Applying Automatic Differentiation to the Community Land Model

Azamat Mametjanov, Boyana Norris, Xiaoyan Zeng, Beth Drewniak, Jean Utke, Mihai Anitescu, and Paul Hovland*

Abstract Earth system models rely on past observations and knowledge to simulate future climate states. Because of the inherent complexity, a substantial uncertainty exists in model-based predictions. Evaluation and improvement of model codes are one of the priorities of climate science research. Automatic differentiation enables analysis of sensitivities of predicted outcomes to input parameters by calculating derivatives of modeled functions. The resulting sensitivity knowledge can lead to improved parameter calibration. We present our experiences in applying OpenAD to the Fortran-based crop model code in the Community Land Model (CLM). We identify several issues that need to be addressed in future developments of tangent-linear and adjoint versions of the CLM.

Key words: Automatic differentiation, forward mode, climate model

1 Introduction

The Community Earth System Model (CESM) [2], developed by NCAR since 1983 and supported by NSF, NASA, and DOE, is a global climate model for simulations of Earth's climate system. Composed of five fully coupled submodels of atmosphere, ocean, land, land-ice, and sea-ice, it provides state-of-the-art simulations for research of Earth's past, present, and future climate states on annual to decadal time scales. The coupled-system approach enables modeling of interactions of physical, chemical, and biological processes of atmosphere, ocean, and land subsystems without resorting to flux adjustments.

The CESM has been used in multicentury simulations of various greenhouse gases and aerosols from 1850 to 2100. It has also been used for various "what-if"

*Corresponding author: hovland@mcs.anl.gov
Mathematics and Computer Science Division, Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439

scenarios of business-as-usual prognoses and prescribed climate policy experiments for acceptable climate conditions in the future up to year 2100.

The Community Land Model (CLM) is a submodel of CESM for simulations of energy, water, and chemical compound fluxes within the land biogeophysics, hydrology, and biogeochemistry.

Because of the complexities of the global climate state, a significant variability exists in model-based predictions. Therefore, the primary goal of climate modeling is to enable a genuinely predictive capability at variable spatial resolutions and subcontinental regional levels.

The increasing availability of computing power enables scientists to not only analyze past climate observations but also to synthesize climate state many years into the future. CESM, for example, is executable not just on leadership-class supercomputers but also on notebook machines. In these settings of unconstrained availability of simulations, one can iteratively run a model in diagnostic mode to tune model parameters and execute prognostic runs with a higher degree of confidence. Nevertheless, because of the large number of parameters and the attendant combinatorial explosion of possible calibrations, uncertainty quantification and sensitivity analysis techniques are needed to estimate the largest variations in model outputs and rank the most sensitive model inputs.

In the optimization of numerical model designs, automatic differentiation (AD) [4] provides indispensable tooling by efficiently computing derivatives of model outputs with respect to inputs. Derivative information allows for estimation of output changes due to changes in some of the inputs, enabling sensitive input classification. Further, derivatives can be obtained at a fraction of the cost of computing model outputs, which makes AD significantly more efficient than manual parameter perturbation and finite difference-based calibration. A wide number of applications of AD have been reported with many numerical computations in physical, chemical, biological, and social sciences [1, 10]. To the best of our knowledge, AD is yet to be applied to the land model climate code. Therefore, we present our initial findings of differentiating the CLM, the commonalities with previous applications, and differences that are specific to the land model code.

We begin with an overview in Sect. 2 of the CLM model and its crop model subunit. Section 3 provides a brief overview of OpenAD – a modular AD tool for transformations of C/C++ and Fortran codes. In Sect. 4, we describe the development of a tangent-linear code with OpenAD. In Sect. 5, we present the results of our experiment in applying OpenAD to the CLM's crop model, including a discussion of our experiences and lessons learned in the differentiation of climate code. Section 6 closes with concluding remarks and future work.

2 Background

The CESM provides a pluggable component infrastructure for Earth system simulations. Each of the five components can be configured in active, data, or stub modes,

allowing for a variety of simulation cases. There is also a choice of a coupler – either MCT [7] or ESMF [3] – to coordinate the components and pass information between them. During the execution of a CESM case, the active components integrate forward in time, exchanging information with other active and data components and interfacing with stub components.

The land component in active mode models the land surface as a nested subgrid hierarchy, where each grid cell can have a number of different land units, each land unit can have a number of different columns and each column can have a number of different plant functional types (PFTs). The first subgrid level – land unit – captures the broadest land surface patterns such as glacier, lake, wetland, urban, and vegetated patterns. The second subgrid level – column – has similar surface patterns to those of the enclosing land unit but captures vertical state variability with multiple layers of water and energy fluxes. The third level – PFT – captures chemical differences among broad categories of plants that include grasses, shrubs, and trees.

In order to improve the modeling of carbon and nitrogen cycles, the CLM has been updated with managed PFTs of corn, wheat, and soybean species. Each PFT maintains a state captured in terms of carbon and nitrogen (CN) pools located in leaves, stems, and roots and used for storage or growth. The CN fluxes among PFT structures determine the dynamics of vegetation. A significant contributing factor that affects CN fluxes is the ratio of CN within different structures. A large uncertainty exists regarding the CN ratios, which therefore are the primary targets of calibration in order to improve the overall model accuracy.

3 Automatic Differentiation and OpenAD

Automatic differentiation [4] is a collection of techniques of evaluating derivatives of functions defined by computer programs. The foundation of AD is the observation that any function implemented by a program can be viewed as a sequence of elementary operations such as arithmetic and trigonometric functions. In other words, a program $\mathscr{P}$ implements the vector-valued function

$$\mathbf{y} = \mathbf{F}(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m \tag{1}$$

as a sequence of p differentiable elemental operations

$$v_i = \phi(\ldots, v_j, \ldots), i = 1, \ldots, p. \tag{2}$$

The derivatives of elemental operations are composed according to the chain rule in the differential calculus.

The key concepts in AD are *independent* variables $u \in \mathbb{R}^a, a \leq n$, and *dependent* variables $v \in \mathbb{R}^b, b \leq m$. Any variable within program $\mathscr{P}$ that depends on (or is *varied* by) values of independent variables and contributes to (or is *useful* for) values of dependent variables is known as *active*. Active variables have value and derivative components.

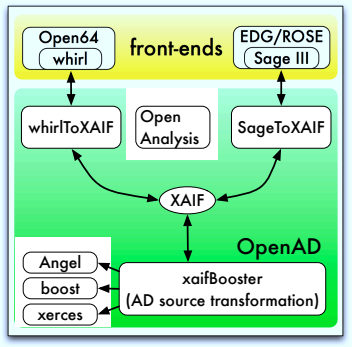

Fig. 1 OpenAD components. Front-ends parse the input source into an IR, which is further translated into XAIF that represents the numerical core of the input. After the AD of the core, the results are unparsed back into source text.

Because of the associativity of the chain rule, there exist two modes of AD. In the *forward* (or *tangent-linear*) mode, derivative computation follows the original program control flow and accumulates derivative values starting from independent variables to dependent variables. In the *reverse* (or *adjoint*) mode, derivative computation follows the reverse of the original control flow and derivatives are accumulated from dependent to independent variables.

Derivative values of active variables can be propagated in at least three ways. First, source-to-source transformations can be used to derive new program $\mathscr{P}'$ that adds new code to the original program code to propagate derivative values. Second, operator-overloading of elemental operations involving active variables can also be used to propagate the derivatives. Third, a complex-step method [6] can be used to view active variables as complex numbers, where the real part stores original variable values and the imaginary part propagates derivative values.

OpenAD [12] is a source-to-source transformation-based AD tool built from components (see Fig. 1). Two front-ends are currently supported: Rose [11] for C/C++ and Open64 [8] for Fortran 90. The intermediate representations (IR) created by the front-ends are translated by using OpenAnalysis [9] into XML abstract interface format (XAIF) [5], which represents the numerical core of the input source. This representation is transformed to obtain the derivatives, and the result is unparsed back into the front-end's IR for further unparsing into the original source language.

4 AD Development Process

The intended process flow for the automatic differentiation of numerical codes is to limit manual intervention to the identification of independent and dependent variables and to let an AD tool generate an efficient code that computes valid deriva-

tives. However, practical implementations of AD are not yet fully autonomous, and manual development is often necessary to pre- or postprocess the codes or to "stitch" together differentiated and other/external code. Such interventions are cross-cutting, requiring a collaborative effort between domain scientists who developed the original numerical code and AD developers who have expertise in source code analysis and transformation.

In order to reduce the need for manual intervention, it is important to identify patterns and antipatterns of effective programming practices in the development of numerical codes as a means of making the code that implements the numerical functions more amenable to sensitivity analysis or other analyses requiring derivative computations. The emerging patterns can then be targeted and automated by either source code refactoring tools or AD preprocessing tools.

In the current work of model optimization and parameter calibration, our initial goal was to identify whether AD can be performed at all, and, if not, to identify obstacles for developing derivative code. To date, we have succeeded in the differentiation of a subunit of the land model code, and we have expended considerable effort into discovering and resolving the obstacles. In the process, we have gained some pattern- and process-related insights, which we report below. As we develop greater expertise in the cross-cutting issues in climate model and intrusive AD analysis domains, we expect greater efficiency and/or automation of AD development. Our goal is to develop and validate AD code for the entire CLM.

4.1 Code Comprehension

The initial step in any AD effort is to understand the original code. Typically, well-maintained codes have documentation in the form of installation guides, user manuals, and HTML documentation generated from source code comments. For AD, one also needs information about source code structures and modules. Dynamic function/procedure call graphs can provide dependency information.

The CLM code is a well-documented Fortran 90 code with a user guide and manual that allow for quick installation and execution. However, most of the documentation is targeted at climate experts, with little information about the source code or how to modify and extend the model code. Therefore, we were on own in the comprehension of code dependencies.

The CLM source code consists of $\approx$70K lines of code in biogeochemistry, biogeophysics, hydrology, and coupler modules. Because of the complexity of differentiating all the CLM code, we chose to prototype a derivative of a smaller subunit. Fortunately, we had access to a climate scientist who had recently extended the biogeochemistry module of the CLM with a model of managed crop species of corn, wheat, and soybeans — CLM-Crop. Therefore, we chose the CLM-Crop unit for the initial AD prototype.

To understand the dependencies between CLM-Crop and other subunits, we constructed a dynamic function call graph. This work entailed porting CESM from PGI

compilers to the GNU compiler suite, which provides a built-in dynamic function call profiler `gprof`. Based on the call graph, the first candidates for AD were the nodes that had minimal calls to and from other nodes.

4.2 Preprocessing

Having identified the subroutines for AD, we started preparing the source code for OpenAD transformations. Since the code for differentiation must be visible to the tool, the recommended development pattern is to identify or create a top-level, or *head*, subroutine that invokes all other subroutines subject to AD. The annotations of independent and dependent variables are inserted into the head subroutine. Then, the head and all invoked subroutines are concatenated into a single file, which is transformed by the tool. The advantage of having a head subroutine is that it enables (1) seeding of derivatives of interest before any call to differentiated code and (2) extraction of computed derivatives upon completion of all computation of interest. Both seeding and extraction can be performed in a *driver* subroutine/program that invokes the head subroutine.

One of the antipatterns that we encountered in the model code is the heavy use of preprocessor directives. They are used to statically slice out portions of code that are not used in a certain model configuration. An example is shown below.

```
psnsun_to_cpool(p)     = psnsun(p) * laisun(p) * 12.011e-6_r8
psnshade_to_cpool(p) = psnsha(p) * laisha(p) * 12.011e-6_r8
#if (defined C13)
  c13_psnsun_to_cpool(p)     = c13_psnsun(p) * laisun(p) * 12.011e-6_r8
  c13_psnshade_to_cpool(p) = c13_psnsha(p) * laisha(p) * 12.011e-6_r8
#endif
```

Here, operations related to *C13* are conditioned on whether that preprocessor flag is set. This kind of programming practice can substantially reduce the amount of code for differentiation, which in turn can produce a more efficient code. However, if the goals of differentiation change (e.g., to include new parameters to calibrate) and include the previously sliced-out code, then the result of the previous AD development effort is not reusable for the new AD goals.

A pattern for improved reusability, maintainability, and differentiability is to use control flow branching to evaluate different sections of code instead of relying on the preprocessor for integrating different semantics. For the example above, the preprocessor directives can be transformed to the following.

```
...
if (is_c13(pft_type(p))) then
  c13_psnsun_to_cpool(p)     = c13_psnsun(p) * laisun(p) * 12.011e-6_r8
  c13_psnshade_to_cpool(p) = c13_psnsha(p) * laisha(p) * 12.011e-6_r8
end if
```

Here, the operations are conditioned on whether the type of PFT p is *C13*. This version of model code promotes reuse by retaining the source code of a different model configuration.

4.3 Transformation

After all the source code has been preprocessed and collected into a file, the code can be passed to OpenAD for transformations. The language-agnostic and modular design of OpenAD allows for incremental transformations as follows:

- *Canonicalize* To reduce variability of the input code, it is preprocessed to make it more amenable for differentiation. For example, Fortran intrinsic functions *min* and *max* accept variable number of arguments and do not have closed form for partial derivatives. Calls to these functions are replaced with calls to three-argument library subroutines, which place the result of the first two arguments into the third.
- *Parse (fortran → ir)* The input source code is parsed with the front-end module and converted into its intermediate representation (e.g. Open64's *whirl*).
- *Translate (ir → xaif)* Differentiation of the numerical core of the input program is performed in XAIF. This step filters out various language-dependent features such as replacing de-references of a user-defined type's element with access to a scalar element.
- *Core transformation (xaif → xaif')* The computational graph of the numerical core is traversed inserting new elements that compute derivative values.
- *Un-translate (xaif' → ir')* This step adds filtered out features.
- *Un-parse (ir' → fortran')* Here, we obtain output source code.
- *Postprocess* This step ensures that the output code is valid. For example, variables that were determined to be active acquire the new library-defined type *active*, and all references are updated with value and derivative component accesses $\%v$ and $\%d$.

4.4 Postprocessing

After a differentiated version of the input code has been obtained, the final stage in the process is to compile and link the output with the rest of the overall code base.

If all the model code is transformed, this step is limited to the invocation of the model's regular build routine. However, if only part of the model code is transformed, then this step requires integration of differentiated (AD) and non-differentiated (external) code. A large part of the reintegration is to convert all uses of activated variables in the external code to reference the value component(e.g., *my_var* → *my_var%v*). OpenAD automates this conversion by generating a summary source file that declares all activated variables during the postprocessing stage.

This file is then used by a library script to convert external code files that reference active variables to de-reference the active variables' value component. The conversion is repeatedly invoked on external code files until an executable of the overall model is built.

In our case, differentiation of the CLM-Crop subunit activated a large number of global state variables in the CLM. Since many of these variables were accessed by external code, the postprocessing stage involved a substantial reintegration effort. Over 60 external source files were modified to properly reference active variable values.

5 Results

In this section, we report the results of the experiment of differentiating the CLM-Crop subunit of the CLM code.

Table 1 Independent and dependent variables for AD-based sensitivity analysis

	Description	*Units*
Inputs		
fleafcn	final leaf CN ratio	gC/gN
frootcn	final root CN ratio	gC/gN
fstemcn	final stem CN ratio	gC/gN
leafcn	leaf CN ratio	gC/gN
livewdcn	live wood CN ratio	gC/gN
deadwdcn	dead wood CN ratio	gC/gN
froot_leaf	new fine root C per new leaf C	gC/gC
stem_leaf	new stem C per new leaf C	gC/gC
croot_stem	new coarse root C per new stem C	gC/gC
flivewd	fraction of new wood that is live	none
fcur	fraction of allocation that goes to current growth	none
organcn	organ CN ratio	gC/gN
Outputs		
leafc	leaf carbon	gC/m^2
stemc	stem carbon	gC/m^2
organc	organ carbon	gC/m^2
leafn	leaf nitrogen	gN/m^2
stemn	stem nitrogen	gN/m^2
organn	organ nitrogen	gN/m^2

The inputs and outputs that were chosen for the AD-based sensitivity analysis are summarized in Table 1. As discussed in Sect. 2, the goal of the analysis was to identify the most sensitive parameters for further calibration of model accuracy.

Table 2 briefly summarizes the results of the analysis. For each of the three managed crop types, it reports the derivatives of leaf, stem, and organ carbon and

nitrogen with respect to the 12 independent variables. For example, the partial derivative of corn's *leafc* with respect to *fleafcn* is $\frac{dleafc}{dfleafcn} = 7.0353917$. Similarly, $\frac{dleafn}{dfleafcn} = -93.0305059$ and so forth for each intersection of rows and columns. These values represent accumulated derivatives for one year, where the model integrates forward in time with half-hour (1800-second) time steps.

Table 2 Derivatives of leaf, stem, and organ C and N with respect to selected CN ratio parameters

	CORN		*WHEAT*		*SOY*	
	C	*N*	*C*	*N*	*C*	*N*
LEAF						
$fleafcn$	7.0353917	-93.0305059	5.0544004	-100.2410991	3.2190047	-59.1898603
$frootcn$	4.6136744	-46.2578484	-10.0406592	-72.8134870	2.7559661	-22.6071324
$fstemcn$	1.9315305	0.1931531	2.6791610	0.1786107	0.6794228	0.0271769
$deadwdcn$	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
STEM						
$fleafcn$	14.4554170	0.2891083	57.2387074	1.1447741	34.5605004	0.6912100
$frootcn$	9.4894822	-12.9865931	-116.1697332	-35.8649400	26.2942184	-20.5785076
$fstemcn$	3.9686602	-25.9019159	30.3402387	-52.7061099	7.2945506	-12.3742598
$deadwdcn$	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
ORGAN						
$fleafcn$	1277.0876953	25.5417539	949.6945091	23.7423627	589.0503999	9.8175067
$frootcn$	809.0263835	16.1805277	-1938.7040252	-48.4676006	436.6213665	7.2770228
$fstemcn$	350.6178413	7.0123568	503.3998741	12.5849969	124.3285795	2.0721430
$deadwdcn$	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Taking a closer look at the table, we can observe that some derivatives are not as large as others, indicating that such parameters are not as important as those with larger derivative values. For example, we can observe that corn parameter *fstemcn* does not contribute to the variability of *leafc* output as much as *fleafcn* does. Further, we see that some derivatives are zero, indicating that such parameters do not affect the outputs. This information is of clear benefit to model designers because it identifies the most sensitive parameters for model accuracy calibrations. For example, these results indicate that it is best to focus on CN ratios within corn leaves rather than stems, in order to optimize carbon production within corn leaves, stems, and organs. Other values can be interpreted similarly.

We have validated the results using finite differences by perturbing some of the independent variables and calculating the difference between the original and perturbed dependent variable values. Table 3 provides an example of perturbing wheat's *fleafcn* parameter by 1.5% and comparing derivative estimates for one time step obtained by OpenAD and by finite differences. We can observe that the derivatives obtained by the two methods are in agreement with the relative error of 0.0001 or better.

Table 3 Comparison of selected derivative estimates

	$\frac{dleafc}{dfleafcn}$	$\frac{dstemc}{dfleafcn}$	$\frac{dorganc}{dfleafcn}$
OpenAD	0.000021273	0.0023837	0.0062778
Finite differences	0.000020848	0.0023359	0.0061479

6 Conclusion

In this paper, we have presented the results of our initial effort in constructing tangent-linear and adjoint codes for the CLM. We have focused on the CLM-Crop subunit that models the growth of managed crops. The results of our effort identified which of the model parameters the outputs of interest are most sensitive to. This information will be used to improve the subunit and the overall land model code. As part of the experiment, we have acquired substantial knowledge about the CLM code. In particular, we have identified the data structures and dependencies in the code that enable preservation and forward integration of climate state. Among the lessons learned is the deeply nested structure of the climate state that makes heavy use of global variables. Activation of a single global variable can lead to numerous changes in the code base. In this context, the utility of automated updates of references to activated global variables — the tooling provided by OpenAD — becomes indispensable. Future work in applying AD to the CLM includes differentiation of the overall model and comparison of results obtained using various approaches of forward- and reverse-mode AD, operator-overloaded AD, and complex-step method AD.

Acknowledgments

This work was supported by the U.S. Dept. of Energy Office of Biological and Environmental Research under the project of Climate Science for Sustainable Energy Future, and by the U.S. Dept. of Energy Office of Science under Contract No. DE-AC02-06CH11357.

References

1. Community Portal for Automatic Differentiation. `http://www.autodiff.org`
2. Community Earth System Model. `http://www.cesm.ucar.edu`
3. Earth System Modeling Framework. `http://www.earthsystemmodeling.org`
4. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA (2000)
5. Hovland, P.D., Naumann, U., Norris, B.: An XML-based platform for semantic transformation of numerical programs. In: M. Hamza (ed.) Software Engineering and Applications, pp. 530–538. ACTA Press, Anaheim, CA (2002)

6. Martins, J.R.R.A., Sturdza, P., Alonso, J.J.: The complex-step derivative approximation. ACM Transactions on Mathematical Software **29**(3), 245–262 (2003). DOI http://doi.acm.org/10.1145/838250.838251
7. Model Coupling Toolkit. `http://www.mcs.anl.gov/mct`
8. Open64 compiler. `http://www.open64.net`
9. OpenAnalysis Web Page. `http://www.mcs.anl.gov/research/projects/openanalysis`
10. Rall, L.B.: Perspectives on automatic differentiation: Past, present, and future? In: H.M. Bücker, G. Corliss, P. Hovland, U. Naumann, B. Norris (eds.) Automatic Differentiation: Applications, Theory, and Implementations, *Lecture Notes in Computational Science and Engineering*, vol. 50, pp. 1–14. Springer, New York, NY (2005). DOI 10.1007/3-540-28438-9_1
11. ROSE compiler. `http://rosecompiler.org`
12. Utke, J., Naumann, U., Fagan, M., Tallent, N., Strout, M., Heimbach, P., Hill, C., Wunsch, C.: OpenAD/F: A modular, open-source tool for automatic differentiation of Fortran codes. ACM Transactions on Mathematical Software **34**(4), 18:1–18:36 (2008). DOI 10.1145/1377596.1377598