Stanford Exploration Project, Report 108, April 29, 2001, pages 1–??

`oclib`
An out-of-core optimization library

Paul Sava[1]

ABSTRACT

This paper introduces a `Fortran90` out-of-core optimization library designed for large-scale problems. The library is centered around the filtering operators and gradient solvers currently in use at SEP.

MOTIVATION

Migration and modelling are adjoint operators. The industrial practice is to use migration (the adjoint operator) instead of inversion (the inverse operator), since the adjoint and inverse operators are not too different from each-other. In various circumstances, however, we can find it useful to experiment with inversion rather than migration, either to asses the quality of our approximations, or to shape our images according to various styling goals.

Since our model and data spaces can be rather large for realistic problems, and since the memory of our computers, although growing, is still not large enough to solve all our problems in-core, I introduce a library that can be used to perform out-of-core optimization.

The main features of the library are: an adjustable memory parameter specifying the amount of data that can be loaded in core at a given time; support for both real and complex optimization; various gradient solvers mimmicking those introduced in Clearbout's most recent book (Geophysical Estimation by Example); and various commonly-used operators (like `helicon` or `polydiv`).

This paper is designed to be more of a reference to the library than a tutorial, although I also present a simple program which illustrates how the library operates.

INTRODUCTION

Geophysical data analysis often calls for reconstruction of a model of the Earth's subsurface (the model, $\mathbf{m}$) from measurements of a physical quantity recorded at some distance (the data, $\mathbf{D}$). The relationship between the model and the data are often non-linear. However, in

[1]**email:** paul@sep.stanford.edu

practice, we linearize this relationship, so that we can formulate and attempt to solve a problem in linear form.

Mathematically, we can represent our optimization problem through an equation like

$$\mathcal{L}\mathbf{m} \approx \mathbf{D},$$

where $\mathcal{L}$ is the linearized operator relating the model $\mathbf{m}$ and the data $\mathbf{D}$ (Claerbout, 1999).

Common industrial practice is to recover the model by applying the adjoint of the operator $\mathcal{L}$ to the recorded data

$$\mathbf{m} \approx \mathcal{L}^*\mathbf{D},$$

where $\mathcal{L}^*$, the adjoint operator, is an approximation to the operator we would really like to apply, $\mathcal{L}^{-1}$.

The adjoint approximation is not only very convenient to use, but it also appears to produce good results in practice. However, there are situations when, given that $\mathcal{L}$ is not at all unitary, the model we produce is only a distant approximation of the true one.

In the particular case of seismic imaging, the model (reflectivity map) and the data (seismic data) are extremely large. It becomes therefore, completely infeasible to compute $\mathcal{L}^{-1}$ other than by an iterative application of the operator $\mathcal{L}$ and its adjoint. Furthermore, the model and the data are far too large to be kept in the RAM memory of current computers.

A solution for the large size problems is to implement inversion in an out-of-core fashion, where only limited chunks of the model and data are kept in memory at any given time. This is the purpose of the optimization library I am introducing in this paper.

Generally speaking, the types of problems that can be solved using this library are regularized inversion in standard form

$$\begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix}\mathbf{m} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}, \tag{1}$$

or in its preconditioned form

$$\begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix}\mathbf{p} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}, \tag{2}$$

where $\mathcal{A}$ is a regularization operator, $\mathcal{W}$ is a weighting operator, $\mathcal{P}$ is a preconditioning operator, and $\mathbf{p}$ is the preconditioned form of $\mathbf{m}$.

The operators $\mathcal{L}$, $\mathcal{A}$, $\mathcal{W}$, $\mathcal{P}$ are application-dependent and therefore have to be externally implemented, likely in an out-of-core manner, by the user of the library. All other operations needed to solve the inversion problem are build into `oclib`.

Although other languages allow for more creative implementation, this entire library is implemented in `Fortran90` mainly because this is still the programming language of choice in scientific computing and also the language most commonly used at SEP (Claerbout, 1999).

Applications

Several applications, including some presented in the current report, use this library. Here is a very brief description of some:

- **Wave-equation migration velocity analysis**

  The inversion solves the problem in Equation (1), where the model **m** is represented by slowness perturbation, and the data **D** is given by the measured image perturbation (Biondi and Sava, 1999; Sava and Biondi, 2000).

- **Least-squares inversion**

  The inversion solves the problem in Equation (1), where the model **m** is the seismic image in the angle-domain, and **D** is the recorded seismic data (Prucha et al., 2001; Rickett, 2001).

Interface design

The library operates with two fundamental objects, model/data vectors and operators. All vectors are SEP files stored on a disk, and are represented inside the library as SEP file tags (Nichols et al., 1994). The operators are function calls that must conform with the following interface: `function oc_operator(adj,add, x_,yy_, op1 ... op9) result(stat)`

where

- `stat`[logical] is a success flag
- `adj` [logical] is a flag signaling the adjoint operator
- `add` [logical] is a flag specifying if the result of the operator is to be added to the output
- `x_` [char(len=128)] is a file tag in the model space
- `yy_` [char(len=128)] is a file tag in the data space
- `op1 ... op9`[integer,optional] are (9) secondary operators used by the main operator.

The remaining of this paper functions as a reference of the main functions and subroutines available in the current version of `oclib`. Appendices F and F present a simple example (the `Makefile` and the main program). Next table presentes a list of the main `Fortran90` modules that compose the library.

Algebraic operations on files	module `oc_file_mod`
	module `of_filealgebra_mod`
	module `oc_filter_mod`
Out-of-core operators	module `oc_adjnull_mod`
	module `oc_scale_mod`
	module `oc_weight_mod`
	module `oc_helicon_mod`
	module `oc_polydiv_mod`
	module `oc_laplacian_mod`
	module `oc_helocut_mod`
	module `oc_dottest_mod`
	module `oc_combine_mod`
Out-of-core gradient solvers	module `oc_solver_mod`
	module `oc_solverreg_mod`
	module `oc_solverpre_mod`
	module `oc_sd_mod`
	module `oc_cg_mod`
	module `oc_cgstep_mod`
	module `oc_cd_mod`
	module `oc_gmres_mod`
Out-of-core LSQR solvers	module `oc_lsqr_mod`
	module `oc_lsqrreg_mod`
	module `oc_lsqrpre_mod`

The appendices A, B, C, and D present a detailed description of the interfaces for the main functions and subroutines in each module.

ACKNOWLEDGEMENTS

Bob Clapp is my main guide through the SEPlib maze. Marie Prucha, Biondo Biondi and James Rickett provided valuable feedback while testing and using `oclib`.

REFERENCES

Biondi, B., and Sava, P., 1999, Wave-equation migration velocity analysis: SEP–**100**, 11–34.

Claerbout, J., 1999, Geophysical estimation by example: Environmental soundings image enhancement: Stanford Exploration Project, `http://sepwww.stanford.edu/sep/prof/`.

Nichols, D., Karrenbach, M., and Urdaneta, H., 1994, What's new in SEPLIB?: SEP–**82**, 257–264.

Prucha, M. L., Clapp, R. G., and Biondi, B. L., 2001, Imaging under salt edges: A regularized least-squares inversion scheme: SEP–**108**, 91–104.

Rickett, J., 2001, Model-space vs data-space normalization for finite-frequency depth migration: SEP–**108**, 81–90.

Sava, P., and Biondi, B., 2000, Wave-equation migration velocity analysis: Episode II: SEP–**103**, 19–47.

APPENDIX A

Module `oc_file_mod`

1. **Purpose**: defines the *fileinfo* type and basic operations on files.
2. **Types**

   (a) *fileinfo*:

member	type	description
name	`character(len=128)::`	file tag
nd	`integer::`	number of file dimensions
esize	`integer::`	file element size
n	`integer(nd)::`	SEPlib n for file
o	`real(nd)::`	SEPlib o for file
d	`real(nd)::`	SEPlib d for file
blon	`integer::`	number of blocks
bloe	`integer::`	number of elements in a block
blob	`integer::`	number of bytes in a block

3. **Functions and subroutines**

   (a) `subroutine oc_allocatefile(file, t_, maxmem)`
   Purpose: allocate an object of type *fileinfo*

   (b) `subroutine oc_deallocatefile(file)`
   Purpose: deallocate an object of type *fileinfo*

   (c) `subroutine oc_infofile(file)`
   Purpose: print the file informations

   (d) `subroutine oc_checksimilarity(file1,file2,caller)`
   Purpose: check if two files have identical spaces and elements
   - `caller`: string identifying the caller (optional)

   (e) `subroutine oc_checkspace(file1,file2,caller)`
   Purpose: check if two files have identical spaces
   - `caller`: string identifying the caller (optional)

   (f) `function oc_allocate(tmp,name,esize,n,o,d) result(t_)`
   Purpose: allocate a new file
   - `t_`: file tag
   - `tmp`: flag for temporary files
   - `name`: file name
   - `esize`: SEPlib esize (optional)
   - `n,o,d`: SEPlib n,o,d (optional)

(g) `function oc_clone(tmp, t0_,name, maxmem) result(t_)`
Purpose: duplicate the structure of a file in a new file

- `t_`: new file tag
- `tmp`: flag for temporary files
- `t0_`: old file tag
- `name`: new file name

(h) `subroutine oc_append(t_,s_, maxmem)`
Purpose: append the contents of file `s_` at the end of file `t_`

(i) `subroutine oc_adddim(t_, nnew)`
Purpose: add a new axis to a file

- `nnew`: SEPlib n

(j) `subroutine oc_shapeheader(t_, esize, n,o,d)`
Purpose: shape a file header

- `t_`: file tag
- `name`: file name
- `esize`: SEPlib esize
- `n`: SEPlib n
- `o,d`: SEPlib o,d (optional)

(k) `subroutine oc_print(t_, maxmem)`
Purpose: print the contents of a file

Module `oc_filealgebra_mod`

1. **Purpose**: defines algebraic operations on files
2. **Functions and subroutines**

(a) `function oc_rdp(t1_,t2_,maxmem) result(dp)`
Purpose: dot product of two real files

- `t1_`: vector of file tags
- `t2_`: vector of file tags

(b) `function oc_cdp(t1_,t2_,maxmem) result(dp)`
Purpose: dot product of two complex files

- `t1_`: vector of file tags
- `t2_`: vector of file tags

(c) `subroutine oc_assign(t_, sca, maxmem)`
Purpose: assign a value to an entire file

- `sca`: scalar (real or complex)

(d) `subroutine oc_linear(t0_, ti_, scai, maxmem)`

Purpose: linear combinations of files

- `t_`: output file tag
- `ti_`: vector of input file tags
- `scai`: vector of scalars to multiply the file tags

(e) `subroutine oc_random(t_, scale, maxmem)`

Purpose: fill a file with random numbers (real or complex)

- `scale`: scaling factor for the ramdom numbers

(f) `subroutine oc_complexify(t_, maxmem)`

Purpose: complexify a file

(g) `subroutine oc_mask(k_,t_,maxmem)`

Purpose: mask a file with another file

- `k_`: mask file tag
- `t_`: data file tag

(h) `subroutine oc_product(t0_, ti_, maxmem)`

Purpose: product of files

- `t0_`: product file tag
- `ti_`: vector of input file tags

(i) `function oc_norm(t_, maxmem) result(norm)`

Purpose: return the norm of a vector

- `t_`: vector of file tags
- `norm`: (real) norm of the data in file `t_`

(j) `subroutine oc_normalize(t_, magnitude, maxmem)`

Purpose: normalize a file

- `t_`: vector of file tags
- `magnitude`: magnitude of the data in file `t_`

Module `oc_filter_mod`

1. **Purpose**: definitions of the out-of-core helix filters

2. **Types**

(a) *rfilter*:

member	type	description
flt	real(:)	filter coefficients
lag	integer(:)	filter lags

(b) *cfilter*:

member	type	description
flt	complex(:)	filter coefficients
lag	integer(:)	filter lags

3. Functions and subroutines

(a) `subroutine allocatehelix(aa, nh)`

Purpose: allocate space for the filter coefficients

- `aa`: helix filter (real or complex)
- `nh`: number of coefficients

(b) `subroutine deallocatehelix(aa)`

Purpose: deallocate filter space

- `aa`: helix filter (real or complex)

(c) `subroutine buildfilter(ff,x_,fbox,nf,maxmem)`

Purpose: build a helix filter

- `ff`: output filter (real or complex)
- `x_`: file tag for the filtering space
- `fbox`: multidimensional array with the filter coefficients
- `nf`: number of coefficients

(d) `subroutine printfilter(ff,nf)`

Purpose: print the filter coefficients

- `ff`: filter (real or complex)
- `nf`: number of coefficients

APPENDIX B

Module `oc_adjnull_mod`

1. **Purpose**: nullify the output of an operator
2. **Functions and subroutines**
   (a) `subroutine oc_adjnull(adj,add, x_,yy_)`
   Purpose: nullify operator output

Module `oc_scale_mod`

1. **Purpose**: scaling operator
2. **Functions and subroutines**
   (a) `subroutine oc_scale_init(eps,maxmem)`
   Purpose: initialize the scaling operator
   - `eps`: scaling parameter (real or complex)

   (b) `function oc_scale(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: scaling operator

Module `oc_weight_mod`

1. **Purpose**: weighting operator
2. **Functions and subroutines**
   (a) `subroutine oc_weight_init(w_,maxmem)`
   Purpose: initialize the weighting operator
   - `w_`: weighting file tag (real or complex)

   (b) `function oc_weight(adj,add, x_,yy_, op1 ... op9) result(stat)`
   Purpose: weighting operator

Module `oc_helicon_mod`

1. **Purpose**: convolution on a helix
2. **Functions and subroutines**
   (a) `subroutine oc_helicon_init(aa,maxmem)`
   Purpose: initialize the helicon operator

- `aa`: helix filter (real or complex)

(b) `function oc_helicon(adj,add, x_,yy_,op1 ... op9) result(stat)`
Purpose: helical convolution

Module `oc_polydiv_mod`

1. **Purpose**: polynomial division on a helix
2. **Functions and subroutines**

   (a) `subroutine oc_polydiv_init(aa,maxmem)`
   Purpose: initialize polydiv
   - `aa`: helix filter (real or complex)

   (b) `function oc_polydiv(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: helical polynomial division

Module `oc_laplacian_mod`

1. **Purpose**: Laplacian and similar operators.
2. **Functions and subroutines**

   (a) `oc_laplacian_init(t_,nf,niter,maxmem)`
   Purpose: initialize the laplacian operators
   - `t_`: filtering file tag
   - `nf`: number of filter coefficients
   - `niter`: number of Wilson iterations

   (b) `subroutine oc_laplacian_factor(bb,t_,nf,niter,maxmem)`
   Purpose: find a laplacian minimum-phase factor
   - `bb`: laplacian minimum-phase factor (real or complex)
   - `t_`: filtering file tag
   - `nf`: number of filter coefficients
   - `niter`: number of Wilson iterations

   (c) `function oc_laplacian(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: laplacian operator

   (d) `function oc_ilaplacian(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: inverse laplacian operator

   (e) `function oc_hderivative(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: helix derivative operator

   (f) `function oc_ihderivative(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: inverse helix derivative operator

Module `oc_helocut_mod`

1. **Purpose**: Helix low-cut filter
2. **Functions and subroutines**
   (a) `subroutine oc_helocut_init(aa,maxmem)`
   Purpose: initialize the helocut operator
   - `aa`: helix filter (real or complex)

   (b) `function oc_helocut(adj,add, x_,yy_,op1 ... op9) result(stat)`
   Purpose: helix low-cut filter

Module `oc_dottest_mod`

1. **Purpose**: dot product test on out-of-core operators
2. **Functions and subroutines**
   (a) `subroutine oc_dottest_init(no_add,adj_first,maxmem)`
   Purpose: init dot product test
   - `no_add`: skip DP test for `add=.true.`
   - `adj_first`: start DP test with the adjoint

   (b) `subroutine oc_dottest(oper, x_,yy_,op1 ... op9)`
   Purpose: dot product test
   - `oper`: out-of-core operator

Module `oc_combine_mod`

1. **Purpose**: combined out-of-core operators.
2. **Functions and subroutines**
   (a) `subroutine oc_chain(A,B,C adj,add, x_,yy_,op1 ... op9)`
   Purpose: chain operators (overloaded)
   $$\mathbf{D} = \mathcal{A}\mathbf{m}$$
   $$\mathbf{D} = \mathcal{A}\mathcal{B}\mathbf{m}$$
   $$\mathbf{D} = \mathcal{A}\mathcal{B}\mathcal{C}\mathbf{m}$$
   - `A,B,C`: out-of-core operators

   (b) `subroutine oc_row(A1,A2, adj,add, x1_,x2_,y1_,op1 ... op9)`
   Purpose: row combined operator
   $$[\mathcal{A}_1 \quad \mathcal{A}_2]\begin{bmatrix}\mathbf{m}_1\\ \mathbf{m}_2\end{bmatrix} = \mathbf{D}$$
   - `A1,A2`: out-of-core operators

   (c) `subroutine oc_column11(A1,eps,A2, adj,add, x_,y1_,y2_, maxmem,op1 ... op9)`
   Purpose:
   $$\begin{bmatrix}\mathcal{A}_1\\ \epsilon\mathcal{A}_2\end{bmatrix}\mathbf{m} = \begin{bmatrix}\mathbf{D}_1\\ \mathbf{D}2\end{bmatrix}$$
   - `A1,A2`: out-of-core operators
   - `eps`: scaling factor

   (d) `subroutine oc_column20(A1,B1,eps, adj,add, x_,y1_,y2_, maxmem,op1 ... op9)`
   Purpose:
   $$\begin{bmatrix}\mathcal{A}_1\mathcal{B}_1\\ \epsilon I\end{bmatrix}\mathbf{m} = \begin{bmatrix}\mathbf{D}_1\\ \mathbf{D}2\end{bmatrix}$$
   - `A1,B1`: out-of-core operators
   - `eps`: scaling factor

   (e) `subroutine oc_column21(A1,B1,eps,A2, adj,add, x_,y1_,y2_, maxmem,op1 ... op9)`
   Purpose:
   $$\begin{bmatrix}\mathcal{A}_1\mathcal{B}_1\\ \epsilon\mathcal{A}_2\end{bmatrix}\mathbf{m} = \begin{bmatrix}\mathbf{D}_1\\ \mathbf{D}2\end{bmatrix}$$

- `A1,A2,B1`: out-of-core operators
- `eps`: scaling factor

(f) `subroutine oc_column30(A1,B1,C1,eps, adj,add, x_,y1_,y2_, maxmem,op1 ... op9)`

Purpose:

$$\begin{bmatrix} \mathcal{A}_1 \mathcal{B}_1 \mathcal{C}_1 \\ \epsilon I \end{bmatrix} \mathbf{m} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}2 \end{bmatrix}$$

- `A1,B1,C1,`: out-of-core operators
- `eps`: scaling factor

APPENDIX C

Module `oc_solver_mod`

1. **Purpose**: implements a basic least-squares gradient solver

$$\boxed{\mathcal{L}\mathbf{m} \approx \mathbf{D}}$$

$$\begin{array}{l} \mathbf{R} = \mathcal{L}\mathbf{m}_0 - \mathbf{D} \\ \text{iterate } \{ \\ \qquad \mathbf{g} = \mathcal{L}^*\mathbf{R} \\ \qquad \mathbf{G} = \mathcal{L}\mathbf{g} \\ \qquad (\mathbf{m},\mathbf{R}) \longleftarrow \texttt{step}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G}) \\ \qquad \} \end{array}$$

2. **Functions and subroutines**

   (a) `subroutine oc_solver_init(niter,maxmem,verb,mmovie,dmovie,resstop)`

   Purpose: initialize the basic solver

   - `niter`: iterations number
   - `verb`: verbose flag (optional)
   - `mmovie`: model movie output flag (optional)
   - `dmovie`: data movie output flag (optional)
   - `resstop`: stop iterations at this residual power (optional)

   (b) `subroutine oc_solver( L,S, x_,yy_, x0_,res_,op1 ... op9)`

   Purpose: simple solver

   - `L`: out-of-core linear operator
   - `S`: gradient step
   - `x0_`: starting model tag
   - `res_`: residual tag

Module `oc_solverreg_mod`

1. **Purpose**: implements a regularized least-squares gradient solver

$$\boxed{\begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix} \mathbf{m} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}}$$

$$\begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix} \mathbf{m}_0 - \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}$$

iterate {

$$\mathbf{g} = \begin{bmatrix} \mathcal{L}^*\mathcal{W}^* & \epsilon\mathcal{A}^* \end{bmatrix} \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{G}_d \\ \mathbf{G}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix} \mathbf{g}$$

$$\left(\mathbf{m}, \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}\right) \longleftarrow \mathtt{step}\left(\mathbf{m}, \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}, \mathbf{g}, \begin{bmatrix} \mathbf{G}_d \\ \mathbf{G}_m \end{bmatrix}\right)$$

}

2. **Functions and subroutines**

   (a) `subroutine oc_solverreg_init(niter,eps,maxmem,verb,mmovie,dmovie,resstop,rescale)`
   Purpose: initialize the regularized solver
   - `niter`: iterations number
   - `eps`: scaling factor
   - `verb`: verbose flag (optional)
   - `mmovie`: model movie output flag (optional)
   - `dmovie`: data movie output flag (optional)
   - `resstop`: stop iterations at this residual power (optional)
   - `rescale`: rescale model (optional)

   (b) `subroutine oc_solverreg( L,A,S, x_,yy_, nreg, W ,k_,x0_,res_,op1 ... op9)`
   Purpose: regularized solver
   - `L`: out-of-core linear operator
   - `A`: out-of-core regularization operator
   - `S`: gradient step
   - `nreg`: dimension of the regularization output
   - `W`: out-of-core weighting operator
   - `k_`: data mask tag
   - `x0_`: starting model tag
   - `res_`: residual tag

Module `oc_solverpre_mod`

1. Purpose: implements a preconditioned least-squares gradient solver

$$\boxed{\begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix} \mathbf{p} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}}$$

$$\begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix} \mathbf{p}_0 - \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}$$

iterate {

$$\mathbf{g} = \begin{bmatrix} \mathcal{P}^*\mathcal{L}^*\mathcal{W}^* & \epsilon I \end{bmatrix} \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{G}_d \\ \mathbf{G}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix} \mathbf{g}$$

$$\left(\mathbf{p}, \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}\right) \longleftarrow \texttt{step}\left(\mathbf{p}, \begin{bmatrix} \mathbf{R}_d \\ \mathbf{R}_m \end{bmatrix}, \mathbf{g}, \begin{bmatrix} \mathbf{G}_d \\ \mathbf{G}_m \end{bmatrix}\right)$$

}

$$\mathbf{m} = \mathcal{P}\mathbf{p}$$

2. **Functions and subroutines**

   (a) `subroutine oc_solverpre_init(niter,eps,maxmem,verb,mmovie,dmovie,resstop,rescale)`

   Purpose: initialize the preconditioned solver

   - `niter`: iterations number
   - `eps`: scaling factor
   - `verb`: verbose flag (optional)
   - `mmovie`: model movie output flag (optional)
   - `dmovie`: data movie output flag (optional)
   - `resstop`: stop iterations at this residual power (optional)
   - `rescale`: rescale model (optional)

   (b) `subroutine oc_solverpre( L,P,S, x_,yy_, npre, W ,k_,p0_,res_,op1 ... op9)`

   Purpose: preconditioned solver

   - `L`: out-of-core linear operator
   - `P`: out-of-core preconditioning operator
   - `S`: gradient step
   - `npre`: dimension of the preconditioning output
   - `W`: out-of-core weighting operator
   - `k_`: data mask tag
   - `p0_`: starting model tag
   - `res_`: residual tag

Module `oc_sd_mod`

1. **Purpose**: steepest descent step

$$
\text{SD}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G}) \ \{
\qquad \alpha = -\frac{(\mathbf{G}^{*}\cdot\mathbf{R})}{(\mathbf{G}^{*}\cdot\mathbf{G})}
\qquad \left\{ \begin{array}{l} \mathbf{m} = \mathbf{m} + \alpha\mathbf{g} \\ \mathbf{R} = \mathbf{R} + \alpha\mathbf{G} \end{array} \right.
\qquad \}
$$

2. **Functions and subroutines**

   (a) `integer function oc_sd(forget,x_,g_,rr_,gg_,s_,ss_,maxmem)`

   - `forget`: re-initialize operator
   - `x_`: model file tag
   - `g_`: gradient file tag
   - `rr_`: residual file tag
   - `gg_`: conjugate gradient file tag
   - `s_`: previous step file tag
   - `ss_`: conjugate previous step file tag

Module `oc_cg_mod`

1. **Purpose**: conjugate-gradient descent step

$$
\text{CG}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G}) \ \{
\qquad \beta = \frac{\|\mathbf{g}_k\|^2}{\|\mathbf{g}_{k-1}\|^2}
\qquad \left\{ \begin{array}{l} \mathbf{s} = \mathbf{g} + \beta\mathbf{s} \\ \mathbf{S} = \mathbf{G} + \beta\mathbf{S} \end{array} \right.
$$

$$
\alpha = -\frac{\|\mathbf{g}\|^2}{\|\mathbf{S}\|^2}
\qquad \left\{ \begin{array}{l} \mathbf{m} = \mathbf{m} + \alpha\mathbf{s} \\ \mathbf{R} = \mathbf{R} + \alpha\mathbf{S} \end{array} \right.
\qquad \}
$$

2. **Functions and subroutines**

   (a) `integer function oc_cg(forget,x_,g_,rr_,gg_,s_,ss_,maxmem)`

   - `forget`: re-initialize operator
   - `x_`: model file tag
   - `g_`: gradient file tag
   - `rr_`: residual file tag
   - `gg_`: conjugate gradient file tag
   - `s_`: previous step file tag
   - `ss_`: conjugate previous step file tag

Module `oc_cgstep_mod`

1. **Purpose**: conjugate-gradient descent step

$$\text{CGSTEP}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G})\ \{$$

$$\Delta = (\mathbf{G}^* \cdot \mathbf{G})(\mathbf{S}^* \cdot \mathbf{S}) - (\mathbf{S}^* \cdot \mathbf{G})(\mathbf{G}^* \cdot \mathbf{S})$$

$$\alpha = -\frac{1}{\Delta}\left[+(\mathbf{S}^* \cdot \mathbf{S})(\mathbf{G}^* \cdot \mathbf{R}) - (\mathbf{G}^* \cdot \mathbf{S})(\mathbf{S}^* \cdot \mathbf{R})\right]$$

$$\beta = -\frac{1}{\Delta}\left[-(\mathbf{G}^* \cdot \mathbf{S})(\mathbf{G}^* \cdot \mathbf{R}) + (\mathbf{G}^* \cdot \mathbf{G})(\mathbf{S}^* \cdot \mathbf{R})\right]$$

$$\left\{\begin{array}{l} \mathbf{m} = \mathbf{m} + \alpha\mathbf{g} + \beta\mathbf{s} \\ \mathbf{R} = \mathbf{R} + \alpha\mathbf{G} + \beta\mathbf{S} \end{array}\right.$$

$$\}$$

2. **Functions and subroutines**

   (a) `integer function oc_cgstep(forget,x_,g_,rr_,gg_,s_,ss_,maxmem)`
   - `forget`: re-initialize operator
   - `x_`: model file tag
   - `g_`: gradient file tag
   - `rr_`: residual file tag
   - `gg_`: conjugate gradient file tag
   - `s_`: previous step file tag
   - `ss_`: conjugate previous step file tag

Module `oc_cd_mod`

1. **Purpose**: conjugate-directions descent step

$$\text{CD}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G})\ \{$$

$$\beta_i = -\frac{(\mathbf{G}^* \cdot \mathbf{S})}{(\mathbf{S}^* \cdot \mathbf{S})} \qquad \left\{\begin{array}{l} \mathbf{s} = \mathbf{g} + \sum_{1}^{k-1} \beta_i \mathbf{s}_i \\ \mathbf{S} = \mathbf{G} + \sum_{1}^{k-1} \beta_i \mathbf{S}_i \end{array}\right.$$

$$\alpha = -\frac{(\mathbf{S}^* \cdot \mathbf{R})}{(\mathbf{S}^* \cdot \mathbf{S})} \qquad \left\{\begin{array}{l} \mathbf{m} = \mathbf{m} + \alpha\mathbf{s} \\ \mathbf{R} = \mathbf{R} + \alpha\mathbf{S} \end{array}\right.$$

$$\}$$

2. **Functions and subroutines**

   (a) `integer function oc_cd(forget,x_,g_,rr_,gg_,s_,ss_,maxmem)`

- `forget`: re-initialize operator
- `x_`: model file tag
- `g_`: gradient file tag
- `rr_`: residual file tag
- `gg_`: conjugate gradient file tag
- `s_`: previous step file tag
- `ss_`: conjugate previous step file tag

Module `oc_gmres_mod`

1. **Purpose**: generalized minimum-residual descent step

$$
\begin{array}{ll}
\multicolumn{2}{l}{\texttt{GMRES}(\mathbf{m},\mathbf{R},\mathbf{g},\mathbf{G})\ \{} \\
\beta_i = -\frac{(\mathbf{g}^*\cdot\mathbf{g}_i)}{(\mathbf{g}_i^*\cdot\mathbf{g}_i)} & \left\{\begin{array}{l}\mathbf{g} = \mathbf{g} + \sum_1^{k-1}\beta_i\mathbf{g}_i \\ \mathbf{G} = \mathbf{G} + \sum_1^{k-1}\beta_i\mathbf{G}_i\end{array}\right. \\
\gamma = \frac{(\mathbf{g}^*\cdot\mathbf{g})}{\left(\mathbf{g}_{k-1}^*\cdot\mathbf{g}_{k-1}\right)} & \left\{\begin{array}{l}\mathbf{s} = \mathbf{g} + \gamma\mathbf{s} \\ \mathbf{S} = \mathbf{G} + \gamma\mathbf{S}\end{array}\right. \\
\alpha = -\frac{(\mathbf{g}^*\cdot\mathbf{g})}{(\mathbf{S}^*\cdot\mathbf{S})} & \left\{\begin{array}{l}\mathbf{m} = \mathbf{m} + \alpha\mathbf{s} \\ \mathbf{R} = \mathbf{R} + \alpha\mathbf{S}\end{array}\right. \\
\} &
\end{array}
$$

2. **Functions and subroutines**

   (a) `integer function oc_gmres(forget,x_,g_,rr_,gg_,s_,ss_,maxmem)`

   - `forget`: re-initialize operator
   - `x_`: model file tag
   - `g_`: gradient file tag
   - `rr_`: residual file tag
   - `gg_`: conjugate gradient file tag
   - `s_`: previous step file tag
   - `ss_`: conjugate previous step file tag

APPENDIX D

Module `oc_lsqr_mod`

1. Purpose simple LSQR solver

$$\boxed{\mathcal{L}\mathbf{m} \approx \mathbf{D}}$$

$$
\begin{array}{llll}
\mathbf{m} = 0 & & & \\
\mathbf{U} = \mathbf{D} & & \beta = \sqrt{\parallel \mathbf{U} \parallel^2} & \mathbf{U} = \frac{1}{\beta}\mathbf{U} \\
\mathbf{v} = \mathcal{L}^*\mathbf{U} & & \alpha = \sqrt{\parallel \mathbf{v} \parallel^2} & \mathbf{v} = \frac{1}{\alpha}\mathbf{v} \\
\mathbf{w} = \mathbf{v} & & & \\
\bar{\rho} = \alpha & \bar{\phi} = \beta & & \\
\text{iterate } \{ & & & \\
 & \mathbf{U} = -\alpha\mathbf{U} & \mathbf{U} = \mathbf{U} + \mathcal{L}\mathbf{v} & \\
 & \beta = \sqrt{\parallel \mathbf{U} \parallel^2} & \mathbf{U} = \frac{1}{\beta}\mathbf{U} & \\
 & & & \\
 & \mathbf{v} = -\beta\mathbf{v} & \mathbf{v} = \mathbf{v} + \mathcal{L}^*\mathbf{U} & \\
 & \alpha = \sqrt{\parallel \mathbf{v} \parallel^2} & \mathbf{v} = \frac{1}{\alpha}\mathbf{v} & \\
 & & & \\
 & \rho = \sqrt{\bar{\rho}^2 + \beta^2} & & \\
 & c = \frac{\bar{\rho}}{\rho} & \bar{\rho} = -c\alpha & \\
 & s = \frac{\beta}{\rho} & \theta = s\alpha & \\
 & \phi = c\bar{\phi} & \bar{\phi} = s\bar{\phi} & \\
 & t_1 = \frac{\phi}{\rho} & t_2 = -\frac{\theta}{\rho} & \\
 & & & \\
 & \mathbf{m} = \mathbf{m} + t_1\mathbf{w} & & \\
 & \mathbf{w} = \mathbf{v} + t_2\mathbf{w} & & \\
 & \} & &
\end{array}
$$

2. **Functions and subroutines**

   (a) `subroutine oc_lsqr_init(niter,maxmem,verb,movie)`

   Purpose: initialize the simple LSQR solver

   - `niter`: iterations number
   - `verb`: verbose flag (optional)
   - `movie`: movie output flag (optional)

   (b) `subroutine oc_lsqr( L, x_,yy_,op1 ... op9)`

   Purpose: simple LSQR solver

   - `L`: out-of-core linear operator

Module `oc_lsqrreg_mod`

1. **Purpose**: regularized LSQR solver

$$\boxed{\begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix} \mathbf{m} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}}$$

$$
\begin{array}{lll}
\mathbf{m} = 0 & & \\
\begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix} & & \beta = \sqrt{\parallel \mathbf{U} \parallel^2} \quad \mathbf{U} = \frac{1}{\beta}\mathbf{U} \\
\mathbf{v} = \begin{bmatrix} \mathcal{L}^*\mathcal{W}^* & \epsilon\mathcal{A}^* \end{bmatrix} \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} & & \alpha = \sqrt{\parallel \mathbf{v} \parallel^2} \quad \mathbf{v} = \frac{1}{\alpha}\mathbf{v} \\
\mathbf{w} = \mathbf{v} & & \\
\bar{\rho} = \alpha & \bar{\phi} = \beta & \\
\text{iterate } \{ & & \\
 & \mathbf{U} = -\alpha\mathbf{U} & \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} + \begin{bmatrix} \mathcal{W}\mathcal{L} \\ \epsilon\mathcal{A} \end{bmatrix} \mathbf{v} \\
 & \beta = \sqrt{\parallel \mathbf{U} \parallel^2} & \mathbf{U} = \frac{1}{\beta}\mathbf{U} \\
 & & \\
 & \mathbf{v} = -\beta\mathbf{v} & \mathbf{v} = \mathbf{v} + \begin{bmatrix} \mathcal{L}^*\mathcal{W}^* & \epsilon\mathcal{A}^* \end{bmatrix} \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} \\
 & \alpha = \sqrt{\parallel \mathbf{v} \parallel^2} & \mathbf{v} = \frac{1}{\alpha}\mathbf{v} \\
 & & \\
 & \rho = \sqrt{\bar{\rho}^2 + \beta^2} & \\
 & c = \frac{\bar{\rho}}{\rho} & \bar{\rho} = -c\alpha \\
 & s = \frac{\beta}{\rho} & \theta = s\alpha \\
 & \phi = c\bar{\phi} & \bar{\phi} = s\bar{\phi} \\
 & t_1 = \frac{\phi}{\rho} & t_2 = -\frac{\theta}{\rho} \\
 & & \\
 & \mathbf{m} = \mathbf{m} + t_1\mathbf{w} & \\
 & \mathbf{w} = \mathbf{v} + t_2\mathbf{w} & \\
 & \} &
\end{array}
$$

2. **Functions and subroutines**

   (a) `subroutine oc_lsqrreg_init(niter,eps,maxmem,verb,movie)`

   Purpose: initialize the regularized LSQR solver

   - `niter`: iterations number
   - `eps`: scaling factor
   - `verb`: verbose flag (optional)
   - `movie`: movie output flag (optional)

   (b) `subroutine oc_lsqrreg( L,A, x_,yy_,nreg,W,op1 ... op9)`

   Purpose: regularized LSQR solver

   - `L`: out-of-core linear operator

- `A`: out-of-core regularization operator
- `nreg`: dimension of the regularization output
- `W`: out-of-core weighting operator

Module `oc_lsqrpre_mod`

1. **Purpose**: preconditioned LSQR solver

$$\boxed{\begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix} \mathbf{p} \approx \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix}}$$

$$\begin{array}{lll}
\mathbf{p} = 0 & & \\
\begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathcal{W}\mathbf{D} \\ 0 \end{bmatrix} & \beta = \sqrt{\parallel \mathbf{U} \parallel^2} & \mathbf{U} = \frac{1}{\beta}\mathbf{U} \\
\mathbf{v} = \begin{bmatrix} \mathcal{P}^*\mathcal{L}^*\mathcal{W}^* & \epsilon I \end{bmatrix} \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} & \alpha = \sqrt{\parallel \mathbf{v} \parallel^2} & \mathbf{v} = \frac{1}{a}\mathbf{v} \\
\mathbf{w} = \mathbf{v} & & \\
\bar{\rho} = \alpha \qquad \bar{\phi} = \beta & & \\
\text{iterate } \{ & & \\
\end{array}$$

$$\begin{array}{ll}
\mathbf{U} = -\alpha\mathbf{U} & \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} = \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} + \begin{bmatrix} \mathcal{W}\mathcal{L}\mathcal{P} \\ \epsilon I \end{bmatrix} \mathbf{v} \\
\beta = \sqrt{\parallel \mathbf{U} \parallel^2} & \mathbf{U} = \frac{1}{\beta}\mathbf{U} \\
& \\
\mathbf{v} = -\beta\mathbf{v} & \mathbf{v} = \mathbf{v} + \begin{bmatrix} \mathcal{P}^*\mathcal{L}^*\mathcal{W}^* & \epsilon I \end{bmatrix} \begin{bmatrix} \mathbf{U}_d \\ \mathbf{U}_m \end{bmatrix} \\
\alpha = \sqrt{\parallel \mathbf{v} \parallel^2} & \mathbf{v} = \frac{1}{\alpha}\mathbf{v} \\
& \\
\rho = \sqrt{\bar{\rho}^2 + \beta^2} & \\
c = \frac{\bar{\rho}}{\rho} & \bar{\rho} = -c\alpha \\
s = \frac{\beta}{\rho} & \theta = s\alpha \\
\phi = c\bar{\phi} & \bar{\phi} = s\bar{\phi} \\
t_1 = \frac{\phi}{\rho} & t_2 = -\frac{\theta}{\rho} \\
& \\
\mathbf{p} = \mathbf{p} + t_1\mathbf{w} & \\
\mathbf{w} = \mathbf{v} + t_2\mathbf{w} & \\
\} &
\end{array}$$

$$\mathbf{m} = \mathcal{P}\mathbf{p}$$

2. **Functions and subroutines**

   (a) `subroutine oc_lsqrpre_init(niter,eps,maxmem,verb,movie)`

   Purpose: initialize the preconditioned LSQR solver

   - `niter`: iterations number
   - `eps`: scaling factor

- `verb`: verbose flag (optional)
- `movie`: movie output flag (optional)

(b) `subroutine oc_lsqrpre( L,P, x_,yy_,npre,W,op1 ... op9)`

Purpose: preconditioned LSQR solver

- `L`: out-of-core linear operator
- `P`: out-of-core preconditioning operator
- `npre`: dimension of the preconditioning output
- `W`: out-of-core weighting operator

APPENDIX E

```
m.H:
        Spike n1=50 n2=30 nsp=6 \
        k1=17,34,27,20,33,41 l1=17,34,27,20,33,41  \
        k2=1,2,15,18,29,30   l2=1,2,15,18,29,30    \
        mag=-1,1,1,-1,1,-1 >$@

d.H: m.H $B/OCsimple
        < m.H Window >d.H
        OCsimple operation=1 model=m.H data=d.H maxmem=600 >$n

view1: m.H d.H
        < m.H Merge axis=3 space=n d.H | Window |\
        Grey pclip=100 gainpanel=a eout=1 > j.T
        < j.T Grey color=g bias=0.5 | Tube

i.H: d.H $B/OCsimple
        < m.H Window | Scale rscale=0. >$@
        OCsimple operation=2 model=i.H data=d.H \
        maxmem=600 niter=20 eps=0.02 verb=y >$n

view2: d.H i.H m.H
        < m.H Merge axis=3 space=n d.H i.H| Window |\
        Grey pclip=100 gainpanel=a eout=1 > j.T
        < j.T Grey color=g bias=0.5 \
        titles=model:data:inversion | Tube

dot: m.H d.H $B/OCsimple
        OCsimple operation=0 model=m.H data=d.H maxmem=600 >$n
```

APPENDIX F

```
program OCsimple
  use sep
  use oc_global_mod
  use oc_file_mod
  use oc_dottest_mod
  use oc_cgstep_mod
  use oc_solver_mod
  use oc_laplacian_mod

  implicit none
  logical             :: verb
  character(len=128)  :: x_,yy_,t_, name
  integer             :: maxmem, stat, niter, nf, operation
  type(fileinfo)      :: file
  type(cfilter)       :: aa
  real                :: eps

  call sep_init()
  call from_param("operation",operation,0)
  call from_param("maxmem",maxmem)
  call from_param("verb",verb,.false.)
  call from_param("nf",nf,5)
  call from_param("niter",niter,10)
  call from_param("eps",eps,1.0)
  x_= "model"; call auxinout(x_ )
  yy_="data" ; call auxinout(yy_)
  name="test.H"; t_=oc_clone(F, x_,name,maxmem)
  call sep_close()

  !! operator init
  call oc_allocatefile(file, x_, maxmem)
  call oc_infofile(file)
  do while(2*nf+1 > file%n(1))
     nf=nf-1
  end do
  call oc_deallocatefile(file)
  call oc_laplacian_init(x_,nf,10,0.0,maxmem)

  select case(operation)
  case(0) !! dot product test
     call oc_dottest_init(maxmem=maxmem)
     call oc_dottest(oc_laplacian, x_,yy_)
  case(1) !! simple forward operator
     stat = oc_laplacian(F,F,x_,yy_)
  case(2) !! inversion
     call oc_solver_init(niter,maxmem,verb)
     call oc_solver(oc_laplacian,oc_cgstep,x_,yy_)
  case default
     call seperr("missing operation")
  end select

  call exit (0)
end program OCsimple
```