Instructions for adding a new microphysics scheme to CM1

George H. Bryan

gbryan@ucar.edu

last modified: 23 October 2006

1 Introduction

Beginning with CM1 release 11 (cm1r11), there is a general method to add a new microphysics routine to the code. I have tried to make the process as painless as possible, but there are still a number of steps that a user is required to complete. The following instructions are divided into sections, based on the file that the user must modify.

Before proceeding, I note that the first step for the user is to create a *new* file, in the `src` directory, that contains the new microphysics subroutine. Currently, there are four microphysics schemes that are available in CM1. I summarize these schemes here, along with the appropriate setting for `ptype`:

Scheme	`ptype`	file name	Description
Kessler	1	`kessler.F`	liquid-only
Goddard LFO	2	`goddard.F`	LFO-type scheme, developed at NASA-Goddard
Thompson	3	`thompson.F`	Thompson microphysics scheme, developed at NCAR
GSR LFO	4	`lfoice.F`	LFO-type scheme, developed by Gilmore et al.

Users will probably find it useful to view the contents of these files to see how I interfaced these microphysics schemes with the model. Users should note that, due to historical reasons, Kessler and Goddard have been intertwined into the code in several complicated ways, mostly involving message passing for distributed memory (i.e., MPI) runs; basically, I tried to overlap computations with communications as much as possible. For the Thompson and GSR-LFO schemes, I did not attempt to maximize the message-passing performance; this does not mean that this scheme runs slowly, but it does mean that there is a potential to improve performance with more tuning. But, users that want to follow an existing section of code to implement their code should mimic the Thompson or GSR-LFO implementation, at least for a first try.

There are three more things for users to keep in mind as they prepare to incorporate their code into CM1:

1.1 sedimentation

There is a general sedimentation (fallout) scheme in cm1 that users are free to use. The subroutine is called `fallout`. The subroutine is at the end of the file `kessler.F`. It is simple to use: the user needs to pass the array containing the water species (`q3d`) and the array containing the terminal fall velocities (`vq`). (Note that, in the `solve` subroutine, I typically use the species' tendency array (`qten`) to store the fall velocities.) This subroutine will simply update the `q3d` array. However, the user is free to code up the sedimentation as part of the microphysics subroutine, which is probably necessary for some higher-order microphysics schemes, anyway. As an example, the `fallout` subroutine is used for the Kessler and Goddard schemes, but is not used for the Thompson scheme.

1.2 mass/energy conserving equations

It will be difficult for most model users to implement the "new equations" option (`neweqts`=1,2) that is described in the paper by Bryan and Fritsch (2002). This is too complicated to describe here, so I recommend that all model users simply implement their code for the `neweqts`=0 option, at least initially, which is a "traditional" cloud model equation set. Anybody that is interested in conserving mass or energy more accurately will need to figure out how to interface their code with `neweqts`=1,2. Contact George Bryan if you are interested in pursuing this option.

1.3 definition of saturation mixing ratios

I strongly recommend that you modify the functions `rslf` and `rsif` appropriately, if you are going to use a different set of definitions for saturation mixing ratios with respect to liquid (`rslf`) and ice (`rsif`). They are located in the `misclibs.F` file. I don't mind if you want to use a different set of definitions, compared to the ones that are currently used by default. However, if you do use a different set, then please add some code into these functions, so that other subroutines will see the atmosphere in a consistent way. It should be fairly straightforward to add the new definitions, and probably some `if` statements depending on what `ptype` scheme is being used. Again, if you have any questions, then just ask me.

OK, now it's time for the step-by-step instructions:

2 `Makefile`

This is easy. In the `Makefile`, insert an additional line to make sure that your subroutine is included in the compile. Piece of cake.

3 `input.incl` subroutine

The `input.incl` file is located in the `include` directory. The variable at the top of this file is called `maxq`; this specifies the maximum possible number of moisture arrays that you wish to add to the model. It is currently set to 100. If your microphysics scheme has more than 100 variables that need to be advected and diffused, then increase `maxq`. Otherwise, leave it set to 100.

4 `param` subroutine

The changes to this subroutine involve quite a bit of detail, but is not too complicated. In the `param` subroutine, find the section that says "Configuration for simulations with moisture." There, you will see the settings for four existing microphysics schemes. Between these four examples, you should be able to figure out what this all means. Most of the variables simply tell the code where the microphysical variables are. There is a specific method to the madness, and this needs to be done correctly.

First, and easiest, is the variable `iice`. Set this = 1 if your scheme includes ice microphysics.

Next is `numq`, which is the total number of moisture arrays that you need added to the model. Notice that the Kessler scheme has three arrays (mixing ratios for vapor, cloud drops, and rain drops), the Goddard scheme has six arrays (mixing ratios for vapor, cloud drops, rain drops, ice crystals, snow, and graupel), and the Thompson scheme has seven arrays (six mixing ratios ... the same as in the Goddard scheme ... plus one number concentration array). Note that you can have any number of arrays, and they can be mixing ratios and/or number concentrations (or, presumably, something else, like a third microphysical moment). There is one catch here: the liquid mixing ratios must be clumped together, the solid (i.e., ice) mixing ratios must be clumped together, and "everything else" (e.g., number concentration arrays) must be clumped together. This is needed so the code can easily compute total liquid water mass and total solid water mass.

I assume that only one water vapor array will ever be needed. With this in mind, you must tell the code where the liquid and solid and "other" arrays begin and end. Allow me to illustrate with a hypothetical example. Say your new microphysics scheme has 3 liquid categories, 6 solid categories, and 4 other things that are being advected and diffused (like number concentration). The following is a valid way to organize this mess:

array number	variable name
1	water vapor
2	liquid category 1
3	liquid category 2
4	liquid category 3
5	solid category 1
6	solid category 2
7	solid category 3
8	solid category 4
9	solid category 5
10	solid category 6
11	number concentration 1
12	number concentration 2
13	number concentration 3
14	number concentration 4

For this example, `iice`=1 and `numq`=14. Then, you would set the bounds for liquid (`nql1` and `nql2`), solid (`nqs1` and `nqs2`), and "others" (`nnc1` and `nnc2`) as follows: `nql1`=2, `nql2`=4, `nqs1`=5, `nqs2`=10, `nnc1`=11, `nnc2`=14. It is not necessary to list vapor first, then all liquids, then all solids, then all number concentrations; actually, you can use any order that you wish, but you must "clump" similar types together. This is necessary to tell the dynamical solver where everything is located.

There are a few extra variables to set, which will help the model solver, and other diagnostic codes. The variable `nqv` tells the model which array holds the water vapor. In the example above, `nqv`=1.

Next, you need to tell the model what arrays define the cloud edge. This is needed for some diagnostic code, and is needed for the calculation of moist Brunt-Vaisala frequency in the turbulence code. This is accomplished with the `cloudvar` array, which is a logical variable that tells the model whether this array defines whether a grid point is "in cloud." As an example, notice that for the Goddard scheme I use the cloud droplet (array number 2) and ice crystal (array number 4) variables to define cloud edge).

Next, using the text array `qname`, you can specify a name of each variable. The array `qname` specifies a three-character name for each moisture variable. Examples are included in this file for the three existing microphysics schemes. Note that the existing names are very traditional for single moment scheme (qv, qc, qr, etc.). You can name them anything that you wish; this is only relevant for output purposes.

Next, specify a value for the variable `nbudget`. This is the number of "budget" variables that the users wishes to calculate and print out, and includes things like total condensation, total rainwater evaporation, total rainfall, etc. If you do not want to do this, then just set `nbudget`=1. The next variable that you need to define is a character array, `budname`, which specifies a six-character name for the budget variables. Note that the Kessler scheme is currently coded to output six budget variables: total condensation, total cloudwater evaporation, total autoconversion, total accretion, total rainwater evaporation, and total rainfall. The actual budget numbers are stored in the double precision array `qbudget`. You can look through the model code to see how I have calculated these variables for the Kessler and Goddard schemes.

Some diagnostic routines in the cm1 code requires a total rainfall variable. So, I have declared another variable, `budrain`, that tells the model where to find total rainfall. In the Kessler example, this is budget variable number 6.

Finally, there is one last piece of code that may need to be modified in `param`. If your microphysics scheme needs to be "initialized," then you can add the call to an appropriate subroutine here in `param`. For example, the Goddard scheme calls `consat`, and the Thompson scheme calls `thompson_init`; in contrast, the Kessler scheme does not need to be initialized.

5 `solve` subroutine

Finally, you need to interface your microphysics subroutine with the model's solver. First, I should say that if you setup everything correctly inside `param`, then every array should be advected and diffused, as well as updated in time, and all boundary condition and distributed-memory message passing should be handled correctly. The only additional trick to make message passing work properly is to set `simple_comm = .true.`; see, e.g., the call to the Thompson scheme within `solve`. So, don't worry about all the boundary condition, message passing, advection, and diffusion stuff; proceed to the section of the `solve` subroutine that says "Explicit microphysics." Insert a new section of code for a new value of `ptype`; values 1, 2, 3, and 4 are already taken (for Kessler, Goddard, Thompson, and GSR-LFO, respectively). Add a call to your subroutine where it says, "insert new microphysics schemes here."

If you would like to use it, there is a generic subroutine that will set any negative values of the `q3d` arrays to zero. It is called `pdefq`. See existing examples (e.g., the Thompson scheme) to see how this works. The first value passed into this subroutine is a real variable, which the subroutine uses as a threshold; anything less than this value is set to zero. For example, for the Goddard scheme, very small values ($\texttt{q} < 1 \times 10^{-14}$), the code sets them to zero; mass conservation may be applicable (see `pdefq` subroutine for more detail). This is used to speedup the code, and to eliminate artificial negative values (if applicable). You do not need to use the pdefq arrays; this is completely optional.

If you have any problems interfacing your subroutine to `solve`, then give George a call. It should be fairly straightforward if you follow what is done for the other microphysics schemes.

6 Contact information

For questions, contact George Bryan: `gbryan@ucar.edu`, 303-497-8989.