Laboratory for Aviation and the Environment

Massachusetts Institute of Technology

Application of the complex step method to chemistry-transport modeling – GEOS-Chem XPLEX

Bogdan V. Constantin and Steven R.H. Barrett

A technical note by the MIT Laboratory for Aviation and the Environment

Date: May 2014
Report No: LAE-2014-006-N
Website: LAE.MIT.EDU

GEOS-Chem XPLEX

Technical Note

by Bogdan Constantin

5/16/2014

This paper is addressed to the users of GEOS-Chem XPLEX as well as to those who want to implement the complex step method in a more recent version of GEOS-Chem (or any other large scale model written in FORTRAN).

Introduction of the complex step method to GEOS-Chem

The complex step (CS) sensitivity (or derivative) is a forward sensitivity numerical method based on complex variable properties (Squire and Trapp, 1998). It mitigates disadvantages of finite difference methods (Martins et al., 2003) but retains the accuracy of forward algorithm differentiation (AD) methods (the decoupled direct method) without the associated effort to implement (Giles and Pierce, 2000). The implementation of the complex step method in different programming languages is presented by Martins et al. (2003). In principle, the implementation requires 3 steps:

1. Changing all the real variables to a complex data type.
2. Overloading intrinsic functions and operators for complex data type usage.
3. Apply a small imaginary complex step to the desired input variable or set of variables.

The result will be that the imaginary components of the outputs of GEOS-Chem will hold the information of the sensitivity of each individual output to the variable or set of variables that had an imaginary complex step inputted. To obtain the sensitivity one just needs to divide the imaginary parts of the outputs by the imaginary complex step. Note that the complex step is in the units of the variable that had the imaginary complex step inputted. Also note the complex step needs to be several orders of magnitude smaller than the real counterpart to avoid coupling of the real and imaginary part and altering the background physical and chemical processes. Martins et al. (2003) suggest 20 orders of magnitude to avoid this problem.

The original GEOS-Chem model and its associated adjoint model are written in FORTRAN and have data types REAL*4 and REAL*8 (mixed precision – essentially single!). The complex data type needs to be double precision to take full advantage of the capability of the complex step method to compute sensitivities with machine precision accuracy. The imaginary components of the variables can therefore have values as low as approximately 10^{-302} under which underflow occurs and the sensitivities are flushed to zero. This is required especially when the CS is used in the adjoint as the real parts of the variables can get as low as 10^{-32}.

In the next sections we first present the reason for using XPLEX and not the intrinsic COMPLEX*16. Then we give instructions on using the model. Finally we present the implementation of XPLEX in GEOS-Chem (which can be used as a guideline for other large FORTRAN applications).

Reason for XPLEX – Numerical "hurdles"

In practice, the implementation is not as simple as 1, 2 and 3 presented in the introduction.

We first tried to convert the code to the FORTRAN double precision intrinsic data type COMPLEX*16 with a word processor script we wrote in Perl and then overload the intrinsic functions and operators as per Martins et al. (2003).

The code ran but after a few time steps the values started to diverge and eventually the code was unable to converge and the simulation crashed. We observed that, during the simulation, the imaginary components of the variables were getting larger in magnitude which indicated the fact that some of the imaginary components were on the same order of magnitude as some of the real components. This led to the real components diverging as well.

This occurred because of certain numerical "kludges" that are implemented in the chemical module of the model which resets values which did not converge to a positive value to 1d-99. These kinds of "kludges" also appear in other modules of the model with values of 1d-32, 1d-20, etc and are not physical. The problem is that when a complex number that has an imaginary non-zero component gets divided by one of these "kludge", the imaginary component of the result will get relatively large compared to the real component of the result and this is non-physical.

For example let 1d-200 be the initial input imaginary complex step and the complex value of a variable at some point during the simulation be (1d0, 1d-250). The imaginary component of the division of this number by (1e-99, 0d0) will be 1d-151 which is 99 orders of magnitude higher than the sensitivity of the numerator. The extra 99 orders come from the kludge. In order to avoid this we decided that, because it is a non-physical value anyway, the imaginary part of the division should be zero (we don't want to compute sensitivities with respect to numerical "kludges" but with actual physical phenomena described by the model).

The most elegant option to implement this idea would have been to overload the operators in such a manner than when they detected division by a kludge then the imaginary component of the result would be set to zero.

Unfortunately the compiler does not allow overloading of intrinsic operators for intrinsic data type, COMPLEX*16 and this is the reason why we had to create a user defined double precision complex data type. We named this data type, XPLEX (hence GEOS-Chem XPLEX)

Using GEOS-Chem XPLEX

One of the main reasons for implementing the complex step method is because of its simplicity. The setup of the simulation is identical to any GEOS-Chem adjoint simulation. This can be found at http://adjoint.colorado.edu/~daven/gcadj_std/.

Once the simulation is set up the following steps need to be taken to compute sensitivities:

1. Input an imaginary complex step in the desired input(s) of the model.

   Example: STT(I_h,J_h,L_h,ID_h)%i = 1d-100 ,

   where:

   - I_h,J_h,L_h are the grid cell coordinates where the complex step is applied
   - ID_h is the tracer ID to which the complex step was applied
   - %i references the imaginary component of the XPLEX variable STT
   - 1d-100= h, the complex step

   This works for any input variables (emissions, temperature, precipitation, etc.).

2. Run the simulation

   Once the simulation is over, the imaginary components of the outputs will contain the information. It is necessary to note that outputs are first cast to COMPLEX*16 before being written to the bpch files to facilitate use of previous scripts to read the data.

   The simulation takes about 4.5× the time of regular GEOS-Chem. Factors that contribute to this may be the double precision of the XPLEX type, the fact that complex arithmetic has more operations and (maybe most of all) checking for "kludges". As a reference, 1-yr XPLEX adjoint simulation (forward and backward) would take roughly 12 days on 10 cores. This is not unreasonable.

3. Divide the imaginary components by the imaginary complex step inputted to obtain sensitivity:

   $Im\{NO_x(I,J,L)\}/h = dNO_x(I,J,L)/dSTT(i_h,j_h,l_h,ID_h)$, for every I,J,L in the grid domain.

Implementation of XPLEX

We created a semi-automatic Perl script, *modifyx.plx*, which converts real variables in the original code to type XPLEX. This scrip also modifies the modules of GEOS-Chem by inserting the lines: USE MYTYPE and USE XPLEXIFY wherever necessary in the code. A requirement for the proper addition of these USE statements is that the module must be written with IMPLICIT NONE statements. A simple grep -i "IMPLICIT NONE" in the command window should indicate if the module has such a statement. If it does not, the statement must be added.

mytype.f90

mytype.f90 is the module which contains the definition of the XPLEX variable as a structure of two double precision real variables: REAL*8 :: r – for the real component and REAL*8 :: i –for the imaginary component. To reference the individual components use "%r" for the real component of the variable or "%i" for the imaginary components.

xplexify.f90

xplexify.f90 is the module which contains the definitions of each operator and function which is performed in the code. This module is based on the *complexify.f90* provided by Martins et al. (2003).

The definitions for operations and functions that are defined for complex variables are performed as such by casting the input XPLEX variables to COMPLEX*16. The exception is the ABS function. As described by Martins et al. (2003), the ABS is defined as the root of the sum of the squares of the components but it needs to be redefined in order to maintain the original real variable code thread. The ABS function was defined as the absolute of the real component of XPLEX, changing the sign of the imaginary part depending on the original sign of the real component (if the real component was positive then the sign stays the same, if the real component was negative, the sign of the imaginary part changes).

ATAN and ACOS are defined as in complexify.f90.

The MIN, MAX, MINVAL, MAXVAL functions as well as the logical operators are performed on the real component of the XPLEX variables to maintain the same thread of the calculation as the original code.

The operators which incorporate divisions of complex numbers and the division operators have checks for "kludges". This was the key for successful implementation of XPLEX.

Converting the code to XPLEX

Start from the original code

As said before it is absolutely necessary that all modules have IMPLICIT NONE! *modifyx.plx* will not work properly if they do not.

Run *modifyx.plx* on all the .f, .F, .f90, .F90 files which have real variable definitions REAL*8, REAL*4, REAL(kind=dp), REAL(kind=4), DOUBLE PRECISION and any other real variable definitions. This will convert to TYPE (XPLEX). It also adds "USE MYTYPE" and "USE XPLEXIFY" to the USE statement block.

```
>perl modifyx.plx *.f* (*.F*)
```

Parameters and data blocks defined as:

```
REAL*8, PARAMETER :: a
REAL*8, PARAMETER :: /a,b,c,d,…,/
```

Could be redefined as

```
TYPE (XPLEX), PARAMETER :: xplex(a,0d0)
TYPE (XPLEX), PARAMETER ::/ xplex(a,0d0)  , xplex(b,0d0),…/
```

but this is not necessary. Parameters do not change value during the simulation so the imaginary part will always remain zero!

The second step in converting the code is to copy-paste *mytype.f90* and *xplexify.f90* in the folder with the main code. Do not do this before running the Perl script as it will convert the real variables you need in those as well.

Modify the Makefile that is being used so that mytype.f90 and xplexify.f90 are the first two modules compiled. The rest of the modules depend on these modules. At this stage make sure that "-r8" double precision flag is being used for the compiler.

The next stage is to change the READ/WRITE statements.

The WRITE statements need to be changed to cast the XPLEX in COMPLEX*16: dcmplx(var%r,var%i). (var is a variable). Also WRITE descriptors need some attention as they could cause runtime errors if they do not match the data. For example if the original descriptor is F10.2, it needs to be 2F10.2.

Because the data on disk is real it needs to be read as real. Dummy real variables need to be defined to read the data from disk and then immediately pass the data in the real components of the XPLEX variables.

An exception from this are the READ statements of the checkpoints in the adjoint code. This data is saved in COMPLEX*16 and so dummy COMPLEX*16 need to be defined to read the data and then pass the real and imaginary components into the real and imaginary components of the XPLEX variables. This is crucial in the case of computing second order sensitivities as the imaginary component of the forward code needs to be passed into the adjoint imaginary code.

Note that, unfortunately, these READ/WRITE need to be done manually.

Possible errors and recommended method of converting code

More often than not making a large number of changes at once will result in some human error. This could cause the code not to compile or to have runtime errors. The most common compilation errors are "exceeding number of columns" or "variable type mismatch". These are easy to fix and should not take an excessively long time. Compilation of the code with the "-traceback -g" flags should make the debugging a very easy process. The most common runtime error is due to descriptors in WRITE statements that have been omitted when converting them for COMPLEX*16 usage.

Our recommendation is for the code to be debugged by using "-traceback -g" flags before starting to change the READ and WRITE statements. Essentially change them where needed. This is not an extremely time consuming process even for a code such as GEOS-Chem adjoint. After the code runs the READ statements for the adjoint checkpoints must be changed. ATTENTION! The XPLEX adjoint will not work if the checkpoints are not read as described above (create dummy COMPLEX*16 variables and cast the values into the corresponding components of the XPLEX variables).

Disclaimer

The code provided in the GC_XPLEX package may not be bug free.

References

Martins, J.R.R.A., Sturdza, P., Alonso, J.J., 2003. The complex-step derivative approximation. Association for Computing Machinery on Mathematical Software 29, 245-262.

Squire, W., Trapp, G., 1998. Using complex variables to estimate derivatives of real functions. Society for Industrial and Applied Mathematics Review 40, 110-112.