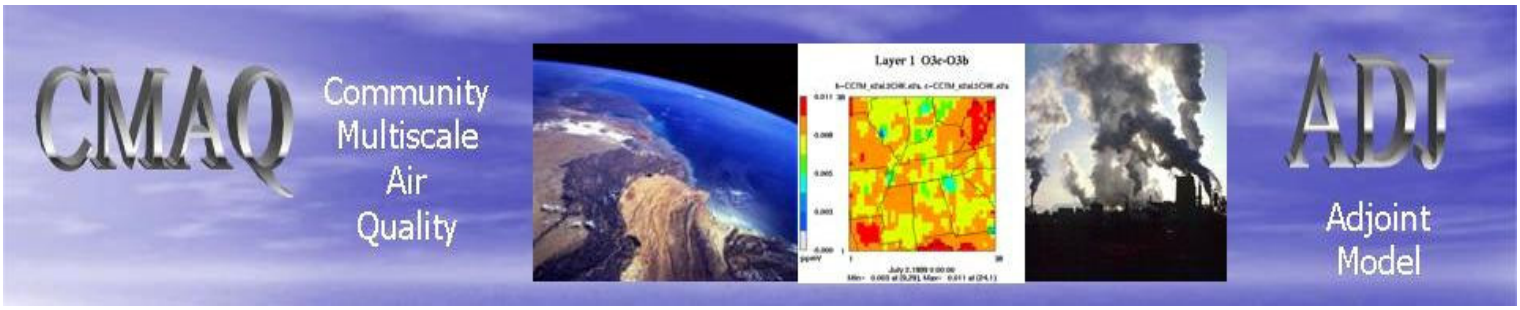


CMAQ v4.5 Adjoint User's Manual

An adjoint model for CMAQ to perform 4D-Var Data Assimilation and Sensitivity Analysis

Kumaresh Singh & Adrian Sandu

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24060, USA
kumaresh@cs.vt.edu
asandu7@cs.vt.edu

Amir Hakami & John Seinfeld

Department of Chemical Engineering
California Institute of Technology, CA

Date: 07/05/2007

Contents

CMAQ_ADJv4.5

1. Installation

The CMAQ_ADJv4.5 adjoint package comes in the same format as CMAQv4.5. To install the adjoint package,

(i) Download the CMAQv4.5 package from CMAS center's website: http://cmascenter.org/help/documentation.cfm?MODEL=cmaq&VERSION=4.5.

(ii) Replace the tar files M3MODELS.CMAQv4.5.tar and M3SCRIPTS.CMAQv4.5.tar of CMAQv4.5 package with those of the adjoint package.

(iii) Follow the $M3HOME/docs/README file for instructions on installing the package. Instructions for subdirectories icon, bcon, jproc, and others in $M3HOME/scripts/ directory remain the same, except for the cctm directory where bldit.cctm.pgf and run.cctm scripts should be executed based on the mode required.

(iv) Follow section 2 on installing and running various modes of adjoint operations provided with this CMAQ_ADJ4.5 package.

The visible changes in the adjoint package as compared to the basic CMAQ package are:

- The addition of an adjoint directory in $M3HOME/models/CCTM/src/ which contains all the adjoint files. These files are extracted by setting some flags in the bldit.cctm.pgf script in $M3HOME/scripts/cctm directory, as discussed in Section 2. A brief description about each of these adjoint files is provided in Section 6.

- Makefile provided explicitly in the $M3HOME/scripts/cctm directory to produce executables, to be used by the run.cctm script based on user's mode selection.

2. Running CMAQ Adjoint model

There are several adjoint run modes provided to the users to select from, based on their needs. First of all, in "$M3HOME/scripts/cctm/bldit.cctm.pgf" file one can choose not to extract the adjoint files if they are planning to run only the forward mode of CMAQ similar to the basic package. To do so, make sure the *ADJOINT MODE* (lines 372-373) is commented out. However, for sensitivity analysis, 4D-Var data-assimilation and finite-difference test modes, adjoint files are required. Executing the bldit.cctm.pgf script creates the BLD_e2a and MOD_DIR directories.

To select various adjoint run modes, follow the instructions provided as follows:

2.1 Mode Selection

File to be edited: $M3HOME/scripts/cctm/run.cctm

SIMULATION MODES (lines 190-212) in this run.cctm script provide various adjoint run modes to choose from. Following are the setting details for each individual mode:

2.1.1 Forward mode

Uncomment *BUILD FORWARD* mode (lines 194-195) and comment out rest of the modes. This will create the fwd executable which is for the forward mode of simulations, similar to the basic CMAQ model run.

2.1.2 Sensitivity Test mode

Uncomment *BUILD SENSITIVITY* mode (lines 199-200) and comment out the rest of the modes. This will create the snst executable which is to perform sensitivity analysis.

2.1.3 4D-Var Data Assimilation mode

Uncomment *BUILD 4D-VAR* mode (lines 204-205) and comment out the rest of the modes. This will create the 4dv executable which is to perform 4D-Var data assimilation experiment.

2.1.4 Finite Difference Test mode

Uncomment *BUILD FINITE-DIFF* mode (lines 209-210) and comment out rest of the modes. This will create the fd executable which is to perform finite difference test for adjoint code validation.

2.2 Data Directories and Files

The important output directories and useful data files (including new adjoint files) are listed as follows:

$M3HOME/data/cctm/

```
------------------------------
CCTM_e2aCONC.e2a  | - concentration file for basic CMAQv4.5 run.
------------------------------
------------------------------
CCTM_e2aCHK.e2a   |
CCTM_e2aL3CHK.e2a | - checkpoint files for sensitivity test and data assimilation.
CCTM_e2aL5CHK.e2a |
CCTM_e2aL6CHK.e2a |
------------------------------
```

$M3HOME/scripts/cctm/

----

cost_fun.m, rms_value.m| - cost function values, rms values and model runs are saved to
model_runs.m | these files at each iteration during the 4D-Var data assimilation.

2.3 Description / Instructions about current settings

The checkpoint files listed above are used to save different datasets in different adjoint run modes. Thus to maintain consistency, all the output data files and executables are deleted each time the run script is executed. It is advisable to save the checkpoint files elsewhere for future use. However, if you are running the same mode and would like to keep the data files, then comment out "*make dataclean*" (line 188, run.cctm file). Also, if you are making changes in the files for the same adjoint run mode and do not want to recompile everything, then comment out "*make clean*" (line 186, run.cctm file). It is recommended to clear the data files and executables during transitions from one mode to the other. Individual setting details for each mode will be provided in the following sections.

3. Code Validation – Finite Difference (FD) Test

To obtain consistent data assimilation and sensitivity results, it is necessary to validate all the adjoint science process subroutines. The validation test is designed to compare the adjoint variable with its finite-difference approximation. A detailed description of the test setup is provided as follows.

The CTM model is first run with original concentration values (say c_0) and an observation grid is extracted. The original concentration field (c_0) is then perturbed at the initial time and a run with the new concentration (say c_{01}) in forward and reverse mode is carried out. A similar run with different perturbation on the original concentration field is carried out next. Let c_{02} be the perturbed concentration at initial time for the second run.

In reverse mode of each run, the adjoint variable λ is initialized with zero concentration values at the final time and is updated with the gradient (g) of the cost function (f) at each dynamic time step during the adjoint calculations.

$$f = (1/2)\sum(c_{op}^{k,m} - c_{obs}^{k,m})^T R_k^{-1}(c_{op}^{k,m} - c_{obs}^{k,m})$$

$$g = \sum R_k^{-1}(c_{op}^{k,m} - c_{obs}^{k,m})$$

$$\lambda = \lambda + g_k$$

where, R_k is the misfit covariance matrix, k=1, 2,…,istep*nsteps, is the total number of science process iterations in the forward mode and m is a 4-tuple observation grid.

Based on this setup, adjoint model for each CMAQ science process is validated separately. The validation procedure is based on the fact that the value of adjoint variable (≡ g) should be close enough to its finite difference approximation calculated with the objective cost functions for the two runs with concentrations c_{01} and c_{02}, i.e.,

$$f(c_{02}) - f(c_{01}) = \lambda_2^T.(c_{02} - c_{01})$$

3.1 Mode Selection

To test the adjoint of each science process individually, one can choose one of the modes discussed as follows:

Caution:

- The files extracted and kept in the BLD_e2a directory are read-only. Thus, to comment/uncomment the science processes one needs to *chmod a+w filenames*. The PING, CLDPROC, AERO and ADJADV processes should remain commented, since the adjoint for these subroutines are not available with this package.

- The results provided for the current test setup are produced with 24 hours of simulation. Hence, make sure that NSTEPS = 240000 in run.cctm script, line 42.

3.1.1 Chemistry

filenames:
sciproc_cadj.F – Uncomment *CHEM_ADJ* subroutine (line 178) and comment out rest of the processes (lines 188-220).

modsciproc.F – Uncomment *CHEM* subroutine (lines 334-336) and comment out rest of the processes (lines 252-323).

3.1.2 Advection

filenames:
sciproc_cadj.F – Uncomment *rhoj checkpoint reading*, *XADV_CAD*, *YADV_CAD*, *ZADV_CAD* subroutines (lines 193-218) and comment out the rest of the processes (lines 178,188,220).

modsciproc.F – Uncomment *couple*, *decouple*, *XADV*, *YADV*, *ZADV* subroutines (lines 258-300, 317-322) and comment out the rest of the science processes (lines 252-254, 311-313, 334-336).

3.1.3 Diffusion

filenames:

sciproc_cadj.F – Uncomment *HDIFF_ADJ*, *VIDFF_ADJ* subroutines (lines 188, 220) and comment out the rest of the science processes (lines 178, 193-218).

modsciproc.F – Uncomment *VDIFF*, *DIFF* subroutines (lines 252-254, 311-313) and comment out the rest of the processes (lines 258-300, 317-336).

3.1.4 All science processes

filenames:
sciproc_cadj.F - Uncomment all the science process adjoints, *VDIFF_ADJ*, *HADV_ADJ*, *ZADV_ADJ*, *HDIFF_ADJ*, *CHEM_ADJ*.

modsciproc.F – Uncomment all the science processes, *VDIFF*, *HADV*, *ZADV*, *HDIFF*, *CHEM*.

3.2 FD Validation Results

In the current code-validation setup, the test parameters have following values:

Variable	Value
Eps	0.001
T_f	24 hrs
C_{01}	CGRID*(1.1+9*eps)
C_{02}	CGRID*(1.1+10*eps)

Table 1: Test Parameter Values

The test parameters presented in Table 1 can be varied according to individuals need. However for the current setup, the validation results for different modes are as follows:

Process	$F(c_{02}) - F(c_{01})$	$\lambda_2^T.(c_{02} - c_{01})$	Relative Difference
Chemistry	587.15439	589.84952	0.456918 %
Advection	390.95876	390.36101	-0.153130 %
Diffusion	624.34778	630.00552	0.898046 %
All	340.66521	329.15051	-3.498310 %

Table 2: CMAQ Adjoint Validation Results

4. Sensitivity Test

Sensitivity analysis is an approach to quantify the changes in output due to different sources of variation. Since adjoint calculations are receptor based, one can calculate the sensitivity of an output with respect to large number of parameters.

In order to perform adjoint sensitivity tests on the validated CMAQ adjoint model, it is required to perform one forward and one reverse mode of simulation. In the forward run, the concentration values and air densities are checkpointed and in the reverse mode they are read to be utilized for adjoint calculations. A receptor (cost-function) measuring certain species (say **x**) is defined at a given location at the final time in the reverse mode. At the end of the simulation, we obtain sensitivities of x with respect to the grid species and emission species.

4.1 Settings

filenames:

sensitivity_driver.F – main driver file to initialize CGRID, open checkpoint files and call subdriver to perform forward and backward runs.

senstdriver_bwd.F – subdriver to perform one forward and one backward run with adjoint variable defined by calling subroutine DEFINE_RECEPTOR and emission adjoint variable initialized with zero concentration.

define_receptor.F – subroutine to initialize the adjoint variable LGRID.

Note:
Adjoint variable LGRID(1:ncols, 1:nrows, 1:nlays, 1:spc) has 4 dimensions. In order to change the species number for which sensitivity test has to be performed, change the spc index at the initialization step in define_receptor.F (line 50).

4.2 Test details and Analysis

filename:
CCTM_e2aL5CHK.e2a – checkpoint file for LGRID values for every dynamic time step.
CCTM_e2aL6CHK.e2a – checkpoint file for cumulative emission adjoint variable.

In order to calculate the sensitivity with respect to grid species and emissions, set the parameters according to your needs using the files described in section 4.1. Choose the snst mode in run.cctm for sensitivity calculations and perform the run. The adjoint variable for grid species initialized at the final time undergoes all the science process adjoints while the emission adjoint variable is accumulated with the gradient of emissions over time.

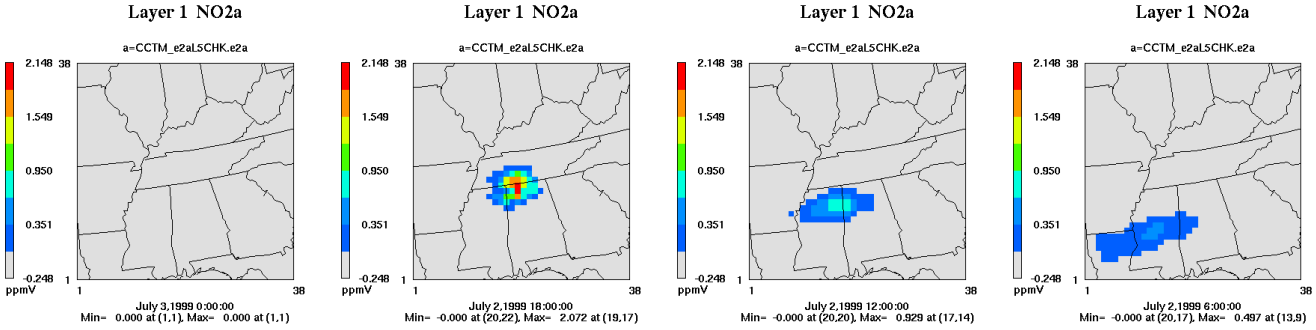
4.2.1 Sensitivity Results

In order to produce the adjoint trajectory for sensitivities with respect to grid species, PAVE the CCTM_e2aL5CHK.e2a checkpoint file. Then from the list of species in formula popup window, add the ones with respect to which sensitivities has to be performed. Then create the tile plot. Similar procedure is followed for sensitivities with respect to emissions. PAVE the CCTM_e2aL6CHK.e2a checkpoint file and select the emission species with respect to which the sensitivities have to be performed. A major difference between the two plots is the time length.

The emissions plot is for the final time step so it has no time series involved, while the grid species plot has NSTEPS*(dynamic time steps) time length.

Grid-species:

In the sample test case, we consider a receptor measuring ozone at the final time at a given location (defined in define_receptor.F). We compute its sensitivity with respect to changes in various grid species at earlier times. A tile plot for dO_3/dNO_2 is provided as follows:



T = 24hrs T = 18hrs T = 12hrs T = 6hrs

Fig 1: Sensitivity plots for dO_3/dNO_2 backwards in time every 6hrs.

In figures 1, the receptor is O_3 at location ncols=18:22, nrows=18:22 and nlays=1 at time T=24hrs.

Emissions:

Under the same settings, the sensitivity of ozone is calculated with respect to various emission species. Tile plots for dO_3/dNO_x emissions are provided as follows:

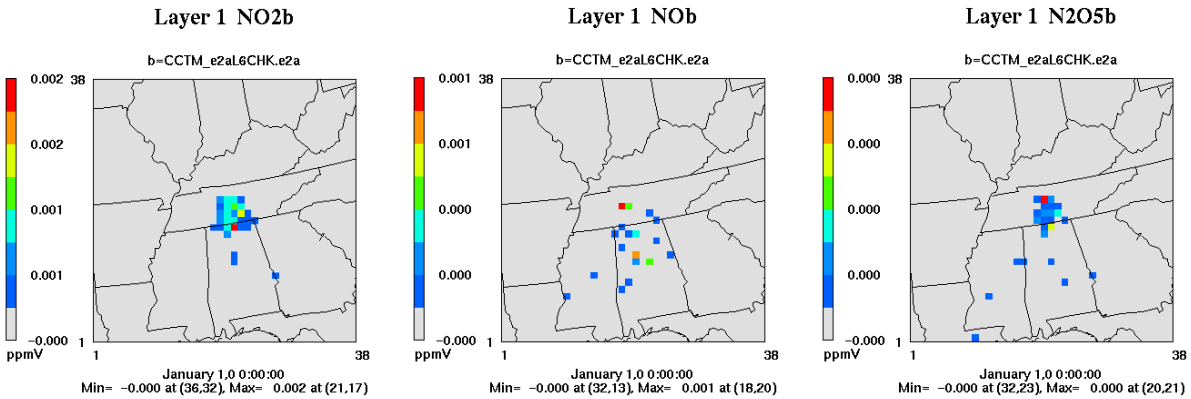


Fig 2: Sensitivity plots for dO_3/dNO_x emissions for 47 hours of simulation

In figures 2, the receptor is O_3 at location ncols=18:22, nrows=18:22 and nlays=1 at time T=47hrs.

4.3 More than one-day simulations:

To perform sensitivity test for N (> 1) days, a set of instructions are provided as follows:

- Set the fwd mode in run.cctm script and run the forward mode for N-1 days. The current setup will run the code for 24hours. However, for rest of the N-2 days we need to make the following changes:

  (i) Change the STDATE in run.cctm (line 39) for each day's simulation.
  (ii) Set the EMISfile to the emission file of that day. The current run script has two emission file names.
  (iii) Uncomment set GC_ICpath = $OUTDIR and set GC_ICfile = $EXEC"CONC".$APPL (lines 119,120) and comment out earlier settings with ICON (lines 121,122).

  Caution: Comment out the "*make dataclean*" mode (line 188, run.cctm file)

  Settings (i) and (ii) are done for each day, while (iii) remains the same for all N-1 days including the Nth day.

  This will create the CCTM_e2aCONC.e2a file for all the N-1 days which acts as the initial condition file for the Nth day simulation. The adjoint calculations are then done backwards for each day N-1, N-2, …, 2, 1.

- For the last day, set the snst mode in run.cctm script and follow steps (i),(ii) and (iii) described above. Make sure the flag "LASTDAY" in senstdriver_bwd.F (line 104) is set to TRUE. The simulation hours for the last day must be less than 24 hours, since interpolation for an extra hour is performed inside the model.

  Caution: Keep the "*make dataclean*" mode (line 188, run.cctm file) commented.

  For the rest of the previous days, follow steps (i),(ii) and (iii) and change the flag "LASTDAY" in senstdriver_bwd.F (line 104) to FALSE.

This will create the necessary checkpoint files for all the days required.

5. 4d-Var Data Assimilation

4D-Var data assimilation allows the optimal combination of three sources of information: an a priori (background) estimate of the state of the atmosphere, knowledge about the physical and chemical processes that govern the evolution of pollutant fields as captured in the chemistry transport model (CTM), and observations of some of the state variables.

In order to integrate 4D-Var data assimilation with the CMAQ adjoint, an interface has been developed to run the CTM model with the optimization routine L-BFGS. This routine is used to solve large scale nonlinear optimization problems with simple bounds. It is based on the gradient

projection method and uses limited memory BFGS matrix to approximate the Hessian of the objective function.[*R. H. Byrd, P. Lu and J. Nocedal.* A Limited Memory Algorithm for Bound Constrained Optimization, SIAM Journal on Scientific and Statistical Computing, 16, 5, pp. 1190-1208.]

The BFGS is a quasi-Newton algorithm which solves

$$min\ f(x), \text{ subject to } l \leq x \leq u.$$

where f(x) is an objective cost-function in x which we are trying to minimize. l and u are the lower and upper bounds on the values of x. The optimization routine takes as input, an initial value x, the cost-function value f and the gradient of the cost function g and gives the next best estimate of x. The same procedure is followed until the solution converges to a single point. The number of iterations can be restricted based on individuals need.

For data assimilation experiments, the cost-function is defined as follows:

$$f = (1/2)\sum(c_{opt}^{k,m} - c_{obs}^{k,m})^T R_k^{-1}(c_{opt}^{k,m} - c_{obs}^{k,m}) + (1/2)\sum(c_{opt} - c_b)^T B^{-1}(c_{opt} - c_b)$$

where, R_k is the misfit covariance matrix and B is the background covariance matrix, k=1, 2,…,istep*nsteps, is the total number of science process iterations in the forward mode and m is a 4-tuple observation grid.

The gradient of this cost-function is then calculated as follows:

$$g = \sum R_k^{-1}(c_{opt}^{k,m} - c_{obs}^{k,m}) + \sum B^{-1}(c_{opt} - c_b)$$

Here c_b, c_{obs} and c_{opt} are the background concentration, observations and the best estimate respectively. In case of real data inputs, c_b acts as apriori concentration and c_{obs} is the observation. c_{opt} acts as the current best estimate which is updated by the L-BFGS subroutine every iteration based on the cost-function and its gradient values.

Note:
The background forcing is added only at the initial time step, while the misfit is added at each dynamic time step. Also the misfit is calculated only on the observation grid points, while the background is calculated at all the grid points on a layer.

5.1 Experiment details and settings

In the sample test case, the data used is the one which comes with CMAQv4.5 package. A reference run of T_f hrs is being performed on this data in order to generate non-constant concentration field. The concentration field thus obtained acts as an initial condition (c_0^0) for our data assimilation test. An observation grid is extracted by performing a forward CMAQ run on the current concentration field (c_0^0).

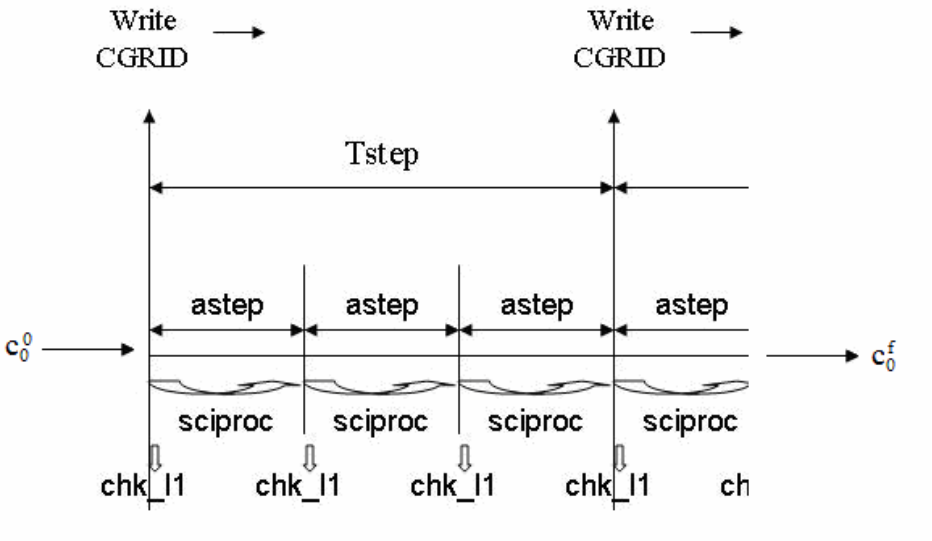


Fig 3: Checkpointing procedure during forward run

A perturbation is then introduced in c_0^0 to produce c_p^0.

$$c_p^0 = c_0^0 + \text{perturbation}$$

This perturbed concentration is then transferred to the optimization subroutine in order to obtain the best estimate c_{opt}^0 of the original concentration c_0^0 after several data-assimilation runs.

At iteration 0, $x_0 = c_p^0$

At each subsequent iteration k ($k \geq 1$),

$$x_{k+1} \leftarrow \text{L-BFGS}\ (x_k, f, g)$$
$$c_{opt}^0 \leftarrow x_{k+1}$$
$$(f, g) \leftarrow \text{reverse\_mode}\ (c_{opt}^0, \text{chk\_obs})$$

where, f is the cost function and g is the gradient of this cost function.

Details about the subroutines involved and the corresponding files where they are defined are as follows:

filenames:
4DVar_driver.F – main driver file to implement 4D-Var data assimilation. Calls sub drivers to lay down observations, checkpoint perturbed data, and perform adjoint calculation to get cost function value and its gradient for the optimization subroutine.

subdriver_fwd.F – holds subroutine to perform forward science process runs. This subdriver is called twice in the main driver. First call produces a non-constant concentration field, required since CMAQv4.5 data is not real data. Second call lays down the observation field.

subdriver_fwdpert.F – contains subroutine to perform forward science process similar to the above subdriver, but lays down checkpoints for perturbed data. This checkpoint is used in the background cost-function calculations.

subdriver_bwd.F – contains subroutine to perform one forward and one reverse run of science processes to calculate the cost-function and its gradient to be used by the L-BFGS subroutine.

calc_obsgrad.F – calculates the cost-function due to misfit and updates the adjoint variables.

calc_bggrad.F – calculates the cost-function due to background errors and updates the adjoint variables.

mask.dat – observation grid index file located in directory: $M3HOME/scripts/run, to be used by OBS_GRAD_UPDATE subroutine (calc_obsgrad.F). In the current test case observation grid, the columns and rows are linearly spaced among 10 points from x1 to x2 using Matlab's linspace() function. The vertical layer slices are double spaced.

Caution:

- In our experiment we are perturbing ozone and retrieving **ozone only** (spc=4). To perturb other species, change the spc index from 4 to the species number of interest in 4DVar_driver.F file(line 301, 310). To retrieve other species, change NSPCOPTSTART and NSPCOPTEND variable values in the same file: 4DVar_driver.F (line 325,326).

- The interface for cost function calculations in files: calc_obsgrad.F and calc_bggrad.F are designed for model data input. One should change the setup for real data intake.

In the current experimental setup, the parameter values are set as follows:

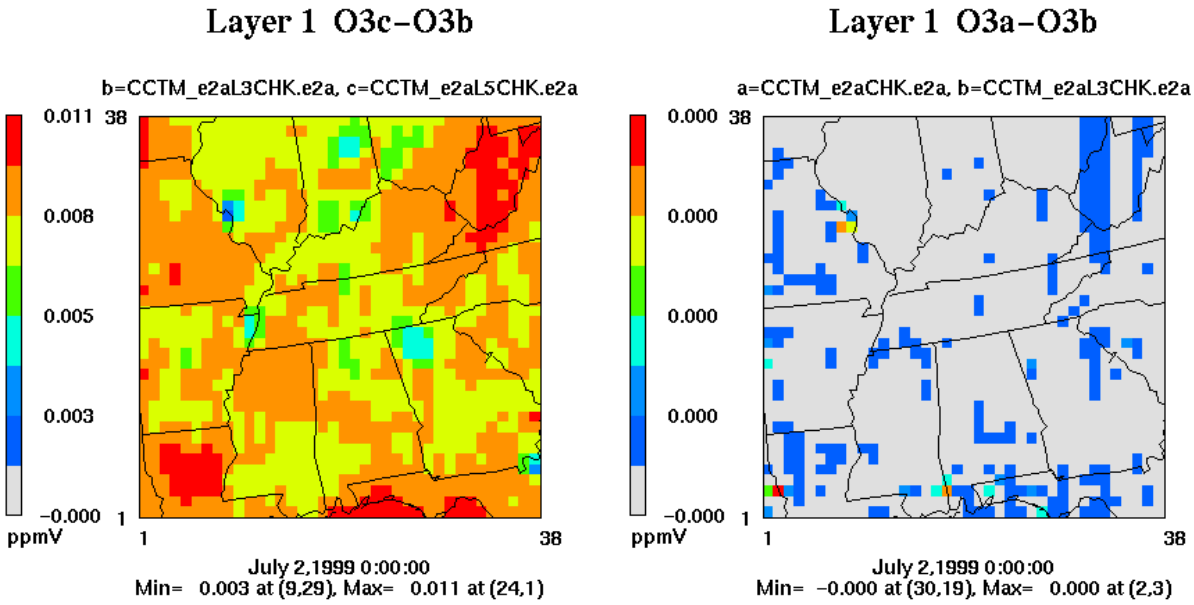
Observation Grid				**Perturbation amount**
Columns	Rows	Layers	Species	$c_p^0 = c_0^0*(1.3+9*eps)$, eps = 0.001
Linspace(x1,x2,10) x1=1, x2 = ncols	Linspace(x1,x2,10) X1=1, x2 = nrows	1:2:NLAYS	4	

Table 3: Data Assimilation experiment parameter values

5.2 4D-Var Data Assimilation Results

filenames:
CCTM_e2aCHK.e2a – optimized concentration checkpoint file
CCTM_e2aL3CHK.e2a – observation checkpoint file
CCTM_e2aL5CHK.e2a – perturbed concentration checkpoint file

To validate the developed test-bed, two sets of plots are being generated; one with the difference c_p^0- c_0^0 (CCTM_e2aL5CHK.e2a - CCTM_e2aL3CHK.e2a) and the second with c_{opt}^0- c_0^0 (CCTM_e2aCHK.e2a - CCTM_e2aL3CHK.e2a). The idea is to illustrate the fact that data assimilated concentration fields are the best estimates of the original concentration fields. Figure 3 reflects that c_{opt}^0 is quite close to c_0^0.



a) Difference between perturbed and reference concentration (c_p^0- c_0^0)

b) Difference between optimized and reference concentration (c_{opt}^0- c_0^0)

Fig 4: Demonstration of recovery of the original concentration field.

A cost-function and root-mean square vs. model runs plot, Figure 4, is also provided for further validations.

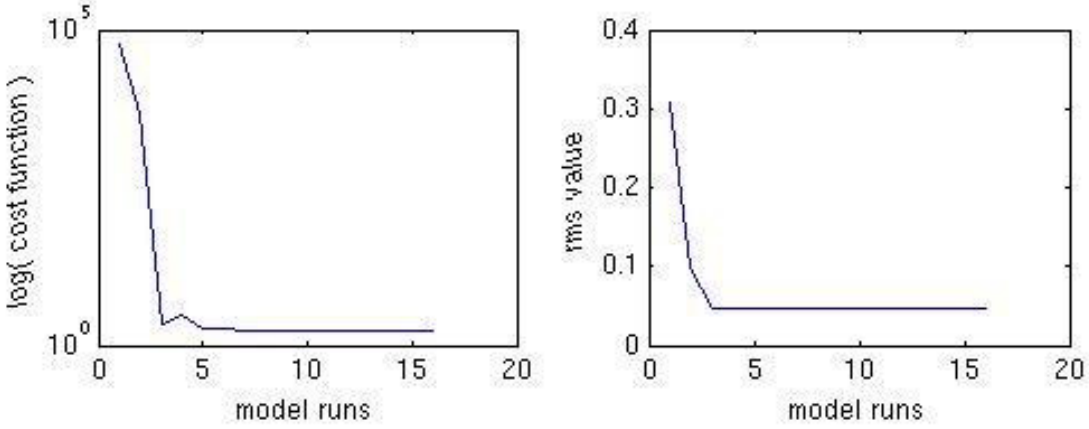


Fig 5: Cost-function and root-mean square values vs. model runs.

As explained in the last section, Figure 3 shows a good recovery of the original concentration field from the perturbed field. Figure 4 on the other hand reflects that there is a significant cost function decrease with model runs, which means that the solution converges to the original concentration field at the observation grid points. However, to illustrate an overall convergence of the solution, we calculate the root-mean square (RMS) value given by:

$$RMS = \| c_{op}^{0} - c_{0}^{0} \| / \| c_{0}^{0} \|$$

The decrease in this RMS value with model runs signifies that there is an overall convergence of the solution.

Caution:

The data assimilation results presented are generated with T_f = 6hrs and 16 iterations.

- T_f = 6hrs → set in $M3HOME/scripts/cctm/run.cctm script, NSTEPS = 060000, line 42.
- Number of iterations = 16 → set in $M3HOME/scripts/cctm/BLD_e2a/4DVar_driver.F, N_CMAQ_RUNS to be changed to desired value, line 477.

6. New Subroutines in CMAQ Adjoint

All the adjoint files are CVS archived under $M3HOME/models/CCTM/src/ and when extracted they are placed under $M3HOME/scripts/cctm/

Description:

6.1 Drivers

Main drivers to perform 4D-Var data assimilation, sensitivity analysis and finite difference test.

- **4DVar_driver.F (** `Subroutine: DRIVER` **)**

```
Function:
     -> CTM driver for 4D-Var DATA ASSIMILATION
     -> Uses L-BFGS optimization routine
     -> Exponential Preconditioning option available with this package.
        To invoke it, change EXP_PRECOND to 'TRUE', line = 127

CAUTION:
     -> Driver designed to perturb and retrieve ozone only
     -> To perturb other species change SPC index of CGRID in
        the Initialization, line = 297 & 302
     -> To retrieve accordingly, change NSPCOPTSTART & NSPCOPTEND
        variables, line = 318

USEFUL FILES:
```

```
     CONC_L3CHK = Reference/Base checkpoint file
     CONC_L5CHK = Initial Guess checkpoint file
     CONC_CHK   = Next Best Guess concentration

Subroutines and functions called:
     SUBDRIVER_FWD     - Forward Run for Reference checkpointing
     SUBDRIVER_FWDPERT - Forward Run for Best Guess checkpointing
     SUBDRIVER_BWD     - Forward and Backward Run to calculate cost function
                         & its gradient at each optimization iteration
```

- **sensitivity_driver.F (** `Subroutine: DRIVER` **)**

```
Function:
     -> CTM driver for SENSITIVITY ANALYSIS
     -> SENSITIVITY TEST: Run this driver to generate the adjoint
        trajectory for species under consideration. To change the
        specie number or the area under consideration, go to
        "define_receptor.F" file.

CAUTION:
     -> Driver designed to perform Sensitivity test with ozone

CHECKPOINT FILES:
     CONC_CHK   = Current concentration

Subroutines and functions called:
     INITSCEN, ADVSTEP, M3EXIT, WRITE3
     STDRIVER_BWD -> performs adjoint model run to create the trajectory
```

- **fd_driver.F (** `Subroutine: DRIVER` **)**

```
Function:
     -> CTM driver for FINITE DIFFERENCE TESTS
     -> FD TEST: Introduce different perturbations and run the adjoint
                 model twice to calculate respective cost funtions f1
                 and f2 and validate the model with the following eqn:
                 "Costfunc2-Costfunc1 = LGRID2'.(CGRID2-CGRID1);"

CAUTION:
     -> Driver designed to perform FD test with ozone
     -> To perturb other species change SPC index of CGRID in
        Finite-Difference Initialization, line = 327

CHECKPOINT FILES:
     ->Relevant output Files
       CONC_L3CHK = Reference/Base checkpoint file
       CONC_CHK   = Current CGRID

     ->Irrelevant intermediate checkpoint files
       CONC_L2CHK = CGRID before starting chemistry
       CONC_L4CHK = Checkpoint air density in forward science process

Subroutines and functions called:
     SUBDRIVER_FWD -> lay down observation grid
     FDDRIVER_BWD  -> calculate cost function
```

```
     RD_CHK        -> read checkpoint file
```

6.2 Subdrivers

Called from the main drivers, these subdrivers are responsible for forward and reverse science process runs and checkpoint CGRID and LGRID concentrations.

- **subdriver_fwd.F (** `Subroutine: SUBDRIVER_FWD` **)**

```
Function:
     Subroutine to perform forward run and lay down the observation grid.

CHECKPOINT FILES:
     CONC_L3CHK = Reference/Base checkpoint file

INPUT:
     CGRID

Subroutines and functions called:
     science processes -> SCIPROC
     write checkpoint files -> WR_L3CHK
```

- **subdriver_fwdpert.F (** `Subroutine: SUBDRIVER_FWD` **)**

```
Function:
     Subroutine to perform forward run to lay down the best initial
     guess/background.

CHECKPOINT FILES:
     CONC_L5CHK = initial guess/background conc checkpoint file

INPUT:
     CGRID

Subroutines and functions called:
     science processes -> SCIPROC
     write checkpoint files -> WR_L5CHK
```

- **subdriver_bwd.F (** `Subroutine: SUBDRIVER_BWD` **)**

```
Function:
     Subroutine to perform one forward and one backward run to calculate the
     observation and background parts of cost function and update LGRID.

CHECKPOINT FILES:
     CONC_L3CHK = Reference/Base checkpoint file
     CONC_CHK   = Current concentration CGRID
     CONC_L5CHK = Initial conc checkpoint file for First perturbation

INPUT:
     CGRID

OUTPUT:
     f,LGRID
```

```
Subroutines and functions called:
     science processes -> SCIPROC, SCIPROC_CADJ
     OBS_GRAD_UPDATE,BG_GRAD_UPDATE -> calculate cost function & update LGRID
     WR_CHK,RD_CHK,RD_L5CHK -> Read and write checkpoint files
```

- **fddriver_bwd.F (** `Subroutine: FDDRIVER_BWD` **)**

```
Function:
     Subroutine to perform one forward and one backward run to calculate the
     cost function and update LGRID for observation misfit only.

CHECKPOINT FILES:
     CONC_L3CHK = Reference/Base checkpoint file
     CONC_CHK   = Current concentration

INPUT:
     CGRID

OUTPUT:
     f,LGRID

Subroutines and functions called:
     SCIPROC, SCIPROC_CADJ -> science processes and their adjoint
     OBS_GRAD_UPDATE       -> calculate cost function & update LGRID
     WR_CHK,RD_CHK         -> Read and write checkpoint files
```

- **senstdriver_bwd.F (** `Subroutine: STDRIVER_BWD` **)**

```
Function:
     Subroutine to perform one forward and one backward run to generate
     adjoint trajectory.

CHECKPOINT FILES:
     CONC_CHK   = CGRID concentration
     CONC_L5CHK = LGRID values
     CONC_L6CHK = EMGRID (emission grid) values

Subroutines and functions called:
     science processes  -> SCIPROC, SCIPROC_CADJ
     Initializing LGRID -> DEFINE_RECEPTOR
     WR_CHK,RD_CHK,WR_L5CHK -> Read and write checkpoint files
```

6.3 Transport Processes

Subroutines involved in reverse transportation of the adjoint concentrations – horizontal diffusion, vertical diffusion, horizontal advection and vertical advection.

- **hdiff_adj.F (** `Subroutine: HDIFF_ADJ` **)**

```
Function:
     Subroutine to perform discrete horizontal diffusion adjoint
     calculations
```

```
INPUT:
     LGRID

OUTPUT:
     LGRID

Subroutines and functions called:
     RHO_J,HCDIFF3D
```

- **vdiffim_adj.F (** `Subroutine: VDIFF_ADJ` **)**

```
Function:
   -> Discrete adjoint of VDIFF subroutine that comes with CMAQv4.5
   -> calculates discrete vertical diffusion adjoint
      controlled by flag THETA, using Crank-Nicolson difference scheme
      THETA : Crank-Nicolson index  [ 1, fully implicit | 0, fully explicit ]

Associated tri-diagonal system is stored in 3 arrays
   DI: diagonal
   LI: sub-diagonal
   UI: super-diagonal
   BI: right hand side function
   XI: return solution from tridiagonal solver

     [ DI(1) LI(2) 0     0     0 ...       0     ]
     [ UI(1) DI(2) LI(3) 0     0 ...       .     ]
     [ 0     UI(2) DI(3) LI(4) 0 ...       .     ]
     [ .           .     .     .           .     ] XI(i) = BI(i)
     [ .                 .     .     .     0     ]
     [ .                       .     .     .     ]
     [ 0                           UI(n-1) DI(n) ]

     where n = NLAYS
```

- **modvdiffim_adj.F (** `Subroutine: VDIFF_ADJ` **)**

```
Function:
   -> Discrete adjoint of VDIFF subroutine that comes with CMAQv4.5
   -> calculates discrete vertical diffusion adjoint
      controlled by flag THETA, using Crank-Nicolson difference scheme
      THETA : Crank-Nicolson index  [ 1, fully implicit | 0, fully explicit ]
   -> designed specially for sensitivity tests

Associated tri-diagonal system is stored in 3 arrays
   DI: diagonal
   LI: sub-diagonal
   UI: super-diagonal
   BI: right hand side function
   XI: return solution from tridiagonal solver

     [ DI(1) LI(2) 0     0     0 ...       0     ]
     [ UI(1) DI(2) LI(3) 0     0 ...       .     ]
     [ 0     UI(2) DI(3) LI(4) 0 ...       .     ]
     [ .           .     .     .           .     ] XI(i) = BI(i)
     [ .                 .     .     .     0     ]
```

```
     [ .                          .     .     .     ]
     [ 0                              UI(n-1) DI(n) ]

     where n = NLAYS

INPUT:
     LGRID, EMGRID

OUTPUT:
     LGRID, EMGRID
```

- **xadvppm_cad.F (** `Subroutine: XADV_CAD` **)**

```
Function:
     Advection CONTINUOUS ADJOINT in the horizontal plane; x1-direction:
     The process time step is set equal to TSTEP(2). Boundary concentrations
     are coupled in RDBCON with SqRDMT = Sq. Root [det ( metric tensor )]
     = Jacobian / (map scale factor)**2
     where Air Density X SqRDMT is loaded into last BCON slot for advection.

INPUT:
     CGRID

OUTPUT:
     CGRID
```

- **yadvppm_cad.F (** `Subroutine: YADV_CAD` **)**

```
Function:
     Advection CONTINUOUS ADJOINT in the horizontal plane; x2-direction:
     The process time step is set equal to TSTEP(2). Boundary concentrations
     are coupled in RDBCON with SqRDMT = Sq. Root [det ( metric tensor )]
     = Jacobian / (map scale factor)**2
     where Air Density X SqRDMT is loaded into last BCON slot for advection.

INPUT:
     CGRID

OUTPUT:
     CGRID
```

- **zadvppm_cad.F (** `Subroutine: ZADV_CAD` **)**

```
Function:
   Advection CONTINUOUS ADJOINT in the vertical, x3-direction:
   The process time step is set equal to TSTEP

INPUT:
     CGRID

OUTPUT:
     CGRID
```

6.4 Chemistry

Subroutines responsible for forward and reverse chemistry processes. These files are generated using Kinetic PreProcessor (KPP). (courtesy: Amir Hakami)

- **kppdriver.F** – main driver to perform forward chemistry.
- **kppdriver_adj.F** – main driver to perform backward chemistry process on adjoint variable.
- **kppcalcks.F** - computes thermal and photolytic reaction rate coefficients for each reaction.
- **KPP_Data_mod.F** - mechanism & solver data for EBI solver.
- **KPP_Init.F**- to initialize species tolerances, arrays, and indices.
- **KPP_Util.F** – provides function to copy concentrations from USER to KPP and vice-versa.
- **KPP_HessianSP.F** - Hessian Sparse Data.
- **KPP_JacobianSP.F** - Sparse Jacobian Data.
- **KPP_JacobianTE.F**- provides subroutines: Jac_SP - the Jacobian of Variables in sparse matrix representation, and JacTR_SP_Vec - sparse multiplication: sparse Jacobian transposed times vector.
- **KPP_Function.F** - time derivatives of variables - Agregate form.
- **KPP_Glob.F** - declaration of global variables.
- **KPP_Pars.F** – model parameter definitions.
- **KPP_Precision.F -** Definition of different levels of accuracy for REAL variables using KIND parameterization.
- **kppModel.F** - Completely defines the model CMAQ_CB4 by using all the associated modules.
- **KPP_HessianTE.F** – provides subroutines: Hessian - function for Hessian (Jac derivative w.r.t. variables) and Hess_Vec - Hessian times user vectors.
- **KPP_LinAlg.F** – Sparse Linear Algebra subroutines.
- **kppIntegrator.F** - Numerical Integrator (Time-Stepping) File.
- **kppIntegrator_adj.F** - Numerical Integrator Adjoint (Time-Stepping) File.

6.5 Checkpoint Files

Subroutines used for reading and writing forward (CGRID) and adjoint (LGRID) concentrations.

- **rd_*chk.F * = ‘ ’, ‘L3’, ‘L4’, ‘L5’, ‘L6’\ (** `Subroutine: RD_*CHK` **)**

```
Function:
     Subroutine to perform reading from checkpoint file CONC_*CHK

INPUT:
     CGRID
```

- **wr_*chk.F * = ‘ ’, ‘L3’, ‘L4’, ‘L5’, ‘L6’\ (** `Subroutine: WR_*CHK` **)**

```
Function:
     Subroutine to perform writing to the checkpoint file CONC_*CHK

INPUT:
```

```
     CGRID
```

6.6 Others

- **modsciproc.F (** `Subroutine: SCIPROC` **)**

```
Function:
    Controls all of the physical and chemical processes for a grid
    Operator splitting symmetric around chemistry

CAUTION:
    This is a modified SCIPROC subroutine with some of the physical processes
    such as ADJADV,PING,CLDPROC and AERO switched off. For the adjoint model
    only chemistry and transport processes are considered with this package.

INPUT:
     CGRID

OUTPUT:
     CGRID

Subroutines and functions called:
    All physical and chemical subroutines,
    DECOUPLE, COUPLE, VDIFF, XADV, YADV, ZADV, HDIFF
```

- **sciproc_cadj.F (** `Subroutine: SCIPROC_CADJ` **)**

```
Function:
     Controls all of the physical and chemical adjoint processes for a grid
     Operator splitting symmetric around chemistry

INPUT:
     LGRID, CGRID

OUTPUT:
     LGRID

Subroutines and functions called:
     All physical and chemical subroutines,
     VDIFF_ADJ, XADV_CAD, YADV_CAD, ZADV_CAD, HDIFF_ADJ
```

- **modsciproc_cadj.F (** `Subroutine: SCIPROC_CADJ` **)**

```
Function:
     ->Controls all of the physical and chemical adjoint processes for a grid
     ->Operator splitting symmetric around chemistry
     ->Designed for use with sensitivity analysis

INPUT:
     LGRID, EMGRID

OUTPUT:
     LGRID
```

```
Subroutines and functions called:
     All physical and chemical subroutines,
     VDIFF_ADJ, XADV_CAD, YADV_CAD, ZADV_CAD, HDIFF_ADJ
```

- **calc_bggrad.F (** Subroutine: BG_GRAD_UPDATE **)**

```
Function:
     Calculates the background cost-function value and updates
     adjoint variable for initial time


CAUTION:
     ->This subroutine is being constructed for current test problem.
     ->One needs to modify the formulas according to his/her needs.

INPUT:
     CGRID - Current concentration field (read from CONC_CHK)
     LGRID - Adjoint variable to be updated
     CF    - Cost-Function update variable
     JDATE,JTIME - Current date and time step values

DATA READ FROM FILES:
     CBGRID - background (perturbed) conc from file: CONC_L5CHK

OUTPUT:
     LGRID,CF
```

- **calc_obsgrad.F (** Subroutine: OBS_GRAD_UPDATE **)**

```
Function:
     Calculates the observation cost-function update value and updates
     adjoint variable for the current dynamic time-step

CAUTION:
     This subroutine uses model-observations to calculate cost function
     and its gradient. One needs to modify the formulaes and I/O procedures
     to incorporate real-observations.

INPUT:
     CGRID - Current concentration field
     LGRID - Adjoint variable to be updated
     CF    - Cost-Function update variable
     JDATE,JTIME - Current date and time step values

DATA READ FROM FILES:
     CHKGRID - observed(reference) conc from file: CONC_L3CHK

OUTPUT:
     LGRID,CF
```

- **tridiag_adj.F (** Subroutine: ADTRIDIAG **)**

```
FUNCTION:
    -> Discrete adjoint of TRIDIAG subroutine that comes with CMAQv4.5
    -> Solves tridiagonal system by Thomas algorithm.  Algorithm fails
```

```
       ( M3ERR ) if first pivot is zero.  In that case, rewrite the
       equation as a set of order KMAX-1, with X(2) trivially eliminated.

Associated tri-diagonal system is stored in 3 arrays
   D : diagonal
   L : sub-diagonal
   U : super-diagonal
   B : right hand side function
   X : return solution from tridiagonal solver

     [ D(1) L(2) 0    0    0 ...       0     ]
     [ U(1) D(2) L(3) 0    0 ...       .     ]
     [ 0    U(2) D(3) L(4) 0 ...       .     ]
     [ .       .     .     .           .     ] X(i) = B(i)
     [ .             .     .     .     0     ]
     [ .                   .     .    L(n)   ]
     [ 0                         .    D(n)   ]

     where n = NLAYS
```

- **routines.f (** `Subroutine: setulb` **)**

```
Function:
     This subroutine partitions the working arrays wa and iwa, and then uses
     the limited memory BFGS method to solve the bound constrained
     optimization problem by calling mainlb.(The direct method will be used
     in the subspace minimization.)

INPUT:
     x,f,g

OUTPUT:
     x
```

|---|---|---|---|---|---|