ANL/MCS-TM-382 Rev 0.10

SICOPOLIS-AD: Quick-Start Manual

Revision 0.10

Mathematics and Computer Science Division

ANL/MCS-TM-382 Rev 0.10

SICOPOLIS-AD: Quick-Start Manual

Revision 0.10

prepared by
Liz C. Logan[1], Sri Hari Krishna Narayanan[2], Ralf Greve[3], Patrick Heimbach[1]

[1]Institute for Computational Engineering and Sciences, University of Texas at Austin

[2]Mathematics and Computer Science Division, Argonne National Laboratory

[3]Institute of Low Temperature Science, Hokkaido University

January 2019

.

Contents

1 Introduction

SICOPOLIS-AD is a version of the ice sheet model SICOPOLIS (originally [1]: `www.sicopolis.net`) used to produce adjoint sensitivities of chosen control variables. It can be used to assess the sensitivity of some quantity of interest to perturbations in variables that may affect that quantity of interest. For example, it can be used to comprehensively and quantitatively assess exactly how the volume of the Greenland Ice Sheet is sensitive to changes in surface temperature at each point on the ice sheet. This document provides basic startup and troubleshooting methods, as well as some best practices for making changes in the code for the purposes of adjoint production.

What is an adjoint model and how did we make it?

The idea of the adjoint may be best understood in terms of the forward, original code construction and execution. If you wish to know the sensitivity of some quantity of interest (e.g., the volume above floatation of the Antarctic Ice Sheet) with respect to some model control variable (e.g., the geothermal heat flux applied to the bottom side of the ice), one method of pursuing such a sensitivity might be by perturbing the control variable, in sequence, at each point within the domain and propagating the perturbation forward in time. The perturbation to the control variable (likely) results in a change in the quantity of interest, and you can calculate the sensitivity of the quantity of interest to the control variable everywhere in the domain. These are finite differences of the quantity of interest, or cost function, with respect to the control variable that approximate the derivatives of the same quantity. After such a sensitivity map is calculated, you can take the analysis further, for instance, by leveraging the sensitivities against some initial guess in model parameters (e.g., the initial or boundary conditions) and approaching some acceptable estimation of their best value given the discretized physics of the numerical model in the form of an optimization problem. Herein lies the greatest advantage of the application of time-dependent adjoint modeling: the reconstruction of important model parameters subject to observations and constrained by the discretized physics of a numerical model.

However the construction of a nonlinear adjoint model often presents a formidable task when solved analytically and hand-coded, e.g., in [2]. As an alternative, in order to approach the ultimate goal of providing the adjoint of a nonlinear ice sheet model, the use of algorithmic differentiation has become increasingly popular [3, 4, 5]. The adjoint of the ice sheet model SICOPOLIS is largely generated by the algorithmic differentiation (AD) tool OpenAD (developed at Argonne National Laboratory; `www.mcs.anl.gov/OpenAD`). It is a source transformation AD tool that differentiates a given model's source code to generate a derivative code which computes derivatives. Algorithmic differentiation (AD) exploits the chain rule of calculus for the computation of derivatives of a function with respect to a set of input variables. Any forward numerical model can be conceived of as a sequence or composition of operations, with a single line representing a single algorithmic step. Via AD methods, then, the derivative of a (often nonlinear) numerical model is provided by exhaustive application of the chain rule, line by line, to the model. The forward or reverse mode (adjoint) of the model may be thought of as the composition in forward or reverse order of the Jacobian matrices of the full forward code's line-by-line algorithmic elements. The derivative SICOPOLIS code is approximately 50,000 executable lines, depending on preprocessor options that are enabled or disabled. By pairing SICOPOLIS with OpenAD, the adjoint of the ice sheet model is provided without the need for hand-coded adjoints, and the adjoint changes automatically following any change made in the source code of the original model.

2 Building SICOPOLIS-AD

Because SICOPOLIS-AD is woven into the source files of SICOPOLIS, downloading from the trunk of the repository is done in the same way as if downloading SICOPOLIS for plain, forward execution. All that you need to produce adjoint sensitivities from SICOPOLIS-AD is (1) a working copy of SICOPOLIS; (2) OpenAD; and (3 - optional) a built version of the Library of Iterative Solvers (LIS), if simulating Antarctic domains with ice shelves.

2.1 Building the source code SICOPOLIS

First, you build the code SICOPOLIS. The QuickStart manual for SICOPOLIS, which can be found at `www.sicopolis.net`, gives excellent and more detailed instructions on how to build the code. Here, we give only a few brief steps.

1. Check out version 5-dev (developmental, latest version) of the trunk using source control Subversion:
   `svn checkout -username anonsvn -password anonsvn \`
   `https://swrepo1.awi.de/svn/sicopolis/trunk sicopolis`
   This places the source code in your local directory sicopolis/ .

2. Go to the new directory `sicopolis` and execute the bash script `copy_templates.sh` :
   `./copy_templates.sh` .

3. Set up your environment:
   Go to `sicopolis/runs/sico_configs.sh` and edit the following flags and their appropriate directory:

   - NetCDF: for forward runs, NetCDF is the best file format to view output results. If using, set `NETCDF_FLAG = "true"` and supply the appropriate location of the NetCDF installation.
   - LIS: if simulating Antarctica with ice shelves, you will need LIS installed. Set `LIS_FLAG = "true"` , and supply the location of your LIS installation. LIS can be downloaded and installed from http://www.netlib.org/misc/lis.
   - For high-resolution simulations (e.g., Greenland at 5 km or Antarctica at 10 km) you will also need to set `LARGE_DATA_FLAG = "true"` .
   - If using the LIS in parallel, you will want to set `OPENMP_FLAG = "true"` .

4. Test the code by executing a template simulation of the Greenland Ice Sheet (this will minimize chances of encountering problems with installations of the bullet-point dependencies above). To do this, copy the template from `sicopolis/runs/headers/templates/` into `headers/` , and from the directory `runs/` type the following.
   `(./sico.sh -m v5_grl20_ss25ka) >out_001.dat 2>&1 &`
   The options "-z" is used if you compiled SICOPOLIS manually, via the bullet points above. See the SICOPOLIS Quick Start Manual (from sicopolis.net) if compiling using GNU Autotools, which is also an option.

5. To see whether the test run was successful, change into the directory `sicopolis/sico_out/v5_grl20_ss25ka` . In it should be several output NetCDF files, a log (.log) of the execution, and a time series ASCII file (.ser) to which various important quantities (ice volume, grounded and floating ice area, etc.) have been written.

2.2 Building OpenAD

While the SICOPOLIS source files are prepared to generate adjoint sensitivities, they will not be able to do so without an operable installation of OpenAD. Fortunately the OpenAD build procedure is straight forward.

1. Download the OpenAD source at `http://www.mcs.anl.gov/OpenAD/`.
2. Change into the source directory, `OpenAD/` .
3. If using bash, type `source ./setenv.sh` . If using c-shell, type `source ./setenv.csh` .
4. Copy the files `IntrinsicXlationTable.cxx` and `inlinable_intrinsics.xaif` from your SICOPOLIS trunk directory `src/subroutines/openad` to the OpenAD directories `OpenAD/OpenADFortTk/src/lib/support/` and `OpenAD/xaif/schema/examples/` , respectively. These two files handle the differentiation of the `atan2()` function invoked in several places in SICOPOLIS, which is not widely distributed in the larger OpenAD src package.
5. Build all components by invoking `make` in that same directory (it takes a few minutes).
6. NOTE: Every time you wish to use the adjoint capability of SICOPOLIS-AD, you must re-source the environment, as in Step 3. We recommend that this be done automatically in your bash or c-shell profile upon login.

You should now have a working copy of OpenAD.

2.3 Optional: Suggestions for building LIS, if simulating Antarctic ice shelves

The adjoint code of SICOPOLIS can differentiate models of Antarctica, including ice shelves. Ice shelves are simulated by using a simplified form of the Stokes equation, called the shallow shelf approximation (termed SSA in parts of the source code). To solve the momentum equation for the SSA, SICOPOLIS uses an external library of solvers, LIS (`https://www.ssisc.org/lis/`). The quick start documentation to build and install this library is nicely detailed on that site. But we suggest that, if you use LIS, you enable the fortran-facing options in the configuration of the installation. That is, after downloading and unzipping the package, you run the following in the lis-<version> directory:

```
[lizlogan % ] ./configure -prefix=<optional/installation/destination> \
-enable-fortran -enable-f90
[lizlogan % ] make
[lizlogan % ] make check # to check that tests have passed
[lizlogan % ] make install
```

3 Running SICOPOLIS-AD

If both SICOPOLIS and OpenAD are installed and built, you are ready to generate adjoint sensitivities. The adjoint code is compiled and invoked from the directory `sicopolis/src/` by using the script `regression_test.sh` .

Everything (except for the `MakefileOpenAD`) pertaining to the adjoint code of SICOPOLIS is located in the subdirectory `src/subroutines/openad/` and comes bundled in the trunk. Here are the steps to take to compile and execute the adjoint code from source files, without your own modifications.

1. Copy `src/subroutines/openad/regression_test.sh`, `src/subroutines/openad/preprocessor.py` to `src/`.
2. Copy the appropriate headers (`sico_specs_ant20_shelves.h` for Antarctic simulation or `sico_specs_new_r` for Greenland) from `src/subroutines/openad/` into the subdirectory `runs/headers/`.
3. Change directory to `src`.
4. Invoke the script using the command `./regression_test.sh`.

The `regression_test.sh` script does several things:

1. Selects a simulation to produce adjoint and finite difference values
2. Runs a "gradient check" to produce finite-difference-based sensitiviites
3. Compiles and runs the adjoint code
4. Compares outputs of both (2) and (3)
5. Writes result of comparison to a file, `regression_log.txt` that remains in `src/` , and moves all other output items to `sico_out/<name of your simulation>/`

Adjoint values must be checked against their corresponding finite differences, in what is termed here the gradient check. Since adjoints offer the sensitivity of a quantity of interest versus a control variable, this can be done by selecting a control variable and perturbing it in the positive and negative direction at a certain location. For example, if we wish to know how the Greenland Ice Sheet volume is sensitive to changes in surface temperature, we might run the original, forward code SICOPOLIS twice to completion to generate the ice sheet volumes for a positive and negative epsilon in surface temperature, and then calculate the sensitivity via

$$\frac{\Delta V}{\Delta x} = \frac{V(x+\varepsilon) - V(x-\varepsilon)}{2\varepsilon}$$

In this case, $V()$ is the "cost" function, or the volume of the Greenland Ice Sheet, and x is the control variable, or temperature, ε is the perturbation in temperature (usually 1e1 works for temperature), and $\frac{\Delta V}{\Delta x}$ is the finite difference-based gradient. This is the value we compare with the adjoint-generated sensitivity to assess whether the adjoint code is producing reasonable values.
The script `regression_test.sh` generates these finite differences along a single line throughout the domains of both Greenland and Antarctica and writes these finite differences to a file that is ultimately moved to `sicopolis/sico_out/<name of your simulation>/` .
The default control variable against which the adjoint code is compared is `H_c` , or the thickness of cold (as opposed to temperate) ice.

3.1 Execution time

The time for adjoint runs of SICOPOLIS to complete are not equal to the length of time for completion of a single forward code sweep. The Quickstart manual for SICOPOLIS (`www.sicopolis.net`) details the completion time of SICOPOLIS for the pre-designed simulations that are contained in the trunk of the SICOPOLIS repository. The defining difference for SICOPOLIS-AD execution time depends essentially on whether or not shallow-shelf dynamics are included in the simulation. In this case, the external solver library LIS is invoked and the completion time of any execution greatly increases.

4 Making changes to the code

We expect that you may wish to make changes to the adjoint source code. This section gives a brief overview of what comes standard in the trunk and where you can make changes. We also present some "best practices" for making changes.

4.1 Default adjoint settings

The adjoint code supplies the sensitivities to multiple control variables in one execution. The default control variables are as follows.

`H_c`	cold ice thickness
`c_drag`	basal drag parameter
`c_slide`	basal sliding parameter
`vx_c`	horizontal velocity in the cold ice in the x-direction
`temp_ma_present`	mean annual temperature
`sigma_c`	effective stress in the cold domain
`vis_int_g`	depth-integrated ice viscosity (ANT only)
`dzs_deta_g`	surface topography derivative of y-coord
`dzs_dxi_g`	surface topography derivative of x-coord
`Q_bm`	basal melting rate
`Q_tld`	water drainage from temperate layer
`acc_fact`	multiplying factor against current precipitation
`calv_grounded`	calving rate of grounded ice
`temp_c(surface)`	temperature of the (upper) cold layer
`temp_c(basal)`	temperature of the (lower) cold layer (only certain simulation options)
`c_drag`	quantity related to computation of basal drag
`precip_present`	present-day mean monthly precipitation rate
`q_geo`	geothermal heat flux

If you wish to calculate sensitivities for variables not specified by default, you will have to modify three different files, and sometimes multiple parts of one file. Following are the places where customizing the control variables takes place.

1. In `src/preprocessor.py` : independent (control) variables are supplied with the syntax

```
!#ifdef ALLOW_OPENAD
!$openad INDEPENDENT(H_c)
!$openad INDEPENDENT(c_drag)
!$openad INDEPENDENT(c_slide)
!$openad INDEPENDENT(vx_c)
!$openad INDEPENDENT(temp_c)
!$openad INDEPENDENT(temp_ma_present)
!$openad INDEPENDENT(sigma_c)
!$openad INDEPENDENT(precip_present)
!$openad INDEPENDENT(q_geo)
!$openad INDEPENDENT(vis_int_g)
!$openad INDEPENDENT(dzs_deta_g)
!$openad INDEPENDENT(calv_grounded)
!$openad INDEPENDENT(dzs_dxi_g)
!$openad INDEPENDENT(Q_bm)
!$openad INDEPENDENT(Q_tld)
!$openad INDEPENDENT(acc_fact)
!#endif
```

   Simply replace `H_c` or or add new lines using the same syntax with the variable of your choice.

2. In `sicopolis/src/subroutines/openad/` modify `ctrl_m.F90` in the following two subroutines.

(a) In subroutine `ctrl_init()` add lines following the syntax used for `xxH_c` or `xxtemp_c`.

```
! CHOICE OF CONTROL VARIABLE IS MADE HERE:
double precision, dimension(0:JMAX,0:IMAX) ::         xxH_c
```

and

```
! 3D controls:
do k=0, KCMAX
  do j=0, JMAX
    do i=0, IMAX
      xxtemp_c(k,j,i) = 0.0d0
      xxvx_c(k,j,i) = 0.0d0
      xxsigma_c(k,j,i) = 0.0d0
    end do
  end do
end do
```

.

Note the changes in syntax for two and three dimensional arrays. Arrays are initialized by nested do-loops, setting each array `xxVar(k,i,j,...) = 0.0d0`, as shown above for the 3D controls. `xxH_c` shown above is a 2D variable, and thus is initialized first by declaring it, including its dimension, and assigning each element in the array to zero, using a nested do loop over the 2 indices.

(b) In subroutine `cost_independent_init()` add lines following the syntax used for `xxH_c` or `xxtemp_c`.

```
subroutine cost_independent_init()

implicit none

integer(i4b) :: i, j, k
double precision, dimension(0:JMAX,0:IMAX) ::         xxH_c
```

and

```
! 3D controls:
do k=0, KCMAX
  do j=0, JMAX
    do i=0, IMAX
      temp_c(k,j,i) = temp_c(k,j,i) + xxtemp_c(k,j,i)
      sigma_c(k,j,i) = sigma_c(k,j,i) + xxsigma_c(k,j,i)
      vx_c(k,j,i) = vx_c(k,j,i) + xxvx_c(k,j,i)
    end do
  end do
end do
```

.

3. In `openad_m.F90` are three subroutines to modify.

(a) In subroutine `grdchk_main()` search for the strings

```
! store original value that will be perturbed
! and then perturb it (note: direction(1) = 0)
! -- H_c
orig_val = H_c(j,i)
H_c(j,i) = orig_val * (1 + direction(d)*perturb_val)
```

and replace `H_c` with the variable of your choice.

(b) In subroutine `print_output()`, search all locations of `H_c` and copy and paste accordingly for your chosen control variable. The default location of the output of that file is in `sicopolis/src/`, which is where the executable operates.

(c) For OpenAD to parse SICOPOLIS and retain adjoint values for your chosen control variable, it must be "active". In this case, go to the subroutine `var_transfer()`. Search for the line with your control variable, and ensure that a `%v` follows it in the declaration. For example, if you decided to include `foo_bar` as a control variable, then search for the line `foo_bar = a_foo_bar` and change it to `foo_bar%v = a_foo_bar`. An example of the syntax for an active variable followed by an inactive variable is as follows.

```
calv_grounded%v = a_calv_grounded
calv_grounded_ap1 = a_calv_grounded_ap1
```

WARNING: Making changes to the source code can be fraught with error! In the process you likely will encounter some error messages because of the changes you made. Proceed with extreme deliberation and caution. If you run into errors send contact to: liz.curry.logan@gmail.com .

4.2 Creating your own adjoint test case

We expect that you will want to create your own adjoint scenario. In `regression_test.sh` are a number of "header" files that can be selected; to create your own case, you make your own header files (2 files) and place them in `sicopolis/runs/headers/` . Currently, the `regression_test.sh` has several different scenarios that can be run by declaring the header file names to be tested. Here is an example.

```
SRC_PATH=$PWD
OUT_PATH=$SRC_PATH/../sico_out/OAD
HEAD_PATH=$SRC_PATH/../runs/headers
OAD_PATH=$SRC_PATH/subroutines/general/openad

# Default header file options to be tested (if you wish to test a custom one,
# modify the header string here):
#declare -a HEADER_FILE=("v5_ant64_b2_spinup09")
declare -a HEADER_FILE=("ant20_shelves_100y")
#declare -a HEADER_FILE=("grl20")
```

This means that the `regression_test.sh` will search for two different header files with the string `ant20_shelves_100y` and compile two different executables based on them to be run in sequence: first the gradient check (executable `HEADER_FILE_GRDCHK.exe`) and second the adjoint code (executable `HEADER_FILE_OAD.exe`). To compile the gradient check executable, `regression_test.sh` searches for the header file in `headers/` that is prepended by "`sico_specs_v5_` " and appended by "`_GRDCHK.h` ", or in this case

`sicopolis/runs/headers/sico_specs_v5_ant20_shelves_100y_GRDCHK.h` .

This file is just like a normal part of the SICOPOLIS development trunk except that at the end it contains the C-preprocessor macro definitions:

```
#define ALLOW_GRDCHK
#define ALLOW_COST
```

Similarly, in the second half of the `regression_test.sh` , the adjoint executable `HEADER_FILE_OAD.exe` is compiled with the analogous file:

`sicopolis/runs/headers/sico_specs_v5_ant20_shelves_OAD.h`
that, instead of the `ALLOW_GRDCHK` preprocessor option at the end contains the option

```
#define ALLOW_OPENAD
#define ALLOW_COST
```

The two files must match each other exactly except for two lines: in their `RUNNAME` and at the end where either `ALLOW_OPENAD` or `ALLOW_GRDCHK` is defined. When you are ready to make your own adjoint test case, simply copy any header file in `sicopolis/runs/headers/templates/` to the directory `headers/` two times, once with `_GRDCHK.h` and once with `_OAD.h` as file name suffixes, and at the bottom of each header file insert the appropriate `#define <MACRO>` .

Then, in `regression_test.sh` , replace (or add) to the `HEADER_FILE` variable your file name. If your test case's header is `sico_specs_v5_foo_bar*.h` , then you will insert

`declare -a HEADER_FILE=("foo_bar")` .

Once you have your two header files created and appointed in the `regression_test.sh`, you are ready to test your own case. Simply run the shell script.

5 General advice for changing the source

The adjoint code of SICOPOLIS was created by algorithmic differentiation: basically, OpenAD (the source transformation tool) parses the source files of SICOPOLIS and differentiates, line by line, to collect and output adjoint sensitivities. This means that, in the trivial case where your source code is simply the line

```
Y = 2.0 * X
```

and your independent and dependent (or control and cost function) variables are **`X`** and **`Y`**, the adjoint sensitivity is `2.0`.

You should try **not** to insert anything non differentiable into the code. If you do, OpenAD will likely complain and fail to compile. Worse, it may silently generate derivative code that is incorrect. Examples of non differentiable functions are

`abs(X)`, or absolute value;
`sqrt(0)`, or square roots evaluated at zero;
`exit`;
`go to`;
`modulo`;

```
if (X.lt.0) then
   Y = X
else
   Y = 2.0 * X
```

For the piecewise linear example given, the slope is 1 for X less than 0, and 2 for X greater than zero, but not defined at X. Another common source of errors is the assumption that the AD tool understands algebraic simplification. For example

```
if (a .eq. 1.0) then
   y = b
elseif (a .eq. 0.0) then
   y = 0
else
   y = a*b
```

The intention may have been to compute $\dot{y}$=a$\star\dot{b}$+b$\star\dot{a}$, but the simplification prevents it.

6 Tested architecture and compute times

SICOPOLIS-AD has been built successfully on computers at the University of Texas, Argonne National Laboratory, and the Massachusets Institute of Technology.

These are the following CPU specifications:

CPU	architecture
Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz	x86_64
Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz	x86_64

The following compute times have been calculated for Antarctic (with ice shelves) and Greenland simulations at different resolutions:

Domain	resolution (km)	time step (years)	compute time (per 100 yr) (minutes)
ANT	64	1	20
ANT	40	0.4	75
ANT	20	0.2	600
GRL	40	2	5
GRL	20	1	10
GRL	10	0.5	140

While SICOPOLIS has the ability to calculate the dynamic and thermodynamic equations at different time steps, we found that calculating the thermodynamics at the same frequency yielded the most stable results.

7 Acknowledgements

This work was funded in part by support from the National Science Foundation and the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

References

[1] Ralf Greve. A continuum–mechanical formulation for shallow polythermal ice sheets. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 355(1726):921–974, 1997.

[2] Tobin Isaac, Georg Stadler, and Omar Ghattas. Solution of nonlinear stokes equations discretized by high-order finite elements on nonconforming and anisotropic meshes, with application to ice sheet dynamics. *SIAM Journal on Scientific Computing*, 37(6):B804–B833, 2015.

[3] Patrick Heimbach and Veronique Bugnion. Greenland ice-sheet volume sensitivity to basal, surface and initial conditions derived from an adjoint model. *Annals of Glaciology*, 50(52):67–80, 2009.

[4] E. Larour, J. Utke, B. Csatho, A. Schenk, H. Seroussi, M. Morlighem, E. Rignot, N. Schlegel, and A. Khazendar. Inferred basal friction and surface mass balance of the northeast Greenland ice stream using data assimilation of icesat (ice cloud and land elevation satellite) surface altimetry and issm (ice sheet system model). *The Cryosphere*, 8(6):2335–2351, 2014.

[5] Daniel N Goldberg, Sri Hari Krishna Narayanan, Laurent Hascoet, and Jean Utke. An optimized treatment for algorithmic differentiation of an important glaciological fixed-point problem. *Geoscientific Model Development*, 9(5):1891–1904, 2016.

SICOPOLIS-AD is free and open-source software. It can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at the user's option) any later version.

SICOPOLIS-AD is distributed WITHOUT ANY WARRANTY and the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg.
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC