Chamber model manual

1 Creating and initializing a chamber object

The chamber model is based on object-oriented programming. That is, every simulation is an object that has functions (methods) and variables. To run a simulation, one has to first create a chamber object by writing

```
simulation_1 = chamber;
```

where *"simulation_1"* is the name of the object, and *"chamber"* tells Matlab that this object is a chamber model object.

Now *"simulation_1"* has all the functions needed to actually run the simulation. But before that, one must define the initial conditions. A simple case would be a log-normal particle distribution that develops by condensation, coagulation and dilution during the simulation.

All the initial parameters of a chamber object are found in structure *chamber.initials*, where *"chamber"* represents the name of the chamber object, so in this case the initial values are seen by typing

```
simulation_1.initials
```

in Matlab, without the semicolon at the end. These values can be modified using function *chamber.initialize()*. All the parameter names and their descriptions are found in chapter 5.

Let us first set the initial log-normal distribution so that its mean diameter is 15 nm and geometric standard deviation is 1.6. The total number of particles is set to 1e5. This is done by writing

```
simulation_1.initialize('mu', 15e-9, 'sigma', 1.6, 'N', 1e5);
```

The name of variable is written first and after that its desired value. In addition, let us define the source and initial concentration of condensing vapor. If we want to have $10^7 \frac{molecules}{cm^3}$ of vapor in the beginning, we shall write

```
simulation_1.initialize('Cvap0', 1e7);
```

To define a constant source of condensing vapor, say $10^5 \frac{molecules}{cm^3}$, we write

```
simulation_1.initialize('gas_source', 1e5);
```

The last setting we are going to do is the time vector. If we want to run the simulation 10 hours (=10*3600 seconds) and we want the results for every 60 seconds, the time vector is set in the following way:

```
simulation_1.initialize('tvect', 0:60:10*3600);
```

These settings are enough for this simple example, because coagulation and dilution are on as default and we do not need to switch them on separately. The initialization can be done parameter by parameter as it is done here, but also all at once, by writing

```
simulation_1.initialize('mu', 15e-9,…
                        'sigma', 1.6,…
                        'N', 1e5,…
                        'Cvap0', 1e7,…
                        'gas_source', 1e5,…
                        'tvect', 0:60:10*3600);
```

The order of parameters is not relevant.

To start the simulation, type

```
simulation_1.run;
```

After this command, the time evolution of simulation is shown in Matlab command window. When the simulation is ready, some results can be studied by typing

```
simulation_1.plot;
```

This command plots the particle size distribution, vapor concentration and the total volume and number concentration of particles as a function of time. All data is stored in *simulation_1.output_data*, whereas the initial values are still found in *simulation_1.initials*.

If one wants to run several simulations, it is recommended to create several chamber objects, so that the results of previous run are not overwritten. For example, if two simulations will be run with different initial values, one should make two chamber objects, for example by typing

```
simulation_1 = chamber;
simulation_2 = chamber;
```

and defining initial conditions with *simulation_1.initialize()* and *simulation_2.initialize()* or by typing

```
simulation(2) = chamber;
```

and defining initial conditions with *simulation(1).initialize()* and *simulation(2).initialize()*.

2 Choosing the sectional method

The chamber model is based on sectional simulation where the size distribution is divided in a number of sections into which the particles are distributed. Coagulation and condensation/evaporation affect the particle size distribution, and in this model there are two alternative methods how these mechanisms are implemented. It is crucial to understand the difference between these methods, so that one can decide which of the methods one should use in different simulations.

The first method is called “moving sectional model”, where condensation/evaporation moves the corresponding diameters of sections and coagulation re-distributes particles between the sections. This method is the fastest and often the best way of the two methods to simulate behavior of particle distribution. This method, however, has its limits: particle sources cannot be used in moving sectional model, so for example modeling experiments where nucleation occurs is impossible. For simulations with nucleation, one has to use “fixed sectional model” which is described below.

In the fixed sectional model, the size distribution is divided in sections just as in moving sectional model. However, the corresponding diameters of sections do not move by condensation or evaporation, but stay fixed throughout the simulation. This allows the implementation of nucleation, because the section into which particles are nucleated does not change its corresponding diameter. This is a simple description of the model. In reality, the corresponding diameters of sections do move, but they have certain limits. The diameter grows by condensation, but if it reaches its upper limit, the particles from that section are moved to the next one and the diameter of the original section is reset to its original value. So the sections are not totally fixed, but they are fixed between their limits, and that is enough for implementation of particle sources (or nucleation). In addition, this is much more accurate method than totally fixed sections. The fixed sectional model is not as fast as moving sectional model, and in addition, the size distribution plots created by the fixed sectional model do not look as smooth as the ones created by the moving sectional model.

The choice between the two different methods is done by a parameter named *method*. If *method* is set to ‘*moving_sectional’* (by writing *simulation_1.initialize(‘method’,’moving_sectional’);*), the moving sectional model is used. This is the default setting. If *method* is *‘moving_center’*, the fixed sectional model with moving center diameters will be used.

3 Setting up vector parameters

Some of parameters in the chamber model can be defined as time-dependent. These parameters are dilution coefficient, vapor source and particle source. The time-dependent values are defined as matrices, where the first column is a time vector and the actual values are in the second column. Thus, the value in the second column corresponds to the time that is in the first column in the same row. The values between two time points are interpolated. An example of time-dependent vapor source vector is below. In that example, the vapor source is defined with different values till 240 seconds. Beginning from 300 seconds to the end (10*3600 seconds), the vapor source will be zero. It is important that both the first value of time vector (0 seconds in this case) and the last value (10*3600 seconds in this case) are defined in the vector so that it is possible to interpolate the values for all time points.

0	1e3
60	1e4
120	1.5e4
180	2e4
240	2.5e4
300	0
10*3600	0

The particle source matrix is an exception among other matrices because it has to be defined as 3-column matrix instead of 2-column matrix like the other time-dependent variables. This is because in addition to nucleation rate, also the size of nucleating particles must be defined. The size is defined in the third column and first row of the matrix, so the time-dependent particle source is of the form:

0	1.5	3e-9
60	1.5	NaN
120	1.5	NaN
180	1.5	NaN
240	1.5	NaN
300	0	NaN
10*3600	0	NaN

Particle source above will create 3 nm particles with source rate of 1.5 particles/cm^3s in the beginning and no particles at all after first 300 seconds. If one wants to define nucleation rates for different particle sizes, the particle source matrix must be a 3d-matrix. For example, if there is one particle source for 3nm particles and another for 6 nm particles, the source should be defined in the following way:

source1=

0	1.5	3e-9
60	1.5	NaN
120	1.5	NaN
180	1.5	NaN
240	1.5	NaN
300	0	NaN
10*3600	0	NaN

source2=

0	0	6e-9
60	0	NaN
120	0	NaN
180	0	NaN
240	0	NaN
300	0.3	NaN
10*3600	0.3	NaN

particle_source(:,:,1) = source1;
particle_source(:,:,2) = source2;
simulation_1.initialize('part_source', particle_source);

The initial particle distribution can be also defined as a vector. This is necessary when the initial distribution consists of two or more log-normal distributions. For example, if the initial distribution is a sum of two distributions, so that the first distribution has mu=10e-9, sigma=1.3 and N=1000 and the second mu=100e-9, sigma=1.4 and N=10 000, it can be defined in the following way:

```
simulation_1.initialize('mu', [10e-9, 100e-9], 'sigma', [1.3, 1.4], 'N', [1000, 10000]);
```

4 Automatizing simulations

One has often a need to run a number of simulations with different initial values. For that use, separate functions "*chamber_runfile*" and *"chamber_runfile2"* were implemented, and they are introduced in next sections.

4.1 Chamber_runfile

Function *chamber_runfile* reads a file that contains different initial settings and runs all the simulations in turn. In addition, this function saves the results of every simulation after it has been run, so that no data is lost even if one or more simulations were interrupted.

The settings file must be like the following

```
#
method = 'moving_center';
sedi_on = 0;
coag_on = 1;
Dp_min = -9;
Dp_max = -6;
tvect = 0:60:32400;
sigma = [1.25, 1.3];

N = [1e3 1e2];
mu=[50e-9, 140e-9];
dilu_on = 0;
sections = 30;
output_sections = 10*sections;
Cvap_const = 1;
Cvap0 = 2e7;

gas_source = 0;

#
method = 'moving_center';
sedi_on = 0;
coag_on = 1;
Dp_min = -9;
Dp_max = -6;
tvect = 0:60:32400;
sigma = [1.25, 1.3];

N = [1e3 1e2];
mu=[50e-9, 140e-9];
dilu_on = 1;
sections = 30;
output_sections = 10*sections;
Cvap_const = 1;
Cvap0 = 2e7;

gas_source = 0;
#
```

The script above defines the initial values for two simulations. The simulations are separated by "#" character, and the same character begins and ends the file. The only difference between these two simulations is that in the first one, dilution is switched off, whereas in the second one it is on.

The script is run by typing

```
[chamb, elapsed]=chamber_runfile(filename);
```

This reads first the file and checks that all definitions are correct. After that, the program runs the simulations and saves the results of each simulation to file *'temp_timestamp.mat'*, where timestamp is the current time (function *datestr(now,30)*). When all the simulations are successfully run, the program saves them all to file *'run_timestamp.mat'*. After that, the temporary files are deleted. If the program is for some reason interrupted before that, the temp files are not deleted. The *run_timestamp.mat* file contains all the results in an array of chamber objects named *'chamb'*. If there were two simulations defined in the definition script, the length of *chamb* is two: results of the first simulation is in *chamb(1).output_data* and the second in *chamb(2).output_data*. In addition, the *run_timestamp.mat* file contains an array named *elapsed*; *elapsed(1)* tells the time (in seconds) elapsed running the first simulation run and *elapsed(2)* the same for the second simulation run.

Things to bear in mind when using *chamber_runfile*:

- The settings file must start with "#" and end with "#". In addition, there must be "#" between two simulation definitions.
- You can address to variables defined in the settings file. For example one can first define the time vector *tvect* first, and then the gas source vector so that

  ```
  gas_source=[tvect', 10.*tvect']
  ```

  which means that the vapor source is directly proportional to time.
- You can define helper variables, that is, variables whose name does not correspond to any name in *chamb.initials*. However, the program will give a warning if this is done.
- You can use functions in the settings file. For example functions *ones()* and *zeros()* are useful when defining particle or vapor sources.

4.2 Chamber_runfile2

Another useful automatizing function is *chamber_runfile2*, which is in many ways similar to *chamber_runfile*: it reads a settings script and saves all the results to a file, using temporary files just as *chamber_runfile.* This function is especially handy when one wants to keep most of parameters constant and vary only some of them. In that case, the parameters that are varied are defined as vectors. An example of this is below

```
#
Dp_min = -9;
Dp_max = -6;
tvect = 0:60:32400;
sigma = [1.6];
mu=[50e-9];
sections = 25;
output_sections = 10*sections;
Cvap_const = 1;
dilu_on = 1;

Cvap0 = [1e6; 5e7];
N = [1e4; 1e5];
#
```

In the script above, the first ten definitions are constants, whereas the two last can have two values: *Cvap0* can be either 10^6 or 5×10^7, and *N* 10^4 or 10^5. The different values must be separated by semicolon. When this script is run by *chamber_runfile2*, it automatically simulates all the possible combinations of *Cvap0* and *N* (number of combinations is 2*2 = 4 in this case).

The distribution parameters can also have several different values, even when the distribution consists of two or more log-normal distributions. Consider a case where user wants to have a distribution that consists of two log-normal distributions, where *mu_1* = 10^{-9}, *mu_2* = 150×10^{-9} and *sigma_1* = 1.3 and *sigma_2* = 1.6. User wants to test three different values for the total number concentration in the distribution. This would be done by writing a following script:

```
#
mu = [10e-9, 150e-9];
sigma = [1.3, 1.6];

N = [1000, 1000; 1000, 10000; 10000, 1000];
#
```

As in the first example, also here the different values for *N* are separated with semicolon.

For time-dependent variables, the definition of several values is different. As the time dependent variables are matrices, it is not possible to separate values by writing them in separate rows. Instead, the matrices must be defined as 3d-matrices. For example, defining two different vapor source matrices is done by writing:

```
#
tvect = 0:60:10800;

gas_source(:,:,1) = [tvect', 10.*tvect'];
gas_source(:,:,2) = [tvect', 100.*tvect'];
#
```

The program will now run two simulations: in the first one, the vapor source is 10*time, and in the second one it is 100*time. Particle source and dilution coefficient must be defined in the same way. Even when the dilution coefficient is constant throughout the simulation, its different values can't be separated with semicolon, but the dilution coefficients have to be defined typing

```
#
dilu_coeff(:,:,1) = 1e-4;
dilu_coeff(:,:,2) = 5e-4;
#
```

A known limit when using *chamber_runfile2* is that it is not possible to define several simultaneous particle sources, because *chamber_runfile2* will see two definitions for *part_source* as two alternative definitions, not simultaneous.

Things to bear in mind when using *chamber_runfile2*

- If one wants to vary both dilution coefficient and whether the dilution is on at all, the *dilu_on* parameter should be kept as one all the time to avoid pointless simulation runs. For example, if the script is like following

  ```
  #
  dilu_on = [1; 0];
  dilu_coeff = [1e-4; 2e-4; 3e-4];
  #
  ```

  the program will run 2*3 = 6 simulations. In three of these simulations, the dilution is off, but dilution coefficient is varied. This is useless as the variation of dilution coefficient has no effect as the dilution is turned off. Instead, the script should be written:

```
#
dilu_coeff = [0; 1e-4; 2e-4; 3e-4];
#
```

Now there will be four runs in total, and the dilution is off (that is, *dilu_coeff* = 0) in one of them.

5 Advanced settings

When using the fixed sectional model, the center diameter of each section has the same value as the value of diameter vector in that section. The limit value between two sections is the geometric mean of these diameters. Figure 1 shows how the fixed sectional model works and what the different parameters mean.

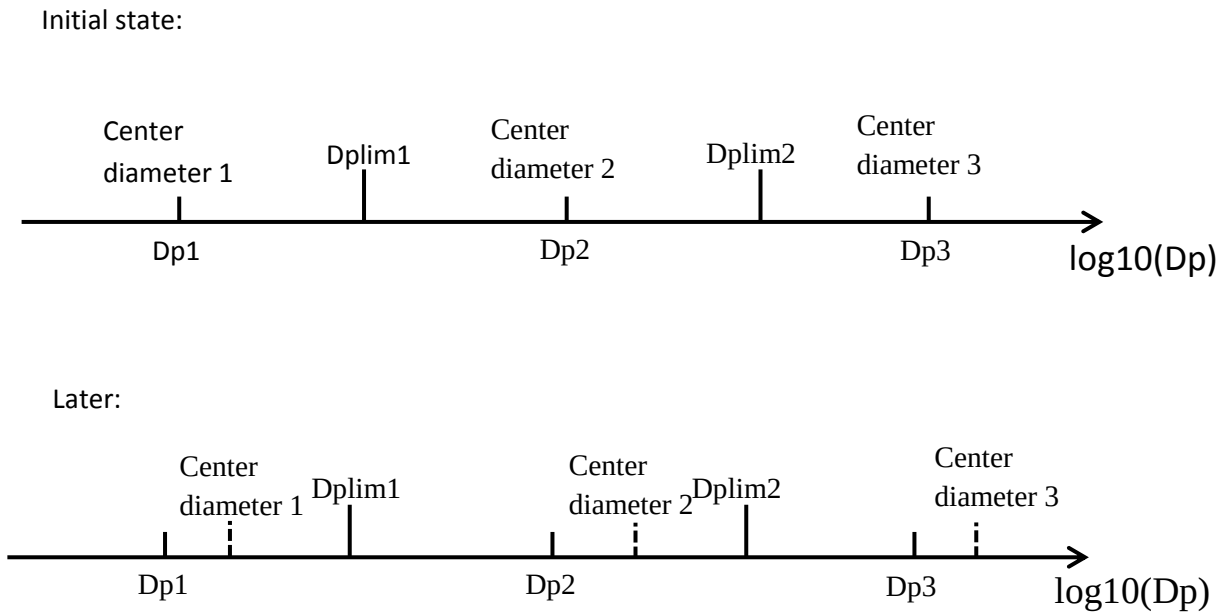

Figure 1: The upper figure shows the initial state and the lower the situation after particles (and the center diameter) have grown by condensation.

In figure 1, the upper figure is the initial state where the center diameters of sections equal the diameters in diameter vector (Dp1, Dp2 etc.). The limits between sections are geometric means of adjacent sections. When the particles grow by condensation, the center diameters of sections move as in the lower part of figure 1. If the center diameter crosses the limit of a section, the particles are moved to the next section.

The diameter vector is usually initialized so that it is logarithmically distributed. However, the user can define the diameter vector using a parameter named '*Dp'*. This parameter should be defined as a vector so that each cell of this vector represents a diameter of corresponding section. The limits are still calculated by the model unless user defines parameter 'Dplims'. The 'Dplims' parameter should be also a vector, so that i:s cell of 'Dplims' is bigger than i:s cell of diameter vector. The length of 'Dplims' should be one cell shorter than the diameter vector because the last section does not have upper limit. The initial center diameters can be defined, too, using parameter 'center_diameters'. In most cases it is better to define the initial diameter vector instead.

The initial particle distribution is by default defined as a log-normal distribution using parameters 'mu', 'sigma' and 'N'. However, also the particle distribution can be defined by user using parameter 'number_distr'. Each cell defines the particle concentration in corresponding section. Thus, the length of 'number_distr' should be of same length as diameter vector, or if diameter vector is not set by user, the number of cells in 'number_distr' should equal the amount of sections.

All the parameters introduced here can be used with fixed sectional model. For moving sectional model, only parameters ‘Dp’ and ‘number_distr’ can be used, as there are no center diameters or limits in moving sectional model.

6 Initial parameters

Parameter	Description	Default value
SWITCHES		
dilu_on	Defines whether the dilution is on or not. If dilu_on = 0, the aerosol will not dilute. If dilu_on = 1, the aerosol will dilute with rate defined by parameter dilu_coeff.	1
coag_on	Defines whether the coagulation is on or not. If coag_on = 0, the particles won't coagulate. If coag_on = 1, the coagulation is set on.	1
sedi_on	Defines whether the sedimentation is on or not. If sedi_on = 0, sedimentation is turned off. If sedi_on = 1, sedimentation will occur. Only usable for sedimentation in SAPPHIR chamber!	0
gas_source_is_vect	Defines whether the parameter gas_source is a vector or scalar. This cannot be set by user. Instead, the program checks if gas_source is an array or not and sets the value to 1 or 0 respectively.	0
dilu_vect_on	Defines whether the parameter dilu_coeff is an array or scalar. This cannot be set by user. Instead, the program checks if dilu_coeff is an array or scalar and sets the value to 1 or 0 respectively.	0
part_source_is_vect	Defines whether the parameter part_source is a time-dependent vector or a constant. This cannot be set by user. Instead, the program checks if part_source is an array or scalar and sets the value to 1 or 0 respectively.	0
coag_mode	Defines whether the particles coagulate 'normally' or agglomerate. Value should be either 'coag' for normal coagulation or 'aggl' for agglomeration. Agglomeration works only for particles in the free-molecule region. Condensation and deposition might not work correctly for agglomerates.	‘coag’
coag_num	Numerical representative of coag_mode. If coag_mode == 'coag' => coag_num = 1. If coag_mode == 'aggl' => coag_num = 0. This cannot be set directly by user, but the program sets it based on the value of coag_mode.	1
method	Defines whether the model will use fixed or moving sections. If method == ‘moving_sectional’, moving sections will be used. If method == ‘moving_center’, the moving center diameters with fixed sections will be used. When the model uses moving sections, the diameter defining the section grows or shrinks by condensation or evaporation. When using fixed sections, the diameters defining sections are constant. If particles grow by condensation so that they reach the limit diameter, they will be moved to the next section. The limit diameter is logarithmically halfway between adjacent sections.	0
max_timestep	Defines ode’s MaxStep value. If max_timestep == 0, ode’s MaxStep option will not be defined, so the step size will not be restricted. If max_timestep ~=0, ode’s MaxStep value will be the same as the value of max_timestep. See section 8, Known issues for more information.	0

<table>
<tr><th>Parameter</th><th>Description</th><th>Default value</th></tr>
<tr><td>Cvap_const</td><td>Defines whether the vapor concentration is constant or not.

If Cvap_const == 1, the vapor concentration stays at value Cvap0 during the whole simulation time, so that the value gas_source has no effect on vapor concentration.

If Cvap_const ~= 1, the vapour concentration is not kept constant, but its value depends on Cvap0 and gas_source.</td><td>0</td></tr>
<tr><td>vap_wallsink_on</td><td>Defines whether the vapor deposits on walls or not</td><td>0</td></tr>
<tr><td>BASIC VALUES</td><td></td><td></td></tr>
<tr><td>part_source</td><td>Particle source rate $\left(\frac{1}{cm^3 s}\right)$.

Particle sources can be defined as time-dependent vectors. The vector should be of following form:
<table>
<tr><td>t_0</td><td>source rate (1/cm^3/s)</td><td>particle size (m)</td></tr>
<tr><td>t_1</td><td>source rate (1/cm^3/s)</td><td>NaN</td></tr>
<tr><td>. . .</td><td>. . .</td><td>. . .</td></tr>
<tr><td>t_end</td><td>source rate (1/cm^3/s)</td><td>NaN</td></tr>
</table>
where the source rate in first row defines the source rate of particles at time t_0, the next row defines it at time t_1 etc. Particle size in first row defines the diameter of particles created; the values of column 3 in the rest of rows do not matter.

The time vector in the first column does not need to have same length as parameter tvect. However, the first and last element of part_source's first column must have the same values as respective elements of tvect. If the length of the array is different than tvect's length, part_source will be interpolated to same length.

If user wants to create particle sources of different sized particles, this can be done by defining part_source as a 3D-matrix. For example lines
part_source(:,:,1) =
<table>
<tr><td>0</td><td>1</td><td>3e-9</td></tr>
<tr><td>60</td><td>1</td><td>NaN</td></tr>
<tr><td>. . .</td><td>. . .</td><td>. . .</td></tr>
</table>
part_source(:,:,2) =
<table>
<tr><td>0</td><td>0.5</td><td>6e-9</td></tr>
<tr><td>60</td><td>0.5</td><td>NaN</td></tr>
<tr><td>. . .</td><td>. . .</td><td>. . .</td></tr>
</table>
will create one 3-nanometer particle/cm3 and 0,5 6-nanometer particles/cm3 every second during the simulation.</td><td>0</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

Parameter	Description	Default value
gas_source	The condensing vapor source rate $\left(\frac{1}{cm^3 s}\right)$. Can be defined as a scalar or two-column matrix. When defined as scalar, the source rate will be constant during the simulation. If *gas_source* is a matrix, it must have two columns; the first column is a time vector as in *part_source* and the second one tells the *gas_source* value at respective time. The time vector does not need to have same length as parameter tvect. However, the first and last element of gas_source's first column must have the same values as respective elements of tvect. If the length of the array is different than tvect's length, gas_source will be interpolated to same length.	1e5
dilu_coeff	Dilution coefficient $\left(\frac{1}{s}\right)$. Dilution affects particle concentration in following way: $\frac{dN}{dt} = -dilu\_coeff \times N$, where *N* is particle concentration. Parameter dilu_coeff can be either a scalar or an array. When defined as a scalar, dilution coefficient will be constant during the simulation. If dilu_coeff is an array, it must have two columns; the first column is a time vector and the second one tells the dilu_coeff value at respective time. The time vector doesn't need to have same length as parameter tvect. However, the first and last element of dilu_coeff's first column must have the same values as respective elements of tvect. If the length of the array is different than tvect's length, dilu_coeff will be interpolated to same length.	2.5342e-04
vap_wallsink	The flux of vapor molecules that condense on walls $\left(\frac{1}{s}\right)$. The change of vapor concentration by deposition on walls is then dCvap/dt = -vap_wallsink*Cvap.	0
satu_conc	The condensing vapor saturation concentration $\left(\frac{1}{cm^3}\right)$.	0
lambda	The condensing vapor mean free path (m).	1.2929e-07
diff_coeff	The condensing vapor diffusion coefficient $\left(\frac{cm^2}{s}\right)$.	0.1093
vap_molmass	Molecular mass of condensing vapor $\left(\frac{g}{mol}\right)$.	100
particle_dens	Density of particle matter $\left(\frac{g}{cm^3}\right)$.	1.8400
stick_coeff	Sticking coefficient. The probability that vapor molecules will stick to aerosol particles.	1.0
Cvap0	Initial condensing vapor concentration $\left(\frac{1}{cm^3}\right)$.	1e7
T	The temperature (K)	290
TIME VECTOR		
tvect	The time vector (seconds). Define as row vector. The vector spacing does not remarkably affect calculation time, but defines only the spacing of the results.	0:60:10800

Parameter	Description	Default value
DISTRIBUTION PARAMETERS		
N	Initial total particle concentration $\left(\frac{1}{cm^3}\right)$. If user wants to create a distribution that consists of several log-normal distributions, N can be defined as a vector. Then the total distribution will be a superposition of log-normal distributions. If N is defined as N = [1e3, 1e5], the first distribution has particle concentration N(1) (= 1e3), standard deviation sigma(1) and mean diameter mu(1) and the second has values N(2), mu(2) and sigma(2). The total particle concentration will then be N(1) + N(2). Note that each distribution must have values for sigma and mu, so if N is a vector, mu and sigma must be vectors of same length.	1e5
mu	The mean of the log-normal size distribution (m). If user wants to create a distribution that consists of several log-normal distributions, mu can be defined as a vector of length *n*. Then the total distribution will be a superposition of *n* log-normal distributions, so that each distribution is defined by N(i), mu(i) and sigma(i). Thus, if mu is a vector, N and sigma must be vectors of same length.	15e-9
sigma	Sigma (standard deviation) of log-normal size distribution. If user wants to create a distribution that consists of several log-normal distributions, sigma can be defined as a vector of length *n*. Then the total distribution will be a superposition of *n* log-normal distributions, so that each distribution is defined by N(i), mu(i) and sigma(i). Thus, if sigma is a vector, N and mu must be vectors of same length.	1.33
Dp_min	The exponent of minimum diameter of the size grid. The program will create a logarithmically spaced size grid between diameters 10^{Dp_min} and 10^{Dp_max}. The particle distribution will use this grid as x-axis and therefore Dp_min and Dp_max define limits inside which the distribution must stay. Thus, if particles for example grow during simulations, Dp_max must be defined high enough so that the distribution stays inside the limits.	-9

Parameter	Description	Default value
Dp_max	The exponent of maximum diameter of the size grid. The program will create a logarithmically spaced size grid between diameters 10^{Dp_min} and 10^{Dp_max}. The particle distribution will use this grid as x-axis and therefore Dp_min and Dp_max define limits inside which the distribution must stay. Thus, if particles for example grow during simulations, Dp_max must be defined high enough so that the distribution stays inside the limits.	-6
sections	The number of sections the model will use. The size grid is created between 10^{Dp_min} and 10^{Dp_max} and the number of grid points is defined by parameter 'sections'. The grid is logarithmically spaced. Particles are then placed in the sections according to their diameter. The bigger the number of sections, the more accurate the model will be. However, increasing the number of sections slows down the model.	30
output_sections	Defines the number of sections in output size grid. The $\frac{dN}{dlogD_p}$ distribution is interpolated to a denser grid after the simulation is finished. The number of grid points is defined by *output_sections*. The more output sections, the smoother the plot of size distribution will be.	300
Dp	Defines the diameters of sections. The number of sections will be the same as Dp's length. If this is used, the parameters Dp_min, Dp_max and sections are not in use.	0
number_distr	A vector that defines the particle concentration in each section. The length of number_distr must equal the number of sections. If this parameter is used, the parameters N, sigma and mu are not in use.	0
Dplims	Defines the limits between sections. Used only for fixed sectional model. The length must be one less than the number of sections, because the last section does not have upper limit and the first section does not have lower limit. Usually this vector is calculated by the model, but can be also defined by user.	0
center_diameters	Defines the initial center diameters of sections. Used only for fixed sectional model. The length must equal the number of sections. Usually this vector is calculated by the model, but can be also defined by user.	0
TOLERANCE PARAMETERS	Define the tolerance settings for ode45.	
Cvap_tol	Vapor concentration tolerance.	100
N_tol	Particle concentration tolerance.	0.01
Dp_tol	Particle diameter tolerance.	1e-10

7 Output data

The simulation results are found in *chamber.output_data.* Descriptions of the contents of *output_data* are in the table below.

Name	Description	Unit
Y	A matrix that contains the total number of particles, the number of particles for each section, the diameter of each section and the amount of particles and vapor that has diluted or lost to wall. Each row of matrix Y represents the values at a certain time point. Row 1 represents the values at time tvect(1) and so on. The first column of Y tells the vapor concentration. Columns 2 to (1+number of sections) contain the concentration of particles in the corresponding section. $\left(\frac{1}{cm^3}\right)$ Columns (2+number of sections) to (1+2*number of sections) tell the diameter of corresponding section. Units in meters. Column (2+2*number of sections) contains the amount of molecules lost to wall as aerosol. $\left(\frac{1}{cm^3}\right)$ Column (3+2*number of sections) contains the amount of molecules that have diluted as aerosol. $\left(\frac{1}{cm^3}\right)$ Column (4+2*number of sections) contains the amount of molecules that have diluted as vapor. $\left(\frac{1}{cm^3}\right)$ Column (5+2*number of sections) contains the amount of molecules lost to wall as vapor. $\left(\frac{1}{cm^3}\right)$	
distr_original	This matrix contains the dN/dlogDp distribution as a function of time. The first row is all zeros. Beginning from the second row, the first column contains the time in seconds, the second column tells the total particle concentration at that time and the rest of columns contain the particle concentration and diameter for each section. Particle concentrations for corresponding sections are found in columns 3 to (2+number of sections). Diameters of sections are found in columns (3+number of sections) to (2+2*number of sections).	
distr	Matrix distr is similar to distr_original, but here the data of distr_original is interpolated to denser grid, that is, there are more sections in distr. The number of sections in distr is chamber.initials.output_sections.	
vap	The vapor concentration for each time point. 1/cm^3	$\frac{1}{cm^3}$
tim	The time vector. Same as chamber.initials.tvect.	s

Ntot	Particle concentration for each time point. 1/cm^3	$\frac{1}{cm^3}$
Vtot	The total volume of particles for each time point. m^3/cm^3	$\frac{m^3}{cm^3}$
Dpmean	Mean particle diameter for each time point. Calculated by dividing Vtot by Ntot.	m
Mtot	Total mass of particles in the chamber as a function of time. Diluted particles or particles lost to wall are NOT counted in this mass. g/cm^3	$\frac{g}{cm^3}$
Mwall	Cumulative mass lost to wall as aerosol for each time point. g/cm^3	$\frac{g}{cm^3}$
Mdilu	Cumulative mass diluted as aerosol for each time point. g/cm^3	$\frac{g}{cm^3}$
Mvdilu	Cumulative mass diluted as vapor for each time point. g/cm^3	$\frac{g}{cm^3}$
Mvwall	Cumulative mass lost to wall as vapor for each time point. g/cm^3	$\frac{g}{cm^3}$
CMD	Count median diameter of particles as a function of time.	m

8 Functions

The list of functions (or methods) of chamber object and some information about them. For more information on the functions, type

```
help chamber.name_of_function
```

1. *chamber.initialize('param_1', value_1)*
   Sets the initial value of *param_1* as *value_1*. In addition, the function checks whether the *value_1* is in correct form. Function *initialize* can be used to set several parameters at once, by typing

   ```
   chamber.initialize('param_1', value_1, 'param_2', value_2);
   ```

   and so on.
2. *chamber.run;*
   Starts the simulation and shows the time evolution in command window.
3. *chamber.plot;*
   Plots three figures. There are three subplots in the first figures: the uppermost subplot shows the $\frac{dN}{dlogD_p}$ distribution as a function of time, the next one the total volume as a function of time and the lowermost the total particle concentration as a function of time. The second figure is a plot of vapor concentration as a function of time and the third one shows the $\frac{dN}{dlogD_p}$ distribution in a separate figure.
   chamber.plot('dist') plots only the distribution.
   chamber.plot('dist', 'original') plots only the original distribution, not the interpolated one.
   chamber.plot('original') plots the same figures as *chamber.plot*, but the distributions plotted are the original ones instead of interpolated.
4. *chamber.copy;*
   Makes a copy of the chamber object. For example, if one wants to copy all the data from chamber object *simulation_1* to *simulation_2*, this should be done by typing

   ```
   simulation_2 = simulation_1.copy;
   ```

   If the same operation is done by writing

   ```
   simulation_2 = simulation_1;
   ```

   the later changes to *simulation_1* affect also to *simulation_2*. For example if *simulation_1* is deleted by typing *delete(simulation_1)*, also the data in *simulation_2* will be erased.
5. *chamber.set_params();*
   Used to set initial parameters. This function should not be used for setting initial parameters by user, but instead *chamber.initialize()* should be used. This is because *set_params* does not check the correctness of parameters.
6. *chamber.check_initials;*
   This function checks the correctness of initial parameters. The function *initialize* runs first *set_params* and afterwards *check_initials.* For normal use, one does not need to run this function separately, because it is already run in *chamber.initialize.*

9 Known issues

- The simulation may not work correctly when using fixed sections and a time vector that is not divisible by the interval. For example setting time vector as

```
tvect = 0:60:3600;
```

or

```
tvect = 2:60:3602;
```

works, but setting it as

```
tvect = 2:60:3600;
```

may not work, but can cause an error message:

```
Error using odearguments (line 22)
When the first argument to ode45 is a function handle, the tspan argument must
have at least two elements.
```

- The differential solver used in this program is ode45. Ode45 may encounter a tolerance error which produces the following error message:

```
Failure at t=5.184000e+04.  Unable to meet
integration tolerances without reducing the
step size below the smallest value allowed
(1.164153e-10) at time t.
```

This error can be caused by big and sudden changes in vapor or particle concentration or just big values of these sources. If this error occurs and vapor source or particle source are defined as functions of time, consider smoothing up these functions by using for example sigmoid function instead of step function. If this error occurs in the beginning of the simulation, try defining the first cell of particle source as zero, so that there is no nucleation at the first time point. Another trick is to set the initial particle concentration to a small, nonzero value (for example 10^{-5}), if it was zero before. Decreasing the amount of sections may also help.

- Sometimes, when ode's time step grows large, the differential solver may miss a short nucleation event or short-duration vapor source. In that case, the setting *'max_timestep'* should be set to nonzero value. This defines ode's *'MaxStep'* option to the same value as *'max_timestep'*. The smaller the value of *max_timestep* is, the slower the simulation will be. The maximum value of *max_timestep* for most problematic cases seems to be four times the time vector's spacing.
- When using fixed sectional model, the time vector must have constant spacing, so for example logarithmic time scale cannot be used.
- Simulation of agglomerating particles assumes that the condensation onto agglomerates is similar to condensation onto spherical particles of same volume. That is, the agglomeration affects only the coagulation process and nothing else.