May 2020

Modeling of Cloud Droplet Formation: Software Development and Sampling Strategies

Niklas Selke
University of Wisconsin-Milwaukee

Modeling of Cloud Droplet Formation: Software Development and Sampling Strategies

by

Niklas Selke

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Mathematics

at

The University of Wisconsin–Milwaukee

May 2020

Abstract

Modeling of Cloud Droplet Formation: Software Development and Sampling Strategies

by

Niklas Selke

The University of Wisconsin–Milwaukee, 2020
Under the Supervision of Professor Vincent E. Larson

Updraft speeds are an important factor in the formation of cloud droplets which play an important role in an atmospheric simulation. The updraft speeds are varying very strongly in small areas of space. Current models do not account for this kind of variability. Support for a probability density function (PDF) based approach in representing the variability of the updraft speeds has been implemented in the Energy Exascale Earth System Model (E3SM). Specifics of the implementation process have been discussed.

Different sampling strategies were tested to analyze the convergence behavior of the new approach to the cloud droplet formation process. It has shown that using a simple Monte Carlo sampling strategy has given good convergence rates for the different variables that are important to the cloud droplet formation process.

To my family

Table of Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank Professor Larson for being my adviser on this thesis and for him giving me the chance to work in his research group for the past year as a research assistant. In this time I have learned so many new and interesting things. I am very thankful for all the support he has given me while working for him and working on my thesis.

Furthermore, I would like to thank Professor Lauko and Professor Spade for serving as members of my thesis committee.

I would also like to thank Professor Dikta at Fachhochschule Aachen in Germany for offering me the opportunity to study abroad for a year and all his support in preparing for this year abroad previous to my departure from Germany.

Chapter 1

Introduction

There are many different atmospheric models used in the scientific community to create the best possible simulation of the atmosphere and its implications for the climate. These models are very complex as there are so many different aspects and processes happening all at once in the atmosphere.

1.1 Atmospheric Models

The models used in atmospheric science are based upon a wide variety of differential equations to describe various processes happening in the atmosphere. To solve these equations using numerical methods the calculations are discretized on a grid and evaluated on the evenly spaced grid points. The grid has a fixed number of grid columns each describing a part of the whole area that is simulated. In every grid column there are a number of vertical levels to get a three dimensional discretization of the simulated area. Each of the vertical levels has its own set of values for different atmospheric components like temperature or pressure.

To start a simulation, an initial state the atmosphere is in has to be provided. This initial state has information on all the atmospheric components with which the simulation will start to operate. These can either be modeled data created by researchers or come from measurement campaigns and are therefore actual conditions that were recorded at some point in time. These sets of initial data are called cases. Depending on the case, simulations will produce very different results. Atmospheric models usually also come with a large set of options for different physical processes that can be adjusted in order to change the behavior of them or to save simulation time. All these possible settings allow for a wide variety of simulations that can be done with the same initial state which makes atmospheric models very versatile.

1.2 Motivation

An important factor for the climate is cloud coverage. The clouds reflect solar radiation, which cools the ground below them. The amount of reflection depends on the concentration and size of cloud droplets (Boucher et al., 2013). Consider two clouds with the same cloud water content (grams of liquid per gram of dry air), but one cloud has more numerous but smaller droplets while the other cloud has fewer but larger droplets. Calculations show that the cloud with more numerous but smaller droplets reflects more solar radiation (Lohmann and Feichter, 2005). Therefore, it is important that atmospheric models calculate the droplet size accurately.

The droplet size is in part determined by droplet activation (i.e. nucleation). Droplet activation is the formation of a new droplet by condensation. In the atmosphere droplet activation requires the presence of a small particle (i.e. aerosol) on which liquid can condense. More aerosol in the air leads to more small droplets and therefore more reflection.

A complicating factor in the activation process is the updraft speed. The higher the updraft speed, the more droplets are activated. This happens due to the air being cooled down more rapidly with higher updraft speeds. The problem is that the updraft speed is hard to represent in a model because the updraft width is approximately a kilometer but a typical climate model grid box might have a width of 100 kilometers (Malavelle et al., 2014). Therefore, the updrafts are not resolved accurately by the numerical grid and potentially misrepresented on the grid.

Currently climate models just feed one representative value (the standard deviation of updraft speeds) into the droplet activation formula and take the result as a representative for the whole grid box. This is potentially not very representative e.g. for a skewed distribution function.

In order to be able to get a better representation of the highly diverse updrafts, many values for updraft speeds have to be taken inside one grid box. The process of doing so consists of the following four steps.

1. Calculate probability density function (`PDF`)
2. Draw samples from `PDF`
3. Feed samples one by one into droplet activation formula
4. Average the result from each sample back to the grid scale

The `PDF` is computed in `CLUBB`[1]. The samples are then drawn using `SILHS`[2] which comes with a lot of different configurable parameters to change the sampling strategy. The profiles of the chosen samples at each grid level are the so-called subcolumns (Larson, 2017). `SILHS` does rely on an external module to provide a `PDF` from which it draws samples, in this case `CLUBB`.

The third step presented a problem as the current implementation of the droplet activation formula does not allow for samples to be fed into the calculation and therefore also does not allow to follow through with step four.

In this thesis I will analyze the convergence of the simulation results when using a sampled droplet activation process. To be able to do that I will first describe the implementation of a version of the droplet activation process which is capable of using a sampled input. Then I will discuss certain challenges and possible refactorings that I found during implementing the necessary changes. Finally, I will look at the convergence in the simulation results with different sampling strategies.

[1]Cloud Layers Unified By Binomials (Larson, 2017)
[2]Subgrid Importance Latin Hypercube Sampler (Larson and Schanen, 2013)

For this thesis I am using the Energy Exascale Earth System Model (`E3SM`) (E3SM Project, 2018) which is a very versatile model. I am concentrating on the atmosphere component of that model as this is the component in which the droplet activation process is located.

Chapter 2

Implementation

In this chapter I will describe the implementation of a droplet activation process which is capable of using a sampled input. For this, I will start by describing the current implementation of the droplet activation process in section 2.1. Then, I will go over the necessary code changes for the droplet activation process to work with subcolumns in section 2.2. Finally, I will give some notes on the refactoring process in section 2.3 which could be useful for potential future changes to other physical processes used in the model to also be able to use subcolumns.

The `E3SM` model used for this thesis is written in the Fortran programming language, in particular the Fortran 90 coding standard (Fortran 90 Standard, 1991). It consists of many different components to model different aspects of the climate. The droplet activation process is located in the atmosphere component.

2.1 Current Implementation

Involved with the droplet activation process are four different subroutines, namely `tphysbc` located in the module `physpkg`, `microp_aero_run` located in the module `microp_aero`, `nucleate_ice_cam_calc` located in the module `nucleate_ice_cam` and `dropmixnuc` located in the module `ndrop`. Figure 2.1 shows the structure of the droplet activation process with the relationships between the mentioned subroutines.

The `tphysbc` subroutine is responsible for calling the different physical processes used in the model and as such is also calling the droplet activation process. After the called process is finished this subroutine gets back tendencies for the atmospheric components effected by

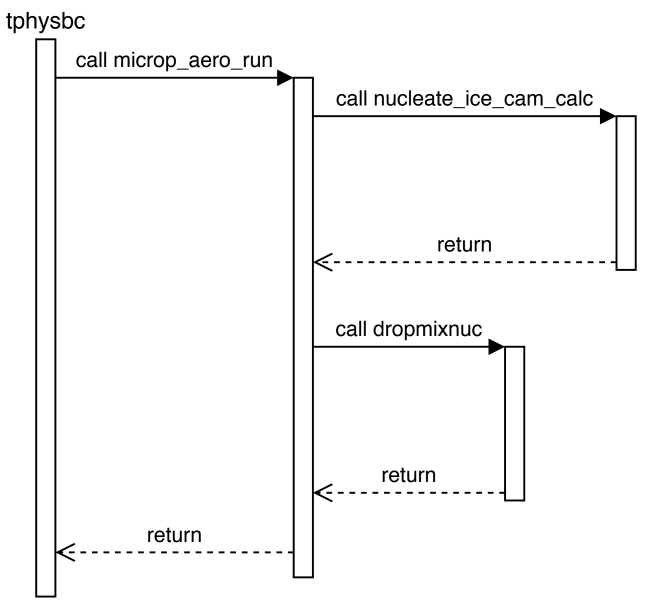

Figure 2.1: Call structure of the droplet activation process

the previously called physical process to update the current atmospheric state.

The `microp_aero_run` subroutine is the main driver of the droplet activation process. It is responsible for calculating the new cloud fractions, calling the ice nucleation process and the droplet activation process itself.

The `nucleate_ice_cam_calc` subroutine calculates the nucleation process on ice particles. It returns the number of activated aerosol for the ice nucleation.

The `dropmixnuc` subroutine takes the calculated cloud fractions and then calculates the vertical diffusion and nucleation of cloud droplets. It returns the change of the droplet number concentration.

There are a few important structures used throughout the whole simulation process. The first structure that is relied on heavily is the `state` variable. It is a Fortran derived type holding all the information about the current state of the atmosphere. That includes information about position on earth, temperature, pressure and wind at the corresponding grid box. It also holds information about the setup of the simulation itself. That is information about the number of active grid columns, the maximum number of columns (i.e. product of grid columns and number of subcolumns) and the total number of active columns. Active columns are columns that actually contain data. If every column is an active column the number of active columns equals the number of columns on the grid and the total number of active columns equals the product of grid columns and subcolumns per grid column. Table 2.1 gives an overview on the different variables that hold information about the columns used.

Variable Name	**Description**
`pcols`	Maximum number of grid columns
`psubcols`	Maximum number of subcolumns
`psetcols`	Maximum number of all columns
`ngrdcol`	Number of active grid columns
`ncol`	Number of all active columns

Table 2.1: Showing an overview on the components of the `state` variable which hold the information about the grid in terms of the used columns (Thayer-Calder et al., 2015).

The `state` variable is fed into each of the subroutines listed above as all of them need some information about the current atmospheric state.

The other important structure is the `pbuf` (physics buffer) variable. It holds a lot of variables that are used throughout the subroutines. Different subroutines read the variables while others calculate and write them to the `pbuf` variable. So, as the `state` variable, the `pbuf` variable is also fed into all of the subroutines mentioned above. The `pbuf` variable has the capability of holding values for each of the grid columns as well as holding values for each of the subcolumns by reserving memory for two separate arrays, one for the grid columns and one for the subcolumns. When a variable is requested from the physics buffer, a pointer is returned to the array that has the requested type, either grid values or subcolumn values.

As illustrated in Figure 2.2, the data in the subcolumn array can either be `duplicated` from the grid, meaning the value of the grid column is copied to each of the subcolumns associated with this grid column, or the data in the subcolumn array is `distributed`, meaning the value of each of the subcolumns is not necessarily equal to the value of the corresponding grid column or the other subcolumns associated with the same grid column. Data is

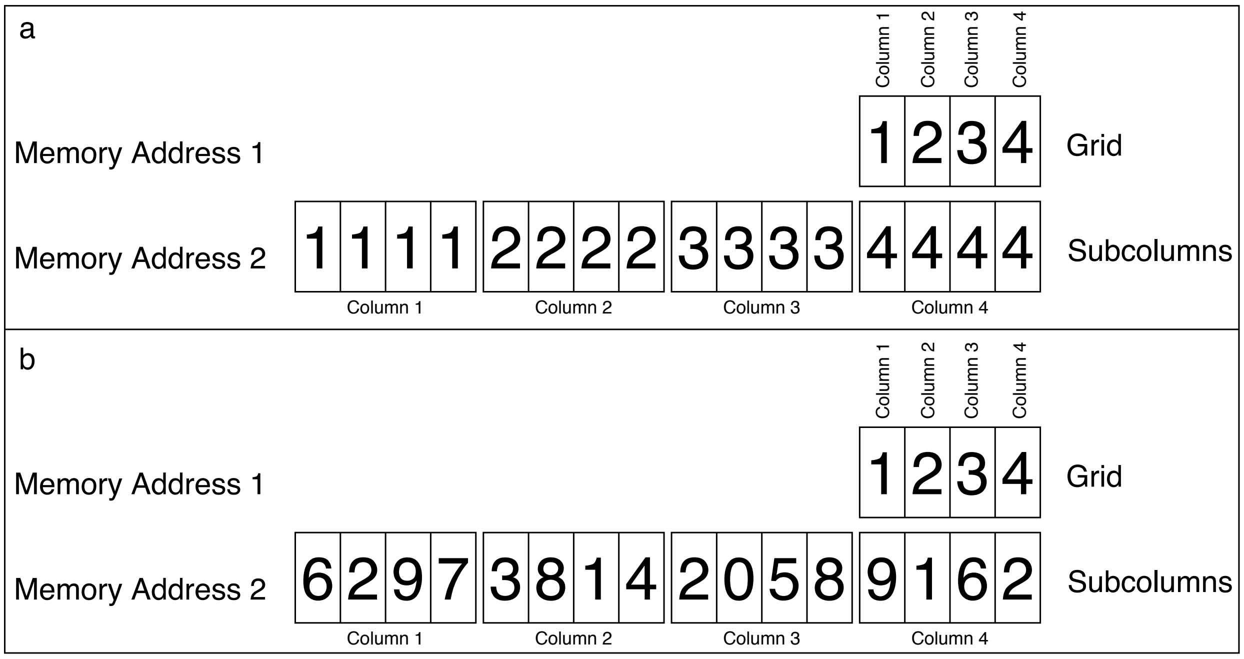

Figure 2.2: Memory layout of the physics buffer. In this example there are four grid columns each having four subcolumns. Part `a` shows a variable that has been `duplicated` on the subcolumns. Part `b` shows a variable that has been `distributed` on the subcolumns.

`duplicated` for variables that are only calculated on the grid in some component of the simulation and are therefore not equipped with values for subcolumns. Data is `distributed` for variables that are sampled and therefore have individual values for each of the subcolumns.

Important to keep in mind is that the two different arrays with values in the memory are independet of one another. If the pointer is pointing to the subcolumn version of the variable (memory address 2, see Figure 2.2) and updates it, the grid version is not automatically updated as well and the other way around. In the current code only the grid versions of variables (memory address 1, see Figure 2.2) are used in the droplet activation process.

2.2 Code Changes to Subcolumnize the Droplet Activation Process

In order to be able to use subcolumns in the droplet activation process, all the involved subroutines had to be checked for necessary changes. Moreover, all the helper routines that are called, needed to be checked as well in order to get the code working correctly.

First, I introduced three new options in order to control the behavior of the subcolumn usage. These options with their possible values are shown in Table 2.2.

Option Name	Possible Values	
`use_subcol_aero`	.true.	.false.
`subcol_aero_mode`	interactive	non-interactive
`subcol_aero_copy_state`	.true.	.false.

Table 2.2: Options used to control the behavior of the subcolumn usage. They are shown with their possible values.

The `use_subcol_aero` option switches the use of subcolumns in the droplet activation process on or off. Only if it is set to `.true.`, the code changes I will describe next are going to be needed as the current code is not using subcolumns in the droplet activation process. The other two options are only relevant when `use_subcol_aero` is `.true.`.

The `subcol_aero_mode` option controls the behavior of how the subcolumn calculations are used. If it is set to `interactive` all the calculations done on the subcolumns are being averaged back to the grid and will update the `state` variable and the `pbuf` variable. If it is set to `non-interactive` all the calculations are first done on the grid as it is done without the code changes and then again on the subcolumns for each timestep. The results of the subcolumn calculations are written to disk but are otherwise not used. The `state` variable and the `pbuf` variable are getting updated with the calculations that were done directly on the grid. This allows for better convergence analysis as the noise in the sampled data is not fed back to the atmospheric state each timestep which keeps the atmospheric state on the right track.

The `subcol_aero_copy_state` option controls how the `state` variable is populated with data on the subcolumns. The normal behavior is that `SILHS` samples values for the components of the `state` variable that should be `distributed` on the subcolumns which will then lead to different values in each of the subcolumns. The other components of the `state` variable are `duplicated`. As a diagnostic tool, this option, if set to `.true.`, overrides the normal behavior for `distributed` components of the `state` variable by simply copying the current grid values to each of the subcolumns, essentially treating them like `duplicated` variables. This leads to the same calculations being done in each of the subcolumns and these calculations are the same that would have been done on the grid itself. When averaging the subcolumn results back to the grid they will have the same result as the calculations done on the grid without the use of subcolumns, with the exception of roundoff level error, if the code changes are working properly.

The advantage of having the `state` variable already used in all relevant subroutines, is that it, as mentioned in section 2.1, holds information over the maximum number of columns. This means that, if subcolumns are used or not, the `psetcols` variable which is a member of the `state` variable holds the maximum length needed to store values for calculations on either the grid or the subcolumns. Therefore, the first step in changing the code is to initialize all local variables in the involved subroutines with the value of `psetcols`.

The fields in the `state` variable are correctly allocated using `SILHS`. If subcolumns are used, `SILHS` samples values to the subcolumns as described before in connection with the `subcol_aero_copy_state` option leading to correctly allocated arrays inside the `state` variable. If no subcolumns are used the `state` variable is staying as it is implemented now.

For the variables saved in the physics buffer, the ones that are needed in the droplet activation process are calculated in parts of the code that is not using subcolumns. This means that only the grid length array is filled with values while the subcolumn array is not allocated at all. However, the physics buffer comes with an option when retrieving arrays from it which, if set to `.true.`, is going to allocate memory and copy the grid values down to the subcolumns, resulting in a `duplicated` variable, and then ultimately will return a pointer to this newly created array.

As described in section 2.1, the physics buffer grid and subcolumn arrays are independet from one another. If subcolumns are used and the variable is copied from the grid, the pointer eventually ends up pointing to the newly allocated array with subcolumn length. When one of these variables is updated in the droplet activation process it has to be averaged back by first retrieving an extra pointer to the grid length array and then writing the averaged value back to that memory address. This is important because other processes after the droplet activation process might not use subcolumns and will just request the grid length array. That is why it has to be up to date as well even though, for the droplet activation process itself, it is not relevant as these arrays are not used when subcolumns are used.

In terms of helper subroutines only two had to be slightly changed. Both of them are part of the `rad_constituents` module, namely the subroutine `rad_cnst_get_mode_num` and the subroutine `rad_cnst_get_mam_mmr_by_idx`. Both routines are used to return pointers to certain information about different aerosol types. The subroutine `rad_cnst_get_mode_num` gives back a pointer to the number mixing ratio for the requested aerosol type and the subroutine `rad_cnst_get_mam_mmr_by_idx` gives back a pointer to the mass mixing ratio information for the requested aerosol type. Depending on the requested aerosol type, the information for number and mass mixing ratios are either contained in the `state` variable or saved in the `pbuf` variable. This is not a problem if no subcolumns are used. However, if subcolumns are used and the information is stored in the `state` variable then the array pointed to is of subcolumn length, but if the information is stored in the `pbuf` variable the pointer will always point to the grid length array in the physics buffer. This led to the same subroutine delivering arrays of different length depending on where the requested information was saved. So, changes had to be made for the aerosol types that are stored in the `pbuf` variable. As the information about number mixing ratio and mass mixing ratio is not changed in the droplet activation process it is only available on the grid. Therefore, the information needs to be copied from the grid down to the subcolumns as seen in Figure 2.2 part `a` and then a pointer needs to be returned to the copied version which has subcolumn length.

2.3 Notes on the Code Refactoring

In this section I am going to talk about the challenges of the code refactoring while also listing some aspects of the refactoring that are worth keeping in mind for potential future changes to other components of the simulation to work with subcolumns.

2.3.1 Notes on Necessary Array Lengths

The first important thing to note is that there was already an implementation at least for the `state` variable to work with subcolumns and that this variable does contain all the information needed to determine the correct length of the arrays for the currently used grid with or without subcolumns. Table 2.1 gives an overview on the relevant components of the `state` variable while they are described in detail in `Appendix A` of Thayer-Calder et al. (2015). With these information it can be determined if subcolumns are used or not, depending on the values of `pcols` and `psetcols`. If no subcolumns are used `pcols` and `psetcols` will equal. On the other hand, if subcolumns are used `psetcols` will be equal to $pcols \cdot psubcols$. So, it is possible to figure out if subcolumns are used by simply comparing the values of `pcols` and `psetcols`.

2.3.2 Notes on Active Columns

Regarding the `ncol` variable and the `psetcols` variable it is worth noting that these two will equal each other if every column, including subcolumns, is active. This holds for a grid with and without subcolumns. Local variables are allocated with the value of `psetcols` as this variable describes the maximum number of possibly active columns. The calculations are done only on the number of actually used columns, i.e. active columns, so the `ncol` value will be used for loops in the subroutines themselves. This allows for setups with all columns being active to work as well as setups with some inactive columns without needing to do any changes to the code depending on the setup.

2.3.3 Notes on Mismatching Indices

One early thing I noticed was that as soon as a `state` variable that had values on subcolumns was plugged into the droplet activation process the process was not able to finish without getting runtime errors. That happend due to a mismatch in array lengths. All loops over the number of used grid columns used the value of the `ncol` variable as the upper index which does describe the number of active columns, thus including the active subcolumns. This meaned that the loops ran over too many indices for all the variables that were only accessed on the grid. That mismatch could have been avoided by simply running loops up to the value of the `ngrdcol` variable which has the information on the number of active grid columns. That would have made the loop indexing inside the subroutines consistent with the hardcoded array lengths of all the local variables that were used. This remark is mainly included to clearly state the difference between `ngrdcol` and `ncol`. For the refactoring it is going to work to always use the value of the `ncol` variable for the loops and allocate local arrays with the length given by the variable `psetcols`.

2.3.4 Notes on Aerosol Types

Another thing worse noting that I found during the code refactoring was the use and handling of different aerosol types and their number mixing ratios and mass mixing ratios. Depending on what type of aerosol is requested, the information about the ratios are either found in the `state` variable or in the `pbuf` variable. As described in the last paragraph of section 2.2, that led to problems with pointers pointing towards different arrays in the memory which had different lengths if no changes were made to the helper subroutines. Furthermore, it was not possible to know outside of the helper subroutine if the pointer returned is pointing to an array belonging to the `state` variable or an array in the `pbuf` variable. A possible future change could be made to create a new structure that will hold all the information on the different aerosol types on its own. This would make the interface to interact with them more easily to work with subcolumns. It would also remove some of the components from the `state` variable which already holds a lot of different information and could therefore be easily reduced on these components.

2.3.5 Notes on Vertical Levels

On a last note in this section I wanted to go over the vertical levels used in each of the grid columns. Having vertical levels in each grid column gives a three-dimensional model. However, some precaution has to be taken when dealing with them. These comments are regarding both the grid with and without subcolumns in the same way. The variables are usually allocated with the value of the variable `pver` which describes the number of vertical levels. In many different processes throughout the simulation a so-called top level is specified and saved in the `top_lev` variable. This also happens in the droplet activation process. Calculations are then done only up until the value of the `top_lev` variable. This leads to uninitialized array elements for the few vertical levels above the top level. For the simulation and the calculations in it that is not a problem because if an upper level is specified it is used consistently throughout so that calculations are never including any uninitialized memory addresses. A problem arises when outputting some of the variables that have uninitialzed elements. Depending on what is currently saved at the memory address it can potentially be uninterpretable for the software component that handles outputs to disk which results in runtime errors. This can happen for different variables each simulation run as it is possible that different memory addresses are used for different runs even without changing the setup in any way. To avoid this, variables should be initialized with a fixed value over all vertical levels before they are first used, ensuring that there will be no problem when outputting them. This is done for many variables but not all of them. This is easy to do and does not require any other change than one line of code initializing the variable but it is something that can easily be forgotten especially because it does not always result in a runtime error as the content of the uninitialized memory address could potentially be interpretable as a number.

Chapter 3

Setup for the Analysis

In this chapter I will go over the setup used for the simulations that are analyzed in chapter 4. I will start with describing the case setup that I used for all the simulations in section 3.1. Next, I will describe the plots I will going to use for analyzing the convergence in section 3.2. Finally, I will describe the general workings of the sampling process in section 3.3 before going into detail with specific sampling strategies in chapter 4 together with the convergence analysis.

3.1 Case Setup

For the simulations, that I will analyze in chapter 4, I used the `BOMEX` (Barbados Oceanographic and Meteorological Experiment) case (Holland and Rasmusson, 1973; Siebesma et al., 2003). It is a case which data was recorded east of Barbados in 1969. I am running a single column model with 72 vertical levels in the grid column. The simulation simulates a period of six hours. The time step size is 30 minutes. The only physical process that uses subcolumns is the droplet activation process. I use the `non-interactive` mode in order to simplify the analysis of the convergence in chapter 4. I am using up to 10000 subcolumns for the analysis of the convergence.

The variable w describes the vertical velocity (i.e. updraft/downdraft speed). As described in section 1.2, this is the variable that will be `distributed` on the subcolumns.

3.2 Plots

In order to see if the sampling is working correctly, I will look at the plot of *wp2* which is the variance of the w variable. The more subcolumns used the better the variance should match the original variance of the underlying `PDF`.

The other plot I will look at is the plot of the variable *rcm* which is the cloud water mixing ratio. This value is dependent on the outcome of the droplet activation process. If the subcolumnization and the sampling are working properly it should converge to the value of *rcm* which is computed in `CLUBB`.

The plots will have the vertical levels on the vertical axis and the corresponding values on the horizontal axis, as this is the convention in atmospheric science. The reference values for *wp2* and *rcm* will be drawn with a black line while the different sampled output lines will rotate through an array of colors to distinguish the sampled lines and the reference line.

3.3 Sampling Process

`CLUBB` is used to construct a multivariate `PDF` that includes the vertical velocity w. The `PDF` chosen is a double Gaussian (Larson et al., 2005). The concrete `PDF` used in each grid box and time step is determined by the grid box means and some higher order moments which are advanced in time (Golaz et al., 2002). `SILHS` will then draw samples from the offered `PDF`. This is done by choosing a starting grid level and generating a correlated, multivariate sample at that level (Larson and Schanen, 2013). To create a profile over all vertical levels `SILHS` chooses similar values for adjacent levels which means that there is a high vertical correlation between adjacent vertical levels which will decrease with the distance (Larson and Schanen, 2013).

`SILHS` has two possible ways to reduce noise in the samples. Both `Stratified Sampling`, namely `Latin Hypercube Sampling`, and `Importance Sampling` can be used within `SILHS` (Larson and Schanen, 2013). In chapter 4 I will describe the different sampling strategies in detail when looking at the convergence for these methods.

The droplet activation process itself does not need to know that the profiles, that are given as input to it, are not grid box averages but instead subcolumns. The calculations are done in the same way. The only thing is that the droplet activation process must allow feeding in more profiles, as described in section 2.2.

Chapter 4

Convergence Analysis of Different Sampling Strategies

In this chapter I will be discussing different sampling strategies and analyse their convergence behavior. The general simulation setup was discussed in the previous chapter in section 3.1.

4.1 Monte Carlo Sampling

The first sampling strategy I will discuss is the Monte Carlo sampling without any special treatment. Hereafter, I will call this strategy `Straight Monte Carlo Sampling`. With this strategy the samples are picked completely relying on some pseudorandom number generator. More information about Monte Carlo sampling in general can be found in Gentle (2003).

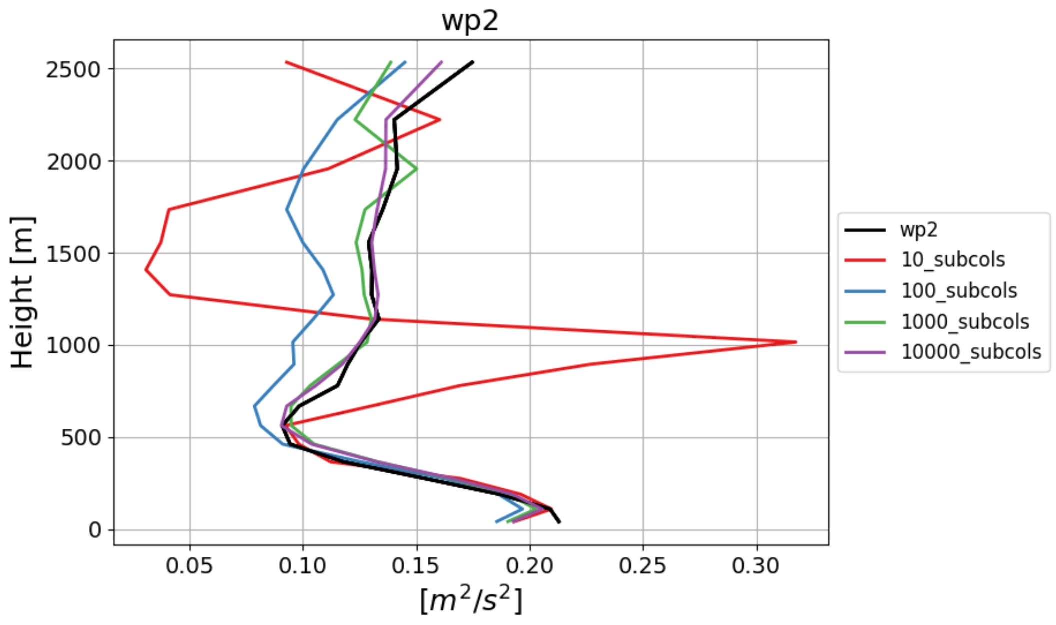

Figure 4.1: Convergence of *wp2* for the `Straight Monte Carlo Sampling`

Figure 4.1 shows that the `Straight Monte Carlo Sampling` is converging for *wp2* but this is happening very slowly. With 10 subcolumns the variance of the vertical velocity w is not following the underlying variance of the `PDF` from which was sampled. With 100 subcolumns the general course of the variance is followed but only with 1000 subcolumns the original *wp2* line is closely followed with the exception of the higher altitudes from around 1800 meters upwards. Only with 10000 subcolumns it looks like *wp2* is really converged.

This result is not surprising as there is no strategy in place to make sure that most of the samples drawn can be put to good use in terms of the convergence.

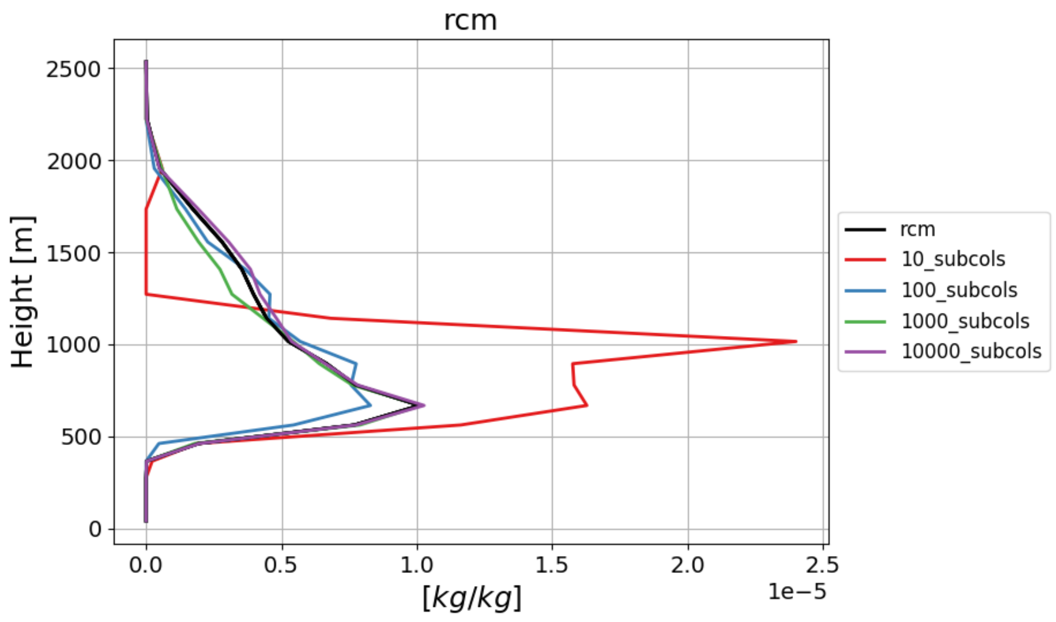

Figure 4.2: Convergence of *rcm* for the `Straight Monte Carlo Sampling`

Figure 4.2 shows that the `Straight Monte Carlo Sampling` is converging for *rcm* but like before with *wp2* it is converging slowly. With 10 subcolumns the cloud water mixing ratio does not follow the course of the grid column version of *rcm*. With 100 subcolumns the overall course of the graph is matched quite good already with the exception of the peak at around 600 meters. With 1000 subcolumns there is a pretty good match to the reference *rcm* line. There are only some slight deviations from around 1200 meters to 1800 meters.

With 10000 subcolumns the line is for the most vertical levels on top of the reference line.

So, even without trying to do anything to improve the convergence rate the convergence for *rcm* is already not too bad.

4.2 Latin Hypercube Sampling

The next sampling strategy I am going to look at is the `Latin Hypercube Sampling`. It does start out by dividing the space covered by the `PDF` into squares, the so-called `hypercubes`. Each of these has equal probability. Then one of these squares is chosen at random and a point inside this square is then randomly chosen. After that the row and column in which the chosen square is, are eliminated from the grid. Then the process starts over again by choosing the next square at random (Larson et al., 2005). This helps spreading the points more evenly across the space that is covered by the `PDF`.

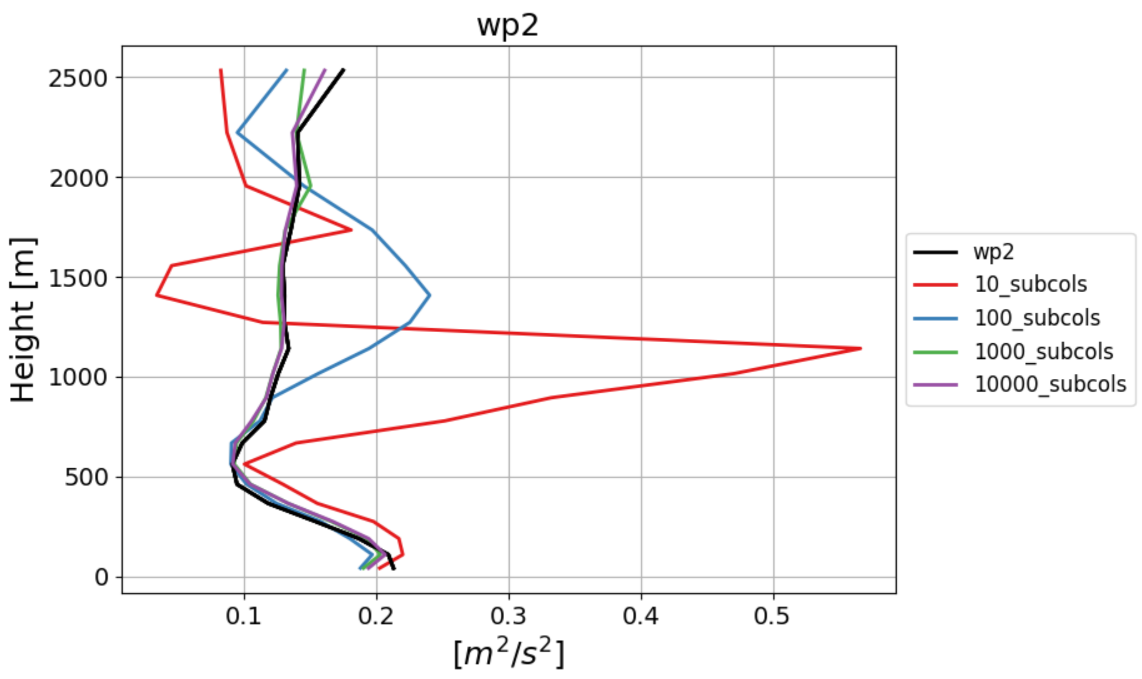

Figure 4.3: Convergence of *wp2* for the `Latin Hypercube Sampling`

Figure 4.3 shows that the `Latin Hypercube Sampling` does give a better convergence rate than with the `Straight Monte Carlo Sampling`. Between 10 and 100 subcolumns there is some improvement in taking the course of the original values of *wp2*. With 1000 subcolumns the process is mostly converged to the reference line. Using 10000 subcolumns does not improve the outcome that much over using 1000 subcolumns.

Using the `Latin Hypercube Sampling` does help the vertical velocity because it is varying very strongly inside one grid box as described in section 1.2. By stretching out the samples it covers more of the many different values and therefore the sample variance is converging faster to the convergence of the underlying `PDF`.

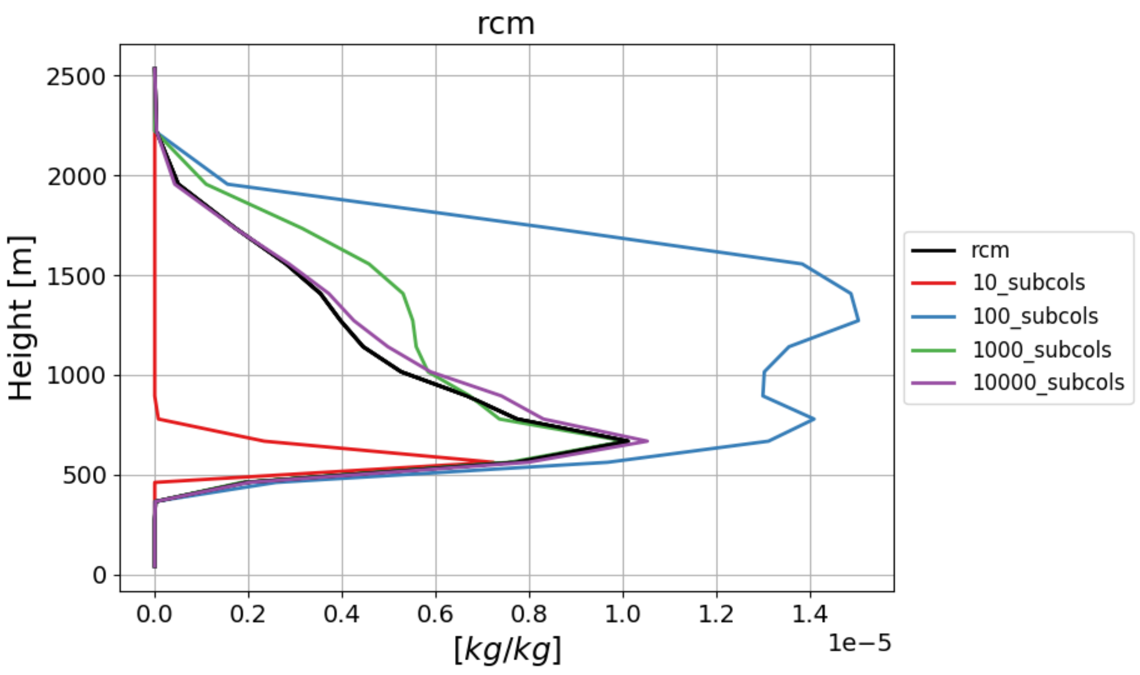

Figure 4.4: Convergence of *rcm* for the `Latin Hypercube Sampling`

Figure 4.4 shows that the `Latin Hypercube Sampling` did slightly worsen the convergence rate of the cloud water mixing ratio in comparison to the `Straight Monte Carlo Sampling`. For 10 subcolumns as well as for 100 subcolumns the lines are far off the reference line. With 1000 subcolumns it does match the reference line at the lower vertical levels.

Using 10000 subcolumns does improve the fit of the cloud water mixing ratio specifically in the upper vertical levels.

The *rcm* plot does still show convergence but the `Latin Hypercube Sampling` did not improve the convergence rate. This is most probably because the drawn samples are now pushed away from each other. The value of *rcm* is basically only dependent on the values in cloud as there is more water than in the surrounding air outside the cloud. Spreading the samples puts many of them out of cloud where the water ratio is significantly smaller than in cloud. This leads to many sample points underrepresenting the actual value of the water ratio in the grid box.

The slightly worse results for 100 subcolumns in comparison to the `Straight Monte Carlo Sampling` could come from the fact that for both strategies only one simulation was done. So, for the `Latin Hypercube Sampling` the subcolumns could just have been sampled unfortunately.

Concluding, the `Latin Hypercube Sampling` has shown that spreading the sampled points more evenly across the space covered by the `PDF` is not necessarly a way to improve convergence. For a variable like the vertical velocity w that is a good approach. On the other hand, for a variable like the cloud water mixing ratio *rcm* that does more harm than good for the convergence rate.

4.3 Importance Sampling

For the next sampling strategy I am going to look at the `Importance Sampling`. For this strategy the samples are picked in a way that there is an equal amount of points inside and outside of liquid clouds as long as the liquid cloud fraction in the grid box is lower than 0.5. Thus, the regions inside of liquid clouds are deemed the important regions. This is done because regions with liquid are often the ones with the most variability (Larson and Schanen, 2013).

One very important thing to remember here is that the samples are drawn on one specific vertical level. The liquid cloud fraction from this one level is used for the `Importance Sampling`. For the other levels the points are chosen in a way that the points decorrelate exponentially with larger distances between the vertical levels (Larson and Schanen, 2013).

The level that is used as the starting level is the one with the greatest liquid water mixing ratio (Larson and Schanen, 2013). This level can easily being identified from the plots in this thesis as it is the level where the value of rcm, the cloud water mixing ratio, has the maximum value. That is at around 650 meters.

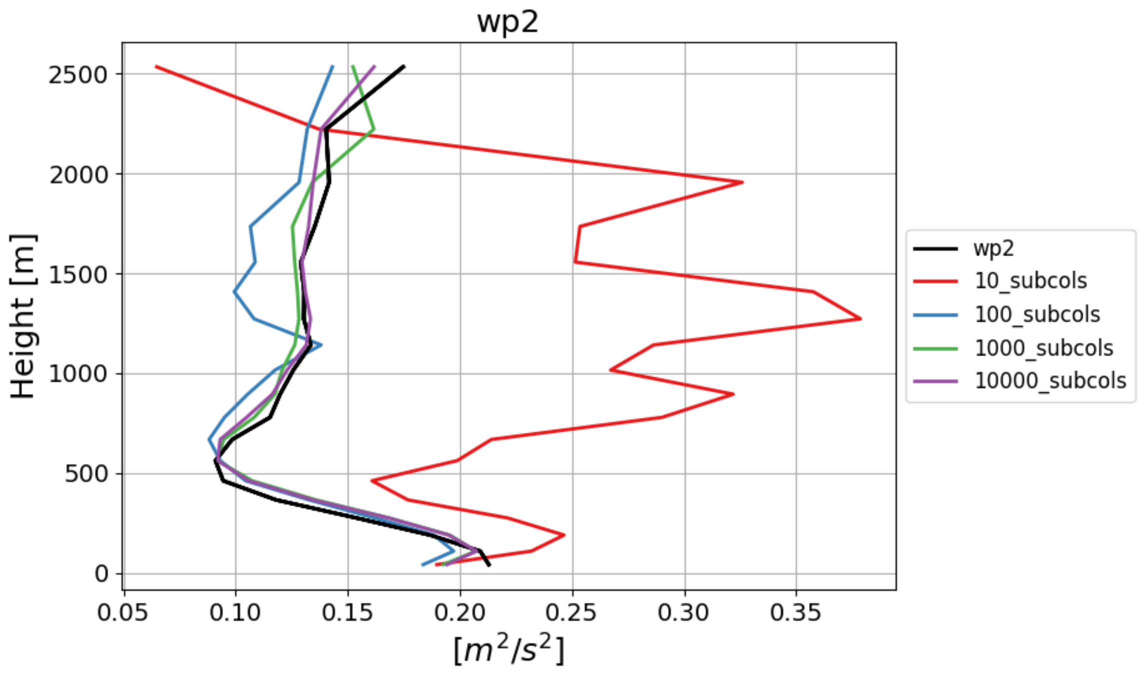

Figure 4.5: Convergence of *wp2* for the `Importance Sampling`

Figure 4.5 shows that for 10 subcolumns there is not really any sort of resemblance for the variance of the vertical velocity w. With 100 subcolumns this already looks a lot better. At the starting level it is pretty much exactly on point with the underlying variance from which was sampled. For around 500 meters up and down from the starting level it is close to the reference line. Above and below that area the line is less converged. For 1000 and 10000 subcolumns it is a similar result. Only the area where the convergence looks really good is larger. The main difference between 1000 and 10000 subcolumns is for the most upper levels of the simulation where the correlation between that level and the starting level is the lowest. With 10000 subcolumns the convergence in that area is slightly better.

Compared to the simple `Straight Monte Carlo Sampling`, `Importance Sampling` did slightly improve the convergence rate. The reason for that could be that the vertical velocity is not only varying inside of clouds but also outside of it. So by deeming the region inside the cloud more relevant it took away samples from the area outside of it.

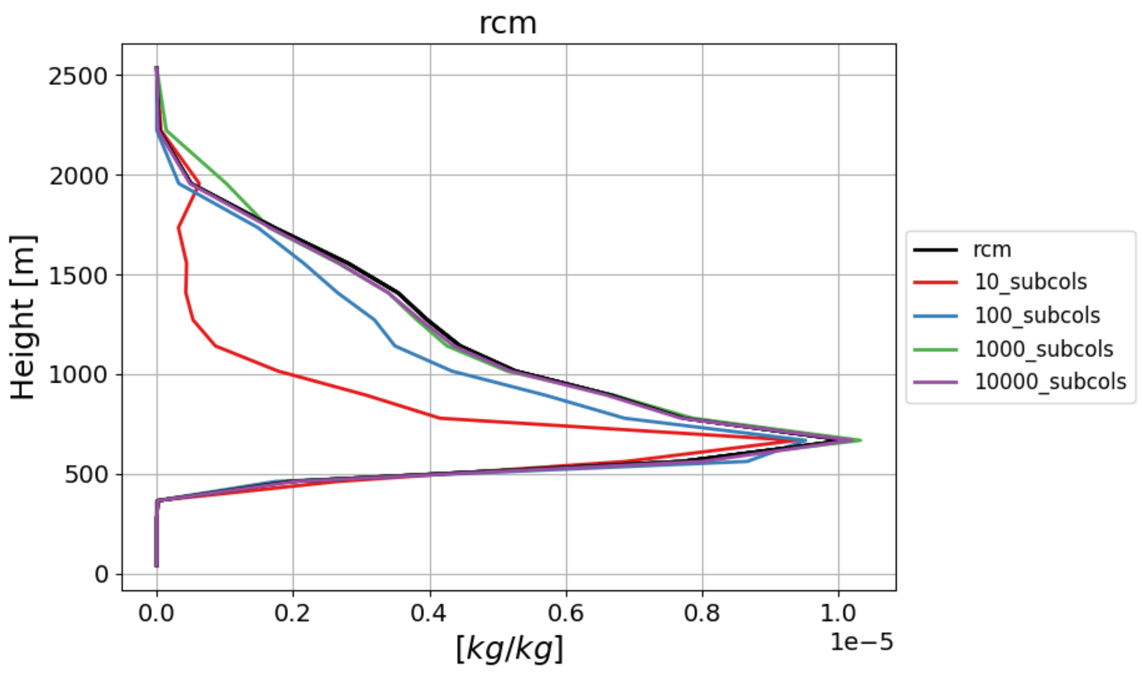

Figure 4.6: Convergence of *rcm* for the `Importance Sampling`

Figure 4.6 shows that, for the cloud water mixing ratio, choosing an important region does work very well. This is mainly due to the fact that the `Importance Sampling` was implemented in `SILHS` with variables like the cloud water mixing ratio in mind. This is why the important regions are the ones with liquid clouds, as most variables vary the most in there, and that the starting level is the one where the cloud water mixing ratio peaks.

With 10 subcolumns there is a good convergence already at least near the starting level. With 100 subcolumns the cloud water mixing ratio is almost converged on all vertical levels. With 1000 subcolumns the cloud water mixing ratio is almost identical with the original *rcm* values with the exception of an area from around 1800 meters to 2200 meters and then with 10000 subcolumns the cloud water mixing ratio is fully converged.

With the `Importance Sampling Strategy` it can be observed that, especially with the convergence of *rcm*, while the area around the starting level is matched very well, due to the exponentially declining correlations between the vertical levels the factor of that strategy does decrease the further away the vertical levels are from the starting level.

4.4 Clustered Sampling

The `Clustered Sampling` is a special case of the important sampling and was published by Raut and Larson (2016). This sampling strategy allows the modeler to give specific regions which should be considered more important than others in the `Importance Sampling`. This is done by dividing the domain of the `PDF` into eight different disjoint categories (Raut and Larson, 2016). They are listed in Table 4.1.

Category Number	Category Description
1	In cloud, In component 1, In precipitation
2	In cloud, In component 2, In precipitation
3	Out of cloud, In component 1, In precipitation
4	Out of cloud, In component 2, In precipitation
5	In cloud, In component 1, Out of precipitation
6	In cloud, In component 2, Out of precipitation
7	Out of cloud, In component 1, Out of precipitation
8	Out of cloud, In component 2, Out of precipitation

Table 4.1: Overview on the eight categories for the `Clustered Sampling` (Raut and Larson, 2016)

The component refers to the double Gaussian `PDF` that is used to model the vertical velocity. “In component 1” means drawing a sample from the first Gaussian and “In component

2" means drawing a sample from the second Gaussian.

Each of the eight categories is assigned a weight by the modeler, the only limitation being that they have to add up to 1. Naturally this allows for countless different settings and I will list only two exemplary cases.

All my settings do not put weight to the categories 1 to 4 as there was no precipitation during the simulation of the BOMEX case.

4.4.1 Setup 1

The first setup I used put equal weights to the categories 5 to 8. This is a setup comparable to the Importance Sampling. This is done to check if the result will be similar.

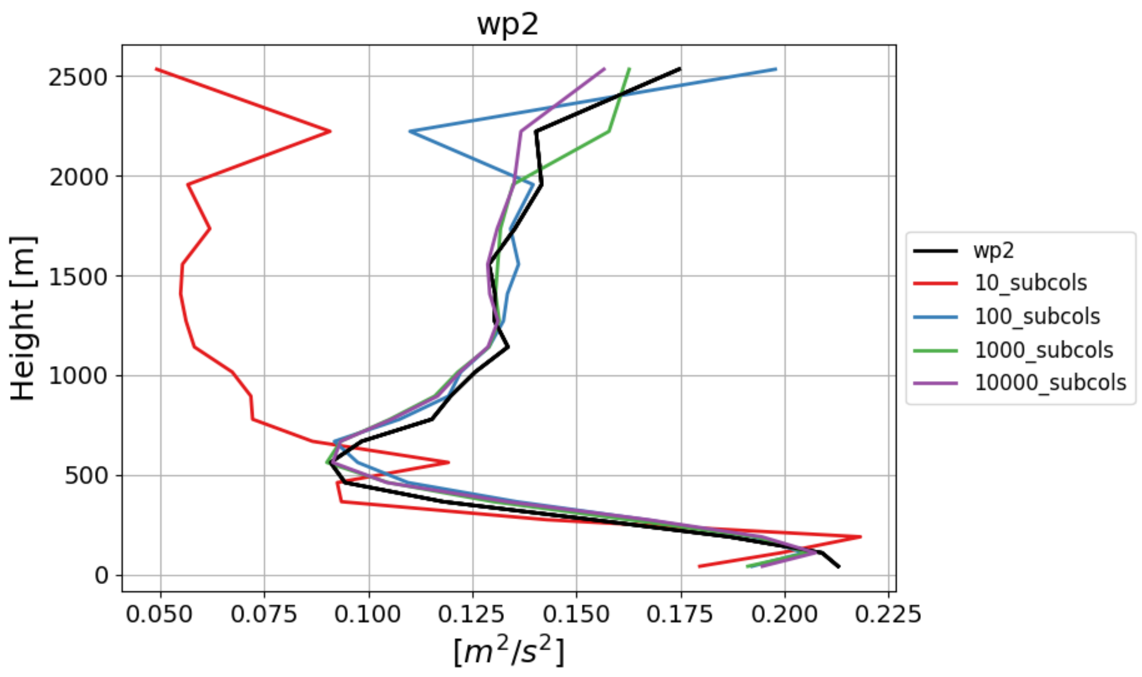

Figure 4.7: Convergence of *wp2* for the Clustered Sampling with setup 1

Figure 4.7 shows in fact similar results to the Importance Sampling. For 10 subcolumns there is a huge difference but this is no surprise when using so few sample points. With 100 subcolumns there is already a good convergence. Mainly the area the furthest away from the starting level is again the main problem. For 1000 and 10000 subcolumns the convergence is improving further.

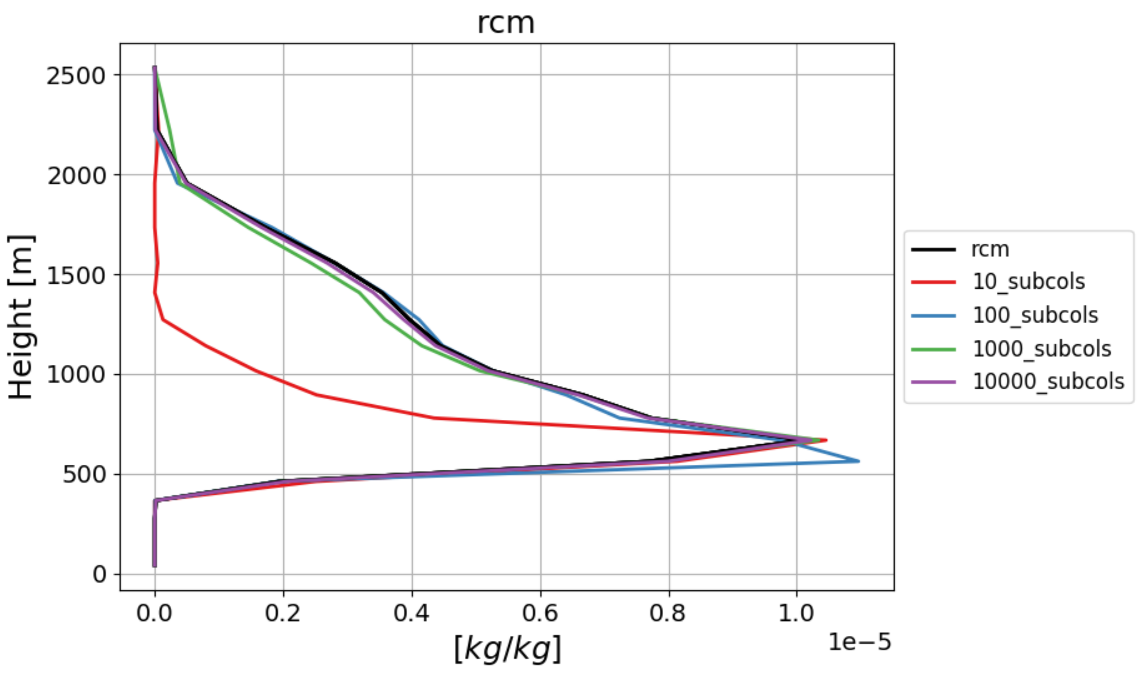

Figure 4.8: Convergence of *rcm* for the `Clustered Sampling` with setup 1

Figure 4.8 shows also similar results to the `Importance Sampling`. For 10 subcolumns there is already very good convergence near the starting level. With 100 subcolumns the cloud water mixing ratio is almost fully converged. 1000 and 100000 subcolumns only give slight improvement.

This first case shows that the `Clustered Sampling` is working properly with the implementation in the droplet activation process as it does produce similar results to the `Importance Sampling` when the categories are set accordingly.

4.4.2 Setup 2

For the next setup I put equal weights to the in cloud and out of cloud categories but I favored the first component three times over the second component. In other words categories 5 and

7 have the same weight, categories 6 and 8 have the same weight and category 5 (7) has three times the weight of category 6 (8). With the BOMEX case there is more emphasize on the first component of the PDF.

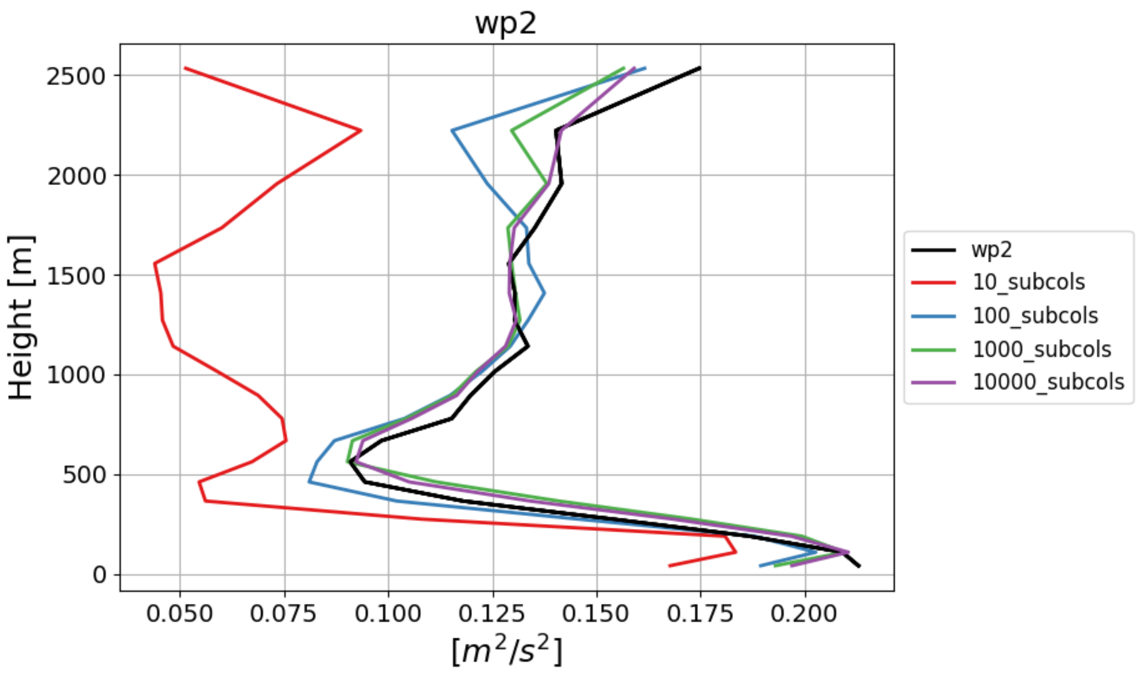

Figure 4.9: Convergence of *wp2* for the Clustered Sampling with setup 2

Figure 4.9 does look similar to the first setup in subsection 4.4.1. For 100 subcolumns there is some good convergence already. The problem is again mainly in the areas furthest away from the starting level. Between 1000 and 10000 subcolumns that is exactly the area that is improving the most and it does look a little better than in the setup before.

Figure 4.10 shows a similar result to Figure 4.9. In general the result does look very similar to the result from the first setup in subsection 4.4.1. However, favoring component 1 over component 2 did help a little with the convergence at the upper vertical levels.

The second setup has shown that changing the weights can influence the behavior at the starting level and the regions furthest away in different ways. Compared to the first setup the overall convergence is pretty similar but in the second setup the region around the

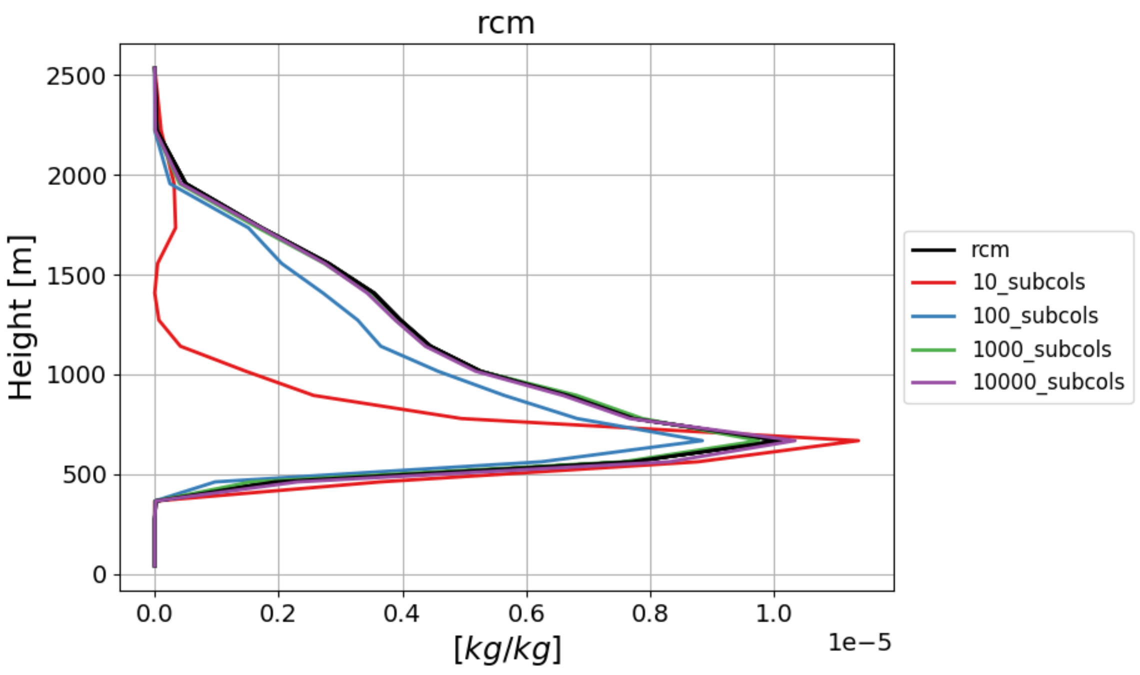

Figure 4.10: Convergence of *rcm* for the `Clustered Sampling` with setup 2

starting level does look slightly worse than in the first setup while the regions furthest away from that level looked a bit better. The worse result around the starting level could come from favoring component 1 too much over component 2.

Chapter 5

Conclusion and Outlook

In this final chapter I will summarize what was found during the previous chapters and will give an outlook on what could be done building on the findings in this thesis.

5.1 Conclusion

Implementing the support of subcolumns for the droplet activation process in `E3SM` has proven to be more difficult than originally expected. Some concepts in the model did help the task of refactoring the code for the use of subcolumns. Mainly, the challenge was to understand all the different variables that are used to specify the grid that is used as some of them were having the same values when running without subcolumns. This led to some mismatches in early stages of the refactoring in regards to array lengths of local variables. The refactoring has shown some specifics of the code that could be changed to help understand certain details of it better in the future, mostly in regards to the information on many different aspects of the simulation being placed in huge derived types.

The analysis of the different sampling strategies has shown that the `Straight Monte Carlo Sampling` does a good job in terms of convergence when compared to the other more complex strategies. Overall the convergence is happening relatively slow as for the most part 1000 or even 10000 subcolumns only seem to give a good convergence. `Importance Sampling` and `Clustered Sampling` for that matter have shown to improve the convergence at the starting level where the samples are actually drawn, at least for the *rcm* plots. This gave very good convergence even for only 10 subcolumns. The good these strategies did around the starting level did unfortunately not go through all the vertical levels as they are more and more uncorrelated to the original samples the further away they are from the starting level. All together this does suggest to just simply go with the `Straight Monte Carlo Sampling` as the preferred method of sampling as the results are good with a certain amount of samples and it does save some time during the simulation because there are no steps necessary besides drawing samples at random from the `PDF`.

5.2 Outlook

For the future there are some things that can be done by building on the findings of this thesis. It could be looked at other physical processes to determine if they would profit from using subcolumns. Such a refactoring could probably be done more easily with my notes on how to integrate subcolumns into physical processes in section 2.2. These remarks are for the most part not specific to the droplet activation process.

The `Importance Sampling Strategy` could be improved in a way that the positive effects at the starting level are carried through all vertical levels.

It could be worth taking the time trying different setups with the `Clustered Sampling Strategy` to find a setup that will better support the convergence of *wp2*. If a setup is found that improves the convergence not only for *rcm* but also for *wp2* that would probably make the `Clustered Sampling Strategy` the preferred strategy.

Bibliography

Boucher, O., D. Randall, P. Artaxo, C. Bretherton, G. Feingold, P. Forster, V.-M. Kerminen, Y. Kondo, H. Liao, U. Lohmann, et al., 2013: Clouds and aerosols. *Climate change 2013: the physical science basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, Cambridge University Press, 571–657.

E3SM Project, 2018: Energy Exascale Earth System Model (E3SM). [Computer Software] https://dx.doi.org/10.11578/E3SM/dc.20180418.36.

Fortran 90 Standard, 1991: ISO/IEC 1539:1991. Standard, International Organization for Standardization.

Gentle, J. E., 2003: *Random Number Generation and Monte Carlo Methods*. 2nd edition, Springer.

Golaz, J.-C., V. E. Larson, and W. R. Cotton, 2002: A PDF-based model for boundary layer clouds. Part I: Method and model description. *J. Atmos. Sci.*, **59**, 3540–3551.

Holland, J. Z. and E. M. Rasmusson, 1973: Measurements of the atmospheric mass, energy, and momentum budgets over a 500-kilometer square of tropical ocean. *Monthly Weather Review*, **101**, 44–55.

Larson, V. E., 2017: CLUBB-SILHS: A parameterization of subgrid variability in the atmosphere. ArXiv preprint arXiv:1711.03675.

Larson, V. E. and D. P. Schanen, 2013: The Subgrid Importance Latin Hypercube Sampler (SILHS): a multivariate subcolumn generator. *Geosci. Model Dev.*, **6**, 1813–1829. doi:10.5194/gmdd-6-1813-2013.

Larson, V. E., J.-C. Golaz, H. Jiang, and W. R. Cotton, 2005: Supplying local microphysics parameterizations with information about subgrid variability: Latin hypercube sampling. *J. Atmos. Sci.*, **62**, 4010–4026.

Lohmann, U. and J. Feichter, 2005: Global indirect aerosol effects: a review. *Atmos. Chem. Phys.*, **5**, 715–737.

Malavelle, F. F., J. M. Haywood, P. R. Field, A. A. Hill, S. J. Abel, A. P. Lock, B. J. Shipway, and K. McBeath, 2014: A method to represent sub-grid scale updraft velocity in km-scale models: Implication for aerosol activation. *J. Geophys. Res.*.

Raut, E. K. and V. E. Larson, 2016: A flexible importance sampling method for integrating subgrid processes. *Geosci. Model Dev.*, **9**, 413–429.

Siebesma, A. P., C. S. Bretherton, A. Brown, A. Chlond, J. Cuxart, P. G. Duynkerke, H. Jiang, M. Khairoutdinov, D. Lewellen, C.-H. Moeng, et al., 2003: A large eddy simulation intercomparison study of shallow cumulus convection. *J. Atmos. Sci.*, **60**, 1201–1219.

Thayer-Calder, K., A. Gettelman, C. Craig, S. Goldhaber, P. A. Bogenschutz, C.-C. Chen, H. Morrison, J. Höft, E. Raut, B. M. Griffin, J. K. Weber, V. E. Larson, M. C. Wyant, M. Wang, Z. Guo, and S. J. Ghan, 2015: A unified parameterization of clouds and turbulence using CLUBB and subcolumns in the Community Atmosphere Model. *Geosci. Model Dev.*, **8**, 3801–3821.