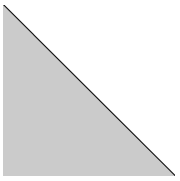

University of Tromsø

A guide to the radiation code of CESM 5

Arthur S. Nørve

15. desember 2017

Just running physical climate models can be hard enough in itself, changing code and verifying the results possibly even harder. In this article we'll cover how to make modifications to the radiation module of the CESM 5 (Community Earth System Model 5). To do this we will go through the code and the functions and routines within it, and explain what the various pieces do. Next, we bring up how to actually change the code, tricks and caveats, and in the last part we give examples of which code to change for some example purposes.

Innhold

1 Introduction

1.1 About CESM 5

The CESM model is written in Fortan and developed by NCAR (National Center for Atmospheric Research). It is a fully coupled model that accounts for the atmosphere, the oceans, ice covers, land mass and topography, carbon and water cycles, and more.

1.2 Main flow of CESM

The CESM model primarily consists of 6 large subcomponents:

1. atm - Atmosphere
2. ice - Sea-ice
3. lnd - Land
4. ocn - Ocean
5. glc - land-ice
6. cpl - Coupler

If CESM was an orchestra, the coupler would be the conductor; it is responsable for shipping the calculated data between the modules (radiation to land for example) and making sure everything happens in the right order for each time step. The other modules have relatively obvious areas of concern. Using different 'component sets' one can combine these subcomponents in different ways depending on available computing resources and data, time or other concerns and purposes.

1.3 Brief intro to the radiation module

CESM 5 includes two separate versions of the radiation code; the first is the old one from CESM 4 (which we will not discuss here), and the new one, the rrtmg module,

which is stands for rapid radiative transfer model, divvides the solar spectrum into bands and calculate the radiative fluxes and heading rates. The structure of the rrtmg code is more complicated than the old CAM4 module, with many more options and deeper code (many function calls from other files). This means that many types of cahnges will be harder and require more fiddeling when working with rrtmg. It is favourable however since the rrtmg uses higher resolution than the old CAM4 method.

1.4 Main flow of the radiation module

Inside the radiation module the main file is radiation.F90, inside this file two main things happen: the module itself and its dependencies are initialized (radiation_init) and the radiation calculation is started and managed (radiation_tend).

In radiation_init, other required modules are loaded, the short and longwave calculation modules are initialized, radiation parameters are fetched and all the outflds are added so that they appear in the history files (addoutlfd). This is the start of the radiation module flow.

In radiation_tend the needed local variables are defined and fetched from their relevant source, the rrtmg_state is loaded, preliminary computation is done with the radiation variables and the two routines rad_rrtmg_sw and rad_rrtmg_lw are called. These two routines reside in the files radsw.F90 and radlw.F90 respectively.

In these two files we find the radlw_init,rad_rrtmg_lw and the radsw_init, rad_rrtmg_sw routines. rad_rrtmg_lw and rad_rrtmg_sw contains code toset up the main calculations and pass parameters to the routines responsible for the actual calculations: rrtmg_lw and rrtmg_sw. These two routines are both found in the ext/ folder in folders with the same name. It is in those files that the rrtmg algoritm is employed for the atmospheric layers.

2 File structure

The rrtmg radiation module is locatet in the 'models/atm/cam/src/physics/rrtmg' (The old CAM4 module is located in the same folder, but under /cam: 'models/atm/cam/src/physics/cam'

3 Code changes intro

Given that all of the sourcecode for the model compiles every time you compile your case, your modifications are indeed modification to the model code/core itself. When you create a CESM case, the model scripts put a SourceMods folder in the case folder itself. Within this folder there are subfolders for each major component of CESM as a whole. When you modify a file from one of these components, you just put it straight into the subfolder corresponding to the component, the model scripts figures out where it goes by the filename. As an example the radiation module lies within the CAM component, let's say we modify the main radiation file (radiation.F90), the file would then need to be put here: SourceMods/src.cam/radiation.F90.

Doing the modifications themselves is harder since you actually have to know the code to modify it. The simplest type of modifications consist of substituting an existing variable for a new one, or equivalently modify an existing variable. In many cases you only want your changes have an effect inside the radiation module and not any other place. For this to happen you have to be careful to make a copy of the variable and then substitute all instances of the original variable with the substitute. In this way you have total control of the flow of data to the radiation calculation. This is exactly what we want when studying perturbations of variables. See the example section for specific code changes.

4 Notes on Standard Code

4.1 addoutlfd()

The addoutfld functions is a CESM base function that tells the CESM logging component to make room for another field in the history files. This is necessary if you want to wrint out a modified or old variable so that you can verify changes after the model run has completed.

4.2 outfld()

This CESM base function actually writes the fortran variable that it is passed to the field corresponding to the given field handle. The handle is the same as added in the addoutfld function.

5 Caveats and Tricks

5.1 physics_buffer

The physics buffer is responsible for keeping arrays and variables that must have a lifetime that extends a single timestep. The caveat here is that some variables, in some modules are fetched from the physics_buffer. Hence it is important to not use/change these variables before they have been filled in from the buffer. This for example happens in the rrtmg module for the cloud fraction vvariable.

6 Examples of changes

6.1 Necessary general changes

For this example we will be changing the cld variable. This is the container for the cloud density for the entire atphosphere. This change requires three steps:

1. Add substitution variable definition

2. Modify the variable

3. Add field output code

All these changes are done in 'radiation.F90', except maybe step 2 which can go deeper than that.

6.1.1 Add substitution variable definition

Here we will have to add a line that can hold our modified variable:

```
real(r8) :: cld_sub(pcols,pver)  ! cloud fraction substitute !
    arthurnoerve
```

This is a quite standard Fortran 90 declaration, but you might wonder: How do I know which type the variable is? The answer is: you don't, and you have to go searching. By searching through the entire source code for the variable you want to substitute (see apendix A), you should be able to find its definition somewhere. Copy that and change it to declare your substitition variable. Lastly rememnber to do a search and replace that changes instances of the old vavriable to the new one. Be careful and don't do a replace all, that will create duplicate definitions and more. The trick is to only replace the occurances of the old variable in the model code itself.

Remember to put your name in a comment behind the given line, that way you'll easily be able to find your changes again later.

6.1.2 Modify the variable

See examples below for different ways of changing the variable themselves.

6.1.3 Add field output code

This step makes sure that you get the modified field written out to the model history files (the product of the simulation). As mentioned above in the notes on standard

code this consists of two steps: first we add our field to the list of avavilable fields, next up we actually write our variable out. We put this code right at the end of the radiation_init() function:

```
call addoutfld ('cld_old','Kg/Kg',pver, 'A','Cld old',phys_decomp,
    sampling_seq='rad_lwsw') ! arthurnoerve
```

Secondly we add the code that writes the variable to the field. Remember that this has to be inserted after the modification has taken place. Right after is fine, but I tend to put it together with the other calls to 'outlfd' (after the calls to rad_rrtmg_sw() and rad_rrtmg_lw():

```
call outfld('cld_old', cld, pcols,lchnk) ! arthurnoerve
```

Again we mention that you should search for the call to addoutfld()/outfld() for the variable you are working with and use that as basis for your new lines. This way you know that you'll get the unit, sampling_rate and the other parameters right.

6.2 Doubling of cloud fraction

For this example we will be changing the cld variable. This is the container for the cloud fraction for the entire atphosphere. This is done in the radiation_tend() function in radiation.F90 after the inital calls to the physics buffer (see section on code caveats). As an example we set the new variable to we twice that of the old one:

```
cld_sub = 2*cld ! arthurnoerve
```

6.3 Seasonal modification of cloud fraction

For this example we will give the cloud fraction variable seasonal dependence by multiplying it with a suitable sinusoid.

```
cld_sub = cld * ( 1 + 0.5*sin(calday/365.*2*3.14159) ) !
    arthurnoerve
```

6.4 Doubling the cloud fraction in the arctic

For this change we will only double the cloud fraction in the arctic. For this purpose we will treat the arctic as everything above 70 degrees North.

```
do i = 1, ncol
    if (clat(i) .ge. 70*3.14159/180.) then
        cld_sub = 2*cld ! arthurnoerve
    end if
end do
```

7 Appendix A: Using scp

This appendix is sligthly out of scope for this article but it will be included anyways due to its practicality. scp is a command line tool that enables you to copy files over ssh to another computer. This is very practical when working on a remote server/supercomputer that runs the model. Unless you're a serious terminal afficionado you will have problems efficiently searching, navigating and editing the code base, so graphical applications are prefered.

7.1 General scp syntax

The square brackets mark optional parts:

```
scp [-r] [other flags] <from_location> <to_location>
```

Where either <from_location> or <to_location> is a remote location

7.2 Downloading with scp

The relevant scp syntax looks like the following:

```
scp -r <username>@<host>:<remote_folder> <local_folder>
```

Where the -r flag tells scp to do recursive fetching; it goes through all files and subfolders. In my specific case the command looked like the following:

```
scp -r arthurnoerve@server.com>:~/CESM_model ~/CESM_model
```

7.3 Uploading with scp

Having downloaded the source, perused it and made your changes you have to upload the given files to the SourceMods folder. The nice thing about scp is that it can do uploads too. To upload the folder /source/ to my remote home directory I would do the following:

```
scp -r ~/source arthurnoerve@server.com>:~/
```

8 Appendix B: Editors and methods

Which editor to use is a highly personal choice (I use Atom) but there are some features that your editor should have:

1. Search and Replace
2. Search entire project
3. Fuzzy finder
4. Syntax highlighting

The search and replace is particularily handy for variable substitution, the entire project search allows you to find every line with a certain variable on it, great for finding declarations and finding the trace of the variable throught the program flow. Syntax highlighting is simply to make the code quicker to scan with the eyes; code does not have bold, italic, headings and paragraphs, the colours therfore steps in for these and makes the code easier on the eyes. Some tried and tested editors that many people use (love) are:

1. Sublime Text

2. Atom

3. BBEdit