JPL ECCO v3.1 Optimization Package

by
Benny Cheng, Ou Wang, and Ichiro Fukumori

Section 1. Introduction

The JPL ECCO v3.1 optimization package employs the iterative Nocedal L-BFGS quasi-Newton minimization method in parallel using MPI. The accompanying line search algorithm for the iterative solution uses a quadratic fit of the cost function using the Hessian with safeguards for large stepsizes.

Section 1.1 General Features

The Nocedal L-BFGS (limited-memory Broyden–Fletcher–Goldfarb–Shanno) quasi-Newton minimization method applies the variable storage technique of using changes in the cost gradient from one iteration to another to obtain information about the local Hessian of the objective (cost) function, without the need for exact line searches. A line search is said to be exact if the minimum value of the cost function along the direction of the line is actually attained by the search algorithm, which is usually quite prohibitive in the amount of time and computational resources required for complicated cost functions. Variable storage refers to the amount of stored information needed to approximate the Hessian matrix that are between O(n) and O(n^2) storage locations in size, where n is the size of the control space. The implementation follows closely that given in the paper "*Some Numerical Experiments with Variable Storage Quasi-Newton Algorithms*" by Gilbert and LeMarechal (1989)[GL].

Section 1.2 Methodology

The cost function **ff** computed at the end of the forward code, is generally a weighted sum of squares of model-data differences. For a given set of control vectors **xx**, the adjoint code computes the cost gradient **gg** with respect to each element of the controls. In optimization, the above pair of forward and adjoint computations (we shall call this an **iteration**) are repeated as many times as necessary, to solve for the control vectors **xx** that decrease the cost function at each iteration towards an acceptable minimum. The optimization process is governed by the executable **optim.x** and its parameter files **data.optim** and **data.ctrl** (see 1.5 below), and is executed after a forward and adjoint model run, as described in the subsequent sections.

Section 1.2.1 Initial iterations (Steepest Descent)

An initial iteration of forward and adjoint run is named iter0, usually done with **xx**=0 as there are no prior gradient information available. In the next iteration, iter1, the best (steepest descent) direction **dd** that will lead to a decrease of the cost function is given by

$$\mathbf{dd} = \mathbf{-gg} \qquad \mathbf{(1)}$$

where **gg** is the gradient obtained from the adjoint run. A suitable choice of stepsize **t** to move in this direction is given by

$$\mathbf{t} = 2(\mathbf{ff} - \mathbf{f0}) / \|\mathbf{gg}\|^2 \qquad \mathbf{(2)}$$

f0 is a preset scalar set to .99***ff**. (see the paper [GL Section 2.6]).

Once a suitable stepsize **t** and direction **dd** has been found, as above, the new control becomes

$$\mathbf{xxnew} = \mathbf{xx} + \mathbf{t} \cdot \mathbf{dd} \qquad \mathbf{(3)}$$

where **xx** is the previous set of controls (**xx** = 0 for iter1) .

Derivation of the new control **xxnew** is considered the **initial solution step**. Running the model forward with **xxnew** is expected to provide a lower cost **ffnew** over the previous iteration's cost **ff** with new gradient **ggnew**. The iteration procedure now repeats (iter1) by setting **xx** = **xxnew**, **gg** = **ggnew**, and treats this as the starting point for the next iteration (iter2 in Section 1.2.2). The following shows the input and output files from the optimization executable **optim.x** for each iteration.

INPUT	data.ctrl	Control variable information
	data.optim	Optimization parameters
	ecco_ctrl	Initial control vector
	ecco_cost	Initial cost gradient
	costfunction	Initial costfunction values
OUTPUT	OPWARMI	Optimization parameters
	OPWARMD	Gradient information
	ecco_ctrl	Steepest descent control vector
	op_i#.64	Diagnostic file for the optimization process where # is the iteration number

Section 1.2.2 Trial Step (Quasi-Newton)

For the next iteration (third iteration, iter2), **i**nstead of proceeding with another steepest descent optimization, a more sophisticated technique using the Nocedal L-BFGS quasi-Newton algorithm is employed to find the next solution. An approximate Hessian **H** is computed with available past gradients (see [GL Section 2.4]), and used to obtain an alternate direction defined as

$$\mathbf{dd} = -\mathbf{H}^{-1}\mathbf{gg} \qquad \textbf{(4)}$$

The above equation is the direction that points to the minimum value if the cost function were entirely a 2nd order quadratic function of the controls. For general cost functions, this is approximated by the 2nd order Taylor expansion. The past information used in evaluating this new direction consists of pairs of gradient and control differences (**gg(k+1)-gg(k),xx(k+1)-xx(k)**) of previous successive iterations which are saved into the file OPWARMD and are employed as described below. To simplify notation, we define

$$\begin{aligned} &\mathbf{B} = \mathbf{H}^{-1}, \\ &\mathbf{y}_\mathbf{k} = \mathbf{gg(k+1) - gg(k)},\ \mathbf{s}_\mathbf{k} = \mathbf{xx(k+1) - xx(k)}, \qquad \textbf{(5)} \\ &\mathbf{p}_\mathbf{k} = 1/\langle \mathbf{y}_\mathbf{k}, \mathbf{s}_\mathbf{k} \rangle \end{aligned}$$

where < , > is the usual inner product. Then the recursive BFGS formula ([GL eqn 2.10])

$$\mathbf{B}_{\mathbf{k}+1} = (\mathbf{I} - \mathbf{p}_\mathbf{k}\mathbf{s}_\mathbf{k}\mathbf{y}_\mathbf{k}^\mathrm{T})\mathbf{B}_\mathbf{k}(\mathbf{I} - \mathbf{p}_\mathbf{k}\mathbf{y}_\mathbf{k}\mathbf{s}_\mathbf{k}^\mathrm{T}) + \mathbf{p}_\mathbf{k}\mathbf{s}_\mathbf{k}\mathbf{s}_\mathbf{k}^\mathrm{T} \qquad \textbf{(6)}$$

provides an approximate inverse Hessian $\mathbf{B_m}$ for a given number of known pairs of $(\mathbf{y_k},\mathbf{s_k})$, **k=m-n,…m-1**, with $\mathbf{B_{m-n}}$ a diagonal matrix (see [GL eqn 4.9]);

$$\mathbf{B}^{(i)}{}_{m-n} = \left(\mathbf{p}_{m-n}\langle \mathbf{y}_{m-n}, \mathbf{y}_{m-n}\rangle + \mathbf{p}_{m-n}\mathbf{y}^{(i)}{}_{m-n}\mathbf{y}^{(i)}{}_{m-n} - \mathbf{p}_{m-n}\mathbf{s}^{(i)}{}_{m-n}\mathbf{s}^{(i)}{}_{m-n}\langle \mathbf{y}_{m-n}, \mathbf{y}_{m-n}\rangle / \langle \mathbf{s}_{m-n}, \mathbf{s}_{m-n}\rangle\right)^{-1} \qquad \textbf{(6b)}$$

. The optimization parameter **NUPDATES = n** in data.optim sets the maximum number of past gradient and control differences to be used for estimating the Hessian, and the default is 4. The choice of **n** defines the "variable storage" of the algorithm. The new control is then given by **xxnew** = **xx** + **dd**, and used as input to the next iteration. This is the **trial step**.

INPUT	data.ctrl	Control variables information
	data.optim	Optimization parameters with latest costfunction value **fc**
	ecco_ctrl	Latest iteration control vector

	ecco_cost	Latest iteration cost gradient
	costfunction	Latest iteration costfunction values
	OPWARMI	Optimization parameters
	OPWARMD	Gradients from previous iterations
OUTPUT	OPWARMI	Updated optimization parameters
	OPWARMD	Updated gradients up to trial step
	ecco_ctrl	Quasi-Newton trial control vector
	op_i#.64	Diagnostic file for optimization process

Section 1.2.3 Solution Step (Line Search with safeguards)

The new controls obtained in the trial step above is generally not a solution (a minimum value of the cost function) to the cost minimization problem, and could even increase the cost significantly. It is in fact only an acceptable iterative (and exact) solution if the cost function is exactly a pure quadratic function of the controls, as mentioned before in 1.2.2. The line search technique applied here fits a quadratic function of stepsize **t** to the points (**xx,ff**) and (**xxnew,ffnew**) of the trial step, and the known slope **<gg,dd>** at **xx** (ie. the directional derivative at t=0). We then find the stepsize **t** by solving for the minimum of this quadratic function, i.e.

$$\mathbf{t = -\langle gg,dd\rangle/(2*(ffnew - ff - \langle gg,dd\rangle))} \qquad \textbf{(7)}$$

The revised control becomes **xxc** = **xx** + **t*dd**.

A set of conditions are employed to check the quality of the line search estimated controls **xxc**. Suppose **F(x)** is the objective function, evaluated at **x**, with line search carried out in direction **d**. α is a positive scalar stepsize. The so-called **Wolfe conditions** for an acceptable stepsize α are:

I) $\mathbf{F(x+\alpha d) \le F(x) + c_1 \alpha \langle d, \nabla F(x)\rangle}$ (sufficient decrease condition) **(8)**

II) $\mathbf{\langle \nabla F(x+\alpha d), d\rangle \ge c_2 \langle \nabla F(x), d\rangle}$ (curvature condition) **(9)**

where $\mathbf{0 < c_1 = .001 < c_2 = .9 < 1}$ (see [GL eqns (2.5) and (2.6)]).

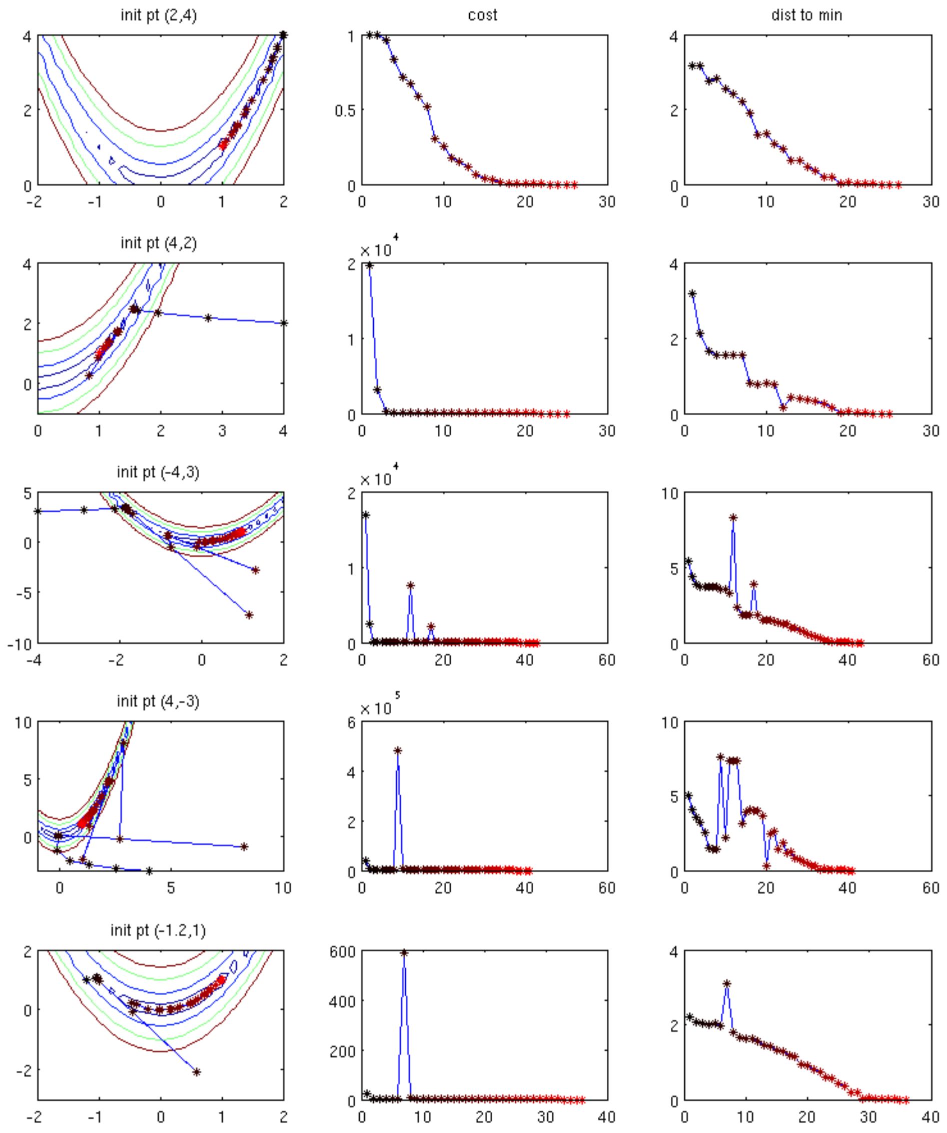
If the Wolfe conditions are satisfied for the stepsize α = **t**, then it is deemed a solution. We set **xx=xxc** and go to the next trial step iteration (Section 1.2.2). However, if either one of the Wolfe conditions are not satisfied, we reduce the stepsize by half (a safeguard), and rerun the forward and adjoint model (another iteration) and check the Wolfe conditions again until they are either satisfied or terminate the procedure after

NFUNC = 3 (defined in data.optim) tries, which then sets **xx**=latest **xxc** and deemed a solution.

INPUT	data.ctrl	Control variable information
	data.optim	Optimization parameters with latest costfunction value **fc**
	ecco_ctrl	Latest iteration control vector
	ecco_cost	Latest iteration cost gradient
	costfunction	Latest iteration costfunction values
	OPWARMI	Optimization parameters
	OPWARMD	Gradients from previous iterations
OUTPUT	OPWARMI	Updated optimization parameters
	OPWARMD	No changes
	ecco_ctrl	Control vector from safeguarded line search
	op_i#.64	Diagnostic file for optimization process

Section 1.2.4 Examples with a 2-D Rosenbrock Function

Tests done with the above methodology with the 2-dimensional Rosenbrock function (see http://en.wikipedia.org/wiki/Rosenbrock_function) verifies that it does converge superlinearly (http://en.wikipedia.org/wiki/Rate_of_convergence) to the required minimum. In the plot below, the first column shows the contours of the 2-d Rosenbrock function, and the trajectory of the solutions (connected dots) for different initial starting points, as it approaches the unique minimum (1,1). The second and third columns show the cost value of the Rosenbrock function and the Euclidean distance of each solution point to the minimum as functions of iterations.



Section 1.3 Checking for solutions

The shell script **checkop** outputs the iterations that are also solution steps. It does this by doing a grep for "1   1.0E+00" in the diagnostic files op_i#.64, since this line only exists when a solution step (section 1.2.3) completes.

For example:

```
% checkop
op_i43.64:   1   1.0E+00    0    1.7837E+08   5.7E+05   3.2E+02   6.3E+00
op_i44.64:   1   1.0E+00    0    1.7658E+08   7.4E+05   3.2E+02   6.0E+00
op_i46.64:   1   1.0E+00    0    1.7518E+08   4.9E+05   3.2E+02   8.1E+00
```

The first column shows that iterations 43,44, and 46 are solutions. The next 3 columns can be ignored. The 4th column is the costfunction value, 5th is the norm of the cost gradient, 6th is the norm of the solution control vector, 7th is the norm of difference between the solution control and the previous iteration control vector. The most important terms to look for are the 4th column which should be monotonically decreasing and the 5th column should be generally decreasing, but not necessarily monotonically.

Section 1.4 Other useful scripts

do_optim_recov: In some situations, such as a machine crash or incorrect restart, OPWARMI and OPWARMD may be erased or corrupted. This script recovers the latest OPWARMI and OPWARMD file starting from iter0's ecco_ctrl and ecco_cost vectors. It requires that all intermediate ecco_ctrl and ecco_cost vectors are preserved and available to the script. To use this script, just specify the optim directory, edit the end iteration number desired, and execute.

Section 1.5 Data.optim and Data.ctrl

The optimization executable requires two input files to be present. The following set of parameters are provided through the standard input file data.optim.

PARAMETER	VALUE
NUPDATE	Maximum number of update pairs (gg(i)-gg(i-1), xx(i)-xx(i-1)) to be stored in OPWARMD to estimate Hessian. Currently set to 4. NUPDATE must be > 0
EPSX	Relative precision on xx below which xx should not be improved (default 1e-6). NOT USED.

EPSG	Relative precision on gg below which optimization is considered successful (default 1e-6). NOT USED.
IPRINT	Controls verbose (>=1) or non-verbose output. Currently set to 10.
NUMITER	Always 1
ITER_NUM	Index of new restart file to be created (not necessarily = NUMITER). NOT USED.
NFUNC	Maximum number of safeguarded iterations allowed (must be > 0). Currently set to 3.
FC	Costfunction value of last iteration
FMIN	NOT USED

In the input file data.ctrl, the following parameters are needed:

PARAMETER	VALUE
CTRLNAME	ecco_ctrl (control vector prefix)
COSTNAME	ecco_cost (cost gradient vector prefix)

Section 1.6 OPWARMI and OPWARMD

The optimization outputs two files, a dynamic parameter file OPWARMI and a dynamic binary file OPWARMD. OPWARMI has the following structure:

PARAMETER	DESCRIPTION
n	Number of control variables per processor
fc	Cost value of last iteration
m	= NUPDATES in data.optim
jmin	Integer pointer for OPWARMD
jmax	Integer pointer for OPWARMD
gnorm	Norm of latest gradient **gg**
sflag	True if line search will be applied in the next iteration
tflag	True if next iteration will be a safeguarded one
safe_iter	Number of safeguarded iterations completed
stepsize	Value of the last iteration stepsize **t**

OPWARMD is a binary file and contains the following array structure:

RECORD	ARRAY	DESCRIPTION
1	xx(i)	Control vector of last iteration i
2	gg(i)	Gradient of last iteration i
3	xdiff(i)	Diagonal preconditioner (see GL eqn 4.9)
2*mod(jmax-1,NUPDATE)+4	gg(i)–gg(i-1)	Gradient difference for last iteration i
2*mod(jmax-1,NUPDATE)+5	xx(i)-xx(i-1)	Control difference for last iteration i

xdiff(i) = $\mathbf{B}_{\mathbf{jmin}}$ is the diagonal matrix defined in section1.2.2.

Jmax is continuously incremented with each iteration, with

$$\text{jmax}=\text{mod}(\text{i-1},\text{NUPDATE})+1,$$

$$\text{jmin}=\begin{cases}1 \text{ for the first NUPDATE values of jmax,}\\ \text{mod(jmax,NUPDATE)}+1 \text{ subsequently.}\end{cases}$$

Section 1.7 Capping

Capping of the cost gradient **gg** is applied in optim_readdata.F. A cap can be set individually for each control variable by changing the corresponding element of the array cvarlimit. Currently, all elements of cvarlimit are set to a very large number (1e4), effectively applying no capping.

Section 1.8 Compilation

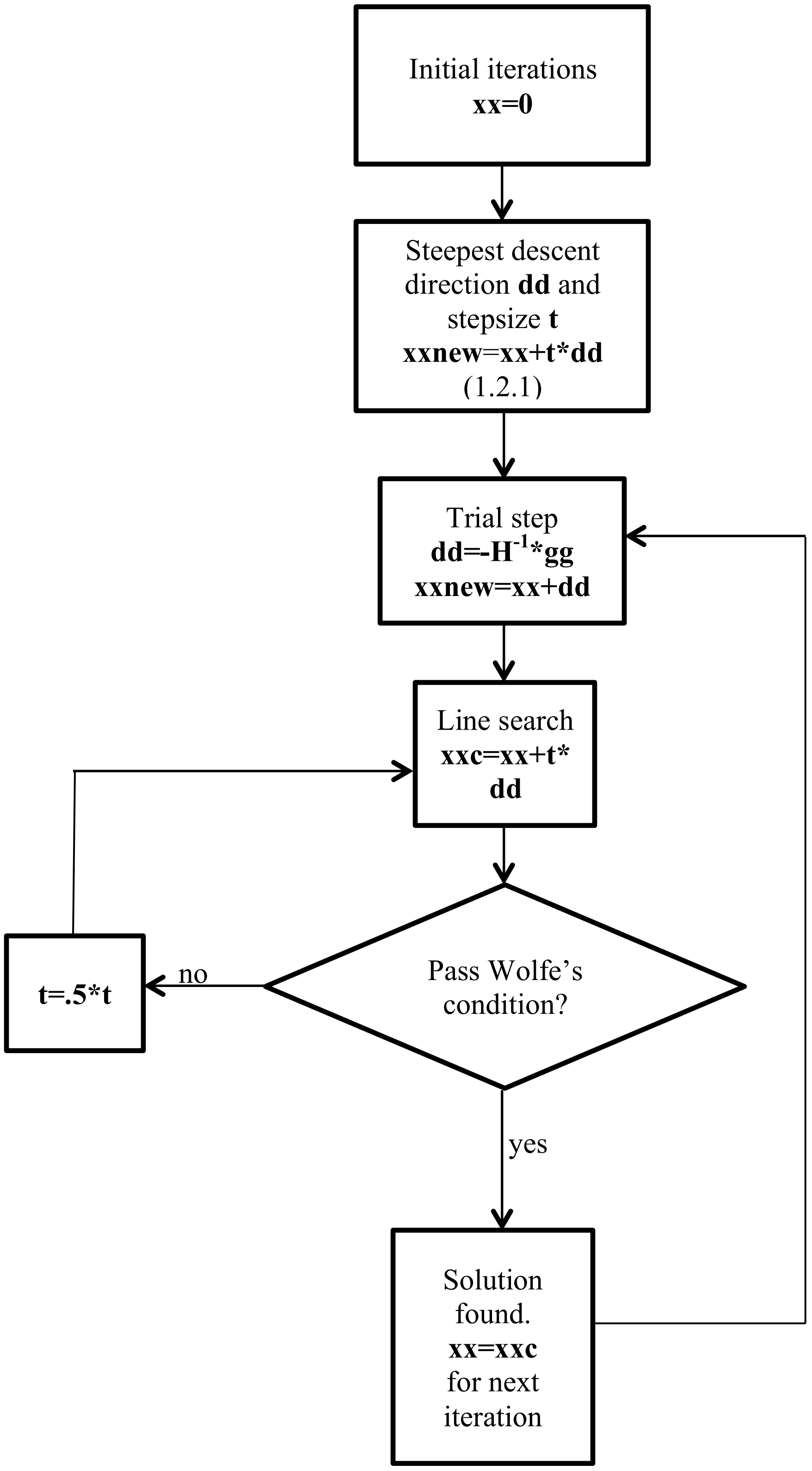
The optimization package consist of two directories, **optim.2** and **lsopt.2**. The following are the instructions for compiling this package:

1. cd to the lsopt.2 directory and edit the Makefile by changing the macro -DMAX_INDEPEND to the total number of control variables in the cost function (for MITgcm, this number can be found in the STDOUT output where it is defined as nvarlength), and customize the compiler flags to your machine (currently using ifort compiler).
2. Type make and this should produce the **liblsopt_ecco.a** library.
3. cd to the optim.2 directory and edit the Makefile by changing the macro -DMAX_INDEPEND and compiler flags as in 1) above, as well as the INCLUDESDIR to point to the MITgcm build directory. The optimization code needs a few header files that can usually be found in the build directory. Also, the directory ../lsopt.2 should be included in LIBDIRS (-L../lsopt.2.benny/) and the **liblsopt_ecco.a** library in LIBS (-llsopt_ecco).
4. Finally, type make and this should generate the **optim.x** executable.

Section 1.9 Run

Create a run directory and cd to this run directory. Copy the executable **optim.x** over to the run directory. Generate the namelists data.optim and data.ctrl as explained in **Section 1.5**. Also copy ecco_cost_MIT_CE_000.opt0000 that contains the initial cost gradient **gg**.

Section 1.10 Flowchart


Initial iterations
xx=0
Steepest descent direction **dd** and stepsize **t**
xxnew=xx+t*dd
(1.2.1)
Trial step
dd=-H^{-1}*gg
xxnew=xx+dd
Line search
xxc=xx+t* dd
Pass Wolfe's condition?
no
t=.5*t
yes
Solution found.
xx=xxc
for next iteration