USERS GUIDE FOR ASPIRE
3D IMAGE RECONSTRUCTION SOFTWARE

Jeffrey A. Fessler

COMMUNICATIONS & SIGNAL PROCESSING LABORATORY
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Jul. 1997
Revised January 20, 2004

Technical Report No. 310
Approved for public release; distribution unlimited.

Users guide for ASPIRE 3D image reconstruction software

Jeffrey A. Fessler
4240 EECS, University of Michigan, Ann Arbor, MI 48109-2122
email: fessler@umich.edu
phone: 734-763-1434

January 20, 2004

Technical Report # 310
Communications and Signal Processing Laboratory
Dept. of Electrical Engineering and Computer Science
The University of Michigan

Abstract

This document is a users guide for the iterative 3D image reconstruction portion of the ASPIRE software suite. This software is available from the author's web site.

Work supported in part by NIH grants CA-60711, CA-54362, and CA-87955, NSF grant BES-9982349, and the Whitaker Foundation.

I. INTRODUCTION

This document is a users guide for the 3D image reconstruction subset of the ASPIRE software suite. When this document was first written in 1997, the *only* 3D reconstruction method available was PWLS with a very limited choice of system models. Now maximum-likelihood and penalized-likelihood image reconstruction for the Poisson emission and transmission problems are also available, which will probably be of more interest to most users, as well as a larger set of system models and regularization methods.

Readers should first be familiar with the ASPIRE documentation for 2D reconstruction [1].

This documentation is certainly incomplete. The best way to find out what the "latest and greatest" options are is to execute the programs with too few arguments, and to examine the built-in documentation that is displayed. For example, running `i` with no arguments will show *all* of the iterative methods that are available, both 2d and 3d, and several other utilities. This documentation focuses on the following three reconstruction methods

- `i empl3`
  Emission tomography (PET and SPECT) under the usual Poisson model and some variations thereof.
- `i trpl3`
  Transmission tomography (X-ray CT or radionuclide transmission scans) under the usual Poisson model and some variations thereof.
- `i pwls3`
  penalized weighted least-squares (PWLS) image reconstruction. Appropriate for non-Poisson measurements. Not recommended if either of the preceding two methods are suitable.

A. Common Considerations

Roughly speaking, all of the 3D reconstruction methods are based on variations of the model

$$\boldsymbol{y} = \boldsymbol{G}\,\boldsymbol{x} + \text{noise},$$

where

- $\boldsymbol{x}$ is the unknown image (volume) to be determined,
- $\boldsymbol{y}$ is the measured projection data, and
- $\boldsymbol{G}$ is the system matrix, which is specified as described in §VI.

In all of ASPIRE's 3D reconstruction methods, $\boldsymbol{x}$ corresponds to a lexicographically ordered $n_x \times n_y \times n_z$ array with the x dimension (image column index) varying fastest and the n_z dimension (slices) varying slowest, as is standard in imaging. The dimensions of the data $\boldsymbol{y}$ are imaging-system dependent. For most of the 3D system models in ASPIRE, the projection data is organized as a set of n_{view} views of size $n_u \times n_v$, so $\boldsymbol{y}$ corresponds to a lexicographically ordered $n_u \times n_v \times n_{\text{view}}$ array, *i.e.*, a stack of projections views like in conventional SPECT imaging.

The goal is to estimate $\boldsymbol{x}$ from $\boldsymbol{y}$ for the user-specified system model $\boldsymbol{G}$. All three reconstruction methods (EMPL, TRPL, PWLS) are based on minimizing a cost function of the general form

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x})\,, \qquad \Psi(\boldsymbol{x}) = \text{data\_fit}(\boldsymbol{y}, \boldsymbol{G}\,\boldsymbol{x}) + R(\boldsymbol{x}), \tag{1}$$

where data_fit is a functional that quantifies how well a given "guess" $\boldsymbol{x}$ fits the measurements, and $R(\boldsymbol{x})$ is a regularizing penalty function that discourages excessive image roughness, thereby controlling noise. By minimizing $\Psi(\boldsymbol{x})$, one finds an image that "fits the data" (where fit is measured by the first term) but is also not too noisy (where roughness is measured by the penalty term).

B. Regularization methods

Currently, all the "3D" penalty functions implemented in ASPIRE penalize 1st-order pixel differences of the following form:

$$R(\boldsymbol{x}) = \frac{1}{2} \sum_{j=1}^{n_{\text{p}}} \sum_{k=1}^{n_{\text{p}}} w_{jk}\, \psi(x_j - x_k)\,. \tag{2}$$

(The $\frac{1}{2}$ is because each pair is counted twice.) The current choices for $\psi(t)$ are

- the quadratic function $t^2/2$
- the Huber function

$$\psi(t) = \begin{cases} t^2/2, & |t| \leq \delta \\ \delta|t| - \delta^2/2, & |t| > \delta. \end{cases}$$

As part of the command-line arguments to the iterative reconstruction program `i`, the user chooses the type of *penalty* desired.

For emission image reconstruction, I recommend (for now) a quadratic penalty function using the 4 nearest neighbors to each pixel within a plane, and the pixels above and below in the adjacent slices. For this penalty function,

$$w_{jk} = \begin{cases} \beta_x, & j = k \pm 1 \\ \beta_y, & j = k \pm n_x \\ \beta_z, & j = k \pm n_x n_y, \end{cases}$$

roughly speaking. (I ignore edge conditions here; the code does consider them appropriately.) For this penalty with $\beta_x = \beta_y = 2^{-6}$ and $\beta_z = 2^{-7}$, the corresponding *penalty* string looks like

```
3d,-6,-7,quad,5,-
```

For transmission tomography, I recommend (for now) a nonquadratic edge-preserving penalty function such as Huber's potential, using all 8 neighbors in plane and at least the pixels above and below in the adjacent slices. For the Huber potential, one can set δ to a value that is well below the important boundary differences in the attenuation map (but not too small or it can slow down convergence and make "block" attenuation maps). For the choice $\beta_x = 2^{16}$ and $\beta_z = 2^{11}$ and $\delta = 0.002$/mm (assuming all units are in mm), the penalty string would be

```
3d,16,11,huber,6,-,0.002,ih,3
```

Regularization is an active research area in my group, so expect more options in the future, particularly with regards to help in choosing the β's to specify the desired resolution, a 3D extension analogous to that in [2], rather than trial and error to determine β_x and β_z. In the mean time, it may be easiest just to use a fairly small value for the β's and do some post-filtering if additional noise reduction is desired.

C. More on quadratic penalties

Here is another way to write the 1st-order smoothness penalty:

$$R(\boldsymbol{x}) = \beta_x \sum_{\{j \,:\, i_x > 0\}} w_j^{\mathrm{x}} \frac{1}{2}(x_j - x_{j-1})^2 + \beta_y \sum_{\{j \,:\, i_y > 0\}} w_j^{\mathrm{y}} \frac{1}{2}(x_j - x_{j-n_x})^2 + \beta_z \sum_{\{j \,:\, i_z > 0\}} w_j^{\mathrm{z}} \frac{1}{2}(x_j - x_{j-n_x n_y})^2, \quad (3)$$

where $j = i_x + i_y n_x + i_z n_x n_y$, and all indices count from zero as in the C programming language. The parameters $\beta_x, \beta_y, \beta_z$ control the resolution-noise tradeoff.

If $\boldsymbol{D}_n$ denotes the $n - 1 \times n$ 1st-order differencing matrix:

$$\boldsymbol{D}_n = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ & & \ddots & \ddots & \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

and we define

$$\boldsymbol{C}_x = \boldsymbol{I}_{n_z} \otimes \boldsymbol{I}_{n_y} \otimes \boldsymbol{D}_{n_x}, \;\; \boldsymbol{C}_y = \boldsymbol{I}_{n_z} \otimes \boldsymbol{D}_{n_y} \otimes \boldsymbol{I}_{n_x}, \;\; \boldsymbol{C}_z = \boldsymbol{D}_{n_z} \otimes \boldsymbol{I}_{n_y} \otimes \boldsymbol{I}_{n_x},$$

then one can write

$$\boldsymbol{R} = \beta_x \boldsymbol{C}_x{}' \boldsymbol{D}(w_j^{\mathrm{x}}) \boldsymbol{C}_x + \beta_y \boldsymbol{C}_y{}' \boldsymbol{D}(w_j^{\mathrm{y}}) \boldsymbol{C}_y + \beta_z \boldsymbol{C}_y{}' \boldsymbol{D}(w_j^{\mathrm{y}}) \boldsymbol{C}_y,$$

where $\boldsymbol{D}(w_j)$ is a diagonal matrix with diagonal elements $w_1, w_2, \ldots$. This form is useful for analysis, but it does clearly describe how the code is implemented. I forget why I put this in here in 1997, but I left it in since maybe it was important.

II. Utility Operations

A. Forward projections

To generate a simulated noise free data, first create an image file (really a 3D volume) that is $n_x \times n_y \times n_z$. If you have Matlab, and if your version of the software was compiled with the Matlab library and flags, then you can create the image using any function and then use the Matlab `save` command to write it to a file. Otherwise, use an AVS `.fld` file as described in the ASPIRE manual [1].

Before proceeding, try `op range` *image.fld* to check if your file has the right size and values.

Now you need to create the projections of your image, *i.e.*, to compute $\boldsymbol{p} = \boldsymbol{G}\,\boldsymbol{x}$. The command for this is

`i proj3` *proj.out image.in sys_type*

Obviously *image.in* should be your `image.fld` file, and *proj.out* should be the name of the output file where the (noiseless) projections $\boldsymbol{p}$ will be stored. The most important parameter is *sys_type*, which is described in §VI.

B. Backprojection

Sometimes you may also need to compute

$$\boldsymbol{b} = \boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{y}.$$

This multiplication is performed by `i back3`, which has usage:

`i back3` *backimage.out* n_x n_y n_z *views.in wi- sys_type mask*

where *views.in* is $\boldsymbol{y}$, and should have dimensions $n_a \times n_b \times n_z$ or $n_u \times n_v \times n_{\text{view}}$ depending on which system model you are using. The argument *backimage.out* is the file to which $\boldsymbol{b}$ will be written, and will have size $n_x \times n_y \times n_z$. And the *sys_type* argument is the same as described in §VI.

If $\boldsymbol{W} \neq \boldsymbol{I}$, *i.e.*, if you want a weighted backprojection, then the argument *wi* should be the file containing the diagonal elements of $\boldsymbol{W}$, arranged in the same way as *views.in*. The default hyphen will give an unweighted backprojection with $\boldsymbol{W} = \boldsymbol{I}$.

C. Nonuniform Chang attenuation correction factors

To improve the accuracy of SPECT FBP reconstruction, *e.g.*, to get a better initial image for iterative SPECT reconstruction, one can apply Chang's approximate correction method [3, 4]. Here is how to compute those factors

```
i back3 t0 nx ny nz - - sys_type -
    op div0 chang.out - t0 na
```

The first line does a backprojection of a set of uniform projections with unity value, storing the result in the intermediate file `t0`. (The *sys_type* should be of the type `3s` and must specify the attenation map as described in §VI.) The second line divides the intermediate result of the backprojection into the scalar n_a, the number of projection views. The resulting file *chang.out* contains the required factors.

III. EMPL3: 3D EMISSION RECONSTRUCTION

For emission tomography, ASPIRE is based on the following measurement model:

$$Y_i \sim \mathsf{Poisson}\left\{ c_i \sum_{j=1}^{n_\mathrm{p}} g_{ij} x_j + r_i \right\},$$

where r_i is additive background that includes random coincidences and possibly scatter, and c_i includes ray-dependent terms such as detector efficiencies and survival probabilities computed from an attenuation map. The array Y_i, c_i, and r_i must all have the same size, typically $n_u \times n_v \times n_\mathrm{view}$.

The corresponding log-likelihood is

$$L(\boldsymbol{x}) = \sum_{i=1}^{n_\mathrm{d}} h_i([\boldsymbol{G}\,\boldsymbol{x}]_i),$$

where

$$h_i(l) = Y_i \log(c_i l + r_i) - (c_i l + r_i),$$

and its negative is the "data fit" term in (1) for maximum likelihood (ML) and penalized-likelihood emission image reconstruction.

Typing `i empl3` will print the syntax of how to perform 3D emission image reconstruction under the above model. The output should include something like:

```
Usage: empl3 file_out {init|-nx,ny,nz,value} file_yi
        file_ci- ci_scale file_ri- ri_factor
        sys_type file_mask- method
        [saver- flag_obj(0) flag_nonneg(1) pix_max scale_init(0) slices-]
        method: @niter@alg@penalty...
```

(The argument order is fixed.)

The arguments followed by a hyphen "–" are optional; using the hyphen will give sensible defaults.

Here is what each argument means.

- *file_out* is the name of the output image file, dimensions $n_x \times n_y \times n_z$.
- *init* is the name of the initial image file, dimensions $n_x \times n_y \times n_z$.

  For a uniform initial image of value 13 (within mask), use $-n_x,n_y,n_z,13$.

  For regularized algorithms I recommend an FBP initial image (corrected by Chang for SPECT), for convergence rate reasons detailed in [5, 6].
- *file_yi* is the $n_u \times n_v \times n_\mathrm{view}$ file containing $\boldsymbol{y}$.
- *file_ci* contains the c_i's, which are scaled by the *ci_scale* argument before use.

  If *file_ci* is the default hyphen, then the program sets the c_i's to be uniformly the value *ci_scale*.
- *file_ri* and *ri_factor* control the model for the r_i's. As explained in the built-in documentation (type `i empl3`), there are a few possible ways to use *file_ri* and *ri_factor*.

  1. The usual and recommened way is to supply a data-sized file for *file_ri* and set *ri_factor* to 1. The *file_ri* should include the effects of randoms, scatter, crosstalk, room background, etc.
  2. For simulations of randoms-precorrected PET data with a simple *constant* shift, use '`-shift`' for *file_ri* and set *ri_factor* to the desired shift. 'The factor of 2' needed for $2r_i$ must be provided by the user as part of the input shift argumtent. The code just adds *ri_factor* to the y_i's and sets the r_i's to *ri_factor* too.

3. For simulations having constant randoms, use '-' for *file_ri* and set *ri_factor* to be the mean randoms per ray.

   See [1] for more discussion of the shifted Poisson model!

4. Ask me if you are interested in saddlepoint approximations. (Not recommended.)

- *sys_type* is as described in §VI.
- *file_mask* is a $n_x \times n_y \times n_z$ file of zeros and ones indicating the support of the object. User should supply this if possible; the default hyphen varies with projector type. For `2z` the default mask is the support read from the `.wtf`. For `3a` the default mask is a big circle.
- *method* specifies how many iterations of what algorithms using which penalties. See below.
- *saver* should usually be a hyphen. Other options are supported to let you write out intermediate iterations. See the built-in documentation that appears when `i empl3` is typed.
- *flag_obj*: if `1`, will compute $-\Psi(\boldsymbol{x}^{(0)})$ initially and $\Psi(\boldsymbol{x}^{(0)}) - \Psi(\boldsymbol{x}^{(n)})$ every iteration and print. Use `0` except for debugging since this is very computationally expensive.
- *flag_nonneg*: if `1`, enforce nonnegativity constraint $\boldsymbol{x} \geq \boldsymbol{0}$. If `0`, unconstrained.
- *pix_max*: maximum allowable pixel value, which can be useful for transmission images if you know the maximum attenuation coefficient. Use a big number like `1e9` otherwise.
- *scale_init*: If you are *sure* that the initial image is properly scaled, then use `0`. Otherwise, use `1`, and ASPIRE will scale your initial image to best fit the date before iterating. This requires an extra projection operation, so it is best to match scaling of FBP with the $\boldsymbol{G}$ matrix by careful bookkeeping (*i.e.*, preserving counts in the emission case). ASPIRE will print out the scale factor it applied to the initial image. If your initial image is scaled correctly, it should print a value within a few percent of 1.

  This may not be implemented; please use `0` and get the initial scaling correct!

- *slices*: Use, say, `7,12` to only reconstruct slices 7 through 12 (numbered from 0). The default hyphen is to do all slices.

A. The method string

As in 2D reconstruction [1], the generic syntax of the *method* argument looks like

$$@niter_1@algorithm_1@penalty_1@niter_2@algorithm_2@penalty_2...$$

This allows you to run $niter_1$ iterations using $algorithm_1$ for an objective with $penalty_1$, followed by $niter_2$ iterations using $algorithm_2$ for an objective with $penalty_2$, etc. Usually you will just have one algorithm. For example, setting *method* to

```
@9@osemc,fast,6,60,1.0@-
```

means 9 iterations of (unregularized) OSEM with 6 subsets out of 60 projection views, whereas

```
@9@ospsc,pc,6,60,0.99,0.1@3d,-6,-7,quad,5,-
```

means 9 iterations of relaxed OSPS [7] with 6 subsets out of 60 projection views, using the quadratic penalty function described in §I-B and the relaxation factor

$$\alpha_n = 0.99 \frac{1}{1 + 0.1 \cdot n}$$

which satisfies the global convergence conditions described in [8]. The general form is

$$\alpha_n = relax\_scale \frac{1}{1 + relax\_rate \cdot n} \tag{4}$$

where I recommend using *relax_scale*=1 and *relax_rate* between 0 and 0.5, and closer to 0, like 0.1 although good values are a subject of ongoing research. Using *relax_rate*=0 corresponds to "classic" OSPS which will not converge but usually works fine enough and saves you the trouble of picking yet another parameter. Using a small value like 0.01 for *relax_rate* will give you behavior nearly identical to classic OSPS for the first 50 iterations or so, yet let you sleep better at night knowing that if you actually ran hundreds of iterations the theory ensures that it will eventually converge.

Caution: OSPS may have problems when the r_i's are close to zero. (This never should happen in PET since random coincidences are ubiquitous and the "shifted Poisson" approach usually ensures that the r_i's are not too small. Scatter estimates should further increase the r_i's.) If your r_i's are close to zero or zero, then you probably need the OSDP algorithm which is on my "to do" list. Bug me!

IV. TRPL3: 3D TRANSMISSION RECONSTRUCTION

For transmission tomography, ASPIRE is based on the following measurement model:

$$Y_i \sim \mathsf{Poisson}\left\{ b_i \exp\left(-\sum_{j=1}^{n_\mathrm{p}} g_{ij} x_j \right) + r_i \right\},$$

where r_i is additive background that includes random coincidences and possibly scatter, and b_i is the blank scan (appropriately scaled for the relative durations of the blank and transmission scans). The arrays Y_i, b_i, and r_i must all have the same size, typically $n_u \times n_v \times n_\mathrm{view}$.

The corresponding log-likelihood is

$$L(\boldsymbol{x}) = \sum_{i=1}^{n_\mathrm{d}} h_i([\boldsymbol{G}\,\boldsymbol{x}]_i),$$

where

$$h_i(l) = Y_i \log\Big(b_i e^{-l} + r_i\Big) - (b_i e^{-l} + r_i),$$

and its negative is the "data fit" term in (1) for maximum likelihood (ML) and penalized-likelihood emission image reconstruction.

Typing `i trpl3` will print the syntax of how to perform 3D transmission image reconstruction under the above model. The output should include something like:

```
Usage: trpl3
        file_out init|-nx,ny,nz,value} file_yi
file_bi- bi_scale file_ri- ri_scale
sys_type file_mask- method
        file_ci- ci_scale file_ri- ri_scale
        sys_type file_mask- method
        [saver- flag_obj(0) flag_nonneg(1) pix_max scale_init(0) slices-]
        method: @niter@alg@penalty...
```

(The argument order is fixed.)

The arguments followed by a hyphen "–" are optional; using the hyphen will give sensible defaults.

Virtually all of these arguments have the identical meanings as in `i empl3`, except that of course *file_bi* and *bi_scale* correspond to the b_i's.

The principal differences are in terms of what *algorithms* are available to include in the *method* string.

Based on the work of Erdoğan [9], I recommend using the *ordered-subsets paraboloidal surrogates* (OSPS) algorithm for transmission reconstruction. (Unlike the emission case, there is no problem with small r_i's in the transmission case.)

Setting *method* to

```
@9@ospsc,pc,6,60,1.0,0.1@-3d,-6,-7,huber,6,-,0.002,ih,3
```

means 9 iterations of OSPS with 6 subsets out of 60 projection views, with the edge-preserving penalty function described in §I-B, and with the same relaxation parameters described in (4).

V. PWLS3: 3D PENALIZED WEIGHTED LEAST-SQUARES

The (quadratically) penalized weighted least-squares (PWLS) approach to image reconstruction computes an estimate of the image by minimizing the following cost function:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x}), \quad \Psi(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x})' \boldsymbol{W} (\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) - \boldsymbol{n}'(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) + \frac{1}{2}\boldsymbol{x}' \boldsymbol{R}\,\boldsymbol{x} \tag{5}$$

where

- $\boldsymbol{x}$ is the unknown image (volume),
- $\boldsymbol{y}$ is (typically a processed version of) the measured projection data,
- $\boldsymbol{G}$ is the system matrix,
- $\boldsymbol{W}$ is a diagonal weighting matrix with nonnegative entries,
- $\boldsymbol{R}$ is a nonnegative definite regularization matrix, and
- $\boldsymbol{n}$ is currently undocumented (assume it is $\boldsymbol{0}$).

By minimizing $\Psi(\boldsymbol{x})$, one finds an image that "fits the data" (where fit is measured by the first term) but is also not too noisy/rough (where roughness is measured by the penalty term).

As noted in §I-C, a quadratic penalty can be written as

$$R(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}' \boldsymbol{R}\,\boldsymbol{x}.$$

Currently only quadratic penalties are documented for PWLS in 3D.

One can verify that the (column) gradient of $\Psi(\boldsymbol{x})$ is given by

$$\begin{aligned} \nabla \Psi(\boldsymbol{x}) &= \boldsymbol{G}' \boldsymbol{W} (\boldsymbol{G}\,\boldsymbol{x} - \boldsymbol{y}) + \boldsymbol{G}'\,\boldsymbol{n} + \boldsymbol{R}\,\boldsymbol{x} \\ &= \boldsymbol{H}\,\boldsymbol{x} - \boldsymbol{b} \end{aligned}$$

where

$$\boldsymbol{H} = \nabla^2 \Psi(\boldsymbol{x}) = \boldsymbol{G}' \boldsymbol{W}\,\boldsymbol{G} + \boldsymbol{R}$$

is the Hessian of the objective function, and

$$\boldsymbol{b} = \boldsymbol{G}'(\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{n})$$

is a weighted backprojection of the data.

Thus, in the absence of nonnegativity constraints, the reconstruction algorithm must solve the linear system of equations

$$\boldsymbol{H}\,\boldsymbol{x} = \boldsymbol{b}.$$

The analytical solution is

$$\hat{\boldsymbol{x}} = \boldsymbol{H}^{-1}\,\boldsymbol{b} = [\boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{G} + \boldsymbol{R}]^{-1}\,\boldsymbol{G}'(\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{n}),$$

but this cannot be computed directly except for small problems, so iterative methods are required. The preconditioned conjugate gradient (PCG) algorithm [10] is well-suited to the above minimization problem (5).

Note that if $\boldsymbol{W} = \boldsymbol{I}$ and $\boldsymbol{R} = 0$, then

$$\hat{\boldsymbol{x}} = [\boldsymbol{G}'\,\boldsymbol{G}]^{-1}\,\boldsymbol{G}'\,\boldsymbol{y},$$

which is the ordinary least-squares solution to the linear model

$$\boldsymbol{y} = \boldsymbol{G}\,\boldsymbol{x} + \text{noise}.$$

Ordinary least-squares is suboptimal for tomography.

A. Execution of PWLS

Typing `i pwls3` will print the syntax of how to perform PWLS-based reconstruction. The output should include something like:

```
Usage: pwls3 out {init|-nx,ny,nz,value} data
sys_type wi- mask- method
[saver- flag_obj(0) flag_nonneg(1) pix_max scale_init(0) slices-]
```

Most of the arguments are identical to those for `i empl3`.
The arguments that differ are as follows.

- *init* is the name of the initial image file, dimensions $n_x \times n_y \times n_z$, or a string $-n_x, n_y, n_z,$ *value* that gives the size and value of a uniform initial image. For PCG I am less sure about the importance of initializing with FBP.
- *data* is either the $n_x \times n_y \times n_z$ file containing $\boldsymbol{b}$, or the projection data $\boldsymbol{y}$. The latter is recommended now.
- *sys_type* is as described in §VI. If $\boldsymbol{b}$ is used for *data*, then make sure that *exactly* the same *sys_type* argument is used here and when $\boldsymbol{b}$ is computed using `i back3`, or the reconstruction results will be garbage!
- *wi* is the (usually) $n_u \times n_v \times n_{\text{view}}$ file containing the w_i's (diagonal of $\boldsymbol{W}$).
- *flag_nonneg* This is not implemented for PCG.
- *pix_max*: maximum allowable pixel value. This is not implemented for PCG.

For the method string, consider

```
@9@cg,none@3d,-6,-7,quad,5,-
```

which means 9 iterations of conjugate gradient with no preconditioner for the quadratic penalty described in §I-B. (Other preconditioners may be documented later.)

B. Applying PWLS to Poisson Emission Data

Poisson emission data has a non-quadratic log-likelihood, but we can make a quadratic approximation. At one time I thought this would be handy for speeding things up. But ordered subsets algorithms are so fast now even for the Poisson likelihoods that I doubt that these quadratic approximations will be important. Here they are for historical interest.

Some of this may be implemented in `op lin,em`.

If the measurements have independent Poisson distributions:

$$Y_i \sim \text{Poisson}\left\{ c_i \sum_{j=1}^{n_\text{p}} g_{ij} x_j + r_i \right\},$$

then the log-likelihood is

$$L(\boldsymbol{x}) = \sum_{i=1}^{n_\text{d}} h_i([\boldsymbol{G}\,\boldsymbol{x}]_i),$$

where

$$\begin{aligned}
h_i(l) &= Y_i \log(c_i l + r_i) - (c_i l + r_i) \\
\dot{h}_i(l) &= c_i \left[\frac{Y_i}{c_i l + r_i} - 1 \right] \\
\ddot{h}_i(l) &= -c_i^2 \frac{Y_i}{(c_i l + r_i)^2}.
\end{aligned}$$

Let $\hat{l}_i$ be an estimate of $[\boldsymbol{G}\,\boldsymbol{x}]_i$, obtained somehow (and called `pivot`). One natural choice for $\hat{l}_i$ is

$$\hat{l}_i = \frac{Y_i - r_i}{c_i}, \tag{6}$$

which is a fairly standard choice ("precorrect the data"), but this is not the only logical choice. In particular, since

$$\boldsymbol{l}^{\text{true}} = \boldsymbol{G}\,\boldsymbol{x}^{\text{true}} \geq \boldsymbol{0},$$

another logical choice is

$$\hat{l}_i = \left[\frac{Y_i - r_i}{c_i}\right]_+.$$

It may also be sensible to smooth the $\hat{l}_i$'s [6].

Regardless of which choice one makes for $\hat{l}_i$, for $l \approx \hat{l}_i$,

$$\begin{aligned}
h_i(l) &\approx h_i(\hat{l}_i) + \dot{h}_i(\hat{l}_i)(l - \hat{l}_i) + \frac{1}{2}\ddot{h}_i(\hat{l}_i)(l - \hat{l}_i)^2 \\
&= h_i(\hat{l}_i) - n_i(l - \hat{l}_i) - \frac{1}{2}d_i(l - \hat{l}_i)^2 \\
&= h_i(\hat{l}_i) + n_i(\hat{l}_i - l) - \frac{1}{2}d_i(\hat{l}_i - l)^2 \\
&= -\left[\frac{1}{2}d_i(\hat{l}_i - l)^2 - n_i(\hat{l}_i - l) - h_i(\hat{l}_i)\right]
\end{aligned}$$

where

$$d_i \stackrel{\triangle}{=} -\ddot{h}_i(\hat{l}_i) = c_i^2 \frac{Y_i}{(c_i\hat{l}_i + r_i)^2}$$

is called `nder2` (since it is the negative of the second derivative) and

$$n_i \stackrel{\triangle}{=} -\dot{h}_i(\hat{l}_i) = c_i\left[1 - \frac{Y_i}{c_i\hat{l}_i + r_i}\right]$$

is called `nder1` (since it is the negative of the first derivative). Note that if we use (6), then $n_i = 0$, so the linear term disappears and we are left with the quadratic term discussed in [6].

Renaming $\hat{l}_i$ just y_i, we can maximize the quadratic objective function

$$\begin{aligned}
L_q(\boldsymbol{x}) &= \sum_{i=1}^{n_{\text{d}}} n_i(y_i - [\boldsymbol{G}\,\boldsymbol{x}]_i) - \frac{1}{2}d_i(y_i - [\boldsymbol{G}\,\boldsymbol{x}]_i)^2 \\
&= \boldsymbol{n}'(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) - \frac{1}{2}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x})'\,\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x})
\end{aligned}$$

or equivalently minimize its negative

$$\Psi(\boldsymbol{x}) = -L_q(\boldsymbol{x}) + R(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x})'\,\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) - \boldsymbol{n}'(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) + R(\boldsymbol{x}).$$

VI. System Models (Projector/backprojectors)

ASPIRE includes implementations of several different tomographic system models. Users specify the choice and parameters of these models with the *sys_type* argument which, not surprisingly, describes the system type[2]. This string tells the software *what type* of system model (*i.e.*, projector/backprojector) to use.

Not all of the system models are documented here. To see what choices are implemented, type `i proj3` and the output will include something like the following.

```
sys_type (imaging system model) choices:
        2dsc@...  separable block 2d system matrix on the fly
        2z@...    separable block 2d system matrix precomputed
        3a@...    forward/back projector
        3b@...    forward/back projector (improved)
        3c@...    forward/back projector for cylindrical geom
        3l@...    cone-beam line integrals
        3s@...    SPECT with depth-dependent response
        3u@...    user-defined forward/back projector
...
```

Additional usage information is printed for each system model choice, and by comparing the examples detailed here to the built-in documentation, you should be able to figure out how to use some of the undocumented ones. Ask me if interested!

A. User-defined projector / backprojector

If you need a different projector / backprojector than those described below, you can write your own projector / backprojector and compile it as a dynamic library, then call it using the `3u` option for *sys_type*. The details are explained in a separate document that can be downloaded from the web page where the ASPIRE binaries are located, including a working example. If you want to attempt this, I recommend that you discuss it with me first.

B. Separable 2D projector

The simplest "3D" projector is the separable block-2D projector having the form:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_{\mathrm{2D}} & & 0 \\ & \ddots & \\ 0 & & \boldsymbol{G}_{\mathrm{2D}} \end{bmatrix} = \boldsymbol{I}_{n_z} \otimes \boldsymbol{G}_{\mathrm{2D}},$$

where $\boldsymbol{I}$ is the $n_z \times n_z$ identity matrix, $\otimes$ denotes Kronecker matrix product, and $\boldsymbol{G}_{\mathrm{2D}}$ is any of the 2D projectors that one can generate using ASPIRE. In this case $\boldsymbol{G}_{\mathrm{2D}}$ has dimensions $(n_a n_b) \times (n_x n_y)$ and thus $\boldsymbol{G}$ has dimensions $(n_a n_b n_z) \times (n_x n_y n_z)$.

To use this system model, the *sys_type* argument is

`2z@`*file.wtf*`@-`

where *file.wtf* is the name of the `.wtf` generated using ASPIRE corresponding to $\boldsymbol{G}_{\mathrm{2D}}$.

- Since this model uses the same $\boldsymbol{G}_{\mathrm{2D}}$ for each slice, it is inappropriate for SPECT problems that require different attenuation maps for each slice. Use the `3s` model for such problems.
- This model requires that the data be organized as a set of n_z *sinograms* each of size $n_b \times n_a$, rather than as projection views. If necessary, apply `op transpose` to your data.

[2]Prior to Sep. 2001, I used the string `fi_type` instead of *sys_type*, as an abbreviation for "Fisher Information Type." I called it the Fisher Information because $\boldsymbol{G}' \boldsymbol{W} \boldsymbol{G}$ is the Fisher information for estimating $\boldsymbol{x}$ from $\boldsymbol{y}$ under the Gaussian model

$$\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{G}\,\boldsymbol{x}, \boldsymbol{W}^{-1}\right).$$

This made some sense for PWLS, but less sense for Poisson models. If you have code that says "`fi_type`" in the built-in documentation, then you have an old version.

- The third argument (hyphen) is work in progress that may lead to somewhat reduced compute time in the future.
- The only reason to do "3D" reconstruction with a separable system model is to obtain the benefit of 3D *regularization*. Conventional PET and SPECT reconstruction using FBP usually involves 3D post-filtering. So for fair comparisons between regularized iterative reconstruction and FBP, one should also use 3D regularization even if a slice-by-slice system model is used (which is reasonably appropriate for "2D" multi-slice PET scans).
- In large problems (such as X-ray CT), it can be impractical to precompuate and store a *file.wtf*. In such cases, one can instead use the following *sys_type*

  `2dsc@`*file.dsc*`@-`

  where *file.dsc* is the ASCII system geometry description file described in [1]. With the system model, the nonzero elements of $\boldsymbol{G}_{\mathrm{2D}}$ are computed *on the fly* during each forward and backprojection. This requires much more compute time, but much less memory than using a *file.wtf*. The `2dsc` projector/backprojector is threaded so it can exploit dual-processor machines [1].

C. Point-based 3D projector/backprojector

A simple 3D forward projection method is to treat each pixel as a point in 3D space, and project its value by bilinear interpolation onto a 2D projection view at some angle θ, ϕ, where θ is the axial "tilt" and ϕ is the transaxial rotation angle. A 3D data set consists of n_{view} such projection views, where each view corresponds to a unique (θ, ϕ) pair. Each view has $n_u \times n_v$ samples. I assume the image volume has unit sampling in x and y, and let s_z denote the (relative) slice spacing (in the z direction), and s_u and s_v the sample spacing in the projection views. The size of $\boldsymbol{G}$ in this case is essentially $(n_u n_v n_{\mathrm{view}}) \times (n_x n_y n_z)$. The *sys_type* for this projector is

`3a@`n_u`,`n_v`,`s_u`,`s_v`,`s_z`@`*iv.file*`@`*angles.file*

The output projection will have dimension $(n_u \times n_v \times n_{\mathrm{view}})$.

The file *iv.file* must have dimension $2 \times n_{\mathrm{view}}$ and should consist of the pairs

$$(i_1^{\min}, i_1^{\max}),\ (i_2^{\min}, i_2^{\max}),\ \ldots,\ (i_{n_{\mathrm{view}}}^{\min}, i_{n_{\mathrm{view}}}^{\max}),$$

where i is an integer index in the v-dimension of projection views. When processing projection view k, the program will only project to and backproject from rows `iv` for which $i_k^{\min} \le \mathtt{iv} < i_k^{\max}$. If the default hyphen `-` is used, then all rows of each view are used. In most geometries, not all rows are valid data, so the ranges almost surely should be provided by the user.

The file *angles.file* must have dimension $2 \times n_{\mathrm{view}}$ and should consist of the pairs

$$(\theta_1, \phi_1), (\theta_2, \phi_2), \ldots, (\theta_{n_{\mathrm{view}}}, \phi_{n_{\mathrm{view}}}),$$

where the angles are in radians. The projection data will be arranged in the same order. Typically $n_{\mathrm{view}} = n_\theta n_\phi$, in which case I recommend making ϕ vary fastest. The reason for this recommendation is the following. After creating the projection data using `i pwls3`, it may useful to display the data in sinogram format rather than projection format. The command

`op transpose` *sino.out proj.in* 1,2

will switch the 2nd and 3rd dimensions of *proj.in*, so the output size will be $(n_u \times n_{\mathrm{view}} \times n_v) = (n_u \times n_\phi \cdot n_\theta \times n_v)$, which may be more intuitive to view, at least for small n_θ.

The `3a` projector is now obsolete due to improved `3b` projector. Ask me if you want documentation for it.

A concrete simple example for *sys_type* is

```
3a@128,64,1,1,1@-@angles.fld
```

D. GE 3D projector

If (and only if) you are associated with GE (*e.g.*, UW), then you should also have access to the 3D projector written by Chuck Stearns, which has the same setup as the `3a` projector above, except you replace `3a` with `3ge`. However, the GE projector and backprojector are not transposes of each other, which may be fine for FBP reconstruction, but causes

problems with iterative reconstruction since they are used repeatedly and therefore any discrepancies will accumulate. I recommend the `3a` projector instead, since the corresponding backprojector is the transpose of the forward projector.

The GE projector also does not have the `iv.file` argument or capability, which may cause problems.

E. Cylindrical PET systems

The `3c` system model is designed for cylindrical PET systems with septa removed. It is still somewhat under development. Bug me if interested.

Type `i proj3` to see brief documentation.

F. SPECT system model

The `3s` system model includes both nonuniform attenuation and user-specified depth-dependent detector response. The implementation is based on a rotate, attenuate, depth-dependent filter, then sum approach. The projector and backprojector are adjoint operators.

The *sys_type* string for SPECT is

`3s@`s_x`,`s_y`,`s_z`,orbit,orbit_start,spline_filter@mumap.file@filter.file@-`n_u`,`n_v`,`n_{view}

- (s_x,s_y,s_z) are the voxel sizes (typically in mm or cm)
- Currently must have $s_x = s_y$, $n_x = n_y$, $n_u = n_x$, $n_v = n_z$
- *orbit* and *orbit_start* are in degrees
- *spline_filter* must be `0` (nearest neighbor), or `1` (for B1, linear interpolation), or `3` (for B3, cubic B-spline interpolation). I recommend using 1 or possibly 3.
- The *filter.file* must be [n1_psf,n2_psf,ny], with n1_psf and n2_psf odd. This file must contain the user-specified depth-dependent PSF for each of the n_y planes parallel to the detector.
- Optionally, *filter.file* can be [n1_psf,n2_psf,ny,nview] for angle-dependent blurs, such as in noncircular orbits.
- The final argument *−nu,nv,nview* is required.

If you plan on using this system model, you probably want to ask me for an example script including a Matlab m-file for generating the PSFs.

REFERENCES

[1] J. A. Fessler. ASPIRE 3.0 user's guide: A sparse iterative reconstruction library. Technical Report 293, Comm. and Sign. Proc. Lab., Dept. of EECS, Univ. of Michigan, Ann Arbor, MI, 48109-2122, July 1995. Available from `http://www.eecs.umich.edu/~fessler`.

[2] J. A. Fessler and W. L. Rogers. Spatial resolution properties of penalized-likelihood image reconstruction methods: Space-invariant tomographs. *IEEE Tr. Im. Proc.*, 5(9):1346–58, September 1996.

[3] L. T. Chang. A method for attenuation correction in radionuclide computed tomography. *IEEE Tr. Nuc. Sci.*, 25(1):638–643, February 1978.

[4] L. T. Chang. Attenuation correction and incomplete projection in single photon emission computed tomography. *IEEE Tr. Nuc. Sci.*, 26(2):2780–2785, April 1979.

[5] K. Sauer and C. Bouman. A local update strategy for iterative reconstruction from projections. *IEEE Tr. Sig. Proc.*, 41(2):534–48, February 1993.

[6] J. A. Fessler. Penalized weighted least-squares image reconstruction for positron emission tomography. *IEEE Tr. Med. Im.*, 13(2):290–300, June 1994.

[7] J. A. Fessler and H Erdoğan. A paraboloidal surrogates algorithm for convergent penalized-likelihood emission image reconstruction. In *Proc. IEEE Nuc. Sci. Symp. Med. Im. Conf.*, volume 2, pages 1132–5, 1998.

[8] S. Ahn and J. A. Fessler. Globally convergent ordered subsets algorithms: Application to tomography. In *Proc. IEEE Nuc. Sci. Symp. Med. Im. Conf.*, volume 2, pages 1064–8, 2001.

[9] H Erdoğan and J. A. Fessler. Ordered subsets algorithms for transmission tomography. *Phys. Med. Biol.*, 44(11):2835–51, November 1999.

[10] J. A. Fessler and S. D. Booth. Conjugate-gradient preconditioning methods for shift-variant PET image reconstruction. *IEEE Tr. Im. Proc.*, 8(5):688–99, May 1999.

Most of my papers are available on WWW: `http://www.eecs.umich.edu/~fessler`.

The remainder of this report is notes that are mostly to myself about various aspects of the implementation.

VII. 3D PROJECTION COORDINATE SYSTEM - 3A

This section describes the coordinate system for the `3a` projector.

Image-volume physical coordinates are x, y, z, with

$$\begin{aligned} x &= c_x + (i_x - (n_x-1)/2)s_x = (i_x - w_x)s_x \text{ where } w_x = (n_x-1)/2 - c_x/s_x \\ y &= c_y + (i_y - (n_y-1)/2)s_y = (i_y - w_y)s_y \text{ where } w_y = (n_y-1)/2 - c_y/s_y \\ z &= c_z + (i_z - (n_z-1)/2)s_z = (i_z - w_z)s_z \text{ where } w_z = (n_z-1)/2 - c_z/s_z. \end{aligned}$$

Pixel coordinate i_x goes from 0 to n_x etc.

Projection view coordinates are u, v, with

$$\begin{aligned} u &= c_u + (i_u - (n_u-1)/2)s_u = (i_u - w_u)s_u \text{ where } w_u = (n_u-1)/2 - c_u/s_u \\ v &= c_v + (i_v - (n_v-1)/2)s_v = (i_v - w_v)s_v \text{ where } w_v = (n_v-1)/2 - c_v/s_v. \end{aligned}$$

For a parallel projection at polar angle θ and azimuthal angle ϕ:

$$\begin{aligned} u &= x\cos\phi + y\sin\phi \\ v &= (-x\sin\phi + y\cos\phi)\sin\theta + z\cos\theta. \end{aligned}$$

Useful relationship:

$$\begin{aligned} i_u &= \frac{u}{s_u} + w_u \\ &= \frac{x\cos\phi + y\sin\phi}{s_u} + w_u \\ &= \frac{(i_x - w_x)s_x\cos\phi + (i_y - w_y)\sin\phi}{s_u} + w_u \\ &= i_x\frac{s_x}{s_u}\cos\phi + i_y\frac{s_y}{s_u}\sin\phi + \left(w_u - \frac{w_x s_x\cos\phi + w_y s_y\sin\phi}{s_u}\right). \end{aligned}$$

Similarly

$$\begin{aligned} i_v &= \frac{v}{s_v} + w_v \\ &= \frac{(-x\sin\phi + y\cos\phi)\sin\theta + z\cos\theta}{s_v} + w_v \\ &= \frac{[-(i_x - w_x)s_x\sin\phi + (i_y - w_y)s_y\cos\phi]\sin\theta + (i_z - w_z)s_z\cos\theta}{s_v} + w_v \\ &= -i_x\frac{s_x}{s_v}\sin\phi\sin\theta + i_y\frac{s_y}{s_v}\cos\phi\sin\theta + i_z\frac{s_z}{s_v}\cos\theta + \left(\frac{[w_x s_x\sin\phi - w_y s_y\cos\phi]\sin\theta - w_z s_z\cos\theta}{s_v} + w_v\right). \end{aligned}$$

VIII. 3D PROJECTION COORDINATE SYSTEM - 3B

This section describes the coordinate system for the `3b` projector.

Image-volume physical coordinates are (x, y, z), with

$$\begin{aligned} x &= [i_x - (n_x - 1)/2 - c_x]s_x = (i_x - w_x)s_x \text{ where } w_x \triangleq (n_x - 1)/2 + c_x \\ y &= [i_y - (n_y - 1)/2 - c_y]s_y = (i_y - w_y)s_y \text{ where } w_y \triangleq (n_y - 1)/2 + c_y \\ z &= [i_z - (n_z - 1)/2 - c_z]s_z = (i_z - w_z)s_z \text{ where } w_z \triangleq (n_z - 1)/2 + c_z. \end{aligned}$$

Pixel coordinate i_x goes from 0 to $n_x - 1$ etc., and the voxel "centers" (c_x,c_y,c_z) are in units of voxels (not mm) since the most likely shifts will be 0 or 0.5 voxels.

Projection view coordinates are (u, v), with

$$\begin{aligned} u &= [i_u - (n_u - 1)/2 - c_u]s_u = (i_u - w_u)s_u \text{ where } w_u \triangleq (n_u - 1)/2 + c_u \\ v &= [i_v - (n_v - 1)/2 - c_v]s_v = (i_v - w_v)s_v \text{ where } w_v \triangleq (n_v - 1)/2 + c_v. \end{aligned}$$

We assume projections are "sampled" at the *center* of each projection view bin. If $c_u = 0$ and $s_u = 1$, then as i_u ranges from 0 to n_u-1, u ranges from $-\frac{n_u-1}{2}$ to $\frac{n_u-1}{2}$.

For a parallel projection at polar angle θ and azimuthal angle ϕ:

$$\begin{aligned} u &= x \cos\phi + y \sin\phi \\ v &= (-x \sin\phi + y \cos\phi) \sin\theta + z \cos\theta. \end{aligned}$$

Useful relationship:

$$\begin{aligned} i_u &= \frac{u}{s_u} + w_u \\ &= \frac{x \cos\phi + y \sin\phi}{s_u} + w_u \\ &= \frac{(i_x - w_x)s_x \cos\phi + y \sin\phi}{s_u} + w_u \\ &= i_x \left(\frac{s_x}{s_u} \cos\phi \right) + \left(\frac{-w_x s_x \cos\phi + y \sin\phi}{s_u} + w_u \right). \end{aligned}$$

The second term is `uu_factor`.

Similarly

$$\begin{aligned} i_v &= \frac{v}{s_v} + w_v \\ &= \frac{(-x \sin\phi + y \cos\phi) \sin\theta + z \cos\theta}{s_v} + w_v \\ &= \frac{[-(i_x - w_x)s_x \sin\phi + y \cos\phi] \sin\theta + (i_z - w_z)s_z \cos\theta}{s_v} + w_v \\ &= -i_x \left(\frac{s_x}{s_v} \sin\phi \sin\theta \right) + i_z \left(\frac{s_z}{s_v} \cos\theta \right) + \left(\frac{[w_x s_x \sin\phi + y \cos\phi] \sin\theta - w_z s_z \cos\theta}{s_v} + w_v \right). \end{aligned}$$

The last term is called `vv_factor`. The middle term is called `iv_inc`, and is essentially di_v/di_z: the change in i_v per slice. The sum of the first and last term is `ivd`. We write:

$$i_v = v_0 + i_z \, \delta \, .$$

If $\theta = 0$ and $s_x = s_y = s_z = s_u = s_v$ and $w_z = w_v$, then $v_0 = 0$ and $\delta = 1$, as expected.

Useful facts about $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$:

$$x \le \lceil x \rceil < 1 + x, \qquad\qquad x - 1 < \lfloor x \rfloor \le x.$$

For bilinear interpolation in the u direction, we must have

$$0 \le \lfloor i_u \rfloor \le \lceil i_u \rceil \le n_u - 1.$$

So there is a problem if $\lfloor i_u \rfloor < 0$ or if $\lceil i_u \rceil \ge n_u$. This can be verified by the `3b_check` routine.

For bilinear interpolation in the v direction, to have *both* points inside the valid range we must have

$$i^{\min} \le \lfloor i_v \rfloor \le \lceil i_v \rceil \le i^{\max} - 1,$$

or equivalently $\lceil i_v \rceil < i^{\max}$ for the last term. So we require

$$i^{\min} \le \lfloor v_0 + i_z \, \delta \rfloor .$$

Suppose we choose as the starting slice:

$$\boxed{k_0 \triangleq \left\lceil \frac{i^{\min} - v_0}{\delta} \right\rceil},$$

then since $x \le \lceil x \rceil$ we have

$$\frac{i^{\min} - v_0}{\delta} \le k_0$$

so (provided of course that $\delta > 0$)

$$i^{\min} \le v_0 + k_0 \, \delta .$$

If $m \le x$ for integer m, then $m \le \lfloor x \rfloor$, so as required

$$i^{\min} \le \lfloor v_0 + k_0 \, \delta \rfloor .$$

We also require that

$$\lceil v_0 + i_z \, \delta \rceil \le i^{\max} - 1.$$

Suppose we choose as the ending slice:

$$\boxed{k_1 \triangleq \left\lfloor \frac{i^{\max} - 1 - v_0}{\delta} \right\rfloor}.$$

Since $\lfloor x \rfloor \le x$,

$$k_1 \le \frac{i^{\max} - 1 - v_0}{\delta}$$

so (for $\delta > 0$ again)

$$v_0 + k_1 \, \delta \le i^{\max} - 1.$$

If $x \le n$ for integer n, then $\lceil x \rceil \le n$. So as required

$$\lceil v_0 + k_1 \, \delta \rceil \le i^{\max} - 1.$$

IX. 3D PROJECTION COORDINATE SYSTEM - 3C (CYLINDER PET)

This section describes the coordinate system for the `3c` projector.

Image-volume physical coordinates are (x, y, z), with

$$\begin{aligned} x &= [i_x - (n_x - 1)/2 - c_x]s_x = (i_x - w_x)s_x \text{ where } w_x \triangleq (n_x - 1)/2 + c_x \\ y &= [i_y - (n_y - 1)/2 - c_y]s_y = (i_y - w_y)s_y \text{ where } w_y \triangleq (n_y - 1)/2 + c_y \\ z &= [i_z - (n_z - 1)/2 - c_z]s_z = (i_z - w_z)s_z \text{ where } w_z \triangleq (n_z - 1)/2 + c_z. \end{aligned}$$

Pixel coordinate i_x goes from 0 to $n_x - 1$ etc., and the voxel "centers" (c_x,c_y,c_z) are in units of voxels (not mm) since the most likely shifts will be 0 or 0.5 voxels.

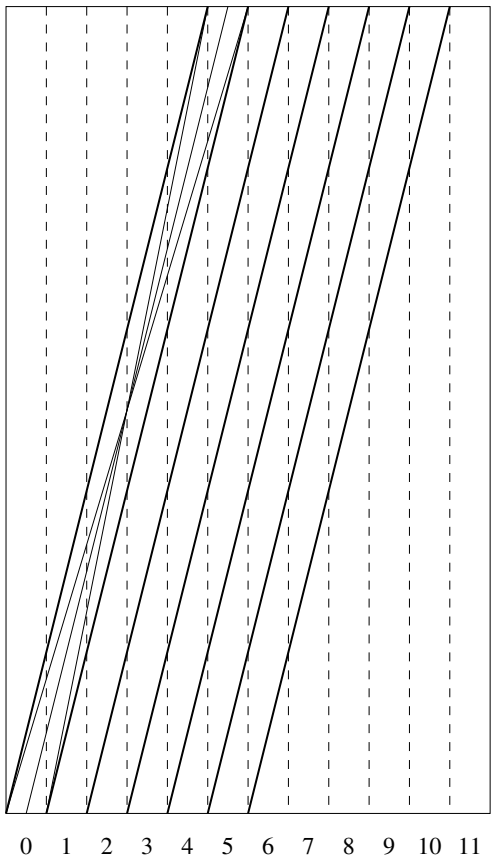
Projection view coordinates are (u, v), with

$$\begin{aligned} u &= [i_u - (n_u - 1)/2 - c_u]s_u = (i_u - w_u)s_u \text{ where } w_u \triangleq (n_u - 1)/2 + c_u \\ v &= [i_v - (n_v - 1)/2 - c_v]s_v = (i_v - w_v)s_v \text{ where } w_v \triangleq (n_v - 1)/2 + c_v. \end{aligned}$$

We assume projections are "sampled" at the *center* of each projection view bin. If $c_u = 0$ and $s_u = 1$, then as i_u ranges from 0 to n_u-1, u ranges from $-\frac{n_u-1}{2}$ to $\frac{n_u-1}{2}$.

For a parallel projection at polar angle θ and azimuthal angle ϕ:

$$\begin{aligned} u &= x\cos\phi + y\sin\phi \\ v &= (-x\sin\phi + y\cos\phi)\sin\theta + z\cos\theta. \end{aligned}$$



$s_v = s_z \cos\theta$, $\tan\theta = \frac{n_s d_c}{2r_c}$ where n_s is the span, and d_c is the crystal width, which for an Exact is 6.75mm, and r_c is the crystal radius, which for an Exact is 412.5mm.

Useful relationship:

$$\begin{aligned} i_u &= \frac{u}{s_u} + w_u \\ &= \frac{x\cos\phi + y\sin\phi}{s_u} + w_u \\ &= \frac{(i_x - w_x)s_x\cos\phi + y\sin\phi}{s_u} + w_u \\ &= i_x\left(\frac{s_x}{s_u}\cos\phi\right) + \left(\frac{-w_x s_x\cos\phi + y\sin\phi}{s_u} + w_u\right). \end{aligned}$$

The second term is `uu_factor`.

Similarly

$$\begin{aligned} i_v &= \frac{v}{s_v} + w_v \\ &= \frac{(-x\sin\phi + y\cos\phi)\sin\theta + z\cos\theta}{s_v} + w_v \\ &= \frac{[-(i_x - w_x)s_x\sin\phi + y\cos\phi]\sin\theta + (i_z - w_z)s_z\cos\theta}{s_v} + w_v \\ &= -i_x\left(\frac{s_x}{s_z}\sin\phi\tan\theta\right) + i_z + \left(\frac{[w_x s_x\sin\phi + y\cos\phi]\tan\theta}{s_z} + (w_v - w_z)\right). \end{aligned}$$

The last term is called `vv_factor`. The middle term is called `iv_inc`, and is essentially di_v/di_z: the change in i_v per slice. The sum of the first and last term is `ivd` v_0. We write:

$$i_v = i_z + v_0.$$

If $\theta = 0$ and $w_z = w_v$, then $v_0 = 0$ as expected.

Useful facts about $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$:

$$x \le \lceil x\rceil < 1 + x, \qquad\qquad x - 1 < \lfloor x\rfloor \le x.$$

For bilinear interpolation in the u direction *without cumbersome end condition checking*, we must have

$$0 \le \lfloor i_u\rfloor \le \lceil i_u\rceil \le n_u - 1.$$

So there is a problem if $\lfloor i_u\rfloor < 0$ or if $\lceil i_u\rceil \ge n_u$. This can be verified by the `3c_check` routine.

For bilinear interpolation in the v direction it is reasonable (and necessary) to check end conditions. For a given slice i_z to contribute something to the valid v range, we must have:

$$i^{\min} - 1 < i_v < i^{\max}.$$

For the left condition we require

$$i^{\min} - 1 < i_z + v_0,$$

so we choose as the starting slice:

$$\boxed{k_0 \triangleq \left\lfloor i^{\min} - v_0\right\rfloor.}$$

Since $x - 1 < \lfloor x\rfloor$, we have $i^{\min} - 1 = (i^{\min} - v_0 - 1) + v_0 < \lfloor i^{\min} - v_0\rfloor + v_0 = k_0 + v_0$ as required.

For the right condition, we require that

$$i_z + v_0 < i^{\max},$$

so we choose as the ending slice:

$$\boxed{k_1 \triangleq \lceil i^{\max} - 1 - v_0\rceil.}$$

Since $\lceil x\rceil < 1 + x$, we have $k_1 + v_0 = \lceil i^{\max} - 1 - v_0\rceil + v_0 < i^{\max}$ as required.

X. Computing the Objective Function

Computing the objective function in the code is somewhat different since only $\boldsymbol{b}$ is available and not $\boldsymbol{y}$ etc. From (5):

$$\begin{aligned}
\Psi(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x})'\,\boldsymbol{W}(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) - \boldsymbol{n}'(\boldsymbol{y} - \boldsymbol{G}\,\boldsymbol{x}) + R(\boldsymbol{x}) \\
&= \frac{1}{2}\boldsymbol{y}'\,\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{x}'\,\boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{y} + \frac{1}{2}\boldsymbol{x}'\,\boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{G}\,\boldsymbol{x} - \boldsymbol{n}'\boldsymbol{y} + \boldsymbol{x}'\,\boldsymbol{G}'\,\boldsymbol{n} + R(\boldsymbol{x}) \\
&= (\frac{1}{2}\boldsymbol{y}'\,\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{n}'\boldsymbol{y}) - \boldsymbol{x}'\,\boldsymbol{G}'(\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{n}) + \frac{1}{2}\boldsymbol{x}'\,\boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{G}\,\boldsymbol{x} + R(\boldsymbol{x}) \\
&= (\frac{1}{2}\boldsymbol{y}'\,\boldsymbol{W}\,\boldsymbol{y} - \boldsymbol{n}'\boldsymbol{y}) - \boldsymbol{b}'\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}'\,\boldsymbol{F}\,\boldsymbol{x} + R(\boldsymbol{x}),
\end{aligned}$$

where $\boldsymbol{F} = \boldsymbol{G}'\,\boldsymbol{W}\,\boldsymbol{G}$. The first term is a constant independent of $\boldsymbol{x}$, so is ignored. The second term is trivial to compute. The third term is computed using `fi_inc`. The final term is computed using `r3_penal`.

A. Line Search

Ignoring the penalty term:

$$\begin{aligned}
f(\alpha) = \Psi(\boldsymbol{x} + \alpha\boldsymbol{d}) - \Psi(\boldsymbol{x}) &= \frac{1}{2}(\boldsymbol{x} + \alpha\boldsymbol{d})'\,\boldsymbol{F}(\boldsymbol{x} + \alpha\boldsymbol{d}) - \frac{1}{2}\boldsymbol{x}'\,\boldsymbol{F}\,\boldsymbol{x} - \boldsymbol{b}'(\boldsymbol{x} + \alpha\boldsymbol{d}) + \boldsymbol{b}'\boldsymbol{x} \\
&= \alpha\boldsymbol{d}'(\boldsymbol{F}\,\boldsymbol{x} - \boldsymbol{b}) + \frac{1}{2}\alpha^2\boldsymbol{d}'\,\boldsymbol{F}\,\boldsymbol{d},
\end{aligned}$$

so

$$\frac{d}{d\alpha}f(\alpha) = \boldsymbol{d}'(\boldsymbol{F}\,\boldsymbol{x} - \boldsymbol{b}) + \alpha\boldsymbol{d}'\,\boldsymbol{F}\,\boldsymbol{d}.$$

XI. Initial Scale

Suppose we have an initial guess $\boldsymbol{x}$ of the image, but that initial guess may be "improperly" scaled. Then we would like to find

$$\arg\min_{\alpha} \Psi(\alpha\boldsymbol{x})\,.$$

Actually, we can just consider the likelihood term and ignore the penalty term to find α, since this one-parameter estimation problem is (very) well-conditioned.

$$\Psi(\alpha\boldsymbol{x}) \equiv -\boldsymbol{b}'(\alpha\boldsymbol{x}) + \frac{1}{2}\alpha^2\boldsymbol{x}'\,\boldsymbol{F}\,\boldsymbol{x}$$

so

$$\frac{\partial}{\partial\alpha}\,\Psi(\alpha\boldsymbol{x}) = -\boldsymbol{b}'\boldsymbol{x} + \alpha\boldsymbol{x}'\,\boldsymbol{F}\,\boldsymbol{x}$$

so

$$\alpha = \frac{\boldsymbol{b}'\boldsymbol{x}}{\boldsymbol{x}'\,\boldsymbol{F}\,\boldsymbol{x}}$$

is the WLS initial scale value.

XII. Penalty Gradient and Hessian

We need to consider the z component of the penalty function more carefully, since the "border of zeros" trick that is used for the in-plane penalty will not work for the z component. For $\boldsymbol{x} = [x_0, x_1, \ldots, x_{n_z-1}]$ (in C notation):

$$R(\boldsymbol{x}) = \sum_{j=1}^{n_z-1} \frac{1}{2}(x_j - x_{j-1})^2.$$

So the gradient is

$$\frac{\partial}{\partial x_j} = \begin{cases} x_0 - x_1, & j = 0 \\ (x_j - x_{j-1}) + (x_j - x_{j+1}), & 1 \le j \le n_z - 2 \\ x_{n_z-1} - x_{n_z-2}, & j = n_z - 1, \end{cases}$$

and the Hessian diagonal is

$$\frac{\partial^2}{\partial x_j^2} = \begin{cases} 1, & j = 0 \\ 2, & 1 \le j \le n_z - 2 \\ 1, & j = n_z - 1. \end{cases}$$