Documentation of

Python Implementation of

All-or-Nothing Coalescence/Aggregation Algorithm

in a box model and a column model

0 Overview

The program package contains two kinds of run scripts and main programs, one for computations (COMP) and the other one for a posteriori plotting (PLOT).

The source code files for both, COMP and PLOT, lie in the same directory as they share common python modules.

COMP and PLOT routines are both executed via csh-scripts that have a similar structure. See examples plotAgg_dummy.csh and ComputeAgg_dummy.csh.

The main COMP and PLOT python module called from the csh script are CompSim.py and PlotResults.py, respectively.

COMP can handle various kernels (Golovin, Hall, Long, product kernel) and use various variants of the AON implementation (treatment of multiple collection, discrete vs continuous distributions, kernel interpolation, linear sampling, explicit overtakes in the column model).

Convention: Use SI units throughout the program.

Distributed calculations: If the sequential computation of the complete ensemble takes long, the ensemble simulation can be split up in several subsimulations, each subsim computing a subset of instances on a separate core.

Box model or Column model computations are possible (**COLUMN** = 0 or 1)

A detailed description of each python module is given below. The section on COMP describes all python modules used in COMP (see third column). The section on PLOT describes the remaining modules that are exclusively used in PLOT.

Module name	Description	COMP	PLOT
CompSim.py	Contains the main program for COMP		
PlotResults.py	Contains the main program for PLOT		
AON_Alg.py	Actual AON implementation		
Kernel.py	Definitions of kernel values		
Sedimentation.py			
SIPInit.py	Initialization of SIPs		
Referenzloesung.py	Definition of reference solution		
PlotSim.py	Plotting routines		
InputOutput.py	reads output data of existing simulations		
Misc.py	Auxiliary tools		

1 The program package COMP

COMP contains the implementation of the AON algorithm in a 0D box model and a 1D column model.

1.1 List of Python modules

1.1.1 CompSim.py

CompSim.py contains the main program of the 0D aggregation box model and 1D aggregation/sedimentation column model.

The routine is called by ComputeAgg_*.csh

Flow control, parameter setup, time logging.

The box model and the column model versions are coded in two separate blocks. Only in the beginning a small code fraction (parameter setup, generation of the kernel lookup tables, if applicable) is present in both versions.

Outline of the **box model structure (COLUMN = 0)**:

outer iteration over instances, SIP initialisation,
time iteration: calls **AON.Aggregation** (as long as N_SIP > 1), removes zero weight SIPs, output of moments and SIP data,
in the end calls to MOM and GV plot routine.

Outline of the **column model structure (COLUMN = 1)**:

outer iteration over instances, SIP initialisation (spectral and spatial distribution), sort SIP-list by z and determine index ranges for all grid boxes, resize SIP-list
time iteration: calls **AON.Aggregation** (as long as N_SIP,GB > 1) for each grid box, calls **SD.fallg** and updates z_SIP (COLUMN =1) for whole SIP-list, deactivate zero weight SIPs, deactivate SIPs that fall out of the domain, track flux across boundaries, output of moments, SIP and flux data, sort SIP-list by z and determine index ranges for all grid boxes

For explicit overtake option across total column (**WELLMIXED = 3**), no SIP-attribution to grid boxes is necessary. Only one call of **AON.Aggregation** with one list of all SIPs

In the end calls to various plot routines depending on **MOM_meanTD, GV_meanTD, RZ_scatter, MOM_prof, FLUX_time.**

By the way, SIPs are "deactivated" by setting z=zNan (after initialization, the SIP-list is resized such that only one SIP with z=zNan remains; it necessary to keep one such SIP to allow for a more efficient implementation of **FK.get_isep**)

Options:

- spatial initialisation i_init_1D: 1 = empty domain, 2 = top GB only, 3 = top half domain, 4 = linearly decaying over total column from 2*g_init to 0, 5 = linearly decaying over top half from 2*g_init to 0, 6 = total domain, 7 = linearly decaying over top quarter from 2*g_init to 0, 8 = sin()-hill over top half with 2*g_init max peak, 9 = sin()-hill over top quarter with 2*g_init max peak
  Note that the sine init evaluates the sine weighting function at grid box vertical centers
- influx across top boundary **INFLUX_TOP**: = 0 no influx , = 1 influx with given SD (not yet implemented), = 2 periodic BC, outfalling SIPs re-enter domain
- toggle processes **PROCESS**: Sedi+Koaleszenz aktiv (=0), Sedimentation (=2) bzw Koaleszenz (=1) ausschalten

Features: if N_SIP = 1, skip **AON.Aggregation** call, but continue time iteration until the end time, in order to have the same amount of output for every instance and simulation.

1.1.2 AON_Alg.py

The actual AON algorithm, pre-computes indices for Hall/Long kernel lookup-table (if applicable), double iteration over SIPs i and j, access/interpolate kernel value, compute n_agg, check for single or multiple collection

Options:

1) DISCRETE:

= 0: a continuous initial size distribution is given, translation into SIP ensemble with a SIPInit-technique, output SD in units *mass / (volume * dlog(r))*, weights are floating point numbers.

= 1, 2: a discrete initial size distribution is given (which simplifies the SIP initialisation), output SD in units *mass/volume,* SIP masses are integer valued and scaled with the specified elemental mass (skal_m), SIPs weights are integer numbers.

= 2: special case: only SIP weights of 1 are allowed. No multiple collections occur; use break command in outer SIP loop, once this SIP is part of a collection event.

= 1: integer values for SIPs weights. Use this option together with **AGG_MC** = 1 to guarantee integer SIP weights throughout the simulation

2) **AGG_MC**:

= 0: no multiple collection; if p_col > 1, then SIP collection event is always realised.
= 1: if p_col > 1, then do multiple collection with FLOOR(p_col) or CEIL(p_col) collections
= 2: if p_col > 1, then do multiple collection with p_col collections; does not conserve integer property of SIP weights.

3) **LINEAR**:

= 0: iterate over all SIP combinations with i < j
= 1: linear sampling of SIP combinations and upscale n_agg, replace double iteration over SIPs i and j by single iteration over i_comb. Upscaling n_agg could lead to case where n_agg>max(n_i,n_j) and which would lead to negative weights. Set **LINEAR_LIMIT**=1 to prevent this by bounding n_agg. Not compatible with **WELLMIXED > 0**

4) **KERNEL**:

= 0: Golovin kernel, direct computation of kernel value
= 1, 2: Long/Hall kernel, pick values from lookup table
= 3: product kernel, direct computation of kernel value.

5) **KERNEL_INTPOL** (only relevant for Long and Hall kernel)

= 0: no interpolation, take "centered" index (produces n_agg = 0 if radii of two SIPs fall in same bin, as the diagonal elements are zero)
= 1: bilinear interpolation in log(r)-log(r) space, take fine Hall/Long kernel lookup table
= 2: customized Hall lookup table with 40 defined linearly gridded radius values, for discrete Alfonso example, no interpolation necessary

6) **COUNT_COLLS**:

= 0: no statistics are produced, this option leads to a much faster execution, as no else part (no collection occurs, simple count of no collection events) appears
= 1: count the number of no collections, single collection and multiple collection and limiter events. Use array count_colls of size ncoll=20 to store number of events. count_colls[1] saves number of single collections, count_colls[0] saves number of no collection (with positive pkrit). Indices 1 to ncoll-3 store multiple collections with min([math.ceil(c_mc -0.5),ncoll-3]). Index ncoll-2 store cases with pkrit=0 (occurs if weight is zero or similar radii with no kernel interpolation). Index ncoll-1 saves number of limiter events.

7) **SPEED_ZERO (only relevant for DISCRETE<=1)**:

= 1: do no test for zero weight SIPs (*ACHTUNG bei DISCRETE =1 ist das wohl nicht so clever, besser auf DISCRETE=0 einschränken*)

8) **SPEED_VECTOR**:

= 0: compute random number for each SIP pair separately
= 1: compute random numbers at once for all SIP pairs (speed up nearly negligible, in particular in the full column treatment, where potentially any two SIPs may collide, but often no overtakes, so more random numbers are computed than needed)

9) **WELLMIXED (only relevant for COLUMN =1)**:

= 0: 3D wellmixed assumption

> 0: 2D wellmixed assumption with explicit overtakes:

= 1: overtakes only with SIPs of same grid box

= 2,3: all possible overtakes are tested (including those across grid box boundaries and for =3 also across lower pBC)

Speed up of explicit overtake options (in particular with **WELLMIXED = 3** by setting **REVERSE = 1**)

Only compatible with **KERNEL = 1/2**

10) **REVERSE**:

The SIP lists are sorted by height starting with the highest SIP in the GB

= 0: start with highest SIP (of grid box or column)

= 1: reversed order, start with lowest SIP (of grid box or column)

1.1.3 Kernel.py

Routines only required if **KERNEL** = 1 (Long) or 2 (Hall). If **KERNEL** = 0 or 3, Golovin or product kernel values are directly computed in **AON_Alg.py**.

Reads Hall or Long kernel values with 400 radius bins (log scale) from an input file.

Alternatively, the Long kernel values can be computed and stored in similarly shaped lookup table (**LongK_Options** = 2).

For the Hall kernel, an alternative input file with 40 linear radius bins exists (**KERNEL_INTPOL** = 2). The latter is used with **INITV** =2 and **DISCRETE** = 2

All input files were generated by a fortran subroutine of the Bott program

1.1.4 Sedimentation.py

contains the function **Fallg** which calculates fall speed parametrisation by Beard (1976).

This function is only called if Long kernel values are explicitly calculated (**KERNEL == 1 && LongK_Options == 2)** or if sedimentation is switched on in the column model (**COLUMN == 1 && PROCESS != 2)**

1.1.5 SIPinit.py

Contains various SIP init routines:

INITV =1: function **InitSIP_ExpVert_singleSIP_WS**

uses SingleSIP-approach to create a SIP-ensemble from a continuous exponential mass distribution (as in Wang, Bott or Berry or GMD-paper). In conjunction with **DISCRETE** = 0

Specify:

- mean droplet radius r0 (default 9.3um)
- mass concentration LWC (default 1 g/m^3)
- bin resolution kappa (n10 number of bins per mass decade, default 40)
- code imlow (specifies minimal radius, default 2 , 0.6 um)
- ratio maximal minial SIP weight eta_nu (default 1e-9)
- For a reasonable dV-sensitivity test, the number of initialised SIP must increase with dV (a simple dV-increase yields identical results for an unchanged SIP-ensemble). dV*dV_skal is the

volume of the grid box and dV_skal specifies how many instances of the same SD are generated in each grid box.

INITV =2: function **InitSIP_Alfonso**()

discrete initial distribution, simple transformation into SIP space. Use in conjunction with **DISCRETE** >= 1.
Use Hall kernel example from Alfonso with 20 17um droplets and 10 21.4um droplets (iSIPinit_discrete = 1).
Use product kernel example from Alfonso with 100 14um droplets (iSIPinit_discrete = 2).

1.1.6 Referenzloesung.py

Reads pre-computed reference solutions according to **IREF** and other PPVs like **DISCRETE** and **KERNEL**.
No reference solutions are provided for **IREF = 0**.
For **IREF = 1**: Solutions exist for the standard initial exponential distribution for **KERNEL** = 0 (from analytic Golovin solution, every 200s), **KERNEL** = 1/2 (Long/Hall solutions of Wang, every 600s).
For **IREF = 2**: the discrete Alfonso Hall kernel example (SD at t=2500s).
For **IREF = 3**: the discrete Alfonso product kernel example ().
For **IREF = 9**: a column model Bott/Wang reference simulation is processed.

For all **IREF ≠ 3**, data on SDs are provided. The routine **PlotGV_0D** reads the data and adds a curve to an existing plot. The parameters ifirstGV and ilastGVonly control which times are selected. For IREF = 9, the data is added in

For all **IREF ≠ 2**, moment values are provided. The function **defineMoments_0D** contains the data and returns the time grid and the moment matrix to **PlotMoments** where data curve are added to a plot.

For **IREF = 2, 3**, Alfonso provides the time evolution of the relative standard deviation across all instances of the largest SIP mass in an ensemble. The reference curve is added to the plot with the function **PlotRStD_0D**.

For **IREF = 9**, fluxes across lower and upper BC and profiles of moments are processed besides SD data and total moments.

1.1.7 PlotSim.py

Contains routines to plot the time evolution of the moments (**PlotMoments**), the size distribution (**PlotGV**), the relative StDev of m_{max} (**plotRelStdDev**), vertical profiles of the moments (**PlotMomentsProf**), the scatter of SIP radius and position (**Scatter_r_z**), fluxes across lower/upper boundary (**PlotFluxesTime**), the mean relative SIP position inside the grid boxes. All routines are programmed such that plots of a single simulation or plots of several simulations can be produced. The plot layout can depend on pre-processor variable settings. Selection of which types of plots are produced is done via pre-processor directives **MOM_meanTD, GV_meanTD, MOM_prof, RZ_scatter, FLUX_time, TRACK_CENTERS**

PlotMoments: MOMsave passes the moment values of a single simulation to the routine (separately for each instance or only the mean over all instances, if optional parameter iMean = 1), can plot all instances or only the mean (depending on the value of the optional parameter iplot_onlyMOMmean). If a single simulation is plotted, the routine opens the figures, adds the curves and closes the figure. If several simulations are plotted, the routine is called for each simulation and a curve is added. The figure is opened with the first routine call and closed with the last routine call (controlled by optional parameter iplot_mode). The temporal resolution is given by t_intervall_MOMsave (interval in seconds). So far, four panels with moments 0 to 3 are produced. In the column model, the average over all grid boxes is taken (**MOM_meanTD = 1**) or integrated moments are used (**MOM_meanTD = 2**).
IREF controls if and which reference solution is plotted. The reference solution is added just

before closing the figure. For **IREF** is 1 or 3, **REF.defineMoments_0D** is called to obtain hardcoded reference data. If **IREF** is 9, then bin column model data from a specified path is read inside routine **REF.get_RefProfileData.** In any case, the returned values are then plotted inside the current routine.
Optional arguments to this routine mostly control the appearance of the plot: label=None, title=None, text=None

PlotGV: nEK_sip_plot, mEK_sip_plot pass the SIP data (SIP weights and masses) of all simulations that are plotted to the routine. The matrix has dimensions (nr_sims,nr_inst,nr_GVplot,nr_SIPs). If only a single simulation is treated, the input 3D data is inflated to 4D with a first dimension of length 1 (set the optional parameter iMultipleSims = 1).
nr_SIPs_plot contains the SIP number for each instance (dimension nr_sims,nr_inst,nr_GVplot). Passing the optional parameter nr_inst_vec handles the case when the simulations have different numbers of instances.
SIP data is saved every t_intervall_GVplot seconds. SDs at all available points of time are displayed, unless ilastGVonly = 1 or the optional argument iTimes_select specifies the selection of times. iTimes_select is a list of lists. For each element of the top-level list a new plot is created and each element is again a list with the times to be displayed. The various plots can contain different numbers of displayed points in time.
In calls from **PlotResults** for a-posteriori plot generation, the total column volume of each simulation has to be specified (optional argument V).
If **DISCRETE** is 0, the discrete SIP ensemble is converted back into a continuous distribution (function **FK.MapSIPBin**). If **DISCRETE** is >=1, it is meaningful to display the discrete SIP ensemble. The function **FK.CountSIPs** counts the numbers of occurrences of each SIP integer mass.
Only the mean size distribution averaged over all instances is displayed. And in the column model it is further averaged over all grid boxes.
For **IREF = 1**, **REF.read_GVdata_Wang()** provides reference data from classical Wang box model simulation. If **IREF** is 9, then bin column model data from a specified path is read inside routine **REF. get_RefSDdata.** In any case, the returned values are then plotted at the specified times.
Further optional arguments are: label, title
Moreover, params_plot.txt includes relevant parameter settings like the choice of the cycler for the various displayed times and simulations (solved with keys of a dictionary)
Usually SD per volume are displayed (outfallingGV = 0). The plot routine can also be used for plotting the accumulated SD of all outfalling SIPs (outfallingGV = 1) or the accumulated SD of all incoming SIPs (outfallingGV = 2). Then only a single time step is displayed (inflate SIP data by 1-element time dimension) and the SD per area is displayed.

PlotRelStdDev {side note: the structure is similar to routine **PlotGV**}**:**

What is the relative standard deviation? For each time, the mass of the largest SIP mass m_{max} of the SIP ensemble is recorded. The standard deviation of m_{max} over all instances is computed and scaled by the mean value of m_{max} over all instances.
mEK_sip_plot passes the SIP masses of all simulations that are plotted to the routine. The matrix has dimensions (nr_sims,nr_inst,nr_GVplot,nr_SIPs). The quantity is computed inside this routine and then plotted. If ihistogram = 1, histograms of m_{max} are computed and plotted in an additional subpanel. . If imaxelement = 1, time series of masses of the largest and second largest SIP are plotted in an additional subpanel.
This kind of evaluation is only done for simulations with **DISCRETE** = 1.

PlotMomentsProf:

This plot type is only useful for column model applications.
MOMsave passes the moment profile values of a single simulation to the routine (separately for each instance or only the mean over all instances, if optional parameter iMean = 1), can plot all

instances or only the mean (depending on the value of the optional parameter iplot_onlyMOMmean). The MOMsave array has dimensions (nr_sims, nr_inst, nr_MOMsave, nz, nr_Mom). If only a single simulation is plotted (optional argument iMultipleSims=0) the array inflated with 1-element time dimension. If only mean values are provided (iMean = 1), the array is inflated with 1-element instance dimension. If multiple simulations are passed, mean values have to be provided (iMultipleSims=1 is only compatible with iMean=1) and vectors nz_vec(nr_sims) and nt_vec(nr_sims) have to be provided.
The optional argument iTimes_select specifies the selection of times to be displayed. For each moment order a separate plot is produced. iMom_select controls which moments are displayed.
The vertical mesh size dz is defined in "params.txt", however it cannot be specified in "params_plot.txt" => if **ICOMPPLOT** = 2, dz is read in and passed to the routine.
IREF controls if and which reference solution is plotted. The reference solution is added just before closing the figure. If **IREF** = 9, then bin column model data from a specified path is read inside routine **REF. get_RefProfileData.**
Further optional arguments to this routine mostly control the appearance of the plot: label=None, title=None, label=None.
Inside the routine, the internal variable smooth_av can be set in order to activate smoothing of the profiles.

Scatter_r_z

This plot type is only useful for column model applications. It depicts the z-position versus the radius of each SIP.

PlotFluxesTime

This plot type is only useful for column model applications. This routine plots the temporal evolution of instantaneous and accumulated fluxes. Fluxes of moments 0 to 3 and of the SIP number across the upper and lower boundary are displayed (thus size nr_MOMs+1 along third dimension). For this, the fields FluxIn, FluxOut, FluxInAcc, FluxOutAcc (each of size nr_sims, nr_inst, nr_GVplot-1, nr_MOMs+1) are passed to the routine. No data is available for t=0 (thus nr_GVplot-1). Instantaneous here means the average flux over the last 200 seconds. Again, the optional argument iMean specifies if mean values or values for each instance are provided (note the optional argument plot_onlyMean is not used currently and mean values are always plotted).
If MultipleSims = 1 or equivalently nr_sims > 1, then nt_vec(nr_sims) has to be provided.
If io_sep = 0, influxes and outfluxes are both plotted in a single plot. If io_sep = 1, separate plots are produced.
If **IREF** = 9, then bin column model data from a specified path is read inside routine **REF. get_RefFluxData.**
Further optional arguments to this routine mostly control the appearance of the plot: label=None, title=None, label=None.

PlotCenter

This plot type is only useful for column model applications. So far, only used in online-plotting in conjunction with **TRACKCENTER=1.** Single simulation, displays mean relative moment weighted SIP positions inside grid boxes before sedimentation and aggregation, in between of the two processes and after both processes. Data is available for all moments, however is most reasonable for the weighting with SIP masses, as total mass is conserved in both processes.

1.1.8 Misc.py

This file contains various routines.

MapSIPBin

Produces bin-based size distribution data for a given SIP ensemble. Use bin grid parameters n10_plot, r10_plot, min10_plot. First the SIP data is sorted by size. Then each SIP is added to its respective bin. Called by **PlotGV** with **DISCRETE = 0.**

CountSIPs

Produces bin-based discrete size distribution data for a given SIP ensemble. For each SIP the integer-valued SIP masses determines the bin. Called by **PlotGV** with **DISCRETE = 1.**

CIO_MOMmean

CIO stands for "**C**ompute or **I**nput/read mean Moments" and **O**utput the data.
The moment data of all instances is passed by data_in or data is read from a specified folder (fp_in, works only for **COLUMN** = 1).
The average over all instances is taken. In the column model version, the optional argument ikeep1D specifies, if averaging is also done over the column (ikeep1D=0) or mean profiles are computed (ikeep1D=1).
If fp_out is supplied, the mean data is written to fp_out + 'MomentsMean.dat'.
Note the data input/output files contain normalised data; the returned moment values however are absolute values scaled by skal_m^{i_order_MOM}/dV.

get_isep, get_isep2

assumes a z-sorted SIP-list. Determine index ranges for SIP blocks for each grid box. The fast implementation **get_isep** (called from **CompSim**) assumes that at least one SIP with zNan is at the end of the SIP list. The slower version **get_isep2** (called from **AON_feed**) includes an additional check in each iteration.

trackcenters

computes mean relative position of all SIPs in a grid box; weighted by sum(v_i μ_i**iMom) or sum(μ_i **iMom). Returns weighted average tmp as well as weighted sum nom and weight denom.

openfile

checks if file or .gz-compressed file exists and returns file handle

1.2 Output Data

All output data is written inside the module Comp.py. Data is written to the output files step by step as the iterations over the instances (outer loop) and over time (inner loop) progress.

1.2.1 Moments.dat

The file ***Moments.dat*** contains values of $\sum_i \nu_i \hat{\mu}_i^k$ with k= 0, 1, 2 and 3. The file contains an array of size (nr_inst, nr_MOMsave, 4) for **COLUMN**=0 or of size (nr_inst, nr_MOMsave, nz, 4) for **COLUMN**=1.

v_i is the number of droplets in a SIP (in units 1) and $\hat{\mu}_i$ is the normalized mass of a droplet (in units 1). To obtain moments values in units (kg^k/m^3) the output data must be scaled with dVi * skal_m**k

The meta file ***Moments_meta.dat*** contains the values of dV , skal_m , nr_inst, nr_MOMsave, and the time vector t_vec_MOMsave.

1.2.2 SIP.dat

1.2.2.1 Box model

The file ***SIP.dat*** contains the values of v_i and $\hat{\mu}_i$. For each output time three lines are output. Line 1 contains the single value nr_SIPs. Line 2 and 3 contain arrays of size nr_SIPs with the v_i and $\hat{\mu}_i$-data. Producing reasonable plots it may be necessary to use v_i dVi and $\hat{\mu}_i$ skal_m.

The meta file SIP_meta.dat contains the values of nr_inst, nr_GVplot, and the time vector t_vec_GVplot.

1.2.2.2 Column model

The file ***SIP.dat*** contains the, values of v_i, $\hat{\mu}_i$ and z_i. For each output time four lines are output. Line 1 contains the single values nr_SIPs and iz. Lines 2 to 4 contain arrays of size nr_SIPs with the v_i, $\hat{\mu}_i$ and z_i-data. Producing reasonable plots it may be necessary to use v_i dVi and $\hat{\mu}_i$ skal_m.

The meta file ***SIP_meta.dat*** contains the values of nr_inst, nr_GVplot, and the time vector t_vec_GVplot. This is followed by nr_inst 2D-arrays of size (nr_GVplot,2). For each instance and time, the first column gives the total number of SIPs nr_SIPs and the second column the maximum number of SIPs nr_SIPs_maxGB in a grid box. The latter information can be used for the following. To save the SIP data in memory, one could define an array of size nr_inst, nr_GVplot, max(nr_SIPs) which neglects the spatial information. Alternatively, one could use an array of size nr_inst, nr_GVplot, nz, max(nr_SIPs_maxGB).

Additionally, all outfalling SIPs are recorded in the file ***SIPout.dat***. Each time step SIPs fall out of the domain, three lines are output. The first line contains the time and the SIP number ntmp. The second and third line contains arrays of size ntmp with v_i and $\hat{\mu}_i$. This is done for all instances.

The file ***SIPout_meta.dat*** saves for each instance the number of output time steps, i.e the number of three line blocks in ***SIPout.dat***. The total number of SIPs that were recorded can be attained from ***Fluxes_out_acc.dat.***

Analogously, data of all incoming SIPs are recorded in ***SIPin.dat***. ***SIPin_meta.dat*** and ***Fluxes_in_acc.dat*** help to process the data.

1.2.3 Flux data files

The output files are ***Fluxes_in.dat, Fluxes_in_acc.dat, Fluxes_out.dat, Fluxes_out_acc.dat*** (only for **COLUMN** = 1). Each output line contains 5 values. Fluxes for moments of order k = 0 to 3 and fifth values is the number of SIPs. The units for the moment fluxes are in kg^k/m^2 (accumulated fluxes) and $kg^k/(s\ m^2)$ (instantaneous fluxes). The SIP numbers are given in unit 1. The time resolution is as for SIP.dat, yet the flux files contain only nr_GVplot-1 time instances, as no output is written at t=0. So each file contains an array of size nr_inst, nr_GVplot-1, 5.

Unlike to the bin model, the instantaneous fluxes are actually no real instantaneous fluxes. This type of evaluation makes no sense in a discrete particle-based model, whereas SIPs cross the boundaries intermittently and accumulated fluxes are piecewise constant functions and the instantaneous function is sequence of “on-off”. Therefore instantaneous is the difference of the accumulated fluxes between two neighbouring instances in time (usually the time interval is 200s and hence the fluxes are averaged instantaneous fluxes over the last 200s).

The influx at the top boundary is non-zero for **INFLUX_TOP** = 1 and additionally equals the outflux for **INFLUX_TOP** = 2.

1.2.4 Track center data

The output files are ***centerIC_full.dat, centerSIP_full.dat, centerIC_av.dat, centerSIP_av.dat*** (only for **COLUMN** = 1). Each output line contains 4*3 values (4 moments and 3 code locations: before, in

between, after). ***centerIC_full.dat, centerSIP_full.dat*** contain height and time resolved data for each instance. The array size is (nr_inst,nr_MOMsave,nz,nr_MOMs,3). ***centerIC_av.dat, centerSIP_av.dat*** contain data of size (nr_MOMsave,4*3) that is averaged over the column and all instances. The output to the latter two files is written after all instances have been completed.

1.2.5 Log file

The log file ***log.txt*** is first generated within the COMP csh script and is further modified within the python COMP program. It contains program and run time statistics. During an ongoing simulation, the log file shows the estimated end time and lies in CWD. After the completion of the simulation, the total computing time is evaluated and documented and the file lies in the subfolder "CodeCompute".

Following information is contained in log file:
the name of csh script, the hostname, the (original) output folder, time statistics for each instance.

2 The program package PLOT

This PLOT package controls the a-posteriori plotting of a single or multiple simulations. The PLOT package uses the modules described in this section and several modules that have already been described in section 2, namely: PlotSim.py, Referenzloesung.py and others

Plots of a single simulation are done for **IMODE** = 1. Plots of multiple simulations are done for **IMODE** = 2. Each type of plot can be selected by setting a specific PPV to >0:

MOM_meanTD
TD = Total Domain
= 1 plot mean moments, average over all instances and full column, units $kg^l\ m^3$
= 2 plot integrated moments, average over all instances, difference to =1, multiply by Lz, units $kg^l\ m^2$
= 3 plot total moments, simply sum up SIP-data, units kg^l

GV_meanTD
plot mean size distribution, average over all instances and full column, TD = TotalDomain

GV_out
plot size distribution over all outfalling SIPs, average over all instances

MOM_prof
plot vertical profiles of mean moments, average over instances

RZ_scatter
(r,z)-scatter plot

FLUX_time
plot fluxes over time

The data of each simulation can be distributed across several SubSim-folders.

2.1 PlotResults.py

Main program for a-posteriori generation of plots; the routine is called by plotAgg_*.csh.

Contains several routines, each of them generates a certain type of plot. First, data has to be read (calls a routine of module InputOutput.py) and then data is plotted (calls a routine of module

PlotSim.py). **InputOutput.py** is constructed such that both, box model and column model data can be processed.

Plots of single simulations (**IMODE** = 1) and optionally one reference solution (**IREF**) are treated in routines: **plotSingleSimulationGV, plotSingleSimulationRStD, plotSingleSimulationMOM, plotSingleSimulationFLUX, plotSingleSimulationGVout**

Plots of multiple simulations (**IMODE** = 2): **plotMultipleSimulationsMOM, plotMultipleSimulationGV, plotMultipleSimulationRStD, plotMultipleSimulationGVout**

2.2 InputOutput.py

Each routine reads a specific type of data of a single simulation (moments, SIPs, fluxes). Any of those routines can handle SubSims. In this case, the simulation folder contains a file named InstanzenMeta.dat, which specifies the number of SubSims, the total number of instances and the names of the subfolders where the data of the SubSims are located.

Routines for **COLUMN**=0 and 1: **readSingleSimulationSIPdata, readSingleSimulationMoments**

Routines only for **COLUMN**=1: **readSingleSimulationFluxes, readSingleSimulationSIP_outfalling_data, get_SubSimsData, get_nz, get_dz, get_MetaData**

3 Technical aspects of the csh scripts

3.1 Sequence of tasks

Before and after calling the respective main program in csh script some preparatory work is done within the shell script.

Steps: Copy source code, generate inlined parameter files, preprocess python code, execute python code, cleaning up.

3.2 Specific steps for COMP

1. The source code is copied to a specified working directory (CWD) in which also the COMP output is saved.
2. A parameter file “params_gcc.txt” with pre-processor variable definitions is generated. A parameter file “params.txt” in Python syntax contains many parameter settings. The commands are not indented and can be included in the beginning of any Python module via a #GCCinclude statement.
3. Invoke the pre-processor for all python files listed in $list_module_gcc_files. Python files that are not pre-processed are copied
4. After completion of the COMP python program, the source files are moved and temporary files are deleted. In the end, the full source is saved in the subfolder “CodeCompute/full_source”, and the preprocessed files are located in “CodeCompute”.

3.3 Specific steps for PLOT

The steps are slightly different, depending on the plot job.

1. The source code is copied to a specified working directory (CWD) in which also the figures are saved. If plots of a single simulation are created, the given CWD coincides with the directory of the COMP data. In such a case, CWD = “CWD/plot/”.
2. A parameter file “params_plot_gcc.txt” with pre-processor variable definitions is generated. A parameter file “params_plot.txt” in Python syntax contains many parameter settings. The commands are not indented and can be included in the beginning of any Python module via a #GCCinclude statement.
3. Invoke the pre-processor for all python files listed in $list_module_gcc_files. Python files that are not pre-processed are copied
4. After completion of the PLOT python program, temporary files are deleted. Typically, the source files are not moved. Hence, preprocessed source files and PLOT output are in the same directory.

3.4 The preprocessing:

Using pre-processor directives in Python needs a little workaround, as both languages use the # symbol for signalling a GCC directive or a Python comment. We introduce that GCC directives always start with #GCC instead of with a single # symbol. All other lines starting with “#” are treated as Python comments. In order to be able to apply the GCC pre-processor to a source file, several steps of text replacements have to be made with sed. First lines with Python comments have to be modified such that they are not interpreted as GCC directives. Lines with #GCC are modified such that the line starts with #if, #endif, … (leading whitespace is removed (important! Otherwise the GCC interpreter ignores the statement), and “#GCC” -> “#”). With a cat command the file params_gcc.txt is put in front and the pre-processor is invoked.

4 Nomenclature:

Mathematical formulas, physical quantities	*italics*
Python variables	green
Python modules and routine names	bold
Preprocessor variables	Red bold
Shell variables	underlined
Shell script name	
Folder names/paths	In “..”
Unix programs and tools	