GUIDE TO USING TIME-DISTANCE INVERSION SOFTWARE: RLS

JAVORNIK

The Time-Distance Inversion Software calculates inversions for local helioseismology using multi-channel deconvolution (MCD) Regularized Least Squares (RLS). This document contains example parameter files along with an explanation of the input and output data files. The document RLSNotes.pdf contains the equations and additional details about the calculations.

1. General Directions

These general directions apply to running all inversion problems. The inversion code consists of a Python script, called *invert*, and several shared libraries for parsing input files and calculating inversions. The *invert* script coordinates parsing of an input file, constructs directories for the results, and passes the information to an inversion function written in C. The shared libraries contain core operations such as matrix and linear equation operations, Fourier transforms, reading and writing binary data files, and parsing input parameters. The libraries provide wrappers for the external software packages: C FITS File Subroutine Library (CFITSIO), Linear Algebra PACKage (LAPACK), Fastest Fourier Transform in the West (FFTW), Sparse Least Squares (LSQR), Lex, and YACC. A software release bundles the libraries, Python scripts, documentation, and sample data into tar files. All installations require core.tar and then any combination of the specific tar files. Doc.tar contains code documentation and the test.tar files contain sample data files along with the expected results.

For RLS inversions download and expand, core.tar and rls.tar. To run the sample data, download the rls_test.tar file. After expanding the tar files, follow the instructions in the README file to build and run the inversion code.

Table 1 contains the possible problem dimensions along with a brief description of the inversion method. Table 2 contains the input parameters for each dimension.

The command to run an inversion is:

```
invert parameterFile -p -c -d
```

The python script, *invert*, verifies files and directories exist, copies input files, finds an empty version of the output directory, and calls the inversion code. Table 3 explains the command-line flags.

1.1. Input and Output Data Files.

Tables 4 and 5 contain a sample parameter file and the corresponding output files. Input and output data files follow the Flexible Image Transport System (FITS) format except the map file for 1-D RLS inversion, which can

Date: September 17, 2012.

TABLE 1. Possible Dimensions

Dimension	Description
RLS	Solve RLS inversion for a number of λ.
1d	One parameter, λ_1, contributes to λ^2 such that $\lambda^2 = \lambda_1^2$.
3d-scalar	Two parameters, λ_1 and λ_2, contribute to λ^2, such that $\lambda^2 = \lambda_1^2 + \lambda_2^2(k_x^2 + k_y^2)$, where $k_x = \frac{2\pi}{N_x}x$ and $k_y = \frac{2\pi}{N_y}y$
3d-vector	Generalize to the case where the model is a set of values at each location $(\boldsymbol{x}, z)$ Examples include inversions for a flow field, or for multiple structure variables (e.g. sound speed, density, pressure). Three kernel files correspond to a single map file.

TABLE 2. Parameter File Contents

Parameter	RLS
dimension	1D, 3D-scalar, 3D-vector
output base directory	yes
base dir for kernels	yes
base dir for maps	yes
kernel-map pairs	M and d
noise covariance	C
Regularization matrix	R
Averaging Kernel & noise	yes
regularization parameters	λ_1, λ_2

TABLE 3. Command-Line Flags

Flag	Action
$-p$	indicates parsing the input file only (no inversion)
$-c$	copies all input files to the output directory
$-d$	dumps inversion intermediate values for debugging
parameterFile	contains the name and location of all input data files and all inversion parameters.

be a text file. If an input FITS file ends with (.gz), it is read as a compressed file and is expanded in memory.

The parameter file contains directory paths, file names and parameter values. The output base directory contains all the output files generated by the inversion program

TABLE 4. Parameter File for 3d_scalar RLS Inversion

3d_scalar rls					
/data/cora/output					
/data/cora/Kernels					
/data/cora/Maps					
Kern1.fits Map1.fits					
Kern2.fits Map2.fits					
Kern3.fits Map3.fits					
/full/path/C.fits					
/full/path/R.fits					
Avg Kernel 10 20					
	λ_1			λ_2	
Min	Max	NSteps	Min	Max	NSteps
0.5	1.0	2	0.0	1.0	1

TABLE 5. Output Files for 3d_scalar RLS Inversion

File Names		Description
$\lambda_1 = 0.5$, $\lambda_2 = 0.0$	$\lambda_1 = 1.0$, $\lambda_2 = 0.0$	
1_1_b.fits	2_1_b.fits	solution
1_1_d∼1.fits.gz	2_1_d∼1.fits.gz	estimate of first travel-time map
1_1_d∼2.fits.gz	2_1_d∼2.fits.gz	estimate of second travel-time map
1_1_d∼3.fits.gz	2_1_d∼3.fits.gz	estimate of third travel-time map
1_1_AvgKern_10.fits.gz	2_1_AvgKern_10.fits.gz	averaging kernel for z=10
1_1_AvgKern_20.fits.gz	2_1_AvgKern_20.fits.gz	averaging kernel for z=20
1_1_Cov_10.fits.gz	2_1_Cov_10.fits.gz	error covariance for z=10
1_1_Cov_20.fits.gz	2_1_Cov_20.fits.gz	error covariance for z=20

along with a copy of the parameter file. The output base directory also contains copies of all the files specified in the parameter file, if the '-c' flag is used. If the output directory does not exist, the invert program creates it. If the output base directory exists and is empty, then the invert program dumps the output files there. If the output directory is not empty, then, the invert program adds a version (_v2) to the output directory name. The invert program continues to append version numbers to the output directory name until an empty directory is found or the directory does not exist. Averaging kernels and noise covariances are calculated for every height (z-value) listed. For RLS inversions, the code calculates averaging kernels and noise covariance at every height, selected heights, or none at all. The inversion code computes the error covariance matrix by the formal propagation of errors.

Output files include the solution, averaging kernels, noise/error covariance, and an estimate of the travel-time maps, (d∼), which is calculated using the model found during inversion. The header of each output FITS file contains the regularization parameters (

λ_1, λ_2) used to generate the data. The naming convention for the output files follows the format λ_1_λ_2_$matrix.fits$. Matrix names for RLS are b, d$\sim$, AvgKern, and Cov. Averaging kernel and covariance file names have the associated height appended to the matrix name for RLS inversions only. All averaging kernel, noise/error covariance and travel-time map estimates are compressed FITS files, with the extension (.fits.gz).

The Examples/testdata directory contains example parameter files for each inversion and dimension. The Examples/testdata/output directory contains sample output files for each parameter file.

2. Examples of RLS Inversions

2.1. **1-D Problem.** The 1-D problem has one kernel file and one map file and is an inversion of the z-dimension only. The map file contains the travel-time measurement and is a one-dimensional vector. All input files are in FITS (Flexible Image Transport System) format. Only the 1-D problem takes either an ASCII file or a FITS file for the travel-time map. A map file formatted as a comma or tab separated ASCII file must have a '.txt' file extension, otherwise, a FITS file is expected. The looping parameters specify a range of values for the lambda parameter in the RLS inversion. See the RLSNotes.pdf file for equations explaining these matrices.

2.1.1. *Contents of Input File.* The input-file contains the following information:

(1) which problem (1d) which inversion method (rls)
(2) output base directory
(3) input map data file (d.txt or d.fits): input map data file is a vector and can be either ASCII text file or .fits file.
(4) input noise covariance matrix (C.fits)
(5) input Kernel file (M.fits)
(6) input Regularization matrix file (R.fits)
(7) lambda range

The sample data for 1-D RLS inversion has $N_z = 30$, $M = 11$, where N_z is the size of the z-dimension, and M is the number of travel-time measurements or maps. The map file is $(M, 1) = (11, 1)$ The noise covariance matrix is a diagonal matrix of $(M, M) = (11, 11)$ elements. The kernel file is a two dimensional matrix, $(M, N_z) = (11, 30)$ and the regularization matrix is $(N_z, N_z) = (30, 30)$. For all the FITS files, the first element of a matrix, M(X=0, Y=0) is equal to FITS Image (X=1,Y=1). The range of values for the lambda parameter contains a minimum value, a maximum value and the number of steps between the minimum and the maximum. If the keyword 'log' specifies the number of steps, then the inversion code takes logarithmic steps from the minimum to the maximum. For lambda set (0.5, 1.0, 2), the inversion is calculated at lambda = 0.5 and 1.0. For lambda set (0, 1000, log), the inversion is calculated at lambda = 0.0, 1, 10, 100, and 1000. An example input file is located in Examples/testdata/testInputFile and is shown below:

```
1d rls
 ./Examples/testdata/testOutput/1d
```

```
! For vector data, either .fits file or an ASCII text (.txt) file
! will work.
./Examples/testdata/input/1d/d.fits
% another comment
./Examples/testdata/input/1d/C.fits
./Examples/testdata/input/1d/M.fits
./Examples/testdata/input/1d/R.fits
! lambdaMin lambdaMax NLambda
0.5 1.0 2
! comments at end of file
```

2.1.2. *Output Files.* In the 1-D problem, there is one solution file, b.fits. B.fits contains column vectors, $b(\lambda_i)$, which are solution vectors for each lambda value, λ_i. The b.fits file contains a 2-D matrix, $(N_\lambda, N_z) = (2, 30)$,

$$[b(\lambda_1)...b(\lambda_N)]$$

The FITS file header contains the lambda value for each averaging kernel and error covariance. For an input file containing lambda set (0.5, 1.0, 2) the output files are

```
/output/base/directory/b.fits
/output/base/directory/1_A.fits
/output/base/directory/1_Cov.fits
/output/base/directory/2_A.fits
/output/base/directory/2_Cov.fits
```

2.2. **3-D Scalar Problem.** The 3-D problem can have multiple kernel files and multiple map files, but there must be the same number of kernel files as map files, since they are paired. Each kernel file contains a 3-dimensional matrix (N_x, N_y, N_z) and each map file contains a 2-dimensional matrix (N_x, N_y). All input files are in FITS format. Since calculating the averaging kernels and error covariance is costly for each lambda value, an additional parameter specifies the frequency of averaging kernel calculations. Lambda set specification is the same as the 1-D problem, with the additional of a second lambda parameter.

2.2.1. *Contents of Input File.* The input-file contains the following information:

(1) which problem (3d_scalar)
(2) output base directory
(3) base directory for Kernel files
(4) base directory for Map files
(5) kernel-map pairs
(6) input noise covariance matrix (C.fits)
(7) input Regularization matrix file (R.fits)
(8) Averaging Kernel specification
(9) lambda1 lambda2

The order of the kernel-map pairs is important. The map numbers go from the first kernel-map pair (1) to the last kernel-map pair (M). In the following example, the map dimension goes from 1 to 3 and $M = 3$. The noise covariance file contains a unique covariance matrix (M, M) for each (N_x, N_y). The noise covariance file is then a four dimensional FITS file with dimensions (N_x, N_y, M, M).

'Avg' and 'Kernel' are keywords and begin the averaging kernel specification. The numbers following the keywords specify the target depths at which to calculate averaging kernels. The target depths are the same as the kernel depth (N_z) and they begin at 1 and end at N_z. A depth of -1 means averaging kernel calculations are omitted. If no numbers follow the keywords, then averaging kernel calculations occur at every depth. The error covariance is calculated at the same target depths as the averaging kernels.

Example input file:

```
! All lines beginning with a ! or a % are comments
! which problem
3d_scalar rls
!
! base directory for all output files
/data/cora/output
!
! base directory for Kernel files
/data/cora/Kernels
!
! base directory for Map files (input data files)
/data/cora/Maps
!
! input Kernel file and Map file pairs; relative paths
Kern1.fits Map1.fits
Kern2.fits Map2.fits
Kern3.fits Map3.fits
!
! input noise covariance matrix (C.fits); Full path
/full/path/CI.fits
!
! input Regularization matrix file (R.fits); Full path
/full/path/R.fits
!
! -1 indicates don't calculate averaging kernel and Cov
! Avg Kernel -1
! numbers indicate which depth to calculate Averaging Kernels and Cov
! the numbers correspond to the 3rd dimension in the M.fits files (e.g. 160x160x35)
! then Avg Kernel can range from 1 to 35
Avg Kernel 10  20
```

```
!
! no values after 'Avg Kernel' keywords, indicates all averaging kernels are calculated
! Avg Kernel
!
! lambda1 set  lambda2 set
! Lambda1Min Lambda1Max NLambda1 Lambda2Min Lambda2Max NLambda2
    0.5  1.0   2      0.0  1.0    1
```

2.2.2. *Output Files.* The output files are the same as in the 1-D problem, except the solution, b, is now a matrix and written to a separate FITS file for each lambda value. An additional output file, d∼, contains the map data calculated from the input kernel and the solution, b. FITS file headers record the specific lambda values used to generate the data.

For the example input file above, the output files are

```
/output/base/directory/1_1_b.fits
/output/base/directory/1_1_d~1.fits
/output/base/directory/1_1_d~2.fits
/output/base/directory/1_1_d~3.fits
/output/base/directory/1_1_AvgKern_10.fits
/output/base/directory/1_1_Cov_10.fits
/output/base/directory/1_1_AvgKern_20.fits
/output/base/directory/1_1_Cov_20.fits

/output/base/directory/2_1_b.fits
/output/base/directory/2_1_d~1.fits
/output/base/directory/2_1_d~2.fits
/output/base/directory/2_1_d~3.fits
/output/base/directory/2_1_Cov_20.fits
/output/base/directory/2_1_AvgKern_10.fits
/output/base/directory/2_1_Cov_10.fits
/output/base/directory/2_1_AvgKern_20.fits
/output/base/directory/2_1_Cov_20.fits
```

2.3. **3-D Vector Problem.** In the 3-D Vector problem, instead of each point being a single numeric value (a scalar), it is a vector with three numeric values. The vector is spread across three kernel files. The first kernel file contains values for the first element in the vector, the second kernel file contains the second element of the vector, and so on. Each kernel-map pair now has three kernel files associated with a single map and a kernel-map pair is really a quadruple instead of a pair. For example, if a flow vector has dimensions x,y, and z, then a kernel-map quadruple might look like this:

```
kernel1_x.fits kernel1_y.fits kernel1_z.fits map1.fits
```

Each kernel file is a 3-dimensional matrix (N_x, N_y, N_z) and each map file is a 2-dimensional matrix (N_x, N_y) – the same as the 3-D scalar problems. The difference for 3-D vector

problems is in the number of kernel files for each map. The noise covariance matrix has dimension (N_x, N_y, M, M), where M is the number of maps. The regularization file has dimensions $(3(N_z), 3(N_z))$. All input files are in FITS format. Lambda set, averaging kernel and error covariance specifications remain the same as in the 3-D scalar problem.

2.3.1. *Contents of Input File.* The input-file contains the following information:

(1) which problem (3d_vector)
(2) output base directory
(3) base directory for Kernel files
(4) base directory for Map files
(5) kernel-map quadruples
(6) input noise covariance matrix (C.fits)
(7) input Regularization matrix file (R.fits)
(8) Averaging Kernel specification
(9) lambda sets ....

The order of the kernel-map quadruples is important and follows the same convention as the 3-D scalar problem.

Example input file:

```
! All lines beginning with a ! or a % are comments
! which problem
3d_vector rls
!
! base directory for all output files
/data/cora/output
!
! base directory for Kernel files
/data/cora/Kernels
!
! base directory for Map files (input data files)
/data/cora/Maps
!
! input Kernel file and Map file pairs; relative paths
Kern1_x.fits Kern1_y.fits Kern1_z.fits Map1.fits
Kern2_x.fits Kern2_y.fits Kern2_z.fits Map2.fits
Kern3_x.fits Kern3_y.fits Kern3_z.fits Map3.fits
!
! input noise covariance matrix (C.fits); Full path
/full/path/C.fits
!
! input Regularization matrix file (R.fits); Full path
/full/path/R.fits
!
! -1 indicates don't calculate averaging kernel and Cov
```

```
! Avg Kernel -1
! numbers indicate which depth to calculate Averaging Kernels and Cov
! the numbers correspond to the 3rd dimension in the M.fits files (e.g. 160x160x35)
! then Avg Kernel can range from 1 to 35
Avg Kernel 10
!
! no values after 'Avg Kernel' keywords, indicates all averaging kernels are calculated
! Avg Kernel
!
! lambda sets ...
! Lambda1Min Lambda1Max NLambda1 Lambda2Min Lambda2Max NLambda2
    0.5  1.0   2      0.0  1.0    1
```

2.3.2. *Output Files.* The output files are similar to the output files in the 3-D scalar problem with each dimension of the vector broken into separate output files.

For the example input file above, the output files are

```
/output/base/directory/1_1_bv_x.fits
/output/base/directory/1_1_bv_y.fits
/output/base/directory/1_1_bv_z.fits
/output/base/directory/1_1_d~1.fits
/output/base/directory/1_1_d~2.fits
/output/base/directory/1_1_d~3.fits
/output/base/directory/1_1_AvgKern_10_xx.fits
/output/base/directory/1_1_AvgKern_10_xy.fits
/output/base/directory/1_1_AvgKern_10_xz.fits
/output/base/directory/1_1_AvgKern_10_yx.fits
/output/base/directory/1_1_AvgKern_10_yy.fits
/output/base/directory/1_1_AvgKern_10_yz.fits
/output/base/directory/1_1_AvgKern_10_zx.fits
/output/base/directory/1_1_AvgKern_10_zy.fits
/output/base/directory/1_1_AvgKern_10_zz.fits
/output/base/directory/1_1_Cov_10_xx.fits
/output/base/directory/1_1_Cov_10_xy.fits
/output/base/directory/1_1_Cov_10_xz.fits
/output/base/directory/1_1_Cov_10_yx.fits
/output/base/directory/1_1_Cov_10_yy.fits
/output/base/directory/1_1_Cov_10_yz.fits
/output/base/directory/1_1_Cov_10_zx.fits
/output/base/directory/1_1_Cov_10_zy.fits
/output/base/directory/1_1_Cov_10_zz.fits

/output/base/directory/2_1_bv_x.fits
```

```
/output/base/directory/2_1_bv_y.fits
/output/base/directory/2_1_bv_z.fits
/output/base/directory/2_1_d~1.fits
/output/base/directory/2_1_d~2.fits
/output/base/directory/2_1_d~3.fits
/output/base/directory/2_1_AvgKern_10_xx.fits
/output/base/directory/2_1_AvgKern_10_xy.fits
/output/base/directory/2_1_AvgKern_10_xz.fits
/output/base/directory/2_1_AvgKern_10_yx.fits
/output/base/directory/2_1_AvgKern_10_yy.fits
/output/base/directory/2_1_AvgKern_10_yz.fits
/output/base/directory/2_1_AvgKern_10_zx.fits
/output/base/directory/2_1_AvgKern_10_zy.fits
/output/base/directory/2_1_AvgKern_10_zz.fits
/output/base/directory/2_1_Cov_10_xx.fits
/output/base/directory/2_1_Cov_10_xy.fits
/output/base/directory/2_1_Cov_10_xz.fits
/output/base/directory/2_1_Cov_10_yx.fits
/output/base/directory/2_1_Cov_10_yy.fits
/output/base/directory/2_1_Cov_10_yz.fits
/output/base/directory/2_1_Cov_10_zx.fits
/output/base/directory/2_1_Cov_10_zy.fits
/output/base/directory/2_1_Cov_10_zz.fits
```

Notice, there are now nine files for each averaging kernel target depth. The important averaging kernels are the diagonal ones (_xx, _yy, and _zz). The off-diagonal averaging kernels (_xy, _zx, etc.) do not really make sense and may be removed in a future release.

3. Possible Errors

The files errorHandler.h *ErrorMessages.h, and scripts/messages.py contain most of the errors reported by the software. Third party software packages report a few additional errors. These include error messages from CFITSIO, LAPACK, and FFTW which the inversion software catches and replaces with error messages contained in errorHandler.h. Some error messages from CFITSIO library propagate without wrapping or replacement.

Generally, the errors come from unexpected input, runtime complications or a problem writing to files.

- Unexpected input:
  - matrix and vectors of incompatible sizes
  - matrix and vectors are too large
  - input files don't exist
- Run-time complications:
  - Linear equation solve fails to converge. The resulting output is NaN and an error message is reported.
  - Out of memory error; check that all mallocs are successful, otherwise, report error and abort.
- Problems with output:
  - cannot write output files
  - directory does not exist