconds for the execution of the algorithm
        iteration    9 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   7.951514183812e-01
        algorithm executed in iteration   10
             0.000 seconds for the execution of the algorithm
        iteration   10 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.250655739411e-01
        algorithm executed in iteration   11
             0.001 seconds for the execution of the algorithm
        iteration   11 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.250655739411e-01
        algorithm executed in iteration   12
             0.000 seconds for the execution of the algorithm
        iteration   12 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.467169748437e-01
        algorithm executed in iteration   13
             0.000 seconds for the execution of the algorithm
        iteration   13 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.467169748437e-01
        algorithm executed in iteration   14
```

```
             0.000 seconds for the execution of the algorithm
        iteration   14 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.628058734349e-01
        algorithm executed in iteration   15
             0.000 seconds for the execution of the algorithm
        iteration   15 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.628058734349e-01
        algorithm executed in iteration   16
             0.000 seconds for the execution of the algorithm
        iteration   16 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.749346145383e-01
        algorithm executed in iteration   17
             0.001 seconds for the execution of the algorithm
        iteration   17 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.749346145383e-01
        algorithm executed in iteration   18
             0.000 seconds for the execution of the algorithm
        iteration   18 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.843623393369e-01
        algorithm executed in iteration   19
             0.002 seconds for the execution of the algorithm
        iteration   19 completed
             0.002 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.843623393369e-01
        algorithm executed in iteration   20
             0.000 seconds for the execution of the algorithm
        iteration   20 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.920147280181e-01
        algorithm executed in iteration   21
             0.000 seconds for the execution of the algorithm
        iteration   21 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.920147280181e-01
        algorithm executed in iteration   22
             0.000 seconds for the execution of the algorithm
        iteration   22 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.985122210336e-01
        algorithm executed in iteration   23
             0.000 seconds for the execution of the algorithm
        iteration   23 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   8.985122210336e-01
        algorithm executed in iteration   24
             0.000 seconds for the execution of the algorithm
        iteration   24 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.042443038619e-01
```

```
        algorithm executed in iteration   25
             0.001 seconds for the execution of the algorithm
        iteration   25 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.042443038619e-01
        algorithm executed in iteration   26
             0.000 seconds for the execution of the algorithm
        iteration   26 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.094448804097e-01
        algorithm executed in iteration   27
             0.001 seconds for the execution of the algorithm
        iteration   27 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.094448804097e-01
        algorithm executed in iteration   28
             0.000 seconds for the execution of the algorithm
        iteration   28 completed
             0.000 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.142500229503e-01
        algorithm executed in iteration   29
             0.001 seconds for the execution of the algorithm
        iteration   29 completed
             0.001 seconds for this iteration
EXTRAPOLATION PARAMETER IN EXTRAP IS   9.142500229503e-01
        algorithm executed in iteration   30
             0.000 seconds for the execution of the algorithm
        reconstruction completed after iteration   30
             0.000 seconds for this iteration
             0.014 seconds for all iterations
             0.015 seconds used for processing command exec


    <*>

    <*>  Reset mode to unconstrained

    <#> MODE
             0.000 seconds used for processing command mode


    <*>

    <#> EVALUATE RESOLUTION

        Example 2a  underrelaxed art4

    <#> WHOLEPIC
Region        cx        cy         u         v        ang      t1        t2

wholepic                                                      -1e+20     1e+20

        iterations    1   2   3   4   5   6   7   8   9  10
                     11  12  13  14  15  16  17  18  19  20
```

```
                    21  22  23  24  25  26  27  28  29  30
                    31  32  33  34  35  36  37  38  39
        last iteration

   residual calculation using line integration
             0.061 seconds used for processing command eval


    <*>

    <#> LINES COLUMNS 15 and 18 are to be printed

        iterations   10  20  30
        last iteration

    lines display of reconstruction using ART iter 10

    phantom name:    Example 2,  A simple test pattern and projection data
    projection data: Example 2,  A simple test pattern and projection data
    execution name:  Example 2a  underrelaxed art4 with variable tolerance
    scaling factor =    1.00000
```

column =	15			18		
row	original	estimate	difference	original	estimate	difference
0	0.00000	0.18210	0.18210	0.00000	0.00909	0.00909
1	0.00000	0.20450	0.20450	0.00000	0.24753	0.24753
2	0.00000	1.10539	1.10539	0.00000	0.52569	0.52569
3	2.50000	2.13754	-0.36246	2.50000	1.94767	-0.55233
4	2.50000	2.16437	-0.33563	2.50000	2.18812	-0.31188
5	2.50000	2.16394	-0.33606	2.50000	2.20758	-0.29242
6	2.50000	2.13317	-0.36683	2.50000	2.17610	-0.32390
7	1.25000	1.49667	0.24667	1.25000	1.68287	0.43287
8	1.25000	1.44121	0.19121	1.25000	1.41529	0.16529
9	1.25000	1.37200	0.12200	1.25000	1.38374	0.13374
10	1.25000	1.34328	0.09328	1.25000	1.34276	0.09276
11	1.25000	1.33396	0.08396	1.25000	1.32222	0.07222
12	1.35000	1.34036	-0.00964	1.25000	1.31411	0.06411
13	1.35000	1.34676	-0.00324	1.25000	1.31521	0.06521
14	1.35000	1.35058	0.00058	1.25000	1.31568	0.06568
15	1.35000	1.35053	0.00053	1.35000	1.31710	-0.03290
16	1.35000	1.35058	0.00058	1.25000	1.31607	0.06607
17	1.35000	1.34676	-0.00324	1.25000	1.31587	0.06587
18	1.35000	1.34035	-0.00965	1.25000	1.31417	0.06417
19	1.25000	1.33395	0.08395	1.25000	1.32131	0.07131
20	1.25000	1.34326	0.09326	1.25000	1.34021	0.09021
21	1.25000	1.37196	0.12196	1.25000	1.38008	0.13008
22	1.25000	1.44114	0.19114	1.25000	1.41090	0.16090
23	1.25000	1.49658	0.24658	1.25000	1.68040	0.43040
24	2.50000	2.13333	-0.36667	2.50000	2.17405	-0.32595
25	2.50000	2.16414	-0.33586	2.50000	2.20535	-0.29465
26	2.50000	2.16455	-0.33545	2.50000	2.18655	-0.31345
27	2.50000	2.13774	-0.36226	2.50000	1.94456	-0.55544
28	0.00000	1.10534	1.10534	0.00000	0.44393	0.44393
29	0.00000	0.20439	0.20439	0.00000	0.29112	0.29112

```
 30     0.00000    0.18198    0.18198    0.00000    0.01326    0.01326


    lines display of reconstruction using ART iter 20

    phantom name:    Example 2,  A simple test pattern and projection data
    projection data: Example 2,  A simple test pattern and projection data
    execution name:  Example 2a  underrelaxed art4 with variable tolerance
    scaling factor =    1.00000

column =              15                              18
row    original   estimate difference   original   estimate difference
  0     0.00000    0.00164    0.00164    0.00000    0.02169    0.02169
  1     0.00000    0.00132    0.00132    0.00000    0.04480    0.04480
  2     0.00000    0.94940    0.94940    0.00000    0.13226    0.13226
  3     2.50000    2.43151   -0.06849    2.50000    2.25420   -0.24580
  4     2.50000    2.43097   -0.06903    2.50000    2.44794   -0.05206
  5     2.50000    2.37936   -0.12064    2.50000    2.44383   -0.05617
  6     2.50000    2.31917   -0.18083    2.50000    2.39051   -0.10949
  7     1.25000    1.46751    0.21751    1.25000    1.65881    0.40881
  8     1.25000    1.31935    0.06935    1.25000    1.33707    0.08707
  9     1.25000    1.29770    0.04770    1.25000    1.31051    0.06051
 10     1.25000    1.27744    0.02744    1.25000    1.27762    0.02762
 11     1.25000    1.28043    0.03043    1.25000    1.26720    0.01720
 12     1.35000    1.30399   -0.04601    1.25000    1.26843    0.01843
 13     1.35000    1.32095   -0.02905    1.25000    1.27985    0.02985
 14     1.35000    1.32945   -0.02055    1.25000    1.28645    0.03645
 15     1.35000    1.32847   -0.02153    1.35000    1.29098   -0.05902
 16     1.35000    1.32944   -0.02056    1.25000    1.28714    0.03714
 17     1.35000    1.32093   -0.02907    1.25000    1.28136    0.03136
 18     1.35000    1.30399   -0.04601    1.25000    1.27000    0.02000
 19     1.25000    1.28044    0.03044    1.25000    1.26871    0.01871
 20     1.25000    1.27744    0.02744    1.25000    1.27815    0.02815
 21     1.25000    1.29767    0.04767    1.25000    1.30973    0.05973
 22     1.25000    1.31930    0.06930    1.25000    1.33504    0.08504
 23     1.25000    1.46737    0.21737    1.25000    1.71927    0.46927
 24     2.50000    2.31941   -0.18059    2.50000    2.39181   -0.10819
 25     2.50000    2.37969   -0.12031    2.50000    2.44490   -0.05510
 26     2.50000    2.43125   -0.06875    2.50000    2.44917   -0.05083
 27     2.50000    2.43179   -0.06821    2.50000    2.25444   -0.24556
 28     0.00000    0.94928    0.94928    0.00000    0.12809    0.12809
 29     0.00000    0.00158    0.00158    0.00000    0.04576    0.04576
 30     0.00000    0.00199    0.00199    0.00000    0.02628    0.02628


    lines display of reconstruction using ART iter 21

    phantom name:    Example 2,  A simple test pattern and projection data
    projection data: Example 2,  A simple test pattern and projection data
    execution name:  Example 2a  underrelaxed art4 with variable tolerance
    scaling factor =    1.00000

column =              15                              18
row    original   estimate difference   original   estimate difference
```

0	0.00000	0.00100	0.00100	0.00000	0.01742	0.01742
1	0.00000	0.00070	0.00070	0.00000	0.03672	0.03672
2	0.00000	0.93506	0.93506	0.00000	0.11759	0.11759
3	2.50000	2.44949	-0.05051	2.50000	2.27648	-0.22352
4	2.50000	2.44615	-0.05385	2.50000	2.45838	-0.04162
5	2.50000	2.39029	-0.10971	2.50000	2.45205	-0.04795
6	2.50000	2.32845	-0.17155	2.50000	2.39929	-0.10071
7	1.25000	1.46233	0.21233	1.25000	1.65516	0.40516
8	1.25000	1.31367	0.06367	1.25000	1.33085	0.08085
9	1.25000	1.29279	0.04279	1.25000	1.30546	0.05546
10	1.25000	1.27400	0.02400	1.25000	1.27412	0.02412
11	1.25000	1.27844	0.02844	1.25000	1.26496	0.01496
12	1.35000	1.30374	-0.04626	1.25000	1.26714	0.01714
13	1.35000	1.32159	-0.02841	1.25000	1.27948	0.02948
14	1.35000	1.33036	-0.01964	1.25000	1.28660	0.03660
15	1.35000	1.32919	-0.02081	1.35000	1.29140	-0.05860
16	1.35000	1.33034	-0.01966	1.25000	1.28726	0.03726
17	1.35000	1.32157	-0.02843	1.25000	1.28096	0.03096
18	1.35000	1.30372	-0.04628	1.25000	1.26874	0.01874
19	1.25000	1.27844	0.02844	1.25000	1.26657	0.01657
20	1.25000	1.27400	0.02400	1.25000	1.27479	0.02479
21	1.25000	1.29277	0.04277	1.25000	1.30483	0.05483
22	1.25000	1.31362	0.06362	1.25000	1.32898	0.07898
23	1.25000	1.46218	0.21218	1.25000	1.65781	0.40781
24	2.50000	2.32869	-0.17131	2.50000	2.40065	-0.09935
25	2.50000	2.39063	-0.10937	2.50000	2.45318	-0.04682
26	2.50000	2.44644	-0.05356	2.50000	2.45967	-0.04033
27	2.50000	2.44977	-0.05023	2.50000	2.27692	-0.22308
28	0.00000	0.93489	0.93489	0.00000	0.11431	0.11431
29	0.00000	0.00120	0.00120	0.00000	0.03840	0.03840
30	0.00000	0.00161	0.00161	0.00000	0.02249	0.02249

```
   lines display of reconstruction using ALP1 iter 10

   phantom name:    Example 2,  A simple test pattern and projection data
   projection data: Example 2,  A simple test pattern and projection data
   execution name:  Example 2b  underrelaxed art with extrapolation
   scaling factor =    1.00000
```

column =	15			18		
row	original	estimate	difference	original	estimate	difference
0	0.00000	0.12670	0.12670	0.00000	-0.02567	-0.02567
1	0.00000	0.17187	0.17187	0.00000	0.16958	0.16958
2	0.00000	1.20151	1.20151	0.00000	0.48350	0.48350
3	2.50000	2.23551	-0.26449	2.50000	2.01480	-0.48520
4	2.50000	2.39456	-0.10544	2.50000	2.31508	-0.18492
5	2.50000	2.34966	-0.15034	2.50000	2.46624	-0.03376
6	2.50000	2.31124	-0.18876	2.50000	2.29509	-0.20491
7	1.25000	1.63106	0.38106	1.25000	1.81482	0.56482
8	1.25000	1.35183	0.10183	1.25000	1.44203	0.19203
9	1.25000	1.42885	0.17885	1.25000	1.45015	0.20015
10	1.25000	1.31859	0.06859	1.25000	1.36973	0.11973
11	1.25000	1.42731	0.17731	1.25000	1.34954	0.09954

12	1.35000	1.31939	-0.03061	1.25000	1.31390	0.06390
13	1.35000	1.37323	0.02323	1.25000	1.32415	0.07415
14	1.35000	1.46590	0.11590	1.25000	1.37584	0.12584
15	1.35000	1.13749	-0.21251	1.35000	1.25214	-0.09786
16	1.35000	1.46594	0.11594	1.25000	1.36902	0.11902
17	1.35000	1.37317	0.02317	1.25000	1.30699	0.05699
18	1.35000	1.31944	-0.03056	1.25000	1.34043	0.09043
19	1.25000	1.42727	0.17727	1.25000	1.33982	0.08982
20	1.25000	1.31861	0.06861	1.25000	1.35654	0.10654
21	1.25000	1.42880	0.17880	1.25000	1.44218	0.19218
22	1.25000	1.35172	0.10172	1.25000	1.45375	0.20375
23	1.25000	1.63100	0.38100	1.25000	1.80751	0.55751
24	2.50000	2.31161	-0.18839	2.50000	2.27967	-0.22033
25	2.50000	2.34937	-0.15063	2.50000	2.50118	0.00118
26	2.50000	2.39545	-0.10455	2.50000	2.30942	-0.19058
27	2.50000	2.23526	-0.26474	2.50000	1.99542	-0.50458
28	0.00000	1.20133	1.20133	0.00000	0.49478	0.49478
29	0.00000	0.17173	0.17173	0.00000	0.15734	0.15734
30	0.00000	0.12647	0.12647	0.00000	-0.02832	-0.02832

```
   lines display of reconstruction using ALP1 iter 20

   phantom name:    Example 2,  A simple test pattern and projection data
   projection data: Example 2,  A simple test pattern and projection data
   execution name:  Example 2b  underrelaxed art with extrapolation
   scaling factor =    1.00000
```

column =	15			18		
row	original	estimate	difference	original	estimate	difference
0	0.00000	-0.09663	-0.09663	0.00000	-0.18986	-0.18986
1	0.00000	-0.12331	-0.12331	0.00000	-0.05402	-0.05402
2	0.00000	1.04579	1.04579	0.00000	0.12373	0.12373
3	2.50000	2.51540	0.01540	2.50000	2.25540	-0.24460
4	2.50000	2.59753	0.09753	2.50000	2.56869	0.06869
5	2.50000	2.55529	0.05529	2.50000	2.67846	0.17846
6	2.50000	2.40841	-0.09159	2.50000	2.45046	-0.04954
7	1.25000	1.52857	0.27857	1.25000	1.73851	0.48851
8	1.25000	1.19882	-0.05118	1.25000	1.29003	0.04003
9	1.25000	1.28220	0.03220	1.25000	1.30029	0.05029
10	1.25000	1.18525	-0.06475	1.25000	1.24281	-0.00719
11	1.25000	1.27819	0.02819	1.25000	1.24380	-0.00620
12	1.35000	1.25011	-0.09989	1.25000	1.22237	-0.02763
13	1.35000	1.30966	-0.04034	1.25000	1.24476	-0.00524
14	1.35000	1.38630	0.03630	1.25000	1.28583	0.03583
15	1.35000	1.11801	-0.23199	1.35000	1.20027	-0.14973
16	1.35000	1.38636	0.03636	1.25000	1.28050	0.03050
17	1.35000	1.30959	-0.04041	1.25000	1.23298	-0.01702
18	1.35000	1.25014	-0.09986	1.25000	1.24848	-0.00152
19	1.25000	1.27817	0.02817	1.25000	1.23597	-0.01403
20	1.25000	1.18526	-0.06474	1.25000	1.23189	-0.01811
21	1.25000	1.28211	0.03211	1.25000	1.29998	0.04998
22	1.25000	1.19865	-0.05135	1.25000	1.30195	0.05195
23	1.25000	1.52839	0.27839	1.25000	1.73195	0.48195

```
 24     2.50000    2.40899   -0.09101    2.50000    2.44156   -0.05844
 25     2.50000    2.55477    0.05477    2.50000    2.70939    0.20939
 26     2.50000    2.59890    0.09890    2.50000    2.56774    0.06774
 27     2.50000    2.51498    0.01498    2.50000    2.24565   -0.25435
 28     0.00000    1.04557    1.04557    0.00000    0.14374    0.14374
 29     0.00000   -0.12338   -0.12338    0.00000   -0.05331   -0.05331
 30     0.00000   -0.09670   -0.09670    0.00000   -0.17428   -0.17428


    lines display of reconstruction using ALP1 iter 30

    phantom name:    Example 2,  A simple test pattern and projection data
    projection data: Example 2,  A simple test pattern and projection data
    execution name:  Example 2b  underrelaxed art with extrapolation
    scaling factor =    1.00000

column =              15                             18
row    original   estimate difference   original   estimate difference
  0     0.00000   -0.10295   -0.10295    0.00000   -0.15511   -0.15511
  1     0.00000   -0.21068   -0.21068    0.00000   -0.09236   -0.09236
  2     0.00000    0.92855    0.92855    0.00000   -0.06300   -0.06300
  3     2.50000    2.64511    0.14511    2.50000    2.37419   -0.12581
  4     2.50000    2.61843    0.11843    2.50000    2.63291    0.13291
  5     2.50000    2.61277    0.11277    2.50000    2.68002    0.18002
  6     2.50000    2.39593   -0.10407    2.50000    2.48350   -0.01650
  7     1.25000    1.47574    0.22574    1.25000    1.66721    0.41721
  8     1.25000    1.11475   -0.13525    1.25000    1.19787   -0.05213
  9     1.25000    1.24880   -0.00120    1.25000    1.25371    0.00371
 10     1.25000    1.15550   -0.09450    1.25000    1.22544   -0.02456
 11     1.25000    1.25545    0.00545    1.25000    1.24429   -0.00571
 12     1.35000    1.26430   -0.08570    1.25000    1.22826   -0.02174
 13     1.35000    1.32865   -0.02135    1.25000    1.25347    0.00347
 14     1.35000    1.40845    0.05845    1.25000    1.28571    0.03571
 15     1.35000    1.14519   -0.20481    1.35000    1.22750   -0.12250
 16     1.35000    1.40851    0.05851    1.25000    1.28178    0.03178
 17     1.35000    1.32857   -0.02143    1.25000    1.24374   -0.00626
 18     1.35000    1.26433   -0.08567    1.25000    1.25314    0.00314
 19     1.25000    1.25545    0.00545    1.25000    1.23609   -0.01391
 20     1.25000    1.15550   -0.09450    1.25000    1.21469   -0.03531
 21     1.25000    1.24869   -0.00131    1.25000    1.25626    0.00626
 22     1.25000    1.11456   -0.13544    1.25000    1.20898   -0.04102
 23     1.25000    1.47547    0.22547    1.25000    1.65907    0.40907
 24     2.50000    2.39664   -0.10336    2.50000    2.47619   -0.02381
 25     2.50000    2.61204    0.11204    2.50000    2.70609    0.20609
 26     2.50000    2.62009    0.12009    2.50000    2.63048    0.13048
 27     2.50000    2.64455    0.14455    2.50000    2.36445   -0.13555
 28     0.00000    0.92830    0.92830    0.00000   -0.04433   -0.04433
 29     0.00000   -0.21071   -0.21071    0.00000   -0.09436   -0.09436
 30     0.00000   -0.10286   -0.10286    0.00000   -0.14676   -0.14676


              0.001 seconds used for processing command line
```

```
<*>

<#> END
```

A.2.5 SNARK14 eval file

```
evaluation name: Example 2a  underrelaxed art4

  global resolution measures

    phantom name:    Example 2,  A simple test pattern and projection data
    projection name: Example 2,  A simple test pattern and projection data

    metrics for test phantom
    region    area   average  distance   rel err  variance   std dev   residual
         0     961    0.9387                        0.9537    0.9766     4.7170

    execution name: Example 2a  underrelaxed art4 with variable tolerance
    metrics for algorithm ART
    iter    area   average  distance   rel err  variance   std dev   residual
       1     961    0.9484    0.8307    0.7211    0.0438    0.2092    22.9038
       2     961    0.9478    0.7000    0.5673    0.1369    0.3701    17.9587
       3     961    0.9478    0.6125    0.4813    0.2360    0.4858    14.6351
       4     961    0.9476    0.5477    0.4250    0.3252    0.5703    12.3604
       5     961    0.9475    0.4997    0.3803    0.4020    0.6340    10.7362
       6     961    0.9476    0.4637    0.3448    0.4650    0.6819     9.5344
       7     961    0.9476    0.4333    0.3139    0.5164    0.7186     8.6035
       8     961    0.9476    0.4087    0.2871    0.5608    0.7489     7.8372
       9     961    0.9476    0.3854    0.2632    0.5993    0.7742     7.2097
      10     961    0.9477    0.3690    0.2434    0.6335    0.7959     6.6760
      11     961    0.9476    0.3556    0.2262    0.6628    0.8141     6.2355
      12     961    0.9475    0.3437    0.2103    0.6892    0.8302     5.8462
      13     961    0.9475    0.3322    0.1952    0.7134    0.8446     5.5007
      14     961    0.9476    0.3242    0.1828    0.7345    0.8571     5.2199
      15     961    0.9476    0.3172    0.1718    0.7530    0.8678     4.9834
      16     961    0.9478    0.3115    0.1620    0.7699    0.8774     4.7767
      17     961    0.9479    0.3057    0.1530    0.7846    0.8858     4.5968
      18     961    0.9479    0.3011    0.1452    0.7974    0.8930     4.4485
      19     961    0.9480    0.2973    0.1385    0.8088    0.8994     4.3199
      20     961    0.9479    0.2925    0.1319    0.8184    0.9047     4.2163
      21     961    0.9479    0.2894    0.1268    0.8266    0.9092     4.1266

    execution name: Example 2b  underrelaxed art with extrapolation
    metrics for algorithm ALP1
    iter    area   average  distance   rel err  variance   std dev   residual
       1     961    0.6390    0.8612    0.7157    0.0587    0.2422    34.9047
       2     961    0.8466    0.6860    0.5398    0.1645    0.4056    19.6639
       3     961    0.9140    0.5957    0.4520    0.2696    0.5192    14.3027
       4     961    0.9832    0.5246    0.4168    0.4052    0.6366    11.0965
       5     961    0.9359    0.5351    0.4042    0.3609    0.6007    11.7289
       6     961    0.9778    0.4478    0.3460    0.5752    0.7584     8.7625
```

```
 7     961    0.9431    0.4901    0.3659    0.4370    0.6610    10.1210
 8     961    0.9642    0.3961    0.2893    0.7134    0.8446     7.1597
 9     961    0.9454    0.4552    0.3349    0.5001    0.7072     8.9677
10     961    0.9543    0.3652    0.2478    0.7967    0.8926     6.1111
11     961    0.9461    0.4273    0.3089    0.5527    0.7434     8.0793
12     961    0.9496    0.3450    0.2210    0.8512    0.9226     5.3651
13     961    0.9463    0.4046    0.2863    0.5972    0.7728     7.3662
14     961    0.9476    0.3313    0.2020    0.8910    0.9439     4.8159
15     961    0.9464    0.3859    0.2664    0.6353    0.7970     6.7790
16     961    0.9467    0.3220    0.1892    0.9204    0.9594     4.4152
17     961    0.9464    0.3703    0.2485    0.6683    0.8175     6.2875
18     961    0.9464    0.3156    0.1820    0.9419    0.9705     4.1232
19     961    0.9464    0.3573    0.2324    0.6972    0.8350     5.8714
20     961    0.9463    0.3111    0.1786    0.9580    0.9788     3.9053
21     961    0.9464    0.3465    0.2182    0.7229    0.8502     5.5163
22     961    0.9462    0.3078    0.1766    0.9707    0.9852     3.7356
23     961    0.9463    0.3373    0.2062    0.7457    0.8635     5.2110
24     961    0.9462    0.3053    0.1746    0.9813    0.9906     3.5972
25     961    0.9463    0.3295    0.1965    0.7662    0.8753     4.9471
26     961    0.9461    0.3035    0.1728    0.9907    0.9953     3.4798
27     961    0.9463    0.3230    0.1887    0.7847    0.8858     4.7175
28     961    0.9461    0.3023    0.1713    0.9993    0.9997     3.3772
29     961    0.9463    0.3174    0.1819    0.8015    0.8953     4.5167
30     961    0.9461    0.3015    0.1706    1.0074    1.0037     3.2857
```

A.3 Realistic x-ray data

In this example we illustrate the capability of SNARK14 to simulate taking of x-ray projection data in a realistic fashion. The test phantom is very similar to the output of an actual CT scan. The data are collected for one projection according to a geometry similar to that of the rotational scanners. The polychromaticity, photon statistics and scatter of x-rays are all simulated, as well as the divergence of the beam between the source and the detector. In addition to the INPUT and OUTPUT files we give a listing of the file11 produced by this run.

A.3.1 SNARK14 INPUT file

```
* EXAMPLE 3
* CREATION OF HEAD PHANTOM. GENERATION OF POLYCHROMATIC
* PROJECTION DATA.
*
* CREATE A PHANTOM
* GENERATE PROJECTION DATA
CREATE
EXAMPLE 3 HEAD PHANTOM
SPECTRUM POLYCHROMATIC 5
40 10 50 30 60 30 80 20 100 10
OBJECTS
SEGM    1.375   -7.5    1.1    0.625    19.2   -0.734
DENS   -0.472   -0.288 -0.116 -0.029
SEGM    1.375   -7.5    1.1    4.32     19.21   0.734
DENS    0.472    0.288  0.116  0.029
ELIP    0.0      0.0    8.625  6.4687   90.0    1.0
```

```
DENS    0.7      0.5    0.3    0.2
ELIP    0.0      0.0    7.875  5.7187   90.0   -0.737
DENS   -0.475   -0.291 -0.118 -0.029
ELIP    0.0      1.5    0.375  0.3      90.0   -0.003
DENS   -0.006   -0.002 -0.001 -0.0
SEGM    0.0     -2.25   1.125  0.375     0.0   -0.003
DENS   -0.006   -0.002 -0.001 -0.0
SEGM   -1.0      3.75   1.0    0.5     135.0   -0.003
DENS   -0.006   -0.002 -0.001  0.0
SEGM    1.0      3.75   1.0    0.5     225.0   -0.003
DENS   -0.006   -0.002 -0.001  0.0
ELIP    0.675   -0.75   0.225  0.15    140.0    0.025
DENS    0.016    0.01   0.008  0.008
ELIP    0.75     1.5    0.375  0.225    50.0    0.005
DENS    0.004    0.004  0.006  0.005
TRIA    5.025    3.75   1.125  0.5     110.75   0.737
DENS    0.475    0.291  0.118  0.029
TRIA   -5.025    3.75   1.125  0.5    -110.75   0.737
DENS    0.475    0.291  0.118  0.029
LAST 1.0
PHANTOM AVERAGE 1
23 0.77
RAYSUM AVERAGE 3
1 1 1
GEOMETRY
DIVERGENT ARC 78.0 110.0
RAYS USER 159 0.213
ANGLE 1
0.0
MEASUREMENT NOISY
QUANTUM 10000000.0 360.0 CALIBRATION 2
SCATTER 0.24 0.4445
SEED 1
BACKGROUND 0.0 0.0 0.0 0.0 0.0
RUN
END
```

A.3.2 SNARK14 OUTPUT file

```
snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


  <*>  EXAMPLE 3

  <*>  CREATION OF HEAD PHANTOM. GENERATION OF POLYCHROMATIC

  <*>  PROJECTION DATA.

  <*>

  <*>  CREATE A PHANTOM

  <*>  GENERATE PROJECTION DATA
```

```
<#> CREATE

    EXAMPLE 3 HEAD PHANTOM


<#> SPECTRUM POLYCHROMATIC 5

    energy spectrum is polychromatic with   5     energy levels
    with the following photon distribution
    energy level   percent
          40          10
          50          30
          60          30
          80          20
         100          10

<#> OBJECTS
    description of objects
                                                                                  density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        40        50        60
   1 segm   1.3750  -7.5000   1.1000   0.6250  19.2000  -0.3275   -0.7340   -0.4720   -0.2880   -0
   2 segm   1.3750  -7.5000   1.1000   4.3200  19.2100   0.3275    0.7340    0.4720    0.2880    0
   3 elip   0.0000   0.0000   8.6250   6.4687  90.0000   0.5400    1.0000    0.7000    0.5000    0
   4 elip   0.0000   0.0000   7.8750   5.7187  90.0000  -0.3300   -0.7370   -0.4750   -0.2910   -0
   5 elip   0.0000   1.5000   0.3750   0.3000  90.0000  -0.0029   -0.0030   -0.0060   -0.0020   -0
   6 segm   0.0000  -2.2500   1.1250   0.3750   0.0000  -0.0029   -0.0030   -0.0060   -0.0020   -0
   7 segm  -1.0000   3.7500   1.0000   0.5000 135.0000  -0.0029   -0.0030   -0.0060   -0.0020   -0
   8 segm   1.0000   3.7500   1.0000   0.5000 225.0000  -0.0029   -0.0030   -0.0060   -0.0020   -0
   9 elip   0.6750  -0.7500   0.2250   0.1500 140.0000   0.0127    0.0250    0.0160    0.0100    0
  10 elip   0.7500   1.5000   0.3750   0.2250  50.0000   0.0046    0.0050    0.0040    0.0040    0
  11 tria   5.0250   3.7500   1.1250   0.5000 110.7500   0.3300    0.7370    0.4750    0.2910    0
```

```
12 tria  -5.0250   3.7500   1.1250   0.5000 -110.7500   0.3300    0.7370    0.4750    0.2910

     scale factor multiplying object densities     1.0000

     seed set to 0
     inhomogeneity set to     0.0000

  <#> PHANTOM AVERAGE 1

     this run will generate a phantom
     density in each pixel is obtained as the average of 1 x 1 points


  <#> 23 0.77
     picture size 23 x 23,  pixel size     0.7700


  <#> RAYSUM AVERAGE 3

     this run will generate projection data
     projection data are calculated by dividing each ray interval into 3 substrips

     with aperture (substrip) weights     1     1     1


  <#> GEOMETRY


  <#> DIVERGENT ARC 78.0 110.0
     rays are divergent from point sources
     source to origin distance     78.0000
     the detectors lie on an arc with source to detector distance =   110.0000


  <#> RAYS USER 159 0.213
     number of rays per projection   159
     at detector spacing     0.2130


  <#> ANGLE 1
     total number of projections     1

     projection angles    0.0000


  <#> MEASUREMENT NOISY
     noise characteristics of projection data follow
           nature          characteristics

  <#> QUANTUM 10000000.0 360.0 CALIBRATION 2
           quantum        mean number of photons in actual measurement   1.0000e+07
                          mean number of photons in calibration measurement is   360.0000    ti
```

```
                              calibration is constant over all projections for any single ray
                              mean numbers of photons entering each substrip are
                                3.3333e+06   3.3333e+06   3.3333e+06

     <#> SCATTER 0.24 0.4445
               scatter        the fraction scattered in line is     0.2400
                              scatter extends to a distance of     0.4445
                              dying linearly from the middle

     <#> SEED 1
          seed for random number generator is              1


     <#> BACKGROUND 0.0 0.0 0.0 0.0 0.0
                                at levels
                                       40         50         60         80        100
          background absorption   0.0000     0.0000     0.0000     0.0000     0.0000


     <#> RUN
               0.001 seconds phantom creation
               0.001 seconds projection data creation
               0.002 seconds used for processing command crea


     <#> END
```

A.3.3 SNARK14 file11

```
EXAMPLE 3 HEAD PHANTOM
spec    poly    5
   40    10    50    30    60    30    80    20   100    10
obje
segm     1.3750   -7.5000    1.1000    0.6250   19.2000     -0.733999999999999986
dens    -0.4720000000   -0.2880000000   -0.1160000000   -0.0290000000
segm     1.3750   -7.5000    1.1000    4.3200   19.2100      0.733999999999999986
dens     0.4720000000    0.2880000000    0.1160000000    0.0290000000
elip     0.0000    0.0000    8.6250    6.4687   90.0000      1.000000000000000000
dens     0.7000000000    0.5000000000    0.3000000000    0.2000000000
elip     0.0000    0.0000    7.8750    5.7187   90.0000     -0.736999999999999988
dens    -0.4750000000   -0.2910000000   -0.1180000000   -0.0290000000
elip     0.0000    1.5000    0.3750    0.3000   90.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000   -0.0000000000
segm     0.0000   -2.2500    1.1250    0.3750    0.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000   -0.0000000000
segm    -1.0000    3.7500    1.0000    0.5000  135.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000    0.0000000000
segm     1.0000    3.7500    1.0000    0.5000  225.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000    0.0000000000
elip     0.6750   -0.7500    0.2250    0.1500  140.0000      0.025000000000000001
dens     0.0160000000    0.0100000000    0.0080000000    0.0080000000
elip     0.7500    1.5000    0.3750    0.2250   50.0000      0.005000000000000000
dens     0.0040000000    0.0040000000    0.0060000000    0.0050000000
tria     5.0250    3.7500    1.1250    0.5000  110.7500      0.736999999999999988
```

```
dens     0.4750000000    0.2910000000    0.1180000000    0.0290000000
tria    -5.0250    3.7500    1.1250    0.5000 -110.7500      0.73699999999999988
dens     0.4750000000    0.2910000000    0.1180000000    0.0290000000
last     1.0000 0     0.0000
phan    aver    1
pixe       23    size        0.7700
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.00000000000000000000    1.00000000000000000000    1.00000000000000000000    0.263000000000000
    1.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    1.00000000000000000000    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    1.00000000000000000000    1.00000000000000000000    0.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    1.00000000000000000000    0.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    1.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    1.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.260000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    1.000000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.00000000000000000000    0.00000000000000000000
    0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
    0.00000000000000000000    0.00000000000000000000
```

```
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.260000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    1.000000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.26300000000000001155    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    0.26300000000000001155    1.00000000000000000000    0.000000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        1.00000000000000000000    0.26300000000000001155    0.26300000000000001155    0.263000000000000
        0.26300000000000001155    1.00000000000000000000    1.00000000000000000000    0.000000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.00000000000000000000    1.00000000000000000000    1.00000000000000000000    0.263000000000000
        1.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.00000000000000000000    0.00000000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    1.000000000000000
        0.00000000000000000000    0.00000000000000000000    0.00000000000000000000    0.000000000000000
        0.00000000000000000000    0.00000000000000000000
EXAMPLE 3 HEAD PHANTOM
spec    poly    5
   40    10    50    30    60    30    80    20   100    10
obje
segm     1.3750   -7.5000    1.1000    0.6250   19.2000     -0.733999999999999986
dens    -0.4720000000   -0.2880000000   -0.1160000000   -0.0290000000
segm     1.3750   -7.5000    1.1000    4.3200   19.2100      0.733999999999999986
dens     0.4720000000    0.2880000000    0.1160000000    0.0290000000
elip     0.0000    0.0000    8.6250    6.4687   90.0000      1.000000000000000000
dens     0.7000000000    0.5000000000    0.3000000000    0.2000000000
```

```
elip     0.0000    0.0000    7.8750    5.7187   90.0000     -0.736999999999999988
dens    -0.4750000000   -0.2910000000   -0.1180000000   -0.0290000000
elip     0.0000    1.5000    0.3750    0.3000   90.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000   -0.0000000000
segm     0.0000   -2.2500    1.1250    0.3750    0.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000   -0.0000000000
segm    -1.0000    3.7500    1.0000    0.5000  135.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000    0.0000000000
segm     1.0000    3.7500    1.0000    0.5000  225.0000     -0.003000000000000000
dens    -0.0060000000   -0.0020000000   -0.0010000000    0.0000000000
elip     0.6750   -0.7500    0.2250    0.1500  140.0000      0.025000000000000001
dens     0.0160000000    0.0100000000    0.0080000000    0.0080000000
elip     0.7500    1.5000    0.3750    0.2250   50.0000      0.005000000000000000
dens     0.0040000000    0.0040000000    0.0060000000    0.0050000000
tria     5.0250    3.7500    1.1250    0.5000  110.7500      0.736999999999999988
dens     0.4750000000    0.2910000000    0.1180000000    0.0290000000
tria    -5.0250    3.7500    1.1250    0.5000 -110.7500      0.736999999999999988
dens     0.4750000000    0.2910000000    0.1180000000    0.0290000000
last     1.0000 0     0.0000
rays    aver    3
    1    1    1
geom
dive    arc     source at   78.0000     det dist  110.0000
rays    user      159    spacing        0.2130
angl        1
    0.0000
meas    nois
quan     10000000.0000          360.0000    cali  2
scat    noise peak         0.2400000000    width         0.4445000000
seed       1
back      0.0000  0.0000  0.0000  0.0000  0.0000
    0.00000000000000000000    0.00000000000000000000
   -0.00007805279554173564    0.00004650918767980816   -0.00004473035394917833
   -0.00075472753879972268    0.00021147322589549765   -0.00017424446235429476
    0.00021021831069931401   -0.00032114616516160734    0.00080472839342380666
   -0.00032644267523053823    0.00048031635563266335   -0.00012797762539250472
    0.00006385006732581144   -0.00089929757471288543    0.00053314583792116199
    0.00012658356922487648   -0.00009455298249226776    0.00059599287131110480
    0.00024332499303416426    0.00002459151306294491    0.00118821246111811777
    0.02750129911123748092    0.33870223840027013917    1.08546976203441358777
    1.51371216702628408335    1.66832459961816637062    1.80667866090861228834
    1.89336338353435951198    1.91448622168842397251    1.97201785537812468441
    2.04306084993125658400    2.09313602387015951223    2.13117505887408276166
    2.16769439166981170430    2.20662677672364315740    2.24504781574611511630
    2.28157818983265059387    2.31971472529811517660    2.35590760276760269321
    2.38995651536542164095    2.42326651583760321884    2.45773800078783954959
    2.48942264905330556957    2.52118141536514261958    2.55214438824605815981
    2.57913726291660738710    2.60717774658142742439    2.63198287710821254848
    2.65866836490095659329    2.68228778490192620865    2.70793170879867606260
    2.73114461968925059665    2.75168052130550000456    2.77233647211504674601
    2.79205038760598167613    2.81140730586357268095    2.83032962110902319708
    2.84629883402569872075    2.86183010574947305216    2.87887052255707676096
    2.89094037834704886691    2.90569690736291308397    2.91739193301171706452
    2.92767866353370775911    2.94100719601768556544    2.95520648731837987100
```

```
2.9658914817953552223   2.9741909365666061582   2.9836389157037102037
2.9893347695163767774   2.9966133294577028678   3.0031553961751384918
3.0087199302688687474   3.0142770783361045872   3.0217144531909232441
3.0215282909454512783   3.0236495229237876358   3.0241147185389354312
3.0257160049727929163   3.0258371688536525034   3.0268226747115325459
3.0266829742259071522   3.0208010559367846070   3.0213013227066203292
3.0171074824814789749   3.0152567442639965911   3.0077465863829186965
3.0052445361557413861   2.9989561970566480653   2.9938719205206782980
2.9852469028948211970   2.9751493342557311372   2.9636606436500172456
2.9574206514654854949   2.9442588116836483358   2.9314466191806118899
2.9266831331660827508   2.9353656596772226272   2.9416613361778689572
2.9624677455417458560   2.9839236189982920599   3.0023639418994423344
2.9996086733191882345   2.9884490845285456295   2.9682489240899712079
2.9281160199773221819   2.8828856593105780525   2.8334569344730997642
2.7844598594783303013   2.7361738300967646253   2.6979466770801505681
2.6575678195289831329   2.6178683979040444640   2.5808281047389942664
2.5494413817170884506   2.5197209256000316912   2.4895148654221368467
2.4573518532685123538   2.4231307001603425987   2.3892111175459742611
2.3552799598376186196   2.3193955664189873644   2.2814366883489651805
2.2432551527852280237   2.2054355830175196295   2.1666922878294623267
2.1300603073291957123   2.0974715448889695856   2.0666598306929300221
2.0491433303943891175   2.0599779034203602456   2.1357451160750535734
2.0661549403636905175   1.8517149986547258855   1.5494647579709885931
1.0873919556990800927   0.3394519305763674110   0.0270940931852073109
0.0015219978504873346  -0.0000983675612951995  -0.0002471335199602000
-0.000403958067957508  -0.0000416596336772553  -0.0004781206038637653
-0.000560500999008892  -0.0003020097374025681   0.0000099751408563083
-0.000505555607006095   0.0001587169458207161   0.0003775823385180326
-0.000202311329168005   0.0002001410745489459   0.0002224941814650473
 0.000807753282472959   0.0000797230936591600  -0.0000016802407837963
 0.000307140875582501   0.0007672140327165831   0.0006617482927674499
```

A.4 Star pattern and pseudo projection data

In this example we illustrate the capability of SNARK14 to calculate pseudo projection data. The test phantom consists of six equally spaced sectors resembling a star test pattern. Pseudo data is taken for twelve views and is reconstructed using ART. The reconstruction is contoured, reproducing the test phantom perfectly.

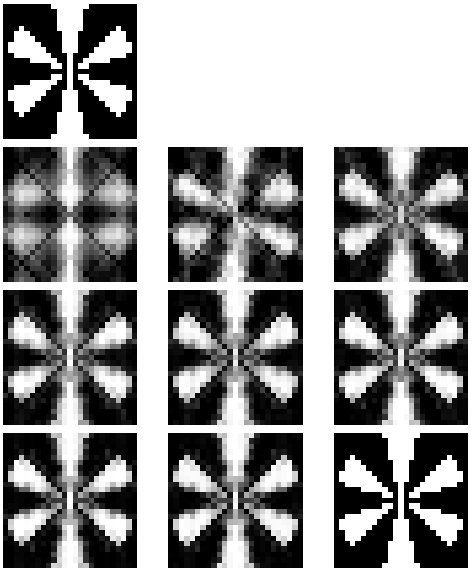

Figure A.3: Half-tone displays from Example A.4. The phantom is on top and the reconstruction at each iteration below.

A.4.1 SNARK14 INPUT file

```
********EXAMPLE 4
* STAR PATTERN WITH PSEUDO PROJECTION DATA
* RECONSTRUCTION BY ART
*
CREATE
EXAMPLE 4 STAR PATTERN
SPECTRUM MONOCHROMATIC 10
OBJECTS
SECT    0.0   -24.0    6.0    24.0    0.0    1.0
SECT   21.0   -12.0    6.0    24.0   60.0    1.0
SECT   21.0    12.0    6.0    24.0  120.0    1.0
SECT    0.0    24.0    6.0    24.0  180.0    1.0
SECT  -21.0    12.0    6.0    24.0  240.0    1.0
SECT  -21.0   -12.0    6.0    24.0  300.0    1.0
LAST 1.0
PHANTOM AVERAGE 1
25 PIXELS OF SIZE 2.0
RAYSUM
*
PICTURE TEST
*
PROJECTION PSEUDO
EXAMPLE 4 PSEUDO PROJECTION DATA FOR TWELVE VIEWS
GEOMETRY
PARALLEL VARIABLE STRIP
RAYS PROGRAM 25 2.0 DETECTOR SPACING AT 2.0
ANGLES 12 EQUALLY SPACED
0.0 165.0
MEASUREMENT PERFECT
BACKGROUND 50.0
```

```
*
MODE LOWER CONSTRAINT IS SET TO 0.0 UPPER CONSTRAINT IS 1.0
*
SELECT SNARK RAYSEQ
STEP 3 1
*
STOP ITERATION 9
*
EXECUTE ART CONTOUR
EXAMPLE 4 ART RECONSTRUCTION
0.5 0.0 1.0
1
ART3
CONSTRAINT BOUND STEPS 204
*
END
```

A.4.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


     <*> *******EXAMPLE 4

     <*>  STAR PATTERN WITH PSEUDO PROJECTION DATA

     <*>  RECONSTRUCTION BY ART

     <*>

     <#> CREATE

         EXAMPLE 4 STAR PATTERN


     <#> SPECTRUM MONOCHROMATIC 10
         energy spectrum is monochromatic at energy level    10


     <#> OBJECTS
         description of objects
                                                                    density at levels
 numb type  x-coord  y-coord x-length y-length    angle  av dens        10

    1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

    2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

    3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

    4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

    5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000
```

```
   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000

    <#> PHANTOM AVERAGE 1

        this run will generate a phantom
        density in each pixel is obtained as the average of 1 x 1 points


    <#> 25 PIXELS OF SIZE 2.0
        picture size 25 x 25,  pixel size     2.0000


    <#> RAYSUM

             0.001 seconds phantom creation
             0.001 seconds used for processing command crea


    <*>

    <#> PICTURE TEST

        EXAMPLE 4 STAR PATTERN


    <#> spec    mono   10
        energy spectrum is monochromatic at energy level    10


    <#> obje
        description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

   3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

   4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

   5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

        scale factor multiplying object densities     1.0000
```

```
    seed set to 0
    inhomogeneity set to     0.0000

<#> phan    aver    1

    density in each pixel is obtained as the average of 1 x 1 points


<#> pixe       25    size        2.0000
    picture size 25 x 25,  pixel size     2.0000

    test picture read
    EXAMPLE 4 STAR PATTERN
         0.000 seconds used for processing command pict


<*>

<#> PROJECTION PSEUDO

    EXAMPLE 4 PSEUDO PROJECTION DATA FOR TWELVE VIEWS



<#> GEOMETRY


<#> PARALLEL VARIABLE STRIP
    rays are parallel with variable spacing between rays
    data collected along strips



<#> RAYS PROGRAM 25 2.0 DETECTOR SPACING AT 2.0
    number of rays per projection    53
    snark computed number of rays    53
    at detector spacing     2.0000


<#> ANGLES 12 EQUALLY SPACED
    total number of projections    12

    projection angles    0.0000   15.0000   30.0000   45.0000   60.0000   75.0000   90.0000  1
                       150.0000  165.0000


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND 50.0
                           at levels
                                  10
     background absorption  50.0000
```

```
    estimate of totlen =     16985.417500
    estimate of totden =      6066.137812
    estimate of average density =     0.3571
    projection data read
    EXAMPLE 4 PSEUDO PROJECTION DATA FOR TWELVE VIEWS
         0.001 seconds used for processing command proj


<*>

<#> MODE LOWER CONSTRAINT IS SET TO 0.0 UPPER CONSTRAINT IS 1.0
 lower constraint set to    0.0000
 upper constraint set to    1.0000
         0.000 seconds used for processing command mode


<*>

<#> SELECT SNARK RAYSEQ

<#> STEP 3 1
    sequential ray selection with ray-step     1 and projection-step     3
         0.000 seconds used for processing command sele


<*>

<#> STOP ITERATION 9
       9 iterations
         0.000 seconds used for processing command stop


<*>

<#> EXECUTE ART CONTOUR

    EXAMPLE 4 ART RECONSTRUCTION

<#> 0.5 0.0 1.0
    reconstruction is contoured after

    last iteration

    threshold =   0.50000      w1 =   0.00000      w2 =   1.00000      w3 =   0.00000

<#> ART3
     art3 method
     relaxation parameter is 1.0
     norm is 2
     tolerance is 0.0

<#> CONSTRAINT BOUND STEPS 204
     constraint set to boundaries
```

```
 picture is not normalized
   204 rays are used for each iteration
 lower constraint =    0.0000
 upper constraint =    1.0000
algorithm executed in iteration    1
     0.000 seconds for the execution of the algorithm
iteration    1 completed
     0.000 seconds for this iteration
algorithm executed in iteration    2
     0.000 seconds for the execution of the algorithm
iteration    2 completed
     0.000 seconds for this iteration
algorithm executed in iteration    3
     0.000 seconds for the execution of the algorithm
iteration    3 completed
     0.000 seconds for this iteration
algorithm executed in iteration    4
     0.000 seconds for the execution of the algorithm
iteration    4 completed
     0.000 seconds for this iteration
algorithm executed in iteration    5
     0.000 seconds for the execution of the algorithm
iteration    5 completed
     0.000 seconds for this iteration
algorithm executed in iteration    6
     0.000 seconds for the execution of the algorithm
iteration    6 completed
     0.000 seconds for this iteration
algorithm executed in iteration    7
     0.000 seconds for the execution of the algorithm
iteration    7 completed
     0.000 seconds for this iteration
algorithm executed in iteration    8
     0.000 seconds for the execution of the algorithm
iteration    8 completed
     0.000 seconds for this iteration
algorithm executed in iteration    9
     0.000 seconds for the execution of the algorithm
reconstruction completed after iteration    9
     0.000 seconds for this iteration
     0.001 seconds for all iterations
     0.002 seconds used for processing command exec


<*>

<#> END
```

A.5 Star pattern with parallel projections

In this example we illustrate various algorithms that are appropriate for parallel projection data. The test phantom is the same as the one described in Section A.4. Parallel strip data is taken for 24 parallel strip projections between 0° and 172.5°. Reconstructions are done using BACKPROJECTION, CONVOLUTION,

RFL, FOURIER, ART, MART, QUADRATIC, and SIRT. In each case the parameters have been set either at their default values or at simple values that are not likely to be optimal for the projection data in question. EVALUATE is used in the analysis phase.

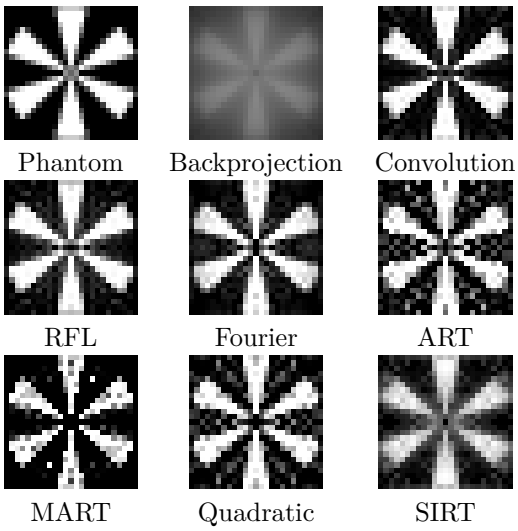

Figure A.4: Half-tone displays from Example A.5. The phantom and the final reconstruction with each method is shown.

A.5.1 SNARK14 INPUT file

```
*EXAMPLE 5
*STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
*RECONSTRUCTION BY ALL APPROPRIATE ALGORITHMS WITH DEFAULT
*OR EASY OPTIONS.
*
CREATE
EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
SPECTRUM MONOCHROMATIC 10
OBJECTS
SECT    0.0   -24.0    6.0    24.0    0.0    1.0
SECT   21.0   -12.0    6.0    24.0   60.0    1.0
SECT   21.0    12.0    6.0    24.0  120.0    1.0
SECT    0.0    24.0    6.0    24.0  180.0    1.0
SECT  -21.0    12.0    6.0    24.0  240.0    1.0
SECT  -21.0   -12.0    6.0    24.0  300.0    1.0
LAST 1.0
PHANTOM AVERAGE 5
25 PIXELS OF SIZE 2.0
RAYSUM AVERAGE 1
1
GEOMETRY
PARALLEL UNIFORM STRIP
RAYS USER 25 DETECTOR SPACING 2.0
ANGLES 24 EQUAL SPACING
0.0 172.5
MEASUREMENT PERFECT
BACKGROUND 0.0
RUN
```

```
*
PICTURE TEST
*
PROJECTION REAL
*
EXECUTE BACKPROJECTION
EXAMPLE 5A CONTINUOUS BACKPROJECTION WITH LINEAR INTERPOLATION
CONTINUOUS 2 MULTIPLICATIVE
*
EXECUTE CONVOLUTION
EXAMPLE 5B CONVOLUTION WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
BANDLIMITING 1.0 2
*
EXECUTE RFL
EXAMPLE 5C RHO FILTERED LAYERGRAM WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
BANDLIMITING 1.0 2
*
EXECUTE FOURIER
EXAMPLE 5D FOURIER WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
BANDLIMITING 1.0 3
*
STOP TERMINATION VARIANCE = 0.1
*
EXECUTE ART
EXAMPLE 5F ART WITH NO TRICKS
ART3
CONSTRAINT ART2 STEPS 888
*
EXECUTE MART
EXAMPLE 5G MULTIPLICATIVE ART A LA LENT
METHOD LENT 888 1.0 0.0
*
EXECUTE QUADRATIC
EXAMPLE 5H OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE
3  3  1  0  1  0.1  0.0  0.0
1  1  1  0  0  0    1.0  0.1 0
*
EXECUTE SIRT
EXAMPLE 5I GENERALIZED SIMULTANEOUS ITERATIVE RECONSTRUCTION TECHNOLOGY
METHOD GSIRT
*
EVALUATE
EXAMPLE 5 RECONSTRUCTION OF STAR PATTERN FROM PARALLEL PROJECTIONS
WHOLEPIC
111111111111
*
END
```

A.5.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM
```

```
     <*> EXAMPLE 5

     <*> STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS

     <*> RECONSTRUCTION BY ALL APPROPRIATE ALGORITHMS WITH DEFAULT

     <*> OR EASY OPTIONS.

     <*>

     <#> CREATE

         EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS


     <#> SPECTRUM MONOCHROMATIC 10
         energy spectrum is monochromatic at energy level    10


     <#> OBJECTS
         description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

   3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

   4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

   5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

         scale factor multiplying object densities     1.0000

         seed set to 0
         inhomogeneity set to     0.0000

     <#> PHANTOM AVERAGE 5

         this run will generate a phantom
         density in each pixel is obtained as the average of 5 x 5 points


     <#> 25 PIXELS OF SIZE 2.0
         picture size 25 x 25,  pixel size     2.0000


     <#> RAYSUM AVERAGE 1

         this run will generate projection data
```

```
    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> GEOMETRY


<#> PARALLEL UNIFORM STRIP
    rays are parallel with uniform spacing between rays
    data collected along strips



<#> RAYS USER 25 DETECTOR SPACING 2.0
    number of rays per projection    25
    at detector spacing     2.0000


<#> ANGLES 24 EQUAL SPACING
    total number of projections    24

    projection angles    0.0000    7.5000   15.0000   22.5000   30.0000   37.5000   45.0000
                        75.0000   82.5000   90.0000   97.5000  105.0000  112.5000  120.0000  1
                       150.0000  157.5000  165.0000  172.5000


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND 0.0
                           at levels
                                  10
     background absorption   0.0000


<#> RUN
         0.000 seconds phantom creation
         0.002 seconds projection data creation
         0.002 seconds used for processing command crea


<*>

<#> PICTURE TEST

    EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS


<#> spec    mono   10
    energy spectrum is monochromatic at energy level    10
```

```
 <#> obje
     description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

   3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

   4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

   5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

     scale factor multiplying object densities     1.0000

     seed set to 0
     inhomogeneity set to     0.0000

 <#> phan    aver    5

     density in each pixel is obtained as the average of 5 x 5 points


 <#> pixe       25    size        2.0000
     picture size 25 x 25,  pixel size     2.0000

     test picture read
     EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
          0.001 seconds used for processing command pict


 <*>

 <#> PROJECTION REAL

     EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS


 <#> spec    mono   10
     energy spectrum is monochromatic at energy level    10


 <#> obje
     description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000
```

```
3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

     scale factor multiplying object densities     1.0000

     seed set to 0
     inhomogeneity set to     0.0000

 <#> rays    aver    1

     projection data are calculated by dividing each ray interval into 1 substrips

     with aperture (substrip) weights     1


 <#> geom


 <#> para unif stri
     rays are parallel with uniform spacing between rays
     data collected along strips



 <#> rays    user       25    spacing        2.0000
     number of rays per projection    25
     snark computed number of rays    37
     at detector spacing     2.0000


 <#> angl       24
     total number of projections    24


     projection angles    0.0000    7.5000   15.0000   22.5000   30.0000   37.5000   45.0000
                          75.0000   82.5000   90.0000   97.5000  105.0000  112.5000  120.0000  1
                         150.0000  157.5000  165.0000  172.5000


 <#> meas    perf
     projection data are noiseless


 <#> back      0.0000
                            at levels
                                   10
      background absorption   0.0000
```

```
    estimate of totlen =     28263.966195
    estimate of totden =     10749.116539
    estimate of average density =     0.3803
    projection data read
    EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
         0.001 seconds used for processing command proj


<*>

<#> EXECUTE BACKPROJECTION

    EXAMPLE 5A CONTINUOUS BACKPROJECTION WITH LINEAR INTERPOLATION

<#> CONTINUOUS 2 MULTIPLICATIVE
     continuous back-projection
      2 order lagrange interpolation
     multiplicative normalization
    algorithm executed in iteration    1
         0.000 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.000 seconds for this iteration
         0.001 seconds used for processing command exec


<*>

<#> EXECUTE CONVOLUTION

    EXAMPLE 5B CONVOLUTION WITH BANDLIMITING FILTER & LINEAR INTERPOLATION

<#> BANDLIMITING 1.0 2
     convolution reconstruction algorithm
     using filter band
     cutoff (or alpha) =     1.0000
     2 point lagrange interpolation
    algorithm executed in iteration    1
         0.000 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.000 seconds for this iteration
         0.000 seconds used for processing command exec


<*>

<#> EXECUTE RFL

    EXAMPLE 5C RHO FILTERED LAYERGRAM WITH BANDLIMITING FILTER & LINEAR INTERPOLATIO

<#> BANDLIMITING 1.0 2
      rho-filtered layergram reconstruction algorithm
      using filter band
      cutoff (or alpha) =    1.0000
     2 point lagrange interpolation
```

```
           backprojection is performed on a    26 x    26 region
         algorithm executed in iteration    1
              0.000 seconds for the execution of the algorithm
         reconstruction completed after iteration    1
              0.000 seconds for this iteration
              0.002 seconds used for processing command exec


     <*>

     <#> EXECUTE FOURIER

         EXAMPLE 5D FOURIER WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
f o u r i e r  r e c o n s t r u c t i o n

strip projections

     <#> BANDLIMITING 1.0 3
filter=    0
cutoff=     1.00
interp=      3
nsize1=      38
nsize2=     26
time for projection transforms :        0.000  seconds
time for interpolations :               0.000  seconds
time for backtransform :                0.000  seconds
         algorithm executed in iteration    1
              0.001 seconds for the execution of the algorithm
         reconstruction completed after iteration    1
              0.001 seconds for this iteration
              0.002 seconds used for processing command exec


     <*>

     <#> STOP TERMINATION VARIANCE = 0.1
         termination test vari
         epsilon =   0.100000
              0.000 seconds used for processing command stop


     <*>

     <#> EXECUTE ART

         EXAMPLE 5F ART WITH NO TRICKS

     <#> ART3
          art3 method
          relaxation parameter is 1.0
          norm is 2
          tolerance is 0.0

     <#> CONSTRAINT ART2 STEPS 888
```

```
 constraint art2 way
 picture is not normalized
   888 rays are used for each iteration
algorithm executed in iteration    1
     0.001 seconds for the execution of the algorithm
iteration    1 completed
     0.001 seconds for this iteration
algorithm executed in iteration    2
     0.001 seconds for the execution of the algorithm
iteration    2 completed
     0.001 seconds for this iteration
algorithm executed in iteration    3
     0.001 seconds for the execution of the algorithm
 iterative process stops at iteration     3
 the change in variance is less than   0.006869 of the variance
reconstruction completed after iteration    3
     0.001 seconds for this iteration
     0.003 seconds for all iterations
     0.004 seconds used for processing command exec


<*>

<#> EXECUTE MART

    EXAMPLE 5G MULTIPLICATIVE ART A LA LENT

<#> METHOD LENT 888 1.0 0.0
 multiplicative art version lent
 no of rays used in each iteration        888
 underrelaxation factor    1.00000
 all projection data with values less than     1.0000000000e-20 are ignored
 reconstruction not normalized
 entropy functional not calculated
algorithm executed in iteration    1
     0.002 seconds for the execution of the algorithm
iteration    1 completed
     0.002 seconds for this iteration
algorithm executed in iteration    2
     0.001 seconds for the execution of the algorithm
 iterative process stops at iteration     2
 the change in variance is less than   0.009779 of the variance
reconstruction completed after iteration    2
     0.001 seconds for this iteration
     0.003 seconds for all iterations
     0.004 seconds used for processing command exec


<*>

<#> EXECUTE QUADRATIC

    EXAMPLE 5H OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE
```

```
<#> 3  3  1  0  1  0.1  0.0  0.0

<#> 1  1  1  0  0  0    1.0  0.1 0
      the conjugate gradient algorithm

            direct gradient computation
     ================================
    algorithm executed in iteration    1
         0.002 seconds for the execution of the algorithm
    iteration    1 completed
         0.002 seconds for this iteration
    algorithm executed in iteration    2
         0.001 seconds for the execution of the algorithm
    iteration    2 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    3
         0.001 seconds for the execution of the algorithm
    iteration    3 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    4
         0.001 seconds for the execution of the algorithm
    iteration    4 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    5
         0.001 seconds for the execution of the algorithm
     iterative process stops at iteration     5
     the change in variance is less than   0.095803 of the variance
    reconstruction completed after iteration    5
         0.001 seconds for this iteration
         0.006 seconds for all iterations
         0.007 seconds used for processing command exec


<*>

<#> EXECUTE SIRT

    EXAMPLE 5I GENERALIZED SIMULTANEOUS ITERATIVE RECONSTRUCTION TECHNOLOGY
     user rays selected

<#> METHOD GSIRT
     gsirt method
     sigma = inverse of relaxation =     1.0000

     this run is unconstrained sirt
     this run is unnormalized sirt
     starting picture is as in execute command
    algorithm executed in iteration    1
         0.002 seconds for the execution of the algorithm
    iteration    1 completed
         0.002 seconds for this iteration
    algorithm executed in iteration    2
         0.000 seconds for the execution of the algorithm
    iteration    2 completed
```

```
     0.000 seconds for this iteration
algorithm executed in iteration    3
     0.000 seconds for the execution of the algorithm
iteration    3 completed
     0.000 seconds for this iteration
algorithm executed in iteration    4
     0.000 seconds for the execution of the algorithm
iteration    4 completed
     0.000 seconds for this iteration
algorithm executed in iteration    5
     0.000 seconds for the execution of the algorithm
iteration    5 completed
     0.000 seconds for this iteration
algorithm executed in iteration    6
     0.000 seconds for the execution of the algorithm
iteration    6 completed
     0.000 seconds for this iteration
algorithm executed in iteration    7
     0.000 seconds for the execution of the algorithm
iteration    7 completed
     0.000 seconds for this iteration
algorithm executed in iteration    8
     0.000 seconds for the execution of the algorithm
iteration    8 completed
     0.000 seconds for this iteration
algorithm executed in iteration    9
     0.000 seconds for the execution of the algorithm
iteration    9 completed
     0.000 seconds for this iteration
algorithm executed in iteration   10
     0.000 seconds for the execution of the algorithm
iteration   10 completed
     0.000 seconds for this iteration
algorithm executed in iteration   11
     0.000 seconds for the execution of the algorithm
iteration   11 completed
     0.000 seconds for this iteration
algorithm executed in iteration   12
     0.001 seconds for the execution of the algorithm
 iterative process stops at iteration    12
 the change in variance is less than   0.086811 of the variance
reconstruction completed after iteration   12
     0.001 seconds for this iteration
     0.007 seconds for all iterations
     0.008 seconds used for processing command exec


<*>

<#> EVALUATE

    EXAMPLE 5 RECONSTRUCTION OF STAR PATTERN FROM PARALLEL PROJECTIONS

<#> WHOLEPIC
```

```
Region        cx        cy         u         v        ang      t1        t2

wholepic                                                      -1e+20     1e+20

        iterations    1   2   3   4   5   6   7   8   9  10
                     11
        last iteration

             0.002 seconds used for processing command eval


    <*>

    <#> END
```

A.5.3 SNARK14 eval file

```
evaluation name: EXAMPLE 5 RECONSTRUCTION OF STAR PATTERN FROM PARALLEL PROJECTIONS

  global resolution measures

    phantom name:    EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
    projection name: EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS

    metrics for test phantom
    region    area   average  distance   rel err  variance   std dev
         0     625    0.3622                        0.1860    0.4312

    execution name: EXAMPLE 5A CONTINUOUS BACKPROJECTION WITH LINEAR INTERPOLATION
    metrics for algorithm BACK
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3803    0.8608    0.9469    0.0079     0.0889

    execution name: EXAMPLE 5B CONVOLUTION WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
    metrics for algorithm CONV
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3482    0.3830    0.3383    0.2160     0.4647

    execution name: EXAMPLE 5C RHO FILTERED LAYERGRAM WITH BANDLIMITING FILTER & LINEAR INTERPOLATI
    metrics for algorithm RFL
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3803    0.4012    0.3521    0.2308     0.4804

    execution name: EXAMPLE 5D FOURIER WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
    metrics for algorithm FOUR
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3803    0.4019    0.3398    0.2136     0.4622

    execution name: EXAMPLE 5F ART WITH NO TRICKS
    metrics for algorithm ART
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3686    0.6318    0.6148    0.2842     0.5331
```

```
   2     625    0.3645    0.6925    0.6650    0.3130     0.5595
   3     625    0.3637    0.7017    0.6784    0.3109     0.5576

execution name: EXAMPLE 5G MULTIPLICATIVE ART A LA LENT
metrics for algorithm MART
iter    area   average  distance   rel err  variance   std dev
   1     625    0.3653    0.8483    0.5747    0.3696     0.6079
   2     625    0.3653    0.8313    0.5462    0.3660     0.6050

execution name: EXAMPLE 5H OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE
metrics for algorithm QUAD
iter    area   average  distance   rel err  variance   std dev
   1     625    0.3696    0.8616    0.9410    0.0076     0.0870
   2     625    0.3509    0.3743    0.3558    0.1763     0.4199
   3     625    0.3599    0.3907    0.3692    0.2142     0.4629
   4     625    0.3570    0.4943    0.4726    0.2569     0.5068
   5     625    0.3582    0.5808    0.5616    0.2815     0.5305

execution name: EXAMPLE 5I GENERALIZED SIMULTANEOUS ITERATIVE RECONSTRUCTION TECHNOLOGY
metrics for algorithm SIRT
iter    area   average  distance   rel err  variance   std dev
   1     625    0.3788    0.8970    0.9925    0.0026     0.0505
   2     625    0.3760    0.8082    0.8900    0.0089     0.0942
   3     625    0.3738    0.7323    0.8016    0.0175     0.1323
   4     625    0.3719    0.6673    0.7245    0.0274     0.1656
   5     625    0.3704    0.6117    0.6570    0.0380     0.1949
   6     625    0.3692    0.5642    0.5986    0.0487     0.2208
   7     625    0.3681    0.5237    0.5475    0.0594     0.2437
   8     625    0.3672    0.4894    0.5051    0.0698     0.2642
   9     625    0.3665    0.4605    0.4694    0.0798     0.2825
  10     625    0.3658    0.4363    0.4385    0.0894     0.2989
  11     625    0.3653    0.4162    0.4117    0.0984     0.3137
  12     625    0.3648    0.3997    0.3889    0.1070     0.3271
point by point resolution measures

phantom name:    EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS
projection name: EXAMPLE 5 STAR PATTERN WITH 24 PARALLEL STRIP PROJECTIONS

execution name:  EXAMPLE 5A CONTINUOUS BACKPROJECTION WITH LINEAR INTERPOLATION
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
BACK       1    0.6022    0.5629    0.4543    0.0981    0.0045

execution name:  EXAMPLE 5B CONVOLUTION WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
CONV       1    1.5231    0.5371    0.1511    0.0693    0.0263

execution name:  EXAMPLE 5C RHO FILTERED LAYERGRAM WITH BANDLIMITING FILTER & LINEAR INTERPOLAT
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
 RFL       1    1.2289    0.3918    0.1144    0.0623    0.0227

execution name:  EXAMPLE 5D FOURIER WITH BANDLIMITING FILTER & LINEAR INTERPOLATION
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
FOUR       1    1.6646    0.5753    0.1553    0.0610    0.0266
```

```
execution name:  EXAMPLE 5F ART WITH NO TRICKS
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
 ART       1    1.1799    0.4325    0.1321    0.0210    0.0082
 ART       2    1.4222    0.6361    0.1312    0.0345    0.0103
 ART       3    1.2532    0.4751    0.1485    0.0135    0.0011

execution name:  EXAMPLE 5G MULTIPLICATIVE ART A LA LENT
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
MART       1    2.4792    0.5436    0.1387    0.0522    0.0179
MART       2    1.8188    0.4809    0.1742    0.0289    0.0070

execution name:  EXAMPLE 5H OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
QUAD       1    0.6207    0.5738    0.4693    0.1074    0.0067
QUAD       2    0.5146    0.2925    0.1209    0.0449    0.0084
QUAD       3    0.8517    0.4441    0.1147    0.0748    0.0204
QUAD       4    1.0666    0.4652    0.0959    0.0415    0.0095
QUAD       5    1.3043    0.6311    0.1126    0.0470    0.0124

execution name:  EXAMPLE 5I GENERALIZED SIMULTANEOUS ITERATIVE RECONSTRUCTION TECHNOLOGY
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
SIRT       1    0.5794    0.5572    0.4491    0.1236    0.0129
SIRT       2    0.5439    0.5090    0.4023    0.1040    0.0109
SIRT       3    0.5123    0.4634    0.3597    0.0885    0.0097
SIRT       4    0.4984    0.4214    0.3211    0.0763    0.0090
SIRT       5    0.4866    0.3868    0.2862    0.0666    0.0086
SIRT       6    0.4763    0.3552    0.2547    0.0588    0.0084
SIRT       7    0.4743    0.3263    0.2264    0.0525    0.0085
SIRT       8    0.4770    0.2995    0.2009    0.0473    0.0086
SIRT       9    0.4782    0.2831    0.1779    0.0431    0.0088
SIRT      10    0.5140    0.2744    0.1573    0.0395    0.0090
SIRT      11    0.5561    0.2662    0.1388    0.0366    0.0093
SIRT      12    0.5962    0.2583    0.1221    0.0341    0.0096
```

A.6 Star pattern with divergent projections

In this example we illustrate various algorithms that are appropriate for divergent projection data. The test phantom is the same as the one described in Section A.4. Divergent projection data are taken for 24 equally spaced views between 0° and 345°. Reconstructions are done using DCONV, ART, MART and QUADRATIC. In each case the parameters have been set either at their default values or at simple values that are not likely to be optimal for the projection data in question. EVALUATE is used in the analysis phase.

Figure A.5: Half-tone displays from Example A.6. The phantom and the final reconstruction with each method is shown.

A.6.1 SNARK14 INPUT file

```
*EXAMPLE 6
*STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
*RECONSTRUCTION BY ALL APPROPRIATE ALGORITHMS WITH DEFAULT
*OR EASY OPTIONS.
*
CREATE
EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
SPECTRUM MONOCHROMATIC 10
OBJECTS
SECT    0.0   -24.0    6.0    24.0    0.0    1.0
SECT   21.0   -12.0    6.0    24.0   60.0    1.0
SECT   21.0    12.0    6.0    24.0  120.0    1.0
SECT    0.0    24.0    6.0    24.0  180.0    1.0
SECT  -21.0    12.0    6.0    24.0  240.0    1.0
SECT  -21.0   -12.0    6.0    24.0  300.0    1.0
LAST 1.0
PHANTOM AVERAGE 5
25 PIXELS OF SIZE 2.0
RAYSUM AVERAGE 1
1
GEOMETRY
DIVERGENT ARC 40.0 40.0
RAYS PROGRAM 25 2.0 DETECTOR SPACING 2.0
ANGLES 24 EQUAL SPACING
0.0 345.0
MEASUREMENT PERFECT
BACKGROUND 0.0
RUN
*
PICTURE TEST
*
PROJECTION REAL
*
EXECUTE DCONV
EXAMPLE 6A DIVERGENT BEAM CONVOLUTION WITH BANDLIMITING FILTER
2 1 25 1.0 0
BANDLIMITING
*
STOP TERMINATION VARIANCE = 0.1
*
EXECUTE ART
EXAMPLE 6C ART WITH NO TRICKS
ART3
CONSTRAINT ART2
*
EXECUTE MART
EXAMPLE 6D MULTIPLICATIVE ART A LA LENT
METHOD LENT 0 1.0 0.0
*
EXECUTE QUADRATIC
EXAMPLE 6E OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE GRADIENT
3  3  1  0  1  0.1  0.0  0.0
```

```
1  1  1  0  0  0    1.0  0.1 0
*
EVALUATE
EXAMPLE 6 RECONSTRUCTION OF STAR PATTERN FROM DIVERGENT PROJECTIONS
WHOLEPIC
1111111111111
*
END
```

A.6.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


     <*> EXAMPLE 6

     <*> STAR PATTERN WITH 24 DIVERGENT PROJECTIONS

     <*> RECONSTRUCTION BY ALL APPROPRIATE ALGORITHMS WITH DEFAULT

     <*> OR EASY OPTIONS.

     <*>

     <#> CREATE

         EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS


     <#> SPECTRUM MONOCHROMATIC 10
         energy spectrum is monochromatic at energy level    10


     <#> OBJECTS
         description of objects
                                                                    density at levels
 numb type  x-coord  y-coord x-length y-length    angle  av dens        10

    1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

    2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

    3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

    4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

    5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

    6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

         scale factor multiplying object densities     1.0000

         seed set to 0
         inhomogeneity set to     0.0000
```

```
<#> PHANTOM AVERAGE 5

    this run will generate a phantom
    density in each pixel is obtained as the average of 5 x 5 points


<#> 25 PIXELS OF SIZE 2.0
    picture size 25 x 25,  pixel size     2.0000


<#> RAYSUM AVERAGE 1

    this run will generate projection data
    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> GEOMETRY


<#> DIVERGENT ARC 40.0 40.0
    rays are divergent from point sources
    source to origin distance     40.0000
    the detectors lie on an arc with source to detector distance =    40.0000


<#> RAYS PROGRAM 25 2.0 DETECTOR SPACING 2.0
    number of rays per projection    45
    at detector spacing     2.0000


<#> ANGLES 24 EQUAL SPACING
    total number of projections    24

    projection angles    0.0000   15.0000   30.0000   45.0000   60.0000   75.0000   90.0000  1
                       150.0000  165.0000  180.0000  195.0000  210.0000  225.0000  240.0000  2
                       300.0000  315.0000  330.0000  345.0000


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND 0.0
                           at levels
                                  10
     background absorption   0.0000


<#> RUN
         0.001 seconds phantom creation
         0.001 seconds projection data creation
```

```
          0.003 seconds used for processing command crea


     <*>

     <#> PICTURE TEST

         EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS


     <#> spec    mono   10
         energy spectrum is monochromatic at energy level    10


     <#> obje
         description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

   3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

   4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

   5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

         scale factor multiplying object densities     1.0000

         seed set to 0
         inhomogeneity set to     0.0000

     <#> phan    aver    5

         density in each pixel is obtained as the average of 5 x 5 points


     <#> pixe       25    size        2.0000
         picture size 25 x 25,  pixel size     2.0000

         test picture read
         EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
              0.001 seconds used for processing command pict


     <*>

     <#> PROJECTION REAL

         EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
```

```
     <#> spec    mono   10
         energy spectrum is monochromatic at energy level    10


     <#> obje
         description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 sect   0.0000 -24.0000   6.0000  24.0000   0.0000   1.0000    1.0000

   2 sect  21.0000 -12.0000   6.0000  24.0000  60.0000   1.0000    1.0000

   3 sect  21.0000  12.0000   6.0000  24.0000 120.0000   1.0000    1.0000

   4 sect   0.0000  24.0000   6.0000  24.0000 180.0000   1.0000    1.0000

   5 sect -21.0000  12.0000   6.0000  24.0000 240.0000   1.0000    1.0000

   6 sect -21.0000 -12.0000   6.0000  24.0000 300.0000   1.0000    1.0000

         scale factor multiplying object densities     1.0000

         seed set to 0
         inhomogeneity set to     0.0000

     <#> rays    aver    1

         projection data are calculated by dividing each ray interval into 1 substrips

         with aperture (substrip) weights     1


     <#> geom


     <#> dive    arc     source at   40.0000     det dist   40.0000
         rays are divergent from point sources
         source to origin distance     40.0000
         the detectors lie on an arc with source to detector distance =    40.0000


     <#> rays    user       45    spacing        2.0000
         number of rays per projection    45
         snark computed number of rays    45
         at detector spacing     2.0000


     <#> angl       24
         total number of projections    24
```

```
    projection angles    0.0000   15.0000   30.0000   45.0000   60.0000   75.0000   90.0000  1
                       150.0000  165.0000  180.0000  195.0000  210.0000  225.0000  240.0000  2
                       300.0000  315.0000  330.0000  345.0000


<#> meas    perf
    projection data are noiseless


<#> back      0.0000
                           at levels
                                  10
     background absorption   0.0000

    estimate of totlen =     32578.803295
    estimate of totden =     11270.139500
    estimate of average density =     0.3459
    projection data read
    EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
         0.001 seconds used for processing command proj


<*>

<#> EXECUTE DCONV

    EXAMPLE 6A DIVERGENT BEAM CONVOLUTION WITH BANDLIMITING FILTER

<#> 2 1 25 1.0 0
     2 point lagrange interpolation
     one out of every     1 projection(s) used for reconstruction
      25 points used on each side of convolution
     pre-smoothing weight on center ray = 1.0000
     pre-smoothing weight on side rays  = 0.0000
     accurate back-projection

<#> BANDLIMITING
     band-limiting filter
     total time for obtaining smoothed projection data was     0.000
     total time for convoluting projection data was            0.000
     total time for back-projecting was                        0.001
    algorithm executed in iteration    1
         0.001 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.001 seconds for this iteration
         0.003 seconds used for processing command exec


<*>

<#> STOP TERMINATION VARIANCE = 0.1
    termination test vari
    epsilon =   0.100000
         0.000 seconds used for processing command stop
```

```
<*>

<#> EXECUTE ART

    EXAMPLE 6C ART WITH NO TRICKS

<#> ART3
     art3 method
     relaxation parameter is 1.0
     norm is 2
     tolerance is 0.0

<#> CONSTRAINT ART2
     constraint art2 way
     relaxation constraints with cr = 1.0
     picture is not normalized
      1080 rays are used for each iteration
    algorithm executed in iteration    1
         0.001 seconds for the execution of the algorithm
    iteration    1 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    2
         0.001 seconds for the execution of the algorithm
     iterative process stops at iteration     2
     the change in variance is less than   0.041353 of the variance
    reconstruction completed after iteration    2
         0.001 seconds for this iteration
         0.001 seconds for all iterations
         0.003 seconds used for processing command exec


<*>

<#> EXECUTE MART

    EXAMPLE 6D MULTIPLICATIVE ART A LA LENT

<#> METHOD LENT 0 1.0 0.0
     multiplicative art version lent
     no of rays used in each iteration       1080
     underrelaxation factor    1.00000
     all projection data with values less than     1.0000000000e-20 are ignored
     reconstruction not normalized
     entropy functional not calculated
    algorithm executed in iteration    1
         0.000 seconds for the execution of the algorithm
    iteration    1 completed
         0.000 seconds for this iteration
    algorithm executed in iteration    2
         0.000 seconds for the execution of the algorithm
     iterative process stops at iteration     2
     the change in variance is less than   0.026757 of the variance
```

```
    reconstruction completed after iteration    2
         0.000 seconds for this iteration
         0.000 seconds for all iterations
         0.001 seconds used for processing command exec


<*>

<#> EXECUTE QUADRATIC

    EXAMPLE 6E OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE GRADIENT

<#> 3  3  1  0  1  0.1  0.0  0.0

<#> 1  1  1  0  0  0    1.0  0.1 0
      the conjugate gradient algorithm

            direct gradient computation
     ================================
    algorithm executed in iteration    1
         0.001 seconds for the execution of the algorithm
    iteration    1 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    2
         0.001 seconds for the execution of the algorithm
    iteration    2 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    3
         0.001 seconds for the execution of the algorithm
    iteration    3 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    4
         0.001 seconds for the execution of the algorithm
    iteration    4 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    5
         0.001 seconds for the execution of the algorithm
     iterative process stops at iteration     5
     the change in variance is less than   0.046725 of the variance
    reconstruction completed after iteration    5
         0.001 seconds for this iteration
         0.004 seconds for all iterations
         0.005 seconds used for processing command exec


<*>

<#> EVALUATE

    EXAMPLE 6 RECONSTRUCTION OF STAR PATTERN FROM DIVERGENT PROJECTIONS

    <#> WHOLEPIC
Region        cx        cy         u         v        ang      t1        t2
```

```
wholepic                                                    -1e+20     1e+20

        iterations    1   2   3   4   5   6   7   8   9  10
                     11  12  13
        last iteration

             0.001 seconds used for processing command eval


     <*>

     <#> END
```

A.6.3 SNARK14 eval file

```
evaluation name: EXAMPLE 6 RECONSTRUCTION OF STAR PATTERN FROM DIVERGENT PROJECTIONS

  global resolution measures

    phantom name:    EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
    projection name: EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS

    metrics for test phantom
    region    area   average  distance   rel err  variance   std dev
         0     625    0.3622                        0.1860    0.4312

    execution name: EXAMPLE 6A DIVERGENT BEAM CONVOLUTION WITH BANDLIMITING FILTER
    metrics for algorithm DCON
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.1208    4.0003    1.9817    3.2118     1.7921

    execution name: EXAMPLE 6C ART WITH NO TRICKS
    metrics for algorithm ART
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3543    0.6235    0.5861    0.2149     0.4635
       2     625    0.3525    0.5613    0.5098    0.2238     0.4730

    execution name: EXAMPLE 6D MULTIPLICATIVE ART A LA LENT
    metrics for algorithm MART
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3517    0.8027    0.5958    0.2824     0.5314
       2     625    0.3533    0.7543    0.5662    0.2749     0.5243

    execution name: EXAMPLE 6E OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE GRADIENT
    metrics for algorithm QUAD
    iter    area   average  distance   rel err  variance   std dev
       1     625    0.3681    0.8683    0.9420    0.0095     0.0972
       2     625    0.3542    0.5688    0.5600    0.1476     0.3842
       3     625    0.3584    0.5546    0.5303    0.1779     0.4217
       4     625    0.3564    0.5635    0.5279    0.1991     0.4462
       5     625    0.3577    0.5685    0.5184    0.2084     0.4565
  point by point resolution measures
```

```
phantom name:    EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS
projection name: EXAMPLE 6 STAR PATTERN WITH 24 DIVERGENT PROJECTIONS

execution name:  EXAMPLE 6A DIVERGENT BEAM CONVOLUTION WITH BANDLIMITING FILTER
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
DCON       1   16.9760    6.5672    2.0533    0.5332    0.1783

execution name:  EXAMPLE 6C ART WITH NO TRICKS
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
 ART       1    2.0575    0.7039    0.1127    0.0568    0.0294
 ART       2    2.2177    0.6297    0.1291    0.0675    0.0352

execution name:  EXAMPLE 6D MULTIPLICATIVE ART A LA LENT
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
MART       1    2.4042    0.5922    0.1526    0.0931    0.0415
MART       2    1.6089    0.4700    0.1179    0.0852    0.0424

execution name:  EXAMPLE 6E OPTIMIZATION OF LEAST SQUARES FUNCTIONAL USING CONJUGATE GRADIENT
algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
QUAD       1    0.6402    0.5899    0.4709    0.1040    0.0065
QUAD       2    1.2944    0.5751    0.1622    0.0753    0.0152
QUAD       3    1.5651    0.5930    0.1275    0.0497    0.0194
QUAD       4    1.8313    0.5988    0.1105    0.0547    0.0144
QUAD       5    2.0261    0.6589    0.1501    0.0406    0.0194
```

A.7 Simulating data collection and image reconstruction for PET

This example illustrates a SNARK14 run that simulates the collection of PET data and uses EMAP to reconstruct a test phantom. The test phantom was obtained from a computerized overlay atlas based on average anatomy of the brain [41]. In this phantom neuroanatomical structures are represented by various ellipses and rectangles at appropriate locations. The activity in the brain background is assigned 1.0 (in arbitrary units), while the activity in the neuroanatomical structures is either 1.95 or 2.0. A 95×95 digitization of this phantom is shown in Figure A.6.

To simulate data obtained from a PET system with a ring of 300 detectors and with each detector in coincidence with 101 detectors opposite it, we generate divergent projection data over 300 view angles with 101 rays per view. In this case we assume that the detectors are arranged on an arc. Poisson noise is introduced into the projection measurements and the phantom densities are scaled by a factor of 0.51 to yield a total photon count that is approximately 2,000,000 (note the actual parameter totden in the output file shown in Section A.7.2).

Figure A.6: A Brain Phantom

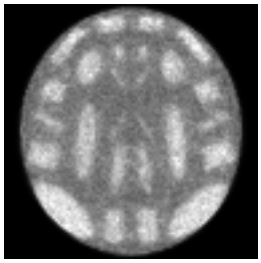

Figure A.7: MAP EM reconstruction after 20 iterations

A.7.1 SNARK14 INPUT file

```
********EXAMPLE 7
* MAP EM ALGORITHM FOR EMISSION TOMOGRAPHY.  RECONSTRUCTION OF BRAIN
* PHANTOM. SIMULATING PET GEOMETRY WITH A RING OF 300 DETECTORS WITH
* EACH DETECTOR IN COINCIDENCE WITH 101 DETECTORS OPPOSITE IT.
* COEFFICIENT OF PENALTLY TERM IS SET TO 10.0
*
CREATE
EXAMPLE 7 Brain Phantom
SPECTRUM MONOCHROMATIC 511
OBJECTS
   1 elip   -7.0   46.0   3.0   6.0   17.0  0.95  1.0
   2 elip    7.0   46.0   3.0   6.0  -17.0  1.0   0.95
   3 rect  -12.0   64.0   7.5   4.5    5.0  1.0   1.0
   4 rect   12.0   64.0   7.5   4.5   -5.0  0.95  0.95
   5 rect  -38.0   51.0   3.5  13.0  -39.0  0.95  1.0
   6 rect   38.0   51.0   3.5  13.0   39.0  1.0   0.95
   7 rect  -46.0   24.0   6.5   6.0  -18.0  0.95  1.0
   8 rect   46.0   24.0   6.5   6.0   18.0  1.0   0.95
   9 rect  -49.0    6.0   2.5  10.0   63.0  1.0   1.0
  10 rect   49.0    6.0   2.5  10.0  -63.0  0.95  0.95
  11 rect  -52.0  -14.0   9.0   7.0  -14.0  0.95  1.0
  12 rect   52.0  -14.0   9.0   7.0   14.0  1.0   0.95
  13 rect  -10.0  -56.0   5.5  10.0   -1.0  0.95  1.0
  14 rect   10.0  -56.0   5.5  10.0    1.0  1.0   0.95
  15 elip  -40.0  -47.0   9.0  22.5   48.0  1.0   1.0
  16 elip   40.0  -47.0   9.0  22.5  -48.0  0.95  0.95
  17 elip   -8.0  -22.0   3.5  15.5   -9.0  1.0   1.0
  18 elip    8.0  -22.0   3.5  15.5    9.0  0.95  0.95
  19 elip  -27.0   -6.0   5.5  23.5   -5.0  0.95  1.0
  20 elip   27.0   -6.0   5.5  23.5    5.0  1.0   0.95
  21 elip  -25.0   38.0   6.5  10.5  -14.0  1.0   1.0
  22 elip   25.0   38.0   6.5  10.5   14.0  0.95  0.95
  23 rect   -8.0   32.0   1.5   6.5   38.0  1.0   1.0
  24 rect    8.0   32.0   1.5   6.5  -38.0  0.95  0.95
  25 rect   -8.0    3.0   1.0   9.0  -33.0  0.95  1.0
  26 rect    8.0    3.0   1.0   9.0   33.0  1.0   0.95
  27 elip    0.0    0.0  66.5  74.0    0.0  1.0   1.0
LAST .51    1   0.05
PHANTOM AVERAGE 7
95 PIXELS OF SIZE 1.6
RAYSUM AVERAGE 1
```

```
1
GEOMETRY
divergent arc 153 306
RAYS USER 101 DETECTOR SPACING 3.2
ANGLES 300 EQUAL SPACING
0.0 358.8
MEASUREMENT NOISY
QUANTUM 1.0 1.0 CALIBRATION 4
SEED 0
BACKGROUND 0.0
RUN
*
PICTURE TEST
*
PROJECTION REAL
*
STOP ITERATION 20
*
EXECUTE AVERAGE EMAP
Example 7 Illustrating the MAP EM algorithm for PET
gamma is 10.0 EVAL
*
END
```

A.7.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


     <*> *******EXAMPLE 7

     <*>  MAP EM ALGORITHM FOR EMISSION TOMOGRAPHY.  RECONSTRUCTION OF BRAIN

     <*>  PHANTOM. SIMULATING PET GEOMETRY WITH A RING OF 300 DETECTORS WITH

     <*>  EACH DETECTOR IN COINCIDENCE WITH 101 DETECTORS OPPOSITE IT.

     <*>  COEFFICIENT OF PENALTLY TERM IS SET TO 10.0

     <*>

     <#> CREATE

         EXAMPLE 7 Brain Phantom


     <#> SPECTRUM MONOCHROMATIC 511
         energy spectrum is monochromatic at energy level   511


     <#> OBJECTS
         description of objects
                                                                  density at levels
```

numb	type	x-coord	y-coord	x-length	y-length	angle	av dens	511
1	elip	-7.0000	46.0000	3.0000	6.0000	17.0000	0.9500	0.9500
2	elip	7.0000	46.0000	3.0000	6.0000	-17.0000	1.0000	1.0000
3	rect	-12.0000	64.0000	7.5000	4.5000	5.0000	1.0000	1.0000
4	rect	12.0000	64.0000	7.5000	4.5000	-5.0000	0.9500	0.9500
5	rect	-38.0000	51.0000	3.5000	13.0000	-39.0000	0.9500	0.9500
6	rect	38.0000	51.0000	3.5000	13.0000	39.0000	1.0000	1.0000
7	rect	-46.0000	24.0000	6.5000	6.0000	-18.0000	0.9500	0.9500
8	rect	46.0000	24.0000	6.5000	6.0000	18.0000	1.0000	1.0000
9	rect	-49.0000	6.0000	2.5000	10.0000	63.0000	1.0000	1.0000
10	rect	49.0000	6.0000	2.5000	10.0000	-63.0000	0.9500	0.9500
11	rect	-52.0000	-14.0000	9.0000	7.0000	-14.0000	0.9500	0.9500
12	rect	52.0000	-14.0000	9.0000	7.0000	14.0000	1.0000	1.0000
13	rect	-10.0000	-56.0000	5.5000	10.0000	-1.0000	0.9500	0.9500
14	rect	10.0000	-56.0000	5.5000	10.0000	1.0000	1.0000	1.0000
15	elip	-40.0000	-47.0000	9.0000	22.5000	48.0000	1.0000	1.0000
16	elip	40.0000	-47.0000	9.0000	22.5000	-48.0000	0.9500	0.9500
17	elip	-8.0000	-22.0000	3.5000	15.5000	-9.0000	1.0000	1.0000
18	elip	8.0000	-22.0000	3.5000	15.5000	9.0000	0.9500	0.9500
19	elip	-27.0000	-6.0000	5.5000	23.5000	-5.0000	0.9500	0.9500
20	elip	27.0000	-6.0000	5.5000	23.5000	5.0000	1.0000	1.0000
21	elip	-25.0000	38.0000	6.5000	10.5000	-14.0000	1.0000	1.0000
22	elip	25.0000	38.0000	6.5000	10.5000	14.0000	0.9500	0.9500
23	rect	-8.0000	32.0000	1.5000	6.5000	38.0000	1.0000	1.0000
24	rect	8.0000	32.0000	1.5000	6.5000	-38.0000	0.9500	0.9500
25	rect	-8.0000	3.0000	1.0000	9.0000	-33.0000	0.9500	0.9500
26	rect	8.0000	3.0000	1.0000	9.0000	33.0000	1.0000	1.0000

```
27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   1.0000    1.0000

     scale factor multiplying object densities     0.5100

     seed set to 1
     inhomogeneity set to     0.0500

  <#> PHANTOM AVERAGE 7

     this run will generate a phantom
     density in each pixel is obtained as the average of 7 x 7 points


  <#> 95 PIXELS OF SIZE 1.6
     picture size 95 x 95,  pixel size     1.6000


  <#> RAYSUM AVERAGE 1

     this run will generate projection data
     projection data are calculated by dividing each ray interval into 1 substrips

     with aperture (substrip) weights     1


  <#> GEOMETRY


  <#> divergent arc 153 306
     rays are divergent from point sources
     source to origin distance    153.0000
     the detectors lie on an arc with source to detector distance =   306.0000


  <#> RAYS USER 101 DETECTOR SPACING 3.2
     number of rays per projection   101
     at detector spacing     3.2000


  <#> ANGLES 300 EQUAL SPACING
     total number of projections   300

     projection angles    0.0000    1.2000    2.4000    3.6000    4.8000    6.0000    7.2000
                         12.0000   13.2000   14.4000   15.6000   16.8000   18.0000   19.2000
                         24.0000   25.2000   26.4000   27.6000   28.8000   30.0000   31.2000
                         36.0000   37.2000   38.4000   39.6000   40.8000   42.0000   43.2000
                         48.0000   49.2000   50.4000   51.6000   52.8000   54.0000   55.2000
                         60.0000   61.2000   62.4000   63.6000   64.8000   66.0000   67.2000
                         72.0000   73.2000   74.4000   75.6000   76.8000   78.0000   79.2000
                         84.0000   85.2000   86.4000   87.6000   88.8000   90.0000   91.2000
                         96.0000   97.2000   98.4000   99.6000  100.8000  102.0000  103.2000  1
                        108.0000  109.2000  110.4000  111.6000  112.8000  114.0000  115.2000  1
                        120.0000  121.2000  122.4000  123.6000  124.8000  126.0000  127.2000  1
                        132.0000  133.2000  134.4000  135.6000  136.8000  138.0000  139.2000  1
```

```
                          144.0000  145.2000  146.4000  147.6000  148.8000  150.0000  151.2000  1
                          156.0000  157.2000  158.4000  159.6000  160.8000  162.0000  163.2000  1
                          168.0000  169.2000  170.4000  171.6000  172.8000  174.0000  175.2000  1
                          180.0000  181.2000  182.4000  183.6000  184.8000  186.0000  187.2000  1
                          192.0000  193.2000  194.4000  195.6000  196.8000  198.0000  199.2000  2
                          204.0000  205.2000  206.4000  207.6000  208.8000  210.0000  211.2000  2
                          216.0000  217.2000  218.4000  219.6000  220.8000  222.0000  223.2000  2
                          228.0000  229.2000  230.4000  231.6000  232.8000  234.0000  235.2000  2
                          240.0000  241.2000  242.4000  243.6000  244.8000  246.0000  247.2000  2
                          252.0000  253.2000  254.4000  255.6000  256.8000  258.0000  259.2000  2
                          264.0000  265.2000  266.4000  267.6000  268.8000  270.0000  271.2000  2
                          276.0000  277.2000  278.4000  279.6000  280.8000  282.0000  283.2000  2
                          288.0000  289.2000  290.4000  291.6000  292.8000  294.0000  295.2000  2
                          300.0000  301.2000  302.4000  303.6000  304.8000  306.0000  307.2000  3
                          312.0000  313.2000  314.4000  315.6000  316.8000  318.0000  319.2000  3
                          324.0000  325.2000  326.4000  327.6000  328.8000  330.0000  331.2000  3
                          336.0000  337.2000  338.4000  339.6000  340.8000  342.0000  343.2000  3
                          348.0000  349.2000  350.4000  351.6000  352.8000  354.0000  355.2000  3


<#> MEASUREMENT NOISY
    noise characteristics of projection data follow
          nature          characteristics

<#> QUANTUM 1.0 1.0 CALIBRATION 4
         Emission tomography

<#> SEED 0
     seed for random number generator is              0


<#> BACKGROUND 0.0
                           at levels
                                 511
     background absorption   0.0000


<#> RUN
         0.027 seconds phantom creation
         0.106 seconds projection data creation
         0.133 seconds used for processing command crea


<*>

<#> PICTURE TEST

    EXAMPLE 7 Brain Phantom


<#> spec    mono  511
    energy spectrum is monochromatic at energy level   511
```

```
<#> obje
    description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens       511

   1 elip  -7.0000  46.0000   3.0000   6.0000  17.0000   0.4845    0.4845

   2 elip   7.0000  46.0000   3.0000   6.0000 -17.0000   0.5100    0.5100

   3 rect -12.0000  64.0000   7.5000   4.5000   5.0000   0.5100    0.5100

   4 rect  12.0000  64.0000   7.5000   4.5000  -5.0000   0.4845    0.4845

   5 rect -38.0000  51.0000   3.5000  13.0000 -39.0000   0.4845    0.4845

   6 rect  38.0000  51.0000   3.5000  13.0000  39.0000   0.5100    0.5100

   7 rect -46.0000  24.0000   6.5000   6.0000 -18.0000   0.4845    0.4845

   8 rect  46.0000  24.0000   6.5000   6.0000  18.0000   0.5100    0.5100

   9 rect -49.0000   6.0000   2.5000  10.0000  63.0000   0.5100    0.5100

  10 rect  49.0000   6.0000   2.5000  10.0000 -63.0000   0.4845    0.4845

  11 rect -52.0000 -14.0000   9.0000   7.0000 -14.0000   0.4845    0.4845

  12 rect  52.0000 -14.0000   9.0000   7.0000  14.0000   0.5100    0.5100

  13 rect -10.0000 -56.0000   5.5000  10.0000  -1.0000   0.4845    0.4845

  14 rect  10.0000 -56.0000   5.5000  10.0000   1.0000   0.5100    0.5100

  15 elip -40.0000 -47.0000   9.0000  22.5000  48.0000   0.5100    0.5100

  16 elip  40.0000 -47.0000   9.0000  22.5000 -48.0000   0.4845    0.4845

  17 elip  -8.0000 -22.0000   3.5000  15.5000  -9.0000   0.5100    0.5100

  18 elip   8.0000 -22.0000   3.5000  15.5000   9.0000   0.4845    0.4845

  19 elip -27.0000  -6.0000   5.5000  23.5000  -5.0000   0.4845    0.4845

  20 elip  27.0000  -6.0000   5.5000  23.5000   5.0000   0.5100    0.5100

  21 elip -25.0000  38.0000   6.5000  10.5000 -14.0000   0.5100    0.5100

  22 elip  25.0000  38.0000   6.5000  10.5000  14.0000   0.4845    0.4845

  23 rect  -8.0000  32.0000   1.5000   6.5000  38.0000   0.5100    0.5100

  24 rect   8.0000  32.0000   1.5000   6.5000 -38.0000   0.4845    0.4845

  25 rect  -8.0000   3.0000   1.0000   9.0000 -33.0000   0.4845    0.4845
```

```
  26 rect   8.0000   3.0000   1.0000   9.0000  33.0000   0.5100    0.5100

  27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   0.5100    0.5100

       scale factor multiplying object densities     0.5100

       seed set to 1
       inhomogeneity set to     0.0500

    <#> phan    aver    7

       density in each pixel is obtained as the average of 7 x 7 points


    <#> pixe       95    size        1.6000
       picture size 95 x 95,  pixel size     1.6000

       test picture read
       EXAMPLE 7 Brain Phantom
            0.003 seconds used for processing command pict


    <*>

    <#> PROJECTION REAL

       EXAMPLE 7 Brain Phantom


    <#> spec    mono  511
       energy spectrum is monochromatic at energy level   511


    <#> obje
       description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens       511

   1 elip  -7.0000  46.0000   3.0000   6.0000  17.0000   0.4845    0.4845

   2 elip   7.0000  46.0000   3.0000   6.0000 -17.0000   0.5100    0.5100

   3 rect -12.0000  64.0000   7.5000   4.5000   5.0000   0.5100    0.5100

   4 rect  12.0000  64.0000   7.5000   4.5000  -5.0000   0.4845    0.4845

   5 rect -38.0000  51.0000   3.5000  13.0000 -39.0000   0.4845    0.4845

   6 rect  38.0000  51.0000   3.5000  13.0000  39.0000   0.5100    0.5100

   7 rect -46.0000  24.0000   6.5000   6.0000 -18.0000   0.4845    0.4845

   8 rect  46.0000  24.0000   6.5000   6.0000  18.0000   0.5100    0.5100
```

```
 9 rect -49.0000   6.0000   2.5000  10.0000  63.0000   0.5100    0.5100

10 rect  49.0000   6.0000   2.5000  10.0000 -63.0000   0.4845    0.4845

11 rect -52.0000 -14.0000   9.0000   7.0000 -14.0000   0.4845    0.4845

12 rect  52.0000 -14.0000   9.0000   7.0000  14.0000   0.5100    0.5100

13 rect -10.0000 -56.0000   5.5000  10.0000  -1.0000   0.4845    0.4845

14 rect  10.0000 -56.0000   5.5000  10.0000   1.0000   0.5100    0.5100

15 elip -40.0000 -47.0000   9.0000  22.5000  48.0000   0.5100    0.5100

16 elip  40.0000 -47.0000   9.0000  22.5000 -48.0000   0.4845    0.4845

17 elip  -8.0000 -22.0000   3.5000  15.5000  -9.0000   0.5100    0.5100

18 elip   8.0000 -22.0000   3.5000  15.5000   9.0000   0.4845    0.4845

19 elip -27.0000  -6.0000   5.5000  23.5000  -5.0000   0.4845    0.4845

20 elip  27.0000  -6.0000   5.5000  23.5000   5.0000   0.5100    0.5100

21 elip -25.0000  38.0000   6.5000  10.5000 -14.0000   0.5100    0.5100

22 elip  25.0000  38.0000   6.5000  10.5000  14.0000   0.4845    0.4845

23 rect  -8.0000  32.0000   1.5000   6.5000  38.0000   0.5100    0.5100

24 rect   8.0000  32.0000   1.5000   6.5000 -38.0000   0.4845    0.4845

25 rect  -8.0000   3.0000   1.0000   9.0000 -33.0000   0.4845    0.4845

26 rect   8.0000   3.0000   1.0000   9.0000  33.0000   0.5100    0.5100

27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   0.5100    0.5100

     scale factor multiplying object densities     0.5100

     seed set to 1
     inhomogeneity set to     0.0500

  <#> rays    aver    1

     projection data are calculated by dividing each ray interval into 1 substrips

     with aperture (substrip) weights     1


  <#> geom
```

```
<#> dive    arc     source at  153.0000     det dist  306.0000
    rays are divergent from point sources
    source to origin distance    153.0000
    the detectors lie on an arc with source to detector distance =   306.0000


<#> rays    user      101    spacing        3.2000
    number of rays per projection   101
    snark computed number of rays   151
    at detector spacing     3.2000


<#> angl      300
    total number of projections   300


    projection angles    0.0000    1.2000    2.4000    3.6000    4.8000    6.0000    7.2000
                        12.0000   13.2000   14.4000   15.6000   16.8000   18.0000   19.2000
                        24.0000   25.2000   26.4000   27.6000   28.8000   30.0000   31.2000
                        36.0000   37.2000   38.4000   39.6000   40.8000   42.0000   43.2000
                        48.0000   49.2000   50.4000   51.6000   52.8000   54.0000   55.2000
                        60.0000   61.2000   62.4000   63.6000   64.8000   66.0000   67.2000
                        72.0000   73.2000   74.4000   75.6000   76.8000   78.0000   79.2000
                        84.0000   85.2000   86.4000   87.6000   88.8000   90.0000   91.2000
                        96.0000   97.2000   98.4000   99.6000  100.8000  102.0000  103.2000  1
                       108.0000  109.2000  110.4000  111.6000  112.8000  114.0000  115.2000  1
                       120.0000  121.2000  122.4000  123.6000  124.8000  126.0000  127.2000  1
                       132.0000  133.2000  134.4000  135.6000  136.8000  138.0000  139.2000  1
                       144.0000  145.2000  146.4000  147.6000  148.8000  150.0000  151.2000  1
                       156.0000  157.2000  158.4000  159.6000  160.8000  162.0000  163.2000  1
                       168.0000  169.2000  170.4000  171.6000  172.8000  174.0000  175.2000  1
                       180.0000  181.2000  182.4000  183.6000  184.8000  186.0000  187.2000  1
                       192.0000  193.2000  194.4000  195.6000  196.8000  198.0000  199.2000  2
                       204.0000  205.2000  206.4000  207.6000  208.8000  210.0000  211.2000  2
                       216.0000  217.2000  218.4000  219.6000  220.8000  222.0000  223.2000  2
                       228.0000  229.2000  230.4000  231.6000  232.8000  234.0000  235.2000  2
                       240.0000  241.2000  242.4000  243.6000  244.8000  246.0000  247.2000  2
                       252.0000  253.2000  254.4000  255.6000  256.8000  258.0000  259.2000  2
                       264.0000  265.2000  266.4000  267.6000  268.8000  270.0000  271.2000  2
                       276.0000  277.2000  278.4000  279.6000  280.8000  282.0000  283.2000  2
                       288.0000  289.2000  290.4000  291.6000  292.8000  294.0000  295.2000  2
                       300.0000  301.2000  302.4000  303.6000  304.8000  306.0000  307.2000  3
                       312.0000  313.2000  314.4000  315.6000  316.8000  318.0000  319.2000  3
                       324.0000  325.2000  326.4000  327.6000  328.8000  330.0000  331.2000  3
                       336.0000  337.2000  338.4000  339.6000  340.8000  342.0000  343.2000  3
                       348.0000  349.2000  350.4000  351.6000  352.8000  354.0000  355.2000  3


<#> meas    nois
    noise characteristics of projection data follow
          nature          characteristics

<#> quan            1.0000            1.0000    cali  4
         Emission tomography
```

```
   <#> seed       0
        seed for random number generator is              0


   <#> back      0.0000
                             at levels
                                   511
        background absorption   0.0000

       estimate of totlen =   4263169.769963
       estimate of totden =   2020362.000000
       estimate of average density =     0.4739
       projection data read
       EXAMPLE 7 Brain Phantom
            0.012 seconds used for processing command proj


   <*>

   <#> STOP ITERATION 20
         20 iterations
            0.000 seconds used for processing command stop


   <*>

   <#> EXECUTE AVERAGE EMAP

       Example 7 Illustrating the MAP EM algorithm for PET

   <#> gamma is 10.0 EVAL

------------------------------------------------------------

 maximum a-posteriori probability expectation maximization

          gamma:  10.000
          evaluation flag is set

------------------------------------------------------------

       algorithm executed in iteration    1
            0.162 seconds for the execution of the algorithm
       iteration    1 completed
            0.162 seconds for this iteration
       algorithm executed in iteration    2
            0.108 seconds for the execution of the algorithm
       iteration    2 completed
            0.108 seconds for this iteration
       algorithm executed in iteration    3
            0.104 seconds for the execution of the algorithm
       iteration    3 completed
            0.104 seconds for this iteration
```

```
algorithm executed in iteration    4
     0.095 seconds for the execution of the algorithm
iteration    4 completed
     0.095 seconds for this iteration
algorithm executed in iteration    5
     0.095 seconds for the execution of the algorithm
iteration    5 completed
     0.095 seconds for this iteration
algorithm executed in iteration    6
     0.095 seconds for the execution of the algorithm
iteration    6 completed
     0.095 seconds for this iteration
algorithm executed in iteration    7
     0.095 seconds for the execution of the algorithm
iteration    7 completed
     0.095 seconds for this iteration
algorithm executed in iteration    8
     0.095 seconds for the execution of the algorithm
iteration    8 completed
     0.095 seconds for this iteration
algorithm executed in iteration    9
     0.095 seconds for the execution of the algorithm
iteration    9 completed
     0.095 seconds for this iteration
algorithm executed in iteration   10
     0.104 seconds for the execution of the algorithm
iteration   10 completed
     0.104 seconds for this iteration
algorithm executed in iteration   11
     0.100 seconds for the execution of the algorithm
iteration   11 completed
     0.100 seconds for this iteration
algorithm executed in iteration   12
     0.105 seconds for the execution of the algorithm
iteration   12 completed
     0.105 seconds for this iteration
algorithm executed in iteration   13
     0.105 seconds for the execution of the algorithm
iteration   13 completed
     0.105 seconds for this iteration
algorithm executed in iteration   14
     0.100 seconds for the execution of the algorithm
iteration   14 completed
     0.100 seconds for this iteration
algorithm executed in iteration   15
     0.100 seconds for the execution of the algorithm
iteration   15 completed
     0.100 seconds for this iteration
algorithm executed in iteration   16
     0.100 seconds for the execution of the algorithm
iteration   16 completed
     0.100 seconds for this iteration
algorithm executed in iteration   17
     0.100 seconds for the execution of the algorithm
```

```
iteration   17 completed
     0.100 seconds for this iteration
algorithm executed in iteration   18
     0.097 seconds for the execution of the algorithm
iteration   18 completed
     0.097 seconds for this iteration
algorithm executed in iteration   19
     0.097 seconds for the execution of the algorithm
iteration   19 completed
     0.097 seconds for this iteration
algorithm executed in iteration   20
     0.095 seconds for the execution of the algorithm
reconstruction completed after iteration   20
     0.095 seconds for this iteration
     2.048 seconds for all iterations
     2.101 seconds used for processing command exec


<*>

<#> END
```

A.7.3 SNARK14 MAPUser1 file

iter	log-likelihood	log-posterior
1	6665320.255000772	6665319.804143723
2	6699592.814686690	6699591.201264398
3	6721050.941879908	6721047.434363294
4	6735133.991341705	6735127.825055615
5	6744784.011184918	6744774.423563549
6	6751612.777032703	6751599.033013058
7	6756561.325021296	6756542.732425543
8	6760215.064699405	6760190.983431549
9	6762956.169333679	6762926.016748638
10	6765043.181165983	6765006.434707592
11	6766654.234460152	6766610.432211500
12	6767914.654328912	6767863.394251612
13	6768913.639648434	6768854.577240884
14	6769715.020938492	6769647.866062297
15	6770365.597993632	6770290.111740321
16	6770899.665669869	6770815.656012718
17	6771342.735730905	6771250.053258957
18	6771713.974127298	6771612.507907540
19	6772027.929014099	6771917.602654658
20	6772295.791661046	6772176.559496485

A.8 Linogram reconstruction

This example illustrates the use of the linogram algorithm in reconstructing the head phantom described in [29, Section 4.3]. A 115×115 digitization of the phantom is shown in Figure A.8. (Note: in the notation in Section 7.12, N=57 for this phantom.) Parallel projection data are collected with variable ray spacing to

provide us with a total of 462 projections and 231 rays per projection. The reconstructions obtained using both the linogram and the filtered backprojection methods are shown in Figures A.9 and A.10, respectively.

Figure A.8: Head phantom

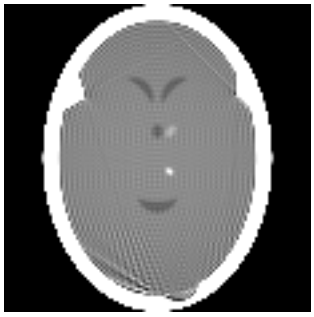

Figure A.9: Linogram reconstruction

Figure A.10: Filtered backprojection reconstruction

A.8.1 SNARK14 INPUT file

```
********* EXAMPLE 8
* RECONSTRUCTION OF HEAD PHANTOM USING LINOGRAM AND FILTERED
* BACKPROJECTION METHODS. SINC FILTER FOR LINOGRAM AND COSINE
* FILTER FOR FBP.
*
create
EXAMPLE 8 LINOGRAM RECONSTRUCTION
spectrum monochromatic 75
objects    cx          cy          u           v         ang         den
```

```
elip      0.000       0.000     8.625        6.4687     90.00      0.416
elip      0.000       0.000     7.875        5.7187     90.00     -0.206
elip      0.000       1.500     0.375        0.3000     90.00     -0.003
elip      0.675      -0.750     0.225        0.1500    140.00      0.010
elip      0.750       1.500     0.375        0.2250     50.00      0.003
segm      1.375      -7.500     1.100        0.6250     19.20     -0.204
segm      1.375      -7.500     1.100        4.3200     19.21      0.204
segm      0.000      -2.250     1.125        0.3750      0.00     -0.003
segm      0.000      -2.250     1.125        3.0000      0.00      0.003
segm     -1.000       3.750     1.000        0.5000    135.00     -0.003
segm     -1.000       3.750     1.000        3.0000    135.00      0.003
segm      1.000       3.750     1.000        0.5000    225.00     -0.003
segm      1.000       3.750     1.000        3.0000    225.00      0.003
tria      5.025       3.750     1.125        0.5000    110.75      0.206
tria     -5.025       3.750     1.125        0.9000   -110.75      0.206
last    1.0
phantom average 5
115    0.1504
raysum average 1
1
geometry
linogram
measurement perfect
background absorption is 0.0
run
picture test
projection real
execute lino
EXAMPLE 8a LINOGRAM METHOD
sinc 1.0
execute conv
EXAMPLE 8b FILTERED BACKPROJECTION
cosine 1.0 2
end
```

A.8.2 SNARK14 OUTPUT file

```
snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


  <*> ******** EXAMPLE 8

  <*>  RECONSTRUCTION OF HEAD PHANTOM USING LINOGRAM AND FILTERED

  <*>  BACKPROJECTION METHODS. SINC FILTER FOR LINOGRAM AND COSINE

  <*>  FILTER FOR FBP.

  <*>

  <#> create

      EXAMPLE 8 LINOGRAM RECONSTRUCTION
```

```
    <#> spectrum monochromatic 75
        energy spectrum is monochromatic at energy level    75


    <#> objects    cx          cy          u           v         ang         den
        description of objects
                                                                  density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 elip   0.0000   0.0000   8.6250   6.4687  90.0000   0.4160    0.4160

   2 elip   0.0000   0.0000   7.8750   5.7187  90.0000  -0.2060   -0.2060

   3 elip   0.0000   1.5000   0.3750   0.3000  90.0000  -0.0030   -0.0030

   4 elip   0.6750  -0.7500   0.2250   0.1500 140.0000   0.0100    0.0100

   5 elip   0.7500   1.5000   0.3750   0.2250  50.0000   0.0030    0.0030

   6 segm   1.3750  -7.5000   1.1000   0.6250  19.2000  -0.2040   -0.2040

   7 segm   1.3750  -7.5000   1.1000   4.3200  19.2100   0.2040    0.2040

   8 segm   0.0000  -2.2500   1.1250   0.3750   0.0000  -0.0030   -0.0030

   9 segm   0.0000  -2.2500   1.1250   3.0000   0.0000   0.0030    0.0030

  10 segm  -1.0000   3.7500   1.0000   0.5000 135.0000  -0.0030   -0.0030

  11 segm  -1.0000   3.7500   1.0000   3.0000 135.0000   0.0030    0.0030

  12 segm   1.0000   3.7500   1.0000   0.5000 225.0000  -0.0030   -0.0030

  13 segm   1.0000   3.7500   1.0000   3.0000 225.0000   0.0030    0.0030

  14 tria   5.0250   3.7500   1.1250   0.5000 110.7500   0.2060    0.2060

  15 tria  -5.0250   3.7500   1.1250   0.9000 -110.7500   0.2060    0.2060

        scale factor multiplying object densities     1.0000

        seed set to 0
        inhomogeneity set to     0.0000

    <#> phantom average 5

        this run will generate a phantom
        density in each pixel is obtained as the average of 5 x 5 points


    <#> 115    0.1504
        picture size 115 x 115,  pixel size     0.1504
```

```
<#> raysum average 1

    this run will generate projection data
    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> geometry


<#> linogram
    rays are parallel with variable spacing between rays
    data collected along lines


    number of rays per projection   231
    at detector spacing     0.1504

    total number of projections   462
    projection angles   45.1243   45.3745   45.6268   45.8814   46.1382   46.3972   46.6584
                        47.7263   47.9991   48.2742   48.5517   48.8316   49.1140   49.3987
                        50.5623   50.8595   51.1592   51.4614   51.7662   52.0735   52.3835
                        53.6495   53.9726   54.2984   54.6269   54.9581   55.2920   55.6285
                        57.0023   57.3526   57.7057   58.0616   58.4202   58.7816   59.1458
                        60.6305   61.0086   61.3895   61.7733   62.1597   62.5490   62.9410
                        64.5367   64.9424   65.3508   65.7619   66.1757   66.5922   67.0113
                        68.7136   69.1455   69.5800   70.0169   70.4562   70.8980   71.3421
                        73.1416   73.5970   74.0546   74.5143   74.9760   75.4397   75.9054
                        77.7867   78.2614   78.7378   79.2157   79.6952   80.1761   80.6584
                        82.6004   83.0888   83.5782   84.0685   84.5597   85.0517   85.5444
                        87.5212   88.0165   88.5121   89.0080   89.5039   90.0000   90.4961
                        92.4788   92.9737   93.4682   93.9622   94.4556   94.9483   95.4403
                        97.3996   97.8869   98.3730   98.8580   99.3416   99.8239  100.3048  1
                       102.2133  102.6863  103.1575  103.6270  104.0946  104.5603  105.0240  1
                       106.8584  107.3116  107.7627  108.2114  108.6579  109.1020  109.5438  1
                       111.2864  111.7158  112.1427  112.5670  112.9887  113.4078  113.8243  1
                       115.4633  115.8664  116.2666  116.6642  117.0590  117.4510  117.8403  1
                       119.3695  119.7449  120.1174  120.4872  120.8542  121.2184  121.5798  1
                       122.9977  123.3453  123.6901  124.0321  124.3715  124.7080  125.0419  1
                       126.3505  126.6710  126.9888  127.3039  127.6165  127.9265  128.2338  1
                       129.4377  129.7323  130.0244  130.3141  130.6013  130.8860  131.1684  1
                       132.2737  132.5441  132.8122  133.0780  133.3416  133.6028  133.8618  1
                       134.8757  135.1243  135.3745  135.6268  135.8814  136.1382  136.3972  1
                       137.4559  137.7263  137.9991  138.2742  138.5517  138.8316  139.1140  1
                       140.2677  140.5623  140.8595  141.1592  141.4614  141.7662  142.0735  1
                       143.3290  143.6495  143.9726  144.2984  144.6269  144.9581  145.2920  1
                       146.6547  147.0023  147.3526  147.7057  148.0616  148.4202  148.7816  1
                       150.2551  150.6305  151.0086  151.3895  151.7733  152.1597  152.5490  1
                       154.1336  154.5367  154.9424  155.3508  155.7619  156.1757  156.5922  1
                       158.2842  158.7136  159.1455  159.5800  160.0169  160.4562  160.8980  1
                       162.6884  163.1416  163.5970  164.0546  164.5143  164.9760  165.4397  1
```

```
                              167.3137  167.7867  168.2614  168.7378  169.2157  169.6952  170.1761  1
                              172.1131  172.6004  173.0888  173.5782  174.0685  174.5597  175.0517  1
                              177.0263  177.5212  178.0165  178.5121  179.0080  179.5039  180.0000  1
                              181.9835  182.4788  182.9737  183.4682  183.9622  184.4556  184.9483  1
                              186.9112  187.3996  187.8869  188.3730  188.8580  189.3416  189.8239  1
                              191.7386  192.2133  192.6863  193.1575  193.6270  194.0946  194.5603  1
                              196.4030  196.8584  197.3116  197.7627  198.2114  198.6579  199.1020  1
                              200.8545  201.2864  201.7158  202.1427  202.5670  202.9887  203.4078  2
                              205.0576  205.4633  205.8664  206.2666  206.6642  207.0590  207.4510  2
                              208.9914  209.3695  209.7449  210.1174  210.4872  210.8542  211.2184  2
                              212.6474  212.9977  213.3453  213.6901  214.0321  214.3715  214.7080  2
                              216.0274  216.3505  216.6710  216.9888  217.3039  217.6165  217.9265  2
                              219.1405  219.4377  219.7323  220.0244  220.3141  220.6013  220.8860  2
                              222.0009  222.2737  222.5441  222.8122  223.0780  223.3416  223.6028  2
                              224.6255  224.8757


     <#> measurement perfect
         projection data are noiseless


     <#> background absorption is 0.0
                               at levels
                                      75
          background absorption   0.0000


     <#> run
               0.007 seconds phantom creation
               0.117 seconds projection data creation
               0.125 seconds used for processing command crea


     <#> picture test

         EXAMPLE 8 LINOGRAM RECONSTRUCTION


     <#> spec    mono   75
         energy spectrum is monochromatic at energy level    75


     <#> obje
         description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75

   1 elip   0.0000   0.0000   8.6250   6.4687  90.0000   0.4160    0.4160

   2 elip   0.0000   0.0000   7.8750   5.7187  90.0000  -0.2060   -0.2060

   3 elip   0.0000   1.5000   0.3750   0.3000  90.0000  -0.0030   -0.0030

   4 elip   0.6750  -0.7500   0.2250   0.1500 140.0000   0.0100    0.0100
```

```
  5 elip   0.7500   1.5000   0.3750   0.2250  50.0000   0.0030    0.0030

  6 segm   1.3750  -7.5000   1.1000   0.6250  19.2000  -0.2040   -0.2040

  7 segm   1.3750  -7.5000   1.1000   4.3200  19.2100   0.2040    0.2040

  8 segm   0.0000  -2.2500   1.1250   0.3750   0.0000  -0.0030   -0.0030

  9 segm   0.0000  -2.2500   1.1250   3.0000   0.0000   0.0030    0.0030

 10 segm  -1.0000   3.7500   1.0000   0.5000 135.0000  -0.0030   -0.0030

 11 segm  -1.0000   3.7500   1.0000   3.0000 135.0000   0.0030    0.0030

 12 segm   1.0000   3.7500   1.0000   0.5000 225.0000  -0.0030   -0.0030

 13 segm   1.0000   3.7500   1.0000   3.0000 225.0000   0.0030    0.0030

 14 tria   5.0250   3.7500   1.1250   0.5000 110.7500   0.2060    0.2060

 15 tria  -5.0250   3.7500   1.1250   0.9000 -110.7500   0.2060    0.2060

      scale factor multiplying object densities     1.0000

      seed set to 0
      inhomogeneity set to     0.0000

  <#> phan    aver    5

      density in each pixel is obtained as the average of 5 x 5 points


  <#> pixe      115    size        0.1504
      picture size 115 x 115,  pixel size     0.1504

      test picture read
      EXAMPLE 8 LINOGRAM RECONSTRUCTION
           0.004 seconds used for processing command pict


  <#> projection real

      EXAMPLE 8 LINOGRAM RECONSTRUCTION


  <#> spec    mono   75
      energy spectrum is monochromatic at energy level    75


  <#> obje
      description of objects
                                                                  density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        75
```

```
 1 elip   0.0000   0.0000   8.6250   6.4687  90.0000   0.4160    0.4160

 2 elip   0.0000   0.0000   7.8750   5.7187  90.0000  -0.2060   -0.2060

 3 elip   0.0000   1.5000   0.3750   0.3000  90.0000  -0.0030   -0.0030

 4 elip   0.6750  -0.7500   0.2250   0.1500 140.0000   0.0100    0.0100

 5 elip   0.7500   1.5000   0.3750   0.2250  50.0000   0.0030    0.0030

 6 segm   1.3750  -7.5000   1.1000   0.6250  19.2000  -0.2040   -0.2040

 7 segm   1.3750  -7.5000   1.1000   4.3200  19.2100   0.2040    0.2040

 8 segm   0.0000  -2.2500   1.1250   0.3750   0.0000  -0.0030   -0.0030

 9 segm   0.0000  -2.2500   1.1250   3.0000   0.0000   0.0030    0.0030

10 segm  -1.0000   3.7500   1.0000   0.5000 135.0000  -0.0030   -0.0030

11 segm  -1.0000   3.7500   1.0000   3.0000 135.0000   0.0030    0.0030

12 segm   1.0000   3.7500   1.0000   0.5000 225.0000  -0.0030   -0.0030

13 segm   1.0000   3.7500   1.0000   3.0000 225.0000   0.0030    0.0030

14 tria   5.0250   3.7500   1.1250   0.5000 110.7500   0.2060    0.2060

15 tria  -5.0250   3.7500   1.1250   0.9000 -110.7500   0.2060    0.2060

     scale factor multiplying object densities     1.0000

     seed set to 0
     inhomogeneity set to     0.0000

 <#> rays    aver    1

     projection data are calculated by dividing each ray interval into 1 substrips

     with aperture (substrip) weights     1


 <#> geom


 <#> lino
     rays are parallel with variable spacing between rays
     data collected along lines


     number of rays per projection   231
     snark computed number of rays   233
     at detector spacing     0.1504
```

```
    total number of projections   462
    projection angles   45.1243   45.3745   45.6268   45.8814   46.1382   46.3972   46.6584
                        47.7263   47.9991   48.2742   48.5517   48.8316   49.1140   49.3987
                        50.5623   50.8595   51.1592   51.4614   51.7662   52.0735   52.3835
                        53.6495   53.9726   54.2984   54.6269   54.9581   55.2920   55.6285
                        57.0023   57.3526   57.7057   58.0616   58.4202   58.7816   59.1458
                        60.6305   61.0086   61.3895   61.7733   62.1597   62.5490   62.9410
                        64.5367   64.9424   65.3508   65.7619   66.1757   66.5922   67.0113
                        68.7136   69.1455   69.5800   70.0169   70.4562   70.8980   71.3421
                        73.1416   73.5970   74.0546   74.5143   74.9760   75.4397   75.9054
                        77.7867   78.2614   78.7378   79.2157   79.6952   80.1761   80.6584
                        82.6004   83.0888   83.5782   84.0685   84.5597   85.0517   85.5444
                        87.5212   88.0165   88.5121   89.0080   89.5039   90.0000   90.4961
                        92.4788   92.9737   93.4682   93.9622   94.4556   94.9483   95.4403
                        97.3996   97.8869   98.3730   98.8580   99.3416   99.8239  100.3048  1
                       102.2133  102.6863  103.1575  103.6270  104.0946  104.5603  105.0240  1
                       106.8584  107.3116  107.7627  108.2114  108.6579  109.1020  109.5438  1
                       111.2864  111.7158  112.1427  112.5670  112.9887  113.4078  113.8243  1
                       115.4633  115.8664  116.2666  116.6642  117.0590  117.4510  117.8403  1
                       119.3695  119.7449  120.1174  120.4872  120.8542  121.2184  121.5798  1
                       122.9977  123.3453  123.6901  124.0321  124.3715  124.7080  125.0419  1
                       126.3505  126.6710  126.9888  127.3039  127.6165  127.9265  128.2338  1
                       129.4377  129.7323  130.0244  130.3141  130.6013  130.8860  131.1684  1
                       132.2737  132.5441  132.8122  133.0780  133.3416  133.6028  133.8618  1
                       134.8757  135.1243  135.3745  135.6268  135.8814  136.1382  136.3972  1
                       137.4559  137.7263  137.9991  138.2742  138.5517  138.8316  139.1140  1
                       140.2677  140.5623  140.8595  141.1592  141.4614  141.7662  142.0735  1
                       143.3290  143.6495  143.9726  144.2984  144.6269  144.9581  145.2920  1
                       146.6547  147.0023  147.3526  147.7057  148.0616  148.4202  148.7816  1
                       150.2551  150.6305  151.0086  151.3895  151.7733  152.1597  152.5490  1
                       154.1336  154.5367  154.9424  155.3508  155.7619  156.1757  156.5922  1
                       158.2842  158.7136  159.1455  159.5800  160.0169  160.4562  160.8980  1
                       162.6884  163.1416  163.5970  164.0546  164.5143  164.9760  165.4397  1
                       167.3137  167.7867  168.2614  168.7378  169.2157  169.6952  170.1761  1
                       172.1131  172.6004  173.0888  173.5782  174.0685  174.5597  175.0517  1
                       177.0263  177.5212  178.0165  178.5121  179.0080  179.5039  180.0000  1
                       181.9835  182.4788  182.9737  183.4682  183.9622  184.4556  184.9483  1
                       186.9112  187.3996  187.8869  188.3730  188.8580  189.3416  189.8239  1
                       191.7386  192.2133  192.6863  193.1575  193.6270  194.0946  194.5603  1
                       196.4030  196.8584  197.3116  197.7627  198.2114  198.6579  199.1020  1
                       200.8545  201.2864  201.7158  202.1427  202.5670  202.9887  203.4078  2
                       205.0576  205.4633  205.8664  206.2666  206.6642  207.0590  207.4510  2
                       208.9914  209.3695  209.7449  210.1174  210.4872  210.8542  211.2184  2
                       212.6474  212.9977  213.3453  213.6901  214.0321  214.3715  214.7080  2
                       216.0274  216.3505  216.6710  216.9888  217.3039  217.6165  217.9265  2
                       219.1405  219.4377  219.7323  220.0244  220.3141  220.6013  220.8860  2
                       222.0009  222.2737  222.5441  222.8122  223.0780  223.3416  223.6028  2
                       224.6255  224.8757

<#> meas    perf
    projection data are noiseless
```

```
<#> back      0.0000
                          at levels
                                75
     background absorption   0.0000

    estimate of totlen =   1054747.357766
    estimate of totden =    154874.529107
    estimate of average density =     0.1468
    projection data read
    EXAMPLE 8 LINOGRAM RECONSTRUCTION
         0.044 seconds used for processing command proj


<#> execute lino

    EXAMPLE 8a LINOGRAM METHOD

<#> sinc 1.0
    linogram reconstruction algorithm
     using filter sinc
     cutoff =    1.0000
    algorithm executed in iteration    1
         0.061 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.061 seconds for this iteration
         0.192 seconds used for processing command exec


<#> execute conv

    EXAMPLE 8b FILTERED BACKPROJECTION

<#> cosine 1.0 2
     convolution reconstruction algorithm
     using filter cosi
     cutoff (or alpha) =     1.0000
     2 point lagrange interpolation
    algorithm executed in iteration    1
         0.109 seconds for the execution of the algorithm
    reconstruction completed after iteration    1
         0.109 seconds for this iteration
         0.240 seconds used for processing command exec


<#> end
```

A.9 Using a user-defined figure of merit in SNARK experimenter

This example illustrates the use of a user-written subroutine to compute a user-defined figure of merit in SNARK experimenter (see Section 8.2.6.7). A user written subroutine is provided in the file `user_fom1.c`. This subroutine computes a figure of merit defined by

$$1 - v^q,$$

where v^q is defined in Section 5.11 with S as the set of pixels whose center lies in a circle centered at the origin with a radius of 29.72. A listing of the user-written subroutine is given in Section A.9.1. This FOM demonstrates that one can reject the null hypotheses that the EM algorithm produces similarly noisy images whether or not the likelihood term is complemented by a regularizing term, in favor of the alternative hypothesis that the regularizing term reduces the noise in the reconstructions.

Section A.9.2 shows a listing of the SNARK experimenter input file. In this listing the files `b.9.virus.ens` and `b.9.projection.ss` are identical to those listed in Sections 8.2.2 and 8.2.4, respectively. The files `b.9.recon.ss` and `b.9.compare.ss` are listed in Sections A.9.6 and A.9.7, respectively. The output file containing the significance results is listed in Section A.9.8.

A.9.1 User-defined figure of merit

```
#include <malloc.h>
#include "experimenter.h"
#include "read_eval_phantom1.h"
#include "read_eval_recon3.h"
#include "user_fom1.h"

/* ----------------------------- user_fom1.c --------------

This function illustrates the use of a user-defined figure
of merit.

INPUTS:
  itr1 - array of length niters containing the iteration numbers for the
 first algorithm.
  itr2 - array of length niters containing the iteration numbers for the
 second algorithm.
  niters - number of iterations to be compared
  keywrd1 - string containing the keyword which defines the first algorithm.
  keywrd2 - string containing the keyword which defines the second algorithm.

OUTPUTS:
  user1 - array of length niters containing the user defined figure of
  merit for the first algorithm.
  user2 - array of length niters containing the user defined figure of
  merit for the second algorithm.

*/

void user_fom1(int* itr1, int* itr2, int niters, char* keywrd1, char* keywrd2, double* user1, doubl
     //int *itr1,*itr2,niters;
     //char *keywrd1,*keywrd2;
     //double *user1,*user2;
{
  double *phantom,*recon1,*recon2;
  int *regions,numstr,i,j,*strarea;
  float totarea=0.0,avgarea;

  /* Read in abnormality index and area for phantom structures.
     The program which does this is read_eval_phantom1.c.
     NOTE: the only information to be used is the value
     'numstr' (which is the number of structures in the phantom) */
  read_eval_phantom1(&regions,&numstr,&phantom,&strarea);
```

```
    /* allocate memory to store abnormality indexes and area
       (i.e number of pixels) for structures in the
       reconstructions */
    recon1 = (double *) malloc(numstr*sizeof(double));
    recon2 = (double *) malloc(numstr*sizeof(double));

    for(i=0;i < niters;i++) {
      /* read in variance for structures in
         the reconstructions. Initialize the arrays
         which will store the user FOM */

      //printf("*** about to read variance\n"); /* TEST (JD) */
      read_eval_recon3(itr1[i],keywrd1,numstr,recon1);
      read_eval_recon3(itr2[i],keywrd2,numstr,recon2);
      /* the varaince of the 'SINGLE' structure to be
         used is the first in the array.  The FOM value
         is 1-variance */
      user1[i]=1-recon1[0];
      user2[i]=1-recon2[0];
    }

    free(phantom); /* free memory for next function call */
    free(recon1);
    free(recon2);
    free(strarea);
  }
```

A.9.2 SNARK14Experimenter input file

```
SEED -1
ENSEMBLE b.9.virus.ens
EXPERIMENT 1 0 95 1.6 7 18 30
DATA b.9.projection.ss
RECONSTRUCTION b.9.recon.ss
ANALYSIS b.9.compare.ss
END
```

A.9.3 SNARK14Experimenter virus.ens file

```
virus_24_1_0.96_0.6.atl
virus_24_1_0.96_0.7.atl
virus_24_1_0.96_0.8.atl
virus_24_1_0.97_0.6.atl
virus_24_1_0.97_0.7.atl
virus_24_1_0.97_0.8.atl
virus_26_1_0.96_0.6.atl
virus_26_1_0.96_0.7.atl
virus_26_1_0.96_0.8.atl
virus_26_1_0.97_0.6.atl
virus_26_1_0.97_0.7.atl
virus_26_1_0.97_0.8.atl
virus_28_1_0.96_0.6.atl
virus_28_1_0.96_0.7.atl
```

```
virus_28_1_0.96_0.8.atl
virus_28_1_0.97_0.6.atl
virus_28_1_0.97_0.7.atl
virus_28_1_0.97_0.8.atl
```

A.9.4 SNARK14Experimenter virus__24__1__0.96__0.6.atl file

```
SPECTRUM MONOCHROMATIC 60
SINGLE
elip 0 0 29.72 29.72 0 0
PAIR
elip 4.14 63.19 3.89 3.89 0 1 0.96 0.5
elip 12.36 62.11 3.89 3.89 0 1 0.96 0.5
elip 20.36 59.97 3.89 3.89 0 1 0.96 0.5
elip 28.01 56.80 3.89 3.89 0 1 0.96 0.5
elip 35.18 52.66 3.89 3.89 0 1 0.96 0.5
elip 41.76 47.61 3.89 3.89 0 1 0.96 0.5
elip 47.61 41.76 3.89 3.89 0 1 0.96 0.5
elip 52.66 35.18 3.89 3.89 0 1 0.96 0.5
elip 56.80 28.01 3.89 3.89 0 1 0.96 0.5
elip 59.97 20.36 3.89 3.89 0 1 0.96 0.5
elip 62.11 12.36 3.89 3.89 0 1 0.96 0.5
elip 63.19 4.14 3.89 3.89 0 1 0.96 0.5
elip 63.19 -4.14 3.89 3.89 0 1 0.96 0.5
elip 62.11 -12.36 3.89 3.89 0 1 0.96 0.5
elip 59.97 -20.36 3.89 3.89 0 1 0.96 0.5
elip 56.80 -28.01 3.89 3.89 0 1 0.96 0.5
elip 52.66 -35.18 3.89 3.89 0 1 0.96 0.5
elip 47.61 -41.76 3.89 3.89 0 1 0.96 0.5
elip 41.76 -47.61 3.89 3.89 0 1 0.96 0.5
elip 35.18 -52.66 3.89 3.89 0 1 0.96 0.5
elip 28.01 -56.80 3.89 3.89 0 1 0.96 0.5
elip 20.36 -59.97 3.89 3.89 0 1 0.96 0.5
elip 12.36 -62.11 3.89 3.89 0 1 0.96 0.5
elip 4.14 -63.19 3.89 3.89 0 1 0.96 0.5
DUMMY
elip 0 0 59.44 59.44 0 0.6
SCALE
1
```

A.9.5 SNARK14Experimenter projection.ss file

```
raysum average 1
1
geometry
divergent arc 153 306
rays user 101 detector spacing 3.2
angles 300 equally spaced
0 358.8
measurement noisy
quantum 0.0 0.0 calibr 4
multiplicative 1.0 0.1
seed
background 0.0
```

```
run
picture test
projection real
```

A.9.6 SNARK14Experimenter recon.ss file

```
stop iteration 15
execute average emap
alg1: testing super snark with ML
gamma is 0.0
execute average emap
alg2: testing super snark with MAP
gamma is 10.0
```

A.9.7 SNARK14Experimenter compare.ss file

```
testem
COMPARE alg1  alg2 HITR  USR1
1     1
3 3
6 6
9 9
12 12
15 15
END
```

A.9.8 SNARK14Experimenter output file (testem.1)

```
     FIGURE OF MERIT: Hit-ratio
alg1  mean alg2 mean significance level
1 0.66720085470085477297  1 0.66857448107448103070  0.26326871460668666014
3 0.66709401709401705549  3 0.67362637362637345362  0.01279497859528494486
6 0.66570512820512817154  6 0.66739926739926735966  0.28449303561745276081
9 0.67206959706959712442  9 0.67136752136752131381  0.41681003501308677706
12 0.66701770451770436843  12 0.66829975579975575695  0.38225459735271893891
15 0.66312576312576299031  15 0.65789072039072016285  0.12231587153007145163

     FIGURE OF MERIT: User-defined-1
alg1  mean alg2 mean significance level
1 0.99998333333333333517  1 1.00000000000000000000  0.01267365933873415562
3 0.99970666666666707645  3 0.99977666666666697992  0.00000229641685589588
6 0.99895333333333291570  6 0.99924000000000012811  0.00000004817048354644
9 0.99784666666666699264  9 0.99844666666666670451  0.00000005544455583539
12 0.99645333333333352410  12 0.99751000000000000778  0.00000005198049951582
15 0.99478666666666681895  15 0.99644333333333356961  0.00000005097848687630
```

A.10 Reconstruction using blobs

This example illustrates reconstructing a simple image using both blob and pixel basis elements.

A.10.1 SNARK14 INPUT file

```
********EXAMPLE 10
* ROTATED RECTANGLE WITH PSEUDO PROJECTION DATA
* RECONSTRUCTION BY ART WITH BLOB BASIS
*
*TRACE 1
CREATE
EXAMPLE 10 ROTATED RECTANGLE
SPECTRUM MONOCHROMATIC 10
OBJECTS
RECT  0.0   0.0    2.0  10.0  -30.0    1.0
LAST 1.0
PHANTOM AVERAGE 1
25 PIXELS OF SIZE 1.0
RAYSUM
RUN
*
PICTURE TEST
*
PROJECTION PSEUDO
EXAMPLE 10 PSEUDO PROJECTION DATA FOR TWELVE VIEWS
GEOMETRY
PARALLEL UNIFORM LINE
RAYS PROGRAM 25 2.0 DETECTOR SPACING AT 2.0
ANGLES 12 EQUALLY SPACED
0.0 165.0
MEASUREMENT PERFECT
BACKGROUND 50.0
*
BASIS BLOB
*
SELECT SNARK RAYSEQ
STEP 3 1
*
STOP ITERATION 10
*
EXECUTE AVERAGE ART
ART WITH BLOB BASIS
ART3
CONSTRAINT ART2
*
BASIS PIXEL
EXECUTE AVERAGE ART
ART WITH PIXEL BASIS
ART3
CONSTRAINT ART2
*
EVALUATE BOTH
COMPARING BLOB AND PIXEL BASIS
WHOLEPIC
2222222222
END
```

A.10.2 SNARK14 OUTPUT file

```
      snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


        <*> *******EXAMPLE 10

        <*>  ROTATED RECTANGLE WITH PSEUDO PROJECTION DATA

        <*>  RECONSTRUCTION BY ART WITH BLOB BASIS

        <*>

        <*> TRACE 1

        <#> CREATE

            EXAMPLE 10 ROTATED RECTANGLE


        <#> SPECTRUM MONOCHROMATIC 10
            energy spectrum is monochromatic at energy level    10


        <#> OBJECTS
            description of objects
                                                                    density at levels
    numb type  x-coord  y-coord x-length y-length    angle  av dens        10

       1 rect   0.0000   0.0000   2.0000  10.0000 -30.0000   1.0000    1.0000

            scale factor multiplying object densities     1.0000

            seed set to 0
            inhomogeneity set to     0.0000

        <#> PHANTOM AVERAGE 1

            this run will generate a phantom
            density in each pixel is obtained as the average of 1 x 1 points


        <#> 25 PIXELS OF SIZE 1.0
            picture size 25 x 25,  pixel size     1.0000


        <#> RAYSUM

                 0.000 seconds phantom creation
                 0.001 seconds used for processing command crea


        <*> RUN
```

```
     <*>

     <#> PICTURE TEST

         EXAMPLE 10 ROTATED RECTANGLE


     <#> spec    mono   10
         energy spectrum is monochromatic at energy level    10


     <#> obje
         description of objects
                                                                    density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens        10

   1 rect   0.0000   0.0000   2.0000  10.0000 -30.0000   1.0000    1.0000

         scale factor multiplying object densities     1.0000

         seed set to 0
         inhomogeneity set to     0.0000

     <#> phan    aver    1

         density in each pixel is obtained as the average of 1 x 1 points


     <#> pixe       25    size        1.0000
         picture size 25 x 25,  pixel size     1.0000

         test picture read
         EXAMPLE 10 ROTATED RECTANGLE
              0.000 seconds used for processing command pict


     <*>

     <#> PROJECTION PSEUDO

         EXAMPLE 10 PSEUDO PROJECTION DATA FOR TWELVE VIEWS



     <#> GEOMETRY


     <#> PARALLEL UNIFORM LINE
         rays are parallel with uniform spacing between rays
         data collected along lines
```

```
<#> RAYS PROGRAM 25 2.0 DETECTOR SPACING AT 2.0
    number of rays per projection    37
    snark computed number of rays    19
    at detector spacing     2.0000


<#> ANGLES 12 EQUALLY SPACED
    total number of projections    12

    projection angles    0.0000   15.0000   30.0000   45.0000   60.0000   75.0000   90.0000  1
                       150.0000  165.0000


<#> MEASUREMENT PERFECT
    projection data are noiseless


<#> BACKGROUND 50.0
                           at levels
                                  10
     background absorption  50.0000

    estimate of totlen =      3776.330760
    estimate of totden =       498.725717
    estimate of average density =     0.1321
    projection data read
    EXAMPLE 10 PSEUDO PROJECTION DATA FOR TWELVE VIEWS
         0.000 seconds used for processing command proj


<*>

<#> BASIS BLOB
     hexagonal grid: there are    25 rows, the middle row has    21 grid points and the row ne
     blob basis with parameters: support =     2.0629 shape =    11.2829 delta =     1.1547
         0.078 seconds used for processing command basi


<*>

<#> SELECT SNARK RAYSEQ

<#> STEP 3 1
    sequential ray selection with ray-step     1 and projection-step     3
         0.000 seconds used for processing command sele


<*>

<#> STOP ITERATION 10
      10 iterations
         0.000 seconds used for processing command stop
```

```
<*>

<#> EXECUTE AVERAGE ART

    ART WITH BLOB BASIS

<#> ART3
     art3 method
     relaxation parameter is 1.0
     norm is 2
     tolerance is 0.0

<#> CONSTRAINT ART2
     constraint art2 way
     relaxation constraints with cr = 1.0
     picture is not normalized
       228 rays are used for each iteration
    algorithm executed in iteration    1
         0.001 seconds for the execution of the algorithm
    iteration    1 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    2
         0.001 seconds for the execution of the algorithm
    iteration    2 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    3
         0.001 seconds for the execution of the algorithm
    iteration    3 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    4
         0.001 seconds for the execution of the algorithm
    iteration    4 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    5
         0.001 seconds for the execution of the algorithm
    iteration    5 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    6
         0.001 seconds for the execution of the algorithm
    iteration    6 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    7
         0.001 seconds for the execution of the algorithm
    iteration    7 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    8
         0.001 seconds for the execution of the algorithm
    iteration    8 completed
         0.001 seconds for this iteration
    algorithm executed in iteration    9
         0.001 seconds for the execution of the algorithm
    iteration    9 completed
         0.001 seconds for this iteration
    algorithm executed in iteration   10
```

```
          0.000 seconds for the execution of the algorithm
     reconstruction completed after iteration   10
          0.001 seconds for this iteration
          0.009 seconds for all iterations
          0.010 seconds used for processing command exec


<*>

<#> BASIS PIXEL
     pixel basis
          0.000 seconds used for processing command basi


<#> EXECUTE AVERAGE ART

    ART WITH PIXEL BASIS

<#> ART3
     art3 method
     relaxation parameter is 1.0
     norm is 2
     tolerance is 0.0

<#> CONSTRAINT ART2
     constraint art2 way
     relaxation constraints with cr = 1.0
     picture is not normalized
       228 rays are used for each iteration
    algorithm executed in iteration    1
          0.000 seconds for the execution of the algorithm
    iteration    1 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    2
          0.000 seconds for the execution of the algorithm
    iteration    2 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    3
          0.000 seconds for the execution of the algorithm
    iteration    3 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    4
          0.000 seconds for the execution of the algorithm
    iteration    4 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    5
          0.000 seconds for the execution of the algorithm
    iteration    5 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    6
          0.000 seconds for the execution of the algorithm
    iteration    6 completed
          0.000 seconds for this iteration
    algorithm executed in iteration    7
```

```
            0.000 seconds for the execution of the algorithm
        iteration    7 completed
            0.000 seconds for this iteration
        algorithm executed in iteration    8
            0.000 seconds for the execution of the algorithm
        iteration    8 completed
            0.000 seconds for this iteration
        algorithm executed in iteration    9
            0.000 seconds for the execution of the algorithm
        iteration    9 completed
            0.000 seconds for this iteration
        algorithm executed in iteration   10
            0.000 seconds for the execution of the algorithm
        reconstruction completed after iteration   10
            0.000 seconds for this iteration
            0.001 seconds for all iterations
            0.002 seconds used for processing command exec


    <*>

    <#> EVALUATE BOTH

        COMPARING BLOB AND PIXEL BASIS

    <#> WHOLEPIC
Region        cx        cy         u         v        ang      t1        t2

wholepic                                                      -1e+20     1e+20

        iterations    1   2   3   4   5   6   7   8   9
        last iteration

   residual calculation using line integration
            0.004 seconds used for processing command eval


    <#> END
```

A.10.3 SNARK14 eval file

```
evaluation name: COMPARING BLOB AND PIXEL BASIS

  global resolution measures

    phantom name:    EXAMPLE 10 ROTATED RECTANGLE
    projection name: EXAMPLE 10 PSEUDO PROJECTION DATA FOR TWELVE VIEWS

    metrics for test phantom
    region    area   average  distance   rel err  variance   std dev   residual
         0     625    0.1328                        0.1152    0.3394     0.0000
```

```
   execution name: ART WITH BLOB BASIS
   metrics for algorithm ART
   iter    area   average  distance   rel err  variance   std dev   residual
      1     625    0.1339    0.5065    0.8109    0.0760    0.2757     4.5150
      2     625    0.1327    0.5022    0.7910    0.0720    0.2684     3.3416
      3     625    0.1318    0.4986    0.7870    0.0717    0.2678     2.7331
      4     625    0.1310    0.4967    0.7866    0.0713    0.2669     2.3763
      5     625    0.1305    0.4957    0.7881    0.0709    0.2663     2.1742
      6     625    0.1302    0.4950    0.7907    0.0707    0.2659     2.0674
      7     625    0.1299    0.4946    0.7928    0.0705    0.2655     2.0114
      8     625    0.1298    0.4943    0.7951    0.0703    0.2651     1.9793
      9     625    0.1298    0.4941    0.7970    0.0702    0.2649     1.9568
     10     625    0.1298    0.4939    0.7982    0.0701    0.2647     1.9369

   execution name: ART WITH PIXEL BASIS
   metrics for algorithm ART
   iter    area   average  distance   rel err  variance   std dev   residual
      1     625    0.1277    0.7316    1.3298    0.0756    0.2749     8.5354
      2     625    0.1251    0.6998    1.2100    0.0644    0.2537     3.2825
      3     625    0.1258    0.6950    1.1952    0.0615    0.2481     1.5104
      4     625    0.1267    0.6939    1.1945    0.0605    0.2459     0.7835
      5     625    0.1272    0.6935    1.1937    0.0601    0.2452     0.4419
      6     625    0.1275    0.6933    1.1932    0.0600    0.2449     0.2689
      7     625    0.1277    0.6932    1.1927    0.0599    0.2447     0.1752
      8     625    0.1277    0.6932    1.1923    0.0599    0.2447     0.1192
      9     625    0.1277    0.6932    1.1919    0.0598    0.2446     0.0827
     10     625    0.1277    0.6932    1.1917    0.0598    0.2446     0.0574
point by point resolution measures

   phantom name:    EXAMPLE 10 ROTATED RECTANGLE
   projection name: EXAMPLE 10 PSEUDO PROJECTION DATA FOR TWELVE VIEWS

   execution name:  ART WITH BLOB BASIS
   algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
    ART       1    0.6508    0.4348    0.1281    0.0191    0.0026
    ART       2    0.6501    0.4530    0.1313    0.0160    0.0014
    ART       3    0.6305    0.4460    0.1371    0.0139    0.0013
    ART       4    0.6192    0.4445    0.1427    0.0172    0.0030
    ART       5    0.6104    0.4432    0.1471    0.0213    0.0044
    ART       6    0.6043    0.4426    0.1505    0.0244    0.0053
    ART       7    0.6000    0.4422    0.1531    0.0267    0.0059
    ART       8    0.5971    0.4419    0.1552    0.0283    0.0063
    ART       9    0.5951    0.4417    0.1567    0.0295    0.0066
    ART      10    0.5939    0.4416    0.1579    0.0303    0.0067

   execution name:  ART WITH PIXEL BASIS
   algn    iter      e(0)      e(1)      e(2)      e(3)      e(4)
    ART       1    0.8749    0.6197    0.2661    0.0617    0.0091
    ART       2    0.8323    0.6423    0.3007    0.0922    0.0149
    ART       3    0.8174    0.6479    0.3138    0.1034    0.0161
    ART       4    0.8160    0.6504    0.3177    0.1071    0.0159
    ART       5    0.8145    0.6510    0.3187    0.1083    0.0156
    ART       6    0.8134    0.6511    0.3190    0.1087    0.0155
    ART       7    0.8128    0.6511    0.3191    0.1089    0.0154
```

```
ART        8     0.8124     0.6510     0.3192     0.1090     0.0154
ART        9     0.8123     0.6509     0.3193     0.1091     0.0155
ART       10     0.8124     0.6509     0.3194     0.1092     0.0155
```

A.11 Simulating data collection and superiorized image reconstruction for PET

The basic setup of this example is very similar to example A.7. The following adjustments have been made in order to improve visibility: the digitization has been increased to 381 x 381 pixels, the view angles have been increased to 500 and the rays per view have been increased to 301. Superiorization is applied with the following settings: N = 16, a = 0.999, b = 1. SMOO is used as secondary optimization criterion and the stopping criterion has been changed from 20 iterations to the Kullback-Leibler distance of getting below 100,000, resulting in 12 iterations.

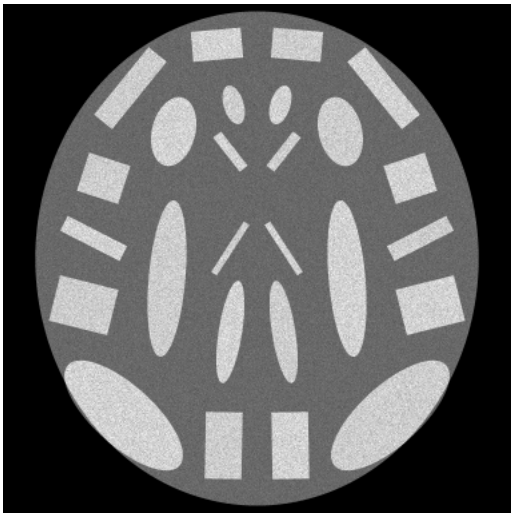

Figure A.11: A Brain Phantom

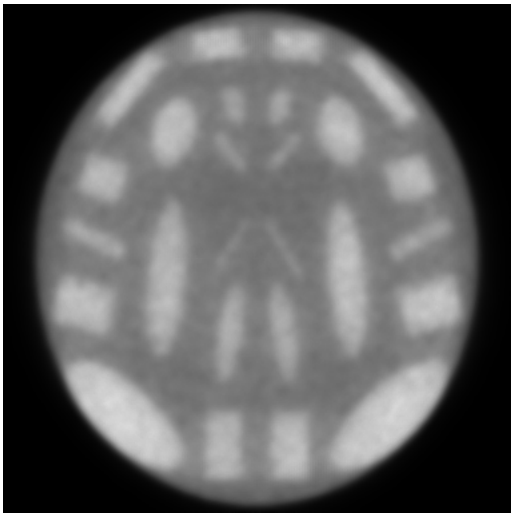

Figure A.12: Superiorized MAP EM reconstruction after 24 iterations

A.11.1 SNARK14 INPUT file

```
********EXAMPLE 11
* SUPERIORIZED MAP EM ALGORITHM FOR EMISSION TOMOGRAPHY. RECONSTRUCTION
* OF BRAIN PHANTOM. SIMULATING PET GEOMETRY WITH A RING OF 300 DETECTORS
* WITH EACH DETECTOR IN COINCIDENCE WITH 101 DETECTORS OPPOSITE IT.
* COEFFICIENT OF PENALTLY TERM IS SET TO 0
*
CREATE
EXAMPLE 11 Brain Phantom
SPECTRUM MONOCHROMATIC 511
OBJECTS
   1 elip   -7.0   46.0   3.0   6.0   17.0  0.95  1.0
   2 elip    7.0   46.0   3.0   6.0  -17.0  1.0   0.95
   3 rect  -12.0   64.0   7.5   4.5    5.0  1.0   1.0
   4 rect   12.0   64.0   7.5   4.5   -5.0  0.95  0.95
   5 rect  -38.0   51.0   3.5  13.0  -39.0  0.95  1.0
   6 rect   38.0   51.0   3.5  13.0   39.0  1.0   0.95
   7 rect  -46.0   24.0   6.5   6.0  -18.0  0.95  1.0
   8 rect   46.0   24.0   6.5   6.0   18.0  1.0   0.95
   9 rect  -49.0    6.0   2.5  10.0   63.0  1.0   1.0
  10 rect   49.0    6.0   2.5  10.0  -63.0  0.95  0.95
  11 rect  -52.0  -14.0   9.0   7.0  -14.0  0.95  1.0
  12 rect   52.0  -14.0   9.0   7.0   14.0  1.0   0.95
  13 rect  -10.0  -56.0   5.5  10.0   -1.0  0.95  1.0
  14 rect   10.0  -56.0   5.5  10.0    1.0  1.0   0.95
  15 elip  -40.0  -47.0   9.0  22.5   48.0  1.0   1.0
  16 elip   40.0  -47.0   9.0  22.5  -48.0  0.95  0.95
  17 elip   -8.0  -22.0   3.5  15.5   -9.0  1.0   1.0
  18 elip    8.0  -22.0   3.5  15.5    9.0  0.95  0.95
  19 elip  -27.0   -6.0   5.5  23.5   -5.0  0.95  1.0
  20 elip   27.0   -6.0   5.5  23.5    5.0  1.0   0.95
  21 elip  -25.0   38.0   6.5  10.5  -14.0  1.0   1.0
  22 elip   25.0   38.0   6.5  10.5   14.0  0.95  0.95
  23 rect   -8.0   32.0   1.5   6.5   38.0  1.0   1.0
  24 rect    8.0   32.0   1.5   6.5  -38.0  0.95  0.95
  25 rect   -8.0    3.0   1.0   9.0  -33.0  0.95  1.0
  26 rect    8.0    3.0   1.0   9.0   33.0  1.0   0.95
  27 elip    0.0    0.0  66.5  74.0    0.0  1.0   1.0
LAST .51    1   0.05
PHANTOM AVERAGE 7
381 PIXELS OF SIZE 0.4
RAYSUM AVERAGE 1
1
GEOMETRY
divergent arc 153 306
RAYS USER 301 DETECTOR SPACING 1.1
ANGLES 500 EQUAL SPACING
0.0 358.8
MEASUREMENT NOISY
QUANTUM 1.0 1.0 CALIBRATION 4
SEED 0
BACKGROUND 0.0
RUN
```

```
*
PICTURE TEST
*
PROJECTION REAL
*
STOP TERMINATION KLDS 100000 RPRT
*
SUPERIORIZE 16 0.999 1 SMOO RPRT
EXECUTE AVERAGE EMAP
Example 11 Illustrating the Superiorized MAP EM algorithm for PET
gamma is 0 EVAL
*
END
```

A.11.2 SNARK14 OUTPUT file

```
   snark14.s171029 - A PICTURE RECONSTRUCTION PROGRAM


     <*> *******EXAMPLE 11

     <*>  SUPERIORIZED MAP EM ALGORITHM FOR EMISSION TOMOGRAPHY. RECONSTRUCTION

     <*>  OF BRAIN PHANTOM. SIMULATING PET GEOMETRY WITH A RING OF 300 DETECTORS

     <*>  WITH EACH DETECTOR IN COINCIDENCE WITH 101 DETECTORS OPPOSITE IT.

     <*>  COEFFICIENT OF PENALTLY TERM IS SET TO 0

     <*>

     <#> CREATE

         EXAMPLE 11 Brain Phantom


     <#> SPECTRUM MONOCHROMATIC 511
         energy spectrum is monochromatic at energy level   511


     <#> OBJECTS
         description of objects
                                                                        density at levels
 numb type  x-coord  y-coord x-length y-length    angle  av dens       511

    1 elip  -7.0000  46.0000   3.0000   6.0000  17.0000   0.9500    0.9500

    2 elip   7.0000  46.0000   3.0000   6.0000 -17.0000   1.0000    1.0000

    3 rect -12.0000  64.0000   7.5000   4.5000   5.0000   1.0000    1.0000

    4 rect  12.0000  64.0000   7.5000   4.5000  -5.0000   0.9500    0.9500
```

```
 5 rect -38.0000  51.0000   3.5000  13.0000 -39.0000   0.9500    0.9500

 6 rect  38.0000  51.0000   3.5000  13.0000  39.0000   1.0000    1.0000

 7 rect -46.0000  24.0000   6.5000   6.0000 -18.0000   0.9500    0.9500

 8 rect  46.0000  24.0000   6.5000   6.0000  18.0000   1.0000    1.0000

 9 rect -49.0000   6.0000   2.5000  10.0000  63.0000   1.0000    1.0000

10 rect  49.0000   6.0000   2.5000  10.0000 -63.0000   0.9500    0.9500

11 rect -52.0000 -14.0000   9.0000   7.0000 -14.0000   0.9500    0.9500

12 rect  52.0000 -14.0000   9.0000   7.0000  14.0000   1.0000    1.0000

13 rect -10.0000 -56.0000   5.5000  10.0000  -1.0000   0.9500    0.9500

14 rect  10.0000 -56.0000   5.5000  10.0000   1.0000   1.0000    1.0000

15 elip -40.0000 -47.0000   9.0000  22.5000  48.0000   1.0000    1.0000

16 elip  40.0000 -47.0000   9.0000  22.5000 -48.0000   0.9500    0.9500

17 elip  -8.0000 -22.0000   3.5000  15.5000  -9.0000   1.0000    1.0000

18 elip   8.0000 -22.0000   3.5000  15.5000   9.0000   0.9500    0.9500

19 elip -27.0000  -6.0000   5.5000  23.5000  -5.0000   0.9500    0.9500

20 elip  27.0000  -6.0000   5.5000  23.5000   5.0000   1.0000    1.0000

21 elip -25.0000  38.0000   6.5000  10.5000 -14.0000   1.0000    1.0000

22 elip  25.0000  38.0000   6.5000  10.5000  14.0000   0.9500    0.9500

23 rect  -8.0000  32.0000   1.5000   6.5000  38.0000   1.0000    1.0000

24 rect   8.0000  32.0000   1.5000   6.5000 -38.0000   0.9500    0.9500

25 rect  -8.0000   3.0000   1.0000   9.0000 -33.0000   0.9500    0.9500

26 rect   8.0000   3.0000   1.0000   9.0000  33.0000   1.0000    1.0000

27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   1.0000    1.0000

      scale factor multiplying object densities     0.5100

      seed set to 1
      inhomogeneity set to     0.0500

  <#> PHANTOM AVERAGE 7

      this run will generate a phantom
```

```
    density in each pixel is obtained as the average of 7 x 7 points


<#> 381 PIXELS OF SIZE 0.4
    picture size 381 x 381,  pixel size     0.4000


<#> RAYSUM AVERAGE 1

    this run will generate projection data
    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> GEOMETRY


<#> divergent arc 153 306
    rays are divergent from point sources
    source to origin distance    153.0000
    the detectors lie on an arc with source to detector distance =   306.0000


<#> RAYS USER 301 DETECTOR SPACING 1.1
    number of rays per projection   301
    at detector spacing     1.1000


<#> ANGLES 500 EQUAL SPACING
    total number of projections   500

    projection angles    0.0000    0.7190    1.4381    2.1571    2.8762    3.5952    4.3142
                         7.1904    7.9094    8.6285    9.3475   10.0665   10.7856   11.5046
                        14.3808   15.0998   15.8188   16.5379   17.2569   17.9760   18.6950
                        21.5711   22.2902   23.0092   23.7283   24.4473   25.1663   25.8854
                        28.7615   29.4806   30.1996   30.9186   31.6377   32.3567   33.0758
                        35.9519   36.6709   37.3900   38.1090   38.8281   39.5471   40.2661
                        43.1423   43.8613   44.5804   45.2994   46.0184   46.7375   47.4565
                        50.3327   51.0517   51.7707   52.4898   53.2088   53.9279   54.6469
                        57.5230   58.2421   58.9611   59.6802   60.3992   61.1182   61.8373
                        64.7134   65.4325   66.1515   66.8705   67.5896   68.3086   69.0277
                        71.9038   72.6228   73.3419   74.0609   74.7800   75.4990   76.2180
                        79.0942   79.8132   80.5323   81.2513   81.9703   82.6894   83.4084
                        86.2846   87.0036   87.7226   88.4417   89.1607   89.8798   90.5988
                        93.4749   94.1940   94.9130   95.6321   96.3511   97.0701   97.7892
                       100.6653  101.3844  102.1034  102.8224  103.5415  104.2605  104.9796  1
                       107.8557  108.5747  109.2938  110.0128  110.7319  111.4509  112.1699  1
                       115.0461  115.7651  116.4842  117.2032  117.9222  118.6413  119.3603  1
                       122.2365  122.9555  123.6745  124.3936  125.1126  125.8317  126.5507  1
                       129.4269  130.1459  130.8649  131.5840  132.3030  133.0220  133.7411  1
                       136.6172  137.3363  138.0553  138.7743  139.4934  140.2124  140.9315  1
                       143.8076  144.5267  145.2457  145.9647  146.6838  147.4028  148.1218  1
                       150.9980  151.7170  152.4361  153.1551  153.8741  154.5932  155.3122  1
```

```
                          158.1884  158.9074  159.6265  160.3455  161.0645  161.7836  162.5026  1
                          165.3788  166.0978  166.8168  167.5359  168.2549  168.9739  169.6930  1
                          172.5691  173.2882  174.0072  174.7263  175.4453  176.1643  176.8834  1
                          179.7595  180.4786  181.1976  181.9166  182.6357  183.3547  184.0737  1
                          186.9499  187.6689  188.3880  189.1070  189.8261  190.5451  191.2641  1
                          194.1403  194.8593  195.5784  196.2974  197.0164  197.7355  198.4545  1
                          201.3307  202.0497  202.7687  203.4878  204.2068  204.9259  205.6449  2
                          208.5210  209.2401  209.9591  210.6782  211.3972  212.1162  212.8353  2
                          215.7114  216.4305  217.1495  217.8685  218.5876  219.3066  220.0257  2
                          222.9018  223.6208  224.3399  225.0589  225.7780  226.4970  227.2160  2
                          230.0922  230.8112  231.5303  232.2493  232.9683  233.6874  234.4064  2
                          237.2826  238.0016  238.7206  239.4397  240.1587  240.8778  241.5968  2
                          244.4729  245.1920  245.9110  246.6301  247.3491  248.0681  248.7872  2
                          251.6633  252.3824  253.1014  253.8204  254.5395  255.2585  255.9776  2
                          258.8537  259.5727  260.2918  261.0108  261.7299  262.4489  263.1679  2
                          266.0441  266.7631  267.4822  268.2012  268.9202  269.6393  270.3583  2
                          273.2345  273.9535  274.6725  275.3916  276.1106  276.8297  277.5487  2
                          280.4248  281.1439  281.8629  282.5820  283.3010  284.0200  284.7391  2
                          287.6152  288.3343  289.0533  289.7723  290.4914  291.2104  291.9295  2
                          294.8056  295.5246  296.2437  296.9627  297.6818  298.4008  299.1198  2
                          301.9960  302.7150  303.4341  304.1531  304.8721  305.5912  306.3102  3
                          309.1864  309.9054  310.6244  311.3435  312.0625  312.7816  313.5006  3
                          316.3768  317.0958  317.8148  318.5339  319.2529  319.9719  320.6910  3
                          323.5671  324.2862  325.0052  325.7242  326.4433  327.1623  327.8814  3
                          330.7575  331.4766  332.1956  332.9146  333.6337  334.3527  335.0717  3
                          337.9479  338.6669  339.3860  340.1050  340.8240  341.5431  342.2621  3
                          345.1383  345.8573  346.5764  347.2954  348.0144  348.7335  349.4525  3
                          352.3287  353.0477  353.7667  354.4858  355.2048  355.9238  356.6429  3

<#> MEASUREMENT NOISY
    noise characteristics of projection data follow
          nature          characteristics

<#> QUANTUM 1.0 1.0 CALIBRATION 4
         Emission tomography

<#> SEED 0
     seed for random number generator is              0


<#> BACKGROUND 0.0
                           at levels
                                 511
     background absorption   0.0000


<#> RUN
         0.361 seconds phantom creation
         1.177 seconds projection data creation
         1.539 seconds used for processing command crea


<*>
```

```
    <#> PICTURE TEST

        EXAMPLE 11 Brain Phantom


    <#> spec    mono  511
        energy spectrum is monochromatic at energy level   511


    <#> obje
        description of objects
                                                                        density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens       511

   1 elip  -7.0000  46.0000   3.0000   6.0000  17.0000   0.4845    0.4845

   2 elip   7.0000  46.0000   3.0000   6.0000 -17.0000   0.5100    0.5100

   3 rect -12.0000  64.0000   7.5000   4.5000   5.0000   0.5100    0.5100

   4 rect  12.0000  64.0000   7.5000   4.5000  -5.0000   0.4845    0.4845

   5 rect -38.0000  51.0000   3.5000  13.0000 -39.0000   0.4845    0.4845

   6 rect  38.0000  51.0000   3.5000  13.0000  39.0000   0.5100    0.5100

   7 rect -46.0000  24.0000   6.5000   6.0000 -18.0000   0.4845    0.4845

   8 rect  46.0000  24.0000   6.5000   6.0000  18.0000   0.5100    0.5100

   9 rect -49.0000   6.0000   2.5000  10.0000  63.0000   0.5100    0.5100

  10 rect  49.0000   6.0000   2.5000  10.0000 -63.0000   0.4845    0.4845

  11 rect -52.0000 -14.0000   9.0000   7.0000 -14.0000   0.4845    0.4845

  12 rect  52.0000 -14.0000   9.0000   7.0000  14.0000   0.5100    0.5100

  13 rect -10.0000 -56.0000   5.5000  10.0000  -1.0000   0.4845    0.4845

  14 rect  10.0000 -56.0000   5.5000  10.0000   1.0000   0.5100    0.5100

  15 elip -40.0000 -47.0000   9.0000  22.5000  48.0000   0.5100    0.5100

  16 elip  40.0000 -47.0000   9.0000  22.5000 -48.0000   0.4845    0.4845

  17 elip  -8.0000 -22.0000   3.5000  15.5000  -9.0000   0.5100    0.5100

  18 elip   8.0000 -22.0000   3.5000  15.5000   9.0000   0.4845    0.4845

  19 elip -27.0000  -6.0000   5.5000  23.5000  -5.0000   0.4845    0.4845

  20 elip  27.0000  -6.0000   5.5000  23.5000   5.0000   0.5100    0.5100
```

```
  21 elip -25.0000  38.0000   6.5000  10.5000 -14.0000   0.5100    0.5100

  22 elip  25.0000  38.0000   6.5000  10.5000  14.0000   0.4845    0.4845

  23 rect  -8.0000  32.0000   1.5000   6.5000  38.0000   0.5100    0.5100

  24 rect   8.0000  32.0000   1.5000   6.5000 -38.0000   0.4845    0.4845

  25 rect  -8.0000   3.0000   1.0000   9.0000 -33.0000   0.4845    0.4845

  26 rect   8.0000   3.0000   1.0000   9.0000  33.0000   0.5100    0.5100

  27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   0.5100    0.5100

       scale factor multiplying object densities     0.5100

       seed set to 1
       inhomogeneity set to     0.0500

    <#> phan    aver    7

       density in each pixel is obtained as the average of 7 x 7 points


    <#> pixe      381    size        0.4000
       picture size 381 x 381,  pixel size     0.4000

       test picture read
       EXAMPLE 11 Brain Phantom
            0.039 seconds used for processing command pict


    <*>

    <#> PROJECTION REAL

       EXAMPLE 11 Brain Phantom


    <#> spec    mono  511
       energy spectrum is monochromatic at energy level   511


    <#> obje
       description of objects
                                                                  density at levels
numb type  x-coord  y-coord x-length y-length    angle  av dens       511

   1 elip  -7.0000  46.0000   3.0000   6.0000  17.0000   0.4845    0.4845

   2 elip   7.0000  46.0000   3.0000   6.0000 -17.0000   0.5100    0.5100

   3 rect -12.0000  64.0000   7.5000   4.5000   5.0000   0.5100    0.5100
```

```
 4 rect  12.0000  64.0000   7.5000   4.5000  -5.0000   0.4845    0.4845
 5 rect -38.0000  51.0000   3.5000  13.0000 -39.0000   0.4845    0.4845
 6 rect  38.0000  51.0000   3.5000  13.0000  39.0000   0.5100    0.5100
 7 rect -46.0000  24.0000   6.5000   6.0000 -18.0000   0.4845    0.4845
 8 rect  46.0000  24.0000   6.5000   6.0000  18.0000   0.5100    0.5100
 9 rect -49.0000   6.0000   2.5000  10.0000  63.0000   0.5100    0.5100
10 rect  49.0000   6.0000   2.5000  10.0000 -63.0000   0.4845    0.4845
11 rect -52.0000 -14.0000   9.0000   7.0000 -14.0000   0.4845    0.4845
12 rect  52.0000 -14.0000   9.0000   7.0000  14.0000   0.5100    0.5100
13 rect -10.0000 -56.0000   5.5000  10.0000  -1.0000   0.4845    0.4845
14 rect  10.0000 -56.0000   5.5000  10.0000   1.0000   0.5100    0.5100
15 elip -40.0000 -47.0000   9.0000  22.5000  48.0000   0.5100    0.5100
16 elip  40.0000 -47.0000   9.0000  22.5000 -48.0000   0.4845    0.4845
17 elip  -8.0000 -22.0000   3.5000  15.5000  -9.0000   0.5100    0.5100
18 elip   8.0000 -22.0000   3.5000  15.5000   9.0000   0.4845    0.4845
19 elip -27.0000  -6.0000   5.5000  23.5000  -5.0000   0.4845    0.4845
20 elip  27.0000  -6.0000   5.5000  23.5000   5.0000   0.5100    0.5100
21 elip -25.0000  38.0000   6.5000  10.5000 -14.0000   0.5100    0.5100
22 elip  25.0000  38.0000   6.5000  10.5000  14.0000   0.4845    0.4845
23 rect  -8.0000  32.0000   1.5000   6.5000  38.0000   0.5100    0.5100
24 rect   8.0000  32.0000   1.5000   6.5000 -38.0000   0.4845    0.4845
25 rect  -8.0000   3.0000   1.0000   9.0000 -33.0000   0.4845    0.4845
26 rect   8.0000   3.0000   1.0000   9.0000  33.0000   0.5100    0.5100
27 elip   0.0000   0.0000  66.5000  74.0000   0.0000   0.5100    0.5100

      scale factor multiplying object densities     0.5100

      seed set to 1
      inhomogeneity set to     0.0500
```

```
<#> rays    aver    1

    projection data are calculated by dividing each ray interval into 1 substrips

    with aperture (substrip) weights     1


<#> geom


<#> dive    arc     source at  153.0000     det dist  306.0000
    rays are divergent from point sources
    source to origin distance    153.0000
    the detectors lie on an arc with source to detector distance =   306.0000


<#> rays    user      301    spacing        1.1000
    number of rays per projection   301
    snark computed number of rays   437
    at detector spacing     1.1000


<#> angl      500
    total number of projections   500


    projection angles    0.0000    0.7190    1.4381    2.1571    2.8762    3.5952    4.3142
                         7.1904    7.9094    8.6285    9.3475   10.0665   10.7856   11.5046
                        14.3808   15.0998   15.8188   16.5379   17.2569   17.9760   18.6950
                        21.5711   22.2902   23.0092   23.7283   24.4473   25.1663   25.8854
                        28.7615   29.4806   30.1996   30.9186   31.6377   32.3567   33.0758
                        35.9519   36.6709   37.3900   38.1090   38.8281   39.5471   40.2661
                        43.1423   43.8613   44.5804   45.2994   46.0184   46.7375   47.4565
                        50.3327   51.0517   51.7707   52.4898   53.2088   53.9279   54.6469
                        57.5230   58.2421   58.9611   59.6802   60.3992   61.1182   61.8373
                        64.7134   65.4325   66.1515   66.8705   67.5896   68.3086   69.0277
                        71.9038   72.6228   73.3419   74.0609   74.7800   75.4990   76.2180
                        79.0942   79.8132   80.5323   81.2513   81.9703   82.6894   83.4084
                        86.2846   87.0036   87.7226   88.4417   89.1607   89.8798   90.5988
                        93.4749   94.1940   94.9130   95.6321   96.3511   97.0701   97.7892
                       100.6653  101.3844  102.1034  102.8224  103.5415  104.2605  104.9796  1
                       107.8557  108.5747  109.2938  110.0128  110.7319  111.4509  112.1699  1
                       115.0461  115.7651  116.4842  117.2032  117.9222  118.6413  119.3603  1
                       122.2365  122.9555  123.6745  124.3936  125.1126  125.8317  126.5507  1
                       129.4269  130.1459  130.8649  131.5840  132.3030  133.0220  133.7411  1
                       136.6172  137.3363  138.0553  138.7743  139.4934  140.2124  140.9315  1
                       143.8076  144.5267  145.2457  145.9647  146.6838  147.4028  148.1218  1
                       150.9980  151.7170  152.4361  153.1551  153.8741  154.5932  155.3122  1
                       158.1884  158.9074  159.6265  160.3455  161.0645  161.7836  162.5026  1
                       165.3788  166.0978  166.8168  167.5359  168.2549  168.9739  169.6930  1
                       172.5691  173.2882  174.0072  174.7263  175.4453  176.1643  176.8834  1
                       179.7595  180.4786  181.1976  181.9166  182.6357  183.3547  184.0737  1
                       186.9499  187.6689  188.3880  189.1070  189.8261  190.5451  191.2641  1
                       194.1403  194.8593  195.5784  196.2974  197.0164  197.7355  198.4545  1
```

```
                          201.3307  202.0497  202.7687  203.4878  204.2068  204.9259  205.6449  2
                          208.5210  209.2401  209.9591  210.6782  211.3972  212.1162  212.8353  2
                          215.7114  216.4305  217.1495  217.8685  218.5876  219.3066  220.0257  2
                          222.9018  223.6208  224.3399  225.0589  225.7780  226.4970  227.2160  2
                          230.0922  230.8112  231.5303  232.2493  232.9683  233.6874  234.4064  2
                          237.2826  238.0016  238.7206  239.4397  240.1587  240.8778  241.5968  2
                          244.4729  245.1920  245.9110  246.6301  247.3491  248.0681  248.7872  2
                          251.6633  252.3824  253.1014  253.8204  254.5395  255.2585  255.9776  2
                          258.8537  259.5727  260.2918  261.0108  261.7299  262.4489  263.1679  2
                          266.0441  266.7631  267.4822  268.2012  268.9202  269.6393  270.3583  2
                          273.2345  273.9535  274.6725  275.3916  276.1106  276.8297  277.5487  2
                          280.4248  281.1439  281.8629  282.5820  283.3010  284.0200  284.7391  2
                          287.6152  288.3343  289.0533  289.7723  290.4914  291.2104  291.9295  2
                          294.8056  295.5246  296.2437  296.9627  297.6818  298.4008  299.1198  2
                          301.9960  302.7150  303.4341  304.1531  304.8721  305.5912  306.3102  3
                          309.1864  309.9054  310.6244  311.3435  312.0625  312.7816  313.5006  3
                          316.3768  317.0958  317.8148  318.5339  319.2529  319.9719  320.6910  3
                          323.5671  324.2862  325.0052  325.7242  326.4433  327.1623  327.8814  3
                          330.7575  331.4766  332.1956  332.9146  333.6337  334.3527  335.0717  3
                          337.9479  338.6669  339.3860  340.1050  340.8240  341.5431  342.2621  3
                          345.1383  345.8573  346.5764  347.2954  348.0144  348.7335  349.4525  3
                          352.3287  353.0477  353.7667  354.4858  355.2048  355.9238  356.6429  3


<#> meas    nois
    noise characteristics of projection data follow
          nature          characteristics

<#> quan            1.0000            1.0000    cali  4
         Emission tomography

<#> seed       0
     seed for random number generator is              0


<#> back      0.0000
                           at levels
                                 511
     background absorption   0.0000

    estimate of totlen =  20962469.574466
    estimate of totden =   9802000.000000
    estimate of average density =     0.4676
    projection data read
    EXAMPLE 11 Brain Phantom
         0.051 seconds used for processing command proj


<*>

<#> STOP TERMINATION KLDS 100000 RPRT
    termination test klds
    reporting is enabled
    reporting file: RPRTklds
```

```
        reporting on every iteration
        epsilon = 100000
             0.000 seconds used for processing command stop


   <*>

   <#> SUPERIORIZE 16 0.999 1 SMOO RPRT
        Superiorization is enabled
        N = 16
        a = 0.999
        b = 1
        secondary criterion: smoo
        reporting is enabled
        reporting file: RPRTsuperiorization
        reporting on every iteration
             0.000 seconds used for processing command supe


   <#> EXECUTE AVERAGE EMAP

        Example 11 Illustrating the Superiorized MAP EM algorithm for PET

   <#> gamma is 0 EVAL

------------------------------------------------------------

 maximum a-posteriori probability expectation maximization

          gamma:   0.000
          evaluation flag is set

------------------------------------------------------------

             value of l: 15
             value of phi before algorithm operator: 4.42622e-28
             value of phi after algorithm operator:  3.14467
        algorithm executed in iteration    1
             3.333 seconds for the execution of the algorithm
        current epsilon (KL distance) = 638282
        iteration    1 completed
             4.308 seconds for this iteration
             value of l: 31
             value of phi before algorithm operator: 0.429607
             value of phi after algorithm operator:  1.65065
        algorithm executed in iteration    2
             2.215 seconds for the execution of the algorithm
        current epsilon (KL distance) = 446785
        iteration    2 completed
             3.228 seconds for this iteration
             value of l: 47
             value of phi before algorithm operator: 0.706119
             value of phi after algorithm operator:  2.49879
        algorithm executed in iteration    3
```

```
     2.225 seconds for the execution of the algorithm
current epsilon (KL distance) = 331430
iteration    3 completed
     3.246 seconds for this iteration
     value of l: 63
     value of phi before algorithm operator: 0.564997
     value of phi after algorithm operator:  2.14542
algorithm executed in iteration    4
     2.242 seconds for the execution of the algorithm
current epsilon (KL distance) = 257464
iteration    4 completed
     3.246 seconds for this iteration
     value of l: 79
     value of phi before algorithm operator: 0.577605
     value of phi after algorithm operator:  2.1505
algorithm executed in iteration    5
     2.293 seconds for the execution of the algorithm
current epsilon (KL distance) = 207738
iteration    5 completed
     3.347 seconds for this iteration
     value of l: 95
     value of phi before algorithm operator: 0.579469
     value of phi after algorithm operator:  2.11049
algorithm executed in iteration    6
     2.523 seconds for the execution of the algorithm
current epsilon (KL distance) = 173224
iteration    6 completed
     3.666 seconds for this iteration
     value of l: 111
     value of phi before algorithm operator: 0.581496
     value of phi after algorithm operator:  2.08666
algorithm executed in iteration    7
     2.553 seconds for the execution of the algorithm
current epsilon (KL distance) = 148708
iteration    7 completed
     4.322 seconds for this iteration
     value of l: 127
     value of phi before algorithm operator: 0.587932
     value of phi after algorithm operator:  2.07545
algorithm executed in iteration    8
     3.349 seconds for the execution of the algorithm
current epsilon (KL distance) = 130971
iteration    8 completed
     4.363 seconds for this iteration
     value of l: 143
     value of phi before algorithm operator: 0.598034
     value of phi after algorithm operator:  2.07283
algorithm executed in iteration    9
     2.207 seconds for the execution of the algorithm
current epsilon (KL distance) = 117930
iteration    9 completed
     3.191 seconds for this iteration
     value of l: 159
     value of phi before algorithm operator: 0.610951
```

```
     value of phi after algorithm operator:  2.07635
algorithm executed in iteration   10
     2.381 seconds for the execution of the algorithm
current epsilon (KL distance) = 108196
iteration   10 completed
     3.380 seconds for this iteration
     value of l: 175
     value of phi before algorithm operator: 0.625866
     value of phi after algorithm operator:  2.08405
algorithm executed in iteration   11
     2.345 seconds for the execution of the algorithm
current epsilon (KL distance) = 100827
iteration   11 completed
     3.616 seconds for this iteration
     value of l: 191
     value of phi before algorithm operator: 0.64192
     value of phi after algorithm operator:  2.09414
algorithm executed in iteration   12
     2.838 seconds for the execution of the algorithm
current epsilon (KL distance) = 95173.6
reconstruction completed after iteration   12
     3.922 seconds for this iteration
    43.835 seconds for all iterations
    43.992 seconds used for processing command exec


<*>

<#> END
```

A.11.3 SNARK14 RPRTsuperiorization file

```
superiorization reporting output
    iter         l     phi pre-algorithm      phi post-algorithm
     1          15           0.000000000           3.144667097
     2          31           0.429607167           1.650648486
     3          47           0.706118937           2.498788058
     4          63           0.564997126           2.145424127
     5          79           0.577604758           2.150495034
     6          95           0.579469392           2.110487832
     7         111           0.581496172           2.086655912
     8         127           0.587931709           2.075449463
     9         143           0.598033798           2.072829250
    10         159           0.610951264           2.076351897
    11         175           0.625866008           2.084050951
    12         191           0.641919981           2.094143205
```

Appendix B

FILE11: A TOOL FOR ENTERING USER DATA

SNARK14 is capable of reconstructing objects using user-supplied data. To do that, you must first create a file11 containing the user data. That file is an ASCII file consisting of the CREATE command Sets 1, 2, 3, 5, 6, and 7 (see Section 5.3.3) followed by the user data. In Set 6 be sure to use the USER option of the RAYS command. Each projection of the user data starts a command line containing the projection angle in both radians and degrees. These angles must be in the same order and agree with the angles specified in CREATE command Set 6. The next lines contain the projection data. The order of the rays within a projection is illustrated in Figures 2.1 and 2.2. The SNARK14 program reads lines until it finds user-rays (the values specified on the RAYS USER command line) values. This is repeated for each projection angle. The SNARK14 commands to reconstruct images from this data should omit the CREATE sequence. The first commands should be:

```
PICTURE RECONSTRUCTION nelem pixel-size
PROJECTION REAL
```

where `nelem` and `pixel-size` describe the reconstruction geometry. These are followed the commands needed for the desired reconstruction algorithm and outputs.

Appendix C

STRUCTURE OF PRJFIL AND RECFIL

Both prjfil and recfil consist of XML headers followed by binary data. The XML header files are described by schema files PRJFIL.xsd and RECFIL.xsd, respectively. Familiarity with XML and XSD schemas is necessary to understand these files. There are many books that can provide that understanding. Users without such understanding can still use files prjfil and recfil, by creating them using the standard SNARK14 commands PICTURE and PROJECTION (Sections 5.4 and 5.5). For those users who wish to create such files themselves, the schema files can be obtained by the following sequence of clicks (starting on the Desktop of the Virtual Machine):
Computer⇒usr⇒local⇒snark14⇒src⇒snark14⇒schema

C.1 PRJFIL.xsd

```
<?xml version = "1.0" ?>

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

           <!-- documentation of schema file -->
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      This is a SNARK09 prjfil schema file.
      See PRJFIL.xml for an instance file of this type,
      it will clear up how such a file should look.
      Version:  $Revision: 1.2 $
      Last Updated:  $Date: 2008/09/06 22:22:46 $
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="DIG_DATA">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="main_header">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
              <xsd:element name="dimensions">
                <xsd:complexType>  <!-- dimension is the vector (USRAYS, 1, 1) -->
```

```
      <xsd:attribute name="x" type="oddPositiveInteger" use="required"/>
      <xsd:attribute name="y" type="xsd:positiveInteger" fixed="1"/>
      <xsd:attribute name="z" type="xsd:positiveInteger" fixed="1"/>
    </xsd:complexType>
  </xsd:element>  <!-- dimensions -->
  <xsd:element name="sampling_rate_x">
    <xsd:complexType>  <!-- sampling_rate_x is the vector (PINC, 0, 0) -->
      <xsd:attribute name="x" type="positiveReal" use="required"/>
      <xsd:attribute name="y" type="nonNegativeReal" fixed="0.0"/>
      <xsd:attribute name="z" type="nonNegativeReal" fixed="0.0"/>
    </xsd:complexType>
  </xsd:element>  <!-- sample_rate_x -->
  <xsd:element name="sampling_rate_y">
    <xsd:complexType>  <!-- sampling_rate_y is the vector (0, 0, 0) -->
      <xsd:attribute name="x" type="nonNegativeReal" fixed="0.0"/>
      <xsd:attribute name="y" type="nonNegativeReal" fixed="0.0"/>
      <xsd:attribute name="z" type="nonNegativeReal" fixed="0.0"/>
    </xsd:complexType>
  </xsd:element>  <!-- sampling_rate_y -->
  <xsd:element name="sampling_rate_z">
    <xsd:complexType>  <!-- sampling_rate_z it the vector (0, 0, 0) -->
      <xsd:attribute name="x" type="nonNegativeReal" fixed="0.0"/>
      <xsd:attribute name="y" type="nonNegativeReal" fixed="0.0"/>
      <xsd:attribute name="z" type="nonNegativeReal" fixed="0.0"/>
    </xsd:complexType>
  </xsd:element>  <!-- sampling_rate_z -->
</xsd:sequence>
<xsd:attribute name="type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="SNARK09 prjfil"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- type -->
<xsd:attribute name="title" type="xsd:string" use="required"/>
<xsd:attribute name="channels" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:pattern value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- channels -->
<xsd:attribute name="number_of_arrays" type="xsd:positiveInteger"
    use="required"/>
<xsd:attribute name="number_of_array_sets" type="xsd:positiveInteger"
    use="required"/>
<xsd:attribute name="endian" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="little|big"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- endian -->
<xsd:attribute name="value_type" use="required">
```

```
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="real"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- value_type -->
      <xsd:attribute name="data_type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="double"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- data_type -->
      <xsd:attribute name="data_format" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="binary"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- data_format -->
      <xsd:attribute name="grid_type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="SC"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- grid_type -->
      <xsd:attribute name="basis_function" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="voronoi"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- basis_function -->
      <xsd:attribute name="unit" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="unspecified"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>  <!-- unit -->
      <xsd:attribute name="other_unit" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>  <!-- main_header -->
  <xsd:element name="application_header">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
        <xsd:element name="geometric_header">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="divergent">
                <xsd:complexType>
                  <xsd:sequence>
```

```
                <xsd:element name="detector_type">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                      <xsd:pattern value="tangent|arc"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>  <!-- detector_type -->
              </xsd:sequence>
              <xsd:attribute name="radius" type="positiveReal"
                  use="required"/>  <!-- value of RADIUS -->
              <xsd:attribute name="source_detector_distance"
                  type="positiveReal" use="required"/>
              <!-- value of STOD -->
            </xsd:complexType>
          </xsd:element>  <!-- divergent -->
          <xsd:element name="parallel">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="detector_type">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                      <xsd:pattern value="uniform|variable"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>  <!-- detector_type -->
                <xsd:element name="projection_type">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                      <xsd:pattern value="strip|line"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>  <!-- projection_type -->
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>  <!-- parallel -->
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>   <!-- geometric_header -->
    <xsd:element name="noise_header">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="quantum" minOccurs="0" maxOccurs="1">
            <xsd:complexType>
              <xsd:attribute name="mean" type="xsd:decimal"
                  use="required"/>   <!-- value of QUANMN -->
              <xsd:attribute name="calibration_measurement"
                  type="xsd:decimal" use="required"/>
                  <!-- value of QUANCM -->
              <xsd:attribute name="gantry_arrangement" use="required">
                  <!-- value of QUANIN -->
                <xsd:simpleType>
                  <xsd:restriction base="xsd:integer">
                    <xsd:pattern value="[1-4]"/>
                  </xsd:restriction>
```

```
            </xsd:simpleType>
          </xsd:attribute>   <!-- gantry_arrangement -->
        </xsd:complexType>
      </xsd:element>      <!--quantum -->
      <xsd:element name="scatter" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="peak" type="positiveReal"
              use="required"/>   <!-- value of SCTNPK -->
          <xsd:attribute name="width" type="positiveReal"
              use="required"/>   <!-- value of SCTNWD -->
        </xsd:complexType>
      </xsd:element>   <!-- scatter -->
      <xsd:element name="additive" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="mean" type="xsd:decimal"
              use="required"/>   <!-- value of ADDNMN -->
          <xsd:attribute name="standard_deviation"
              type="xsd:decimal" use="required"/>
              <!-- value of ADDNSD -->
        </xsd:complexType>
      </xsd:element>   <!-- additive -->
      <xsd:element name="multiplicative" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="mean" type="xsd:decimal"
              use="required"/>   <!-- value of MULTNMN != 0 -->
          <xsd:attribute name="standard_deviation"
              type="xsd:decimal" use="required"/>
              <!-- value of MULTNSD -->
        </xsd:complexType>
      </xsd:element>   <!-- multiplicative -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>   <!-- noise_header -->
<xsd:element name="spectrum_header">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="energy_level" minOccurs="1" maxOccurs="7">
        <xsd:complexType>  <!-- should occur number_of_energy times -->
          <xsd:attribute name="energy" type="xsd:integer"
              use="required"/>   <!-- values of -->
          <xsd:attribute name="ratio" type="nonNegativeReal"
              use="required"/>  <!-- values of PERCENT/100 -->
          <xsd:attribute name="background" type="xsd:decimal"
              use="required"/>  <!-- values of backgr -->
        </xsd:complexType>
      </xsd:element>   <!-- energy_level -->
    </xsd:sequence>
    <xsd:attribute name="number_of_energies" use="required">
      <xsd:simpleType>  <!-- value of NERGY -->
        <xsd:restriction base="xsd:integer">
          <xsd:pattern value="[1-7]"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>   <!-- number_of_energies -->
```

```
              </xsd:complexType>
            </xsd:element>   <!-- spectrum_header -->
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>  <!-- application_header -->
      <xsd:element name="array_set_header" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
            <xsd:element name="parameters" type="xsd:decimal"/>
            <!-- angle in radians -->
            <xsd:element name="array_header" minOccurs="1" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="enumeration_number"
                    type="xsd:nonNegativeInteger" use="required"/>
              </xsd:complexType>
            </xsd:element>  <!-- array_header -->
          </xsd:sequence>
          <xsd:attribute name="type" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:pattern value="projection"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>  <!-- type -->
          <xsd:attribute name="title" type="nonNegativeInteger" use="required"/>
        </xsd:complexType>
      </xsd:element>  <!-- array_set_header -->
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>  <!-- DIG_DATA -->

           <!-- type definitions for PRJFIL schema -->
<xsd:simpleType name="nonNegativeReal">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0.0"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="positiveReal">
  <xsd:restriction base="xsd:decimal">
    <xsd:minExclusive value="0"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="nonNegativeInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
  <xsd:simpleType name="oddPositiveInteger">
    <xsd:restriction base="xsd:integer">
      <xsd:pattern value="[0-9]*[13579]"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

C.2 RECFIL.xsd

```
<?xml version = "1.0" ?>

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

            <!-- documentation of schema file -->
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      This is an SNARK09 recfil schema file.
      See RECFIL.xml for an instance file of this type,
      it will clear up how such a file should look.
      Version:  $Revision: 1.2 $
      Last Updated:  $Date: 2008/09/06 22:22:46 $
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="DIG_DATA">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="main_header">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
              <xsd:element name="dimensions">
                <xsd:complexType>  <!-- dimensions is vector (NELEM, NELEM, 1) -->
                  <xsd:attribute name="x" type="oddPositiveInteger" use="required"/>
                  <xsd:attribute name="y" type="oddPositiveInteger" use="required"/>
                  <xsd:attribute name="z" type="oddPositiveInteger" fixed="1"/>
                </xsd:complexType>
              </xsd:element>  <!-- dimensions -->
              <xsd:element name="sampling_rate_x">
                <xsd:complexType>  <!-- sampling_rate_x is vector (PIXSIZ, 0, 0) -->
                  <xsd:attribute name="x" type="positiveReal" use="required"/>
                  <xsd:attribute name="y" type="nonNegativeReal" fixed="0"/>
                  <xsd:attribute name="z" type="nonNegativeReal" fixed="0"/>
                </xsd:complexType>
              </xsd:element>  <!-- sampling_rate_x -->
              <xsd:element name="sampling_rate_y">
                <xsd:complexType>  <!-- sampling_rate_y is vector (0, PIXSIZ, 0) -->
                  <xsd:attribute name="x" type="nonNegativeReal" fixed="0"/>
                  <xsd:attribute name="y" type="positiveReal" use="required"/>
                  <xsd:attribute name="z" type="nonNegativeReal" fixed="0"/>
```

```
    </xsd:complexType>
  </xsd:element>  <!-- sampling_rate_y -->
  <xsd:element name="sampling_rate_z">
    <xsd:complexType>  <!-- sampling_rate_z is vector (0, 0, 0) -->
      <xsd:attribute name="x" type="nonNegativeReal" fixed="0"/>
      <xsd:attribute name="y" type="nonNegativeReal" fixed="0"/>
      <xsd:attribute name="z" type="nonNegativeReal" fixed="0"/>
  </xsd:complexType>
</xsd:element>  <!-- sampling_rate_z -->
</xsd:sequence>
<xsd:attribute name="type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="SNARK09 recfil"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- type -->
<xsd:attribute name="title" type="xsd:string" use="required"/>
<xsd:attribute name="channels" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:pattern value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- channels -->
<xsd:attribute name="number_of_arrays" type="xsd:positiveInteger"
    use="required"/>
<xsd:attribute name="number_of_array_sets" type="xsd:positiveInteger"
    use="required"/>
<xsd:attribute name="endian" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="little|big"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- endian -->
<xsd:attribute name="value_type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="real"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- value_type -->
<xsd:attribute name="data_type" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="double"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>  <!-- data_type -->
<xsd:attribute name="data_format" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="binary"/>
```

```
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>  <!-- data_format -->
    <xsd:attribute name="grid_type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="SC"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>  <!-- grid_type -->
    <xsd:attribute name="basis_function" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="voronoi"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>  <!-- basis_function -->
    <xsd:attribute name="unit" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="unspecified"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>  <!-- unit -->
    <xsd:attribute name="other_unit" use="optional">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern value=""/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>  <!-- other_unit -->
  </xsd:complexType>
</xsd:element>  <!-- main_header -->
<xsd:element name="array_set_header" minOccurs="1" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
      <xsd:element name="parameters" type="xsd:string"/>
      <xsd:element name="array_header" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="comments" type="xsd:string" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="enumeration_number"
              type="xsd:nonNegativeInteger" use="required"/>
        </xsd:complexType>
      </xsd:element>  <!-- array_header -->
    </xsd:sequence>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:pattern
value="PHAN|BACK|CONV|RFL|FOUR|DCON|ART|MART|QUAD|SIRT|EMAP|LINO|ALGP|ALGB"/>
        </xsd:restriction>
```

```
              </xsd:simpleType>
            </xsd:attribute>  <!-- type -->
            <xsd:attribute name="title" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string"/>
              </xsd:simpleType>
            </xsd:attribute>  <!-- title -->
          </xsd:complexType>
        </xsd:element>  <!-- array_set_header -->
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>  <!-- DIG_DATA -->

           <!-- type definitions for RECFIL schema -->
  <xsd:simpleType name="nonNegativeReal">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="positiveReal">
    <xsd:restriction base="xsd:decimal">
      <xsd:minExclusive value="0"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="oddPositiveInteger">
    <xsd:restriction base="xsd:integer">
      <xsd:pattern value="[0-9]*[13579]"/>
    </xsd:restriction>
  </xsd:simpleType>


</xsd:schema>
```

C.3 Creating a prjfil or recfil

Some users may wish to create their own prjfil or recfil for use with snark14. Those files consist of an XML header following the appropriate schema, PRJFIL.xsd or RECFIL.xsd, immediately followed by the binary data. The data file is a raw binary file containing the data in any IEEE 754 representation in either big or little endian byte order. There are several ways in which to create the XML header. Two possibilities are:

1. extract the header from an existing prjfil or recfil and edit it to match the binary data,
2. create the header from scratch following the schema.

In either case, the header needs to be checked for validity. Although every effort has been made to ensure that SNARK14 will accept all valid XML headers, some invalid headers are known to cause SNARK14 to terminate in unexpected ways. One possible way to validate a user-generated XML header is to use an *XML schema validator*. In the prjfil case, another possibility is to use the CREATE sequence (see Section 5.3) with no objects to describe the data collection geometry and then the PROJECTION REAL command (see Section 5.5) to create a prjfil and then run the command

```
# tr -d '\000' < prjfil > prjfil.xml
```

to strip off the data.
If "data" is the file containing the projection data in the specified format, then they can be combined into a prjfil by the command

```
# cat prjfil.xml data > prjfil
```

A similar approach can be used to create a recfil from its XML header data.

Appendix D

SNARK14 SOURCE CODE AND SCRIPTS

D.1 SNARK14 source code

The SNARK14 source code can be obtained by the following sequence of clicks (starting on the Desktop of the Virtual Machine):
Computer⇒usr⇒local⇒snark14⇒src⇒snark14⇒snark

D.2 snark14GetExamples

Executing the snark14GetExamples script in a directory will create a copy of the inputs needed to recreate the eleven examples of Appendix A in a directory of the user's choosing. The default is to create a directory called snark14Examples in the current directory and place the examples there. Each example includes a script called run that will execute SNARK14 and recreate the example. In addition, Example 2 contains a script called build that is used to recreate the SNARK14 executable containing the user-defined algorithm and Example 9 includes files needed for the experimenter run of that example.

D.3 snark14GetUserDefined

The snark14GetUserDefined script is located in the same directory where SNARK14 was installed, which, under normal circumstances, is Root/usr/local/snark14/bin. To get into Root you should click on Computer on the Virtual Machine Desktop. To use the script, you should first create a directory, here we refer to it as the current directory. All other directories referred to below are sub-directories of the current directory. Executing the script in the current directory will create a directory that can be used as a starting point for a user-written algorithm, one of the user-written functions described in the ART (Section 7.6) or QUADRATIC (Section 7.9) algorithms, or a user-defined FOM (Section 8.2.6.7). The default is to create a directory called snark14UserDefined in the current directory and place the skeleton files there. The src subdirectory contains the skeletons for the user-written algorithms: five algorithms that use pixels (ALP1, ALP2, ALP3, ALP4, ALP5), five algorithms that use blobs (ALB1, ALB2, ALB3, ALB4, ALB5), the skeletons for the ART and QUADRATIC user-written functions, and the skeleton for the user-defined figures of merit USR1, USR2, USR3, USR4 and USR5. These routines are found in their respective filenames (i.e., alp*.cpp, alp*.h, alb*.cpp, alb*.h, art*.cpp, quad*.cpp, user*.cpp and user*.h). The user may also place any additional .h and .cpp files into the src subdirectory that contain user-written auxiliary subroutines. For each .cpp file that the user adds, a new object with the same name needs to be added manually to the list of objects in the Makefile located inside the src subdirectory. In the directory snark14UserDefined the script snark14UserDefinedBuild will compile the source code and place a new SNARK14 executable in the bin

directory. In order to minimize confusion between this executable and the standard SNARK14 executable, this version is named snark14UserDefined. Enter the command:

```
# ./bin/snark14UserDefined
```

while in the snark14UserDefined directory to execute this modified version of SNARK14.

Warning: The program snark14Input has a special provision that if it is run from a directory containing a bin/snark14UserDefined executable, it will use that version of SNARK14 instead of the standard version.

Warning: The files used by snark14GetUserDefined are located in the SNARK14 installation directory. Do not create your user algorithm there. Those files should remain unmodified so that every user of snark14GetUserDefined gets untouched copies of the skeleton files.

Warning: The files exalg.cpp, stopex.cpp, term.h and termtest.h in the snark14UserDefined directory should remain in their original form and should not be modified by the user.

D.4 Building (a modified version of) SNARK14 from source code

The content of this subsection is intended for users who are familiar with C++ programming, building large projects and the make build automation tool; modifications made without such knowledge to the files described in this subsection may lead to unexpected results.

The following procedure creates a working copy of the programs snark14, snark14Input, and snark14Display as they have been built in the original distribution of SNARK14. To build these programs from source code, a user needs to invoke the following script

```
# snark14GetTrunk
```

The script snark14GetTrunk will create a directory, named snark14Trunk, containing the Makefiles and source-code files necessary for building the executable SNARK14 programs. When invoked, the script will ask for a location where to create the directory snark14Trunk; if the user does not provide a location, the script will create the directory snark14Trunk in the location where the script was invoked from.

The snark14Trunk directory contains a Makefile file already configured to succesfully build the executable SNARK14 programs using the make build automation tool. Upon invoking the snark14GetTrunk script, the directory snark14Trunk will also contain the following three directories: src, include, and tools. The src directory contains files with the source code while the include directory contains header files necessary for building the executable SNARK14 program (i.e., snark14). In turn, the tools directory contains the directories Display and Input containing the source code for building snark14Display and snark14Input, respectively.

To build the executable SNARK14 programs, the user should get inside the recently created directory snark14Trunk and issue the following command

```
# make
```

This process will take some time to finish. In a normal case, after the building process is complete the process has created three extra directories inside the snark14Trunk directory: bin, build, and lib. The build directory contains the object-code files for building the final programs and, typically, a user would not want to access this directory. The lib directory contains the libraries libDIGFile, libDIGFileSnark, and libDIGRand; these are libraries containing routines necessary for the proper operation of SNARK14 programs and in normal circumstances a user would not want to modify neither the source code nor the libraries for these routines. It is worth noticing that similar libraries are already part of the original SNARK14 distribution and they are installed in the directory /usr/local/snark14/lib. Finally, the directory bin contains the executable SNARK14 programs. One may invoke any of the recently created programs by accessing the bin directory and invoking either `snark14`, `snark14Input`, or `snark14Display`.

An important note: A user may want to modify only the routines in the directory src/snark and, thus, is not necessary to modify the system-wide LD_LIBRARY_PATH variable. However, if a user modifies the behavior of any of the routines making up the libraries libDIGFile, libDIGFileSnark, or libDIGRand, this variable needs to point to the directory snark14Trunk/lib instead of the usual /usr/local/snark14/lib when executing any of the newly built executable SNARK14 programs.

Bibliography

[1] jSNARK: A Programming System for the Reconstruction of 2D and 3D Images from their Projections. http://jsnark.sourceforge.net/, accessed June 17, 2017. 12

[2] A.H. Andersen and A.C. Kak. Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm. *Ultrason. Imag.*, 6:81–94, 1984. 95

[3] R.S. Anderssen and G.H. Golub. Richardson's non-stationary matrix iterative procedure. Technical Report 304, Department of Computer Science, Stanford University, Stanford, CA, 1972. 99

[4] E. Artzy, T. Elfving, and G.T. Herman. Quadratic optimization for image reconstruction II. *Comput. Vision Graph.*, 11:242–261, 1979. 97

[5] F. B. Bouallègue, J. F. Crouzet, and D. Mariano-Goulart. A heuristic statistical stopping rule for iterative reconstruction in emission tomography. *Annals of Nuclear Medicine*, 27:84–95, 2013. 47

[6] R.N. Bracewell and A.C. Riddle. Inversion of fan-beam scans in radio astronomy. *Astrophys*, 150:427–434, 1967. 79

[7] E.A. Brigham. *The Fast Fourier Transform*. Prentice–Hall, 1974. 77

[8] J.A. Browne and G.T. Herman. Software for evaluating image reconstruction algorithms. In *Conf. Rec. 1994 Nucl. Sci. Symp. Med. Imag. Conf.*, pages 1940–1944, Norfolk, VA, November 3-5 1994. Published by IEEE, 1995. 107

[9] J.A. Browne and G.T. Herman. Computerized evaluation of image reconstruction algorithms. *Internat. J. Imag. Systems Tech.*, 7:256–267, 1996. 120

[10] J.A. Browne, G.T. Herman, and D. Odhner. SNARK93 – a programming system for image reconstruction from projections. Technical Report MIPG198, Dept. of Radiol., Univ. of Pennsylvania, Philadelphia, PA, 1993. 8

[11] T. Chang and G.T. Herman. A scientific study of filter selection for a fan-beam convolution reconstruction algorithm. *SIAM J. Appl. Math*, 39:83–105, 1980. 88

[12] D. Chesler and S.J. Riederer. Ripple suppression during reconstruction in transverse tomography. *Phy. Med. Biol*, 20:632–636, 1975. 79

[13] P. Edholm and G.T. Herman. Linograms in image reconstruction from projections. *IEEE Trans. Med. Imag.*, 6:301–307, 1987. 105

[14] P. Edholm, G.T. Herman, and D.A. Roberts. Image reconstruction from linograms: implementation and evaluation. *IEEE Trans. Med. Imag.*, 7:239–246, 1988. 105

[15] S.S. Furuie, G.T. Herman, T.K. Narayan, P. Kinahan, J.S. Karp, R.M. Lewitt, and S. Matej. A methodology for testing for statistically significant differences between fully 3–D PET reconstruction algorithms. *Phys. Med. Biol.*, 39:341–354, 1994. 107, 120

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Professional Computing Series. Addison-Wesley, 1994. 61

[17] P. Gilbert. Iterative methods for three-dimensional reconstruction of an object from projections. *J. Theoret. Biol.*, 36:105–117, 1972. 102

[18] R. Gordon, R. Bender, and G.T. Herman. Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. *J. Theoret. Biol.*, 29:471–482, 1970. 95

[19] R. Gordon and G.T. Herman. *Three dimensional reconstruction from projections: a review of algorithms In International Review of Cytology.* Bourne and J. F. Danielli, Eds. Academic Press, 1974. 85, 95

[20] C. Hastings Jr. *Approximations for Digital Computers.* Princeton University Press, 1955. 79

[21] G. T. Herman, E. Garduño, R. Davidi, and Y. Censor. Superiorization: An optimization heuristic for medical physics. *Medical Physics*, 39:5532–5546, 2012. 9, 48, 50

[22] G.T. Herman. Reconstruction of binary patterns from a few projections. In A. Günther, B. Levrat, and H. Lipps, editors, *International Computing Symposium 1973*, pages 371–379, Amsterdam, 1974. North-Holland. 91

[23] G.T. Herman. A relaxation method for reconstructing objects from noisy x-rays. *Math. Programming*, 8:1–19, 1975. 91

[24] G.T. Herman. Correction for beam hardening in computed tomography. *Phys. in Med. Biol.*, 24:81–106, 1979. 29, 30

[25] G.T. Herman. Demonstration of beam hardening correction in computerized reconstruction of the head. *J. Comput. Assist. Tomo.*, 3:373–378, 1979. 41

[26] G.T. Herman, editor. *Image Reconstruction from Projections: Implementation and Applications.* Springer-Verlag, 1979. 8

[27] G.T. Herman. On modifications to the algebraic reconstruction techniques. *Comp. in Biol. Med*, 9:271–276, 1979. 91

[28] G.T. Herman. *Geometry of Digital Spaces.* Birkhäuser, 1998. 14

[29] G.T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections (2nd Edition).* Springer, 2009. 7, 8, 29, 32, 44, 81, 83, 85, 88, 201

[30] G.T. Herman, A.R. De Pierro, and N. Gai. On methods for maximum a posteriori image reconstruction with a normal prior. *J. Visual Comm. Image Represent.*, 3:316–324, 1992. 104, 105

[31] G.T. Herman, H. Hurwitz, A. Lent, and H.-P. Lung. On the Bayesian approach to computerized image reconstruction. *Information and Control*, 42:42, 1979. 91

[32] G.T. Herman, A.V. Lakshminarayanan, and A. Naparstek. Convolution reconstruction techniques for divergent beams. *Comp. Biol. Med.*, 6:259–271, 1976. 88

[33] G.T. Herman and A. Lent. Iterative reconstruction algorithms. *Comp. Biol. Med*, 6:273– 294, 1976. 91, 95, 102

[34] G.T. Herman and A. Lent. Quadratic optimization for image reconstruction part i. *Comp. Graph. Image Proc*, 5:319–332, 1976. 97, 98

[35] G.T. Herman and A. Lent. A family of iterative quadratic optimization algorithms for pairs of inequalities, with applications in diagnostic radiology. *Math Progr. Studies*, 9:15–29, 1978. 91

[36] G.T. Herman, A. Lent, and P. Lutz. Relaxation methods for image reconstruction. *Comm. Assoc. Comp. Mach.*, 21:152–158, 1978. 91

[37] G.T. Herman, R.M. Lewitt, D. Odhner, and S.W. Rowland. SNARK89 – a programming system for image reconstruction from projections. Technical Report MIPG160, Dept. of Radiol., Univ. of Pennsylvania, Philadelphia, PA, 1989. 8

[38] G.T. Herman and L.B. Meyer. Algebraic reconstruction techniques can be made computationally efficient. *IEEE Trans. Med. Imag.*, 12:600–609, 1993. 8, 43, 91

[39] G.T. Herman and A. Naparstek. Fast image reconstruction based on a Radon inversion formula appropriate for rapidly collected data. *SIAM J. Appl. Math.*, 33:511–533, 1977. 88, 89, 90

[40] G.T. Herman and D. Odhner. Evaluation of reconstruction algorithms. In G.T. Herman, A.K. Louis, and F. Natterer, editors, *Mathematical Methods in Tomography*, pages 215–228. Springer-Verlag, 1991. 107

[41] G.T. Herman and D. Odhner. Performance evaluation of an iterative image reconstruction algorithm for positron emission tomography. *IEEE Trans. Medical Imaging*, 10:336–346, 1991. 107, 117, 118, 120, 189

[42] G.T. Herman and S.W. Rowland. SNARK77 – a programming system for image reconstruction from projections. Technical Report MIPG130, Dept. of Comp. Sci., SUNY at Buffalo, Amherst, NY, 1978. 8

[43] G.T. Herman and K.T.D. Yeung. Evaluators of image reconstruction algorithms. *Internat. J. Imag. Syst. Tech.*, 1:187–195, 1989. 107, 120

[44] G.T. Herman, Guest Ed. Special issue on computerized tomography. *Proc. IEEE*, 71:291–435, 1983. 8

[45] R.H. Huesman, G.T. Gullberg, W.L. Greenberg, and T.F. Budinger. *RECLBL Library Users Manual. Donner algorithms for reconstruction tomography.* Lawrence Berkeley Laboratory, University of California, Berkeley, CA, 1977. 8

[46] M. Jiang and G. Wang. Convergence of the simultaneous algebraic reconstruction technique (SART). *IEEE Trans. Image Process.*, 12:957–961, 2003. 48, 95

[47] A.C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging.* SIAM, 2001. 95

[48] J. Klukowska, R. Davidi, and G.T. Herman. SNARK09 - A software package for the reconstruction of 2D images from 1D projections. *Comput. Meth. Prog. Bio.*, 110:424–440, 2013. 7, 8, 12

[49] D.E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley Professional, 3rd edition, 1997. 80

[50] K. Kouris, H. Tuy, A. Lent, G.T. Herman, and R.M. Lewitt. Reconstruction from sparsely sampled data by art with interpolated rays. *IEEE Trans. Med. Imag.*, MI-1:161–167, 1982. 33

[51] A.V. Laksminarayanan and A. Lent. Methods of least squares and sirt in reconstruction. *J. Theor. Biol*, 76:267–295, 1979. 102

[52] A. Lent. A convergent algorithm for maximum entropy image restoration, with a medical x-ray application. image analysis and evaluation. In *SPSE Conference Proceedings*, pages 249–257, Toronto, Canada, 1976. R. Shaw, Ed. 95

[53] E. Levitan and G.T. Herman. A maximum a posteriori probability expectation maximization algorithm for image reconstruction in emission tomography. *IEEE Trans. Med. Imag.*, 6:185–192, 1987. 50

[54] R.M. Lewitt. Multidimensional digital image representations using generalized Kaiser-Bessel window functions. *J. Opt. Soc. Amer. A*, 7:1834–1846, 1990. 8, 14

[55] P. Lutz. Fourier image reconstruction incorporating three sample interpolation techniques. Technical Report 104, Department of Computer Science, State University of New York at Buffalo, Amherst, NY, 1975. 85

[56] R. Marabini, G.T. Herman, and J.M. Carazo. 3D reconstruction in electron microscopy using ART with smooth spherically symmetric volume elements (blobs). *Ultramicrosc.*, 72:53–65, 1998. 8

[57] R.M. Mersereau. Direct Fourier transform techniques in 3-d image reconstruction. *Comput. Biol. Med*, 6:247–258, 1976. 85

[58] R.F. Mould. *Introductory Medical Statistics*. CRC Press, 3rd edition, 1998. 120

[59] T.K. Narayan and G.T. Herman. Prediction of human observer performance by numerical observers: An experimental study. *J. Opt. Soc. Am. A*, 16:679–693, 1999. 8, 118

[60] B.E. Oppenheim. Reconstruction tomography from incomplete projections. In M.M. Ter-Pogossian, M.E. Phelps, G.L. Brownell, J.R. Cox, Jr., D.O. Davis, and R.G. Evens, editors, *Reconstruction Tomography in Diagnostic Radiology and Nuclear Medicine*, pages 155–183. University Park Press, 1977. 74

[61] T.M. Peters. *Image reconstruction from projections. Ph.D. dissertation.* PhD thesis, Department of Electrical Engineering, University of Canterbury, Christchurch, New Zealand, 1973. 78

[62] R.J. Polge and B.K. Bhagavan. Efficient fast Fourier transform programs for arbitrary factors with one step loop unscrambling. *IEEE Trans. Comps.*, C-25:534–539, 1994. 77

[63] G.N. Ramachandran and A.V. Lakshminarayanan. Three-dimensional reconstruction from radiographs and electron micrographs: application of convolutions instead of Fourier transforms. *Proc. Nat. Acad. Sci. U.S.A.*, 68:2236–2240, 1971. 79

[64] S.W. Rowland. Computer implementation of image reconstruction formulas. In G.T. Herman, editor, *Image Reconstruction from Projections: Implementation and Applications*, pages 9–79. Springer-Verlag, 1979. 79, 83, 84

[65] H.E. Schaffer. Algorithm 369: generator of random numbers satisfying the Poisson distribution. *CACM*, 13:49, 1970. 80

[66] L.A. Shepp and B.F. Logan. The Fourier reconstruction of a head section. *IEEE Trans. Nucl. Sci.*, NS-21:21–43, 1974. 79

[67] L.A. Shepp and Y. Vardi. Maximum likelihood reconstruction in positron emission tomography. *IEEE Trans. Med. Imag.*, 1:113–122, 1982. 104, 105

[68] P.R. Smith, T.M. Peters, and R.H.T. Bates. Image reconstruction from finite numbers of projections. *J. Phys. A: Math., Nucl. Gen.*, 6:361–382, 1973. 84

[69] J. Wimp. Polynomial approximations to integral transforms. *Math. of Computation*, 15:174–178, 1961. 79

Index