EDITED BY **DAVID TOPPING** • **MICHAEL BANE**

INTRODUCTION TO AEROSOL MODELLING

FROM THEORY TO CODE

WILEY

Introduction to Aerosol Modelling

Introduction to Aerosol Modelling

From Theory to Code

Edited by

David Topping
University of Manchester
Manchester, UK

Michael Bane
Manchester Metropolitan University &
High End Compute Ltd
Manchester, UK

WILEY

This edition first published 2022
© 2022 John Wiley & Sons Ltd

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by law. Advice on how to obtain permission to reuse material from this title is available at https://www.wiley.com/go/permissions.

The right of David Topping and Michael Bane to be identified as the authors of the editorial material in this work has been asserted in accordance with law.

Registered Offices
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

Editorial Office
9600 Garsington Road, Oxford, OX4 2DQ, UK

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Limit of Liability/Disclaimer of Warranty
The contents of this work are intended to further general scientific research, understanding, and discussion only and are not intended and should not be relied upon as recommending or promoting scientific method, diagnosis, or treatment by physicians for any particular patient. In view of ongoing research, equipment modifications, changes in governmental regulations, and the constant flow of information relating to the use of medicines, equipment, and devices, the reader is urged to review and evaluate the information provided in the package insert or instructions for each medicine, equipment, or device for, among other things, any changes in the instructions or indication of usage and for added warnings and precautions. While the publisher and authors have used their best efforts in preparing this work, they make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives, written sales materials or promotional statements for this work. The fact that an organization, website, or product is referred to in this work as a citation and/or potential source of further information does not mean that the publisher and authors endorse the information or services the organization, website, or product may provide or recommendations it may make. This work is sold with the understanding that the publisher is not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for your situation. You should consult with a specialist where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

A catalogue record for this book is available from the Library of Congress

Paperback ISBN: 9781119625650; ePub ISBN: 9781119625711; ePDF ISBN: 9781119625667

Cover image: Cover illustration kindly provided by Harri Kokkola, Finnish Meteorological Institute.
Photograph ©Tuukka Kokkola
Cover design by Wiley

Set in 9.5/12.5pt STIXTwoText by Integra Software Services Pvt. Ltd, Pondicherry, India

Contents

Contributors *ix*
Preface *xi*
Acknowledgments *xii*
About the Companion Website *xiii*

1 Introduction and the Purpose of this Book *1*
1.1 Aerosol Science and Chapter Synopses *4*
1.2 Computers and Programming Languages *10*
1.3 Representing Aerosol Particles as Model Frameworks *15*
1.3.1 Size Distributions *18*
1.3.2 The Sectional Distribution *22*
1.3.3 The Modal Distribution *25*
1.4 Code Availability *28*
Bibliography *28*

2 Gas-to-particle Partitioning *32*
2.1 Adsorption *33*
2.2 Equilibrium Absorptive Partitioning *37*
2.3 Knudsen Regimes and the Kelvin Effect *43*
2.4 Kinetic Absorptive Partitioning: The Droplet Growth Equation *46*
2.4.1 Solving the Droplet Growth Equation: A Sectional Approach *49*
2.5 Cloud Condensation Nuclei Activation *65*
2.5.1 Köhler Theory *65*
2.5.2 Hygroscopic Growth Factors and Kappa Köhler Theory *71*
Bibliography *75*

3 Thermodynamics, Nonideal Mixing, and Phase Separation *78*
3.1 Thermodynamics and Nonideal Mixing *78*
3.1.1 Chemical Thermodynamics *78*
3.2 Activity Coefficient Model *83*
3.3 BAT Model Implementation *85*
3.3.1 Example 1: Calculation of Binary Mixture Activities Using the BAT Model *89*
3.4 Phase Separation *96*
3.4.1 Example 2: Detection and Computation of LLPS in a Binary System *97*

3.5 Multicomponent Aerosol Thermodynamics Models *117*
3.6 Activity and LLE Computations with the AIOMFAC Model *118*
3.6.1 Customizing and Running AIOMFAC-LLE *121*
Bibliography *129*

4 Chemical Mechanisms and Pure Component Properties *133*
4.1 Chemical Mechanisms *134*
4.1.1 Gas Phase Only Model *134*
4.1.2 Coupling the Gaseous and Condensed Phase Using a Fully Moving Sectional Approach *153*
4.1.3 An Example Using the Sectional Model Generator JlBox *158*
4.1.4 Modal Model for Condensational Growth *163*
4.2 Chemical Identifiers and Parsing Structures *169*
4.3 Coding Property Prediction Techniques *172*
4.3.1 Group Contribution Methods *175*
4.3.2 Vapor Pressure Prediction Methods *176*
4.3.3 Example: Adding the SIMPOL Method to UManSysProp *176*
4.4 Subcooled Liquid Density *181*
Bibliography *183*

5 Coagulation *187*
5.1 Coagulation Probabilities and Rates *187*
5.2 Stochastic Coagulation with Discrete Particle Masses *189*
5.2.1 Gillespie's Basic Algorithm for Discrete Number Concentrations *189*
5.2.2 First Speedup: Binning Particles *192*
5.2.3 Second Speedup: Discretize Time and Use Tau-leaping *194*
5.2.4 Third Speedup: Large-number-limit Using Continuous Number Concentration *197*
5.3 Coagulation with Continuous Particle Masses *202*
5.3.1 Particle-resolved Approach for Coagulation *202*
5.3.2 Sectional Approach for Coagulation *206*
5.3.3 Modal Approach for Coagulation *208*
5.4 Advanced Coagulation Topics *213*
5.4.1 Coagulation for a Multi-dimensional Composition Space *213*
5.4.2 Other Considerations *215*
5.5 Introduction to Particle-resolved Monte Carlo (PartMC) *217*
5.5.1 PartMC Input (.dat) Files Preparation *218*
5.5.2 Spec File Preparation *219*
5.5.3 PartMC Execution and Postprocessing *220*
Bibliography *220*

6 Nucleation: Formation of New Particles from Gases by Molecular Clustering *223*
6.1 Modelling Particle Formation: From Atoms to Molecular Cluster Populations *224*
6.1.1 The Discrete General Dynamic Equation *227*
6.1.2 The Discrete Cluster GDE vs. the Continuous Aerosol GDE *229*

6.1.3 Rate Constants of the Cluster Dynamics Processes *229*
6.1.4 Cluster Formation in Different Atmospheric Environments *232*
6.2 Coding the Discrete GDE: The Straight-forward Case of a One-component System *233*
6.3 Multi-component Systems: Need for an Equation Generator *239*
6.4 Brief Introduction to Atmospheric Cluster Dynamics Code *241*
6.4.1 ACDC Input *242*
6.4.2 Running an ACDC Simulation *246*
6.4.3 Code Features Useful for Studying Clustering Mechanisms *248*
6.4.4 ACDC Applications *250*
6.5 From Clustering to Particle Growth: Implementation of Initial Particle Formation in Aerosol Dynamics Models *252*
6.5.1 The Default Approach: Particle Formation Rate as an Input Parameter *252*
6.5.2 The Dynamic Approach: Combination of Molecular Cluster and Aerosol GDEs *255*
Bibliography *256*

7 Box Models *259*
7.1 *box_model.py* *260*
7.2 Remapping Size Distribution When Using the Sectional Method *264*
7.2.1 Quasi-Stationary Sectional Method *265*
7.2.2 Moving Center Method *269*
7.2.3 Hybrid Bin Method *270*
7.3 Simulating Absorptive Uptake and New Particle Formation Simultaneously *279*
7.4 Cloud Parcel Models *280*
7.4.1 Sectional Cloud Parcel Model *281*
7.5 SALSA *284*
Bibliography *286*

8 Software Optimization *288*
8.1 Portability *288*
8.2 Performance *289*
8.2.1 Compiler Optimization *289*
8.2.2 Profiling *290*
8.2.3 Case Study: Speeding-up Box Model *291*
8.2.4 Vectors *297*
8.2.5 Hand Holding the Compiler *298*
8.2.6 Case Study: PartMC *298*
8.2.7 Interpretted Languages *302*
8.2.8 Case Study: Droplet Growth Equation *303*
8.3 Parallelization *305*
8.3.1 Making the Most of a Single Node *307*
8.3.2 Making Use of Multiple Nodes *308*
8.3.3 Other Technologies *309*
8.4 Collaborative Software Engineering *310*
8.4.1 Coding Stylesheets *311*
8.4.2 Modularity and Re-use *311*

8.4.3 Version Control 312
8.4.4 Software Development Life Cycle 312
8.4.5 Continuous Integration and Unit Tests 312
8.5 In Conclusion 313
Bibliography 314

A Appendix A *316*
A.1 Exercises 316
A.2 Physical Constants 329
A.3 Conversion Factor 330
A.4 Variable Definitions 330
Bibliography 345

Index *347*

Contributors

Michael Bane has spent several decades in optimizing codes and helping others understand how they can optimize codes. He is currently a Lecturer at Manchester Metropolitan University, UK, and Director of High End Compute LTD. Michael wrote Chapter 8 and gave supportive input to other chapters, particularly Chapter 1.

Jeffrey Curtis, Postdoctoral Research Associate in the Department of Atmospheric Sciences at the University of Illinois at Urbana-Champaign. Jeffrey contributed to Chapter 5.

Harri Kokkola, Group leader of the Atmospheric Modeling Group at the Atmospheric Research Centre of Eastern Finland. He has developed and applied models simulating atmospheric aerosol, clouds, and climate from process scale to global scale. Harri wrote Chapter 7.

Benjamin Murphy, Physical Scientist in the United States Environmental Protection Agency Office of Research and Development at Research Triangle Park, North Carolina. Benjamin contributed to Chapters 1, 4, and 5.

Tinja Olenius, research scientist in the Air Quality Unit at Swedish Meteorological and Hydrological Institute (SMHI). Tinja's research interests include developing the chain of modeling tools from quantum mechanics to large-scale models for improving the representation of secondary particle formation from vapors. Tinja wrote Chapter 6.

Olli Pakarinen, university lecturer at the Institute for Atmospheric and Earth System Research (INAR), University of Helsinki, Finland. Olli contributed to Chapter 6.

Nicole Riemer, Professor in the Department of Atmospheric Sciences at the University of Illinois at Urbana-Champaign, IL, USA. Nicole's research interests include the development of aerosol models, from the process level to the global scale. Nicole wrote Chapter 5.

Petroc Shelley, PhD student in the Department of Earth and Environmental Science at the University of Manchester, UK. His current specialism is in the measurement and prediction of pure component properties. Petroc contributed to Chapter 4.

David Topping, Professor in the Department of Earth and Environmental Science at the University of Manchester, UK. Since his PhD, David has developed and applied

models of aerosol particles across a range of scales. David convened the writing team behind this book and wrote Chapters 1, 2, and 4.

Matthew West, Associate Professor in the Department of Mechanical Science and Engineering at the University of Illinois at Urbana-Champaign, IL, USA. His research interests include stochastic time integration methods and scientific computing. Matt wrote Chapter 5.

Zhonghua Zheng, PhD in Environmental Engineering in Civil Engineering with a concentration in Computational Science and Engineering at the University of Illinois at Urbana-Champaign, IL, USA. Zhonghua contributed to Chapter 5.

Andreas Zuend, Professor in the Department of Atmospheric and Oceanic Sciences at McGill University, Montreal, Quebec, Canada. His research interests include the development of predictive thermodynamic aerosol models for gas–particle partitioning, as well as reduced-complexity methods for mixture properties of aerosols and cloud droplets. Andreas wrote Chapter 3.

Preface

Aerosol science is one that straddles many disciplines. There is a natural tendency for the aerosol scientist to therefore work at the interface of the traditional academic subjects of physics, chemistry, biology, mathematics, and computing. The impacts that aerosol particles have on the climate, air-quality, and thus human health are linked to their evolving chemical and physical characteristics. Likewise, the chemical and physical characteristic of aerosol particles reflect their sources and subsequent processes they have been subject to. Computational models are not only essential for constructing evidence-based understanding of important aerosol processes, but also to predict change and potential impact. Seminal publications provide an extensive overview on the history and basis of core theoretical frameworks that aerosol models are based on. However there is little on how we can translate such theory into code. While we focus on atmospheric aerosol in this book, the theory and tools developed are based on core aerosol physics that translate across multiple disciplines. Likewise, demonstrating a programming solution to common numerical operations is valuable to a large number of scientific disciplines. You may be reading this book as an undergraduate, postgraduate, seasoned researcher in the private/public sector or as someone who wishes to better understand the pathways to aerosol model development. Wherever you position yourself, it is hoped that the tools you will learn through this book will provide you with the basis to develop your own platforms and to ensure the next generation of aerosol modelers are equipped with foundational skills to address future challenges in aerosol science.

Manchester
July 2022

Professor David Topping

Acknowledgments

The identified need for this book was inspired by the Aerosol and Droplet Science Centre for Doctoral Training [CDT], funded by the UK Engineering and Physical Sciences Research Council (EPSRC)(grant no. EP/S023593/1) (https://www.aerosol-cdt.ac.uk/).

The authors would also like to thank Hanna Vehkamäki, Ana Cristina Carvalho, Manu Thomas and Cecilia Bennet for useful discussions and comments. The Swedish Research Council VR (grant no. 2019-04853) and the Swedish Research Council for Sustainable Development FORMAS (grant no. 2019-01433) are acknowledged for financial support for Chapter 6.

About the Companion Website

All of the code snippets and examples provided with the book can also be downloaded from the project Github repository:

https://github.com/aerosol-modelling/Book-Code.git

1

Introduction and the Purpose of this Book

An aerosol particle is defined a solid or liquid particle suspended in a carrier gas. The term "aerosol" technically includes both the particle and carrier gas, though it is common to often hear this used when referring to just the particle. In this book, we will retain the use of the term "aerosol particle". Whilst we often treat scientific challenges in a siloed way, aerosol particles are of interest across many disciplines. For example, atmospheric aerosol particles are key determinants of air quality [1–3] and climate change [4–6]. Improving our understanding of sources, processes and sinks is important as we develop strategies to lesson the impacts we have on human health and environmental systems. Knowledge of aerosol physics and generation mechanisms is key to all factors of fuel delivery [7] and drug delivery to the lungs [8]. Likewise, various manufacturing processes require optimal generation, delivery and removal of aerosol particles in a range of conditions [9].

The purpose of this book is to provide you, the reader, with the tools to translate theory on which numerical aerosol models are based into working code. In following the content provided in this book, you will be able to reproduce models of key processes that can either be used in isolation or brought together to construct a demonstrator 0D box-model of a coupled gaseous-particulate system.

You may be reading this book as an undergraduate, postgraduate, seasoned researcher in the private/public sector or as someone who wishes to better understand the pathways to aerosol model development. Wherever you position yourself, the coupling between experimental and modeling infrastructure is important in any discipline. Whilst the driving factors that influence both can vary, Figure 1.1 presents an idealized workflow of model development and model scales both in response to and as a driving force behind aerosol experiments. Particular emphasis is given to atmospheric aerosol particles in this workflow where, as we move from left to right, we move from aerosol models at the molecular and single particle scale to aerosol models acting as an import component in regional and global scale models. The purpose of this figure is to represent a workflow that migrates our understanding of aerosol processes to a framework that may be used to predict impacts. In a perhaps controversial approach, we can imagine a scale at the bottom of the figure that assumes as we move from left to right we reduce the physical and chemical complexity of our aerosol models. This sets the scene for understanding the research landscape of much of the developments you will find in this book.

Introduction to Aerosol Modelling: From Theory to Code.
First Edition. Edited by David Topping and Michael Bane.
© 2022 John Wiley & Sons Ltd. Published 2022 by John Wiley & Sons Ltd.

If we start at the left-hand side of the figure, we use the term mechanistic model. In aerosol modeling parlance, a mechanistic model is one that is built around a numerical representation of an underlying physical theory. For example, this might include a set of coupled differential equations that describe the movement of mass between a gaseous and condensed phase, or between different compartments of a condensed phase. Parameters in these mechanistic models may describe chemical and physical properties that are included in these differential equations and have been derived from a series of experiments or provided through separate models. In a mechanistic model, our mathematical framework provides a clear numerical narrative and separation of the processes we wish to include. We can then choose an appropriate numerical method to provide, for example, a time-varying solution to a set of initial conditions or predict a point of equilibrium. Once we have constructed our mechanistic model, we can consider uncertainties associated with the model architecture itself and/or errors associated with the parameters we use in our simulation. Indeed, the next phase in our workflow in Figure 1.1 is to compare with targeted laboratory experiments that serve to quantify the accuracy of our model or identify uncertainties that need further reduction. You may find mechanistic frameworks used at the single particle level, or indeed in models that are designed to capture the evolution of a population of particles. Where mechanistic models cannot replicate observed behavior within a specific level of accuracy, or simply do not have an appropriate theoretical basis to build on, parameterizations can be developed. This can be used in combination with, or as an alternative to, the mechanistic model. We often state that a parameterization has a higher computational efficiency than an equivalent mechanistic model. Specifically, the time to solution is reduced. As we move further right in Figure 1.1, and typically start to consider populations of particles and multiple processes, we might refer to hybrid models that combine both mechanistic and parameterizations. We may also start to consider the computational resource available to conduct more complex simulations. At the global scale, an aerosol model is one of many components in a framework that attempts to capture the dynamics of multiple components of the earth system (e.g., ocean, biosphere, land-air interactions etc.). The level of physical and chemical complexity retained in our aerosol model is dictated by a number of factors. These include the computational resource available, the associated detail carried in components that drive and respond to aerosol processes (e.g., how many emissions that lead to aerosol formation are captured) and ongoing efforts to resolve how much detail is needed to resolve potential impacts on, for example, human health. Of course, this narrative is an ideal one but at least provides an insight into factors that dictate the methods we use to construct our aerosol models. In this manner, we start to appreciate the ecosystem of aerosols models and why they exist. The aerosol scientist may come across a range of "simple" and "complex" models that have been designed to provide benchmark simulations in isolation. You will find a description of these benchmark models in the chapters of this book. Once we begin to capture processes across multiple scales, an aerosol model developer starts to consider any approximations that may be needed according to the numerical methods and compute resource available.

Whilst we focus on atmospheric aerosol to define our composition space in this book, the theory and tools developed are based on core aerosol physics that translate

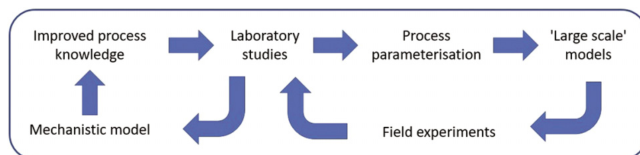

Figure 1.1 Ideal workflow of aerosol model development in environmental science.

across multiple disciplines. Likewise, demonstrating a programming solution to common numerical operations is valuable to a large number of scientific disciplines.

Research developments often move at a rapid pace and, as the global aerosol community develop new observational and modeling platforms, we continually hypothesize and verify new species and/or processes deemed important to improve our understanding. We do not provide a comprehensive coupling of all known and emerging chemical and process complexity in this book. Indeed, there are remaining challenges on how we actually do that from a programming and real-world validation perspective. The landscape of computing hardware and software also moves at a rapid pace. The choice of programming language to solve a particular problem, or provide a particular service, is influenced by a number of factors ranging from required time-to-solution and ability to share across multiple platforms. As with numerical representations of aerosol processes, we do not provide a comprehensive multilanguage demonstration in this book. It is anticipated that readers of this book will have a wide range of programming experience; from those who have no prior experience to those who regularly develop their own applications. We expect therefore that you will take away different lessons from working through the material provided, whether it be the solution to a set of common operations or learning how to develop your first numerical model in your first programming language. You will find there are often multiple ways to write a piece of code that performs a particular task. You will also find that as we often have our own style in writing, so too can we develop our own style of developing code. In this book, we provide complete demonstrations of how to develop working code around key concepts (highlighted in figure 1.2), but we do not force a particular style beyond requirements of the language syntax. We also however provide examples of how we can optimize the code we develop. By looking at a range of examples, this will help you start to more broadly consider how efficient your code is and perhaps embed these considerations as you start to develop mode applications. We also discuss best practice in sharing any code in the public domain and ensuring reproducibility.

Seminal publications [e.g., [2, 19]] provide an extensive overview on the history and basis of core theoretical frameworks that aerosol models are based on. We do not repeat that content in this book; rather we present the theoretical basis used in constructing a model and then focus on how we map this to code development.

The tools you will learn through this book are foundational. As the research community explore new hardware platforms and programming languages in an attempt to tackle growing complexity, these foundational skills will provide you with the basis to develop your own platforms. Will anyone tackle the entirety of aerosol modeling complexity? Maybe it will be you.

1.1 Aerosol Science and Chapter Synopses

Aerosol science is multidisciplinary by nature. This is reflected in the huge body of literature that now exists in peer-reviewed journals. There is a natural tendency for the aerosol scientist to therefore work at the interface of the traditional academic subjects of physics, chemistry, biology, mathematics, and computing. Of course, an aerosol scientist working in either medicine or climate change will find themselves focusing on distinct areas and the level of understanding in each will be dependent on the research challenge. However, chances are that both will, at some point, require training in key concepts of aerosol science that apply to both domains. Indeed, one benefit of becoming an aerosol scientist is that understanding, refinement and application of core concepts is transferable between disciplines.

In 2018, the Aerosol Society of the United Kingdom published the outcomes of an industrial engagement workshop [12], defining a pipeline of research, innovation

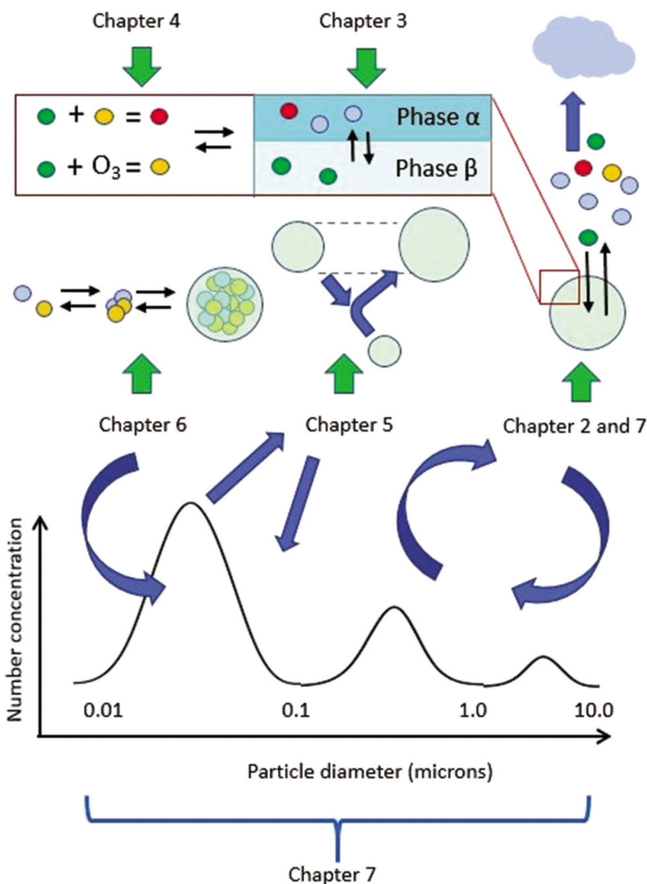

Figure 1.2 The topics covered in each chapter, summarized in the main body of text in this chapter, as a visual schematic that connects processes across the aerosol size spectrum. In this hypothetical example, the aerosol size distribution has three peaks represented as multiple log-normal contributions. We start to discuss log-normal distributions in Sections 1.3.2 and 1.3.3.

and technology development for aerosol science. In this they note that estimates of the global aerosol market size suggest it will reach $84 billion per year by 2024 with products in the personal care, household, automotive, food, paints, and medical sectors. However, they also note that despite the growing interest into the macro-effects and industrial exploitation of aerosols, aerosol science is a relatively young discipline encompassing research topics which can concomitantly be understood as biological, chemical, engineering, environmental, material, medical, pharmaceutical, or physical science.

We focus on atmospheric aerosol particles for the remaining portions of this book. The impacts that aerosol particles have on the climate, air-quality and thus human health, are linked to their evolving chemical and physical characteristics [3, 5]. Likewise, the chemical and physical characteristic of aerosol particles reflect their sources and subsequent processes they have been subject to [2, 14]. Atmospheric aerosol particles can range in size from a few nanometers to hundreds of microns. They can be of primary or secondary origin. Primary particles are directly emitted into the atmosphere, whilst secondary particles are produced from gas-to-particle conversion processes. An aerosol particle may comprise inorganic and/or organic components which can be associated with both primary and secondary particles. Whilst the inorganic fraction may be comprised by a relatively small number of components [2], the organic fraction may comprise many thousands of compounds from multiple sources [15, 16]. As an aerosol particle resides in the atmosphere, we know that many processes taking place in/on atmospheric aerosol particles are accompanied by changes in the particles' morphology (size and shape) [17]. These processes also change the chemical composition of aerosol particles according to the availability of, for example, key gas phase oxidants and ambient conditions [16]. Likewise, particles of primary original (e.g. desert dust, volcanic ash, soot, pollen) can have widely varying morphological features [19–21]. Mechanisms are important in understanding lifetimes and potential impacts [22].

Let us begin with an idealized spherical representation of aerosol particles. Their size and composition will vary, but we wish to simulate how their concentration changes over time. Whilst controlled experiments may be able to isolate single levitated particles [23], under atmospheric conditions we expect a range of particle sizes and number densities. Let us take an isolated particle with diameter d_p and density ρ with units in m and $kg \cdot m^{-3}$, respectively. A single particle has a mass M in kg calculated using Equation (1.1):

$$M = \frac{4}{3}\pi\left(\frac{d_p}{2}\right)^3 \rho \tag{1.1}$$

If we observe N_L particles per cubic centimeter, we can use Equation (1.2) to calculate the total concentration of particles with size d_p, M_{tot}, using the common air quality metric of $\mu g.m^{-3}$.

$$M_{tot} = N_L M 10^{15} \tag{1.2}$$

where the factor 10^{15} is a product of converting cm^{-3} to m^{-3} (10^6) and kg to µg (10^9). This formula has no information on the particle composition or morphology. Indeed, our idealized spherical representation may be wrong under certain conditions and/or

for certain aerosol types. Nonetheless, we can use our particle representation, which has a volume V (m) given by Equation (1.3), as a common particle reference.

$$V = \frac{4}{3}\pi\left(\frac{d_p}{2}\right)^3 \tag{1.3}$$

If we now specify that the concentration (number density) of particles with this specific volume, at a specific time t, can be represented by a variable $n_{v,t}$, we can start to formulate an expression that captures the evolution of particles with variable volumes into an algebraic form.

For example, in Figure 1.3 on the left-hand side we have a population of particles with a specific volume at a given point in time. The total concentration of these particles is $n_{v,t}$. On the right-hand side, this population has evolved after a time increment Δt and we have particles at smaller and larger volumes, represented by a discrete change Δv. Specifically, $n_{v,t+\Delta t}$, $n_{v-\Delta v,t+\Delta t}$ and $n_{v+\Delta v,t+\Delta t}$ represent the concentration of particles that have the original volume, the concentration of particles with a smaller volume and the concentration of particles with a larger volume at a new time $t + \Delta t$, respectively. In reality these particles will have been created through primary and/or secondary mechanisms.

Equation (1.4) is the continuous general dynamic equation [24]. This ordinary differential equation (ODE) describes the rate of change of $n_{v,t}$ resulting from key processes which are identified as nucleation, coagulation and condensation. We also include a generic term that represents emission and removal mechanisms.

$$\frac{dn_{v,t}}{dt} = \left(\frac{dn_{v,t}}{dt}\right)_{nucleation} + \left(\frac{dn_{v,t}}{dt}\right)_{coagulation} + \left(\frac{dn_{v,t}}{dt}\right)_{condensation} + \left(\frac{dn_{v,t}}{dt}\right)_{emission} - \left(\frac{dn_{v,t}}{dt}\right)_{removal} \tag{1.4}$$

Equation (1.4) is the first algebraic formulation of aerosol evolution in this book. Each contribution to this equation has its own theoretical basis and thus algebraic

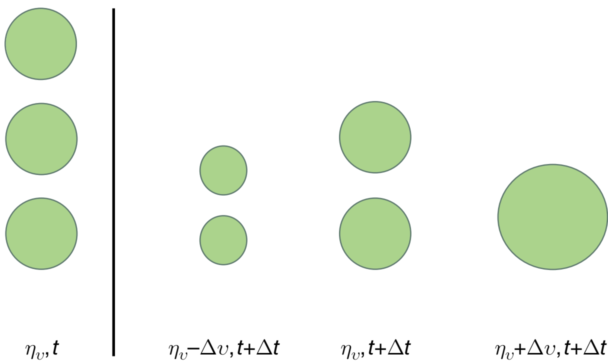

Figure 1.3 An initial population of particles with volume V on the left-hand side of this figure evolve to a population with multiple volumes after a time increment Δt.

formulation. These are presented and discussed in specific chapters, and also highlighted in figure 1.2 and discussed shortly. We also need to translate our algebraic formulations into code. You will find a range of approaches are presented to perform this translation that provide you with a grounding in common approaches used across aerosol science. Figure 1.5 provides a schematic that maps processes on our book structure by highlighting the relevant chapters that focus on each in turn. Whilst we isolate processes including nucleation and coagulation in distinct chapters, we also cover properties and bridging technologies that are not explicitly highlighted in Equation (1.4) but are nonetheless important to providing a solution to it. These include developing models that capture condensed phase thermodynamics and simulate diffusion within a particle, simulating a reactive gas phase and presenting the structure required to build a simple model of aerosol to cloud droplet activation. A synopsis of each chapter is provided below. At the beginning of each chapter you will find an overview of the language chosen and use of any functions provided within that environment. You will also find that, in some chapters, we combine a breakdown of translating theory to code with instructions on how to run existing community models which have been written in a variety of programming languages.

- Chapter 1: Following on from an introduction to aerosol science, here we discuss general concepts around representing aerosol particles as a numerical model. This encourages you to consider what information an aerosol model may need to carry and how we then map that into a numerical model through the programming language constructs. For example, at the single particle level we may wish to construct a model that simulates the partitioning of mass between a gaseous phase and homogeneous droplet. If we have n compounds in the gas phase and one particle, then it is reasonable to design a model built around a one dimensional array of size $n + 1$. Of course, once we consider populations of particles with varying sizes and, perhaps, interparticle morphology, then our choice becomes more complex. In this chapter, we therefore present some general considerations around these issues which help contextualize the design choices you will come across in the proceeding chapters. We follow this by introducing our first theory to code demonstrations where we implement two different approaches to represent a population of particles in Python. Known as the modal and sectional methods, this provides a basis for subsequent chapters.
- Chapter 2: In this chapter, we present and implement theories that allow us to simulate movement of mass between a gas and condensed phase. We first introduce equations that provide the basis for predicting the composition of both the gas and condensed phase at equilibrium. This is split between the processes of adsorption and absorption and considers both the particulate and gaseous phase to be nonreactive. Up to this point we assume our particulate phase to have no size, considering a total mass that compounds in the gas phase can adsorb or absorb to. Following on from this we then move to simulating dynamic absorptive partitioning by introducing and solving the droplet growth equation. Whilst both our gas and particulate phase remains nonreactive, we use Python to simulate the growth of both mono- and polydisperse populations as a function of time. The size and composition of the simulated aerosol particles influence the partitioning process through a change in equilibrium pressure above the condensed phase, but the condensed phase components do not interact with each other. We use both the modal and sectional distributions covered

in Chapter 1, where we compare Python and Julia models for the sectional approach. In each case, when designing our model structure, we also need to be aware of how we utilize any specific solver routines. In this case, we use existing ordinary differential equation (ODE) solvers within popular Python and Julia packages. Attention is then given to predicting aerosol water uptake through equilibrium frameworks, at the single particle level, below and above 100% relative humidity.

- Chapter 3: In this chapter, we introduce theoretical frameworks that underpin thermodynamics and nonideal mixing. Whilst in Chapter 2 we assumed our aerosol particles were constructed of a homogeneous ideal mixture, here we account for nonideal mixing. We specifically move to treating nonideal interactions that dictate the predicted equilibrium state and, in some cases, lead to phase separation in the particulate phase. You will see the use of Python again in constructing simulations of simple mixtures and the required structure of our code. You will also find information of how to use an existing community model to simulate phase separation in complex mixtures, written in Fortran. Once again, translating the relevant theory into code here requires some consideration of an appropriate model structure. In Chapter 2 we discuss separation of aerosol particle size as we design the arrays that will track information on aerosol composition through the simulation; likewise here we need to consider how we track the composition of our particle.
- Chapter 4: In all previous chapters, we have used a nonreactive "static" gas phase. As aerosol particles reside in a gaseous medium, they are subject to processes that are driven by the availability of compounds in said medium. Likewise, the availability of gaseous compounds is driven by the complex chemistry that unfolds as compounds react with each and a range of oxidants. In this chapter, we begin with an example of how to simulate, and thus track, the variable concentration of compounds in a gas phase. We discuss the concept of a chemical mechanism in the context of a file that holds information about the interaction of compounds in the gas phase. We then use tools in Python to extract information in these files and create a code structure that allows us to simulate the concentration of each compound as a function of time. Once again we consider an appropriate structure that is driven by the information we wish to track and our chosen ODE solvers. With an evolving gas phase, we can also consider the properties of individual compounds that dictate gas-to-particle partitioning. Predicting those properties requires us to map an algebraic form of a predictive technique to a chemical structure. You will therefore find the use of Python and existing informatics packages to extract and automate the prediction of properties for many thousands of compounds. With all of the previous work on required code structures and solvers for simulating condensational growth and a reactive gas phase, we also provide a demonstration of how to use existing community-driven models that have been designed to automate the process from reading a chemical mechanism file and then creating a model that will simulate the evolution of a coupled particulate and gas phase, written in both Python and Julia.
- Chapter 5: If the population of our particles have different velocities, there is a chance they will collide. Two particles colliding to produce a larger particle reduce the total number concentration over time. This process is called coagulation and is influenced by a number of factors including ambient conditions, the concentrations of aerosol particles and their phase state. In this chapter, we develop stochastic and deterministic representations of the coagulation process in Python. In all other chapters,

we have built deterministic models. As part of this chapter's development of the stochastic and deterministic representations, we discuss the impact of design choices on computational efficiency, including a comparison between a modal and sectional approach. Following on from this we start to consider more complex coagulation scenarios including, for example, the treatment of nonspherical particles.

- Chapter 6: Whilst Chapter 2 focuses on gas-to-particle partitioning to an existing condensed phase, in this chapter we present underlying theories that are used to predict new particle formation. This is referred to as aerosol nucleation. As Chapter 5 treats the interaction between aerosol particles, here we move our focus to the molecular level. We contextualize this work in terms of aerosol size ranges we have met in all other chapters and now consider clusters of molecules as discrete units. The boundary between a molecule and an aerosol particle becomes blurred, but the challenge to design an appropriate numerical model remains. We build a Matlab solution to a discrete form of the "birth-death" equation for molecular clusters. You will find a discussion on how models we develop have a place and dependency in a wider ecosystem of numerical models. In this instance, that specifically includes those conducting molecular dynamics and quantum chemistry simulations. Moving beyond the Matlab based examples we build here, you will also find an introduction and tutorial on using the Atmospheric Cluster Dynamics Code (ACDC) which has components written in the languages Perl, Matlab and Fortran.
- Chapter 7: Chapters 1–6 can be looked at in isolation and the code examples allow you the reader to further develop them or integrate them into other software. Of course, we know from the general dynamic equation that we wish to connect these process descriptions such that we can simulate the life-cycle of an aerosol particle, or population of particles. In this chapter we therefore introduce the concept of a box-model and present a numerical and code design strategy to integrating nucleation, coagulation and condensational growth. Focusing on a sectional approach, you will find that the mechanism for tracking particle size presented in Chapter 2 has limitations when we wish to include nucleation and coagulation. We present methods for restructuring our numerical arrays in Python. We finish this chapter by assuming our aerosol distribution is within a rising parcel of air which leads to increasing relative humidity above saturation and leads to formation of cloud droplets. In this manner you will learn how to create a cloud parcel model and lay the foundation for another large area of research in capturing and predicting cloud micro-physics from a population of aerosol particles. In the end of the chapter, we will present a box aerosol-cloud model SALSA which comprises atmospherically relevant micro-physical processes which interact with aerosol particles, cloud droplets, precipitation droplets and ice nuclei.

Translating theory into code requires us to consider how we can represent information on the properties of the aerosol system we are interested in. This is influenced by choice of programming language, which in turn may be driven by the availability of numerical methods or your reliance on existing legacy code that may have been developed in your community. This also fundamentally requires consideration of how we represent aerosol particles in numerical models. Whilst each chapter provides you with a grounding in common approaches to solving the relevant process, with this in mind, in Section 1.2 we provide you with a brief overview of computer programming

languages used in this book, should this be needed. Following this, in Section 1.3 we then present a general consideration of how we represent aerosol particles and their physical and chemical characteristics within numerical constructs in our software. This is designed to provide extra context to the content you will find in each chapter.

1.2 Computers and Programming Languages

A computational model is built around a set of rules; a single of set of algorithms. The interface between the aerosol scientist, who has drawn up these rules, and the computer is provided by a programming language. There are many programming languages available, each with its own set of advantages and disadvantages depending on what the problem is you wish to solve. There are general "rules of thumb" that can be followed in order to select the most appropriate language, and the more time you spend coding and exploring the multitude of applications available, the clearer this choice will become. One should avoid the pitfalls of snobbery that may surround software development, especially if you are starting your journey on becoming an aerosol model developer. If you select relatively simple examples to start practicing translating theory into code then you are free to pick a range of languages. True that some are faster, and others have a much broader ecosystem (e.g. for visualization, connecting numerical simulations to machine learning libraries etc). However, it is important to enjoy and explore the world of programming as best you can. When you start to require that element of speed, or specific dependencies, then you can revisit which language you wish to develop in. In this book you will indeed come across a range of languages. Python is used throughout the book, but we also present examples in Fortran, Julia and Matlab. These are influenced in part by the existing language in which aerosol community models are currently based, but it also provides us with an opportunity for you to explore these varying languages.

In order to appreciate the differences in computer languages, we need to understand a little about the hardware we wish to run our aerosol models on.

Let us start from the top. Imagine we have received a blue print of a high performance computing (HPC) center with plans for a cluster, nodes and individual processors. As illustrated in Figure 1.4, a **cluster** will have several nodes (represented by each yellow box); a **node** can be thought of as a motherboard with one of more sockets; a **socket** contains a CPU processor (represented by each blue box); and a **CPU processor** may have one or more **cores** as represented by each black box in the diagram. Much closer to home, you may have a laptop or desktop computer at your disposal that likewise contains a motherboard with (probably) one socket; the **socket** contains a CPU processor; and the **CPU processor** may have one or more **cores**. Your desktop will likely have fewer cores than the CPU processor in the HPC center. The CPU and other components will have been manufactured by a set of popular vendors that include Intel, AMD, ARM and IBM. As Section 8.3.2 discusses, in a cluster the nodes are connected by some form of **interconnect** as illustrated in orange in the figure.

We also need to consider how a high-level programming language that humans can read and understand ends up getting run on a computer.

Traditional computers (as opposed to the likes of quantum computing discussed in Section 8.3.3) have a very limited set of instructions they recognize. This is known

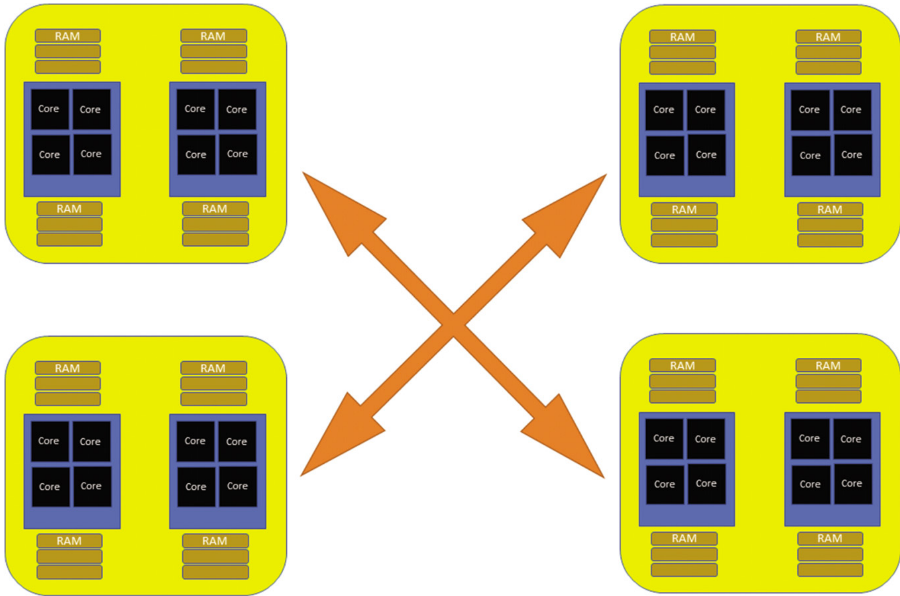

Figure 1.4 Illustration of hardware components of an HPC cluster.

formally as the **Instruction Set Architecture** (ISA) and it is necessary to translate the high-level programming language in to **machine code** that adheres to a given CPU processor (since we may presume all cores on a given CPU processor have the same ISA).

There are two approaches to translating from the chosen programming language to machine code. **Compiled languages** do this translation in a separate step prior to running any of the code. Having compiled the source codes to an **executable** (shorthand for "executable file") for a given CPU processor, the executable can be run again and again on that CPU processor architecture, with further compilation only required when you make changes to the source codes. This is in contrast to **interpreted languages** which translate each line of code as it is required during the actual running of the source code. Many popular languages are "interpreted," which can be thought of as a basic "compile each line of code as we want to run it." From a functional viewpoint there is nothing wrong with this. Many interpreted languages are highly popular, due to their ease of learning, including Python.

Table 1.1 illustrates and compares these two approaches.

The main compiled languages are C, C++, Fortran and popular interpreted languages that include Python and Julia. You may be wondering why there is a choice and perhaps which you should use. In terms of which can be used to implement the algorithms discussed in this book, it doesn't matter—you can code each algorithm in any or all of these languages. But the ease of doing so, and the resulting performance, may vary. There are some pros and cons relating to compiled versus interpreted languages, as listed in Table 1.1. Programming languages themselves are based on different approaches, and most evolve over time. For example, Java was designed with the "object oriented" (OO) style in mind, where the focus of programming is the consideration of the hierarchical classes of objects and the various "methods" operating upon them.

Table 1.1 Comparison of compiled and interpreted languages.

Comparison of approaches	
Interpreted languages	**Compiled languages**
–	Requires a compiler
Translated line by line at run time	All of code is compiled in separate stage before running
A line may be translated many times	–
No opportunity to see bigger picture	Compiler can analyze full code to make deep optimizations
Generally felt to be easier to learn	Generally felt to give better performance
Can use same source on different platforms	Need to compile for each ISA

Other languages such as C++, Julia, and Python (and to some extent modern Fortran) also now support OO programming.

Most programming languages scientists encounter are **imperative languages**, where the control is implemented by a series of statements that manipulate the data of the simulation. Fortran, C, C++, Julia, Python, and MATLAB are all imperative programming languages. The alternative is where it is the flow of data that is described by the programmer, leaving the implementation of how this is achieved to a compiler. These are known as **functional languages** and include Haskell, Lisp, Erlang, and Clojure. Data flow is key to programming concurrently and there is growing interest in use of functional programming for FPGAs (see Section 8.3.3 in Chapter 8) and highly concurrent systems.

Let us take a quick look at the languages used in this book in terms of their provenance and use. We reiterate that the best way to learn and become more familiar with each is to solve a particular problem, which you can do across the chapters in this book.

- Python [Official documentation: https://www.python.org/]: First released in 1991, Python is a universally popular language finding use across not just scientific domains, but from database design to web development. At of the time of writing, Python is released as version 3.8 with support for packages built in version 2 discontinued. Python is an interpreted language and is often regarded as the best choice to start programming. This is in part related to the readability of the code, but also the huge ecosystem of tools and facilities that you can now integrate with your own personal developments. For example, the ability to integrate with chemical informatics tools [25], machine learning [26], or powerful visualizations [27] is an attractive prospect for the multidisciplinary scientist with a number of examples used across aerosol science. Whilst speed has been a concern when developing relatively large numerical models using Python alone, Python can also act as the "glue" to connect libraries built in other languages whilst the Numba "High Performance Python Compiler" [28] translates Python functions to optimized machine code at runtime with minimal effort. Even if a researcher does not work directly with any form of numerical models, Python modules such as Pandas offer tremendous flexibility in

ingesting and manipulating data. Similarly, Python provides an interface to a range of popular machine-learning modules such as Scikit-learn [26] and Keras [29]. Perhaps the most common route to installing Python on a personal machine is to use the Anaconda distribution (https://www.anaconda.com/products/individual), which comes with a flexible package manager called `Conda`. The Conda package manager can be very helpful for those who wish to build an environment with a number of modules that need to be installed separately or are not included in the default Anaconda distribution. This is because Conda manages the required dependencies and will install additional modules where required. Alongside Conda there is also the `pip` package installer which can be useful where there is no Conda channel for a particular module. There are multiple ways you can interact with and use Python for your own projects, but they all require us to write some code! For a graphical user interface (GUI) experience, the Spyder Interactive Development Environment (IDE) combines a text editor with interactive console and variable explorer (https://www.spyder-ide.org/). Jupyter notebooks are hugely popular environment for teaching Python. A Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text (https://jupyter.org/) and is not limited to Python. Throughout this book we will provide you with Python examples written as individual text files that have a *.py* extension. Imagine we have created a file called *test.py*. Once you have a distribution of Python installed, we would open up a terminal window, or Anaconda prompt, and ask the Python Interpreter to run our file as follows:

```
python test.py
```

- Fortran [Official documentation: https://fortran-lang.org/]: The Fortran language has a rich history of, to date, more than six decades as a high-level language, with special emphasis on being a structured, compiled language offering good optimization potential for performance-critical numerical programs and libraries (e.g. 26). In particular, the FORTRAN 77 language (name at those times written in capital letters) and the substantially revised and improved Fortran 90 standard (released in the year 1991; perhaps the origin of "modern" Fortran) have found wide-spread applications in many fields of science and engineering and have been a popular choice for decades in those disciplines. The Fortran language also keeps evolving, with a new standard typically being released every five to ten years. The more recent versions of Fortran, described by the Fortran 2003, 2008, and 2018 standards, have added improved support of dynamic data types, object-oriented programming, new intrinsic array functions, native parallel programming, standardized interoperability with the C language, and clarified/revised language definitions (see https://wg5-fortran.org for official language standard publications and descriptions of added features). The standardized interoperability with the C language also supports improved interoperability of Fortran modules with Python (e.g., via the "f2py" method within Python's NumPy library).

  One valuable feature of the Fortran language development and revision process is that it remains backward-compatible with older standards and programs (so long as they were standard-conforming at their time). In the geoscience disciplines, Fortran programs have been at the core of numerous applications; many of the computationally expensive, highly parallelized programs for operational weather forecasting,

atmospheric chemistry and transport modeling, and Earth system or climate simulations have been written in Fortran.

As Fortran is a compiled language we need a compiler. The choice of compiler may depend on the operating system and your personal preference. For example, Fortran code presented in this book has been developed on both the Microsoft Windows and the Linux platforms. Under Windows we have chosen to use the Intel® Fortran compiler, known as "ifort", via integration into Microsoft Visual Studio Community 2019 (which provides an edit-compile-run IDE (see below)). For students, the Intel® Fortran compiler is available for free as part of the Intel® oneAPI HPC Toolkit. We also compile and execute example programs using the free GNU "gfortran" compiler on a Linux environment.[1] In this case we create a text file, say `test.f90`. If we were to use the gfortran compiler, then within a terminal or command prompt we would create an executable file `example.out` by entering the following command:

```
gfortran  -o example.out test.f90
```

Upon successful compilation, we can run this executable within the Linux terminal by entering the command `./example.out`. In this very simple example we have omitted a number of additional compiler options that can tell the compiler what level of optimization is required, which we cover in more detail in Section 8.2.1.

- Julia [Official documentation: https://julialang.org/]: At the time of writing, Julia is a relatively new programming language positioned at version 1.6. There is growing interest in Julia as it was created with the understanding that *Scientific computing has traditionally required the highest performance, yet domain experts have largely moved to slower dynamic languages for daily work* (https://julia-doc.readthedocs.io/en/latest/manual/introduction/). Indeed, once you start creating code in Julia, you may find a number of similarities to the Python syntax though the speed of the code, thus time-to-solution, can approach that of Fortran [31]. Julia achieves this using just-in-time (JIT) compilation, where the code we write is compiled to machine code during execution of a program rather than before execution. There are a number of very useful features provided in Julia, including the ability to implement automatic differentiation of the code you write. Likewise, whilst the ecosystem of tools in Julia may not match that of Python, for numerical computing there are over one hundred differential equations solvers available. Like Python, Julia also has a package manager to help integrate a range of modules into your workflow. You can install Julia following the instructions on the official Julia web page. Once installed, you can open a Julia console and type `]` to enter Package management state, also known as `Pkg`. Please refer to the documentation for installing packages (https://docs.julialang.org/en/v1/stdlib/Pkg/). Again, we can create a text file that contains Julia code which we then wish to run. Imagine we create a file called `test.jl`. Once Julia is installed on your machine, open up a terminal [Max/Linux] or command prompt and we can run our files as follows:

```
julia test.jl
```

1 Note that on some Linux distributions, "f95" is a synonym for "gfortran".

- Matlab [official web site: https://uk.mathworks.com/products/matlab.html]: Matlab is a proprietary software product that provides a user with an interactive development environment, through a GUI. Developed by Mathworks, Matlab has been used across academia and engineering widely for a number of decades and is particularly useful when prototyping new ideas. When working within the Matlab GUI you will have access to a range of tools packaged in to a series of `toolboxes`, depending on which license you have access to. For example, according to the official documentation, the *Partial Differential Equation Toolbox™ provides functions for solving structural mechanics, heat transfer, and general partial differential equations (PDEs) using finite element analysis.* We can write Matlab code in a text file that has the `.m` extension. In the most recent versions, the user can execute Matlab code from a number of languages, including C/C++, Fortran, Java, and Python. Likewise, Matlab code can call functions developed in other languages. Whilst you can run Matlab code from the command line, it is more common to run code from within the Matlab GUI using the menu options provided (clicking on the run icon provided).

Each language has its own unique syntax. As you move between different languages, you will find some are similar in style and this can make it easier to translate the same theory into multiple languages or convert an existing code base into another language. Products such as Matlab come with their own integrated development environment (IDE), providing a user with a code editor that is able to highlight the variable syntax you use to construct a code file. Alternatively, you may be using the Spyder IDE to develop Python or Microsoft Visual Studio to develop Fortran code that likewise provide syntax highlighting. However, there are a number of alternative text editors that can be used in isolation. **Syntax highlighting** refers to the keywords of the given programming language being highlighted in bold, or as another color, often with auto-repeat and hints on usage. Using an IDE as you code up the many code snippets provided in this book, you will notice that some of the text commands and words used are colored in a particular way. This allows us to more easily identify different structural components to our code. In terms of available text editors, Atom (https://atom.io/) is a cross-platform, free to use text editor that also integrates with Git and GitHub directly (see Section 8.4.3). For Windows users, Notepad++ likewise provides a flexible syntax highlighting environment (https://notepad-plus-plus.org/downloads/). Try a few different text editors; often the choice can be somewhat personal depending on the machine and how your project evolves.

1.3 Representing Aerosol Particles as Model Frameworks

In the previous section, we provide a brief overview on the physical and chemical characteristics of atmospheric aerosol particles and the subsequent processes covered in this book. As you read each chapter, you will find a specific method for translating the theory that underline these processes into code.

There are multiple approaches to represent an aerosol particle, and population of particles, within a numerical model. The information that we represent and therefore track throughout a simulation brings together layers of chemical complexity and metrics that represent the physical state of the condensed phase. Likewise, we have to