F Solutions

1. Chapter 1

(a) Using the template file provided, the formulas required to calculate the volume and diameter ratios can be implemented as follows (and found in the file Answer_1_1.py)

```
V_rat = np.power((d_Nb/d1),3.0/(Nb-1.0)) # Volume ratio between
    bins
d_rat = V_rat**(1.0/3.0) # Diameter ratio between bins
```

Likewise one can finish the loop expressions to populate an array of diameters using the following:

```
d_i=np.zeros((Nb), dtype=float)
d_i[0]=d1
for step in range(Nb):
    if step > 0:
       d_i[step]=d_i[step-1]*d_rat
```

The diameter width of each size bin is calculated using:

```
d_width = d_i*np.power(2,1.0/3.0)*((np.power(V_rat,1.0/3.0)-1.0)/
          (np.power(1+V_rat,1.0/3.0))) # Diameter width array of
    size bins
```

To finish, the parameters of the log normal distribution and thus discretized formulation are given as:

```
# Parameters of log-normal distribution
sigmag1 = np.log(1.7) # Geometric standard deviation
mean1 = np.log(0.15) # Mean particle size [150nm]
# Seperate out the probability density function
distribution_1 = (np.exp(-(log_di - mean1)**2 / (2 * sigmag1**2))
     /
     (sigmag1 * np.sqrt(2 * np.pi)))

Ntot = 600.0 # Total number of particles [per cm-3]
# Use pre-calculated distribution_1 to implement equation 1.16
N_dist = Ntot*(distribution_1*(d_width/d_i)) # Discrete number
    distribution
```

(b) We begin by initialising the measured concentrations of the particle population and calculating the corresponding moments:

```
# Initialize Number, Surface Area, and Volume Concentrations
N = 2300. * 1.0E6  #particles m-3
S = 2.144e-4       #m2 m-3
M = 27.2           #ug m-3

# Calculate Moments of Modal Distribution
V  = M*1.0e-9 / 1400 # m3 m-3
M3 = V / np.pi * 6 # m3 m-3
M2 = S / np.pi     # m2 m-3
```

Then we calculate the median diameter μ and the standard deviation σ_g of the mode:

```
# Calculate and print properties of log-normal mode
```

```
sigma = np.exp( np.sqrt( 1./3. *np.log(N) \
                -np.log(M2) + 2./3. * np.log(M3) ) )
mu = ( M3 / N / \
       np.exp( 9./2. * (np.log(sigma))**2 ) ) **(1./3.)
```

(c) With the modal properties calculated in Exercise 2, we can set the lower and upper bounds of the diameter range we are interested in and calculate the number, surface area, and mass concentrations for that range. To do this we integrate the number concentration (Nslice) using Eq. 1.32.

```
# Define lower and upper limit of the size distribution [
    microns]
Dlo = 50e-9
Dhi = 400e-9

# Calculate Number Concentration between Dlo and Dhi
mug = mu
Nslice = N * 0.5 * \
        ( math.erf((np.log(Dhi) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma)))   \
         -math.erf((np.log(Dlo) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma))) )
```

Then we convert the number median diameter μ to the surface area and volume median diameters using equations 1.36 and 1.37, respectively, and integrate those expressions with equation 1.32 to get the surface area concentration (Sslice) and volume concentration, which is then converted to mass concentration (Mslice).

```
# Calculate Surface Area Concentration between Dlo and Dhi
mug = np.exp( np.log(mu) + 2 * np.log(sigma)**2 )
Sslice = S * 0.5 * \
        ( math.erf((np.log(Dhi) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma)))   \
         -math.erf((np.log(Dlo) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma))) )

# Calculate Mass Concentration between Dlo and Dhi
mug = np.exp( np.log(mu) + 3 * np.log(sigma)**2 )
Vslice = V * 0.5 * \
        ( math.erf((np.log(Dhi) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma)))   \
         -math.erf((np.log(Dlo) - np.log(mug))/  \
                   (np.sqrt(2)*np.log(sigma))) )
Mslice = 1400 * Vslice * 1.0e9 # ug m-3
```

2. Chapter 2

(a) Code listing A.3 displays a suggested portion of code found in the file Answer_2_1.py that initialises a moni-disperse population with number density 300 $cm.^{-3}$ and diameter of 150nm. Please note that in this example we use a variable num_bins even though we only have 1 bin. This is to help reaffirm the developments needed to expand to a poly-disperse case. When running the file Answer_2_1.py you are told our initial absorptive mass is 0.74 $\mu g.m^{-3}$ and the final secondary mass is 1.98 $\mu g.m^{-3}$.

```
# Define a monodisperse size distribution
# Assume each particle starts with an involatile core
```

```
# of absorptive organic with a mass of 200 g/mol and
# density 1400 km/m3. We store this information in an array
# as 'core'. This will ensure, in this example, that we do
# not get 100% evaporative loss

# Define total number of particles [per cc]
N_total = 300.0

# We carry an inert and involatile in each size bin
core = np.zeros((num_bins), dtype=float)
core_abundance = np.zeros((num_bins), dtype=float)
density_core = np.zeros((num_bins), dtype=float)
core_mw = np.zeros((num_bins), dtype=float)
density_core[:] = 1400.0 #kg.m-3

# The number of particles is only carried in one bin
N_per_bin = np.zeros((num_bins), dtype=float)
N_per_bin[0] = N_total

# Define the diameter of our particles [microns]
size_array = np.zeros((num_bins), dtype=float)
size_array[0] = 0.150 #microns

# Use the size to now calculate a concentration of a 'core' in
    molecules / cc
# This aligns with the units used for volatile species
core_abundance[0] = (N_per_bin[0])*((4.0/3.0)*
                    np.pi*np.power(size_array[0]*0.5e-6,3.0)*
                    density_core[0]*1.0e3)

# What is our existing dry mass in micrograms per cubic metre?
dry_mass = np.sum((core_abundance)*(1.0e12))

print("Initial mass = ", dry_mass)
```

Listing A.3 Creating a mono-disperse representation to initialise an existing absorptive mass

(b) In code listing A.4 we simply modify the value of variable density_core to 1200. Running the file Answer_2_1.py now outputs initial absorptive mass is 0.63 $\mu g.m^{-3}$ and the final secondary mass is 1.96 $\mu g.m^{-3}$.

```
# We carry an inert and involatile in each size bin
core = np.zeros((num_bins), dtype=float)
core_abundance = np.zeros((num_bins), dtype=float)
density_core = np.zeros((num_bins), dtype=float)
core_mw = np.zeros((num_bins), dtype=float)
density_core[:] = 1400.0 #kg.m-3
```

Listing A.4 Modifying the density of a mono-disperse population

(c) In code listing A.5 we add a new line to the end of our script where we use the equilibrium value for our secondary mass, COA_final, to calculate the partitioning coefficients of every volatility bin. Running the file Answer_2_1.py now outputs the values of ε_i indicating that, for the highest volatility bin, less than 0.2 % partitions from the gas to the condensed phase.

```
# We can also calculate the partitioning coefficients for all
    volatility bins
epsilon = np.power(1.0+(Cstar/(COA_final+dry_mass)),-1.0)
print("Partitioning coefficients = ", epsilon)
```

Listing A.5 Outputting the values for partitioning coefficients at the end of the simulation

(d) File Answer_2_2.py contains a suggested solution to this exercise, with two modifications highlighted in code listings A.6 and A.7. In the first, we create a new Numpy array that holds all of our factors f_i. In the second, as we already calculate the mole fraction of each compound in solution, we create a new array that holds our activity coefficients according to the prescribed formula, and then multiply these to the existing expression to arrive at the equilibrium pressure above droplets through variable Pressure_eq. By running this modified file, without and with the inclusion of activity coefficients, you will find the final secondary mass changes from 1.82 to 1.67 $\mu g.m^{-3}$ respectively, representing a ~ 8 % decrease in predicted secondary mass.

```
# Additional array to include hypothetical activity coefficients
gamma_factor = np.zeros((num_species), dtype=float)
gamma_factor[0]= 3.0
gamma_factor[1]= 3.0
gamma_factor[2]= 3.0
gamma_factor[3]= 4.0
gamma_factor[4]= 5.5
gamma_factor[5]= 6.5
gamma_factor[6]= 7.0
gamma_factor[7]= 10.0
gamma_factor[8]= 15.0
gamma_factor[9]= 20.0
```

Listing A.6 Creating a new array of factors used to generate hypothetical activity coefficients

```
        # Calculate a hypothetical contribution from non-ideality
        activity_coeffs = 1.0+gamma_factor*mole_fractions
        # Calculate the equilibrium concentration [molecules/cc]
        Pressure_eq=kelvin_factor*mole_fractions*P_sat*
    activity_coeffs
```

Listing A.7 Creating a new array of factors used to generate hypothetical activity coefficients

3. Chapter 3

The complete source code for solutions to the first three exercises of chapter 3 is provided in archive file chap3_alg19_Exercises_BAT_Program_Solutions.zip, which includes a Visual Studio project. To aid referencing with the exercises we thus retain the same file naming convention.

(a) The solutions to problem 1 are based on a modified version of BAT_example_2; see the code in file chap3_alg16_BAT_exercise_1.f90. This solution can be reproduced by selecting exercise_no = 1 in Prog_BAT_exercises.

(a) The produced BAT light plots for the activities and the normalized Gibbs energy of mixing curves indicate that

LLPS is not predicted to occur in this case.

This is because no local maxima or minima are found in the mixed composition space. However, we can see that this binary mixture containing

1,6-hexanediol is clearly behaving nonideally and comes close to a case where LLPS would occur. Running code section 2.7 confirms this observation; variable phase_sep_detected returns "false".

(b) Increasing the O:C ratio by +15 % to ∼ 0.38333 in code section 2.1 of BAT example 2 results in no LLPS.

> Decreasing the O:C ratio by 15 % to 0.28333 results in LLPS. The extent of the LLPS is marked by the limits $x_{org}^{\alpha} \approx 0.0351$, $x_{org}^{\beta} \approx 0.382$ (rounded values).

(c) Yes, variations of the organic molar mass matter and they also impact the estimated organic component density.

> For M_{org} increased by 15 % to 135.91 g mol^{-1}, LLPS is predicted. Rounded to three significant figures, the miscibility gap limits are $x_{org}^{\alpha} \approx 5.48 \times 10^{-2}$, $x_{org}^{\beta} \approx 2.29 \times 10^{-1}$. For a decreased M_{org} to 100.45 g mol^{-1}, a single phase is predicted to be stable.

(d) Based on the results from (b) and (c), we know that in this system LLPS occurs only in two of the tested cases of organic compound property variations: either a 15 % increase in the M_{org} or a 15 % decrease in the O:C ratio. Among those two, the O:C decrease leads to the wider miscibility gap.

> Qualitatively, with increasing O:C ratio, a miscibility gap is expected to become smaller or to disappear. This is expected since a higher O:C ratio makes the organic compound more polar and this tends to favour miscibility with water.

This can be confirmed by consulting the plots generated from code sections 2.5 and 2.6. In the case of molar mass changes, an increase in M_{org} leads to a wider difference in the molar masses and a smaller M_w/M_{org} ratio value, which impacts the BAT model in several ways, e.g. refer to equations (3.14) and (3.15). It also increases the organic-to-water density ratio and tends to increase the organic volume fraction at a given mole fraction.

> Qualitatively, increasing M_{org} makes the organic compound less similar in size compared to water, which leads to an increase in the Gibbs energy (and entropy) of mixing. Hence, we would expect a larger deviation from ideal mixing, making phase separation more likely.

(b) The precise values for this exercise 2 depend on a few parameter choices that were not further specified in the problem statement. Therefore, your results may slightly differ from those stated here; see also the table below.

> For np = 15 and lower and upper bounds on the critical O:C value set as 0.0 and 0.9, the determined critical O:C ratio and LLPS range are: OtoC crit ≈ 0.3245 and $x_{org}^{\alpha} \approx 0.09833$, $x_{org}^{\beta} \approx 0.2094$.

One approach for solving this problem is outlined in the following. Determining the critical value of organic O:C ratio for which LLPS just occurs means

we need to write a procedure that allows us to determine the onset of LLPS as function of O:C. We can write our solution code based on the code from subroutine BAT_example_2. See the example solution code in file chap3_alg17_BAT_exercise_2.f90. This solution can be reproduced by selecting exercise no = 2 in Prog BAT exercises.

* First, we could remove code sections 2.5 to 2.6 from the code template, since these sections are unnecessary in this case.
* Given the limited resolution by which we characterize the activity curves (np = 15) and considering code section 2.7, which determines whether LLPS is present based on presence or absence of a local minimum or maximum of the activity curves, we can rely on that existing approach for LLPS onset detection.
* One option is to write a function that evaluates the BAT model at np points for every input of O:C ratio for our specific case, similar to the combination of code sections 2.1, 2.3 and 2.7. The output of the function is arbitrarily set to +1.0 if LLPS is detected and −1.0 if LLPS is not detected. The goal is then to determine the critical O:C value for LLPS onset, that is, the O:C value for which our step function seemingly crosses zero (within the numerical tolerance, tol, set). In the associated solution code, you find an implementation of such a function as fnc_LLPS_det at the end of module Mod_BAT_model (declared public in that module).
* To provide our function fnc_LLPS_det with additional inputs about the organic component and the number of points for the computation of activity curve data, we have extended the customized type foptions in module Mod_NumPrec_Types to include type components np and HtoC. This makes our function a bit more flexible in terms of accepting other component properties and resolutions, if we would like to use those in future.
* Now we need to evaluate function fnc_LLPS_det systematically for different input O:C values to determine the critical value for which our function switches sign (crosses zero). This could be achieved by a simple bisection method. In this example solution, we employ Ridders' method via function Ridders_zero from Mod_NumMethods, which essentially carries out a bisection in this step function case. Lines 50 to 56 of subroutine BAT_exercise_2 show the relevant code for applying this method and solving for the critical O:C value. As shown there, we need to provide an initial guess for lower and upper bounds of OtoC_crit; here we set them as 0.0 and 0.9. Note that the choice of these bounds, together with the number of points np evaluated, will affect the accuracy of the detected critical O:C ratio. We also set the numerical tolerance of the method to be at maximum 10^{-5} in terms of resolving the O:C value.
* Lines 61 to 66 within subroutine BAT_exercise_2 show that after solving for OtoC_crit, we need to ensure that the found value refers to an O:C that results in LLPS (and not just slightly outside the LLPS onset). If not, we adjust the OtoC_crit by the tolerance value to arrive at a LLPS case.
* Tabulated below are the values obtained with the outlined procedure. In addition, using our solution approach, we can explore the effect of increasing the resolution in terms of np and other parameters. Such higher-resolution results are also provided. As the number of points for LLPS detection is

increased and the numerical tolerance tol used with Ridders' method reduced, we find that the LLPS extent narrows toward $x_{org}^{\alpha} = x_{org}^{\beta} \approx 0.147$. That is, a tiny miscibility gap. Note that this convergence to a single point is expected for a critical point in a phase diagram.

np	O:C bounds		O:C$_{crit}$	tol	LLPS extent	
	low	high			x_{org}^{α}	x_{org}^{β}
15	0.0	0.9	3.2453×10^{-1}	10^{-5}	9.83255×10^{-2}	2.09377×10^{-1}
15	0.2	0.5	3.2454×10^{-1}	10^{-5}	9.83630×10^{-2}	2.09315×10^{-1}
30	0.0	0.9	3.2894×10^{-1}	10^{-5}	1.27030×10^{-1}	1.68851×10^{-1}
300	0.0	0.9	3.2966×10^{-1}	10^{-5}	1.43702×10^{-1}	1.50249×10^{-1}
3000	0.0	0.9	3.2966×10^{-1}	10^{-5}	1.44682×10^{-1}	1.49245×10^{-1}
10000	0.0	0.9	3.2966×10^{-1}	10^{-5}	1.44682×10^{-1}	1.49245×10^{-1}
10000	0.0	0.9	3.2967×10^{-1}	10^{-7}	1.46881×10^{-1}	1.47017×10^{-1}

(c) For solving exercise problem 3, we can again use subroutine BAT_example_2 from the BAT application example 2 as our template; see the code in file chap3_alg18_BAT_exercise_3.f90. The outlined solution can be reproduced by selecting exercise_no = 3 in Prog_BAT_exercises.

(a) Given the sum formula ($C_{10}H_{22}O_7$) for the organic compound, we can determine the molar mass, O:C ratio, and H:C ratio for use in the density estimation and the BAT model. Running subroutine BAT_example_2 will generate a plot of the logarithm of the activity coefficients (among other outputs). We find that in this case no LLPS is predicted to occur, so the equilibrium state output is simply for the single-phase case. We can also create an additional plot to show the activity coefficients of water and the 2-methyltetrol dimer compound on a non-logarithmic scale. Following the example of the existing plot (f) in code section 2.11, code for generating a plot (g) has been added near the end of subroutine BAT_exercise_3. The plot of the activity coefficients (see Figure A.3) shows that

> the largest deviations from ideal mixing occur in the approximate mole fraction range of $0.02 < x_{org} < 0.2$.

Recall that ideal mixing would imply activity coefficients equal to 1.0.

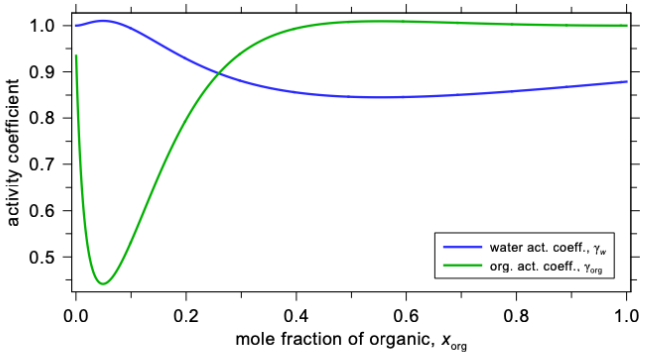

Figure A.3 activity coefficients as a function of mole fraction of organic.

As shown graphically, the 2-methyltetrol dimer exhibits a larger activity coefficient range. For a more precise answer, we can add code to output the maximum and minimum values of the activity coefficients.

> For water, γ_w varies from 1.01 at $x_{org} = 0.049$ to 0.845 at $x_{org} = 0.553$, while γ_{org} varies from 0.441 at $x_{org} = 0.049$ to 1.01 at $x_{org} = 0.553$.

(b) The equilibrium gas-phase mass concentration, C_j^g, can be computed according to modified Raoult's law: $C_j^g = C_j^\circ x_j^\alpha \gamma_j^\alpha = C_j^\circ a_j^\alpha$. Therefore, to solve this problem we need to compute the activities for $x_{org} = 0.3$ with BAT and we need to determine the pure-component liquid-state saturation vapor pressures of water and the organic at the given temperature of 20 °C. We could look up experimental data for the pure-component (liquid-state) vapor pressures of water and the 2-methyltetrol dimer; however, for this organic compound finding such data is unlikely. Instead, let us use estimation methods. For the organic, we use the provided SMILES descriptor of the molecular structure and visit the UManSysProp online model for an estimation. In case of the 2-methyltetrol dimer, using the EVAPORATION method at $T = 293.15$ K, the online model returns $\log_{10}(p^\circ/\text{atm}) \approx -13.99114$. Converted to SI units, this corresponds to $p_{org}^\circ \approx 1.034 \times 10^{-9}$ Pa. For water, we could look up experimental data or use a parameterization. From the NIST chemistry webbook (https://webbook.nist.gov), we find for water under "phase change data" a vapor pressure parameterization in the form of the Antoine equation, application of which yields (after unit conversion) a value of $p_w^\circ \approx 2336.7$ Pa at 293.15 K. Applying in each case the ideal gas law, using the molar mass of the component, and converting to units of $\mu g\,m^{-3}$, we determine that $C_{org}^\circ \approx 1.079 \times 10^{-4}\ \mu g\,m^{-3}$ and $C_w^\circ \approx 1.73 \times 10^7\ \mu g\,m^{-3}$. Thus, at this temperature water would be classified as volatile, while the 2-methyltetrol dimer is of extremely low volatility (an ELVOC).
Using modified Raoult's law for $x_{org} = 0.3$, we determine

$C_w^g \approx 1.064 \times 10^7\ \mu g\,m^{-3}$ and $C_{org}^g \approx 3.044 \times 10^{-5}\ \mu g\,m^{-3}$.

For comparison, you can also compute the equilibrium gas phase concentrations when assuming ideal mixing in the liquid phase. The $C_j^{g,ideal}$ values are greater than the corresponding ones computed for nonideal mixing, meaning that for $x_{org} = 0.3$ this (nonideal) binary system is predicted to have a lower total equilibrium vapor pressure (and gas phase mass concentration).

This represents a negative deviation from Raoult's law.

Note that this result was already clear from the activity coefficient curves shown in the plot generated for part (a), since both activity coefficients are less than 1.0 at the mole fraction of interest.

(d) For solving exercise 4, we make use of the provided AIOMFAC-LLE project source code and the input file. Copy the file (input_0606.txt) into the directory /Inputfiles, then modify the source code near line 89 of the main program in file Main_AIOMFAC_LLE_prog.f90 to state
txtfilein = './Inputfiles/input_0606.txt'. This will use the specified input file by default, which will make working on this exercise easier.

(a) Running the AIOMFAC-LLE program a first time with the input file, we can generate equilibrium phase composition output as well as a set of plots (from subroutine OutputLLE_plots) with the use of the Dislin library. Inspecting those figures reveals that this system will undergo LLPS near a water activity of $a_w \approx 0.89$. For example, see the plot of mass fraction of water versus a_w generated in plot (c) of OutputLLE_plots and shown in the top panel of Figure A.4.

For the calculation and plotting of phase-specific pH values as shown in the bottom panel of Figure A.4, we can write a block of code similar to that for plot (c). Since we need the molal activity of the H^+ ion, we need to first identify the species number for H^+ to locate it within the output data array. This can be done using the findloc function and knowing that H^+ is identified as ion (subgroup) number 205 in AIOMFAC. Line 5 of Listing A.8 achieves this task.

```
1  block !for chapter 3, exercise 4 solution;
2    integer :: iH
3    !(f) Plot equil-state phase pH vs. water activity (for
4    !    each phase in LLPS case), where applicable:
5    iH = findloc(int(out_data_A(7,1,:)), VALUE=205, DIM=1)
6    !iH = index location of H+ ion
7    if (iH > 0) then
8      call add_plot_xydata(xv=out_data_A(6,:,1), &
9        & yv=-log10( out_data_A(6,:,iH) ), &
10       & ltext='pH of phase $\alpha$', &
11       & pen_wid=8.0_wp, lstyle='solid', plot_symb='curve')
12
13     if (LLPSpresent) then
14       call add_plot_xydata(xv=out_data_B(6,iloc1:iloc2,1), &
15         & yv=-log10( out_data_B(6,iloc1:iloc2,iH) ), &
16         & ltext='pH of phase $\beta$', &
17         & pen_wid=8.0_wp, lstyle='dashed_medium', &
18         & plot_symb='curve')
19     endif
20
21     !set overall plot properties and generate plot:
```

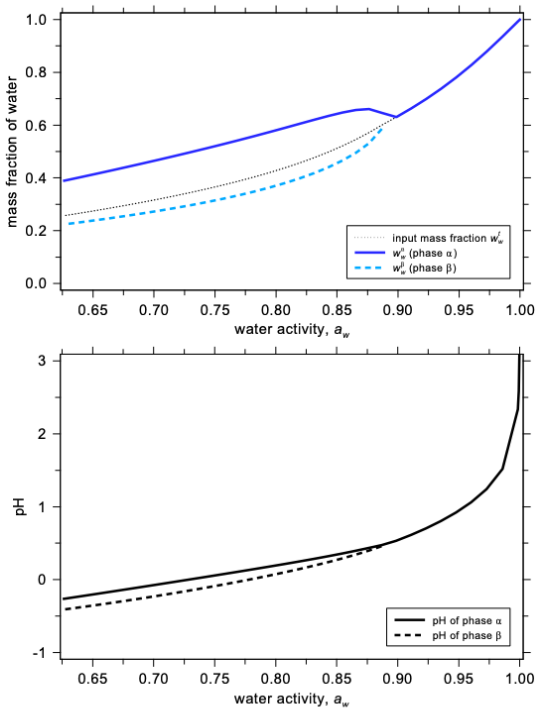

Figure A.4 Predicted water content (top) and pH (bottom) of the system from exercise 4a.

```
    xlabel = 'water activity, $a_w$'
    ylabel = 'pH'
    yax_lim = [-1.0_wp, 3.0_wp]
    call dislin_plot(xlabel, ylabel, yaxis_mod=0.6_wp, &
      & yaxis_limits=yax_lim, legend_position=3, &
      & metafile='pdf', out_file_name=trim(fname)//'_pH')
  endif
end block
```

Listing A.8 Plotting code for phase-specific pH from AIOMFAC-LLE output; (snippet from file Mod_InputOutput.f90).

Alternatively, one could use the tabulated molal H^+ activity reported in the AIOMFAC-LLE output file (such as AIOMFAC-LLE_output_0606.txt) to create a plot with your software of choice.

The pH plot (Figure A.4) shows that in the LLPS case, phases α and β are of similar but not identical pH. A difference in pH implies a difference in the activities: $a_{H^+}^{(m),\alpha} \neq a_{H^+}^{(m),\beta}$, which seemingly violates the isoactivity condition

for H^+ in the LLE case. So, how can this result be correct?
To resolve this apparent contradiction, we need to recall the definition of the chemical potential of electrolyte components (see Section 3.1.1.2).

> From that definition and the fact that chemical potentials of neutral cation–anion combinations (i.e., electrolyte components like H_2SO_4; see Equation 3.7) are equal across phases in the LLE case, we can conclude that the pH difference must arise due to the use of the molal single-ion activity of H^+ in the definition of pH.

In other words, molal activities of single ions may differ among coexisting liquid phases (due to differences in the local electrostatic potentials between phases), while the LLE condition (only) requires that the ion activity products of any neutral cation–anion combinations must match across phases. The distinct pH values in LLE phases α, β are therefore the result of a quirk in the definition of pH.
Note that the difference on the pH scale is usually small. However, had we simply used the H^+ molalities from the two phases for an estimation of each pH (i.e., ignoring the activity coefficient), the resulting differences would tend to be larger (as you can confirm from the output for this example).

(b) For part (b), we create a new AIOMFAC-web input file (see attached file `input_0607.txt` by deleting the organic components and their input mole fraction columns from file `input_0606.txt`. Set line 89 in `Main_AIOMFAC_LLE_prog` to use the new file. Running AIOMFAC-LLE for this system shows that no LLPS will form in the covered composition range, which in this case is limited to $a_w > 0.76$. For a fair comparison of the pH values from (b) with the organics-containing system from (a), we select output data taken at similar water activities, meaning this would correspond to comparing aerosols of the given dry compositions at approximately the same equilibrium relative humidities. The following table provides a comparison at five distinct a_w levels.

	with organics (a)		w/o organics (b)
a_w	pH α	pH β	pH
0.999	2.59	-	2.34
0.95	0.92	-	0.91
0.90	0.53	-	0.58
0.85	0.36	0.29	0.36
0.80	0.18	0.06	0.20

For comparable water activity levels, the predicted pH from the organics-free system is very similar to that of the inorganic-rich phase α from the system with organics.

Given the same inorganic dry composition, this result is perhaps not too surprising. It suggests that in these systems the main control over the pH at a given RH stems from the aqueous inorganic electrolyte components. Moreover, beyond answering the question from problem 4b, one could hypothesize that the pH is in these cases primarily controlled by a buffering effect due to the partial dissociation of aqueous bisulfate. We can test this idea by computing the bisulfate dissociation degree in the liquid phases. The molal dissociation degree is defined by [e.g. 38]:

$$\alpha_{HSO_4^-} = 1 - \frac{m_{HSO_4^-}}{m^{max}_{HSO_4^-}}. \tag{A.4}$$

Figure A.5 shows a plot of the bisulfate dissociation degrees for both systems. The code for this plot is shown near the end of subroutine OutputLLE_plots in file chap3_alg20_Mod_InputOutput.f90.

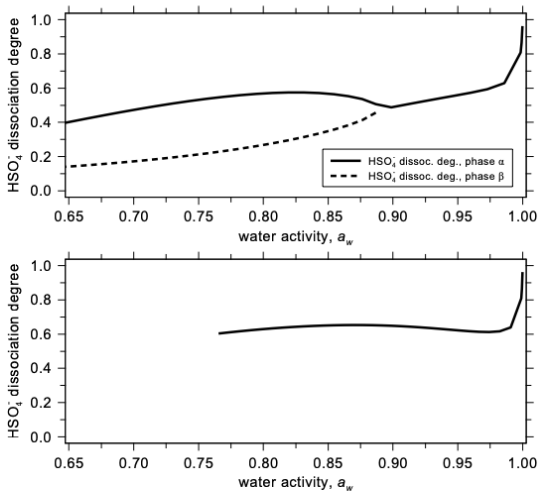

Figure A.5 Predicted bisulfate dissociation degrees for the system from exercise 4a (top) and the organic-free system (4b, bottom).

The predicted bisulfate dissociation degree is greater than 0.1 and less than 1.0, particularly so in the inorganic-rich phases, in which it varies from about 0.4 to 0.8 for water activities between 0.65 and 0.99. This confirms that HSO_4^- is present in substantial amounts and that there is room for a shift in the

bisulfate dissociation, allowing it to partially buffer the pH via the established bisulfate–sulfate equilibrium. The precise dissociation degree depends on the interplay of pH, the sulfate, and the bisulfate activities; thus, the pH can still vary within a limited range, as shown in Figure A.4.

4. Chapter 4

(a) File Answer_4_1.py provides a suggested solution to this exercise that is based on the structures described in the main body of text. In this file we have built on the initial structures used in Chapter 2 to initialised a VBS model. However, as code listings A.9 and A.10 show, we first specify the concentration of Ozone and construct a set of RHS expressions that comprise our function **dy_dt** that we pass to our ODE solver. In that function we explicitly specify the rate expressions for each reaction, pointing to the appropriate index of array concentration. For example, the rate of the second order reaction between the highest volatility bin and ozone refers to the 9th cell in concentration as the 10th component. Please note that we also do not pass the value of variable ozone_concentration. Since we define this constant value prior to our function definition within the same script, this acts as a global variable that can be accessed within our function. When you run the file Answer_4_1.py, you should produce Figure A.6 provided below.

```
# Initialise concentration of a gas phase oxidant in ppb
ozone = 10.0
Cfactor= 2.46310e+10 #ppb to molecules/cc
# convert this to  molecules / cc
ozone_concentration = ozone*Cfactor
# Define a rate coefficient between every volatility bin and
    ozone
rate_coefficient = 1.0e-15
```

Listing A.9 Specifying the concentration of Ozone and rate coefficient for each second order reaction

```
# RHS function [with ozone as a global variable]
def dy_dt(array,t):

    # Calculate the rate of each reaction explicitly
    # rate_1 = ozone_abundance*array[0]*rate_coefficient
    rate_1 = ozone_concentration*array[1]*rate_coefficient
    rate_2 = ozone_concentration*array[2]*rate_coefficient
    rate_3 = ozone_concentration*array[3]*rate_coefficient
    rate_4 = ozone_concentration*array[4]*rate_coefficient
    rate_5 = ozone_concentration*array[5]*rate_coefficient
    rate_6 = ozone_concentration*array[6]*rate_coefficient
    rate_7 = ozone_concentration*array[7]*rate_coefficient
    rate_8 = ozone_concentration*array[8]*rate_coefficient
    rate_9 = ozone_concentration*array[9]*rate_coefficient

    # Now write out the RHS for each volatity bin according to
    our simple
    # mechanism
    dy_dt_array = np.zeros((num_species), dtype=float)
    dy_dt_array[0] = rate_1
    dy_dt_array[1] = rate_2 - rate_1
    dy_dt_array[2] = rate_3 - rate_2
    dy_dt_array[3] = rate_4 - rate_3
```

```
    dy_dt_array[4] = rate_5 - rate_4
    dy_dt_array[5] = rate_6 - rate_5
    dy_dt_array[6] = rate_7 - rate_6
    dy_dt_array[7] = rate_8 - rate_7
    dy_dt_array[8] = rate_9 - rate_8
    dy_dt_array[9] = -1.0*rate_9

    return dy_dt_array
```

Listing A.10 Modifying the RHS function to account for the second order reactions redistributing material between volatility bins

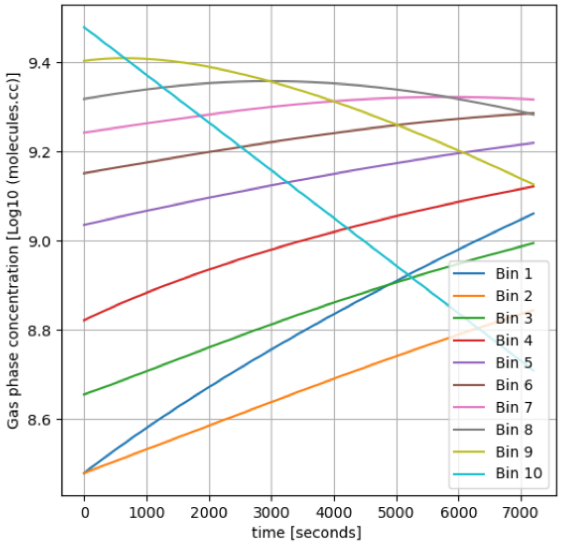

Figure A.6 Gas phase concentration as a function of time.

(b) In this exercise we need to modify our function **dy_dt** to account for the variable loss and gain of both Ozone and the highest volatility bin. As with the previous exercise we provide a suggested solution to this within the file Answer_4_2.py. Code listing A.11 highlights the first set of modifications. Not only do we specify the initial concentration of Ozone, but we expand our array concentration to include Ozone so we can track and modify how the concentration changes over time. Whilst we specify the initial concentration in ppb, we convert this to *molecules.cm*$^{-3}$. We also then specify an

emission rate for both ozone and the highest volatility bin through variables emission_rate_ozone and emission_rate_bin respectively. Note that, as we use units of $molecules.cm^{-3}$ in our RHS function, we also convert the units provided from $(\mu g.m^{-3}.h^{-1})$ to $molecules.cm^{-3}.s^{-1}$.

```
# Initialise concentration of material in each volatility bin [
    micrograms/m3]
concentration = np.zeros((num_species+1), dtype = float)
concentration[0] = 0.1
concentration[1] = 0.1
concentration[2] = 0.15
concentration[3] = 0.22
concentration[4] = 0.36
concentration[5] = 0.47
concentration[6] = 0.58
concentration[7] = 0.69
concentration[8] = 0.84
concentration[9] = 1.0

# Initialise concentration of a gas phase oxidant in ppb
ozone = 10.0
Cfactor= 2.46310e+10 #ppb-to-molecules/cc
ozone_concentration = ozone*Cfactor
# Now convert to micrograms/m3
ozone_concentration = (ozone_concentration/(NA*1.0e-6))*(mw_array
    [10]*1.0e6)
concentration[10] = ozone_concentration # Ozone
```

Listing A.11 Expanding the array concentration to include 11 components with the addition of ozone so we can track variable concentrations

In code listing A.12 we highlight the changes made to our function **dy_dt**. Notice now that all rates are calculated by referencing the 11th component of array. We also modify the *dy_dt* expressions for ozone and the highest volatility bin through the expressions that define dy_dt_array[9] and dy_dt_array[10] respectively. If you run file Answer_4_2.py you will produce the Figure A.7 provided below.

```
def dy_dt(array,t):

    # Calculate the rate of each reaction explicitly
    # We now track the concentration of ozone in our array
    rate_1 = array[10]*array[1]*rate_coefficient
    rate_2 = array[10]*array[2]*rate_coefficient
    rate_3 = array[10]*array[3]*rate_coefficient
    rate_4 = array[10]*array[4]*rate_coefficient
    rate_5 = array[10]*array[5]*rate_coefficient
    rate_6 = array[10]*array[6]*rate_coefficient
    rate_7 = array[10]*array[7]*rate_coefficient
    rate_8 = array[10]*array[8]*rate_coefficient
    rate_9 = array[10]*array[9]*rate_coefficient

    # Now write out the RHS for each volatity bin according to
    our simple
    # mechanism
    dy_dt_array = np.zeros((num_species+1), dtype=float)
    dy_dt_array[0] = rate_1
    dy_dt_array[1] = rate_2 - rate_1
    dy_dt_array[2] = rate_3 - rate_2
    dy_dt_array[3] = rate_4 - rate_3
    dy_dt_array[4] = rate_5 - rate_4
```

```
    dy_dt_array[5] = rate_6 - rate_5
    dy_dt_array[6] = rate_7 - rate_6
    dy_dt_array[7] = rate_8 - rate_7
    dy_dt_array[8] = rate_9 - rate_8
    dy_dt_array[9] = -1.0*rate_9 + emission_rate_bin
    # Define the dydt for Ozone
    dy_dt_array[10] = emission_rate_ozone - (rate_1+rate_2+rate_3
        +rate_4+rate_5+rate_6+rate_7+rate_8+rate_9)

    return dy_dt_array
```

Listing A.12 Modifying the RHS function to include 11 components. Includes the addition of ozone with non zero emission rate for this oxidant and the highest volatility bin

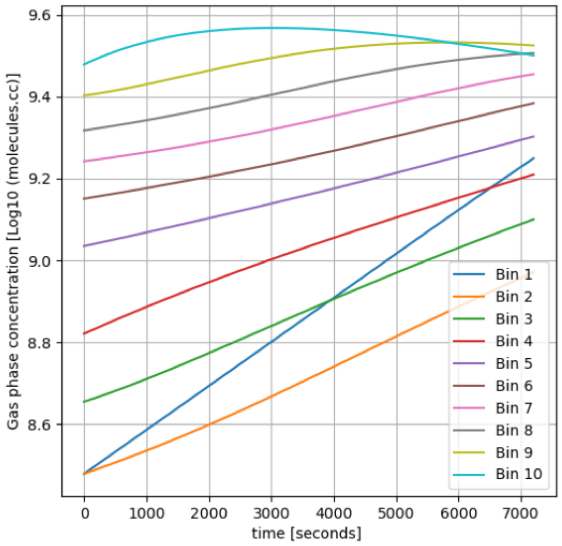

Figure A.7 Gas phase concentration as a function of time.

(c) File Answer_4_3.py provides a suggested solution where we specify two separate RHS functions **dy_dt1** and **dy_dt2** in one file. Each includes the structure discussed in the solutions to Exercise 1 and 2 respectively. As highlighted in code listing A.13, we create two separate solutions by separately solving each set of differential equations.

```
solution1 = odeint(dy_dt1, gas_concentration, t, rtol=1.0e-6,
    atol=1.0e-4, tcrit=None)
```

```
solution2 = odeint(dy_dt2, gas_concentration, t, rtol=1.0e-6,
    atol=1.0e-4, tcrit=None)
```

Listing A.13 Creating two seperate solutions to a gas phase only model

By running file Answer_4_3.py you will produce the Figure A.8 shown below, where the ratio of the highest volatility bins is generated by dividing the 10th column of each solution array (solution1[:,9]/solution2[:,9]).

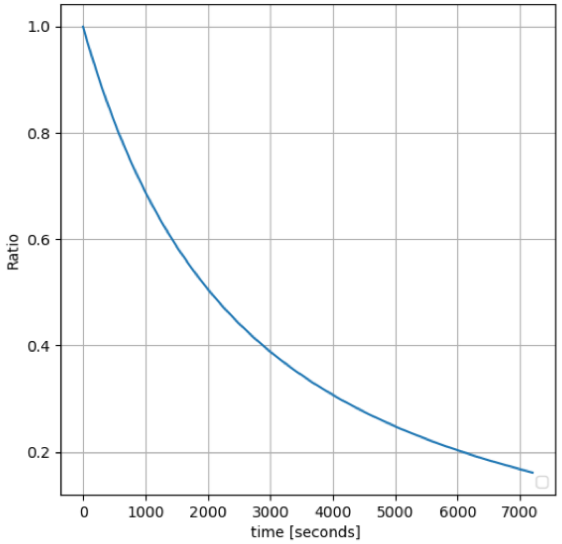

Figure A.8

(d) This is a difficult exercise and requires some thought about how we modify the structure of our numerical arrays. Whilst we have ran simulations for an evolving gas phase in isolation, now we need to couple this to a moving sectional distribution. The difficult factor here is that we have a gas phase compound (Ozone) that has variable concentrations throughout the simulation but does not partitioning to the condensed phase. If you recall, in Figure 2.5 (which is repeated below as Figure A.9), we designed our numerical arrays so that the first entries represented the concentration of compounds in the gas phase. In file Answer_4_4.py we add an additional cell to store the concentration of ozone. However, we do not expand the cells that store the concentration of each VBS

compound in the condensed phase. Rather, we modify how we extract a slice from our numerical array (array) in our RHS function **dy_dt**. Where we have defined the properties of our gases, we likewise need to ensure we only select the first 10 entries where we predict gas to particle partitioning. This includes variables mw_array (molecular weight), gamma_gas (mean free path), and DStar_org (diffusion coefficients). After running the file Answer_4_4.py you should find the final predicted secondary mass is 3.7 $\mu g.m^{-3}$.

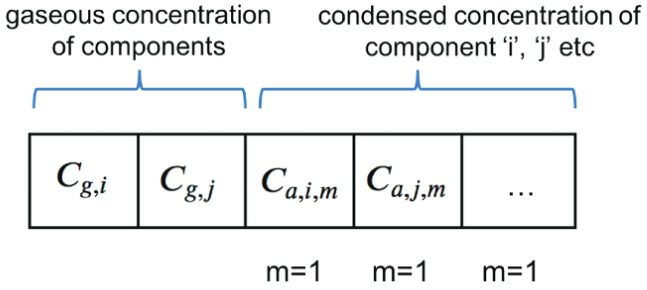

Figure A.9 Schematic illustration of a numerical array ordered in such a way to hold the concentrations of each specie in the gas phase ($C_{g,i}$,$C_{g,j}$, etc.) and then the concentration of each specie in each size bin in the preceding cells ($C_{a,i,m=1}$,$C_{a,i,m=2}$....$C_{a,j,m=1}$, etc.).

(e) To remove the second moment from code snippet 4.17, we simply remove the calculation of $\frac{dM_2}{dt}$ and shrink the size of the time-dependent solution vector.The total condensed mass without ozonolysis reaction decreases from 1.98 to 1.93 $\mu g\, m^{-3}$ and decreases that of the ozonolysis reaction from 3.29 to 3.07 $\mu g\, m^{-3}$.

```
# Parameters of log-normal distribution
sigmag1 = np.log(1.7) # Natural Log of Geometric standard
    deviation
Dp      = 150 * 1.0E-9 # Geometric Mean particle diameter [150nm]
N       = 100.0 # Total number of particles [per cm-3]

# Initialize all 3 moments of initial distribution
M0 = N * 1.0E6  # [particles per m-3]
M2 = M0 * Dp**2 * np.exp( 2 * sigmag1**2 )   # [m2 m-3]
M3 = M0 * Dp**3 * np.exp( 4.5 * sigmag1**2 ) # [m3 m-3]
# ------------------------------------------------------

# -------- Now initialise a core abundance using the above size
    distribution -------
# Use the 3rd moment to now calculate a concentration of a 'core'
     distribution in molecules / cc
density_core = 1400.0  # kg m-3
core_mw      = 200.0   # g mol-1

dry_mass = np.pi/6.0 * density_core * M3 * 1.0E9 # ug m-3
print("Initial dry aerosol mass = ", dry_mass)
core_abundance = dry_mass * 1.0E-6 / core_mw * Avo * 1.0E-6  # [
    molec cm-3]
print("Initial core abundance = ", core_abundance )

```

```
# New define an array that holds the molecular abundance of each gas and its concentration
# in each aerosol mode [molecules / cc]
# Add 3 additional terms for each mode at the end representing M0, M2, and M3
array = np.zeros(( num_species + num_species + 2 ), dtype=float)
array[ 0:num_species ] = gas_abundance
array[ num_species:2*num_species ] = 1.0e-10 # assuming we start with no condensed
                                             # material in the aerosol phase
nchem = num_species + num_species
array[nchem     ] = M0
array[nchem + 1 ] = M3

########################################
########################################
# RHS function [with no ozone effects]
def dy_dt(array,t):

    # Load gas species concentrations in local array
    Cg_i_m_t = array[0:num_species]

    # We are working with 1 aerosol mode, which has an involatile core
    dy_dt_array = np.zeros(( nchem + 2 ), dtype='float64')
    dM0_dt = 0.
    dM3_dt = 0.

    # 1) Calculate total moles and mole fraction of each species in this mode,
    #    and assign modal parameters
    temp_array      = array[ num_species:nchem ]          # [molec/cc]
    total_moles     = np.sum(temp_array) + core_abundance # [molec/cc]
    mole_fractions = temp_array / total_moles            # [frac]

    M0 = array[ nchem   ]  # 0th moment = Number
    M3 = array[ nchem+1 ]  # 3rd moment prop. to volume

    # 2) Calculate density
    density_array = np.zeros((num_species+1), dtype='float64')
    density_array[0:num_species] = density_org[0:num_species] # [kg m-3]
    density_array[num_species]   = density_core               # [kg m-3]

    mass_array = np.zeros((num_species+1), dtype='float64')
    mass_array[0:num_species] = (temp_array/Avo) * mw_array   # [g cm-3]
    mass_array[num_species] = (core_abundance/Avo)*core_mw    # [g cm-3]

    total_mass = np.sum(mass_array)                           # [g cm-3]
    mass_fractions_array = mass_array / total_mass            # [frac]
    density = 1.0 / (np.sum( mass_fractions_array / density_array )) # [kg m-3]

    #3) Calculate Diameter and sigma from the three moments above.
    Dp = ( M3 / ( M0 * np.exp( 4.5 * sigmag1**2 ) ) ) **(1.0/3.0) # [m]
```

```
    M1 = M0 * Dp * np.exp( 0.5 * sigmag1**2 )   # [m m-3]

    #4) Free Molecular Regime Calculation for the 2nd and 3rd moments
    Ifm3 = np.pi * alpha_d_org * mean_them_vel / 4. * M2  # [ m3 m-3 s-1]

    #5) Continuum Regime Calculation for the 2nd and 3rd Moments
    Ict3 = 2.0 * np.pi * DStar_org[:]*1.0e-4 * M1   # [m3 s-1 m-3]

    #6) Combine Ifm and Ict for moment- and species-dependent Beta correction terms
    Beta3 = ( Ifm3 * Ict3 ) / ( Ifm3 + Ict3 )  # [m3 m-3 s-1]

    #7) Modal growth equation
    Cstar_i_m_t = ((Cstar*1.0e-6)/(mw_array))*1.0e-6*Avo

    dm3_dt = 6.0 / np.pi * np.sum( (Cg_i_m_t - Cstar_i_m_t*mole_fractions) * 1.0e6 / rho_l * Beta3 )
    dy_dt_array[ num_species:nchem ]  = ( Cg_i_m_t[:] - Cstar_i_m_t*mole_fractions) * Beta3
    dy_dt_array[ 0:num_species ]  = -1.0 * dy_dt_array[ num_species:2*num_species]
    dy_dt_array[ nchem      ] = 0.  # Particle number does not change. This is included in the ODE
                                    # set for demonstration since it would change for other processes
                                    # like coagulation. It should be removed for operational use if
                                    # only condensation is being considered.
    dy_dt_array[ nchem + 1 ] = dm3_dt

    #print( dy_dt_array[num_species:nchem]/Avo*mw_array*1.0e6 )
    return dy_dt_array
#######################################

#######################################
# RHS function [with  ozone effects]
def dy_dt_ozone(array,t):

    # Load gas species concentrations in local array
    Cg_i_m_t = array[0:num_species]

    # We are working with 1 aerosol mode, which has an involatile  core
    dy_dt_array = np.zeros(( nchem + 2 ), dtype='float64')
    dM0_dt = 0.
    dM3_dt = 0.

    # Calculate the rate of each reaction explicitly
    # rate_1 = ozone_abundance*array[0]*rate_coefficient
    rate_1 = ozone_abundance*array[1]*rate_coefficient
    rate_2 = ozone_abundance*array[2]*rate_coefficient
    rate_3 = ozone_abundance*array[3]*rate_coefficient
    rate_4 = ozone_abundance*array[4]*rate_coefficient
    rate_5 = ozone_abundance*array[5]*rate_coefficient
    rate_6 = ozone_abundance*array[6]*rate_coefficient
    rate_7 = ozone_abundance*array[7]*rate_coefficient
    rate_8 = ozone_abundance*array[8]*rate_coefficient
    rate_9 = ozone_abundance*array[9]*rate_coefficient

    # Now write out the RHS for each volatity bin according to our simple
```

```
    # mechanism
    dy_dt_array[0] = rate_1
    dy_dt_array[1] = rate_2 - rate_1
    dy_dt_array[2] = rate_3 - rate_2
    dy_dt_array[3] = rate_4 - rate_3
    dy_dt_array[4] = rate_5 - rate_4
    dy_dt_array[5] = rate_6 - rate_5
    dy_dt_array[6] = rate_7 - rate_6
    dy_dt_array[7] = rate_8 - rate_7
    dy_dt_array[8] = rate_9 - rate_8
    dy_dt_array[9] = -1.0*rate_9

    # 1) Calculate total moles and mole fraction of each species in this mode,
    #    and assign modal parameters
    temp_array      = array[ num_species:nchem ]          # [molec/cc]
    total_moles     = np.sum(temp_array) + core_abundance # [molec/cc]
    mole_fractions = temp_array / total_moles             # [frac]

    M0 = array[ nchem   ]  # 0th moment = Number
    M3 = array[ nchem+1 ]  # 3rd moment prop. to volume

    # 2) Calculate density
    density_array = np.zeros((num_species+1), dtype='float64')
    density_array[0:num_species] = density_org[0:num_species] # [kg m-3]
    density_array[num_species]   = density_core               # [kg m-3]

    mass_array = np.zeros((num_species+1), dtype='float64')
    mass_array[0:num_species] = (temp_array/Avo) * mw_array   # [g cm-3]
    mass_array[num_species] = (core_abundance/Avo)*core_mw    # [g cm-3]

    total_mass = np.sum(mass_array)                           # [g cm-3]
    mass_fractions_array = mass_array / total_mass            # [frac]
    density = 1.0 / (np.sum( mass_fractions_array / density_array ))  # [kg m-3]

    #3) Calculate Diameter and sigma from the three moments above.
    Dp = ( M3 / ( M0 * np.exp( 4.5 * sigmag1**2 ) ) ) **(1.0/3.0)  # [m]
    M1 = M0 * Dp * np.exp( 0.5 * sigmag1**2 )   # [m m-3]

    #4) Free Molecular Regime Calculation for the 2nd and 3rd moments
    Ifm3 = np.pi * alpha_d_org * mean_them_vel / 4. * M2  # [ m3 m-3 s-1]

    #5) Continuum Regime Calculation for the 2nd and 3rd Moments
    Ict3 = 2.0 * np.pi * DStar_org[:]*1.0e-4 * M1   # [m3 s-1 m-3]

    #6) Combine Ifm and Ict for moment- and species-dependent Beta correction terms
    Beta3 = ( Ifm3 * Ict3 ) / ( Ifm3 + Ict3 )  # [m3 m-3 s-1]

    #7) Modal growth equation
    Cstar_i_m_t = ((Cstar*1.0e-6)/(mw_array))*1.0e-6*Avo
```

```
    dm3_dt = 6.0 / np.pi * np.sum( (Cg_i_m_t - Cstar_i_m_t*
    mole_fractions) * 1.0e6 / rho_l * Beta3 )
    dy_dt_array[ num_species:nchem ]  = ( Cg_i_m_t[:] -
    Cstar_i_m_t*mole_fractions) * Beta3
    dy_dt_array[ 0:num_species ]  = dy_dt_array[0:num_species] -
    dy_dt_array[num_species:2*num_species]
    dy_dt_array[ nchem      ] = 0.  # Particle number does not
    change. This is included in the ODE
                                    # set for demonstration since
    it would change for other processes
                                    # like coagulation. It should
    be removed for operational use if
                                    # only condensation is being
    considered.
    dy_dt_array[ nchem + 1 ] = dm3_dt

    return dy_dt_array
```

Listing A.14 Simplifying the modal condensation problem to two moments

(f) Expanding the code in 4.17 and 4.18 to consider the case of two particle modes both receiving condensed vapour mass requires multiple adjustments throughout both solutions. First, the vector containing the time-dependent variables must nearly double in length to track the aerosol phase composition of the mode, the mode's number and its third moment. In practice, you do not need to have a place in the solution vector for number concentration since it is assumed to not change during condensation. We also do not need to predict the direct change in M3, as all species-dependent mass concentrations could be summed up to get the total mass of the system.

(g) in the following code listings we highlight the required developments that are provided in the file Answer 4 5.py:

(i) Solution to part 1

```
1    [#6]
2    [OX2;H1][CX4]
3    [#6][CX3](=[OX1])[OX2;H1]
```

Here within the .smarts file the first column contains the value for k, followed by the corresponding SMARTS in the second column. The first line gives the sMARTS for the total number of carbons, the second line for total number of hydroxyl groups bonded to non-aromatic carbon and the third line for the total number of carboxylic acid groups.

(ii) Solution to part 2

```
SIMPOL_SMARTS = _read_smarts('4_5_SIMPOL.smarts')
```

Here a variable has been defined that uses the _read_smarts function to read in the SMARTS contained within 4_5_SIMPOL.smarts.

(iii) Solution to part 3

```
def simpol_groups (compound):
    m = matches(SIMPOL_SMARTS, compound)
    result = {}
    # Zeroth group (constant used in all calculations)
    result['0'] = 1
```

```
    # total number of carbons present
    result['1'] = m[1]
    # Hydroxyl group bonded to NON-aromatic carbon
    result['2'] = m[2]
    # carboxylic acid
    result['3'] = m[3]

    return result
```

Here the function simpol_groups is defined. Within this function, the matches function is used to count how many occurrences of each SMARTS there are within each input pybel object. result = {} creates an empty dictionary that is populated by the rest of the function. result['0'] adds an entry to the dictionary where the key is '0' corresponding to a value of 1. This is the zeroth group within SIMPOL and is a constant used in all calculations. result['1'] adds an entry to the dictionary where the key is '1' corresponding to a value of the total number of matches to the SMARTS for the total number of carbons. result['2'] and result['3'] are the same for the number of hydroxyl groups bonded to non-aromatic carbon and the number of carboxylic acid groups respectively.

(iv) Solution to part 4

```
SIMPOL_1 = _read_data('4_5_SIMPOL.data', value_col=1)
SIMPOL_2 = _read_data('4_5_SIMPOL.data', value_col=2)
SIMPOL_3 = _read_data('4_5_SIMPOL.data', value_col=3)
SIMPOL_4 = _read_data('4_5_SIMPOL.data', value_col=4)
```

Here variables have been defined that uses the read data function to read in the 4 model parameters used in SIMPOL contained within 4_5_SIMPOL.data. value_col specifies which column to look for each of the 4 model parameters within 4_5_SIMPOL.data

(v) Solution to part 5

```
def simpol_vapour_pressure(compound, temperature):
    m = simpol_groups(compound)

    b1 = aggregate_matches(m, SIMPOL_1)
    b2 = aggregate_matches(m, SIMPOL_2)
    b3 = aggregate_matches(m, SIMPOL_3)
    b4 = aggregate_matches(m, SIMPOL_4)

    bkT = b1 / temperature + b2 + b3 * temperature + b4 * log(
    temperature)
    return bkT
```

Here the function to calculate vapour pressure using SIMPOL is defined. Within this function the simpol_groups function that was defined earlier within the exercise is called. This will create a dictionary containing keys for each group, and values for how many times each group occurs within each compound. The variable b1 uses the aggregate matches function to calculate the contribution of each group to the first model parameter of SIMPOL. b2 calculates the contribution for the second model parameter of SIMPOL and so on. Finally bkT is the implementation of equation 4.23 to calculate the vapour pressure of each compound.

5. Chapter 5

(a) (i) The solution can be found in file Answer_5_1.py which will produce the figure A.10.

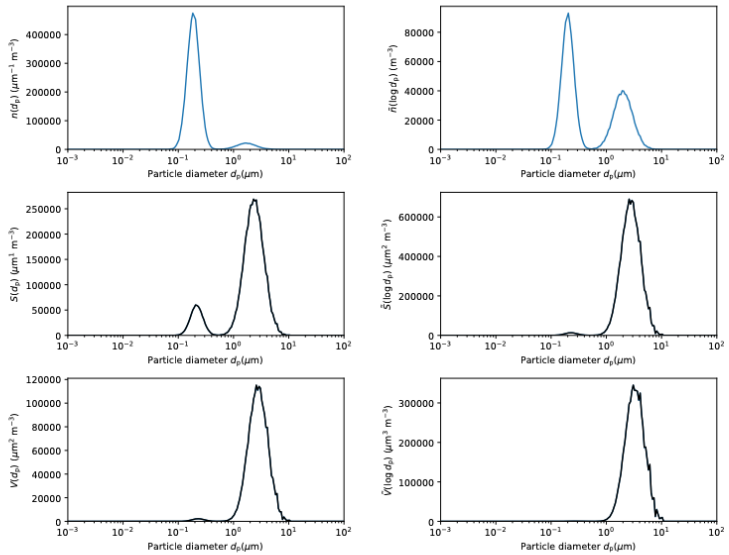

Figure A.10

(ii) The three modes can be described by the following parameters:

Mode	Geo. median diameter D_{pg} (m)	Geo. standard deviation σ_g	Number concentration N (m^{-3})
A	3×10^{-8}	1.3	1×10^{9}
B	4×10^{-7}	1.8	5×10^{8}
C	2×10^{-6}	2.0	2.5×10^{8}

To visually estimate these parameters, you can do the following: The geometric mean diameter of each mode is easy to read off. It is the diameter where the maximum of the curve is reached. To read off σ_g, find the two diameters where the curve is $\exp(-0.5)$ times its peak height. Call these diameters d_L and d_R. Then $\sigma_g = \exp(\log(d_R) - \log(d_{pg}))$ or $\sigma_g = \exp(\log(d_{pg}) - \log(d_L))$ both should give the same value for σ_g. To get the total number concentration, read off the peak height value of the curve and multiply by $\sqrt{2\pi}\log\sigma_g$.

(iii) —

(b) (i) Read K off Figure 5.1 (approximate value is fine for this question).

$$N = \frac{10^{11}}{4}\, m^{-3}$$

$$K \approx 10^{-9}\, cm^3\, s^{-1} = 10^{-15}\, m^3\, s^{-1}$$

$$\frac{dN}{dt} = -KN^2 = -6.25 \times 10^5\, m^{-3}\, s^{-1}$$

Right at the start of the experiment, there are 6.25×10^5 coagulation events per second.

(ii)

$$t = \frac{N}{dN/dt} = 1.6 \times 10^5\, s = 44\, h$$

(iii) It will take longer in reality because coagulation will slow down as the number concentration decreases.

(iv) The initial rate decreases by a factor of 4, so it'll take longer for the number concentration to decrease.

(v) Nothing changes (since the number concentration does not change).

(c) (i) The solution can be found in file `Answer_5_3_a.py` that produce the figures A.11 and A.12.

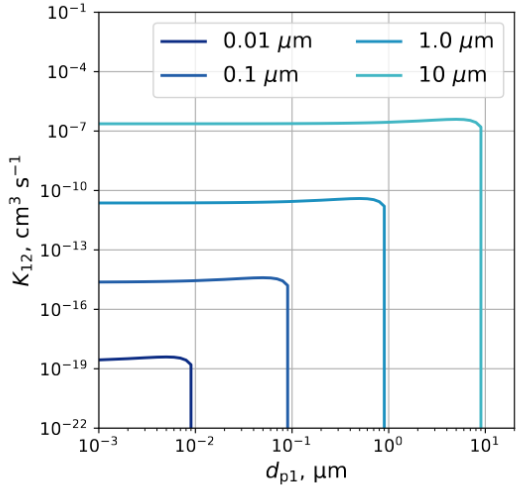

Figure A.11 Answer to Question 5(c, i)

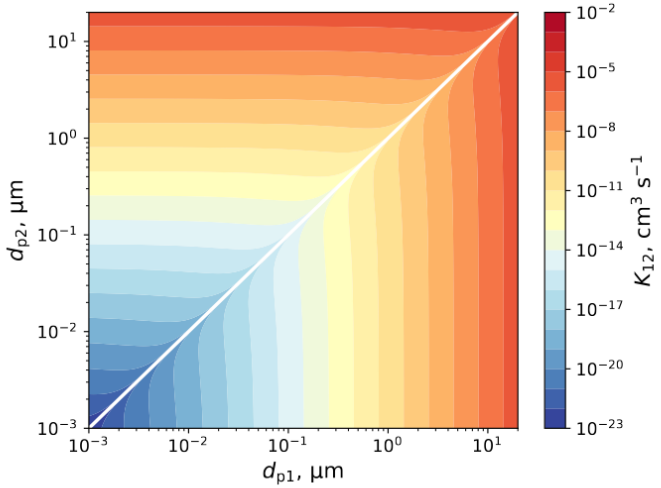

Figure A.12 Answer to Question 5(c, i)

(ii) The solution can be found in file Answer_5_3_b.py
When the diameter is smaller than 2.08 μm, Brownian is more important. When the diameter is greater than 2.12 μm, gravitational settling is more important. When the diameter is between 2.08 μm and 2.12 μm, the two processes are of approximately similar importance.

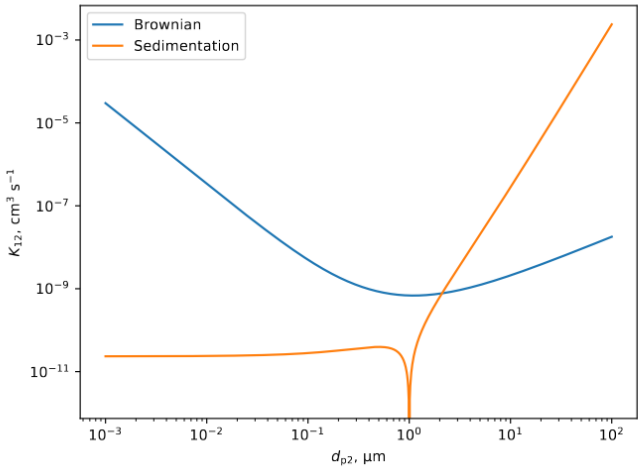

Figure A.13 Answer to Question 5(c, ii)

(d) By modifying the files noted in this exercise, you should discover dependencies displayed in figure A.14.

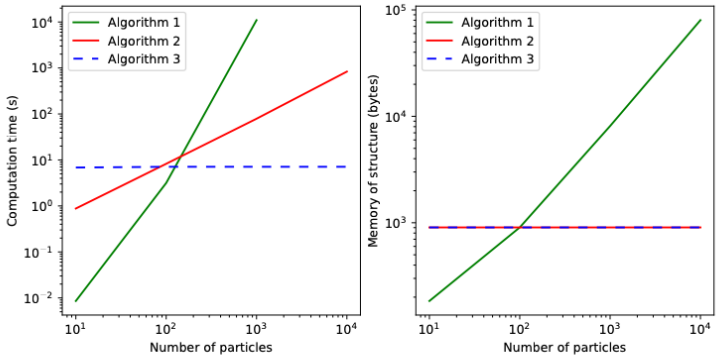

Figure A.14 Answer to Question 5(d).

(e) The solution can be found in file Answer 5 5.py

(i) The mean of value of τ is $\frac{1}{6}$ but note that it may not be exact given sample size.

(ii) Please see figure {fig:ch05$_q$05

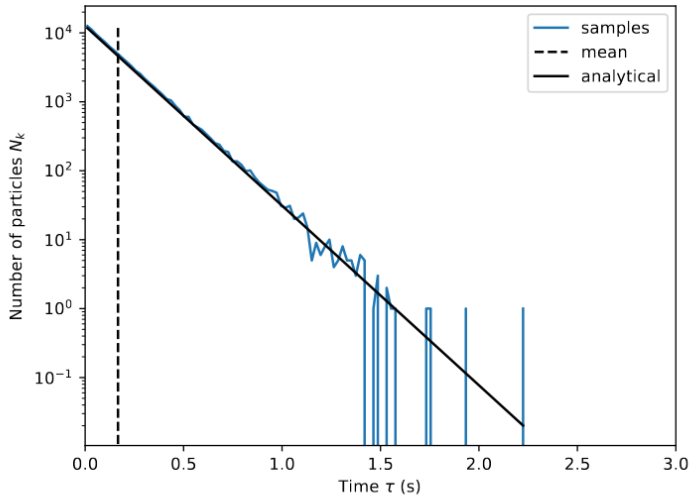

Figure A.15 Histogram of the values of τ for 100 000 samples. Note that plot of τ will be noisy at larger values of τ as very few events occur even with sample size of 100 000.

(f) (i) The solution can be found in file `Answer_5_6_a.py`
(ii) The solution can be found in file `Answer_5_6_b.py`

(g) This is the case for $k = \ell$, and k and ℓ are indices of the bins, i.e, we are considering coagulation events within one bin. Let's assume we have N particles in this bin. Figure A.16 illustrates the particle pairs that we need to count, indicated by an "x". By counting the collision between P1 and P2, we have accounted for this pair, and don't need to additionally count P2 and P1. Coagulations of particles with themselves are impossible, so the diagonal can also be discarded. Overall, we only need to count the pairs below the diagonal, which is expressed by the formula $\frac{1}{2}(N^2 - N) = \frac{1}{2}N(N-1)$.

	P1	P2	P3	P4	...
P1	O	O	O	O	...
P2	X	O	O	O	...
P3	X	X	O	O	...
P4	X	X	X	O	...
...	X	X	X	X	...

Figure A.16 Illustration needed for Question 5(g). The "x" indicate particle pairs that need to be counted, the "o" indicate pairs that do not need to be counted.

(h) The solution can be found in file `Answer_5_8.py` which will lead to the figure A.17.

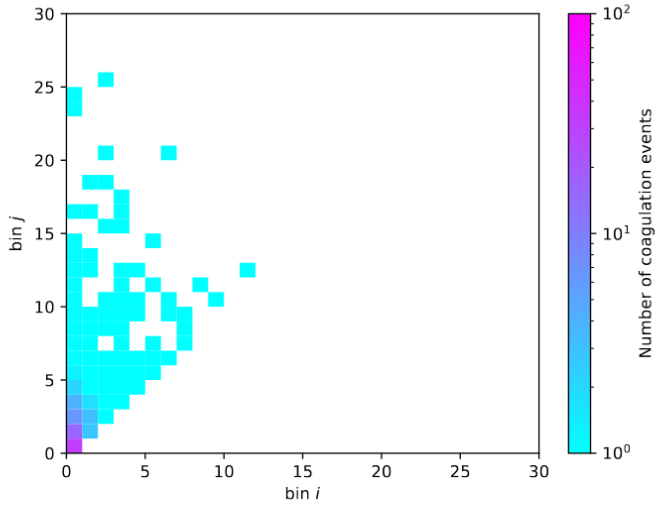

Figure A.17 Number of coagulation events over the course of the simulation.

(i) The solution can be found in file Answer_5_9.py which will lead to figure A.18.

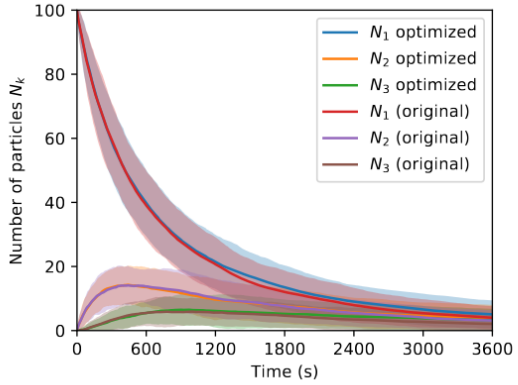

Figure A.18 Solution to 5(i).

(j) (i) The solution can be found in file Answer_5_10.py
(ii) Please see figure A.19.

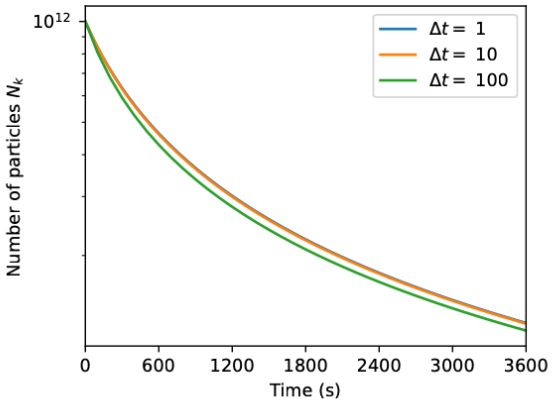

Figure A.19 Solution to 5(j).

(k) In this question we ask you to re-write algorithms 5.1–5.5. We therefore retain the file naming convention of the code snippets provided in the main text and the solutions can be found in the files: ch05_q11_alg1.py, ch05_q11_alg2.py, ch05_q11_alg3.py, ch05_q11_alg4.py. Please refer to figure A.20.

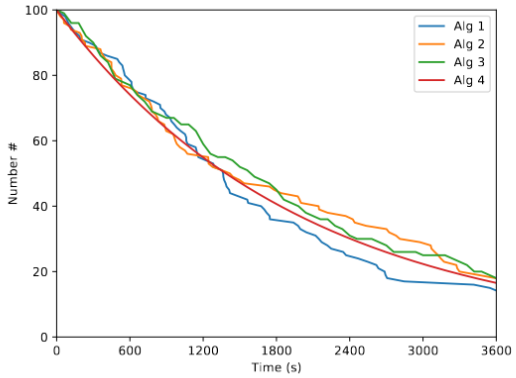

Figure A.20

(l) In this question we ask you to setup a PartMC run as described in Section 5.5. You may find solution in the file solution.py (or solution.ipynb) that is provided in the zip file Answer_5_12.zip which also contains the ancillary setup files.
   (1) number fraction of particles that are purely soot: 0.35
   (2) number fraction of particles that contain ammonium sulfate and soot: 0.12
   (3) number fraction of particles that that are soot-free: 0.53

(m)

$$n(m)dm = \tilde{n}(v)dv$$
$$m = \rho v$$
$$dm = \rho dv$$
$$n(m) = \frac{1}{\rho}\tilde{n}(v)$$

In Equation 5.11, replace $n(m)$, $n(m')$, dm by these relationships. We obtain:

$$\frac{1}{\rho}\frac{\partial \tilde{n}(v,t)}{\partial t} = -\int_0^\infty \tilde{K}(v',v)\tilde{n}(v',t)\frac{1}{\rho}\tilde{n}(v,t)\,dv' + \frac{1}{2}\int_0^v \tilde{K}(v-v',v')\frac{1}{\rho}\tilde{n}(v-v',t)\tilde{n}(v',t)\,v'.$$

The factor $1/\rho$ can be cancelled and the final result is

$$\frac{\partial \tilde{n}(v,t)}{\partial t} = -\int_0^\infty \tilde{K}(v',v)\tilde{n}(v',t)\tilde{n}(v,t)\,dv' + \frac{1}{2}\int_0^v \tilde{K}(v-v',v')\tilde{n}(v-v',t)\tilde{n}(v',t)\,v'.$$

(n) You may find the solution in the file Answer_5_14.py
   (i) Please see figure A.21.

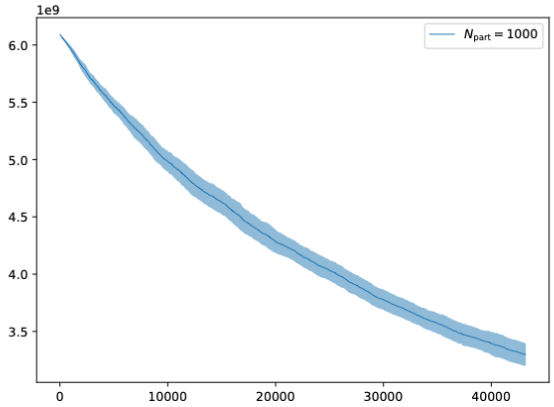

Figure A.21 Solution to n(i) using 1000 particles and 10 runs.

   (ii) Please see figure A.22

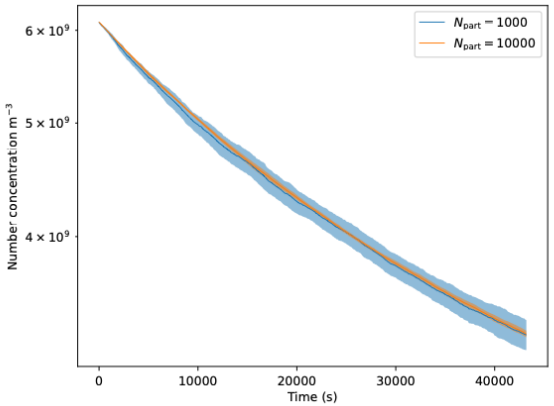

Figure A.22 Solution to n(ii) using 10000 particles and 10 runs compared to previous 1000 particles.

(iii) Please see figure A.23

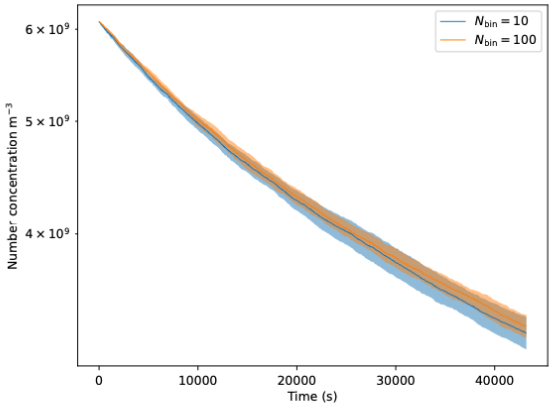

Figure A.23 Solution to n(iii) using 10 and 100 bins with 10 runs.

6. Chapter 6

(a) (i) Clusters at the system boundary are affected by the system size if larger clusters outside the system evaporate significantly. Here, it seems that obtaining

reliable concentrations for clusters up to the 5-mer requires including at least a couple of clusters larger than 5 molecules.

(ii) Coagulation and scavenging decrease the concentrations, with the effect accumulating for larger clusters. As the formation rate over a given particle size follows directly from concentrations, such effects must be included in formation rate predictions.

(iii) The modification has a significant effect on the concentrations of the simulated clusters, demonstrating the importance of quantifying the evaporation rates. In general, classical rates do not describe small clusters, and should not be used.

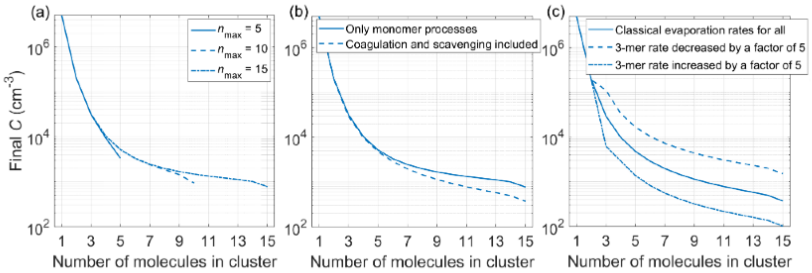

Figure A.24

(b) (i) The highest steady-state concentrations correspond to the highest evaporation rates. The size-dependent concentration decreases steeply at the smallest sizes with high evaporation, and becomes flatter as the monomer collision rate exceeds the evaporation rate.

(ii) In a particle formation situation, the net flux between consecutive clusters is positive. For the smallest sizes, the collision frequency $\beta_{i,1}C_1$ of a given cluster is lower than the evaporation frequency $\gamma_{(i+1)\rightarrow i,1}$ of the consecutive cluster. To compensate for this, the concentration of the smaller cluster must be higher than that of the larger cluster so that $\beta_{i,1}C_1C_i > \gamma_{(i+1)\rightarrow i,1}C_{(i+1)}$.

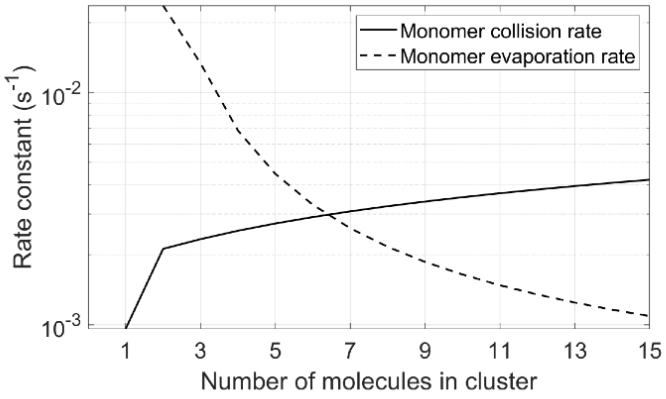

Figure A.25

(c) (i) We can examine the evaporation rates and the relative molecular collision vs. evaporation rates for the cluster set by function "rates_and_deltags_ABe.m" in the ACDC code repository. It is seen that at $T = 280$ K, collisions begin to exceed evaporations for the largest clusters, suggesting that the set is reasonably large. However, note that the cluster stability depends on vapor concentrations and temperature: simulation results may involve system-size-dependent uncertainties at higher temperatures due to higher relative evaporation at the low vapor concentrations used here. (But maybe not at higher concentrations—try it yourself.)

Another potential issue is cluster charge: these electrically neutral clusters seem relatively unstable, which means that ions may have an important increasing effect on cluster stability and thereby formation rate. Fig. 6.7 shows that the purely neutral formation rates are at least a couple of orders of magnitude lower than typical atmospheric ion production rates (a couple of ion pairs s^{-1} cm^{-3}), which is the upper limit for ion-mediated formation rate. Thus, at least in theory ions may increase the formation rate.

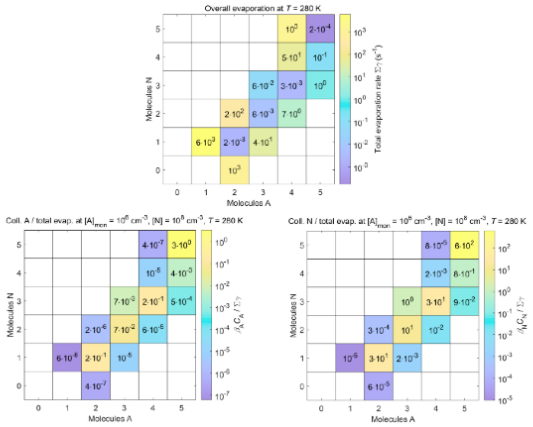

Figure A.26

(ii) Formation rate as a function of $[H_2SO_4]$ can be readily plotted by "run_steadystate_ABB.m" in the ACDC code repository. As the most stable clusters in the set contain n H_2SO_4 and $(n-1)$ NH_3 molecules, we can first test allowing clusters containing at least 6 H_2SO_4 and 5 NH_3 to grow out. Next, we can try tightening the criterion by requiring at least 6 H_2SO_4 and 6 NH_3. This significantly decreases the formation rate. On the other hand, loosening the criterion by requiring at least 6 H_2SO_4 and 5 NH_3, or at least 5 H_2SO_4 and 6 NH_3 has only minor effects, even with excess NH_3.
The 6+6 criterion is not reasonable since it prevents clusters from growing out by collisions with vapor monomers. While the effect of including the 5+6 criterion is not large, this growth path seems less realistic based on the stability trends and growth routes (e.g. fig. 6.7) within the cluster set. We can also deduce that allowing the 5+6 route must alter the concentration of at least the $(H_2SO_4)_5 \cdot (NH_3)_5$ cluster (why?), which is an artefact if the route is negligible in reality. Thus, 6+5 seems to be the most justifiable criterion.

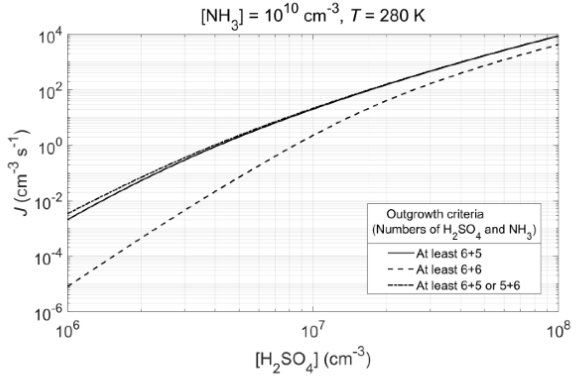

Figure A.27

(d) The dependencies can be examined, for example, by the following script:

```
clear variables

% Plot the ACDC formation rate as a function of vapor
    concentrations or temperature

% Generate the equations
% Use the --variable_temp option to easily change the temperature
    after generating the files
commandstr=['perl acdc_2021_03_23.pl',...
' --i cluster_set_file_acdc.inp',...
' --e formation_free_energy_file_acdc.txt',...
' --variable_temp',...
' --use_cs --cs exp_loss --exp_loss_coefficient 0.001 --
    exp_loss_exponent -1.6'];
lrun=system(commandstr);
if lrun ~= 0
    error('Running the Perl script failed!')
end
rehash pathreset

% Define vapor concentrations (cm^-3) and temperature (K)
% Here, one of these must be a vector (= the x-axis) and the
    others are scalars
%Ca = 1e6;              % H2SO4 concentration
Ca = 10.^(6:0.1:8);     % (Logarithmically evenly spaced values)
Cb = 1e10;              % NH3 concentration
%Cb = 10.^(8:0.1:10);
temp = 280;             % Temperature
%temp = 260:10:320;
%IPR = 3;               % (Ion production rate in cm^-3 s^-1 for
    tests where the cluster set includes charged clusters)

% Define which quantity is on the x-axis - note that also some
    settings below need to be revised accordingly
% For clarity, this is set manually but as an extra exercise you
    can design an automatic way to select the x-axis quantity :)
x_vector = Ca;

```

```
J = zeros(size(x_vector));

% Loop over the x-axis points
for nxval = 1:length(x_vector)

    % Set the x-axis quantity
    Ca = x_vector(nxval);
    %Cb = x_vector(nxval);
    %temp = x_vector(nxval);

    % Set the vapor concentrations
    fid=fopen('driver_input.txt','w');
    %fprintf(fid,'constant 1A %e\n',Ca);        % (Assuming H2SO4 monomers)
    fprintf(fid,'constant 1A %e -1A1N\n',Ca);  % Assuming also H2SO4 clustered with base
    fprintf(fid,'constant 1N %e\n',Cb);
    %fprintf(fid,'source neg %e\n',IPR);        % (For tests where the cluster set includes charged clusters)
    %fprintf(fid,'source pos %e\n',IPR);
    fclose(fid);

    % Call the driver
    [C,T,ok,clust,~,~,~,J_out]=driver_acdc(1e5,'Sources_in','driver_input.txt','repeat','Temperature',temp);

    % Check if the steady state has been reached
    if ok ~= 1
        fprintf('Did not reach steady state for Ca = %.2e cm^-3, Cb = %.2e cm^-3\n',Ca,Cb)
        % Check for negative concentrations
        if ok == -1
            fprintf('Negative concentrations for Ca = %.2e cm^-3, Cb = %.2e cm^-3\n',Ca,Cb)
        end
        continue
    end

    % Save the formation rate
    J(nxval) = J_out(end);

end

%% Hint: Note that one could also utilize the existing ACDC Matlab routines in https://github.com/tolenius/ACDC, for example:
% for nxval = 1:length(x_vector)
%     temp = x_vector(nxval);
%     % The wanted output can be directly returned by adding the output variables, here J, in file run_steadystate_ABB.m
%     J(nxval) = run_steadystate_ABB('input_run_steadystate_AN.m','temp',temp);
% end


figure(1)

set(gca,'XScale','log')
set(gca,'YScale','log')
hold on; set(gcf,'Color','white'); box on

plot(x_vector,J,'Linewidth',1.5)

% Set the x-axis quantity
xlabel('[H_2SO_4] (cm^{-3})')
```

```
%xlabel('[NH_3] (cm^{-3})')
%xlabel('{\itT} (K)')
ylabel('{\itJ} (cm^{-3} s^{-1})')
```

The formation rate increases with vapor concentrations, as expected, with the slope depending on the species. The increase (on a logarithmic scale) becomes weaker with increasing vapor concentrations as the system saturates with respect to the compounds. Increasing temperature decreases the formation rate as evaporation increases, with the formation rate decreasing more rapidly at lower vapor concentrations at which the role of evaporation is more significant. Note that the behavior is different and more complex when ions are included (why?)—try it yourself.

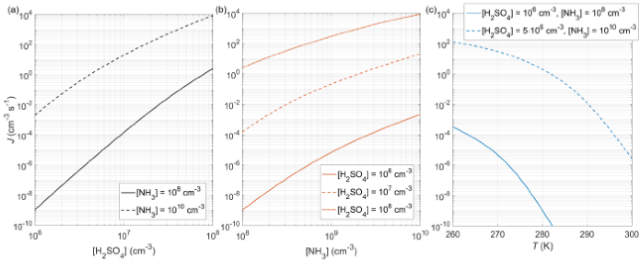

Figure A.28

7. Chapter 7

(a) Running the box model with model_setup=1 will result in size distributions shown in Figure A.29

(i) The hybrid bin method, both linear sub-grid size distribution and flat-top sub-grid size distribution match very closely the curve of the full-moving method. When using the quasi-stationary method, the size distribution is wider than for the full-moving method while the moving center method results in a size distribution with large variation between the number concentrations in bins.

(ii) When using moving center method, two adjacent bins sometimes become merged if only the smaller of the two bins exceeds its upper bin boundary. In such a case, the larger bin will receive the aerosol mass and number of both of the bins and the smaller bin becomes empty. In a 0-dimensional box model, the number of merged bins can increase throughout the simulation, but their number will never decrease unless new particles are emitted to the system. In a three dimensional framework this is not an issue since the advection of aerosol from surrounding grid boxes can fill the empty bins.

(iii) The two hybrid bin methods are almost identical and in most atmospheric conditions these methods differ very little. Only in case of very strong

microphysical processing the method yield in significantly different size distributions.

(b) Running the box model with model_setup=2 will result in size distributions shown in Figure A.30. In this setup, the hybrid-bin method performs better since the full-moving method fails to solve nucleation and condensation simultaneously. This is because condensation grows the smallest size bin and newly formed particles are merged to the smallest bin particles. Thus they are artificially growing them to sizes that towards the end of the simulation are much larger than their formation size of 10 nm in diameter. When hybrid-bin method is used, the smallest bin has the lower limit at 10 nm.

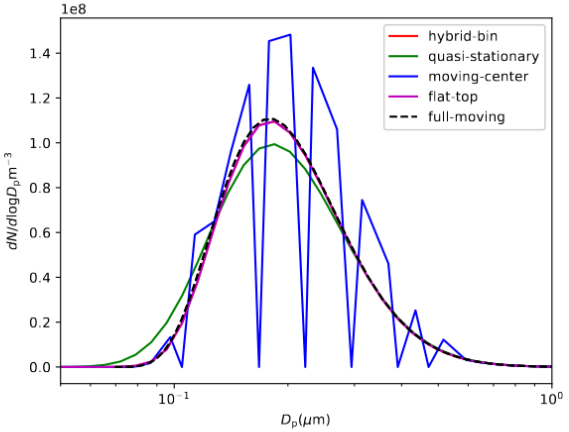

Figure A.29 Size distributions at the end of the simulation.

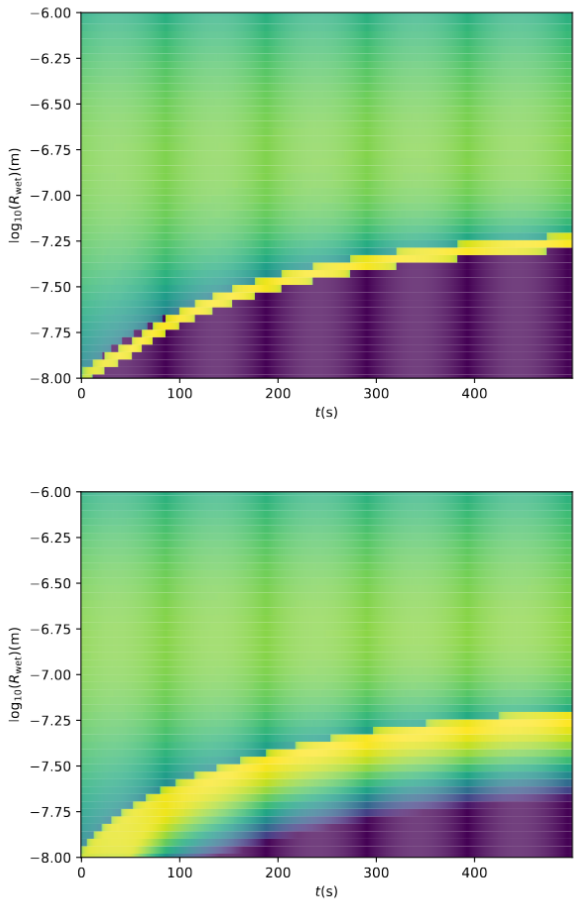

Figure A.30 Upper panel: The evolution of the size distribution using the full-moving method. Lower panel: The evolution of the size distribution using the hybrid-bin method.

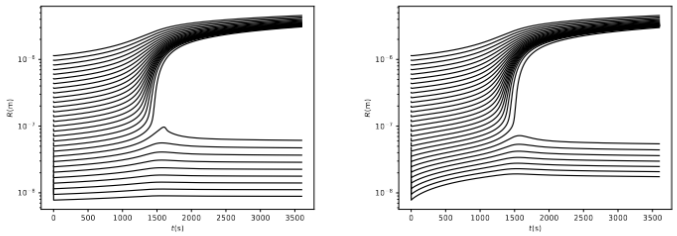

Figure A.31 Left panel: Saturation ratios of water at the droplet surface and for saturation ratio for water vapor in the air. Right panel: The same figure zoomed to the maximum supersaturation of water in air.

(c) Simulation with model_setup=3:

(i) Wet radii are shown in the left panel of Figure A.31.

· At the beginning of the simulation, smallest bins lose their size since their wet sizes are initialized without taking the Kelvin effect into account which results in too much water in the droplets (see Section 7.4.1). The droplets reach very close their equilibrium size in seconds after which they start to grow due to adiabatic expansion of the cloud parcel. At approx 1500 s, the population splits into two; activated and non-activated droplets. The activated droplets grow spontaneously by condensation while the non-activated droplets start to shrink due since condensation to activated droplets depletes the gas phase water. The growth of the largest droplets saturates slowly since the condensation gets less efficient in growing the droplets the larger the droplets grow.

(ii) Saturation ratios of water at the droplet surface for the four different bins are shown in Figure A.32

· The saturation ratio of water in air (black curve) increases due to the adiabatic cooling of the ascending cloud parcel. At approx 1400 s, aerosol droplets start to activate to cloud droplets forming a cloud. Once the cloud droplets are formed, condensation of water onto the depletes the gas phase water reducing the saturation ratio of water in air despite the fact that the cloud parcel continues to cool.

(iii) (continued) Simulation with model_setup=3:

· The smallest droplet bin (magenta curve) stays near its Köhler equilibrium never reaching its maximum saturation ratio. When the saturation ratio of water in air increases, the droplet bin will grow. After the cloud formation, once the saturation ratio of water in air starts decreasing the droplet bin will shrink following its Köhler curve.

· The smallest activated droplet bin (red dashed curve) reaches the maximum saturation ratio shortly after 1400 s into the simulation after which

the growth of the droplet decreases its Kelvin effect decreasing the saturation ratio. Since the saturation ratio of water in air exceeds the saturation ratio at the droplet surface, water keeps condensing on the droplet spontaneously growing the droplet.

- The largest non-activated droplet bin (blue dashed curve) also reaches the maximum saturation ratio at before 1600 s. However, the saturation ratio of water in air decreases below the saturation ratio at the droplet surface causing water to evaporate from the droplets returning the droplet to the "ascencing part" of its Köhler curve deactivating the droplet after which, the droplet bin will be near its Köhler equilibrium.
- The largest droplet never reaches a maximum value since condensation is too slow for the droplet to reach the maximum. However, the droplet can still be considered activated since it grows spontaneously by condensation.

(d) Simulation with model_setup=4.

* Plot the wet radii as a function of time. Wet radii for model_setup=4 are shown in the right panel of Figure A.31. Comparing the radii to the left panel, especially the smallest droplets grow faster in the beginning of the simulation.
* The difference between the number of activated droplets compared to a case where only water condenses on droplets is 414561 m^{-3}. This is because condensing organic mass increases the hygroscopic material in the droplets thus decreasing the critical supersaturation required for activation.
* The concentration of VBS species is at its highest in the smallest activated bin since they have the highest area-to-volume ratio of the activated droplets making condensation onto the more efficient than for larger droplets.

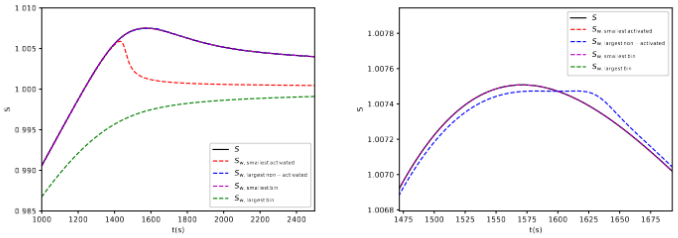

Figure A.32 Left panel: Saturation ratios of water at the droplet surface and for saturation ratio for water vapor in the air. Right panel: The same figure zoomed to the maximum supersaturation of water in air.