Jörn Behrens

Adaptive Atmospheric Modeling
Key Techniques in Grid Generation, Data Structures, and Numerical Operations with Applications

With 74 Figures and 3 Tables

Springer
Für Katja, Laila und Janka.
Ihr seid das größte Glück in meinem Leben.
Preface

This work represents the essence of nearly 16 years of work in scientific computing for ocean and atmospheric modeling. This journey started at Alfred-Wegener-Institute for Polar and Marine Research in Bremerhaven, Germany, where – as a post graduate – I was assigned to optimize multi-grid solvers for elliptic partial differential equations evolving from ocean modeling. I am lucky and grateful that I had the chance to go all that way, to have the opportunity to explore the subject from many different angles, to have wonderful teachers and colleagues, and to have the chance to work and visit such exquisit places as the National Center for Atmospheric Research in Boulder, Colorado, USA, the Frontier Research System for Global Change at the Yokohama Institute for Earth Science (Earth Simulator) in Yokohama, Japan, the Max-Planck-Institute for Meteorology in Hamburg, Germany, the aforementioned Alfred-Wegener-Institute for Polar and Marine Research in Bremerhaven and Potsdam, Germany, the Technische Universität München in Garching, Germany, the Fields Institute in Toronto, Canada, the Naval Research Laboratory in Monterey, California, USA, the Department of Informatics at the University of Bergen, Norway and the Centre for Mathematical Sciences at the University of Cambridge, UK, to name only the most influential ones.

My own interest in adaptive methods arose from the exploration of finite element methods. When comparing finite elements on simple model problems with finite differences, the former method is often disregarded for computational efficiency and implementation complexity issues. However, finite element methods are much more versatile and flexible when it comes to irregular domains and locally refined meshes. Besides, finite elements are mathematically the more elegant approach. Being an aesthete, the wish for a method that fully unfolds the beauty of finite element methods was created. That was the start for the development of the adaptive semi-Lagrangian finite element method, better to be called adaptive Lagrange-Galerkin method [34]. Since then, my research was focused on adaptivity and the solution of geophysical fluid dynamics problems with advanced adaptive numerical methods. This was
during my PhD period at Alfred-Wegener-Institute (AWI), where a prototype implementation of a shallow water solver was accomplished.

With a grant from the Federal Ministry of Education and Research (BMBF), I started the development of a parallelizable adaptive mesh generation tool for oceanic and atmospheric applications in the mid 1990’s. At that time there was no such tool available. This work was continued after funding ran out by an internal grant from AWI, before I decided to enlarge my scientific background by changing to Technische Universität München (TUM). The time at TUM was great in that it provided me with a lot of new knowledge in numerical analysis. I owe my teacher Folkmar Bornemann a great debt of gratitude for his patience and his clearly structured and precise way of teaching.

By now, several groups have gained a lot of experience in adaptive modeling. Yet in atmospheric sciences, the number is still limited. To fulfill my requirements of a German Habilitation, I considered to just compose some of my articles and reports for a short written document of my work in adaptive atmospheric modeling. However, thinking again, I am now convinced that taking the chance of having to write a monograph, is the best excuse for doing this a bit more carefully and summarizing what has been done in adaptive atmospheric modeling so far. I am aware that this snapshot can only be incomplete. However, the references and approaches mentioned may at least give a good starting point for research in adaptive atmospheric modeling.

Acknowledgements

The work of the last few years has been supported by different sources, which I am grateful for:

- Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF), grant no. 07/VKV01/1.
- BMBF, grant no. 01 LD 0037 within the DEKLIM research program.
- Deutsche Forschungsgemeinschaft (DFG), grant no. BE 2314/3-1.
- DFG travel grants, no’s. BE 2314/1-1, BE 2314/2-1, and BE 2314/4-1.
- Deutscher Akademischer Austauschdienst (DAAD) PPP-Norway grant no. 02/29189 (with Tor Sørevik).

Without money no work can be done, but even more important are the people that had influence on my research.

Wolfgang Hiller of Alfred-Wegener-Institute, Bremerhaven, created the scientific freedom to start with the development of adaptive software for atmospheric modeling. The development of the software package *amatos* has been tremendously accelerated by Natalja Rakowsky, (Technische Universität
Hamburg-Harburg, formerly Alfred-Wegener-Institute, Bremerhaven), Lars Mentrup, Florian Klaschka (both Technische Universität München), Matthias Läuter (Alfred-Wegener-Institute Potsdam, he contributed figs. 8.9 and 8.21), Stephan Frickenhaus (Alfred-Wegener-Institute, Bremerhaven), and Thomas Heinze (Deutscher Wetterdienst, formerly TU München). I thank my students who contributed single pieces in the whole adaptive modeling puzzle.

I owe gratefulness to Folkmar Bornemann (TU München), who supported my work at TU München tremendously. Tor Sørevik’s (University of Bergen, Norway) support and his suggestions to the first version of this manuscript are gratefully acknowledged. I obtained several reviews by Joseph Egger, Frank Giraldo, Thomas Heinze, Matthias Läuter, Lars Mentrup, Volkmar Wirth, and anonymous reviewers that improved the text tremendously.

Many fruitful scientific discussions as well as very nice non-scientific experiences are owed to my friend Francis X. Giraldo from Monterey, CA, USA. Dave Williamson made my visit at NCAR, Boulder, CO, USA, possible and his mentorship was very valuable to me. The inspiring atmosphere in Annick Pouquet’s group, including Aimé Fournier and Duane Rosenberg, helped to clarify ideas and concepts. Rich Loft, Stephen Thomas, Ram Nair, Amik St-Cyr among others, all contributed to the scientifically fruitful stay at NCAR.

Finally, and most importantly, my family gave me the support and strength to keep the direction in all these years. I am grateful for the chances that I were offered and the experience I could gain.

Munich, Jörn Behrens
April 2006
4.5 Example for a Grid Handling Software Package – amatos

5 Issues in Parallelization of Irregularly Structured Problems
5.1 Partitioning Strategies
  5.1.1 Spectral Bisection Partitioning
  5.1.2 Kernighan-Lin Partitioning
  5.1.3 Multi-Level Methods
  5.1.4 Space-Filling Curves

6 Numerical Treatment of Differential Operators on Adaptive Grids
6.1 Approximating the Gradient
  6.1.1 Finite Difference Approach
  6.1.2 Least Squares Approximation
  6.1.3 Galerkin Approach
  6.1.4 Radial Basis Function Approach
6.2 Evaluating the Gradient Approximation
  6.2.1 Further References

7 Discretization of Conservation Laws
7.1 Conservation Laws of Interest
  7.1.1 Conservation of Mass
  7.1.2 Conservation of Momentum
  7.1.3 Conservation of Energy
  7.1.4 Equations of State – Closing the System
7.2 Finite Volume Methods
  7.2.1 One-Dimensional Model Problem and Basic Algorithmic Idea
  7.2.2 Multidimensional Extension
  7.2.3 Convergence, Consistency and Stability
7.3 Discontinuous Galerkin Methods
  7.3.1 The Basic Idea for Ordinary Differential Equations
  7.3.2 Application of DG Methods to Multi-Dimensional Scalar Conservation Laws
7.4 Conservative Semi-Lagrangian Methods
  7.4.1 Adaptive Semi-Lagrangian Scheme
  7.4.2 Mass-Conserving Algorithms

8 Example Applications
8.1 Tracer Advection
  8.1.1 Filamentation of Trace Gas in the Arctic Polar Vortex
  8.1.2 Adaptive Tracer Transport Model
  8.1.3 Experiments and Results
8.2 Inverse Tracer Advection
8.3 Shallow Water Equations ........................................ 132
  8.3.1 Vorticity-Divergence Formulation ......................... 133
  8.3.2 Velocity Formulation ....................................... 136
8.4 Wave Dispersion .................................................. 138
8.5 Test Cases .......................................................... 143

9 Conclusions ......................................................... 153
  9.1 Evaluation of Methods ........................................... 153
  9.2 Road Map for the Next Five Years ............................... 155

Some Basic Mathematical Tools ................................. 161

Metrics for Parallelizing Irregularly Structured Problems .... 163

Rotating Shallow Water Equations in Spherical Geometries ... 167

List of Notations ...................................................... 173

References ............................................................ 175

Index ................................................................. 201
List of Figures

1.1 Meteosat satellite image ........................................ 2
1.2 Lee wave cloud pattern ........................................... 3
1.3 Locally refined grid over Antarctica ........................... 4

2.1 Locally adapted mesh for tracer dispersion .................... 10
2.2 Locally adapted mesh for error equilibration .................. 11

3.1 Hanging nodes ....................................................... 25
3.2 Bisection of marked edge ........................................... 28
3.3 Iterative refinement by bisection ................................ 29
3.4 Admissible patch .................................................... 29
3.5 Regular triangular refinement ................................. 30
3.6 Green and blue refinement ....................................... 30
3.7 Quadrilateral refinement algorithm ............................. 32
3.8 Closures for quadrilateral refinement ......................... 33
3.9 Quadrilateral patch refinement ................................... 34
3.10 Closures for quadrilateral patch refinement .................... 34
3.11 Generalization of closures for quadrilateral patches ......... 35
3.12 Voronoi cells and Delaunay graph .............................. 36
3.13 Delaunay triangulation ............................................ 37
3.14 Local refinement of Delaunay triangulation .................... 37
3.15 Tetrahedron with marked edge in standard position .......... 38
3.16 Bisection of a tetrahedron ....................................... 39
3.17 Red and Black tetrahedra ......................................... 40
3.18 Hexahedral refinement ............................................ 41
3.19 Octahedral triangulation of sphere ............................. 42
3.20 Icosahedral triangulation of sphere ............................ 43
3.21 Truncated icosahedral triangulation of sphere ................. 44
3.22 Lat-Lon and reduced Lat-Lon grid ............................. 46
3.23 Capped and cubed sphere grids ................................ 47
3.24 Yin Yang grid ..................................................... 48
4.1 Two phases of an adaptive algorithm .......................... 50
4.2 Example grid with numbered nodes/elements 51
4.3 Example of a refinement tree .......................... 53
4.4 Triangular mesh numbering and connectivity matrix .......... 56
4.5 Cuthill-McKee and minimum degree reordering ......... 57
4.6 Space filling curve in bisection refinement .............. 58
4.7 Space filling curve reordering .......................... 59
5.1 Examples of graphs related to meshes .................. 73
5.2 Examples of partitioned adaptively refined meshes .... 78
6.1 Piecewise linear basis function \( b_i \) .................. 84
6.2 The basic test problem for gradient computation ......... 87
6.3 Convergence of the gradient approximation methods .... 88
6.4 Gradient error plots for uniform mesh .................. 89
6.5 Gradient dependence on non-uniformity of mesh ........ 89
6.6 Gradient error plots for non-uniform mesh ............. 89
6.7 Timing results for the gradient approximation .......... 90
7.1 Sketch of the shallow water set up ..................... 99
7.2 Idea of the finite volume method ..................... 101
7.3 The dual cell ............................................. 114
7.4 Upstream dual cell in cell-integrated scheme .......... 116
7.5 Quadrature points in a triangle ...................... 118
7.6 Definition of volumes in the MPSLM .................. 120
7.7 Mass packet refinement within one cell ............... 120
7.8 Mass packet mapping step .......................... 121
8.1 January monthly mean geopotential (m) for the Arctic .... 125
8.2 Adaptively refined grid for advection example ........ 126
8.3 Initial configuration for the artificial model problem .... 127
8.4 Filamentary results of advection with artificial model problem 128
8.5 Initial tracer distribution for the experiments .......... 128
8.6 Tracer concentration using high and low resolution wind data . 128
8.7 Tracer concentration for different horizontal resolutions .... 130
8.8 Inverse tracer advection modeling result ................ 133
8.9 Simulation of Rossby waves with PLASMA-P ............ 136
8.10 Williamson test case 2 initial values .................. 138
8.11 Time series for Williamson test case 2 ................ 138
8.12 Adaptive meshes for Williamson test case 2 ............ 139
8.13 Williamson test case 5 situation after 15 days ......... 139
8.14 Williamson test case 5 adaptively refined mesh ........ 140
8.15 Basis functions for wave dispersion example .......... 141
8.16 Domain for wave dispersion example .................. 142
8.17 Time series of grids for wave dispersion example ..... 143
8.18 Accelerating wind test case in 2D .............................................. 146
8.19 Accelerating wind test case in 3D .............................................. 146
8.20 Initial configuration convergence test case ................................. 147
8.21 Velocity vectors and geopotential height contours
  for test case 8.5.8 ................................................................. 150
List of Tables

3.1 Naming convention for adaptive grids ......................... 27
6.1 Common radial basis functions .............................. 87
8.1 Grid resolution and type for advection example ............ 126
Adaptive modeling in atmospheric sciences has evolved to a state of maturity that it seems to be the right time to summarize what has been achieved so far and to sketch the near future of research directions. This work gives an overview of current approaches to adaptive atmospheric modeling. The author has included material and results cited from other sources in order to give a broader overview of the different approaches. It is clear that his own work is described in more detail, even if this might not in all cases be the mainstream in technological development. Many of the achievements of the author’s group are reported in a way that tries to be as understandable as possible, yet detailed enough to be reproducible. However, if in doubt, readability was given the preference.

In this introductory chapter some motivation for aiming at adaptive atmospheric modeling is given. This is essentially a collection of arguments, the author came across defending his approach against traditionalists especially in the early 1990’s when adaptive modeling was still a small academic exercise and widely disregarded. However, it is not only a chain of arguments, but also a reasoning, why it might be advisable to take the challenge of additional programming complexity, computational overhead and mathematical sophistication to achieve consistent and efficient adaptive models.

When collecting the references, publications and talks on adaptivity, it turns out that many people did in fact think about adaptive methods for atmospheric problems. So, we give a (certainly not concise) list of people and approaches in adaptive atmospheric modeling. As a historic remark on adaptive mesh refinement we refer to Babuška’s pioneering work [15], which was not related to atmospheric modeling. He coined the term of self-adaptation, which terms a method that adapts itself to the solution properties, while it computes the solution. We will omit the prefix “self” and call such methods adaptive methods.
1.1 Why Adaptivity?

When looking at atmospheric phenomena, one almost always observes interaction of a variety of scales. Looking at a satellite image (fig. 1.1), one can observe fine scale structures like a cyclone over the Atlantic ocean with filamentary structures and fronts that comprise a relevant length scale of approximately 5 to 10 km, while the global scale that drives the dynamics is of five orders of magnitude larger. Another example is depicted in fig. 1.2. Here we see clouds forming behind orographic features (a mountain range). The mountain range covers an expansion of the order of 10 to 20 kilometers. The typical lee wave cloud has a scale of several kilometers while the high frequency perturbation visible in the cloud patterns at the top of the picture have length scales of only a few tens of meters.

Adaptivity can in fact help to resolve local (small) scales that interact with global scales in one consistent way. It is however paramount to understand, that an adaptive method is only as good as the refinement criterion that controls adaptivity. The criterion has to be able to capture and predict the occurrence of critical features, while these are evolving as sub-grid processes. On the other hand, an adaptive method cannot be better than the underlying model. If the model equations don’t consider turbulence, even a fine grid will not be able to reproduce turbulence.

In contrast to locally refined fixed mesh methods, adaptive refinement can capture moving features, like meandering jets or moving fronts. Again, this requires an \textit{a posteriori} dynamic refinement criterion. While prescribed refinement regions can rely on the knowledge of experts, adaptivity requires a formal description of the criterion for refinement.

Adaptive methods can help to speed up the computation by avoiding unnecessary calculations and saving memory. This is especially true for localized
1.1 Why Adaptivity?

Fig. 1.2. Lee waves visible in a cloud pattern at the morning sky over Arches National Park (Utah, USA)

phenomena like tracer clouds, point sources or isolated vortices. On the other hand, the adaptation control requires some overhead for the refinement criterion, grid management and irregular data access. To our experience, even if the saving is not worth the effort time-wise, in most cases memory requirements are drastically reduced.

Once a balanced adaptive method is established, it is probably also very efficient on static or even uniform grids. So, in cases where no localized phenomena are expected, adaptivity control can be switched off, reducing the mentioned overhead. Still, these tools are available and can be used in the course of model simulation or analysis. With these methods at hand, one has the possibility to make a quantitative assessment of the quality of solution. Moreover, one is able to control certain inevitable inaccuracies. For example, by increasing the order of approximation (refinement), one can reduce the numerical dissipation of a method (locally) in order to avoid unwanted effects that spoil the solution. An adaptive method also helps in cases, where normally statically refined grids would be used. The user could use the adaptive control during a start-up phase and switch off adaptive control as soon as the method has converged to a stable grid spacing. By this, it can be guaranteed that the region of refinement is optimally chosen.

Irregular boundary shapes often influence the solution of equations describing atmospheric motion. On the other hand, irregular boundary features like mountains, valleys, shore lines, etc. introduce interesting atmospheric patterns that have to be resolved, when accurate modeling is aimed at. Many non-adaptive modeling techniques are not capable of resolving these features well enough. With an adaptive mesh refinement strategy, it is no additional effort to define an orography based refinement criterion that can capture small scale topographic features. An example of this is shown in fig. 1.3, where the shore of the shelf ice can be clearly identified by the refined grid.
It is the author’s vision that we will be able to develop numerical methods that are robust and efficient enough to be combined with adaptivity in standard (operational) environments in atmospheric research. When we look at the development of numerical methods for ordinary differential equations (see for example [120, 121, 182, 183]) we see that today it is just a standard to use adaptive integrators. These methods have reached a state of maturity that nobody really thinks about grids and adaptive control anymore, but the user decides on accuracy and the software fulfills the needs. This is the vision of adaptive atmospheric modeling (and probably adaptive solution of PDEs in general).

Thus, adaptivity triggers the concern about accuracy and local error much more than non-adaptive methods do. Once there is control over local approximation order, one has to decide on a tolerable error measure. At that point the question arises on what “error” really means in a fully equipped atmospheric model.

To summarize, the promise of adaptive atmospheric modeling is the

- locally accurate
- efficient
- error controlled
- truly multi-scale

simulation of atmospheric processes.
1.2 Who’s Who in Adaptive Atmospheric Modeling

Early approaches to adaptive mesh refinement in atmospheric modeling were presented by researchers in Hurricane or Typhoon prediction that nested finer meshes into coarse mesh models of the large scale simulation. One example of this not-yet adaptive approach is documented in [266]. Others are given in [247, 248]. Nesting is still a wide spread technique for achieving local high resolution [157]. However, this review is concerned with adaptive methods, which means methods that dynamically adapt to flow features during run time.

Early truly adaptive atmospheric models were developed by Klemp and Skamarock and by Dietachmayer and Droegemeier [125, 367, 369]. Klemp and Skamarock based their refinement strategy on a truncation error estimate [275].

There are two major adaptation principles, discussed in sect. 2.4. One does not change the grid topology (i.e. the number of grid points and the inter-connectivity) but changes the spacing between grid points by transformation functions [70]. The other principle refines/coarsens the mesh by inserting/deleting grid points and re-meshing (locally). Examples for the first approach are given in [125] and more recently in [77, 215]. The author’s work and much of what follows in this book, follows the second approach.

The first – and to the author’s knowledge only – adaptive and operational weather and dispersion model OMEGA uses a finite volume approach on an unstructured Delaunay mesh that is locally modified and smoothed [20, 174]. OMEGA has been developed by Bacon and coauthors at Science Applications International Corp., McLean, VA. Giraldo based at Naval Research Lab in Monterey, CA, has used Delaunay meshes for adaptively refined Lagrange-Galerkin methods to solve the shallow water equations [166]. Recently, a nodal spectral element method for triangular adaptively refined meshes has been proposed [171]. Iselin and coauthors published a dynamically adaptive version of the MPDATA scheme [215, 214]. MPDATA is a finite difference scheme that has several advantageous characteristics like conservation properties. Prusa and Smolarkiewicz combine a dynamic grid adaptation (movement) method with an Eulerian or semi-Lagrangian conservative advection scheme in [330]. Recently, Smolarkiewicz and Szmelter have presented an unstructured grid formulation of MPDATA [372]. There are several groups involved in adaptive air quality modeling [373, 240, 314].

Jablonowsky’s dissertation, defended at the University of Michigan, is an example of a finite volume scheme on quadrilateral adaptively refined meshes applied to the 2D shallow water equations and the 3D baroclinic equations on the sphere [221]. Additionally, she gives a good introduction to adaptive methods in atmospheric modeling. Barros and Garcia recently introduced a variable resolution semi-Lagrangian model for global circulation with the shallow water equations [26]. Fiedler uses dynamic grid adaptation to resolve the boundary layer [146]. A wavelet approach has been utilized in adaptive ocean
modeling in [223]. And very recently Rosenberg and coauthors demonstrated a parallel adaptive method to study geophysical turbulence problems [347].

In the UK a strong adaptive tradition in atmospheric modeling is represented by publications of Tomlin and coworkers. A fully 3D approach on unstructured tetrahedral meshes is documented in [394] and [163]. Hubbart and Nikiforakis use a 3D quadrilateral locally refined mesh for simulating global tracer transport in [207]. For the ocean Pain, Ford, Piggott and coworkers developed an advanced adaptive modeling tool for 3D unstructured computations [151, 152, 308, 321]. Ivanenko and Muratova introduce a quadrilateral distorted grid based finite difference shallow water model with local mesh refinement [220]. Blayo and Debreu use local mesh refinement in the context of finite difference methods for ocean modeling [61]. Kashiyama and Okada use adaptive mesh generation techniques for solving shallow water flow problems [236]. Again in ocean modeling an adaptive 3D shallow water model has been proposed by Ham and coworkers in the Netherlands [185]. Hundsdorfer et al. proposed an adaptive method of lines based finite difference approach for atmospheric transport problems [212].

In Germany the author’s group is active since the early 1990’s in adaptive atmospheric modeling [33, 34, 35, 36, 39]. Läuter introduced a shallow water model in vorticity-divergence formulation on the sphere [255, 256]. Hess has developed an adaptive and parallel multi-grid solver for the shallow water equations [201].

To summarize, research in adaptive atmospheric (and ocean) modeling is gaining momentum in recent years. This is documented in a study on numerical schemes for non-hydrostatic models that explicitly addresses suitability of schemes to adaptively refined meshes [376]. However, the group of people working on adaptive modeling in this field is still moderate, even if one takes into account that the above list is certainly not complete.

1.3 Structure of the Text

This work is intended to give an overview of adaptive atmospheric modeling techniques in a way that makes it possible to extract the basic ideas for the reader’s own development, together with additional pointers to the more detailed literature. Several aspects are covered in more detail. At the end of this book, a reader should be able to draft a simple adaptive atmospheric modeling technique for his/her own needs and find literature with detailed descriptions on the construction.

Chapter 2 introduces several principles of adaptive modeling. It is of a more abstract and basic character, since adaptive methods are not well established in the atmospheric modeling community yet.

In chap. 3 grid generation techniques are covered in some detail. Grid refinement is the major tool for multi-scale adaptivity in atmospheric modeling
to date. Therefore, some effort has been put into the compilation of 2D and 3D triangular (unstructured) and quadrilateral (structured) mesh refinement strategies. Additionally basic initial grids for spherical geometries are introduced.

Computational aspects are treated in chap. 4. Grid generation – especially for unstructured grids – needs adequate data structures. On the other hand, numerical algorithms need consecutive data structures for efficient execution. Both requirements are covered there.

Chapter 5 treats issues in parallelization. Efficient execution on modern computing architectures calls for parallelizable data structures and dynamic load balancing. In this regard, chaps. 4 and 5 are interrelated. While the description in the former is valid for uniprocessor architectures alike, the latter is only valid for parallel architectures.

Chapters 6 and 7 deal with the more mathematical aspects of adaptive modeling: the representation of differential operators on adaptive grids and the discretization of conservation laws. While conservation laws are the basic equations relevant in atmospheric modeling, they are mostly given in terms of partial differential equations. Thus, the discretization of conservation laws requires differential operators to be represented on adaptive meshes.

Chapter 8 gives examples of the successful application of adaptive methods to atmospheric modeling. Additionally, test cases for studying basic properties of adaptive methods are introduced.

In Chap. 9 we draw conclusions from the material in the previous chapters and try to derive basic rules of good practice. Finally, we try to pave the way to the future of adaptive techniques in atmospheric modeling. It is the authors belief that adaptivity is a basic principle of multi-scale processes. Therefore, adaptive techniques will help to understand and solve future problems in atmospheric modeling.

In the appendix, we collect some of the basic mathematical tools and more technical descriptions that hindered the fluent reading in the chapters described above.
Principles of Adaptive Atmospheric Modeling

In this chapter, major principles of adaptivity and especially adaptive atmospheric modeling will be discussed. We start with reconsidering the paradigms of adaptivity in the mathematical and in the meteorological communities respectively in order to clarify the basic notations. After this clarification, and a description of principle challenges in adaptivity, we introduce concepts of adaptive refinement techniques and discuss refinement criteria.

2.1 Paradigms of Grid Refinement – Resolution Enhancement Versus Error Equilibration

A common cause for misunderstanding when talking about adaptive mesh refinement, or adaptivity in general, is the existence of two different approaches. In order to avoid ambiguity or confusion this section explains the two paradigms.

The first approach is interested in increasing resolution locally. This is the prevailing aim in most meteorological applications, and fixed local mesh refinement has long been used to achieve this goal (see for example [101, 117]). The approach is driven by the physical understanding that increased resolution reveals more structure in the simulated phenomena.

An example for this approach is depicted in fig. 2.1. The locally refined mesh region follows the tracer constituent in a trace gas advection and dispersion simulation. By this measure, the number of unknowns can be kept low, while maintaining high resolution, where this is necessary to capture fine scale features (in this case the point source and the sharp front of the tracer cloud). In fact, the adaptive computation needs between an order of $10^3$ and $10^4$ cells in contrast to an order of $2 \cdot 10^5$ cells for a uniformly refined mesh of same resolution.

The other approach is motivated from the mathematical side. Assuming that a well defined partial differential equation is given, adaptive refinement
of the approximation order or mesh width intends to equilibrate the numerical error. This approach requires that a consistent formulation of the error is available and also that a capable estimation of that error is feasible. It also requires that the numerical method together with the discretization of the problem are both convergent. This might not be the case in real life meteorological applications, where discrete models of sub-grid processes can destroy convergence.

This paradigm was introduced in the early 1970’s probably by Babuška [15, 18]. It was originally used with finite element methods (FEM). For a historical draft of the development of FEM see [90, 390]. An example is depicted in fig. 2.2, namely the solution of Poisson’s equation on a circular domain with a Dirac delta function load (right hand side). This example is taken from the MATLAB PDE toolbox demos [277].

In many cases the two approaches will find similar solutions. Since intuitively, areas of high physical activity will be those of high mathematical error (e.g. steep gradient regions, high local curvature of some constituent). However, there are cases, where the mathematically induced error-driven criterion refines counter-intuitively. This statement is especially valid for transport dominated problems, found in atmospheric modeling, since error can be propagated and the propagation direction may not be predictable. In general, refinement criteria are very problem dependent and have to be chosen after a careful analysis of the problem’s requirements (see sect. 2.6 for a more detailed description of refinement criteria).

2.2 Principle Difficulties with Adaptivity

While an adaptive method promises to be more efficient for certain cases of localized features, it incurs several new difficulties that need to be treated with care. Some difficulties can be circumvented by reasonable choices of software methods, others need to be tackled by mathematically more sophisticated methods. This section lists some of the principle difficulties that will be treated in the course of this presentation.

Let us first look at the computer science part of adaptive methods. Since locally refined meshes are mostly unstructured, indirect addressing and the
corresponding performance loss is an issue. Even in the case of locally structured grids, an overhead of address information has to be taken into account.

When dealing with adaptively refined meshes, advanced software technology has to be used for the dynamically changing data structures. Most adaptive programs utilize object oriented data structures for grid handling. While object orientation is efficient for grid manipulations, it is not well suited for high performance numerical calculations. To achieve high performance, consecutive data structures like arrays or (blocked) matrices are needed.

When applying adaptivity to time-dependent problems, re-meshing may occur in almost every time-step. This results in load-imbalance for parallel computations. So, efficient dynamical load balancing schemes are necessary in order to maintain parallel efficiency of adaptive schemes.

There are also principle difficulties concerning the mathematics of adaptive methods. For explicit time-stepping schemes, the grid size of the finest mesh level dominates the CFL stability criterion. Therefore, either locally adaptive time sub-stepping or an unconditionally stable time-integrator is required, in order to avoid instability.

The most demanding difficulty for adaptive methods is to find a suitable refinement criterion. In general, an adaptive mesh refinement algorithm is only as good as the criterion for refinement. Thus, if one aims at substantial improvement in accuracy of the solution, a good understanding is needed for what accuracy really means in context of the problem and full insight into the reason for error is required. An effective refinement criterion needs to detect those areas that cause the highest error or that are of highest interest, concerning the relevant physical features.

When dealing with different orders of convergence in the same application, sub-grid scale process parameterizations need to be tailored such that they are independent of grid size or polynomial order. Up to now, the interleaving of continuous modeling and discretization has lead to parametrization that is optimal to a fixed mesh.

When coupling an adaptive method to other external models or post processing systems, data often have to be represented on structured regular grids. Interpolation onto such interface grids needs to be consistent with
the modeling error and conservation properties of the physical constituents that are communicated (e.g. fluxes have to adhere to mass conservation).

2.3 Abstract Adaptive Algorithm

In this section we will introduce an abstract formulation of the adaptive algorithm for atmospheric transport-dominated problems. This algorithm is based on *a posteriori* refinement criteria. We will also define basic notations.

It is important to distinguish between *a priori* and *a posteriori* error control. While a priori refinement criteria were used even in the very first paper describing finite element methods [397], a posteriori error control is a later concept.

With *a priori* error control or *a priori* refinement criteria we denote those methods that have prior knowledge about the solution of the problem and react by refining the approximation accordingly. Examples of this are models that refine over areas of interest [101, 117, 287], or in areas of known high turbulent activity, e.g. near steep topographical features (mountains, sea mounts, valleys, etc.) [91, 107, 152, 185, 426].

For an adaptive algorithm, some kind of a posteriori refinement criterion is necessary. The algorithm starts with solving the problem on a provisional grid. Then, a posteriori (i.e. after solving), a criterion determines those regions of the computational domain that need to be refined. Finally, a new solution is computed on the refined grid.

In general the true error is not available during the computation. So, for a mathematical determination of refinement areas, i.e. areas of large local error, an error estimator is needed. The concept of *self-adaptation* has been coined by Babuška [15]. The mathematical error estimation approach requires deep theoretical insight into the nature of error of the numerical method together with the problem to be solved.

A posteriori refinement control can also be derived from heuristical criteria. In many cases gradients of relevant physical quantities or curvature properties are used for refinement. It is also possible to use proxy data in order to obtain a refinement criterion. So, even without deep theoretical error analysis, and even without a clear concept of error, an abstract a posteriori adaptively refined algorithm can be formulated.

Before stating the adaptive algorithm, the basic non-adaptive algorithm should be formulated. For transport dominated atmospheric flow problems, it is assumed that a discretization of the governing equations is performed in space and time separately. Thus, we first semi-discretize in time to obtain a time-stepping scheme, and then discretize the remaining stationary problem in space.

Let us assume that the time interval of interest $I$ is normalized to start at time 0, $I = [0, T] \in \mathbb{R}$. Let us denote the spatial domain with $\mathcal{G} \subset \mathbb{R}^d$ and denote $\rho_h$ the discrete variable corresponding to the continuous variable $\rho$. In
general, $\rho$ defines a map $\rho : G \times I \rightarrow \mathbb{R}^r$, where $r = 1$ for scalar and $r = d$ for vector-valued variables. Then $\rho_h$ defines a map $\rho_h : G_h \times I_h \rightarrow \mathbb{R}^r$. Here $G_h$ is some discrete representation of $G$, for example a polygonal approximation of the true domain, which can be discretized by a triangulation $T = \{\tau_1, \ldots, \tau_M\}$ (see sect. 3.1 for a formal definition of a triangulation). $I_h$ is the discretized time interval $I_h = \{t_i : 0 \leq t_i \leq T(i = 0 : K)\}$. If we deal with systems then $\rho$ and $\rho_h$ represent vectors of all prognostic variables. We formulate the abstract problem by the following equation

$$\frac{\partial}{\partial t}\rho + D\rho = r,$$  

\hspace{3cm} (2.1)

with a differential operator $D = D(t, x, \rho)$ and a right hand side $r = r(t, x, \rho)$. Additionally initial values $\rho(t = 0) = \rho_0$ and boundary values $\rho(t)|_{\partial G} = \rho_b(t)$ are assumed to be given. Furthermore, we assume that the time derivative can be represented as an evolution

$$\rho(t > 0) = \Psi(0, t; \rho(0)),$$

and that the stationary problem can be solved separately and is represented by

$$D\xi = r,$$

where $\xi$ can then be used in the evolution, $\rho(t > 0) = \Psi(0, t; \xi)$. With these assumptions, the basic non-adaptive algorithm reads as follows

**Algorithm 2.3.1 (Basic non-adaptive algorithm)**

1. initialize prognostic variables $\rho_h$, i.e. set values at all relevant time levels $t_i < t_0$: $\rho_h(t_i < 0) = \rho_0(t_i)$;
2. FOR all time steps $t_i \in I_h$, $i = 1 : K$ DO:
3. integrate problem in time interval $[t_{i-1}, t_i]$:
   a) evaluate discrete differential operators to obtain right hand side
      $$r = r(t \leq t_{i-1}, x(t \leq t_{i-1}), \rho_h(t \leq t_{i-1})), $$
      depending only on variable values at times $t \leq t_{i-1}$;
   b) Solve stationary problem
      $$D_h\xi_h(t_i) = r$$
      with $\xi_h$ an intermediate representation of the constituents;
   c) update prognostic variables at time $t_i$:
      $$\rho_h(t_i) = \Psi_h(t_i, t_{i-1}; \xi_h(t \leq t_{i-1})),$$
      where $\Psi_h$ is the discrete evolution operator;
4. perform any other diagnostic calculations depending on values $\rho_h(t_i)$;
5. END DO.
Remark 2.3.2 Note that algorithm 2.3.1 tries to be rather general. For example, an explicit method would have a diagonal system matrix \( S_h = \text{diag}(s_h) \), where \( s_h \) is a vector of coefficients. However, this algorithm does not cover all possible discretization schemes for PDEs as they occur in atmospheric modeling, because it assumes that the discretization leads to a time stepping scheme and that an evolution operator can be derived from the system of equations.

With this formal abstract definition of a time-stepping algorithm, it requires just two additional steps to define a basic adaptive algorithm:

**Algorithm 2.3.3 (Basic adaptive algorithm)**

1. initialize prognostic variables \( \rho_h \) (as in algorithm 2.3.1);
2. FOR all time steps \( t_i \in I_h, i = 1 : K \) DO:
3. integrate problem in time interval \([t_{i-1}, t_i] \):
   a) evaluate discrete differential operators to obtain right hand side
   \[
   r = r(t \leq t_{i-1}, x(t \leq t_{i-1}), \rho_h(t \leq t_{i-1})),
   \]
   depending only on variable values at times \( t \leq t_{i-1} \);
   b) Solve stationary problem
   \[
   D_h \xi_h(t_i) = r
   \]
   with \( \xi_h \) an intermediate representation of the constituents;
   c) update prognostic variables at time \( t_i \):
   \[
   \rho_h(t_i) = \Psi_h(t_i, t_{i-1}; \xi_h(t \leq t_{i-1})),
   \]
   where \( \Psi_h \) is the discrete evolution operator;
   d) calculate local refinement criterion \( \eta_\tau \) for each \( \tau \in T \), and refine those \( \tau \), where \( \eta_\tau > \theta_{\text{ref}} \), with \( \theta_{\text{ref}} \) a given refinement tolerance;
   e) IF grid changed, THEN: return to step 3a, ELSE: continue with step 4;
4. perform any other diagnostic calculations depending on values \( \rho_h(t_i) \);
5. END DO.

Note that only steps 3d and 3e in algorithm 2.3.3 have been added to the original basic algorithm 2.3.1. However, in these two steps, a huge amount of computational intelligence is involved. We will specify the details of the refinement strategy in sects. 2.4 and 2.5 below. Furthermore, grid refinement in geometric terms is discussed in chap. 3, while some remarks on refinement criteria are made in sect. 2.6.

### 2.4 Dynamic Grid Adaptation – h-Refinement

One principle of adaptivity is to refine the computational mesh. This will be the predominant method throughout this text. Commonly, the spatial mesh
size is denoted by $h$, therefore, refining the mesh corresponds to refining $h$. So, a common name for adaptive mesh refinement (AMR) in the mathematical community is $h$-refinement.

$h$-refinement is based on an approximation property of the kind

$$\varepsilon_h = \|\rho - \rho_h\| \leq Ch^\nu \|\rho\|,$$  

(2.2)

where we denote by $\rho$ the true solution of some given problem, $\rho_h$ is a discrete approximation, $C$ is a constant, $h$ is a parameter indicating the mesh width, and $\nu$ is the order of convergence. Equation (2.2) states that for $h \to 0$ the discrete solution $\rho_h$ tends to the true solution $\rho$ or in other words, the error $\varepsilon_h \to 0$.

In the following we want to denote by

- $\varepsilon_h$ the discretization error: $\varepsilon_h = \|\rho - \rho_h\|$ in an appropriate norm;
- $\varepsilon_\tau$ the local discretization error $\varepsilon_h|_\tau$;
- $[\varepsilon_h]$, and $[\varepsilon_\tau]$ an estimation of the global and local discretization error, resp.;
- $\eta_\tau$ or $\eta_i$ for each $\tau_i$, a local criterion that can be an error estimate or some other local measure that can be used to derive a refinement.

There are several strategies to select the refinement regions. Ultimately, one is interested in getting the most accurate solution with least effort. Therefore, an algorithm for refinement has to be given. For now, let us assume that we can compute a local error, denoted by $\varepsilon_\tau$ (we will present common error estimators and refinement criteria in sect. 2.6). A first simple approach is based on the maximum norm of the local error vector $\varepsilon_m := (\varepsilon_{\tau_1}, \ldots, \varepsilon_{\tau_m})$. With $\varepsilon_h = \varepsilon_{\text{max}} = \|\varepsilon_m\|_\infty = \max_{k=1:m} \varepsilon_{\tau_k}$ the global error in the $\infty$-norm, one can try to minimize the global error by refining those elements of the mesh that have largest local errors.

More precisely, with a given tolerance $0 \leq \theta_{\text{ref}} \leq 1$, a simple algorithm for local mesh refinement is given by

**Algorithm 2.4.1 (\infty-Norm based $h$-refinement)**

1. Let $T_0$ be a given initial triangulation, $\varepsilon^0$ the vector of local error components $\varepsilon_{\tau_i}$, $i = 1 : m$, and $\varepsilon_{\text{max}}^0 = \|\varepsilon^0\|_\infty$.
2. FOR all $\tau \in T_0$ DO:
3. IF $\varepsilon_\tau > \theta_{\text{ref}} \varepsilon_{\text{max}}^0$ THEN:
   a) refine $\tau$
4. END DO.

While algorithm 2.4.1 is scaling invariant (i.e. if $\varepsilon^0$ is multiplied by a scalar $\alpha$, then the result of refinement does not change) and extremely simple, there is in fact no theoretical guarantee, that the discretization error is always reduced, or that an algorithm equipped with this refinement procedure really converges. However, in practical applications this strategy has been used successfully (see [39] for its application in a tracer transport problem).
Another $p$-norm based approach is the so called *equi-distribution strategy* \[^{[139]}\]. It is best illustrated by assuming a uniform distribution of local errors $\varepsilon_\tau = \bar{\varepsilon}$ for all $\tau \in T_0$. Now, we define the global error to be the $p$-norm of the local error vector:

$$\varepsilon = \|\varepsilon_m\|_p = \left( \sum_{k=1:m} \varepsilon_{\tau_k}^p \right)^{\frac{1}{p}} = (m\bar{\varepsilon}^p)^{\frac{1}{p}} = m^{\frac{1}{p}} \bar{\varepsilon}.$$  

We require $\varepsilon \leq \theta_{\text{ref}}$. If we let $\varepsilon = \theta_{\text{ref}}$, we can see that $\bar{\varepsilon} = \frac{\theta_{\text{ref}}}{m^{\frac{1}{p}}}$. From this, we derive the heuristic

**Algorithm 2.4.2 (equi-distribution $h$-refinement)**

1. Let $T_0$ be a given initial triangulation.
2. FOR all $\tau \in T_0$ DO:
3. IF $\varepsilon_\tau > \frac{\theta_{\text{ref}}}{m^{\frac{1}{p}}}$ THEN:
   a) refine $\tau$
4. END DO.

Läuter uses this strategy for $p = 2$ in an application to adaptive shallow water modeling in \[^{[256]}\].

Dörfler introduces a guaranteed error reduction strategy \[^{[128]}\]. This strategy – only outlined here, since it is hardly ever used in atmospheric modeling so far – is based on the idea that a subset of mesh elements is refined with the sum of their local errors being a fixed part of the total error $\varepsilon$. Thus, given a parameter $0 < \theta < 1$, find a minimal set $S = \{\tau_1, \ldots, \tau_k\} \subseteq T_0$, such that

$$\sum_{\tau \in S} \varepsilon_\tau^p \geq (1 - \theta)^p \varepsilon^p.$$  \[2.3\]

It follows that if $S$ is being refined, the error will be reduced by a factor depending on $\theta$ and the properties of the problem data (right hand side, etc.). Dörfler proposes a strategy for selecting the elements in $S$ by an inner iteration. In this inner iteration, the $\infty$-norm strategy 2.4.1 is used with a decreasing threshold $\theta_{\text{ref}}$ until $S$ is large enough to meet the requirement in (2.3).

So far, only refinement strategies have been considered. In transport dominated atmospheric modeling the area of interest may move over time, so that we may need to coarsen the mesh after a while. Both of the above strategies 2.4.1 and 2.4.2 can be inverted easily to be used as coarsening strategies. As an example, we demonstrate this for the $\infty$-norm strategy 2.4.1. Let $0 \leq \theta_{\text{crs}} < \theta_{\text{ref}}$. Then one may coarsen an element $\tau$ of the triangulation if

$$\varepsilon_\tau < \theta_{\text{crs}} \varepsilon_{\text{max}}^0.$$  

There are two topics that have to be kept in mind. First, the choice of $\theta_{\text{crs}}$ has to be done carefully in order to avoid incompatible refinement and coarsening.
conditions. These in turn can cause oscillating refinement/coarsening of single elements. In mathematical a posteriori error estimation, one can usually give an approximation of $\varepsilon_\tau$ in terms of the element (mesh) width $h_\tau$:

$$\varepsilon_\tau \leq C h_\tau^{\nu},$$

where $\nu$ is an appropriate exponent. For linear approximation order ($\nu = 1$), coarsening the element such that the mesh width is doubled, causes the local error to increase by a factor of 2. Therefore, we have to choose

$$\theta_{\text{crs}} \leq \frac{\theta_{\text{ref}}}{2},$$

in order to avoid oscillatory behavior. For higher approximation order this argument has to be modified appropriately.

Secondly, we need to define a coarsening strategy, since an element can only be coarsened, if the corresponding siblings are also coarsened. This strategy is described in sect. 3 in connection with the refinement strategies.

### 2.5 Adapting the Order of Local Basis Functions – $p$-Refinement

In contrast to the $h$-refinement of the previous section, where the grid is refined, one can also increase the order of polynomial approximation, while leaving the mesh unchanged. This $p$-refinement, where $p$ stands for the order of approximation, has been developed in the context of finite element methods in the early 1970’s [19, 382]. The foundation for the $p$-refinement is given by a similar approximation property as (2.2). Let $T_0$ be a given (and fixed) triangulation of the computational domain $G \subset \mathbb{R}^d$, and let $p$ be the order of approximation. Then at typical error approximation holds

$$\varepsilon_p = \|\rho - \rho_p\| \leq C p^{\nu} \|\rho\|,$$  \hspace{1cm} (2.4)

where $\rho_p$ denotes an approximation of order $p$ to the true solution $\rho$ and – as in (2.2) – $\nu$ is an appropriate exponent related to the convergence and $C$ is a constant.

In the context of finite elements let $P_p(T_0)$ be the space of continuous functions that are piecewise polynomials of degree $p$ in each element $\tau \in T_0$ and globally continuous. Then for $\rho \in H^k(G)$ there exists a sequence $\rho_p \in P_p(T_0)$, $p = 1, 2, \ldots$ such that for $0 \leq l \leq k$

$$\|\rho - \rho_p\|_l \leq C p^{-(k-l)} \|\rho\|_k.$$  \hspace{1cm} (2.5)

For a proof of (2.5) see [19]. Equation (2.4) – or similarly (2.5) – states that by increasing the order of polynomial approximation the error is minimized, $\varepsilon_p \rightarrow 0$. 

In order to create a sequence of approximations of increasing order, a hierarchy of supporting functions needs to be constructed. In finite element analysis, these hierarchical basis function families are easily constructed starting from the well known linear basis function \[89\]. For demonstration purposes, we give an example of the first three orders of basis functions for 2D triangular \(C^0\) finite elements here \[382\]. For the linear (order \(p = 1\)) element, we define the basis functions \(b_i, i = 1:3\) by

\[
\begin{align*}
    b_1(x_1, x_2) &= 1 - (x_1 + x_2), \\
    b_2(x_1, x_2) &= x_1, \\
    b_3(x_1, x_2) &= x_2,
\end{align*}
\]

where we assume \(b_i\) to be defined on the unit triangle \(\tau_1\) being the convex hull of the three vertices \{(0,0), (1,0), (0,1)\}.

Now, the second order basis functions are composed hierarchically, by the first order basis functions and three additional basis functions normalized by a factor of \(-\frac{1}{2}\) and evaluating the second derivatives of the approximating polynomials at the vertices (since the approximating polynomials are quadratic, the second derivatives are constant). The basis functions are given by

\[
\begin{align*}
    b_4 &= b_1 b_2, \\
    b_5 &= b_2 b_3, \\
    b_6 &= b_3 b_1.
\end{align*}
\]

Finally, the third order basis functions can be constructed by re-using the first and second order basis functions and adding four more terms: the evaluations of the three third derivatives at edge midpoints, a normalizing factor of \(\frac{1}{12}\), and an internal node value at the barycenter of the triangle. With this additional information the basis functions are given by

\[
\begin{align*}
    b_7 &= b_1^2 b_2 - b_1 b_2^2, \\
    b_8 &= b_2^2 b_3 - b_2 b_3^2, \\
    b_9 &= b_3^2 b_1 - b_3 b_1^2, \\
    b_{10} &= b_1 b_2 b_3.
\end{align*}
\]

In the early papers, authors showed superior properties of the \(p\)-refinement over the \(h\)-refinement. However, increased computing power and principle difficulties to increase the order of approximation arbitrarily, helped the \(h\)-refinement or AMR methods to become dominant. \(p\)-refinement is still a viable and very promising option in the \(hp\)-refinement methods, where one attempts to balance either mesh refinement or increase of order by appropriate error analysis techniques \[16, 303, 321, 358\]. A combination of \(p\)-refinement in a moving mesh method and additional \(h\)-refinement is given in \[251\]. A comparison of \(h\) and \(p\) refinement for solutions of the shallow water equations has been conducted in \[405\].
2.6 Refinement Criteria

Refinement criteria are crucial to the whole process of adaptive modeling. The adaptive model can only be as good as the adaptation criterion. If the criterion over-estimates the error, in other words if it is too selective, then it detects too large refinement areas, and the method cannot be efficient. On the other hand, if the criterion under-estimates the error, the method will be inaccurate. So, the quest is to find a criterion that detects the refinement area as accurately as possible in order to maximize efficiency and accuracy of the adaptive method.

There are three basic types of refinement criteria:

1. Heuristic error proxies;
2. Physics based criteria;
3. Mathematical (discretization) error estimators.

Mathematical error estimators are often based on the residual of a given operator. These estimators developed along with the mathematical adaptivity paradigm of equilibrating the discretization error (see sect. 2.1). Physics based criteria are more common on the application side of adaptive modeling. They represent the perspective of resolution enhancement by adaptive refinement. Error proxies often bridge the gap between these two, since they represent a heuristic way to estimate the error by means of easily accessible (physical or functional) values. This section shall present several examples of the above types.

2.6.1 Error Proxies

Error proxies are criteria that are derived from easily available data. For example, it is known that steep orography gradients create complex flow patterns. Since the orography is available \textit{a priori}, these gradients can be easily computed and be used to refine locally.

In tracer advection the gradient of the advected constituent can be used as a refinement criterion. One can argue that especially for semi-Lagrangian methods, the accuracy of interpolation drops where the gradient is steep. Therefore a gradient-based criterion can be used successfully for semi-Lagrangian advection (see sect. 8.1.2, and [34, 39]). For purely advection modeling, Kessler showed that a gradient based criterion results in even better approximation quality than a truncation error estimate [240, 393].

In general, proxy criteria often interleave with physics based criteria. A gradient or a curvature of a constituent is only a proxy, since not the gradient of the constituent causes the real problem but the low approximation quality of a numerical scheme. Karni and coworkers use a smoothness indicator for adaptive refinement control in a solver for hyperbolic systems [230].
In multi-constituent chemical transport modeling, a proxy based refinement, where the criterion is composed of several individual proxies, is indispensable. Tomlin et al. use such a weighted sum of proxy refinement indicators in [394]. Belwal et al. demonstrate a similar principle in [47].

2.6.2 Physics Based Criteria

Many physics based criteria have been developed in the context of cyclone tracking in nested or adapting hurricane modeling. For example the 850 hPa relative vorticity extreme values have been used to track cyclones. Successful approaches applied geopotential or temperature extremes as well as sea level pressure (see [221] for a review of methods). Läuter uses the direct values of vorticity and divergence to derive an efficient and accurate refinement criterion. More precisely, the refinement criterion \( \eta_\tau \) in each cell \( \tau \) of the triangulation is computed from vorticity \( \zeta \) and divergence \( \delta \) by

\[
\eta_\tau = \left( \int_\tau \zeta^2 + \delta^2 \, dx \right)^{1/2} = \| \zeta + \delta \|_{L^2(\tau)}.
\]

The grid is refined, wherever \( \eta_\tau \) is above a user defined threshold, that is scaled with an absolute error factor derived from the previous time step [256]. Different strategies for marking elements for refinement have been introduced sect. 2.4.

Jablonowski assesses three different physical adaptation criteria, the (relative) vorticity \( \zeta_r \), the gradient of the geopotential height field \( \nabla \Phi \) and the curvature of the geopotential height, in other words the Laplacian \( \Delta \Phi \) [221]. Her conclusions result in an error criterion based on either \( \nabla \Phi \) or \( \zeta_r \), as in Läuter. However, she reports on alternating refinement/de-refinement behavior in her simulations as well as mislead refinement by high frequency oscillations in the solution due to numerical (instability) effects.

2.6.3 Mathematical Error Estimation – Basic Principles

Most mathematical error estimators rely on the residual of the differential operator with a discrete solution. We want to briefly outline the principle on an abstract example equation:

\[
\mathcal{D}(\rho) = f \text{ in } \mathcal{G} \subset \mathbb{R}^d.
\]  

(2.7)

\( \mathcal{D} \) represents a given linear (differential) operator and \( f \) a right hand side; we assume suitably defined boundary and initial conditions on \( \rho \). Let \( \rho_h \) be a numerically computed solution to the discrete analogue to (2.7). Then the residual is defined by

\[
R_h = R_h(\rho_h) = \mathcal{D}(\rho_h) - f \text{ in } \mathcal{G}.
\]

(2.8)
2.6 Refinement Criteria

If we subtract the residual from both sides in (2.7) we obtain

\[ D(\rho - \rho_h) = f - R_h(\rho_h) \Rightarrow \varepsilon = D^{-1}(f - R_h(\rho)), \tag{2.9} \]

where \( \varepsilon = \|\rho - \rho_h\| \) denotes the (discretization) error. One can easily see that the solution of (2.9) is at least as computationally expensive as the solution of the original problem (2.7). The trick is to find a method that solves (2.9) approximately and/or locally with much less effort than the original problem. \([\varepsilon]\) shall denote this approximate error estimator.

Another commonly used error estimation technique does not rely on the residual, but on the computed solutions themselves. Let \( \rho_h \) be the discrete solution of (2.7), while \( \rho_H \) is another solution obtained with a higher order scheme (if \( h \) and \( H \) represent the mesh size, we would required \( h > H \)). The Richardson extrapolation criterion defines \([\varepsilon]\) by

\[ [\varepsilon] = \tilde{C} \|\rho_h - \rho_H\|, \tag{2.10} \]

with \( \tilde{C} \) a constant, correcting over- or under-estimation. In fact for \( H \rightarrow 0 \) we have that \([\varepsilon] \rightarrow \varepsilon\), provided that the method to compute \( \rho_H \) converges.

For an adaptive refinement control, we do not need to know the global error, but a local (element-wise) error. This can be achieved by translating the above idea to individual cells \( \tau \). Furthermore, we can define local area based residuals by

\[ R_\tau(\rho_h) = D(\rho_h) - f \text{ in } \tau, \]

and edge based (in 2D) or face based (in 3D) jumps, defined by

\[ R_e(\rho_h) = \left| \frac{\partial \rho_h}{\partial n_e} \bigg|_{\tau_1} - \frac{\partial \rho_h}{\partial n_e} \bigg|_{\tau_2} \right|, \]

where \( \tau_1 \) and \( \tau_2 \) are the two cells interfacing at edge/face \( e \), and \( n_e \) is the corresponding outer normal direction at edge \( e \).

For finite element-like computations (i.e. for finite elements, as well as spectral elements and polynomial basis finite volume methods) we can distinguish five different basic approaches to derive a local a posteriori mathematical error estimator \([\varepsilon]_\tau\) (see also [71, 400, 401]).

1. **Residual estimators**: the local error is approximated by \([\varepsilon]_\tau\) by means of \( R_\tau \) and \( R_e \), defined above. This approach is due to Babuška and Rheinboldt [18]. Hugger has reviewed this approach [210] and Thomas and Sonar propose a residual estimator for nonlinear hyperbolic conservation laws [389].

2. **Estimators based on local Dirichlet problems**: For every element \( \tau \) we solve a local problem of the form

\[ D(\chi) = f \text{ in } \tau, \quad \chi = \rho_h \text{ on } \partial \tau \]

where \( \tau \) is a slightly extended domain surrounding \( \tau \). The approximation order of this local problem is higher than the approximation order of \( \rho_h \).
The estimator $[\varepsilon]_\tau$ is then derived from $\|\chi - \rho_h\|_{L^1(\tau)}$. Again this approach has been introduced by Babuška and Rheinboldt [17].

3. **Estimators based on local Neumann problems**: In this case a local problem of the form

$$D(\chi) = R_\tau(\rho_h) \text{ in } \tau, \quad \frac{\partial \chi}{\partial n} = R_e(\rho_h) \text{ on } e \in \partial\tau,$$

is solved by an approximation order higher than the order of $\rho_h$. The error estimator is based on the energy norm $\|\chi\|_\tau$. These error estimators were originally proposed by Bank and Weiser [22].

4. **Estimators based on averaging**: Let us describe this method for Poisson’s equation, i.e. $D = -\Delta$. The method tries to construct an approximation $\sigma^{(2)}$ to the true $\Delta \rho$, by applying the following scheme for the gradient operator twice. Let $\sigma$ be the weighted average of the gradients corresponding to one node patch:

$$\sigma = \sum_{\{i : \tau_i \in \text{patch of node } i\}} |\tau_i| \cdot \nabla \rho_h|_{\tau_i}.$$

As usual, $|\tau|$ denotes the area of $\tau$. Extend $\sigma$ to the whole element $\tau$ by linear interpolation. The error estimator is derived from $|\sigma^{(2)} - \Delta \rho_h|$. This approach can be found in [428]. Additionally, Carstensen shows that averaging methods are reliable and efficient [78].

5. **Hierarchical error estimators**: These estimators use an expanded finite element space and take the difference of both solutions by applying a strengthened Cauchy inequality. For the details see [122].

In atmospheric modeling the second approach can be found in [36]. An application of a variant of the averaging technique has been used in a mesh-less modeling technique in [41]. Furthermore, Skamarock derived a truncation error estimate following the lines of a Richardson extrapolation as outlined above [368]. Further reading in error estimation can include work by Bank and Xu [23, 24]. Recently, error estimation techniques based on computing error bounds have been proposed [86, 87, 272]. For hyperbolic problems, the error in an element is not only influenced by the local residual but also by the residual in the cells in the domain of influence. This dependency has been studied by Houston and Suli [206]. A proposal for efficient error estimates for computational fluid dynamics, based on hydrodynamic stability can be found in [226]. For biodegradation transport schemes Klöfkorn and coworkers propose an a posteriori error estimate that is suited for advection dominated problems [243].

We can conclude that there is still a large potential for improvement in finding good refinement criteria for atmospheric multi-component modeling. There is hardly any literature on combinations of error and refinement criteria in the presence of multiple refinement objectives. And a theoretically sound treatment of sub-grid processes in the error estimation is not available.
Grid Generation

We start the more formal description of adaptive atmospheric modeling methods with an introduction to mesh generation. In contrast to non-adaptive methods, where a fixed grid is given, in adaptive methods, the grid generating parts of a modeling software play a fundamental role. While in fixed grid applications, the software design is often derived from a computational stencil, in adaptive methods, the grid management forms the underlying basis for other derived data structures. We will discuss efficient data structures later in chap. 4 and will concentrate only on grid generation issues here.

Two different tasks characterize the art of mesh generation: Automatic generation of a (coarse) mesh for a given – in most cases polygonal – domain; and efficient refinement and un-refinement of a given initial mesh. Automatic mesh generation is still an open field of research, since up to now, there is no fully automatic meshing tool available that can cover all demands. However, some very impressive results have been gained and the reader is referred to the literature for an overview [229, 289, 391]. Since we are mostly concerned with spherical geometries without boundaries, we will focus on the second part, mesh refinement and un-refinement strategies, in this chapter. Section 3.5 reviews several initial grids for a special domain: the sphere.

3.1 Notation

In order to describe meshes formally correct, we need to introduce some notation. Let us begin with the basic definition of a triangulation. Note that by triangulation we do not necessarily mean a triangular mesh. So let us define a cell first, were we want to assume that cells in a triangulation are k-simplices.

**Definition 3.1.1 (k-simplex)**

Let $\mathcal{P} = \{P_1, \ldots, P_k\} \subset \mathbb{R}^d$ be a (linearly independent) set of points (vertices) in the d-dimensional space. Then the convex hull of $\mathcal{P}$ is a k-simplex, where the convex hull is the intersection of all convex sets in $\mathbb{R}^d$ that contain $\mathcal{P}$. 
Example 3.1.2 (Simplicial cell types)

- A triangle in $\mathbb{R}^2$ is a 3-simplex.
- A hexahedral cell in $\mathbb{R}^3$ is a 8-simplex.
- A tetrahedral cell in $\mathbb{R}^3$ is a 4-simplex.

Definition 3.1.3 (Triangulation)

Let $\mathcal{G} \subset \mathbb{R}^d$ be the finite d-dimensional and polygonal computational domain. Then $T = T(\mathcal{G}) = \{\tau_1, \ldots, \tau_M\}$, $M$ number of cells, is an admissible triangulation, if

1. the cells $\tau_i$, $i = 1 : M$, are open disjoint $k$-simplices in $\mathcal{G}$, which cover the whole domain:

   $\tau_i \cap \tau_j = \emptyset$ for $i \neq j$; $\bigcup_{i=1:M} \tau_i = \mathcal{G}$.

2. for $i \neq j$ is $\tau_i \cap \tau_j$ either empty, or a common $l$-simplex with $l < k$ (a common edge, face, etc.).

We used the following notation:

- $\tau$ denotes the interior of a $k$-simplex, and
- $\overline{\tau}$ is the the closed $k$-simplex, i.e. its interior together with its boundary.

Admissible triangulations are often called conforming triangulations.

Remark 3.1.4 When considering triangular cells in $\mathbb{R}^2$, definition 3.1.3 is equivalent to the definition given in many graph-based textbooks (see [111]): Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of points in the plane ($\mathbb{R}^2$). A maximal planar subdivision $\mathcal{S}$ is defined as a subdivision in which no edge can be added without destroying its planarity, i.e. any edge that is not in $\mathcal{S}$ intersects an edge in $\mathcal{S}$. An admissible triangulation is defined as a maximal planar subdivision with vertex set $\mathcal{P}$.

Remark 3.1.5 Note that definition 3.1.3 is cell based. Therefore, when dealing with node based methods, e.g. finite difference methods, only vertices of a triangulation are considered.

Remark 3.1.6 (Hanging node)

The above definition 3.1.3 defines an admissible triangulation, that is one without hanging nodes. A hanging node is a node on an unrefined edge (see fig. 3.1). Since many grid types incorporate hanging nodes we do not necessarily require admissible triangulations. A non-admissible triangulation is given by definition 3.1.3, omitting the second requirement. A non-admissible triangulation is often called non-conforming.

In what follows, we will mostly omit the argument $\mathcal{G}$ for simplicity, since in most cases it will be clear or irrelevant how specifically $\mathcal{G}$ is defined. In order to characterize a triangulation we need a measure for the quality and an expression for hierarchies of triangulations.
Definition 3.1.7 (Hierarchies of triangulations)
Let \( T_1 = \{ \tau_1, \ldots, \tau_M \} \) and \( T_2 = \{ \tau_1, \ldots, \tau_K \} \) two triangulations. If \( T_2 \) is created from \( T_1 \) by refinement, we write \( T_1 \prec T_2 \).

Definition 3.1.8 (Mesh width, inner angle and regularity)
Let the following parameters be defined for a cell \( \tau \) in a triangulation \( T \).
\[
\begin{align*}
h_{e,\tau} & : \text{length of longest edge in } \tau, \\
h_{B,\tau} & : \text{radius of circum-circle of } \tau, \\
h_{b,\tau} & : \text{radius of inner circle of } \tau.
\end{align*}
\]
We define the following parameters for characterizing the mesh size of \( T \):
\[
\begin{align*}
h_e &= \min_{\tau \in T} \{ h_{e,\tau} \}, \\
h_B &= \min_{\tau \in T} \{ h_{B,\tau} \}, \\
h_b &= \min_{\tau \in T} \{ h_{b,\tau} \}.
\end{align*}
\]
Let furthermore \( \vartheta_\tau \) be the smallest angle in \( \tau \). The smallest inner angle of \( T \) is given by
\[
\vartheta = \min_{\tau \in T} \{ \vartheta_\tau \}.
\]
Note that for \( \mathbb{R}^d \) with \( d > 2 \) the inner angle definition has to be replaced by an inner cone. Finally, \( T \) is called regular, if \( 0 \ll \vartheta \ll \pi \).

Note that since
\[
h_{b,\tau} < h_{e,\tau} < h_{B,\tau} < C_\vartheta h_{b,\tau},
\]
where \( C_\vartheta \) is a constant that depends only on the smallest inner angle, all mesh width parameters are equivalent to each other. Therefore, we often just use the notation \( h \) for mesh width, without specifying which specific measure is to be used.

Remark 3.1.9 Since the sum of all inner angles of a triangle is \( \pi \), for triangulations with triangular elements, the definition of regular in definition 3.1.8 implies that \( \vartheta \) is close to \( \frac{\pi}{3} \). For quadrilateral triangulations a similar remark holds with \( \vartheta \) close to \( \frac{\pi}{2} \).

Note that the regularity condition is equivalent to \( 0 \ll \frac{h_b}{h_B} = C_\vartheta^{-1} < 1 \).
3.2 Grid Types

In this section different grid types are classified. A first attempt to classify grid types can be found for example in [88, 204]. These classifications have been developed mainly for automatic mesh generation methods in engineering. However, since we are concerned with adaptively refined grids, we propose a distinct approach here.

The first criterion of distinction between grids is the admissibility condition. Admissible (or conforming) grids do not allow hanging nodes. Both quadrilateral and triangular grids can have hanging nodes, when refined locally, depending on the refinement strategy.

A second important distinction between grids is their orthogonality. Quadrilateral grids have orthogonality properties, making them suitable for using 1D basis functions with tensor product expansion to $d$ dimensions. When considering higher dimensional triangulations, also mixed grid types can occur, where the horizontal grid is (non-orthogonal) triangular while the vertical dimension is discretized by an orthogonal quadrilateral extension, to give an example.

An important distinction is nesting. For locally refined meshes non-nested grids are created by defining new nodes and then completely re-meshing the triangulation. With nested grids, local refinement of single mesh elements with a certain refinement strategy is performed. Nested meshes allow for (more or less) straightforward implementation of multi-grid methods. Note that definition 3.1.7 is more general, since it does not detail the refinement strategy.

A fourth category of distinction concerns structure, which we formally define.

**Definition 3.2.1 (Structured and unstructured mesh)**

A mesh is called **structured**, if the nodes can be ordered by an (incrementing) index array of $d$ dimensions $(i_1, i_2, \ldots, i_d)$ and if neighboring nodes can be accessed by incrementing/decrementing the index, e.g. $(i_1, i_2 + 1, \ldots, i_d)$. Otherwise a mesh is called **unstructured**.

Note that for an unstructured mesh additional data is required to define neighborhood relations (this is the so called connectivity). In adaptive grid generation, even logically structured meshes are sometimes treated like unstructured meshes, because then no different approaches have to be used in uniform (structured) and locally refined (unstructured) areas.

For a short and precise descriptions of meshes we introduce a naming convention for grid types given in table 3.1. The first four positions of the name-code correspond to the above classification, while the fifth position corresponds to a name for the geometry of grid cells. In case a grid does not have one or the other of the tabulated characteristic properties, the naming code is omitted. With this naming convention, a locally refined block-structured quadrilateral grid as used e.g. in [221] could be called a ONSH-grid, while a
Table 3.1. Naming convention for adaptive grids

<table>
<thead>
<tr>
<th>Position</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>C: conforming</td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>O: orthogonal</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td>N: nested</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>S: structured</td>
</tr>
<tr>
<td>5</td>
<td>T/H/P</td>
<td>T: triangular/tetrahedral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H: quadrilateral/hexahedral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P: polygonal/polyhedral</td>
</tr>
</tbody>
</table>

A bisected triangular grid like in amatos would be called CNT-grid. A Delaunay triangulation is of type CT.

In order to further characterize a mesh, we can look at the number of refinements as an indicator of resolution ratio. Most gridding schemes start with a given initial mesh, which is uniformly refined up to a given level of refinement and then further refined locally. For triangular meshes derived from an icosahedral initial discretization of the sphere, Baumgardner and Frederickson and more recently Giraldo have introduced a classification of uniform triangular refinements [167, 30]. For a more universal convention and assuming that we have nested grids, we propose a pair of numbers \([c : f]\) where \(c\) indicates the number of uniform refinements, i.e. \(c\) represents the coarsest mesh level used in computations, and \(f\) is the finest grid level.

Extending the above naming convention with this pair, we have a ONSH\([3 : 6]\)-grid in \([221]\), while in \([39]\) the finest mesh is of type CNT\([4 : 19]\). If we have uniform grids, then we can omit one of the entries, obtaining CNT\([15]\) \(\equiv\) CNT\([15 : 15]\).

If refinement levels are inadequate or not available for the classification, we could consider resolutions of the coarsest grid cells and the finest grid cells. Thus, CNT\([4 : 19]\) \(\equiv\) CNT\((230 : 5)\)\([\text{km}]\). For distinction, we have put the resolution in brackets instead of square brackets and added a subscript denoting the unit (Kilometers here). As another example, the operational global model (GME) of the German Weather Service (DWD) has type CSH\((40)\)\([\text{km}]\).

3.3 Refinement Strategies in 2D

In this section we give an overview of commonly used refinement strategies. Selection of a strategy depends on the application and the objectives of meshing. One could – for example – either optimize grid quality, but faces algorithmic complexity. Or it could be important to achieve orthogonality, but then admissibility cannot be maintained. There is no single strategy that optimizes all of the existing desirable properties of a mesh. For a good overview of mesh generation especially in engineering applications, see [391]. A comparison of refinement strategies in the context of finite element methods is given in [290].
Triangular Bisection

A very simple, yet powerful refinement strategy for triangular meshes is the bisection strategy. It constructs two daughter triangles from one mother. In order to maintain regularity of the mesh, only marked edges are allowed to be refined. Defined in an iterative procedure, this strategy leads to locally adaptive, nested, and admissible meshes.

Algorithm 3.3.1 (Bisection of marked edge)

Let \( \tau \) be a triangle, defined by its vertices \( v_1, v_2, v_3 \). Without loss of generality let \( e_m = \{v_1, v_2\} \) be a marked edge.

1. Insert new vertex \( v_n \) at center of \( e_m \), i.e. \( v_n = (v_1 + v_2)/2 \).
2. Define new cells \( \tau_1 \) and \( \tau_2 \) by:
   \[
   \tau_1 = \{v_1, v_n, v_3\}, \quad \tau_2 = \{v_n, v_2, v_3\}.
   \]
3. Mark unrefined edges of \( \tau \) in elements \( \tau_i \), i.e. \( e_{\tau_1} = \{v_1, v_3\} \), and \( e_{\tau_2} = \{v_2, v_3\} \).

This algorithm is depicted in fig. 3.2. Note that the order of vertices is important to maintain the orientation of triangles. The above definition maintains orientation in the daughter elements.

Remark 3.3.2 (Bisection of longest edge)

There is a variant of algorithm 3.3.1, called bisection of longest edge. It differs from the above given algorithm in that it refines the longest edge only. When starting with a sufficiently regular initial triangle, both algorithms are equivalent. However, marking an edge with the above scheme results in less computational effort, because longest edge bisection needs to evaluate the edge length for each of the three edges in each step.

In order to create a complete mesh with regions of local refinement we have to use the bisection algorithm iteratively:

Algorithm 3.3.3 (Admissible bisection triangulation)

Let \( \mathcal{T} = \{\tau_1, \ldots, \tau_M\} \) be a given admissible coarse triangulation, and let \( S \subset \mathcal{T} \) be the set of cells marked for refinement.

\[
\text{WHILE } (S \neq \emptyset)
\]
3.3 Refinement Strategies in 2D

Algorithm 3.3.3 is visualized in fig. 3.3. It can be shown that algorithm 3.3.3 converges, i.e. $S$ is in fact empty after a finite number of iterations. Furthermore, it leads to an admissible triangulation and – except for single pathological cases – local refinement has no global effect on the mesh (see [25, 196, 341]).

This algorithm is simple from a coding point of view. It leads to admissible triangulations and maintains regularity. In fact, the inner angles are bounded by $\frac{1}{2} \vartheta_0$, where $\vartheta_0$ is the smallest inner angle of the initial mesh. Triangulations of different levels are nested. Each level decreases the mesh size by a factor of $\sqrt{2}$. An example of application of this algorithm can be found in [39].

Definition 3.3.4 (Admissible patch)

An admissible patch is a grid part consisting of four triangles grouped around a refined edge (see fig. 3.4).

Remark 3.3.5 (Coarsening bisected triangulations)

In order to coarsen a bisected triangulation, we look for admissible patches. If at least three elements of an admissible patch are flagged for coarsening, then all four elements are deleted, obtaining a new mesh section with only the two mother triangles.
Regular Triangular Refinement

Regular refinement of triangles divides each edge of a given initial triangle:

**Algorithm 3.3.6 (Regular triangular refinement)**

Given a triangle $\tau$, defined by its vertices $v_1, v_2, v_3$ with edges $e_1 = \{v_2, v_3\}$, $e_2 = \{v_1, v_3\}$, $e_3 = \{v_1, v_2\}$.

Insert new vertices $v_{n,i}$ at center of $e_i$, e.g. $v_{n,1} = (v_2 + v_3)/2$.

Define new cells $\tau_1, \ldots, \tau_4$ by:

- $\tau_1 = \{v_1, v_{n,3}, v_{n,2}\}$
- $\tau_2 = \{v_{n,3}, v_2, v_{n,1}\}$
- $\tau_3 = \{v_{n,2}, v_{n,1}, v_3\}$
- $\tau_4 = \{v_{n,1}, v_{n,2}, v_{n,3}\}$

The algorithm is depicted in fig. 3.5. This refinement strategy leads to non-admissible meshes. Therefore, if admissibility is required, we have to discuss methods to achieve this goal. As in the previous section, the sequence of vertices given in the algorithm above preserves orientation.

Most authors introduce a *green refinement* for admissibility. This is a temporary bisection refinement of those cells that have one hanging node. With green refinement the hanging node and its opposite vertex are connected to build an edge, dividing the cell into two daughter cells (see figure 3.6). We now formulate an algorithm that is commonly used to create admissible triangulations (see [34] for an example).

**Algorithm 3.3.7 (Admissible regularly refined triangulation)**

Let $T = \{\tau_1, \ldots, \tau_M\}$ be a given admissible coarse triangulation, and let $S \subset T$ be the set of cells marked for regular refinement.

1. Remove all green closures from previous refinement step.
2. WHILE ($S \neq \emptyset$)
a) FOR EACH ($\tau \in S$): refine $\tau$ according to algorithm 3.3.6, and obtain a new triangulation $T^*$, remove $\tau$ from $S$;
b) FOR EACH ($\tau \in T^*$): IF $\tau$ has more than one hanging node, THEN: 
$S = S \cup \{\tau\}$. 
END WHILE.
3. FOR EACH ($\tau \in T^*$): IF $\tau$ has hanging node, THEN: refine $\tau$ by green refinement.

Remark 3.3.8 As in subsect. 3.3, it can be shown that algorithm 3.3.7 leads to an admissible triangulation. The inner angles are bounded, since regular refinement creates four similar daughter triangles form each mother triangle.

Remark 3.3.9 Some authors introduce a blue refinement to treat cells with two hanging nodes. In order to obtain a blue refinement, proceed as follows (see fig. 3.6):

1. bisect the cell by taking the hanging node on the longer edge and its opposite vertex as bisection edge;
2. bisect the daughter element of step 1, which contains the remaining hanging node by connecting this hanging node with the first hanging node.

Remark 3.3.10 (Coarsening regularly refined triangulation)
Since green or blue refinements are removed from the grid before refinement/coarsening, we only have to consider regularly refined elements for coarsening. So, in order to coarsen a regularly refined triangular element, the four daughter elements are removed, leaving the mother on the now finest mesh level. Coarsening is performed if three or more daughters are flagged for coarsening.

Quadrilateral Refinement

Quadrilateral grids are very popular since several advantages can be exploited for efficient numerical calculations. Quadrilaterals are well suited for efficient interpolation/quadrature schemes, because the local coordinate system can be formulated orthogonally. A structured data layout can be achieved easily. And the ratio of cell number to node number (which is a measure for computational efficiency) is smaller than in triangular meshes. On the other hand, admissibility is often hurt. And more importantly, complex geometries are not easily represented by quadrilaterals. As a simplification and requirement for orthogonality, we want to assume convex quadrilaterals throughout this section.

The most simple refinement strategy, is to insert one vertex and add edges to the centers of all cell edges:

Algorithm 3.3.11 (General quadrilateral refinement)
Given a Quad $\chi$, defined by its vertices $v_1, \ldots, v_4$ with edges $e_1 = \{v_1, v_2\}$,
$\ldots$, $e_4 = \{v_4, v_1\}$. 
Algorithm 3.3.11 for general quadrilateral refinement (a) and 3.3.12 for regular quadrilateral refinement acting on a cell

1. Insert new vertex $v_n$ inside $\chi$.
2. Create new edge vertices $v_{n,1}, \ldots, v_{n,4}$ by
   $$v_{n,1} = \frac{(v_1 + v_2)}{2}, \ldots, v_{n,4} = \frac{(v_4 + v_1)}{2}.$$
3. Define new cells $\chi_1, \ldots, \chi_4$ by:
   $$\chi_1 = \{v_1, v_{n,1}, v_n, v_{n,4}\}, \quad \chi_2 = \{v_{n,1}, v_2, v_{n,2}, v_n\}$$
   $$\chi_3 = \{v_n, v_{n,2}, v_3, v_{n,3}\}, \quad \chi_4 = \{v_{n,4}, v_n, v_{n,3}, v_4\}$$

**Remark 3.3.12** (Regular quadrilateral refinement)
A less general quadrilateral refinement is given by a variant of algorithm 3.3.11, where we omit step 1 and define the newly inserted vertex as the intersection of the new edges $e_{n,1} = \{v_{n,1}, v_{n,3}\}$ and $e_{n,2} = \{v_{n,2}, v_{n,4}\}$. Both variants are depicted in fig. 3.7.

**Remark 3.3.13** (Closure for general quadrilateral refinement)
Quadrilateral refinement leads to hanging nodes in the triangulation so we have to deal with closures. A common closure uses triangular cells to remove hanging nodes. In this method we have to differentiate five different cases (see fig. 3.8 for the first four cases):

1. **One hanging node:** connect hanging node with both vertices of opposite edge, obtaining 3 new triangular cells;
2. **two hanging nodes on opposite edges:** connect both hanging nodes, obtaining 2 new quadrilateral cells;
3. **two hanging nodes on adjacent edges:** insert new vertex at cell center, connect new vertex with hanging nodes and opposite cell vertex, obtaining 3 new quadrilateral cells;
3.3 Refinement Strategies in 2D

4. **three hanging nodes**: insert new vertex at cell center, connect new vertex nodes with all hanging, and with vertices of unrefined edge, obtaining 2 new quadrilateral and 3 new triangular cells;

5. **four hanging nodes**: refine regularly.

Most of the current quadrilateral grid management packages treat no single cells but patches of cells. Thus, let the patch size be $\nu \times \nu$, then refinement consists of creating a new $\nu \times \nu$-patch in each cell marked for refinement.

**Algorithm 3.3.14 (Quadrilateral $3 \times 3$-patch refinement)**

Let $v_1, \ldots, v_4$ be the vertices of a quadrilateral cell $\chi$ in counter-clockwise order starting from the bottom left vertex. Then denote by index $i$ the local $x$-direction and by index $j$ the local $y$-direction correspondingly. We want to denote the new vertices by $v_{i,j}$. Note that for a $3 \times 3$-patch $i,j = 0 : 3$, with $v_1$ corresponding to index pair $(0,0)$, i.e. $v_1 = v_{0,0}$, $v_2 = v_{3,0}$, $v_3 = v_{3,3}$, $v_4 = v_{0,3}$.

1. **Insert new vertices** $v_{i,j}$, $i,j = 0 : 3$ using the following formula:

$$
\begin{align*}
    v_{i,j}(x) &= \frac{1}{9} \left( (3-j) [(3-i)v_1(x) + iv_2(x)] + j [(3-i)v_4(x) + iv_3(x)] \right); \\
    v_{i,j}(y) &= \frac{1}{9} \left( (3-i) [(3-j)v_1(y) + jv_4(y)] + i [(3-j)v_2(y) + jv_3(y)] \right);
\end{align*}
$$

where $v_{i,j}(x)$ denotes the $x$-component of $v_{i,j}$, and $v_{i,j}(y)$ the $y$-component respectively.

2. **Now, the nine new cells of the refined $3 \times 3$-patch are given by** $\{v_{i,j}, v_{i+1,j}, v_{i+1,j+1}, v_{i,j+1}\}$ with $i,j = 0 : 2$.

Figure 3.9 shows the action of algorithm 3.3.14.

---

**Fig. 3.8.** Closures for avoiding hanging nodes in quadrilateral grid refinement. Four cases are shown, as described in the text.
Remark 3.3.15 (Closures for quadrilateral $3 \times 3$-patch refinement)

Again, in order to find admissible triangulations, a closure has to be found. In the $3 \times 3$-patch case, four situations are considered (see fig. 3.10 for the first two cases):

1. **One edge has hanging nodes:** Without loss of generality, let $v_{1,0}$, and $v_{2,0}$ be the hanging nodes. Then insert $v_{1,1}$, and $v_{2,1}$. The four closing quadrilateral sub-cells are given by

$$\{v_{0,0}, v_{1,0}, v_{1,1}, v_{0,3}\}, \quad \{v_{1,0}, v_{2,0}, v_{2,1}, v_{1,1}\},$$

$$\{v_{2,0}, v_{3,0}, v_{3,3}, v_{2,1}\}, \quad \{v_{1,1}, v_{2,1}, v_{3,3}, v_{0,3}\}.$$

2. **Two adjacent edges have hanging nodes:** Without loss of generality, let $v_{1,0}$, $v_{2,0}$, $v_{0,1}$, and $v_{0,2}$ be the hanging nodes. Then insert $v_{1,1}$ and $v_{2,2}$. The five closing quadrilateral cells are given by

$$\{v_{0,0}, v_{1,0}, v_{1,1}, v_{0,1}\},$$

$$\{v_{1,0}, v_{2,0}, v_{2,2}, v_{1,1}\}, \quad \{v_{2,0}, v_{3,0}, v_{3,3}, v_{2,2}\},$$

$$\{v_{0,1}, v_{1,1}, v_{2,2}, v_{0,2}\}, \quad \{v_{0,2}, v_{2,2}, v_{3,3}, v_{0,3}\}.$$

3. **Two opposite edges have hanging nodes:** In this case, just connect the corresponding opposing hanging nodes, obtaining three closing quadrilaterals.

4. **Three or four edges have hanging nodes:** Refine regularly.
Remark 3.3.16 For uneven numbers of sub-cells in each patch cell, a convenient closure method can be derived that avoids triangular closures. We gave an example for $3 \times 3$-patches in algorithm 3.3.15 that can easily be generalized to higher uneven numbers. It should be mentioned, however, that for high numbers $\nu = 2k + 1$, $k \in \mathbb{N}$ very thin quadrilaterals can occur which might degrade numerical stability. See fig. 3.11 for an example with a $5 \times 5$-patch.

Finally, in order to obtain an algorithm that creates an admissible mesh, the refinement and closure strategies have to be combined. This algorithm has been used in [154].

Algorithm 3.3.17 (Admissible quadrilateral triangulation)

Let $Q = \{\chi_1, \ldots, \chi_M\}$ be a given admissible coarse quadrilateral triangulation, and let $S \subset Q$ be the set of cells marked for regular refinement.

1. Remove all closures from previous refinement step.
2. WHILE ($S \neq \emptyset$)
   a) FOR EACH ($\chi \in S$): refine $\chi$ according to algorithm 3.3.11 or 3.3.14, and obtain a new triangulation $Q^*$, remove $\chi$ from $S$;
   b) FOR EACH ($\chi \in Q^*$): IF $\chi$ has more than 2 resp. 3 edges with hanging nodes, THEN: $S = S \cup \{\chi\}$.
END WHILE.
3. FOR EACH ($\chi \in Q^*$): IF $\chi$ has hanging nodes, THEN: refine $\chi$ using remarks 3.3.13 and 3.3.15, resp.

Remark 3.3.18 Again, it can be shown that algorithm 3.3.17 leads to an admissible quadrilateral triangulation. Inner angles in quadrilateral triangulations are obviously bounded, provided that the initial triangulation is well behaved.

Delaunay Triangulation – Non-Nested Refinement

For the definition of a Delaunay triangulation, we start with the definition for the Voronoi Diagram of a point set $P = \{v_1, \ldots, v_n\}$. As in the previous sections, we will denote the points by $v_i$, since we also call them vertices.
Definition 3.3.19 (Voronoi Diagram)

Let $\mathcal{P} = \{v_1, \ldots, v_n\} \subset \mathbb{R}^2$ be a given set of points in the plane. Then the Voronoi Diagram of $\mathcal{P}$ is the subdivision of the plane into $n$ sub-domains $V_i$ corresponding to the $n$ vertices $v_i$, $i = 1 : n$, such that

$$V_i = \{x \in \mathbb{R}^2 : \text{dist}(x, v_i) \leq \text{dist}(x, v_j), \ j \neq i\},$$

where $\text{dist}(x, y)$ is the Euclidean distance between $x$ and $y$. The sub-domain $V_i$ is called the Voronoi cell corresponding to vertex $v_i$.

Now, we proceed with the Delaunay Graph of a point set $\mathcal{P}$ (see fig. 3.12 for a Voronoi diagram together with its Delaunay graph).

Definition 3.3.20 (Delaunay Graph)

As above, let $\mathcal{P}$ be a set of points/vertices, and let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be the Voronoi diagram. Then the Delaunay graph is defined by the graph nodes $\{v_1, \ldots, v_n\}$ and connecting arcs for each pair $(v_i, v_j)$, $i \neq j$, wherever the Voronoi cells $V_i$ and $V_j$ share an edge.

Finally, connecting all graph-connected vertices by straight edges, induces the Delaunay Triangulation (see fig. 3.13). Neglecting some pathological cases, it can be proven that in deed, this procedure yields an admissible triangulation in the sense of defining remark 3.1.4. Furthermore, it can be shown (see e.g. [111]) that the Delaunay triangulation of a point set $\mathcal{P}$ is maximizing the inner angles of the Triangulation. Or in other words, all interior angles are bounded away from $0^\circ$ and $90^\circ$, resp.

The Delaunay triangulation of a point set $\mathcal{P}$ can be computed in $O(n \log n)$ operations with $O(n)$ required storage positions, where $n$ is the number of vertices in $\mathcal{P}$. A practical algorithm (also for 3D Delaunay triangulations) is given in [383].

In order to construct an adaptive method using a Delaunay triangulation, we need to specify a vertex insertion method. Again, we assume that there is a given initial triangulation with a set of triangles marked for refinement. In principle, one could take a random distribution of points as an initial triangulation.
3.3 Refinement Strategies in 2D

Fig. 3.13. The Delaunay triangulation (solid lines) of a given point set together with the Voronoi diagram (dotted lines)

Algorithm 3.3.21 (Unstructured Vertex insertion)

Given a triangle $\tau$, defined by its vertices $v_1, v_2, v_3$.
Insert a new vertex $v_n$ at the barycenter of $\tau$: $v_n = (v_1 + v_2 + v_3)/3$.
Define new cells $\tau_1, \ldots, \tau_3$ by:

$$
\tau_1 = \{v_1, v_n, v_2\}, \tau_2 = \{v_2, v_n, v_3\}, \tau_3 = \{v_3, v_n, v_1\}
$$

This algorithm is used in [20]. The insertion of new cells is depicted in fig. 3.14.

Remark 3.3.22 Note that algorithm 3.3.21 does not prevent inner angles to collapse. Therefore, in the algorithm for creating an complete triangulation, a re-meshing step is required (see algorithm 3.3.24, resp. remark 3.3.25).

Remark 3.3.23 Another vertex-based possibility to find locations for new nodes has been used in [41]. Here, vertices are flagged for refinement instead of cells. Then, each corner of the corresponding Voronoi-cell is inserted as new vertex. After insertion of new nodes, re-meshing has to be performed. This method is especially suited for mesh-free methods, where only point locations are considered for calculations.

The complete algorithm for obtaining an admissible locally refined triangulation from a given Delaunay triangulation follows:

Algorithm 3.3.24 (Admissible unstructured triangulation)

Let $T = \{\tau_1, \ldots, \tau_M\}$ be a given admissible coarse triangulation, and let $S \subset T$ be the set of cells marked for refinement.

Fig. 3.14. Local refinement of a Delaunay triangulation by insertion of centroid and re-meshing
Fig. 3.15. Unfolding a tetrahedron into standard position, and notations for local vertex ordering and globally marked edge (dotted line)

1. FOR EACH $(\tau \in S)$: refine $\tau$ according to algorithm 3.3.21;
2. FOR EACH $(\tau \in S)$:
   a) FOR EACH edge $e_i$, ($i = 1 : 3$):
      IF $e_i$ is not a boundary edge, then if edge flipping yields larger minimum inner angle, flip edges;
   b) remove $\tau$ from $S$.

Remark 3.3.25 Note that step 2 of the above algorithm could be exchanged by the following instruction:

2a Create Delaunay triangulation of new point set, obtained by taking all vertices after insertion.

The last algorithm was used in [166], but it is often not an efficient way of adaptive computation.

3.4 Refinement in 3D

In this section, we will extend the methods described in the previous section to three dimensional space $\mathbb{R}^3$. This will not be done in such detail, since most of the extensions presented are rather straight forward and a complete technical description would render this section somewhat tedious and lengthy. We will concentrate on the 3D-extension of the triangle bisection (algorithm 3.3.1) and the regular quadrilateral refinement (algorithm 3.3.11) together with its closure (algorithm 3.3.13).

Tetrahedral Bisection

In this section we follow an approach introduced by Bänsch [25]. We start with defining a tetrahedron with marked edges.

Definition 3.4.1 (Tetrahedron with globally marked edge)

Let $\tau^{3D}$ be a tetrahedron, defined by its four vertices $\{v_1,\ldots,v_4\} \subset \mathbb{R}^3$, with faces $\tau_1 = (v_1,v_2,v_3)$, $\tau_2 = (v_2,v_4,v_3)$, $\tau_3 = (v_1,v_3,v_4)$, and $\tau_4 = (v_1,v_4,v_2)$ (see fig. 3.15. Let each of the $\tau_i$ have one marked edge such that exactly two of the faces share their marked edge. We will call this edge a globally marked edge (of tetrahedron $\tau^{3D}$).
3.4 Refinement in 3D

Remark 3.4.2 Note that the representation in fig. 3.15 is called standard position. After rotation and scaling, this representation is unique, assuming that there is only one globally marked edge \((v_1, v_2)\).

When refining a tetrahedron, we can simply apply the 2D algorithms for refining triangles to the faces. However, we have to define, how to set the marked edge in the newly created face separating the two new tetrahedra (see fig. 3.16). Bängsch [25] uses the notion of red and black tetrahedra as given in fig. 3.17.

Algorithm 3.4.3 (Tetrahedral Bisection)

Given a tetrahedron \(\tau^{3D}\) in standard position as defined in definition 3.4.1, with a globally marked edge \(e_m = (v_1, v_2)\). We will denote the mother of \(\tau^{3D}\) by \(\mathcal{M}(\tau^{3D})\).

1. Insert new vertex \(v_n\) at center of \(e_m\).
2. Refine both faces sharing \(e_m\) according to algorithm 3.3.1 for 2D bisection.
3. Insert new face with vertices \(\{v_n, v_4, v_3\}\) and define the marked edge \(e_m^{\text{new}}\) according to the following scheme:
   a) IF \(\tau^{3D}\) is black and \(\mathcal{M}(\tau^{3D})\) is red: \(e_m^{\text{new}} = (v_3, v_4)\).
   b) IF \(\tau^{3D}\) is red: \(e_m^{\text{new}} = (v_3, v_4)\).
   c) IF \(\tau^{3D}\) is light black and \(\mathcal{M}(\tau^{3D})\) is black: \(e_m^{\text{new}} = (v_n, v_4)\).
   d) IF \(\tau^{3D}\) is dark black and \(\mathcal{M}(\tau^{3D})\) is black: \(e_m^{\text{new}} = (v_n, v_3)\).
   e) ELSE: \(e_m^{\text{new}} = (v_3, v_4)\).

Now, using this refinement scheme, the algorithm for obtaining an admissible 3D triangulation (or rather tetrangulation) proceeds exactly as in the 2D case (see algorithm 3.3.3). Bängsch showed for the 3D case the same termination property as for the 2D algorithm [25]

Algorithm 3.4.4 (Admissible 3D bisection triangulation)

Let \(\mathcal{T} = \{\tau_1^{3D}, \ldots, \tau_M^{3D}\}\) be a given admissible coarse triangulation, and let \(S \subset \mathcal{T}\) be the set of cells (tetrahedra) marked for refinement.

WHILE \((S \neq \emptyset)\)

1. FOR EACH \((\tau^{3D} \in S)\): refine \(\tau^{3D}\) according to algorithm 3.4.3, and obtain a new triangulation \(\mathcal{T}^*\), remove \(\tau^{3D}\) from \(S\);
2. FOR EACH \((\tau^{3D} \in \mathcal{T}^*)\): IF \(\tau^{3D}\) has hanging node THEN: \(S = S \cup \{\tau^{3D}\}\).

END WHILE

Fig. 3.16. Bisection of a tetrahedron (action of algorithm 3.4.3)
Hexahedral Refinement

As for the extension of 2D triangular refinement strategies to 3D, the hexahedral refinement is just an extension of the 2D quadrilateral refinement given in algorithm 3.3.12. But instead of using arbitrary vertex numbers \( v_n \), as in the 2D case, we introduce local coordinates, taking advantage of the virtually orthogonal structure of quadrilateral grids. The lower left front corner has coordinates \([0, 0, 0]\), where \([·, ·, ·]\) denotes the local coordinate system, while the upper right back corner has coordinates \([1, 1, 1]\) or \([2, 2, 2]\) after re-normalization and refinement (see fig. 3.18).

Algorithm 3.4.5 (Regular hexahedral refinement)

Given a hexahedron \( \tau^{3D} \), defined by its vertices \( v_1 = [0, 0, 0], \ldots, v_8 = [1, 1, 1] \), with faces \( \nu_1, \ldots, \nu_6 \).

1. Refine each face \( \nu_i \) \((i = 1 : 6)\), using algorithm 3.3.12.
2. Connect opposite center vertices. The intersection of these edges yields another new vertex \( v_n \).
3. Define new cells \( \tau_{0,0,0}^{3D}, \ldots, \tau_{1,1,1}^{3D} \) by:

\[
\tau_{i,j,k}^{3D} = \{ [0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1] \} + [i, j, k], \quad i, j, k = 0, 1.
\]

As in the 2D case, hexahedral refinement causes non-admissible triangulations, i.e. hanging nodes. Therefore, a closure mechanism has to be defined, if admissible triangulations are required. For an extension of the 2D algorithm, pyramidal elements can be introduced. However, the number of cases that have to be tackled and the region of influence for closures is by far not as simple as in the 2D case. In fact, it is not possible to define a point insertion algorithm that results in a conforming (purely) hexahedral mesh in complete generality (see chap. 21 in [391] by R. Schneiders). So we will omit the discussion of all those cases here.

Most applications with hexahedral mesh refinement use solution techniques that can handle hanging nodes. Thus, a non-conforming block-wise refinement strategy for local refinement is often applied (see for example [207]).
3.5 Spherical Geometries

In order to find discrete representations of the sphere, we have to start with an initial triangulation \( T \). In most cases, the sphere is triangulated by a 2D triangulation, which is extended to fully 3D geometries by layers of identical 2D grids.

**Triangular Initial Triangulations**

In this subsection we will introduce three different possibilities to triangulate a polygonal approximation to the sphere. In order to represent a spherical domain with triangular elements, we need to either define spherical triangles, or agree to be satisfied with a polygonal representation of the sphere.

When considering the triangulation as a basis for further numerical calculations, one has to keep in mind that a polygonal representation with plane cells introduces first order error terms. Thus, combining a plane cell type with a high order method in spherical geometries is a waste of computational effort. The mathematically correct triangulation which preserves the order, would use curved parametric elements. An introduction to this type of elements can be found in [359]. In order to maintain the order of accuracy it is possible to use a plane polygonal representation but to lift the high order sample points within each element to positions on the sphere’s surface. We will proceed with this approach, and can now start to introduce basic polygonal representations of the sphere.

A natural starting point for spherical triangulations are the platonic solids. Since we are interested in triangular elements in this section, we do not consider the cube in first place. The next choice could be the tetrahedron, but it does not represent a sphere too well. A tetrahedron would be better suited for a hemi-sphere. The next simplest polygonal representations of the sphere is an octahedron with six vertices, 12 edges and eight triangular faces.
Definition 3.5.1 (Octahedral triangulation of the sphere)

The octahedral triangulation of the sphere is defined by the following six vertices:

<table>
<thead>
<tr>
<th>index</th>
<th>spherical coordinate</th>
<th>index</th>
<th>spherical coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0, $\pi/2$)</td>
<td>2</td>
<td>(0, $-\pi/2$)</td>
</tr>
<tr>
<td>3</td>
<td>(0, 0)</td>
<td>4</td>
<td>($\pi/2$, 0)</td>
</tr>
<tr>
<td>5</td>
<td>($\pi$, 0)</td>
<td>6</td>
<td>($3\pi/2$, 0)</td>
</tr>
</tbody>
</table>

The corresponding eight elements are defined by

<table>
<thead>
<tr>
<th>index</th>
<th>vertex</th>
<th>index vector</th>
<th>index</th>
<th>vertex</th>
<th>index vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3, 4, 1</td>
<td></td>
<td>2</td>
<td>4, 5, 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5, 6, 1</td>
<td></td>
<td>4</td>
<td>6, 3, 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3, 4, 2</td>
<td></td>
<td>6</td>
<td>4, 5, 2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5, 6, 2</td>
<td></td>
<td>8</td>
<td>4, 5, 2</td>
<td></td>
</tr>
</tbody>
</table>

Remark 3.5.2 If we consider the unit sphere, with origin (0, 0, 0), the vertices can be given in cartesian coordinates as follows:

<table>
<thead>
<tr>
<th>index</th>
<th>coordinate</th>
<th>index</th>
<th>coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>2</td>
<td>(0, 0, -1)</td>
</tr>
<tr>
<td>3</td>
<td>(1, 0, 0)</td>
<td>4</td>
<td>(0, 1, 0)</td>
</tr>
<tr>
<td>5</td>
<td>(-1, 0, 0)</td>
<td>6</td>
<td>(0, -1, 0)</td>
</tr>
</tbody>
</table>

Another possibility for defining an initial triangulation is the icosahedron. It is a polyhedron with 12 vertices, 20 faces and 30 edges.

Definition 3.5.3 (Icosahedral triangulation of the sphere)

The icosahedral triangulation of the sphere is defined by the following 12 vertices (given in Cartesian coordinates):
3.5 Spherical Geometries

Fig. 3.20. Icosahedral triangulation of sphere: (left) initial triangulation, with 20 triangles; (right) and after refinement, with 1280 triangles

<table>
<thead>
<tr>
<th>index</th>
<th>Cartesian coordinate</th>
<th>index</th>
<th>Cartesian coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>2</td>
<td>((\frac{\sqrt{5}}{2}, 0, \frac{1}{\sqrt{5}}))</td>
</tr>
<tr>
<td>3</td>
<td>((\sqrt{\frac{10-4\sqrt{5}}{25-5\sqrt{5}}}, \sqrt{\frac{2}{5-\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
<td>4</td>
<td>((-\sqrt{\frac{5+\sqrt{5}}{25-5\sqrt{5}}}, \frac{-1+\sqrt{5}}{\sqrt{10-2\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
</tr>
<tr>
<td>5</td>
<td>((-\sqrt{\frac{5+\sqrt{5}}{25-5\sqrt{5}}}, \frac{1-\sqrt{5}}{\sqrt{10-2\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
<td>6</td>
<td>((\sqrt{\frac{10-4\sqrt{5}}{25-5\sqrt{5}}}, -\frac{2}{\sqrt{5-\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
</tr>
<tr>
<td>7</td>
<td>((\sqrt{\frac{5+\sqrt{5}}{25-5\sqrt{5}}}, \frac{1-\sqrt{5}}{\sqrt{10-2\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
<td>8</td>
<td>((-\sqrt{\frac{10-4\sqrt{5}}{25-5\sqrt{5}}}, \frac{1}{\sqrt{5}}, \frac{-1}{\sqrt{5}}))</td>
</tr>
<tr>
<td>9</td>
<td>((-\frac{2}{\sqrt{5}}, 0, -\frac{1}{\sqrt{5}}))</td>
<td>10</td>
<td>((-\sqrt{\frac{10-4\sqrt{5}}{25-5\sqrt{5}}}, \frac{1}{\sqrt{5}}, \frac{-1}{\sqrt{5}}))</td>
</tr>
<tr>
<td>11</td>
<td>((\sqrt{\frac{5+\sqrt{5}}{25-5\sqrt{5}}}, \frac{1-\sqrt{5}}{\sqrt{10-2\sqrt{5}}}, \frac{1}{\sqrt{5}}))</td>
<td>12</td>
<td>(0, 0, -1)</td>
</tr>
</tbody>
</table>

The corresponding 20 elements are defined by (e.g.):

<table>
<thead>
<tr>
<th>index</th>
<th>vertex index vector</th>
<th>index</th>
<th>vertex index vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(3, 4, 1)</td>
<td>2</td>
<td>(4, 5, 1)</td>
</tr>
<tr>
<td>3</td>
<td>(5, 6, 1)</td>
<td>4</td>
<td>(6, 7, 1)</td>
</tr>
<tr>
<td>5</td>
<td>(7, 3, 1)</td>
<td>6</td>
<td>(3, 7, 12)</td>
</tr>
<tr>
<td>7</td>
<td>(11, 12, 7)</td>
<td>8</td>
<td>(7, 6, 11)</td>
</tr>
<tr>
<td>9</td>
<td>(10, 11, 6)</td>
<td>10</td>
<td>(6, 5, 10)</td>
</tr>
<tr>
<td>11</td>
<td>(9, 10, 5)</td>
<td>12</td>
<td>(5, 4, 9)</td>
</tr>
<tr>
<td>13</td>
<td>(8, 9, 4)</td>
<td>14</td>
<td>(4, 3, 8)</td>
</tr>
<tr>
<td>15</td>
<td>(12, 8, 3)</td>
<td>16</td>
<td>(8, 12, 2)</td>
</tr>
<tr>
<td>17</td>
<td>(9, 8, 2)</td>
<td>18</td>
<td>(10, 9, 2)</td>
</tr>
<tr>
<td>19</td>
<td>(11, 10, 2)</td>
<td>20</td>
<td>(12, 11, 2)</td>
</tr>
</tbody>
</table>

**Remark 3.5.4** The rather obscure coordinate values in definition 3.5.3 can be circumvented, when considering a sphere with radius \(r = \frac{1}{2} \sqrt{5}\). Then, the coordinates can be given in 3D cartesian form by
Fig. 3.21. Truncated icosahedral triangulation of sphere (buckyball): (left) initial triangulation, with 180 triangles; (right) after uniform refinement, with 1440 triangles

<table>
<thead>
<tr>
<th>index i</th>
<th>x-coordinate</th>
<th>y-coordinate</th>
<th>z-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>r</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
<td>−r</td>
</tr>
<tr>
<td>3:7</td>
<td>(\cos\left(\frac{2(i-3)\pi}{5}\right))</td>
<td>(\sin\left(\frac{2(i-3)\pi}{5}\right))</td>
<td>0.5</td>
</tr>
<tr>
<td>8:12</td>
<td>(\cos\left(\frac{\pi+2(i-8)\pi}{5}\right))</td>
<td>(\sin\left(\frac{\pi+2(i-8)\pi}{5}\right))</td>
<td>−0.5</td>
</tr>
</tbody>
</table>

Remark 3.5.5 In spherical coordinates \((\lambda, \phi, r)\) the vertex positions can be given numerically and for arbitrary radius \(r\) by

<table>
<thead>
<tr>
<th>index spherical coordinate</th>
<th>index spherical coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ((0, \frac{\pi}{2}))</td>
<td>2 ((0, \frac{-\pi}{2}))</td>
</tr>
<tr>
<td>3 ((0, 0.4636))</td>
<td>4 (\left(\frac{2\pi}{5}, 0.4636\right))</td>
</tr>
<tr>
<td>5 (\left(\frac{4\pi}{5}, 0.4636\right))</td>
<td>6 (\left(-\frac{3\pi}{5}, 0.4636\right))</td>
</tr>
<tr>
<td>7 (\left(-\frac{3\pi}{5}, 0.4636\right))</td>
<td>8 (\left(\frac{\pi}{5}, -0.4636\right))</td>
</tr>
<tr>
<td>9 (\left(\frac{2\pi}{5}, -0.4636\right))</td>
<td>10 (\left(1, -0.4636\right))</td>
</tr>
<tr>
<td>11 (\left(-\frac{3\pi}{5}, -0.4636\right))</td>
<td>12 (\left(-\frac{\pi}{5}, -0.4636\right))</td>
</tr>
</tbody>
</table>

Note that the icosahedron is the dual to the dodecahedron, a polyhedron with 12 pentagonal faces. Therefore a triangulation of the dodecahedron results in the same grids as a triangulation of the icosahedron. An application using the icosahedral grid to derive a spherical geodesic grid is given in [339].

Finally, another possibility to define the initial triangulation of a sphere is to use the truncated icosahedron (better known as buckyball or simply soccer ball). The truncated icosahedron consists of 12 pentagons and 20 hexagons. These polygons are triangulated by adding their midpoint to the vertex list and connecting it to all surrounding polygon vertices. A concise description of the construction of such triangulation is given in [193]. We only give the vertex list for the truncated icosahedron here.

Definition 3.5.6 (Truncated icosahedral triangulation of the sphere)

The truncated icosahedral triangulation of the sphere is defined by the following 92 vertices:
<table>
<thead>
<tr>
<th>index</th>
<th>sph. coord.</th>
<th>index</th>
<th>sph. coord.</th>
<th>index</th>
<th>sph. coord.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(0, \frac{\pi}{3})$</td>
<td>2</td>
<td>$(0.6283, 1.2204)$</td>
<td>3</td>
<td>$(1.8850, 1.2204)$</td>
</tr>
<tr>
<td>4</td>
<td>$(-\pi, 1.2204)$</td>
<td>5</td>
<td>$(-1.8850, 1.2204)$</td>
<td>6</td>
<td>$(-0.6283, 1.2204)$</td>
</tr>
<tr>
<td>7</td>
<td>$(1.2566, 0.9184)$</td>
<td>8</td>
<td>$(2.5133, 0.9184)$</td>
<td>9</td>
<td>$(-2.5133, 0.9184)$</td>
</tr>
<tr>
<td>10</td>
<td>$(-1.2566, 0.9184)$</td>
<td>11</td>
<td>$(0, 0.9184)$</td>
<td>12</td>
<td>$(-0.6283, 0.8141)$</td>
</tr>
<tr>
<td>13</td>
<td>$(0.6283, 0.8141)$</td>
<td>14</td>
<td>$(1.8850, 0.8141)$</td>
<td>15</td>
<td>$(-\pi, 0.8141)$</td>
</tr>
<tr>
<td>16</td>
<td>$(-1.8850, 0.8141)$</td>
<td>17</td>
<td>$(-0.2376, 0.5409)$</td>
<td>18</td>
<td>$(0.2376, 0.5409)$</td>
</tr>
<tr>
<td>19</td>
<td>$(1.0190, 0.5409)$</td>
<td>20</td>
<td>$(1.4942, 0.5409)$</td>
<td>21</td>
<td>$(2.2757, 0.5409)$</td>
</tr>
<tr>
<td>22</td>
<td>$(2.7509, 0.5409)$</td>
<td>23</td>
<td>$(-2.7509, 0.5409)$</td>
<td>24</td>
<td>$(-2.2757, 0.5409)$</td>
</tr>
<tr>
<td>25</td>
<td>$(-1.4942, 0.5409)$</td>
<td>26</td>
<td>$(-1.0190, 0.5409)$</td>
<td>27</td>
<td>$(0.6283, 0.4636)$</td>
</tr>
<tr>
<td>28</td>
<td>$(1.8850, 0.4636)$</td>
<td>29</td>
<td>$(-\pi, 0.4636)$</td>
<td>30</td>
<td>$(-1.8850, 0.4636)$</td>
</tr>
<tr>
<td>31</td>
<td>$(-0.6283, 0.4636)$</td>
<td>32</td>
<td>$(1.2566, 0.1887)$</td>
<td>33</td>
<td>$(2.5133, 0.1887)$</td>
</tr>
<tr>
<td>34</td>
<td>$(-2.5133, 0.1887)$</td>
<td>35</td>
<td>$(-1.2566, 0.1887)$</td>
<td>36</td>
<td>$(0, 0.1887)$</td>
</tr>
<tr>
<td>37</td>
<td>$(0.4220, 0.1725)$</td>
<td>38</td>
<td>$(0.8346, 0.1725)$</td>
<td>39</td>
<td>$(1.6787, 0.1725)$</td>
</tr>
<tr>
<td>40</td>
<td>$(2.0912, 0.1725)$</td>
<td>41</td>
<td>$(2.9353, 0.1725)$</td>
<td>42</td>
<td>$(-2.9353, 0.1725)$</td>
</tr>
<tr>
<td>43</td>
<td>$(-2.0912, 0.1725)$</td>
<td>44</td>
<td>$(-1.6787, 0.1725)$</td>
<td>45</td>
<td>$(-0.8346, 0.1725)$</td>
</tr>
<tr>
<td>46</td>
<td>$(-0.4220, 0.1725)$</td>
<td>47</td>
<td>$(2.7195, -0.1725)$</td>
<td>48</td>
<td>$(2.3070, -0.1725)$</td>
</tr>
<tr>
<td>49</td>
<td>$(1.4629, -0.1725)$</td>
<td>50</td>
<td>$(1.0504, -0.1725)$</td>
<td>51</td>
<td>$(0.2063, -0.1725)$</td>
</tr>
<tr>
<td>52</td>
<td>$(-0.2063, -0.1725)$</td>
<td>53</td>
<td>$(-1.0504, -0.1725)$</td>
<td>54</td>
<td>$(-1.4629, -0.1725)$</td>
</tr>
<tr>
<td>55</td>
<td>$(-2.3070, -0.1725)$</td>
<td>56</td>
<td>$(-2.7195, -0.1725)$</td>
<td>57</td>
<td>$(-\pi, -0.1887)$</td>
</tr>
<tr>
<td>58</td>
<td>$(1.8850, -0.1887)$</td>
<td>59</td>
<td>$(0.6283, -0.1887)$</td>
<td>60</td>
<td>$(-0.6283, -0.1887)$</td>
</tr>
<tr>
<td>61</td>
<td>$(-1.8850, -0.1887)$</td>
<td>62</td>
<td>$(2.5133, -0.4636)$</td>
<td>63</td>
<td>$(1.2566, -0.4636)$</td>
</tr>
<tr>
<td>64</td>
<td>$(0, -0.4636)$</td>
<td>65</td>
<td>$(2.1566, -0.4636)$</td>
<td>66</td>
<td>$(-2.5133, -0.4636)$</td>
</tr>
<tr>
<td>67</td>
<td>$(2.1226, -0.5409)$</td>
<td>68</td>
<td>$(1.6473, -0.5409)$</td>
<td>69</td>
<td>$(0.8659, -0.5409)$</td>
</tr>
<tr>
<td>70</td>
<td>$(0.3907, -0.5409)$</td>
<td>71</td>
<td>$(-0.3907, -0.5409)$</td>
<td>72</td>
<td>$(-0.8659, -0.5409)$</td>
</tr>
<tr>
<td>73</td>
<td>$(-1.6473, -0.5409)$</td>
<td>74</td>
<td>$(-2.1226, -0.5409)$</td>
<td>75</td>
<td>$(-2.9040, -0.5409)$</td>
</tr>
<tr>
<td>76</td>
<td>$(2.9040, -0.5409)$</td>
<td>77</td>
<td>$(1.2566, -0.8141)$</td>
<td>78</td>
<td>$(0, -0.8141)$</td>
</tr>
<tr>
<td>79</td>
<td>$(-1.2566, -0.8141)$</td>
<td>80</td>
<td>$(-2.5133, -0.8141)$</td>
<td>81</td>
<td>$(2.5133, -0.8141)$</td>
</tr>
<tr>
<td>82</td>
<td>$(1.8850, -0.9184)$</td>
<td>83</td>
<td>$(0.6283, -0.9184)$</td>
<td>84</td>
<td>$(-0.6283, -0.9184)$</td>
</tr>
<tr>
<td>85</td>
<td>$(-1.8850, -0.9184)$</td>
<td>86</td>
<td>$(-\pi, -0.9184)$</td>
<td>87</td>
<td>$(2.5133, -1.2204)$</td>
</tr>
<tr>
<td>88</td>
<td>$(1.2566, -1.2204)$</td>
<td>89</td>
<td>$(0, -1.2204)$</td>
<td>90</td>
<td>$(-1.2566, -1.2204)$</td>
</tr>
<tr>
<td>91</td>
<td>$(-2.5133, -1.2204)$</td>
<td>92</td>
<td>$(0, \frac{\pi}{2})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Remark 3.5.7** It should be noted that the triangulations obtained from regular triangular refinement (algorithm 3.3.6) from an icosahedron are equivalent to refinement of a truncated icosahedron and to refinement of a dodecahedron. If we start with an icosahedron and refine two levels, and compare the grid with one refinement of a triangulated dodecahedron and finally with a triangulated truncated icosahedron as described in [193], we get the exactly same triangulations. However, with a bisection refinement strategy this is not true and this is the reason for dealing with it here.
Fig. 3.22. (Left) Lat-Lon grid on the sphere. It can be easily seen that the poles form a singularity. (Right) The reduced Lat-Lon grid relaxes the problem of small mesh size near the pole

Quadrilateral Initial Triangulations

The simplest and most straightforward quadrilateral triangulation of the sphere is a grid consisting of latitudinal and longitudinal coordinates, a so-called Lat-Lon grid. Thus a 1° grid consists of 360 × 180 vertices. However, the two poles (0, 90) and (0, −90) are singular points in this grid. In fact, all grid cells adjacent to the north and south poles are triangles, since all points (λ, ±90) with λ ∈ {0, 360} coincide. Furthermore, many numerical algorithms are restricted in their stability by the mesh size of the smallest grid cell in the mesh. The mesh size at the poles in the Lat-Lon grid approaches 0, which poses severe problems to such algorithms. A 4° × 4° Lat-Lon grid is depicted in fig. 3.22.

One possibility to circumvent the small mesh size near the poles is the Reduced Lat-Lon grid, an example of which is also depicted in fig. 3.22. Here the mesh size is doubled (several times) in the high latitudes, yielding an almost uniform mesh size on the whole sphere. Note that the pole singularity is not removed by this remedy. A semi-Lagrangian solution technique for the shallow water equations on a reduced grid can be found in [259].

A combination of spherical grid and spherically projected unit square grid at the poles, a so-called capped sphere has been proposed by Lanser et al. [252], based on ideas dating back to Phillips, Starius and Browning et al. [74, 317, 318, 319, 375]. This grid avoids the diminishing mesh size at the poles by introducing a regular grid at the poles and combining this grid with the Lat-Lon grid in mid latitudes in an admissible way. Following the notation in [252], a capped sphere is defined by a Lat-Lon grid in the spherical coordinate range of 0 < λ < 2π and φ_{min} < φ < φ_{max}, where 0 > φ_{min} > −π/2 and φ_{max} = −φ_{min}. When we denote N_λ the number of grid points in zonal direction (i.e. the number of longitudinal lines) and N_x the number of grid points in x-direction of the cap grid (we assume N_x = N_y, i.e. the number of grid points in x and y-direction to be identical) then in order to obtain an admissible mesh, we need to require N_λ − 1 = (N_x − 1) * 4. Finally, the cap grid is
3.5 Spherical Geometries

Fig. 3.23. (Left) The capped sphere combines two regular unit square grids projected to the poles with a Lat-Lon grid. (Right) The cubed sphere is a projection of a rectangular grid on the surface of a cube to the sphere. It exposes eight exceptional points, four of them visible in this figure.

given in Cartesian coordinates and projected to the (unit) sphere. For the example given in fig. 3.23 we use $N_x = 15$, $N_\lambda = 57$, $\phi_{\text{min}} = -0.35\pi$ and $(x, y, z) \in [-0.4, 0.4] \times [-0.4, 0.4] \times \{-1, 1\}$.

The cubed sphere is another quadrilateral mesh that avoids the pole problem [345]. It is relatively easy to construct: We project a cube composed of six quadrilateral faces to the sphere, for example by a spherical projection. We start with the grid coordinates given in cartesian coordinates. For an equidistant grid this is given by

\[
\begin{align*}
  x_i^{[n]} &= -1 + i \cdot \Delta x, & y_i^{[n]} &= -1, & z_i^{[n]} &= -1 + i \cdot \Delta x, \\
  x_i^{[f]} &= -1 + i \cdot \Delta x, & y_i^{[f]} &= 1, & z_i^{[f]} &= -1 + i \cdot \Delta x, \\
  x_i^{[l]} &= -1, & y_i^{[l]} &= -1 + i \cdot \Delta x, & z_i^{[l]} &= -1 + i \cdot \Delta x, \\
  x_i^{[r]} &= 1, & y_i^{[r]} &= -1 + i \cdot \Delta x, & z_i^{[r]} &= -1 + i \cdot \Delta x, \\
  x_i^{[b]} &= -1 + i \cdot \Delta x, & y_i^{[b]} &= -1 + i \cdot \Delta x, & z_i^{[b]} &= -1, \\
  x_i^{[t]} &= -1 + i \cdot \Delta x, & y_i^{[t]} &= -1 + i \cdot \Delta x, & z_i^{[t]} &= 1.
\end{align*}
\]

Here, $\Delta x = \frac{2}{N}$, $N$ is the number of grid points in each direction, $i = 0 : N - 1$ is the grid point index. Superscripts $(\cdot)^{[n,f,l,r,b,t]}$ denote near, far, right, left, bottom, and top faces. A basic cubed sphere mesh is shown in fig. 3.23.

It should be noted that there are diverse projection methods for the cubed sphere that result in different uniformity properties of the grid cells as pointed out in [297]. We are not going to dive too deeply into the details here but refer to the literature. See [387] for a spectral element method on the cubed sphere.

The Yin-Yang grid represents an overset gridding method [228]. That means two different grids are composed to construct a mesh covering the whole sphere. This technique is often termed composite mesh generation [84]. In case of the Yin-Yang grid, the complete mesh is constructed of two rotated
Fig. 3.24. (Left and center) The Yin Yang grid consists of two separate grids, composed to one overlapping mesh (right), thereby omitting the pole singularity in mesh size. However it adds the problem of consistency in the overlap region.

partial Lat-Lon grids, the Yin and the Yang grids resp. The Yang grid can be obtained by rotating the grid points of the Yin grid with the following matrix operation

\[ x_{\text{Yang}} = R \cdot x_{\text{Yin}} \]

where \( x \) is the cartesian coordinate of a grid point in either the Yin or the Yang grid and \( R \) is the rotation matrix, given by

\[ R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \]

We are left with defining the Yin grid. This is just the spherical \( j_{\text{max}} \times k_{\text{max}} \)-grid given by coordinates \((\lambda_j, \phi_k), j = 0 : j_{\text{max}} - 1, k = 0 : k_{\text{max}} - 1 \) and

\[ \lambda_j = \lambda_{\text{min}} + j \cdot \Delta \lambda, \]
\[ \phi_j = \phi_{\text{min}} + j \cdot \Delta \phi, \]

where \( \Delta \lambda = (\lambda_{\text{max}} - \lambda_{\text{min}})/(j_{\text{max}} - 1), \Delta \phi = (\phi_{\text{max}} - \phi_{\text{min}})/(k_{\text{max}} - 1), \lambda_{\text{min}} = -\frac{\pi}{4} - \omega, \lambda_{\text{max}} = \frac{\pi}{4} + \omega, \phi_{\text{min}} = -\frac{3\pi}{4} - \omega, \phi_{\text{max}} = \frac{\pi}{4} + \omega. \omega \) denotes the overlap of the two meshes. Figure 3.24 shows the Yin and the Yang grid together with the composite grid for \( \omega = 0 \). There are several modifications, for example for non-overlapping matching Yin and Yang grids or for different methods to minimize the overlap.
Each adaptive algorithm for solving atmospheric flow problems can be separated into two basic parts:

1. grid generation and management,
2. numerical computations discretizing the differential operators.

While the first part consists of mainly integer operations and database management, the second represents the number crunching part. So, the grid generation part needs tree data structures or database-like data structures (hash tables or heaps) for efficient creation, destruction, and search of mesh items. Conversely, in the numerical part, one tries to achieve consecutive data structures which are well blocked such that hierarchical memory architectures can be utilized beneficially. Many AMR methods mix both parts in that they try to refine the mesh in blocks that are large enough to do efficient computations. Additionally the blocks are self similar, such that operators, defined on a block can be re-used on every level of the grid. These operators can be optimized for performance on corresponding processor architectures.

Another class of applications strictly separates the above two basic parts by introducing a gather and scatter step between the grid generation part and the numerical part. This principle is depicted in fig. 4.1.

4.1 Data Structures for Grid Management

In order to assess data structures for grid management, let us first recall an abstract view of a grid. In adaptive atmospheric computations, grids consist of nodes, edges, faces, and cells. Most computations take place on nodes and cells, however, in Finite Volume techniques for example, edges/faces between cells play a crucial role as well.
Fig. 4.1. Two phases of an adaptive algorithm, clearly separated by a gather and scatter between numerical data structures and grid data structures

**Definition 4.1.1 (Grid Atoms)**

- **Node:** A node $v_i$ is a point in the computational domain, defined by its coordinates $x_i \in G \subset \mathbb{R}^d$, $d$ the space dimension ($d = 1, 2,$ or $3$).
- **Edge:** An edge $e_j$ is a (possibly curved) line connecting two nodes $v_1^j$ and $v_2^j$, defined by the pair of nodes $(v_1^j, v_2^j)$.
  Note that in a 2D grid, edges are the interfaces between each two grid cells.
- **Face:** A face $\tau_k$ is a polygon in $\mathbb{R}^2$, defined either by the nodes $v_i^k$, $i = 1 : K$ or the edges $e_j^k$, $j = 1 : K$.
  Note that in a 2D grid, faces represent the grid cells. Note further, that in a 3D grid, faces are the interfaces between each two cells.
- **Cell:** A cell $\tau_{l}^{3D}$ is a polyhedron in $\mathbb{R}^3$, defined by either its vertices (nodes) $v_i^k$, $i = 1 : L_v$, or by its edges $e_j^l$, $j = 1 : L_e$, or by its faces $\tau_k^l$, $k = 1 : L_\tau$.
  It is more common to take either the nodes, or the faces as defining items for cells.

**Remark 4.1.2**

1. For grids in 1D edges represent the cells, while nodes are the interfaces between each 2 cells. In 2D faces represent the cells, while in 3D the definition of cells is consistent with the definition 4.1.1.
2. The number of adjacent grid items also depends on the dimension. For $d = 1$, nodes can have 2 adjacent edges, while for $d = 2$, there can be more. Similarly, an edge can have two adjacent faces for $d = 2$, while for $d = 3$ there might be many more.
3. We do not consider grids for $d > 3$ here.
4. While this abstract list of grid items is valid for all kinds of grids, including quadrilateral locally structured ones, it is probably not as useful for structured as for unstructured grids.
While the grid atoms defined just above are the building blocks of a grid, the
definition of connectivity makes them a grid. There are in principle two ways
to define the connectivity: One list-oriented and one object-oriented approach.
The list-oriented approach originates from times where most technical soft-
ware was implemented in FORTRAN77 where no dynamic data allocation
was available and data structures were essentially restricted to arrays.

**Definition 4.1.3 (List-oriented grid)**

A list-oriented grid data structure can be given by three lists:

1. **Coordinates**: a list with coordinates, this is a \((d \times N)\)-array, where \(d\) is
   the number of dimensions and \(N\) the number of nodes.
2. **Cells**: a list of indices to the cell’s nodes (of size \(K \times M\), where \(M\) is the
   number of cells, \(K\) the number of nodes per cell).
3. **Connectivity**: a list of indices to the cell’s neighbors of size \(L \times M\), where
   \(L\) is the number of faces.

**Example 4.1.4** An example grid is given in fig. 4.2. The corresponding cell
and connectivity arrays according to definition 4.1.3 are given in the following
equation. \(N = 9, M = 10, d = 2,\) and \(K = L = 3\). We denote the cell array
by \(el\) and the connectivity array by \(ne\). If the cell is at the boundary of the
domain and has no neighbor at one side, we indicate this by an entry 0 in the
connectivity array.

\[
\begin{align*}
el &= \begin{bmatrix} 1 & 3 & 4 & 4 & 7 & 7 & 9 & 8 & 8 & 5 \\ 3 & 5 & 5 & 3 & 4 & 5 & 8 & 5 & 6 & 6 \\ 2 & 2 & 3 & 1 & 1 & 4 & 4 & 4 & 5 & 2 \end{bmatrix}, \\
ne &= \begin{bmatrix} 2 & 1 & 0 & 2 & 1 & 4 & 7 & 8 & 3 & 10 \\ 0 & 1 & 4 & 5 & 0 & 5 & 6 & 7 & 8 & 2 \\ 4 & 3 & 8 & 3 & 6 & 0 & 0 & 9 & 0 & 9 \end{bmatrix}.
\end{align*}
\]

Note that the three lists of definition 4.1.3 define a grid completely. One
can easily see, that this type of data structure needs a lot of care when in-
roducing adaptive refinement with changing numbers of cells. Additionally,
when removing cells due to coarsening the mesh, gaps will occur in the arrays raising the need for packing/defragmentation.

A more convenient data structure is an object oriented one. There is much more choice on the design here. Therefore, we will restrict ourselves to one concept, used in the grid generation software amatos [38].

**Definition 4.1.5** (Object-oriented grid)
An object oriented grid data structure consists of grid atoms according to definition 4.1.1, where the interface atoms (edges in 2D and faces in 3D) have information about their two neighbors.

**Example 4.1.6** Looking again at fig. 4.2, we have

- \( N = 9 \) node objects defined by their coordinates (not given here);
- \( L_e = 18 \) edge objects defined by their nodes and knowing their neighbors:

<table>
<thead>
<tr>
<th>item</th>
<th>nodes</th>
<th>neigh’s</th>
<th>item</th>
<th>nodes</th>
<th>neigh’s</th>
<th>item</th>
<th>nodes</th>
<th>neigh’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td>14</td>
<td>9</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>17</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

- \( M = 10 \) elements, defined by their edges:

<table>
<thead>
<tr>
<th>item</th>
<th>edge1</th>
<th>edge2</th>
<th>edge3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

While the object-oriented grid data structure looks less compact (and in fact uses slightly more memory) it is much more versatile. All atoms can be stored in consecutive lists, linked lists, or hash tables, and removing an atom due to coarsening of the grid, leads to only local updates. In fact, only the edges of the concerned elements have to be updated. No defragmentation is necessary and the information needed for the update is immediately at hand. Objects can be combined in derived data structures thus, a convenient data movement in multi-process environments is possible. For more details on grid data structures see chap. 14 of [391]

Finally, we need to create, find and delete elements in an efficient way. From the description of refinement strategies in chap. 3 it is obvious that we will use tree-like data structures for this task.

**Definition 4.1.7** (Tree)
A tree is either

- empty (no nodes), or
- a root and has zero or more sub-trees.
A root is a node in the tree, that has no parents. While a node is either a leaf that has no children or an internal node with parent and children.

Setting up a tree data structure for grid refinement, we need to define a root and then sub-trees. Each element of the initial (coarse) triangulation can be defined a root of a tree. Then, each child of the root element is root of a sub-tree.

**Remark 4.1.8** (Binary tree – bisection refinement)

The bisection refinement strategy (algorithm 3.4.3 or 3.3.1) yields a binary tree, that is a tree where each node has at most two children. In fact, if a node (triangle) has children, then (with bisection) it has exactly two children.

An example of a binary tree together with the corresponding mesh is given in fig. 4.3.

**Remark 4.1.9** (Quad tree – regular refinement)

The 2D regular triangular refinement (algorithm 3.3.6) and the 2D regular quadrilateral refinement (algorithm 3.3.11) both yield a quad tree, where each node has at most four children.

**Remark 4.1.10** (Oct tree – 3D hexahedral refinement)

The 3D quadrilateral refinement (algorithm 3.4.5) yields an oct tree, where each node has at most eight children.
Now, when searching one element of the triangulation, given a coordinate, a fast and efficient way to do so is to traverse the tree. Suppose that we start with only one initial element (root) and have a uniformly refined mesh. Then searching for a leaf element takes \( l \) steps, where \( l \) is the number of refinement levels. The total number of elements (neglecting closures) for \( l \) refinements is \( M = b^l \), where we have \( l = 0 \) for the root element, and \( b = 2, 4, 8 \) the basis according to the correspondences in the above remarks. Thus, a tree search takes \( l = \log_b(M) \) steps, and is almost optimal.

Refinement of elements is possible only at the leaves of the tree data structure. When an element is refined the tree is expanded. Coarsening means cutting of sub trees. A locally refined mesh can therefore have different depths of a tree. Thus, the above calculation of the computational effort for searching is an upper bound. Note that if the initial triangulation consists of more than one element, then we have multiple trees, thus a search takes

\[
M_0 \cdot \log_b(M)
\]

steps, where \( M_0 \) is the number of initial triangles.

### 4.2 Data Structures for Efficient Numerical Calculations

Data structures for numerical calculations are likely to be well suited for optimization if they are in vector like shape. There is a lot of experience with vectorization. Most vectorized code structures can also easily be parallelized. However, this is not the whole story. This section has to investigate efficient neighborhood preserving data structures for adaptive grid refinement.

While the description of grid data structures in the previous section was valid for both unstructured triangular as well as semi-structured quadrilateral refinement strategies, we have to consider structured and unstructured grids separately here. For the semi-structured meshes, a common technique of optimization is the definition of operators on grid patches. We will deal with this in the next section. Here, we will focus on simple optimizations when working on unstructured triangular meshes.

#### 4.2.1 Vectorizable Data Structures

There are several abstract tasks that have to be executed on grid unknowns. Let us denote by \( \rho(x_i, t_j), i = 1 : N, j = 0 : I \), a value \( \rho \) at unknown position \( x_i \) and time step \( t_j \).

1. update unknowns with independent function values: \( \rho(x_i, t_j) = f(x_i, t_j) \),
2. update unknowns with spatially dependent function values:
   \[
   \rho(x_i, t_j) = F(x_i, t_j; \rho(\chi, t_j)), \quad \text{where } \chi \subset \{ x_i | i = 1 : N \}.
   \]
3. update unknowns with space-time dependent function values:
   \[
   \rho(x_i, t_j) = F(x_i, t_j; \rho(\chi, \tau)), \quad \text{where } \chi \subset \{ x_i | i = 1 : N \} \text{ and } \tau \subset \{ t_j | j = 1 : I \}.
   \]
Example 4.2.1 (Finite difference operators)

We can easily formulate a backward time difference \( \partial_t \) or a first order spatial centered difference operator \( \partial_x \) by the above abstract notation:

\[
\partial_t \rho(x_i, t_j) = \frac{\rho(x_i, t_j) - \rho(x_i, t_j-1)}{\Delta t} =: F_t(x_i, t_j; \{ \rho(x_i, t_j), \rho(x_i, t_j-1) \}),
\]

\[
\partial_x \rho(x_i, t_j) = \frac{\rho(x_{i+1}, t_j) - \rho(x_{i-1}, t_j)}{2\Delta x} =: F_x(x_i, t_j; \{ \rho(x_{i+1}, t_j), \rho(x_{i-1}, t_j) \}).
\]

If \( x = (x_1, \ldots, x_N) \) is the consecutive vector of unknown positions and \( r_j = (\rho_1, \ldots, \rho_N) \), where \( \rho_i = \rho(x_i, t_j) \) is the consecutive vector of values, than the first task can be efficiently vectorized (and therefore parallelized) by the vector operation

\[
r_j = F_t(x, t_j).
\]

In order to formulate a vectorizable version of the second task above, we need to divide \( \chi \) into \( \chi_o = \{ x_l | l<i \} \) and \( \chi_n = \{ x_l | l \geq i \} \). Provided that \( F \) is a linear operator, we can write (with the above vector notations)

\[
r_j = F(x, t_j; \rho(\chi_o, t_j)) + F(x, t_j; \rho(\chi_n, t_j)).
\]

When computing this on a parallel or vector computer, a data dependency can occur for the first term on the right hand side. However, this type of data dependency can be treated by common techniques [205].

For the third task in the list above, a similar separation ansatz can be chosen:

\[
r_j = F(x, t_j; \rho(\chi_o, t_j)) + F(x, t_j; \rho(\chi_n, t_j)) + \sum_{t_k \in \tau} F(x_i, t_j; \rho(\chi, t_k)).
\]

A data dependency can again be observed in the first term and has to be resolved.

With the formalization and the examples, most numerical operations in atmospheric modeling software can be covered. And with a vectorizing programming model, we can optimize these operations efficiently.

4.2.2 Maintaining Data Locality – Reordering Schemes

Another aspect of optimization is data locality. So, in addition to vector-like data structures, data in the vectors should preserve some kind of locality properties when mapped from a \( d \)-dimensional mesh to a 1-dimensional vector (see for example [284]).

Example 4.2.2 (Data locality for finite difference Laplace operator)

The second order Laplace operator is defined by \( \Delta \rho = \frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial y^2} \). It can be discretized by using forward and backward finite difference operators with the well known five point stencil (we omit the time index, and use the 2D position \( x_i = (x_i, y_i) \)).
Fig. 4.4. A simple triangular mesh, created by a bisection refinement. The refinement induces a numbering of nodes as given in the left mesh picture. The corresponding connectivity matrix (right) has a non-local structure

$$\Delta \rho(x_i, y_i) \approx \frac{1}{\Delta x^2} \left[ 4 \rho(x_i, y_i) - \rho(x_{i+1}, y_i) - \rho(x_{i-1}, y_i) - \rho(x_i, y_{i+1}) - \rho(x_i, y_{i-1}) \right].$$

Applying this operator to a vector $r_j = (\rho_1, \ldots, \rho_N)$ results in a corresponding matrix $A$ with entries $\frac{4}{\Delta x^2}$ in the diagonal and $\frac{-1}{\Delta x^2}$ in four sub-diagonals. Preserving locality means keeping the sub-diagonals close to the diagonal.

The example suggests a common technique for neighborhood preservation in mapping $d$-dimensional data to 1-dimensional vectors, i.e. matrix reordering. Well-known techniques for data reordering are the Cuthill-McKee algorithm [104], and the minimum degree ordering [161]. We illustrate the reordering strategies by a simple grid and connectivity matrix example illustrated in fig. 4.4.

The Cuthill-McKee algorithm orders a graph by the following procedure. Note that we use the one-to-one correspondence of a graph to the grid, where each edge of our grid corresponds to an edge in the connectivity graph, referred to in the algorithm.

**Algorithm 4.2.3** (Cuthill-McKee reordering)

Let $n_i, i = 1 : N$ be nodes of a graph and let $e_j, j = 1 : M$ be the edges connecting nodes. A node $n_l$ is called adjacent to node $n_i$, if there exists an edge $e_k$ connecting $n_i$ and $n_l$.

1. Start the procedure by selecting a starting node (e.g. the one with lowest degree). Relabel it with 1 (obtaining $n_1$).
2. Relabel all nodes adjacent to $n_1$ in consecutive order (taking the nodes with lower degree first). These nodes are called level 1 nodes, since they have distance 1 from $n_1$ in terms of graph connections.
3. Repeat step 2 for each level $k$ node, obtaining level $k+1$ nodes, $k = 1, 2, \ldots$.

Note that algorithm 4.2.3 terminates when all connections have been traveled which means that all nodes have been numbered (since our grid is equivalent to a connected graph). Cuthill and McKee could show that this algorithm results

---

1 For a formal definition of a connectivity matrix, see sect. 5.1.
in a sparse matrix structure with almost minimum bandwidth. In other words, nodes that are adjacent are close to each other in the consecutive 1D ordering as well. The effect of reordering is depicted in fig. 4.5.

A second and also well introduced reordering scheme is called (approximate) minimum degree ordering. It was introduced by Rosen [346] for minimizing the bandwidth of sparse matrices and is still used in modern software (see e.g. [109]). The original algorithm reads

**Algorithm 4.2.4 (Minimum degree reordering)**

1. Determine those nodes that cause the largest bandwidth in the sparse matrix (i.e. that are farthest away from each other in the consecutive numbering scheme).
2. Determine, whether an interchange of the higher node index, say $i$, with some other node index, say $j$, yields a smaller bandwidth (i.e. moves it closer to its neighbors).
3. If yes, interchange $i$ and $j$.
4. Repeat steps 1. to 3. until no further interchanges are possible.

There are much more efficient ways to compute the minimum degree reordering (see e.g. [110]); however, this discussion is not in the scope of our presentation. We used an implementation of the minimum degree column ordering as given in MATLAB [8] to compute the reordering given in fig. 4.5.

A very efficient and in most cases even better reordering mechanism stems from space-filling curves (SFC). A geometric construction of SFC was originally proposed by Hilbert [203], based on ideas by Peano [310], for proving a set theoretical problem. A space-filling curve passes through all points of a multi-dimensional domain, thereby, mapping the multi-dimensional index space to a one-dimensional set (curve). One can show that SFC preserve neighborhood relations. Moreover, space-filling curves are well suited for parallelization, since they induce a partition of the domain (see sect. 5.1).

It is easy to calculate SFC for adaptive grids. In fact, for triangular bisection, the index calculation consists of just one bit-manipulation per triangle [43]. The SFC for triangular bisection grids corresponds to a Sierpinsky curve.
[45, 429]. In contrast, a well suited SFC for quadrilateral grids is the Hilbert curve, used for example by Roberts et al. [344] and by Griebel and Zumbusch [178]. Hilbert’s curve can also be constructed by recursion [73]. Several alternative patterns are proposed in [270].

The following data have to be known a priory:

a) the number of triangles in the initial triangulation $N_0$,
b) the maximum number of refinement levels $l$.

With these data, for each element we need a bit structure of length $b = \log_2(N_0) + l$. While the first $b - l$ bits are used for consecutively numbering the initial elements arbitrarily, each level is then represented by an additional bit. To illustrate the following algorithm, observe fig. 4.6.

**Algorithm 4.2.5** (Space-filling curve for 2D bisection)

Let $\tau^k$ be a cell on level $k$ of the grid, and we denote with $\tau^k_i$, $(i = 1 : 2)$ both daughters of cell $\tau^{k-1}$. For simplicity, we assume only one cell $\tau^0$ in the initial triangulation, therefore $b = l$.

1. The algorithm starts with a zero bitmap of length $b$ in $\tau^0$.
2. FOR each level ($k = 1 : l$) DO:
   a) copy the mother’s ($\tau^{k-1}$) bitmap to both daughter cells ($\tau^k_i$);
   b) determine left or right side cell $\tau^k_e$ according to the level:
      \[
      \begin{cases}
      \tau^k_e = \text{left}, & \text{if } \text{mod}(k, 2) = 0, \\
      \tau^k_e = \text{right}, & \text{if } \text{mod}(k, 2) = 1;
      \end{cases}
      \]
   c) set the $k$-th bit of daughter $\tau^k_e$ to 1.
3. END FOR

The effect of the SFC reordering on data locality, i.e. the connectivity matrix is shown in fig. 4.7. Although the SFC reordering causes some entries far from the diagonal, which means far away in memory, in most cases the locality is very well maintained. Note that for many numerical computations involving sparse matrices the inverse of the SFC numbering is even better, since it prevents creation of fill-in.
Vector-like data structures with neighborhood-preserving ordering are a handy tool for optimized numerical calculations. One technique to gain performance in adaptive (unstructured) programs is therefore to gather data from grid data structures into vector-like data structures before entering the numerical computation parts of the program. This paradigm of code structuring has been used in the grid generation software amatos described in sect. 4.5 below.

4.3 Working With Grid Patches

In contrast to triangular refinement strategies, where cells are created individually, in block-structured quadrilateral methods, a whole block of cells is the building block of the mesh. When using self-similar blocks (i.e. the same number of cells in each block, regardless of the refinement level) discrete mathematical operators like difference operators (for finite difference methods) or flux operators (for finite volume methods) can be defined in an abstract and optimized way.

Definition 4.3.1 (Grid Patch)

A grid patch is an abstract data structure defined by the following properties:

1. the position, for example the \((x,y,z)\)-coordinate of the lower left corner;
2. the grid level within the refinement hierarchy (alternatively or additionally the grid spacing can be used as defining parameter);
3. the number of grid-points in each space dimension (in other words, the size of the grid patch).

There may be additional information like neighborhood relations, ghost cells, mother/daughter grid patches, etc. attached to the data object grid patch.

With this definition an AMR supporting software can define iterators that handle all grid patches in an adaptively refined mesh. By defining grid patches by some (few) characteristic parameters, a simplified handling of grids is possible.
Now, mathematical operators like differential operators that are needed to
discretize the underlying model equations, can be defined patch wise. Using
grid patches of a limited variability in size even allows for hardware opti-
mized implementations of operators. Jablonowski [221] reports on optimal
patch sizes, where the cache size of the microprocessor architecture plays an
important role. Balancing the size of a grid patch is an optimization problem
stated in the following way:

**Remark 4.3.2 (Balancing grid patch size)**

The minimum time required for solving a given problem by an AMR method
with grid patches is obtained by choosing the optimum between the following
competing functions:

1. the time per unknown decreases with increasing patch size up to a satura-
tion limit (usually the cache size);
2. the overall time for solution grows linearly with the number of unknowns;
3. the number of (overhead) unknowns grows with increasing patch size;
4. the number of patches grows with the number of refinement regions/ local-
ized features.

Note that this type of balancing is not necessary (and not applicable) for
triangular grids with refinement strategies described in chap. 3. On the other
hand, optimized algorithmic blocks defined on grid patches are not possible
on triangular grids either. Giving up some flexibility in the grid adaptation
yields enhanced simplicity and performance in grid operations.

### 4.4 Software Packages for Adaptive Grid Management

Several authors have contributed packages that ease the implementation of
adaptive models. Many of these tools are either purely research tools or have
been implemented for a broad application spectrum. One such tool is called
**ALBERTA** and maintained by Kunibert Siebert and Alfred Schmid [354, 353].
It is a finite element toolbox that is well structured in a hierarchical way
and is well suited for academic research and education. However, since it
is not parallelized, it cannot be used for real life computationally intense
applications. It can be regarded as a reference for data structures and code
design.

Several other tools for triangular and quadrilateral grid generation and
finite element support are available. We want to reference only a few. A com-
prehensive overview of finite element software can be found at the Internet
**Finite Element Resources** [424]. **QMG** is a research tool that works within
either MATLAB or with a Tcl/Tk interpreter. It aims at solving finite el-
ment discretized problems with complex geometries in 2D and 3D [399].
**CUBIT** from Sandia National Laboratory is a more production oriented tool
for finite element simulation [306]. **CUBIT** is a collection of tools for gen-
erating unstructured 2D and 3D meshes, based on tetrahedra or hexahedra,
and geometry preparation. It is mainly built to generate meshes from given geometries. A comprehensive collection on mesh generation tools can be found online [307, 355].

AMROC by Ralf Deiterding is a block structured AMR tool implemented in an object oriented manner [114, 113]. It is parallelized for distributed memory architectures and serves as an object oriented framework for flow solvers. An example is given for LeVeque’s CLAWPACK (see below). SAMRAI is a framework for structured adaptive mesh refinement, developed at Lawrence Livermore National Lab [416].

Collela et al. at Lawrence Berkeley National Laboratory have developed CHOMBO a quite sophisticated tool for parallel structured adaptive mesh refinement [98, 97]. Cart3D is a Cartesian mesh based grid generation and computational fluid dynamics package [4]. Of course, the mother of all block structured AMR packages CLAWPACK (formerly AMRCLAW) by LeVeque and coworkers is available free of charge for research purposes [262]. Blikberg parallelized an older version of AMRCLAW in an application to solve the shallow water equations [63]. The latest version (4.2 at the date of writing this) is parallelized by the message passing interface (MPI) and is therefore capable of computing on distributed memory machines.

There are hardly any grid generation packages available that generically support two-dimensional spherical geometries as required by global or large scale atmospheric modeling. For triangular unstructured meshes such a tool, amatos, is presented in the next section [38, 44, 32]. A tool that supports spherical geometries with block structured meshes is used by Jablonowski in her adaptive atmospheric model [221]. The blocked spherical adaptive mesh refinement tool has been developed by Oehmke and Stout [304].

4.5 Example for a Grid Handling Software Package – amatos

amatos is designed to reduce the code complexity for the application programmer. At the same time, amatos aims at efficiency, parallelization and ease of use. Applications that have been realized using amatos include stationary problems of simple potential calculations, tracer advection in 2D and 3D, and implementation of dynamical cores for weather prediction and climate simulation.

amatos supports different numerical schemes but has been primarily designed with semi-Lagrangian respectively Lagrange-Galerkin schemes in mind. The library supports finite element (FEM) type spatial discretizations, including spectral elements.

This section introduces the key techniques behind the grid generating library. These topics cover grid generation, efficient data structures, including space-filling curve (SFC) ordering, interpolation of scattered data, including
radial basis function (RBF) interpolation, FEM support issues, and computational geometry issues. For a detailed technical documentation of amatos the reader is referred to [32, 37].

Main Data Structures

We start this technical description with an introduction of the main data structures of amatos. While implemented in Fortran 90, the design of amatos tried to adhere to object oriented principles. The objects of a mesh are nodes, edges, elements (or faces), and tetrahedra. Each of these objects is defined by a certain set of data, has attributes and may be linked to other items. Thus, each grid item is represented by a derived structured data type with substructures def, att, and lnk, for definition, attributes, and links resp.

The node objects are defined by their physical coordinates r.coor. Additionally each node has a unique identifying index i.indx. Attributes of a node are it’s time stamp, an indicator for it’s time of occurrence, it’s patch (surrounding elements), the edge index it divides, and it’s physical values. Finally a node links to it’s periodic partner, when periodic boundary conditions are applied.

Edge objects are defined by their node indices p.node and again by a unique identifier i.indx. Attributes of an edge are among others the time stamp, it’s state of refinement, it’s boundary condition (if it is a boundary edge), it’s values, and if parallelized it’s processor number. Links are set to daughter edges (forming a 1-way binary tree) and periodic partner.

Elements (faces) are defined by their nodes (vertices) p.node or alternatively by their edges p.edge. amatos uses both definitions redundantly (and takes care for consistency). Element’s attributes are time stamp, indicator for it’s time of occurrence, state of refinement, level of refinement, it’s SFC index, and it’s values. Links are set to the mother and daughter elements (forming a 2-way binary tree structure).

In principle, hash tables or linked lists would have done the job of managing all items elegantly, but for the sake of quick random access, pointer arrays have been chosen to store the addresses of all items. The main disadvantage of this management strategy is the need for packing/defragmenting these arrays from time to time, when coarsening has been conducted.

Several index lists are maintained for convenience and to achieve higher efficiency for certain grid access structures. Index lists formally are just permutation or selection vectors. There is an index list for all elements of the finest local level (i.e. all elements that form the actual computational mesh), there is a list for all boundary edges, etc.

Mesh Refinement Strategy

Currently, there are triangular (for 2D) and tetrahedral meshes implemented in the amatos-framework. In order to refine a given triangular/tetrahedral
mesh cell, amatos employs a bisection refinement strategy, originally proposed by Rivara [341] and refined by Bänsch [25]. This strategy is based on bisecting a marked edge and a marking algorithm that prevents small angles in the refined mesh, as described in sect. 3.3. With this algorithm no special treatment is necessary for hanging nodes, since hanging nodes can be avoided by an iterative closure. A space-filling curve for ordering the mesh cells can be constructed extremely easily as shown in [43] and described in sect. 4.2.2. With uniform refinement every two levels all edges are halved. One disadvantage of this refinement strategy is the potential for criss-cross meshes that have shown numerical instabilities in certain situations (see for example [300]). For three-dimensional tetrahedral grids, again an extension to the 2D bisection algorithm proposed by Bänsch is followed. With these refinement and coarsening strategies, amatos is capable of creating arbitrarily refined meshes.

**Gather/Scatter and Efficient Data Structures**

amatos clearly separates the two phases of an adaptive calculation:

1. grid generation/modification phase,
2. numerical calculation phase.

Calculations in the grid generation phase are performed on trees of grid atoms (i.e. tetrahedra, triangles, edges, nodes). Each atom in the grid has a unique identification index and can be accessed efficiently.

Once the mesh has been computed, a gather step collects data required for the numerical computation from the grid data structures within amatos. The result of the gather is a vectorized data structure, e.g. a vector of all unknowns (with dimension \(N\), where \(N\) is the total number of unknowns), or a matrix of all coordinates (with dimension \((d,N)\), where \(d\) is the coordinate dimension and \(N\) as above). Once the numerical computing phase has been completed, a scatter step has to be performed to write back new data into mesh data structures. Finally, the mesh can be manipulated, and data is automatically propagated with the mesh. Figure 4.1 at the beginning of this chapter illustrates this principle.

While the gather/scatter steps add a certain overhead to the whole procedure, this is compensated by the higher efficiency in both the grid generation phase and computational phase. In many of our practical problems, we see less than 1% overhead introduced by gather/scatter.

It should be mentioned that the gather/scatter routines contain a lot of intelligence, since a re-numbering of the items takes place according to the space-filling curves. Additionally these routines react differently depending on the action required by the user. Since either tetrahedral, triangular, edge or nodal values have to be accessed, this is a nontrivial task. When higher order elemental unknowns are gathered, they have to be accessed from several item types, adding to the logical complexity of the gather/scatter operation.
Space-Filling Curves for Ordering Unstructured Grids

One of the unique features of amatos is the integrated utilization of space-filling curve (SFC) techniques for ordering and partitioning grid items. The discrete form of space-filling curves is used in computer sciences in several different situations as indicated in [429]. There are two properties of SFC that are beneficial for computations (see sect. 4.2.2):

1. mapping of $d$-dimensional domain to 1 dimension,
2. preservation of data locality.

In the context of amatos’s meshes, we construct a discrete SFC that is fine enough to have a curve-node in each element, thus inducing a consecutive numbering of elements (this is a result of the first property above). The second property guarantees connected and locally compact partitions, when the consecutive numbering is used for partitioning the computational triangulated domain.

To obtain an ordering of other mesh items like nodes or edges, we follow the ordering of elements and then collect those nodes/edges contained in the corresponding elements that have not yet been collected. This strategy results in data locality not only for elemental data but also for all other unknowns.

Interpolation of Scattered Data

For several numerical applications, interpolation of data on given grids is required. One main class of applications is the semi-Lagrangian time discretization where upstream values have to be interpolated. Since amatos’s grids are irregular, interpolation methods have to deal with scattered data.

There are three different types of interpolation methods available in amatos:

1. Bi-cubic spline interpolation based on a Hermite representation on elements;
2. radial basis function interpolation;
3. interpolation based on the finite element representation of functions.

Interpolation based on cubic splines has been described in [33, 34]. We basically estimate the gradients at element’s nodes and calculate a cubic interpolating function from nodal values and nodal (estimated) derivatives. For more details on computing gradients on unstructured grids, see sect. 6.1.

The radial basis function (RBF) interpolation is described in detail in e.g. [40, 75, 217] and is covered as well in sect. 6.1.4. amatos uses thin plate spline RBF for interpolating scattered data. A RBF interpolation for $l$ given function values $\rho(x_i)$ at sample points $x_i$, $i = 1 : l$, looks as follows

$$I_f(x) = \sum_{i=1:l} [\lambda_i \cdot \phi(||x_i - x||)] + P(x),$$
where \( I_f(x_i) = \rho(x_i) \) for \( i = 1 : l \) fulfills the interpolation condition, \( \|\| \) is the Euclidean norm, and \( P(x) \) is a polynomial that is required to close the arising problem of calculating coefficients \( \lambda_i \) from the given interpolation conditions. \( \phi \) are the radial basis functions, in this case

\[
\phi(r) = r^2 \log(r).
\]

The finite element based interpolation is straight forward. Let \( x_i, i = 1 : n \) be the unknowns of the element that contains the interpolation coordinate \( x \). Then the interpolating function is just the FEM expansion

\[
I(x) = \sum_{i=1:n} \rho_i b_i(x),
\]

with \( b_i \) the FEM basis functions and \( \rho_i \) the corresponding FEM coefficients corresponding to nodal values \( \rho(x_i) \).

Note that all these interpolations are included in \texttt{amatos}'s programming interface, since interpolation needs access to individual nodes and this can be implemented much more efficiently by accessing the (object oriented) grid data directly. Moreover, the unified interface allows for rapid exchange of different orders of interpolation for testing purposes.

As an example of how to call the interpolation, we demonstrate the interpolation of upstream values in an application of atmospheric tracer transport.

\textbf{Example 4.5.1} The following code segment shows the usage of three different routines in \texttt{amatos} programming interface: \texttt{grid_domaincheck}, \texttt{grid_boundintersect}, and \texttt{grid_coordvalue}. While the first two routines are used to guarantee that an upstream point in the semi-Lagrangian time stepping scheme is positioned in the domain, interpolation is invoked by \texttt{grid_coordvalue}. The following data structures are used:

- \texttt{p_mesh} is a grid handle data structure referring to the actual mesh;
- \texttt{r_coord} is an array with node coordinates;
- \texttt{r_upstr} is an array with upstream coordinates;
- \texttt{GRID_highorder} is pre-defined in \texttt{amatos} and invokes cubic spline interpolation;
- \texttt{i_val} is a pointer to a registered variable in \texttt{amatos}.

We demonstrate the loop over all nodes computing the interpolated upstream value:

```fortran
!---------- loop over nodes: find element containing upstream point
node_loop: DO i_cnt=1, i_arlen
!---------- check if upstream value is outside of the domain
   i_out= grid_domaincheck(p_mesh(i_time), r_upstr(:,i_cnt))
!---------- take the intersection of the trajectory with the
!---------- boundary as new upstream point
   out_domain: IF(i_out /= 0) then
      r_upstr(:,i_cnt)= grid_boundintersect(p_mesh(i_time), &
      r_coord(:,i_cnt), r_upstr(:,i_cnt), i_info=i_stat)
```

```fortran
```
no_intersect: IF(i_stat /= 0) THEN
    r_rside(i_cnt)= 0.0
    CYCLE node_loop
END IF no_intersect
END IF out_domain
!---------- interpolate
    r_rside(i_cnt)= grid_coordvalue(p_mesh(i_time), r_upstr(:,i_cnt), &
    i_interpolorder=GRID_highorder, i_valpoint=i_val)
END DO node_loop

**Finite Element Support**

*amatos* provides a flexible support for finite elements. The definition of element types includes the assignment of unknowns (degrees of freedom, DOFs) to grid items like elements (interior DOFs), edges (edge DOFs) and nodes. Element types are defined by a signature file.

In order to provide a flexible interface to all kinds of finite elements, *amatos* uses a signature data structure. The signature of a specific finite element contains the essential information of that element type:

- name, order, and total number of DOFs,
- number of DOFs on the element’s vertices,
- number and position of DOFs on element’s edges,
- number and position of DOFs within the element,
- coefficients of (Lagrange) basis and quadrature,
- evaluation of basis functions at quadrature points.

The signature allows for a generic programming of finite element related routines (e.g. computing of stiffness or mass matrix), independent of the basis chosen. Examples for this generic programming are given in the accompanying test driver software for *amatos*. With this support, it is not only possible to work with finite elements but also to apply spectral elements for high order approximations on triangles [309].

**Computational Geometry Applied**

Some applications related to amatos require geometric manipulations of grid items. Important classes of geometric manipulations are the point intersection of a given vector with the domain boundary, and area intersection of a given polygon with grid elements. Usually, these calculations require direct access to mesh items, so following *amatos*’ philosophy they are included in the programming interface.

Geometric intersection of a given vector with the boundary can be computed with the routine `grid_boundintersect`, where a vector is given by a start and end point and the intersection point’s coordinates are returned. The algorithm behind the intersection is based on the parametric representation of lines. It can be found in almost every book on computational geometry.
Algorithm 4.5.2

1. Input: Line $AB$ and line $XY$;
2. Check whether $A$ on same side of $XY$ as $B$, if yes exit;
3. Check whether $AB$ parallel to $XY$, if yes exit;
4. Calculate intersection by

$$
\frac{(Y - X) \times (X - A)}{(X - Y) \times (B - A)} \cdot (B - A) + A.
$$

The algorithmic realization of the area intersection can also be found in standard literature on computational geometry [9]. It follows a very simple principle, but requires both intersected polygons to be oriented and convex (convexity has to be guaranteed by the user). Given a polygon and a triangle of the triangulation. Then the algorithm runs through each edge of the triangle and cuts off those parts of the polygon that are to the right (outside) of the edge. This algorithm effectively results in calculating line intersections of the polygon lines and the triangle lines. The main ingredient is the calculation of cross products.

Algorithm 4.5.3

1. Input: Polygon $v_j$ ($j = 1:n$) and triangle $x_i$ ($i = 1:3$);
2. For each $i = 1:3$ DO:
3. For each $j = 1:n$ DO:
   a) IF $v_j$ is right of $x_i x_{i+1}$ (i.e outside of triangle) THEN
   b) calculate line intersection $w_i$ of $x_i x_{i+1}$ with $v_j v_{j+1}$;
   c) take $w_i$ as new node in intersection polygon.
   d) END IF
4. END DO

Now, to intersect a given polygon with all mesh cells that have an intersection area, requires to find those cells. amatos uses a recursive algorithm/implementation to achieve this, accessible by the interface routine grid_polygridintersect (see [38] for a list of routines).

Algorithm 4.5.4

1. Input: Polygon $P$ and mesh $M$;
2. For each cell $\tau$ of the coarsest level in $M$ DO:
   a) calculate area intersection of $\tau$ with $P$, $A = \tau \cap P$;
   b) IF $A \neq \emptyset$ AND $\tau$ has children $\tau_1$ and $\tau_2$ THEN:
   c) Set $\tau \leftarrow \tau_i$ ($i = 1:2$), GO TO step 2a;
   d) ELSE return.
   e) END IF
3. END DO
Structure of amatos Application

To finalize this chapter we give an abstract structure of a typical application that uses amatos as a mesh handling tool. We will not specify the individual tasks but assume that there is some method available for

- computing new values on grid data (function `compute`);
- computing an element based refinement criterion (function `criterion`);
- initialize data (function `initialize`);
- define domain geometry (array `r.poly`);
- obtain initial (coarse) grid (file `triang.file`);
- define finite element signature (file `signature.file`).

Example 4.5.5 (Fragment of an application using amatos)

```fortran
!---------- initialize the fem signature
i_typ = grid_registerfemtype(signature.file)
!---------- initialize amatos data structures
CALL grid_initialize
!---------- register variable pointer i_var
i_var = grid_registerfemvar(i_typ)
!---------- set coarse and fine grid level
CALL grid_setparameter(p_grid, i_coarselevel= i1, i_finelevel= i2)
!---------- define domain and create initial triangulation
CALL grid_definegeometry(num.vertices, r_vertexarr= r_poly)
CALL grid_createinitial(p_grid, c_filename=triang.file)
!---------- initialize data
CALL initialize(p_grid)
!
!---------- this is the main adaptive loop
!
adapt_loop: while(l_adapt)
!---------- gather data from grid, use values in pointer i_var
CALL grid_getinfo(p_grid, i_femtype=i_typ, r_dofvalues=r_vals, &
i_arraypoint=(/i_var/))
!---------- do computations on r_vals, r_vals is overwritten with new values
CALL compute(r_vals)
!---------- scatter computed values back
CALL grid_putinfo(p_grid, i_femtype=i_typ, r_dofvalues=r_vals, &
i_arraypoint=(/i_var/))
!---------- refinement criterion based on r_vals, returns flags
CALL criterion(r_vals,i_flag)
!---------- apply flags to the grid
CALL grid_putinfo(p_grid, i_elementstatus=i_flag)
!---------- adapt the grid according to flags
CALL grid_adapt(p_grid,l_adapt)
END DO node_loop
!---------- terminate application
CALL grid_terminate
```

The example shows a very simple structure of an amatos-based program. After initialization that comprises registration of FEM type, variables and geometry definition, the main adaptation loop consist of only a few steps. First
data is gathered from the grid by \texttt{grid\_getinfo}. Then some user specified computations take place. The data has to be written back to the grid before any adaptation in order to maintain consistency of data. Flags for refinement or coarsening have to be derived from the refinement criterion and scattered into the grid data structure. A simple call to \texttt{grid\_adapt} invokes refinement and coarsening and leads to a new conforming mesh.
Issues in Parallelization of Irregularly Structured Problems

Problems in atmospheric modeling were counted among the grand challenges a few years ago. For almost a decade there has been extensive research in order to develop modeling software that is capable of utilizing modern high performance computing architectures (see e.g. [3, 103, 118, 154, 176, 181, 186, 201, 235, 237, 302, 333, 342, 418]). High performance inevitably means parallel computing. When dealing with adaptive mesh refinement, parallelization becomes a non-trivial issue for several reasons.

- Load balancing has to be performed dynamically, during the simulation, since load imbalances can (and will) occur frequently, when local phenomena change their location.
- Adaptively refined grids are not topologically simple, therefore partitioning strategies have to be more sophisticated than for non-adaptive grids.
- Since partitioning has to take place during the simulation, it needs to be parallelizable, scalable, and fast/efficient to be useful.
- Data movement has to be minimized even with changing locations of refinement.
- Boundaries between processor’s partitions have to be as short as possible to minimize inter-process communication. In other words, one is interested in maximizing the volume to surface ratio of partitions.
- With irregular (e.g. triangular) meshes it becomes difficult to maintain halo regions, i.e. overlapping redundant boundary regions for lower communication costs.

Most of the above problems can be tackled by advanced data structures as discussed in chap. 4. However, a partitioning strategy is still required. This chapter will therefore focus on partitioning methods. Load balancing for the partitions is essential. Therefore, the load balancing problem will be dealt with in more detail. We will only investigate in more detail load balancing of the computational grid. It should be noted that load balancing in adaptive methods has to consider the (hierarchical) refinement [396].
For parallel strategies to generate unstructured meshes for adaptive mesh refinement there is a rich literature. We can only mention a selection here. For some general approaches and data structures for irregular tetrahedral meshes see [173, 225, 227]. An error estimate driven domain decomposition method that uses parallel instances of serial grid generation software has been proposed by Bank and Holst [21]. Finally, it should be noted (however, not elaborated) that the efficiency of adaptive applications is severely dependent on programming models on the target computing architectures [364].

5.1 Partitioning Strategies

In order to distribute the computational load uniformly among processors, in adaptive methods it is not sufficient to just split the domain naively. Since refinement regions may occur and vanish and also may move, partitioning has to take place frequently. The partitioning strategy should lead to a distribution of mesh atoms such that the above mentioned quality criteria are met, i.e.

- equilibrate the load,
- minimize the edge cut, and
- minimize the re-partitioning cost.

Solving this problem exactly is NP-hard, therefore heuristics or approximate methods have to be employed. Additionally, the partitioning problem is in most cases an unstructured problem. In this section, we will introduce four different classes of partitioners.

- Spectral bisection partitioning;
- Kernighan-Lin partitioning;
- Multi-level methods;
- Space-filling curves.

Each such procedure is in principle capable of handling quadrilateral as well as triangular meshes. Of course there are more strategies to partitioning. The reader is referred to [62, 76, 108, 123, 124, 150, 250, 404]. An overview of parallel load balancing schemes for adaptive methods can be found in [224].

To start the discussion of (unstructured) grid partitioning, we introduce the notion of graphs. The meshes described in sect. 3 can all be represented by an undirected graph.

**Definition 5.1.1 (Undirected Graph, vertex degree)**

Let $V$ denote the set of graph vertices, and $E$ the set of edges. $|\cdot|$ denotes the number of set members. Then by $G(V,E)$ we denote the graph of vertices $v_i \in V$ connected by edges $e(v_i,v_j) \in E$.

Let $v$ be a vertex of the graph. Then the degree $g$ of the vertex is defined as $|e_n|$, where $e_n \subset E$ is the subset of $E$ of all edges that contain $v$. 

Note that we assume implicitly that undirected graphs do not contain self edges, i.e. edges $e(v_i, v_i)$, and do not contain multiple edges between two vertices. We will also use the term node instead of vertex.

Example 5.1.2 (Triangular 2D finite element mesh)

For a finite element mesh it is most convenient to identify the elements with graph vertices. Then, the graph edges can be identified with the mesh edges. Even if the mesh edges (faces) separate two elements, they provide the neighborhood information required for the graph edges to connect two elements (see fig. 5.1). The graph edges, therefore, represent the neighborhood relations in the mesh.

Example 5.1.3 (Quadrilateral 2D finite difference mesh)

In a finite difference mesh the graph edges correspond to the mesh edges, since the vertices correspond to the grid nodes (see fig. 5.1). The graph edges, like in example 5.1.2, represent the neighborhood relations, in other words the finite difference stencil.

Now, we are able to formulate the partitioning problem in terms of a graph. Let $p$ be the number of processors, and let $G(V, E)$ be the graph corresponding to the given (adaptively refined) mesh, with $n = |V|$ the number of grid atoms (graph vertices) to be partitioned. Then we look for a partitioning $P = (V_1, \ldots, V_p)$ into sub-graphs $G(V_k, E_k)$ ($k = 1 : p$) with the following properties:

1. $V_k \cap V_l = \emptyset$ for $k \neq l$, $k, l = 1 : p$.
2. $\bigcup_{k=1}^{p} V_k = V$.
3. $|V_k| = \left\lceil \frac{n}{p} \right\rceil$ for all $k = 1 : p$.

In most cases, we will not be able to fulfil the last property exactly, since we need an integer number of vertices in each $V_k$, so we might relax this to $|V_k| = \left\lceil \frac{n}{p} \right\rceil$, where $\left\lceil \xi \right\rceil = \min_{i \in \mathbb{Z}}(i \geq \xi)$ for $\xi \in \mathbb{R}$ is the ceiling function.

Note that the above properties imply that the partitions $G(V_k, E_k)$ are connected graphs, inducing connected sub-meshes. If this cannot be guaranteed by the partitioning method, we might relax the properties, by allowing $k = 1 : q$, with $q > p$. 

---

**Fig. 5.1.** Examples of graphs related to meshes. (Left) An element-based graph for triangular finite element meshes; (right) a node-based graph for quadrilateral finite difference meshes.
In sect. 4.2 a connectivity matrix was used in illustrations. Here, we want to formally define the (sparse) matrices associated to a graph, and therefore, to the corresponding mesh. With this association of

\text{Mesh} \leftrightarrow \text{Graph} \leftrightarrow \text{Matrix}

we are able to use powerful mathematical tools for matrix analysis in order to derive appropriate partitions.

**Definition 5.1.4 (Incidence/ Laplacian/ connectivity matrix)**

Let \( G(V, E) \) be an undirected graph and let \(|V| = n, |E| = m\). Then the incidence matrix \( I \in \mathbb{N}^{n \times m} \) is defined by

\[
I_{i,j} = \begin{cases} 
1, & \text{if } e(v_i, v_k) \text{ is the } j\text{-th edge}, \\
-1, & \text{if } e(v_k, v_i) \text{ is the } j\text{-th edge}, \\
0, & \text{otherwise},
\end{cases}
\]

for some \( 1 \leq k \leq n \). The connectivity matrix \( C \in \mathbb{N}^{n \times n} \) corresponding to \( G(V, E) \) is defined by

\[
C_{i,j} = \begin{cases} 
1, & \text{if } i = j, \\
1, & \text{if } e(v_i, v_j) \text{ exists}, \\
0, & \text{otherwise}.
\end{cases}
\]

The Laplacian matrix \( L \in \mathbb{N}^{n \times n} \) is defined by

\[
L_{i,j} = \begin{cases} 
d, & \text{if } i = j, \text{ degree of node } i \\
-1, & \text{if } e(v_i, v_j) \text{ exists}, \\
0, & \text{otherwise}.
\end{cases}
\]

Note that the Laplacian matrix and the connectivity matrix, defined above, have the same sparsity pattern. Some authors define the diagonal of the connectivity matrix to be zero. Then one can write \( L = D - C \), where \( D \) is the diagonal matrix of all vertex degrees. Note that the definition of the incidence matrix is somewhat ambiguous, since we consider undirected graphs. Therefore, each row of the incidence matrix is defined uniquely up to the sign.

For simplicity, we will restrict ourselves in the following introduction to different partitioning methods to \( p = 2 \), with partitions of \( V \) into the vertex sets \( V_1, V_2 \).

**5.1.1 Spectral Bisection Partitioning**

Spectral bisection partitioning has been motivated and introduced by Fiedler [147, 148]. In the context of sparse matrix ordering, spectral bisection has been utilized by Pothen et al. [325]. The algorithm is based on properties that can be proved for the matrices, defined in definition 5.1.4 (for the proofs, see textbooks on algebraic graph theory, e.g. [60]).
Proposition 5.1.5 (Properties of the matrices)
1. $L = I I^T$ is a symmetric matrix.
2. From 1 follows that all eigenvalues of $L$ are real: $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$. Furthermore the eigenvectors are real and orthogonal.
3. If $e = (1, \ldots, 1)^T$, then $Le = 0$.
4. For $0 \neq b \in \mathbb{R}^n$ eigenvector of $L$, i.e. $Lb = \lambda b$, $\lambda \in \mathbb{R}$, we have
   $$\lambda = \frac{\|I^T b\|^2_2}{\|b\|^2_2} = \frac{\sum_{e(v_i,v_j) \in E} (b_i - b_j)^2}{\sum_i b_i^2},$$
   where $\|b\|^2_2 = \sum_i b_i^2$ for $b \in \mathbb{R}^n$.
5. The eigenvalues of $L$ are non-negative, $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.
6. From 3 follows that $\lambda_1 = 0$. The multiplicity of the trivial eigenvalue is equal to the number of connected components of the graph.
7. If the graph is connected, then $\lambda_2 > 0$.

Note that the second smallest eigenvalue $\lambda_2$ is called algebraic connectivity of the graph $G(V,E)$.

Using these properties the algorithm of spectral bisection is simply

Algorithm 5.1.6 (Spectral bisection partitioning)
1. Compute the eigenvector $b^{(2)} = (b_1^{(2)}, \ldots, b_n^{(2)})$ corresponding to the algebraic connectivity $\lambda_2$.
2. FOR each vertex $v_i \in V$ DO:
   a) IF $b_i^{(2)} < 0$ put vertex $v_i$ in partition $V_1$.
   b) ELSE put vertex $v_i$ in partition $V_2$.
   c) END IF
3. END FOR

Fiedler proves that if $G(V,E)$ is connected and $V_1$ and $V_2$ are partitions obtained by algorithm 5.1.6 then $V_1$ is connected. Furthermore, if $b_i^{(2)} \neq 0$ for all $i = 1 : n$, then also $V_2$ is connected.

Since we have to compute a specific eigenvalue-eigenvector pair, the spectral bisection algorithm is hardly ever used for large meshes. In most cases it is combined with a multi-level approach as described in subsect. 5.1.3.

5.1.2 Kernighan-Lin Partitioning

The Kernighan-Lin heuristic minimizes the edge cut. It was motivated by electronic circuit design problems [238]. We have to introduce the edge weight first.

Definition 5.1.7 (Edge weight)
Let $e(v_i,v_j) \in E$ be an edge in the graph $G(V,E)$. Then we associate an edge weight $\omega(v_i,v_j) \in \mathbb{R}$ to the edge.
Note that with this definition, we can reformulate the edge cut (see appendix B) more formally:

**Definition 5.1.8 (Edge cut revisited)**

Let $G(V, E)$ be a graph with a vertex set $V = V_1 \cup V_2$ partitioned into two subsets. Then the cut weight $W_{V_1 \cap V_2}$ can be defined by

$$W_{V_1 \cap V_2} = \sum_{v_i \in V_1, v_j \in V_2} \omega(v_i, v_j).$$

With this and the total weight $W = \sum_{e(v_i, v_j) \in E} \omega(v_i, v_j)$, the (weighted) edge cut can be defined by

$$\mathcal{E}_\omega = \frac{W_{V_1 \cap V_2}}{W}.$$

The Kernighan-Lin heuristic tries to minimize $\mathcal{E}_\omega$, resp. $W_{V_1 \cap V_2}$ by means of a greedy algorithm.

**Algorithm 5.1.9 (Kernighan-Lin algorithm)**

1. Compute a random partition $V_1 \cup V_2 = V$.
2. **FOR** all edges $e(v_i, v_j)$ with $v_i \in V_1$ and $v_j \in V_2$ **DO**:
   a) Exchange vertices $v_i$ and $v_j$ virtually (i.e. put vertex $v_i$ into $V_2$ and $v_j$ into $V_1$).
   b) Compute new $W_{V_1 \cap V_2}$.
3. **END DO**
4. Select the pair $(v_i, v_j)$ with the lowest value for $W_{V_1 \cap V_2}$ and obtain new partition $V_1 \cup V_2 = V$.
5. **GOTO** step 2 and repeat the procedure until no further gain is obtained.

Note that the given algorithm is equally valid for graphs with unit weights, i.e. un-weighted graphs. The Kernighan-Lin algorithm has computational complexity of order $O(n^3)$, where $n = |V|$. Therefore, it is hardly used for large meshes/graphs, but most commonly combined with multi-level methods.

### 5.1.3 Multi-Level Methods

Multi-level methods follow a simple and widely used principle: Do the bulk work on a coarse sub-set of the global problem, that represents the main features, and then recursively refine the result on larger and larger sub-sets, computing only corrections to the coarse solution. The gain in efficiency results from the fact that corrections to a given (even poor) solution can often be computed in much less time than the solution itself.

We follow the original description of a multi-level graph partitioning algorithm by Hendrickson and Leland [200], also used e.g. in [233] and [352]. We first sketch the basic algorithm and then describe the single steps.
Algorithm 5.1.10 *(Multi-level graph partitioning)*

1. Start with \( k = 0 \), \( G(V^{(k)}, E^{(k)}) = G(V, K) \).
2. **WHILE** graph is too large, **DO**
   a) coarsen graph: \( k := k + 1 \), \( G(V^{(k)}, E^{(k)}) \leftarrow \text{coarsen}[G(V^{(k-1)}, E^{(k-1)})] \)
3. **END WHILE**
4. On coarsest level \( (k = k^*) \), partition graph into two partitions \( V^{(k^*)} = V_1^{(k^*)} \cup V_2^{(k^*)} \).
5. **WHILE** refined graph is not original graph, i.e. \( V^{(k)} = V_1^{(k)} \cup V_2^{(k)} \neq V \), **DO**:
   a) refine the graph, i.e. \( (k := k - 1, l = 1, 2) \ \hat{V}^{(k-1)}_l \leftarrow \text{uncoarsen}[V^{(k)}_l] \).
   b) optimize partition boundaries, \( V_l^{(k-1)} \leftarrow \text{optimize}[\hat{V}^{(k-1)}_l] \).
6. **END WHILE**

The coarsening of a graph can be achieved by first finding a *maximal matching*, that is a maximal set of edges, such that no two of them are incident on the same node. Then the nodes, corresponding to the edges in the maximal matching are collapsed. The weights of collapsed nodes are summed, while edges retain their weights, unless they are adjacent, i.e. there are two edges that connect each of the two collapsed nodes to the same third node. Then the edge weights are summed as well. In the coarsening step, each node in the fine graph is uniquely mapped to a node in the coarse graph. Therefore, a partition of the coarse graph induces a unique partition of the fine graph. Furthermore, a coarse graph has the same local properties regarding edge cut/weight and node weight as the fine (original) graph.

Since the refined partitions have more degrees of freedom (i.e. more vertices), the coarse partition might not be the optimum partition for this level of refinement. Therefore, in the refinement step, a further (local) optimization of partition boundaries improves the partition quality.

Note that this algorithm can be implemented with \( O(n) \) complexity. Furthermore, this algorithm can be parallelized as shown by Karypis and Kumar \[232, 233\]. So most of the current state of the art mesh partitioners use a multi-level approach \[199, 234, 312, 326, 403\].

**5.1.4 Space-Filling Curves**

A special space-filling curve (SFC) for triangular and tetrahedral meshes induced by bisection refinement, has been introduced by algorithm 4.2.5 in sect. 4.2. There is a large variety of space-filling curves in use. An overview of SFC is given in \[411, 429\]. Other applications of SFC in parallelizing adaptively refined meshes are given in \[51, 116, 179, 322\].

The basic algorithm consists of only 3 steps:

**Algorithm 5.1.11 *(Space-filling curve partitioning)***

1. Construct a SFC such that each graph vertex contains at least one point of the (discrete) SFC.
2. **Renumber the graph vertices along the SFC consecutively.**
3. **Partition the thus created index set into \( p \) equal sized sets.**

We are not going to introduce the diverse algorithms for creating a space-filling curve for a given graph. Sierpinsky’s curves have proven to be adequate for triangular meshes (resp. the corresponding graphs), while Hilbert curves or Morton curves proved to be advantageous for quadrilateral elements. Zimmermann shows that Hilbert’s curve can be easily extended to 3D [203, 429]. Most modern algorithms for creating space-filling curves work with recursively altering bit patterns. The creation of SFC in most cases can be accomplished very efficiently with parallelizable codes.

Note that the load balance for SFC induced partitions is almost optimal by construction, since the partition is constructed by cutting the 1D consecutive index set into equal sized sub-sets. Since SFC have a neighborhood preserving property, the partitions are connected for most types of SFC. Finally the edge cut is not optimal, however, it can be shown that the edge cut remains within a limited bounded neighborhood of the optimal edge cut [213, 430].

In fig. 5.2 we show the result of a multi-level partitioning and a SFC partitioning of a mesh, obtained from an application of tracer transport in the arctic stratosphere [39]. The SFC partitioning has been obtained by the built in SFC ordering in amatos (see sect. 4.5). The multi-level partitioning has been obtained by applying Metis, a state of the art graph partitioner [231]. A more thorough comparison of the results can be found in [45].
This chapter is concerned with the realization of differential operators, mainly on unstructured and nonuniform grids. While a finite difference approximation to a differential operator can be easily derived for orthogonal and quadrilateral grids, it is not straightforward to do the same for unstructured and non-orthogonal grids. We will focus the presentation on the gradient operator

$$\nabla \rho = \left( \frac{\partial \rho}{\partial x_1}, \ldots, \frac{\partial \rho}{\partial x_d} \right),$$

where \( x = (x_1, \ldots, x_d) \in \mathbb{R}^d \) and \( \rho : \mathbb{R}^d \rightarrow \mathbb{R} \) is a scalar function. We restrict ourselves to the gradient operator, since all of the other operators used in this book can be derived from the gradient:

$$\Delta \rho = \nabla \cdot (\nabla \rho) \quad \text{Laplace’s operator},$$
$$\text{rot}(u) = \nabla \times v \quad \text{curl operator},$$
$$\text{div}(u) = \nabla \cdot v \quad \text{divergence operator},$$

where \( \rho \) is a scalar and \( u \) a vector valued function, which is sufficiently regular to be differentiated in the above given way.

6.1 Approximating the Gradient

There are different methods to approximate a gradient operator. In this section we will introduce four different approaches to compute a discrete gradient. Note that these are all numerical gradient approximations.

- Finite difference approach;
- Least squares approximation;
- Galerkin approach;
- Radial basis function interpolatory approach.
We do not consider methods of automatic or algorithmic differentiation. For an introduction and description of current technological knowledge of automatic differentiation techniques the reader is referred to [180].

6.1.1 Finite Difference Approach

On quadrilateral and orthogonal grids, it is easy to discretize the gradient component-wise by a finite difference formula:

$$\frac{\partial x_i \rho}{\partial x_i} \approx \frac{\rho(x_i) - \rho(x_i - \Delta x_i)}{\Delta x_i}, \quad (6.2)$$

where $\Delta x_i$ is a step size in the $x_i$-coordinate direction. It is well known that this (backward) finite difference approximation is first order accurate and there are many higher order finite difference schemes available (see e.g. [288, 370]).

For unstructured grids, this approach has to be modified. There are two already quite old methods, that will be reviewed here, one proposed by Klucwicz [244] and implemented by Renka [337], the other one proposed by Akima [5]. All the mentioned references are originally concerned with interpolation of irregularly spaced data. However, the interpolation procedures rely on gradient data, thus the papers contain descriptions of gradient estimation. These methods have been described and tested for unstructured triangular grids in [33], so we only briefly introduce them here. Note that a more general approach to representing differential operators on unstructured meshes can be found in [356].

Klucwicz’s method [244] is based on a mean of all gradients in grid cells surrounding the corresponding node. So for locally adapted meshes the algorithm in two dimensions is given by

**Algorithm 6.1.1 (Gradient estimation according to Klucwicz)**

Let $x_i = (x_i, y_i) \in \mathbb{R}^2$ be the node at which a gradient is to be calculated. Let

$$I_i = \{(j, k) : j \neq k \neq i, \text{ such that } (x_i, x_j, x_k) \text{ form a triangle}\}$$

be the index set of node indices in the patch of node $x_i$.

1. Calculate the plane that represents the slope in each triangle defined by $I_i$. The plane is given by the equation

$$p_{ijk}(x, y) = \rho(x_i) + c_{ijk}^{(x)}(x - x_i) + c_{ijk}^{(y)}(y - y_i), \quad (6.3)$$

where

$$c_{ijk}^{(x)} = \frac{\partial p_{ijk}}{\partial x}, \quad c_{ijk}^{(y)} = \frac{\partial p_{ijk}}{\partial y}$$

have to be chosen such that

$$p_{ijk}(x_j, y_j) = \rho(x_j), \quad p_{ijk}(x_k, y_k) = \rho(x_k).$$
2. Form the convex combination

\[ p_i(x, y) = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} p_{ijk}(x, y) \]

with \( w_i = \sum_{j,k \in I_i} w_{ijk}, \) \( w_{ijk} \) weights.

3. The gradient is then given by

\[ \nabla \rho \big|_x (x_i) = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} c_{ijk}^{(x)}, \quad \nabla \rho \big|_y (x_i) = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} c_{ijk}^{(y)}. \]

Note that algorithm 6.1.1 is based on the finite difference approximation of the gradient components in (6.3). Since in 3D a similar representation of a hyperplane can be given, an extension to three dimensional settings is straightforward.

The basic idea behind the second method in this section is similar to the above one. It also computes means of surrounding slopes. However, the implementation differs, since it is based on the unit normal vectors to the planes representing such slopes. It turns out that this method is slightly advantageous compared to Klucwicz’s method, so it is selected for the comparison of different methods in sect. 6.2, denoted by FDM.

**Algorithm 6.1.2 (Gradient estimation according to Akima)**

Let \( x_i \) and \( I_i \) be defined as in algorithm 6.1.1.

1. For each element in the patch of node \( x_i \), defined by \( I_i \), compute the vectors

\[ \xi_j = (x_j - x_i), \quad \text{and} \quad \xi_k = (x_k - x_i). \]

Furthermore, compute the vector product \( \xi_{j,k} = \xi_j \times \xi_k \).

2. Compute the mean \( \chi \) of the \( \xi_{j,k} \) over all \( N \) patch elements

\[ \chi = \frac{1}{N} \sum_{j,k \in I_i} \xi_{j,k}. \]

3. Normalize \( \chi \) such that its third component is non-negative. \( \chi = (\chi_1, \chi_2, \chi_3) \) is perpendicular to the plane defined by the slope of \( \rho \) in node \( i \), given by the plane equation

\[ \widetilde{p_i}(x, y) = \frac{\chi_1}{\chi_3} x - \frac{\chi_2}{\chi_3} y - C, \]

where \( C \) is a constant chosen suitably. Thus, the gradient is given by

\[ \nabla \rho \big|_x (x_i) = -\frac{\chi_1}{\chi_3}, \quad \nabla \rho \big|_y (x_i) = -\frac{\chi_2}{\chi_3}. \]
6.1.2 Least Squares Approximation

In order to estimate the gradient at any position in the grid, a least squares approach seems natural. Let us consider a two-dimensional problem with a scalar function \( \rho : \mathbb{R}^2 \to \mathbb{R} \) for which the gradient \( \nabla \rho \) is to be estimated. Let \( \{(x_i, y_i), \rho_i\} \in \mathbb{R}^2 \times \mathbb{R} \) be coordinate-value pairs representing the (discrete) function \( \rho \). Then the aim is to find a plane (representing the gradient), given by the linear parametric representation

\[
p(x, y) = ax + by + c,
\]

with \((a, b, c)\) a parameter triplet. With the least squares approach, \(p\) can be found by requiring it to be the best fit with respect to the \(l^2\)-norm:

\[
\|p - \rho\|_2 = \min!
\]

This can be translated into an over-determined system of linear equations \(Ax = b\) with

\[
A = \begin{bmatrix}
x_1 & y_1 & 1 \\
x_2 & y_2 & 1 \\
\vdots & & \vdots \\
x_m & y_m & 1
\end{bmatrix}, \quad x = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad b = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_m \end{bmatrix},
\]

(6.4)

Note that we use the Vandermonde matrix \(A\), which for larger number of parameters becomes very ill conditioned. Solving this system of equations by a \(QR\)-algorithm, yields the three parameters \([a, b, c]^T\). Now, observing the definition of \(p\), we see that we obtain an approximation to the gradient by

\[
\nabla \rho \approx \nabla_h \rho := \nabla p = (a, b).
\]

A much more elaborate approach for upwind finite difference type methods for fluid flow problems on arbitrary meshes can be found in [105].

6.1.3 Galerkin Approach

A method for solving partial differential equations that was historically developed on unstructured meshes is the finite element method. We will briefly introduce the basic principle here and derive from that principle a method to compute gradients as well.

We start with the formulation of a model problem, namely Poisson’s equation on a bounded domain \(\mathcal{G} \subset \mathbb{R}^d\) with boundary \(\partial \mathcal{G}\):

\[
-\Delta \rho = f, \quad \text{in } \mathcal{G}, \quad \rho = g, \quad \text{on } \partial \mathcal{G}.
\]

(6.5)

For simplicity we will assume for the function \(g : \partial \mathcal{G} \to \mathbb{R}\) that \(g \equiv 0\), meaning that we assume homogeneous Dirichlet boundary conditions. \(\rho : \mathcal{G} \to \mathbb{R}\) is the
unknown function and \( f : \mathcal{G} \to \mathbb{R} \) a given right hand side. We assume suitable regularity for the functions involved without specifying the details here.

The variational formulation of (6.5) is given by

\[
\text{Find } \rho \in \mathcal{V} \text{ such that }
\quad a(\rho, b) = f(b), \text{ for all } b \in \mathcal{V},
\]

with \( a \) a bilinear form and \( f \) a linear form defined below, \( b \in \mathcal{V} \) a test function or basis function and \( \mathcal{V} = H_0^1(\mathcal{G}) \) a suitably chosen function space, in this case the Sobolev space of compactly supported integrable functions.

\[
a(\rho, b) = \int_{\mathcal{G}} \nabla \rho \cdot \nabla b \, dx,
\]
\[
f(b) = \int_{\mathcal{G}} f b \, dx.
\]

The derivation of the variational form, Sobolev spaces, and existence and convergence theorems can be found in common finite element books (see e.g. [2, 71]).

The \textit{Galerkin approach} to solving (6.6) introduces suitable basis functions \( b_i \in \mathcal{V}_h \) such that \( \rho \) can be represented by the expansion

\[
\rho(x) = \sum_{i=1}^{N} \rho_i b_i(x),
\]

where \( \rho_i \in \mathbb{R} \) are unique coefficients. In fact, \( \text{span}\{b_1, \ldots, b_N\} = \mathcal{V}_h \), which means that the set \( \{b_1, \ldots, b_N\} \) forms an \( N \)-dimensional basis of \( \mathcal{V}_h \). Furthermore we denote by \( \mathcal{V}_h \) the corresponding finite dimensional approximation to \( \mathcal{V} \). Since \( \mathcal{V}_h \subset \mathcal{V} \) we can now formulate the discrete form of the variational equation (6.6):

\[
\text{Find coefficients } (\rho_1, \ldots, \rho_N) \in \mathbb{R}^N \text{ such that }
\quad \sum_i a(\rho_i b_i, b_j) = f(b_j), \text{ for } i, j = 1 : N.
\]

\( a \) and \( f \) are taken as defined above in the non-discrete formulation. Note that since the \( b_i \) form a basis of \( \mathcal{V}_h \), we only have to test (6.8) over a finite number of basis functions. In other words, we have formulated a system of equations

\[
A_h \rho_h = f_h,
\]

with

\[
A_h = \begin{bmatrix}
  a(b_1, b_1) & \cdots & a(b_1, b_N) \\
  \vdots & \ddots & \vdots \\
  a(b_N, b_1) & \cdots & a(b_N, b_N)
\end{bmatrix},
\quad \rho_h = \begin{bmatrix}
  \rho_1 \\
  \vdots \\
  \rho_N
\end{bmatrix},
\quad f_h = \begin{bmatrix}
  f(b_1) \\
  \vdots \\
  f(b_N)
\end{bmatrix}.
\]
Fig. 6.1. Piecewise linear basis function $b_i$

The finite element method (FEM) constructs basis functions $b_i$ such that $A$ becomes a sparse matrix, by defining $b_i$ only on a local (and small) support. This is done by introducing a triangulation of the domain $\mathcal{G}$, to obtain a polygonal approximation $\mathcal{G}_h$. Then, basis functions are defined piecewise on each element. Conforming FEM require continuity from element to element, while non-conforming FEM allow for discontinuities along cell edges. The simplest conforming finite element that can be used for solving Poisson’s equation and which is continuously differentiable, consists of piecewise linear basis functions. Let $x_i, i = 1 : N$ be nodes of a triangulation. Then we define

$$b_i(x_j) = \delta_{ij},$$

where $\delta_{ij}$ is the discrete Dirac delta function, and linear in between the nodes. A sketch of this hat function is depicted in fig. 6.1.

Depending on the problem, the system of equations may be nonlinear. In the case of the above formulated model problem (Poisson’s equation (6.5)) the system is linear.

To summarize this very brief outline of the finite element method, the following steps are required to realize a discretization of an equation of the form

$$\mathcal{D}(\rho) = r,$$

where $\mathcal{D}$ denotes a differential operator, $\rho$ is the unknown function and $r$ a known right hand side.

1. Triangulate the domain $\mathcal{G}$ with a suitable triangulation to obtain $\mathcal{G}_h$.
2. Choose appropriate function spaces $\mathcal{V}$ and $\mathcal{V}_h$ to represent the differential operator together with boundary and initial conditions in continuous and discrete space.
3. Define basis functions $b_i \in \mathcal{V}_h$ such that the discretization matrix $A_h$ becomes sparse. In fact there are more requirements on the basis functions, like preservation of characteristic properties of the equation, that might influence the choice of the basis.
4. Set up the system of equations, defined by the bilinear form $a$ and the linear form $f$ and solve it.
5. Reconstruct the function $\rho$ by the Galerkin expansion (6.7).

Once the solution vector of coefficients $(\rho_1, \ldots, \rho_N)^T$ has been computed, the function $\rho$ can be reconstructed by means of the expansion (6.7). If we look for the gradient $\nabla \rho$ of $\rho$, we can use the expansion again. Apply the $\nabla$-operator on both sides of (6.7) to get

$$\nabla \rho(x) = \sum_{i=1:N} \rho_i \nabla b_i. \quad (6.9)$$

When computing $\nabla \rho$ at nodes, based on piecewise functions, we face a problem. For example, the piecewise linear basis functions defined above and depicted in fig. 6.1 are not differentiable at cell interfaces and especially not at nodes. To solve this problem, one could define higher order basis functions with higher regularity conditions along cell interfaces. Another possibility is to compute a weighted sum or mean of local gradients:

$$\nabla \rho(x_i) = \frac{1}{n} \sum_{\tau_j \in I_i} |\tau_j| \cdot \nabla \rho|_{\tau_j}(x_i),$$

where $n = \sum_{\tau_j \in I_i} |\tau_j|$ is the area of the patch corresponding to node $x_i$, defined by $I_i$ (see sect. 6.1.1), $\tau_j$ are the cells, and $|\cdot|$ is the area of a cell.

### 6.1.4 Radial Basis Function Approach

Interpolatory methods for deriving the gradient operator on unstructured grids work according to the following simple idea:

1. Given a set of nodes $\{x_1, \ldots, x_l\} \subset \mathbb{R}^d$ and a set of function values corresponding to the nodes, $\{\rho_1, \ldots, \rho_l\}$, find a polynomial interpolation function $p: \mathbb{R}^d \rightarrow \mathbb{R}$ such that

$$p(x_i) = \rho(x_i), \quad \text{for all } i = 1 : l.$$

2. Since $p$ is a polynomial, it is easy to derive the analytical partial derivative $\nabla p$.

3. Then set

$$\nabla_h \rho := \nabla p.$$

This principle works with all kinds of polynomials $p$, but an especially interesting class of polynomials for locally unstructured grids and therefore irregularly distributed nodes $x_i$ are the radial basis functions. This approach dates back to Hardy [189] and Micchelli [286]. The interpolating function $p$ is formed by one-dimensional basis functions that are radially symmetric. Thus, an ansatz

$$p(x) = \sum_{i=1:l} \lambda_i \varphi(\|x - x_i\|) \quad (6.10)$$
can be chosen, where \( \lambda_i \) are coefficients to be determined from the interpolation problem and \( \varphi : \mathbb{R} \to \mathbb{R} \) is a sufficiently smooth basis function. Together with the interpolation problem
\[
p(x_i) = \rho_i, \quad \text{for all } i = 1 : l
\]
We have to solve a \( l \times l \) linear system of equations \( A\lambda = \rho \). Denoting \( \varphi_{ij} = \varphi(||x_i - x_j||) \) we have
\[
A = \begin{bmatrix}
\varphi_{11} & \cdots & \varphi_{1l} \\
\vdots & \ddots & \vdots \\
\varphi_{l1} & \cdots & \varphi_{ll}
\end{bmatrix}, \quad \lambda = \begin{bmatrix}
\lambda_1 \\
\vdots \\
\lambda_l
\end{bmatrix}, \quad \rho = \begin{bmatrix}
\rho_1 \\
\vdots \\
\rho_l
\end{bmatrix}.
\]
Depending on the particular basis function, sometimes we have to add a polynomial part to the ansatz (6.10), in order to be able to invert \( A \) in (6.11). This means, we define \( p \) by
\[
p(x) = \sum_{i=1}^{l} \lambda_i \varphi(||x - x_i||) + q(x).
\]
To be solvable, we restrict the coefficients to fulfill
\[
\sum_{i=1}^{l} \lambda_i p(x_i) = 0.
\]
From the interpolation problem and the above restriction it follows, that we now solve a blocked system \( B\Lambda = \Xi \), where
\[
B = \begin{bmatrix}
A & P \\
P^T & 0
\end{bmatrix}, \quad \Lambda = \begin{bmatrix}
\lambda \\
\mu
\end{bmatrix}, \quad \Xi = \begin{bmatrix}
\rho \\
0
\end{bmatrix},
\]
where \( \mu \) is the vector of coefficients or the polynomial \( q \),
\[
q(x) = \sum_{i=1}^{r} \mu_i x^{i-1},
\]
and \( P \) is formed by evaluations of \( q(x_i) \). Table 6.1 lists some common radial basis functions together with the order \( r \) of polynomial \( q \). The algorithm for computing the gradient operator at any given point is finally given by

**Algorithm 6.1.3** (Radial basis function interpolatory gradient calculation)

Let \( x \) be a given coordinate, \( \{x_1, \ldots, x_l\} \) be the set of nearest mesh points with corresponding values \( \rho_i, i = 1 : l \).

1. Evaluate \( \varphi(||x - x_i||) \) for \( i = 1 : l \) in order to derive the coefficients of \( A \), and evaluate \( q(x_i) \) for setting up \( P \), if necessary.
2. Solve the linear system of equations given by either (6.11) or (6.12).
Table 6.1. Common radial basis functions with their corresponding polynomial expansion order

<table>
<thead>
<tr>
<th>name</th>
<th>formula</th>
<th>order of $q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>thin plate splines</td>
<td>$\varphi(r) = r^2 \log r$</td>
<td>$r = 2$</td>
</tr>
<tr>
<td>multi-quadrics</td>
<td>$\varphi(r) = \sqrt{(r^2 + 1)}$</td>
<td>$r = 1$</td>
</tr>
<tr>
<td>Gaussians</td>
<td>$\varphi(r) = \exp(-r^2)$</td>
<td>$r = 0$</td>
</tr>
<tr>
<td>inverse multi-quadrics</td>
<td>$\varphi(r) = (r^2 + 1)^{-1/2}$</td>
<td>$r = 0$</td>
</tr>
</tbody>
</table>

3. Evaluate $\nabla p(x)$, where

$$\nabla p = \sum \varphi' + \nabla q.$$ 

Note that we have to solve a (small) system of equations for each set of neighboring grid points $\{x_1, \ldots, x_l\}$. Since the radial basis functions are only dependent on the distance of mesh point $x_i$ to the given coordinate $x$, there is no mesh required in order for the calculation to be well defined. For more details on radial basis functions in general, the reader is referred to [75]. Fornberg and Flyer study the accuracy of radial basis functions for derivative estimation [153]. For radial basis functions in advection problems, see [40, 41]. Hubbert has studied interpolation problems by radial basis function approaches on the sphere [208, 209]. Another study of scattered data interpolation on spheres can be found in [299]. A compactly supported RBF approach is taken to solve the shallow water equations in [417].

6.2 Evaluating the Gradient Approximation

In order to evaluate the approximations derived in the previous section, we utilize a two-dimensional test case. Let

$$\rho(x, y) = \cos\left(2\pi(x - \frac{1}{4})\right) \cos\left(2\pi(y - \frac{1}{4})\right)$$

Fig. 6.2. The basic test problem for gradient computation: (left) the scalar function $\rho$, (center) $\nabla \rho |_x$, and (right) $\nabla \rho |_y$
Fig. 6.3. Convergence of the gradient approximation methods. The dotted lines represent third and fourth order ideal convergence rates.

be given on the two-dimensional unit square \([0, 1]^2 \subseteq \mathbb{R}^2\). A plot of this function together with the two gradient components is given in fig. 6.2. The gradient of this function can be given analytically by

\[
\begin{align*}
\nabla \rho|_x &= -2\pi \sin \left( 2\pi (x - \frac{1}{4}) \right) \cos \left( 2\pi (y - \frac{1}{4}) \right), \\
\nabla \rho|_y &= -2\pi \cos \left( 2\pi (x - \frac{1}{4}) \right) \sin \left( 2\pi (y - \frac{1}{4}) \right).
\end{align*}
\]

In order to evaluate some of the methods, described in the previous sections, we compare the numerically computed gradient \(\nabla_h \rho\) with the exact one, \(\nabla \rho\) in the \(l^2\)-norm (\(N\) the number of grid points):

\[
\varepsilon_{l^2} = \|\nabla_h \rho - \nabla \rho\|_2 = \frac{1}{N} \left( \sum_{i=1:N} \left( \nabla_h \rho(x_i, y_i) - \nabla \rho(x_i, y_i) \right)^2 \right)^{\frac{1}{2}}.
\]

Figure 6.3 shows the convergence behavior of the different methods. We denote the thin plate spline interpolatory gradient approximation by \(TPS\), the finite difference based method by \(FDM\), a linear finite element method based Galerkin approach by \(LOW\), and the least squares based method by \(LSQ\). Figure 6.3 shows that the LSQ method on a uniform grid performs best, together with the TPS method.

When looking at the difference plots (fig. 6.4) we see that the largest (and only visible) error occurs at the boundary of the domain. This is not surprising, since all methods rely on collecting surrounding node/element values in order to approximate the gradient operator. Since at the boundary, only node/element values on one side of the corresponding position exist, accuracy drops drastically.

Next, we investigate the dependency of the corresponding method to local irregularities in the mesh. It is of great importance to keep these effects limited,
6.2 Evaluating the Gradient Approximation

Fig. 6.4. Error plots for four different gradient approximation methods on a uniform mesh (CNT[15]), corresponding to $h \approx 5.5 \cdot 10^{-3}$. From left to right: TPS, FDM, LOW, LSQ. Differences are hardly visible and are limited to the boundary.

Fig. 6.5. Dependence on non-uniformity of the mesh: (left) a locally refined mesh (the CNT[11:17] case); (right) $l^2$-norm of the error for cases CNT[11:17], CNT[13:17], CNT[15:17], resp. (mesh ratio is given).

Fig. 6.6. Error plots for four different gradient approximation methods (like fig. 6.4) on a adaptively refined mesh (see fig. 6.5 for the mesh).

as shown e.g. in [131, 184]. In fact, all methods show dependence on the mesh geometry. Surprisingly the TPS method is affected most. Looking at the difference plots (fig. 6.6), we see that the largest error occurs at grid interfaces.

Finally, the computational effort has to be considered. The TPS method, although not numerically exceptionally advantageous, is by a factor of 4 to 5 more costly than the LOW method, while the LSQ method is a factor of 1.5 to 2 more expensive than the LOW method. However, all three methods show
perfect linear order of computational complexity, i.e. for increasing the number of mesh points, the computational cost grows linearly with this number.

### 6.2.1 Further References

For a better representation of gradients on unstructured meshes, it might be helpful to look at different approaches. Abgrall and Harten introduce multi-level approaches to represent multi-scale functions on unstructured meshes [1]. An old and successful method is the combination of finite element theory with least squares approximation [149]. Since the thin plate spline radial basis function approach in our examples did not perform too satisfactorily, one could think of using other basis functions that are better suited for the given data [75, 217]. Radial basis functions have been used successfully to recover vector valued functions [29]. A rapid evaluation method that further minimizes the effort of solving a dense system of equations for each set of interpolation points can be found in [348]. A stencil based approach that considers data locality and is suitable for parallelization is proposed in [59] and in a different way in [133]. Interpolatory methods might profit from insight in the spatial error of the underlying interpolation method, which is investigated in [79]. Kirby and Karniadakis introduce de-aliasing techniques in order to improve results of continuous and discontinuous Galerkin methods on non-uniform grids [241]. A stable variational approach for numerical differentiation is presented in [245].
Many of the basic equations in atmospheric modeling are based on conservation laws. Conservation of mass constitutes the continuity equation, and conservation of momentum establishes the momentum equations. When conservation properties are present in the continuous equations, the numerical (discrete) counterparts should also have conservative properties. Examples for numerical conservation of vorticity or other state variables can be found in [46, 350]. More generally we want a numerical method to adhere to a structure preservation property. To achieve conservation is paramount for all kinds of numerical methods that try to discretize conservation laws. However, for adaptive methods this often poses an additional challenge.

In this chapter we first introduce some of the basic mathematical tools like the divergence theorem and others. We will then deal with deriving the basic systems of equations used in atmospheric modeling from conservation principles. Once these equations have been established, we present several methods to solve these. Our main emphasis will be on semi-Lagrangian methods in sect. 7.4. Two other sections are devoted to the finite volume method (sect. 7.2) and the discontinuous Galerkin method (sect. 7.3).

There are of course many more approaches to achieve conservation in adaptive flow problems. A survey of (mostly geosciences related) advection methods can be found in [141]. Mixed-hybrid finite element methods achieve conservation for fluid flow. An overview can be found in [145]. For a projection method in the context of adaptive mesh refinement the reader is referred to [6]. A modified method of characteristics Galerkin scheme with local conservation properties has been proposed in [10]. Spectral methods for hyperbolic conservation laws are reviewed in [175]. A comparison of different finite element methods for solving the shallow water equations is given in [188]. High order finite differences are employed in [420], while a positive advection method for tracer transport on unstructured grids is introduced in [413]. An investigation of conservation of smoothed particle hydrodynamics for the shallow water equations can be found in [155].
In this work we will not go into the very details of geophysical fluid flow. However, most geophysical flow problems comprise multiple time scales. In many cases, fast modes are not considered or filtered in order to maintain numerical stability. A comparison of several such methods, is given in [27]. More generally an overview of unstructured grid techniques in CFD can be found in [278]. In sect. 7.4, we consider semi-implicit methods. For further evaluation of implicit schemes see [48, 49, 50, 261, 293, 335]. In order to integrate (semi-) implicit schemes, large systems of equations have to be solved. We will not consider to present solution techniques here, since this would go beyond the scope of this book. However, we refer to [65, 66, 68, 82, 127, 144, 156, 202, 239, 422] among others.

### 7.1 Conservation Laws of Interest

In this section we derive some of the common equations of atmospheric modeling from basic conservation laws. Especially in climate modeling, it is essential to preserve these conservation properties in the discretization. Our presentation will be restricted to conservation of mass, momentum, and energy and we will restrict ourselves to the very basic principles.

It should be noted that even if we pretend momentum to be a conservation quantity, the equation derived in subsect. 7.1.2 strictly speaking is not a conservation principle. Additionally, in meteorological applications the conservation of angular momentum is often much more important and can be stated as a strict conservation principle. We refer the reader to the dynamical meteorology literature for a more comprehensive presentation of the derivation of the main equations [134, 165, 311].

#### 7.1.1 Conservation of Mass

The most basic conservation principle is conservation of mass. We are going to derive the continuity equation from this principle in a one-dimensional setting. This example corresponds to fluid flow through a tube, where we assume the fluid to be constant across each cross section of the tube. Let $x$ represent the space coordinate, and $t$ the time. $\rho = \rho(x, t)$ is the density of the fluid at time $t$ and position $x$. Now, the fluid mass within a volume $V$ (given by the two spatial points $x_1$ and $x_2$, say $V = [x_1, x_2]$) at a time $t$, denoted by $M_{V,t}$, is given by the integral

$$M_{V,t} = \int_V \rho(x, t) \, dx = \int_{x_1}^{x_2} \rho(x, t) \, dx.$$

If we assume that mass cannot enter through the tube walls (i.e. there is no mass production or destruction), then mass within a given volume $V$ can only change due to fluid flow. Let $v = v(x, t)$ be the velocity of the fluid at time $t$
7.1 Conservation Laws of Interest

and position \( x \). The flow rate (or flux), denoted by \( f = f(x, t) \) of fluid past a given position \( x \) at time \( t \) is given by

\[
f(x, t) = \rho(x, t)v(x, t).
\]

Thus, the rate of change in a volume \( V \) is

\[
\frac{dM_{V,t}}{dt} = \frac{d}{dt} \int_{x_1}^{x_2} \rho(x, t) \, dx = \rho(x_2, t)v(x_2, t) - \rho(x_1, t)v(x_1, t).
\]

Integration of both sides of the above equation over a time interval \([t_1, t_2]\) gives an expression of the mass evolution in terms of a pre-existing mass and an amount of mass flowing in and out of the volume at both boundaries:

\[
\int_{x_1}^{x_2} \rho(x, t_2) \, dx = \int_{x_1}^{x_2} \rho(x, t_1) \, dx + \int_{t_1}^{t_2} \rho(x_2, t)v(x_2, t) \, dt - \int_{t_1}^{t_2} \rho(x_1, t)v(x_1, t) \, dt.
\]

In order to derive the differential form of the conservation of mass, we assume that both \( \rho \) and \( v \) are differentiable. Then, with the relations

\[
\rho(x, t_2) - \rho(x, t_1) = \int_{t_1}^{t_2} \frac{\partial \rho}{\partial t}(x, t) \, dt, \quad \text{and} \quad \rho(x_2, t)v(x_2, t) - \rho(x_1, t)v(x_1, t) = \int_{x_1}^{x_2} \frac{\partial}{\partial x}(\rho(x, t)v(x, t)) \, dx,
\]

we derive

\[
\int_{t_1}^{t_2} \int_{x_1}^{x_2} \left[ \frac{\partial \rho(x, t)}{\partial t} + \frac{\partial}{\partial x}(\rho(x, t)v(x, t)) \right] \, dx \, dt = 0.
\]

Since this holds for arbitrary \( x_i \) and \( t_j \) \((i, j = 1, 2)\) the differential form of the conservation of mass reads

\[
\partial_t \rho + \partial_x (\rho v) = 0. \tag{7.1}
\]

For higher dimensions the conservation of mass can be derived in an analogous way yielding

\[
\partial_t \rho + \nabla \cdot (\rho v) = 0, \tag{7.2}
\]

where \( \mathbf{v} \) is a vector valued velocity now.

Note that with the transport theorem A.0.5, we could have derived (7.2) in a different way. Assuming that mass in a material volume is conserved over time, we can formulate the following equation:

\[
\frac{d}{dt} \int_{V(t)} \rho \, dx = 0.
\]

It states that the rate of change of mass in \( V(t) \) is zero. Now, applying the transport theorem yields
\[
\int_{V(t)} \left( \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) \right) \, dx = 0.
\]

Since the above integral equation is true for any material volume \( V(t) \), (7.2) follows.

**Remark 7.1.1 (Mass conservation in incompressible flow)**

An incompressible flow is defined by the property that the density of each material particle remains constant over time:

\[ \rho(x_{0}, t_{0}, t) = \rho(x_{0}, t_{0}). \]

In other words, \( \frac{d\rho}{dt} = 0 \). From this and (7.2) it follows that \( \nabla \cdot \mathbf{v} = 0 \), since

\[ \nabla \cdot (\rho \mathbf{v}) = \rho \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla \rho. \]

It should be noted that this incompressibility definition does not contradict the usual definition by the statement of a solenoidal velocity field (i.e. \( \nabla \cdot \mathbf{v} = 0 \)). In oceanography, this is often used as the definition of incompressibility and density is taken as an equation of state (e.g. \( \rho = \rho(S, T, P) \), where \( S \) is salinity, \( T \) is temperature, and \( P \) is pressure, see sect. 7.1.4). Here, we claimed that density in each material particle is constant!

Another, often more useful alternative formulation of (7.2) in terms of the material derivative is given by

\[
\frac{d\rho}{dt} + \rho \nabla \cdot \mathbf{v} = 0. \tag{7.3}
\]

**7.1.2 Conservation of Momentum**

Newton’s law of conservation of momentum (\( \mathbf{vm}, m \) the mass) states that the rate of change of momentum of an object is balanced by the sum of all forces on it. This can be generalized to fluids whose momentum is given as the integral of velocity times density in an (infinitesimal) material volume. In mathematical terms this relation is reflected in

\[
\frac{d}{dt} \int_{V(t)} \rho \mathbf{v} \, dx = \int_{V(t)} \mathcal{F}_{\text{body}} \rho \, dx + \int_{S(t)} \mathcal{F}_{\text{surf}} \, ds. \tag{7.4}
\]

The first integral on the right hand side of (7.4) represents the internal (body) forces in the material volume \( V(t) \), while the second integral represents the surface forces on the surface \( S(t) \) of the material volume.

We can apply the transport theorem A.0.5 to the left hand side of (7.4) which gives us:

\[
\int_{V(t)} \left( \frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot [(\rho \mathbf{v}) \mathbf{v}] \right) \, dx = \int_{V(t)} \mathcal{F}_{\text{body}} \rho \, dx + \int_{S(t)} \mathcal{F}_{\text{surf}} \, ds.
\]
Now, assuming that the surface forces can be represented by means of the stress tensor $\sigma_{\text{surf}}$ by

$$F_{\text{surf}} = \sigma_{\text{surf}} n,$$

where $n$ is the outward unit normal on $ds$, and applying a scalar version of the divergence theorem A.0.1, we obtain

$$\int_{V(t)} \left( \rho \frac{\partial (\rho v)}{\partial t} + \nabla \cdot \left( (\rho v)^2 \right) \right) \, dx = \int_{V(t)} (F_{\text{body}} \rho + \nabla \sigma_{\text{surf}}) \, dx.$$

Since this equation holds for arbitrary $V(t)$, we conclude with the differential form of the conservation of momentum:

$$\frac{\partial (\rho v)}{\partial t} + \nabla \cdot \left( (\rho v)^2 \right) - F_{\text{body}} \rho - \nabla \sigma_{\text{surf}} = 0. \quad (7.5)$$

In order to obtain a formulation with respect to the material derivative, we assume incompressibility and perform some more elementary transformations to obtain

$$\rho \frac{dv}{dt} - F_{\text{body}} \rho - \nabla \sigma_{\text{surf}} = 0. \quad (7.6)$$

Assuming that that we can restrict ourselves to the following three force terms

- the pressure gradient $\nabla p$,
- the gravity acting on density as a potential $\rho g$,
- frictional forces summarized by a term $F = F(\mu, v)$,

where $\mu$ is the molecular viscosity, we obtain the momentum equation in analogy to [311]:

$$\rho \frac{dv}{dt} + \nabla p - \rho g - F(\mu, v) = 0. \quad (7.7)$$

Note that often not a molecular viscosity is assumed but an eddy viscosity, assuming certain scale dependencies. Furthermore, we have assumed that the fluid is a Newtonian fluid (see e.g. [410] for a definition of Newtonian fluids; water and air can be considered Newtonian).

It should also be noted that equation (7.7) does not strictly state a conservation principle, since the force terms lead to (at least locally) diminishing momentum. Thus it is probably more correct to call it a “balance equation”.

### 7.1.3 Conservation of Energy

Internal energy $e$, like density $\rho$ or pressure $p$, is a state variable of a fluid flow system. The first law of thermodynamics states that the total energy $E$ of a closed system is balanced by the the work applied to the system and the heat added to it. Thus, total energy is the sum of kinetic energy, potential energy, and internal energy. The kinetic energy per unit mass of a material particle is
\( \frac{1}{2} v^2 \), where \( v \) is the velocity. In order to derive a conservation law for energy, we apply the first law of thermodynamics to a material volume \( V(t) \):

\[
\frac{d}{dt} \int_{V(t)} \rho E \, dx = W + Q, \tag{7.8}
\]

where \( W \) is the rate of work acting on the fluid and \( Q \) is the rate of heat added. Body force works at a rate of \( v \cdot F_{\text{body}} \rho \, dx \) and the surface force at a rate of \( v \cdot F_{\text{surf}} \, ds \), which gives

\[
W = \int_{V(t)} v \cdot F_{\text{body}} \rho \, dx + \int_{S(t)} v \cdot F_{\text{surf}} \, ds.
\]

Using the same argument as in the derivation of momentum conservation in sect. 7.1.2, we obtain

\[
W = \int_{V(t)} (\rho v \cdot F_{\text{body}} + \nabla (v \cdot \sigma_{\text{surf}})) \, dx.
\]

We assume a heating rate of \( q \) per unit of mass on each material particle and a heat flux of \( \varsigma \) over the surface \( S(t) \). Then

\[
Q = \int_{V(t)} \rho q \, dx + \int_{S(t)} \varsigma \, ds.
\]

Further on, we assume that heat diffusion is governed by Fourier’s law, i.e.

\[
\varsigma = k n \cdot \nabla T,
\]

where \( k \) is the thermal conductivity and \( T \) the temperature. Using this and the divergence theorem, we obtain

\[
Q = \int_{V(t)} (\rho q + \nabla (k \nabla T)) \, dx.
\]

Taking the above results, substituting into (7.8) and applying the transport theorem we get

\[
\int_{V(t)} \left( \frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho v E) \right) \, dx = \int_{V(t)} (\rho v \cdot F_{\text{body}} + \rho q + \nabla (v \cdot \sigma_{\text{surf}})
+ \nabla (k \nabla T)) \, dx.
\]

Since this is true for arbitrary \( V(t) \), we derived the energy conservation equation

\[
\partial_t (\rho E) + \nabla \cdot [(\rho E)v] = \rho v \cdot F_{\text{body}} + \rho q + \nabla (v \cdot \sigma_{\text{surf}}) + \nabla (k \nabla T). \tag{7.9}
\]

A more suitable form of the energy equation that can be derived by using the mass conservation equation (7.2) and some more elemental manipulations
This yields an equation for the internal energy, which we give in a formulation involving a material derivative for $e$:

$$\rho \frac{de}{dt} = -p \nabla \cdot \mathbf{v} + \rho q + \nabla (k \nabla T) + \mathcal{H}. \quad (7.10)$$

The left hand side represents the advective energy transport which is balanced by the terms on the right hand side, which are

- the work done by pressure, $p \nabla \cdot \mathbf{v}$,
- external heating, $\rho q$,
- diffusive energy transport, $\nabla (k \nabla T)$, and
- dissipative (frictional) heating $\mathcal{H}$, which is non-negative.

### 7.1.4 Equations of State – Closing the System

In the above sections, we derived a set of five equations:

1. conservation of mass (7.3), involving the density $\rho$ and velocity $\mathbf{v}$;
2. conservation of momentum (7.7) for the three velocity components of $\mathbf{v} = (u, v, w)$, involving $\mathbf{v}$, $\rho$, and $p$;
3. conservation of energy (7.10), involving $e$, $\mathbf{v}$, $T$, $\rho$, and $p$.

Summarizing, we have seven state variables $u$, $v$, $w$, $\rho$, $p$, $e$, $T$ but only five equations. Thus, in order to close the system, we need two more equations, the **equations of state**.

The perfect gas equation of state establishes a relation between pressure and density/temperature by

$$p = \rho RT, \quad (7.11)$$

where $R$ is a constant specific to the individual gas. It is given by

$$R = \frac{R_g}{\mu m_u N_A},$$

where $R_g = 8.3143 \text{ JK}^{-1} \text{ mol}^{-1}$ is the gas constant, $N_A = 6.022 \cdot 10^{23}$ mol$^{-1}$ is Avogadro’s number, $m_u = 1.66 \cdot 10^{-27}$ kg is the atomic mass unit and $\mu$ is the mean molecular weight. For dry air we have $R = 287 \text{ JK}^{-1} \text{ kg}^{-1}$.

For a perfect gas, the internal energy depends only on temperature, therefore we obtain a second equation of state

$$e = e(T). \quad (7.12)$$

The derivation of conservation laws and equations of state so far are valid for any fluid dynamics application in a non-rotating frame of reference. We have not specified any specialities for geophysical fluid dynamics or atmospheric fluid flow, and in particular, we did not consider the flow on a rotating sphere. We will postpone this to the appendix, and will proceed by simplifying the equations by certain assumptions.
If we assume that the relation of $\rho$ and $p$ can be described by a function $\psi$ in the following sense:

$$\frac{1}{\rho} \delta p = \delta \psi,$$

where $\delta(\cdot)$ symbolizes a small variation in the corresponding function, then a flow is called **barotropic**. A common assumption is that $p = p(\rho)$ is a function of density alone, or that $\rho = c = \text{const.}$ in which case $\psi = c^{-1} p$.

Introducing specific entropy $s$, defined by $T \frac{ds}{dt} = \frac{de}{dt} + p \frac{d}{dt} \frac{1}{\rho}$ and assuming equations of state of the form $\rho = \rho(p,T), s = s(p,T)$, Pedlosky [311] derives several alternatives to the energy equation (7.10). For ideal gases the potential temperature equation is

$$\frac{d\Theta}{dt} = \Theta \frac{C_p}{T} \left( \frac{1}{\rho} \nabla(k \nabla T) + q \right),$$

with $\Theta$ the **potential temperature**, defined by $\Theta = T \left( \frac{p_0}{p} \right)^{R/C_p}$, $p_0$ a reference pressure, and $C_p$ the specific heat at constant pressure. Note that in the absence of internal and conductive heating (i.e. $q = 0$ and $\nabla(k \nabla T) = 0$), potential temperature is a conserved quantity for each material particle.

Another convenient form of the first thermodynamic law can be obtained from the assumption that $\rho$ can be expressed as

$$\rho = \rho_0(1 - \alpha(T - T_0)).$$

Then the heat equation takes the form

$$\frac{dT}{dt} = \kappa \Delta T + \frac{q}{C_p},$$

where $\kappa = \frac{k}{\rho C_p}$ is the coefficient of thermal diffusivity.

### 7.1.5 Deriving the Shallow Water Equations

In global atmospheric and oceanic modeling, the principal assumption that the typical vertical length scale is small compared to the typical horizontal length scale is valid. A systematic scale analysis can be found in [311]. We will derive the shallow water equations along that line.

We consider the fluid with constant and uniform density, $\rho = \text{const.}$, and let the fluid be inviscid, which means that viscosity $\mu$ is negligible or in other words there are no internal frictional forces and $\mathcal{F}(\mu, \mathbf{v})$ vanishes. The height of the surface is given by $h = h(x, y, t)$, and the rigid bottom is given by $h_b = h_b(x, y)$. The body force from a potential $\psi$ is modeled as a vector $g$.
perpendicular to the horizontal \((x, y)\)-plane, i.e. antiparallel to the \(z\)-axis. See fig. 7.1 for a sketch of the set up.

The specification of uniform and constant density leads to the incompressibility condition. This follows from mass conservation (7.2):

\[
\nabla \cdot \mathbf{v} = 0.
\]

Furthermore, the thermodynamics is decoupled from the system, leaving only the mass conservation and momentum equations.

Let \(D\) be the typical scale for the water depth (or the vertical length scale), while \(L\) be the typical horizontal length scale. Then the above principal assumption is given by

\[
\delta = \frac{D}{L} \ll 1.
\]

With the above assumptions and a rigorous scale analysis, not repeated here, one sees that the vertical velocity is at most of the order of \(W \leq \mathcal{O}(\delta U)\), where \(W\) is the typical vertical and \(U\) the typical horizontal scale of velocity. Furthermore, the vertical pressure variation \(\hat{p}\) can be neglected, where we wrote the total pressure \(p = p(x, y, z, t) = -\rho gz + \hat{p}(x, y, z, t)\). This also yields the hydrostatic assumption (to order \(\mathcal{O}(\delta^2)\) accuracy)

\[
\frac{\partial p}{\partial z} = -\rho g.
\]

Assuming a constant surface pressure \(p_0\) one can write \(p = \rho g(h - z) + p_0\). The independence of the horizontal components of the pressure gradient from \(z\) gives

\[
\frac{\partial p}{\partial x} = \rho g \frac{\partial h}{\partial x}, \quad \frac{\partial p}{\partial y} = \rho g \frac{\partial h}{\partial y}.
\]

One can reason that the horizontal velocities have to be independent of \(z\). Therefore, the horizontal momentum equations (in component form) become

\[
\partial_t u + u \partial_x u + v \partial_y u + g \partial_z h = 0, \quad (7.13)
\]

\[
\partial_t v + u \partial_x v + v \partial_y v + g \partial_y h = 0. \quad (7.14)
\]
Using the $z$-independence of $u$ and $v$, integrating the incompressibility condition above in $z$ and taking the boundary conditions for $w$ at the bottom $z = h_b$ and at the surface $z = h$ yields

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x} [(h - h_b)u] + \frac{\partial}{\partial y} [(h - h_b)v] = 0.$$  

Defining a total depth $H = h - h_b$ one obtains

$$\frac{\partial t}{\partial t} H + \frac{\partial}{\partial x} (Hu) + \frac{\partial}{\partial y} (Hv) = 0,$$

which together with (7.13) and (7.14) form the shallow water equations.

Using the material derivative and using vector notation $\xi_H = (\xi|_x, \xi|_y)$, with $\xi|_x$ and $\xi|_y$ the $x$ and $y$ components of $\xi_H$ resp., for the horizontal components, one can rewrite the system to the following form

$$\frac{d\mathbf{v}_H}{dt} + g \nabla H h_H = 0,$$

$$\frac{dH}{dt} + H \nabla H \cdot \mathbf{v}_H = 0.$$

In global atmospheric modeling, the rotating shallow water equations in spherical geometry are often used for testing numerical schemes. A standard set of test cases and a collection of formulations of shallow water equations can be found in [415]. We list some additional forms in the appendix C. We did not report on the full 3-dimensional set of equations that are usually used in numerical models of the atmosphere. We refer to the standard literature (e.g. [311]). Some 3D adaptive model formulations can be found in [20, 221, 308].

### 7.2 Finite Volume Methods

We are now going to explore methods for discretizing the equations of fluid motion, derived so far. One of the objectives of our discretization is to preserve structure of the equations under discretization. For instance, if we discretize the continuity equation (or conservation of mass equation) then the discrete form shall also conserve mass.

One numerical scheme that is especially suited for discretization of conservation laws is the finite volume method. We will start the presentation with a one-dimensional example.

#### 7.2.1 One-Dimensional Model Problem

**and Basic Algorithmic Idea**

In order to derive the basic idea behind finite volume methods (FVM), we start with a 1D model problem, namely the advection equation in flux form for a density distribution $\rho = \rho(x, t)$:
Fig. 7.2. Idea of the finite volume method: $F_{k-}^t$ represents the flux from the upstream side of cell $\tau_k$, contributing to the mean value $Q_k^{t+\Delta t}$. The figure shows Lagrangian fluxes that are translated into the Eulerian flux contribution for each cell interface in finite volume methods

$$\partial_t \rho + \partial_x F(\rho) = 0,$$

where $F = u\rho$ is the flux function. By using the transport theorem A.0.5 we get an integral form of (7.18)

$$\frac{d}{dt} \int_G \rho(x,t) \, dx = 0,$$

where $G$ is the whole domain. Assume we have a fixed mesh of cells $\tau_i$, $i = 1 : N$, then the mass in volume $\tau_k$ can only change by inflow and outflow over the left and right borders of $\tau_k$ (see fig. 7.2). This can be formulated by integrating (7.18) over the time interval $[t, t+\Delta t]$ and the grid cell $\tau_k = [x_k, x_{k+1}]$:

$$\int_{\tau_k} \rho(x,t+\Delta t) \, dx = \int_{\tau_k} \rho(x,t) \, dx + \int_{t}^{t+\Delta t} F(\rho(x_{k+1},t)) \, dt - \int_{t}^{t+\Delta t} F(\rho(x_k,t)) \, dt.$$  

(7.20)

Taking $Q_k^t$ as a discrete mean value of $\rho$ in the grid cell $\tau_k$, i.e. $Q_k^t \approx \frac{1}{\Delta x} \int_{\tau_k} \rho(x,t) \, dx$, where we assumed that $|\tau_k| = x_{k+1} - x_k = \Delta x$, and defining the time averaged mass fluxes

$$F_{k-} \approx \frac{1}{\Delta t} \int_{t}^{t+\Delta t} F(\rho(x_k,t)) \, dt, \quad \text{and}$$

$$F_{k+} \approx \frac{1}{\Delta t} \int_{t}^{t+\Delta t} F(\rho(x_{k+1},t)) \, dt,$$

we obtain the discrete finite volume representation of (7.20)

$$Q_k^{t+\Delta t} = Q_k^t + \frac{\Delta x}{\Delta t} (F_{k+} - F_{k-}).$$  

(7.21)

This equation represents the main idea of finite volume methods. Now, there are a lot of more or less subtle issues in order to construct a stable, efficient and accurate discrete method from this principle. There are choices to be taken for
Discretization of Conservation Laws

- the time discretization \( \frac{d}{dt} \) in (7.19);
- the calculation of fluxes \( F_{k\pm} \);
- the cell averages \( Q_k^t \);
- the finite volumes \( \tau_k \).

For these details the reader is referred to [263, 264].

7.2.2 Multidimensional Extension

A multi-dimensional representation of (7.18) reads

\[
\partial_t \rho + \nabla \cdot F(\rho) = 0,
\]

(7.22)

where \( F(\rho) = \rho \mathbf{v} \) is the flux function, and \( \mathbf{v} \) is a given velocity.

An equivalent form of this conservation law is given by the integral form

\[
\frac{d}{dt} \int_V \rho(x,t) \, dx = 0,
\]

where we have used the transport theorem A.0.5. Here, \( V \) denotes the whole computational domain as a control volume. Furthermore, we assumed that no boundary conditions impose a flux of material into or out of the domain.

In order to derive a finite volume scheme, we consider a (small) reference volume \( V \). Since the above conservation law (7.22) represents conservation of mass, we state that mass within \( V \) can only change due to mass fluxes over the boundary \( \partial V \):

\[
\frac{d}{dt} \int_V \rho(x,t) \, dx = - \oint_{\partial V} f(\xi,t) \cdot \mathbf{n}(\xi) \, dx,
\]

(7.23)

where \( \xi \in \partial V \) is on the boundary of \( V \), \( \mathbf{n} \) is the outer normal unit vector, and \( f \) represents the contribution over \( \partial V \). Note that by the minus sign, we state that \( f \) denotes the inward contribution. Integrating both sides over the time interval of one time step \( I = [t, t + \Delta t] \) yields:

\[
\int_V \rho(x, t + \Delta t) \, dx = \int_V \rho(x, t) \, dx + \int_t^{t+\Delta t} \oint_{\partial V} f(\xi, t) \cdot \mathbf{n}(\xi) \, dx \, dt.
\]

(7.24)

Note that we assume \( V \) to be fixed (but arbitrary) in time and space.

Let us assume that there is a mesh with simplicial cells (for triangular and quadrilateral cells see chap. 3). Let the reference volume \( V = \tau \) be a cell with edges/faces \( e_j, \, j = 1 : M \).

**Algorithm 7.2.1 (Basic finite volume method)**

A finite volume method (FVM) can be derived utilizing the following components. For each cell \( \tau \in T \) in the triangulation:
1. Let $Q^t_\tau$ be the cell mean value of $\rho$,

$$Q^t_\tau \approx \frac{1}{|\tau|} \int_\tau \rho(x,t) \, dx,$$

where $|\tau|$ is the area/volume of cell $\tau$.

2. Approximate the in-/outflow over cell edges/faces by

$$F^t_j \approx \frac{1}{\Delta t} \oint_{t}^{t+\Delta t} \left( \frac{1}{h_j} \int_{e_j} f(\xi,t) \cdot n \, dx \right) \, dt,$$

where $h_j$ denotes the edge length or face area resp., and $F^t_j$ is the time averaged mean edge/face flux.

3. Then equation (7.24) can be discretized by

$$Q^{t+\Delta t}_\tau = Q^t_\tau + \Delta t \sum_{j=1:M} h_j F^t_j.$$

Note that we have to divide by $|\tau|$ and multiply by $h_j$ and $\Delta t$ in order to reproduce (7.24). Furthermore, the terms on the right hand side only depend on time $t$.

There is a large family of different finite volume methods. The main differences concern the computation of fluxes $F^t_j$ (see [264] for a comprehensive presentation of finite volume methods, and Thuburn for shape preserving flux computation [392]).

Note that a solution of the above algorithm may be discontinuous. The algorithm requires a solution to be piecewise continuous only. Therefore, FVM are well suited for problems that develop shocks. A typical class of equations that are solved with FVM are hyperbolic conservation laws.

In the simplest case, where $Q_\tau$ represents a constant value in the cell $\tau$, in general we will have that $Q_{\tau_j} \neq Q_{\tau_i}$ for two distinct but neighboring cells $\tau_i \cap \tau_j = e_j$. Then calculating the flux over $e_j$ corresponds to solving a one-dimensional Riemann problem in the normal direction with respect to $e_j$.

Methods that utilize Riemann problems to derive numerical flux functions $F_j$ are called Godunov’s methods. Examples for Godunov’s method in an adaptive mesh refinement atmospheric simulation can be found in [207] and in combination with mesh moving adaptivity in [13]. On triangular uniform meshes Pesh has implemented a finite volume scheme for the shallow water equations in the scalar potential vorticity-divergence formulation [316]. Choi and coworkers present a spectral finite volume method for the shallow water equations [85]. An upwinding finite volume type method for the shallow water equations can be found in [160] while a comparison of several upwind difference methods is given in [172]. Botta et al. propose a finite volume scheme that is well balanced in the sense that it is numerically stable for an ill conditioned pressure gradient in terrain following coordinates [67]. Another approach to
a similar gravity wave related problem is given in [242], utilizing a finite volume method with an orography adapted grid. A general approach of finite volume methods on unstructured meshes can be found in [115], while a high-order accurate scheme for unstructured meshes is given in [305]. Generally constraint-preserving finite volume schemes for advection are given in [395].

7.2.3 Convergence, Consistency and Stability

In order to assess the convergence of the scheme, we need to introduce some notation. We will use the fundamental theorem of numerical analysis which is well known as Dahlquist’s equivalence theorem for ordinary differential equations or as the Lax equivalence theorem for partial differential equations. It states that a method converges if and only if it is consistent and stable. Let the numerical method be given by the evolution operator $\Psi$ that maps a solution at time $t$ to a new solution at time $t + \Delta t$: 

$$\Psi : V \rightarrow V$$

$$Q^t \mapsto Q^{t+\Delta t} = \Psi(Q^t),$$

where $V$ is the solution space, and $Q^t$ is a discrete solution (e.g. the cell mean values for all cells) at time $t$. Then we can define the one-step error:

**Definition 7.2.2 (One-step error, truncation error)**

The one-step error $\varepsilon^1$ is defined by 

$$\varepsilon^1 = \Psi(q^t) - q^{t+\Delta t},$$

where $q^t$ is the true solution at time $t$. The local truncation error $\varepsilon^t$ is defined as the one-step error scaled by the time step:

$$\varepsilon^t = \frac{1}{\Delta t} (\Psi(q^t) - q^{t+\Delta t}).$$

A simple possibility to compute the mass flux between cells consists of the first order finite difference scheme $F_j = n_j \cdot \mathbf{v} (Q_{T_j} - Q_{T_i})$. Here $n_j$ denotes the outward unit normal vector on edge $e_j$. Calculating the numerical flux in this way, results in a first order upwind method. One can show that the truncation error for the upwind scheme is of order

$$\varepsilon^t = O(\Delta x) + O(\Delta t^2),$$

which means that it is dominated by the first order spatial error [264]. We say that the upwind scheme is of first order consistency. Essentially, this means that when we decrease the mesh size by a factor of two, then the local truncation error will also decrease at the same rate.

Now, that we showed consistency, we have to tackle the stability of the method. The stability assessment is concerned with a global bound of the
error when the time step approaches zero \((\Delta t \to 0)\), and hence the number \(n\) of time steps approaches infinity \((n \to \infty)\). To see the principle, we assume that in the \(n\)-th time step \(t = n\Delta t\) there is a corresponding error \(\varepsilon^n\) such that

\[
Q^t = q^t + \varepsilon^n.
\]

Using the evolution representation, we obtain \(Q^{t+\Delta t} = \Psi(Q^t) = \Psi(q^t + \varepsilon^n)\).

From this we deduce for the global error:

\[
\varepsilon^{n+1} = Q^{t+\Delta t} - q^{t+\Delta t} = \{\Psi(q^t + \varepsilon^n) - \Psi(q^t)\} + \{\Psi(q^t) - q^{t+\Delta t}\} = \{\Psi(q^t + \varepsilon^n) - \Psi(q^t)\} + \Delta t \varepsilon^t.
\]

Thus, the global error is defined by two terms: the one-step error of the current time step, and the error induced by the numerical method (represented by the evolution operator) applied to the perturbed data at time \(t\). This second term is bounded in a stable numerical scheme.

If \(\Psi\) is a contractive operator, i.e.

\[
\|\Psi(Q_1) - \Psi(Q_2)\| \leq \|Q_1 - Q_2\|,
\]

for some norm \(||\cdot||\) and two discrete functions \(Q_1\) and \(Q_2\), then from (7.26) we get

\[
\|\varepsilon^{n+1}\| \leq \|\Psi(q^t + \varepsilon^n) - \Psi(q^t)\| + \Delta t \|\varepsilon^t\| \leq \|\varepsilon^n\| + \Delta t \|\varepsilon^t\|.
\]

Applying this recursively for all \(n\) time steps and taking \(\|\varepsilon_{\text{max}}\| = \max_{0 \leq t \leq n\Delta t} \|\varepsilon^t\|\), we get

\[
\|\varepsilon^n\| \leq \|\varepsilon^0\| + t \|\varepsilon_{\text{max}}\|, \quad \text{with } t = n\Delta t.
\]

\(\|\varepsilon^0\|\) is the error in the initial data (continuous versus gridded data) and we require \(\|\varepsilon^0\| \to 0\) for \(\Delta t \to 0\) in order to solve the correct initial value problem. If the method is consistent (see above), then also \(\|\varepsilon_{\text{max}}\| \to 0\) for \(\Delta t \to 0\), which proves the stability and therefore the convergence of the method.

Note that a relaxed condition on the operator \(\Psi\) is sufficient to prove convergence:

\[
\|\Psi(Q_1) - \Psi(Q_2)\| \leq (1 + \alpha \Delta t) \|Q_1 - Q_2\|,
\]

where \(\alpha\) is a constant independent of \(\Delta t\). If the method is linear, i.e. \(\Psi\) is a linear operator, then the analysis becomes even more simple, since \(\Psi(q^t + \varepsilon^n) = \Psi(q^t) + \Psi(\varepsilon^n)\). So from the requirement

\[
\|\Psi(q^t + \varepsilon^n) - \Psi(q^t)\| = \|\Psi(\varepsilon^n)\| \leq (1 + \alpha \Delta t) \|\varepsilon^n\|
\]

we deduce that \(\|\Psi\| \leq 1 + \alpha \Delta t\). It suffices for \(\Psi\) to be bounded by a constant \((\|\Psi\| \leq C)\) to show stability of a linear method. This is referred to as the Lax-Richtmyer stability. One can show that the above given upstream method is a linear method fulfilling the requirement of a bounded evolution operator.
A von Neumann stability analysis for the upwind method discretizing the advection equation

\[ Q^{t+\Delta t}_T = (1 - \nu)Q^t_T + \nu Q^t_{T_{\text{upwind}}} \]

yields that the method is stable if the Courant number \( \nu = v \frac{\Delta t}{\Delta x} \) fulfils \( 0 \leq \nu \leq 1 \). This stability criterion is referred to as the Courant Friedrichs Levy (CFL) stability condition [102].

Generally, FVM are restricted by the CFL condition, which makes them less advantageous for adaptive methods, since the time step is restricted by the smallest grid size. However, many current AMR methods rely on FVM, and employ time sub stepping in order to retain stability while maintaining efficiency even with locally refined grids [14, 52, 54, 55, 283, 291, 384]. A more rigorous convergence analysis can be found in [94] or [301]. Stability of time-implicit finite volume schemes has been studied in [158]. Extensions to the finite volume methods and to the computation of fluxes on unstructured tetrahedral meshes can be found in [11]. A relaxation of the CFL condition has been proposed in a so called h-box method [53]. A convergence proof and relation of higher order finite volumes to discontinuous Galerkin methods is given in [246]. Spectral finite volumes are employed in [407] to solve conservation laws in 2D.

Lin and Rood [269, 268] extend the finite volume approach to long time steps by observing that fluxes can be computed by following the Lagrangian trajectories of the cells. They use a splitting scheme for the 2D case, where \( F_x \) and \( F_y \) are the 1D flux functions in \( x \)- and \( y \)-direction respectively. Then the splitting scheme can be written as

\[ Q^{t+\Delta t} = Q^t + F_x \left[ Q^t + \frac{1}{2} F_y(Q^t) \right] + F_y \left[ Q^t + \frac{1}{2} F_x(Q^t) \right] \]

Now the fluxes are decomposed into an integer part (corresponding exactly to integer multiples of the time step) and a fractional part that can be computed by any available Eulerian upstream flux form scheme. This method is shown to be unconditionally stable for uniform flow fields. The original CFL stability criterion is replaced by a so called Lipschitz stability criterion that states that the trajectories must not cross or that the upstream cell must not collapse (see also [371]). Jablonowsky uses the Lin-Rood scheme in an adaptive dynamical core [221]. Lanser and coworkers proposed an Osher type finite volume scheme for shallow water equations on the sphere [253, 254].

### 7.3 Discontinuous Galerkin Methods

Discontinuous Galerkin methods have been introduced by Reed and Hill [334] and a good overview of the state of the art up to 1999 can be found in [95]. A comparison of different finite difference, finite volume and discontinuous
Discontinuous Galerkin (DG) methods has been conducted by Shu [365]. An extension to nonlinear hyperbolic conservation laws can be found in [92]. Today, discontinuous Galerkin methods have been applied to all kinds of problems, including elliptic partial differential equations [12]. An adaptive DG method based on an orthogonal basis is proposed in [336]. The popularity of DG methods stems from their scalability in parallel applications and their accuracy when combined with high order approximation methods.

### 7.3.1 The Basic Idea for Ordinary Differential Equations

In order to derive the discontinuous Galerkin method, we start with the one-dimensional case again. Assume, we want to solve an ordinary differential equation (ODE) of the form

$$\frac{d\rho(t)}{dt} = f(t)\rho(t), \quad (7.27)$$

on the time interval \( t \in I = [0, T] \) and with initial condition \( \rho(t = 0) = \rho_0 \).

We will denote the discrete approximation to the true solution \( \rho \) of (7.27) by \( \rho_h \). For the discontinuous Galerkin (DG) discretization, the interval \( I \) is partitioned into sub-intervals \( I^n = [t^n, t^{n+1}], \ n = 0 : N - 1, \) where \( t^0 = 0 < \cdots < T = t^N \). The discrete solution \( \rho_h \) is determined on each \( I^n \) as the polynomial of degree at most \( p^n \) (the polynomial degree is allowed to change on the sub-intervals) such that

$$-\int_{I^n} \rho_h(s) \frac{db(s)}{dt} \, ds + \hat{\rho}_h b|_{t^{n+1}} = \int_{I^n} f(s)\rho_h(s)b(s) \, ds, \quad (7.28)$$

for all polynomials \( b \) of at most degree \( p^n \). This is the classical weak formulation after using integration by parts on the left hand side to shift over the derivative from the solution to the test function \( b \). Note that the link between two adjacent intervals is only due to the trace function \( \hat{\rho}_h \). In the simple case of constant polynomial degree \( \hat{\rho}_h \) can be interpreted as a flux function and the DG method is a representative of the finite volume methods. More precisely

$$\hat{\rho}_h(t^n) = \begin{cases} \rho_0, & \text{if } t^n = 0, \\ \lim_{\epsilon \downarrow 0} \rho_h(t^n - \epsilon), & \text{otherwise}. \end{cases}$$

Here \( \lim_{\epsilon \downarrow 0} \rho_h(t^n - \epsilon) \) denotes the upstream value of \( \rho_h \) at the interface \( t^n \).

We summarize the main properties of DG methods and refer for more details to the literature (e.g. [93, 95, 96]).

- DG methods comprise discontinuous discrete solution functions \( \rho_h \), so that no inter-element continuities have to be considered. This feature makes DG methods well suited for \( hp \)-adaptive methods and also efficient in parallel computing environments.
DG methods utilize a weak formulation of the problem, which makes them preferable for problems with irregular solution behavior.

DG methods rely on the definition of a suitable interface function, the numerical trace function \( \hat{\rho}_h \). The trace function is crucial to the convergence of DG methods.

The method introduced so far is consistent, if for the true solution \( \hat{\rho} = \rho \) holds.

For suitable choices of the numerical trace function \( \hat{\rho}_h \) one can prove the stability of the method.

DG methods are locally conservative.

### 7.3.2 Application of DG Methods to Multi-Dimensional Scalar Conservation Laws

We will first apply the DG method to the spatial discretization and later extend the DG method by Runge-Kutta time discretization. Let us consider the \( d \)-dimensional conservation law (7.22) again

\[
\rho_t + \nabla \cdot F(\rho) = 0,
\]

with \( F(\rho) = v\rho \), \( \rho \) and \( v \) functions of \((x, t), x \in \mathbb{R}^d, t \in [0, T]\) and an initial condition \( \rho|_{t=0} = \rho_0 \).

The weak form of the above equation with a discrete (piecewise polynomial) function \( \rho_h \) is given by

\[
\int_{\tau} \rho_h \, dx - \int_{\tau} v\rho_h \cdot \nabla b \, dx + \int_{\partial\tau} \nabla \rho_h \cdot n \, ds = 0,
\]

(7.29)

where \( b \) denotes a test function, \( \tau \) denotes an element of a triangulation \( T \) of the computational domain \( \mathcal{G} \subset \mathbb{R}^d \), and \( n \) is the outward normal unit vector with respect to \( \tau \). If we choose the numerical trace function to be

\[
\hat{v}\rho_h = v \lim_{\epsilon \to 0} \rho_h(x - \epsilon v, \cdot),
\]

then the corresponding DG method coincides with the classical upwind finite volume scheme. Again, this method is locally conservative, i.e.

\[
\frac{d}{dt} \int_{\tau} \rho_h \, dx + \int_{\partial\tau} \nabla \rho_h \cdot n \, ds = 0.
\]

This follows from setting \( b \equiv 1 \) in (7.29). Note that \( \tau \) is a fixed volume, so we have to consider the change of mass due to fluxes over the cell’s boundary \( \partial\tau \).

The above space discretization of the scalar conservation law can be combined with a Runge-Kutta (RK) scheme for the time discretization to form the Runge-Kutta Discontinuous Galerkin (RKDG) methods. Using a multi-step explicit RK scheme allows very efficient and scalable parallel solution of
the conservation equation. Starting with (7.29), we reformulate the equation substituting \( F(\rho) = \rho \mathbf{v} \) again

\[
\int_{\tau} (\rho_h)_{t} b \, dx - \int_{\tau} F(\rho_h) \cdot \nabla b \, dx + \int_{\partial \tau} \hat{F}(\rho_h) \cdot \mathbf{n} b \, ds = 0.
\]

Discretizing the integrals we see easily, that we obtain a system of ordinary differential equations of the form

\[
\frac{d\rho_h}{dt} = L(\rho_h),
\]

where \( L \) is a linear operator. This ODE can be discretized by a multi-step RK scheme in the following way:

**Algorithm 7.3.1 (Multi-step Runge-Kutta DG method)**

1. Set \( \rho_h^{(0)} := \rho_h^{t} \), where \( \rho_h^{t} \) is the solution of the conservation law at time \( t \).
2. For \( k = 1 : K \) compute intermediate functions

\[
\rho_h^{(k)} = \sum_{j=0}^{k-1} \alpha_{k,j} w_h^{k,j}, \quad w_h^{k,j} = \rho_h^{(j)} + \frac{\beta_{k,j}}{\alpha_{k,j}} \Delta t L_h \left( \rho_h^{(j)} \right),
\]

with \( L_h \) the discretized form of the operator \( L \) above, and \( \alpha_{k,j}, \beta_{k,j} \) given by the corresponding RK scheme.
3. Set \( \rho_h^{t+\Delta t} = \rho_h^{(K)} \).

Note that, in order to obtain a stable method, there are more requirements on the RK scheme. Usually, the RKDG method has to be combined with a generalized slope limiter that stabilizes the forward Euler step \( \rho_h^{(k)} \to w_h^{k,j} \). For the details consult [28] and the references therein. Another stability constraint follows from the von Neumann analysis of the RK method. When combining a \((k+1)\)-step RK method with a DG method with local polynomials of at most degree \( k \), then we obtain an method of order \((k+1)\) accuracy, provided that

\[
|\mathbf{v}| \frac{\Delta t}{\Delta x} \leq \frac{1}{2k+1},
\]

which, for high orders of approximation, is a severe CFL restriction.

RKDG methods are used by Giraldo and coauthors in an application to the spherical shallow water equations [169]. A spectral element based DG method for the shallow water equations has been proposed in [140]. Nair and coauthors use a total variation diminishing RK scheme for the time integration that renders additional filtering of high frequency oscillations obsolete [297, 296]. Discontinuous Galerkin methods are suited for multi-scale problems that comprise hyperbolic and parabolic character [81]. DG methods can also

\[1\] It should be noted that TVD schemes have to be applied in time and space to yield non-oscillatory filter-free RKDG methods.
be combined with ADER schemes, known to yield high order in finite volume
discretizations [132]. An implicit-explicit scheme with DG approach can be
found in [211]. Iskandarani et al. compared several schemes, including DG and
finite volume schemes for advection equations in [216]. A convergence analysis
of DG methods for advection dominated flows in the case of discontinuous
solutions is given in [402] while smooth solutions are considered in [427].

7.4 Conservative Semi-Lagrangian Methods

Semi-Lagrangian schemes, probably introduced in the late 1950's by Wiin-
Nielsen [412] (see also [343, 351]), have gathered a wide acceptance for solv-
ing advection-dominated problems, especially in the atmospheric sciences. An
overview of the state of the art of semi-Lagrangian methods up to 1990 can
be found in [374]. Combining the semi-Lagrangian characteristics based time
discretization with a finite element space discretization yields the Lagrange-
Galerkin methods [56, 129, 190, 380]. These methods are appreciated for their

• numerical stability, allowing long time-steps,
• accuracy, which allows arbitrary high order approximation, and
• algorithmic simplicity facilitating the parallelization and easy implemen-
tation.

One of the main drawbacks of the method, however, is the lack of conser-
vation properties in its original formulation. Therefore, several investigations
aimed to find semi-Lagrangian procedures that ensure conservation of mass
or other quantities. Some of the early approaches just fixed the global mass
in the advection procedure more or less intelligently (see e.g. [57, 327, 381]).
Other approaches rely on integrating the upstream concentration function and
preserve the integral along the trajectory, like in [329]. In [298] the integral
form is used together with a constrained interpolation approach to gain a
conservative semi-Lagrangian method.

A conceptually different approach is to use geometric properties of the
flow. Here, volumes are advected along trajectories and physical values are
preserved within these volumes. For rectangular and uniform meshes Scroggs
and Semmazzi [360] and Machenhauer and Olk [271] have shown geometry
based approaches. Priestley applied this to triangular meshes [328]. Peng et
al. have implemented a conservative semi-Lagrangian scheme in the framework
of MM5, the meso-scale model of Pennsylvania State University and NCAR
[313].

The Eulerian-Lagrangian Localized Adjoint Method (ELLAM) has been
introduced by Celia et al. [80] and Russell and Trujillo [349] (see also [406]). An
arbitrary Lagrangian-Eulerian method suitable for moving grids is proposed in
[162]. An efficient remapping scheme for Eulerian-Lagrangian schemes for tri-
angular meshes can be found in [363], and an ELLAM scheme for unstructured
meshes with non-uniform time stepping is proposed in [423]. These methods are related to the weak Lagrange-Galerkin method used by Giraldo [168] and introduced by Benque and Labadie [249]. For the characteristic Galerkin method, see [267]. Essentially, the weak form of the conservation form of the equation is integrated at upstream positions. The difficulty consists of transferring the upstream integrated values to downstream grid points. Recently, Chen proposed a combination of characteristics based (semi-Lagrangian) and discontinuous (nonconforming) finite element methods [83]. A combination of Eulerian-Lagrangian and finite volume methods that circumvents the CFL restriction for finite volume schemes can be found in [192]. A forward-in-time semi-Lagrangian conservative method can be found in [377]. Another short characteristics based approach has been proposed in [112]. Falcone and Ferretti apply a semi-Lagrangian method to evolutive Hamilton-Jacobi equations and prove certain conservation properties. For 1D implementations they even show a correspondence to Godunov methods [143]. Israeli and coworkers combine a semi-Lagrangian scheme with a conservative splitting scheme to obtain an unconditionally stable and conservative method for the shallow water equations [219]. Finally, we like to mention the possibility of even extending the framework of Lagrangian techniques to the broad field of geometric integrators (see [265]).

Although some of the above papers mention the possibility of extending the respective approach to unstructured, adaptively refined meshes, there are hardly any demonstrations of these assumptions. Recently, Iske and Käser presented work related to the schemes presented here [218]. Some of the material in this chapter has been published in [42]. All these methods are suitable for adaptive mesh refinement.

In order to set up a framework for conservative semi-Lagrangian advection, let us first consider the (Eulerian) flux form formulation of conservation of a density, say \( \rho = \rho(x, t) \in \bar{G} \):

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \quad \text{in} \; \bar{G},
\]

(7.30)

where the computational domain is defined by \( (x, t) \in \bar{G} = G \times I, \; G \subset \mathbb{R}^d, \; d = 2, 3, \) the spatial domain, \( I \subset [0, \infty[ \) the time interval, and \( \mathbf{v} = \mathbf{v}(x, t) \in \mathbb{R}^d \) is a given multi-dimensional flow. On the other hand, the Lagrangian formulation of the homogeneous advection equation, assuming a divergence-free wind \( \mathbf{v} \), is given by:

\[
\frac{d\rho}{dt} = 0 \quad \text{on} \; \bar{G},
\]

(7.31)

where \( \frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho \) is the material derivative. For being well defined, (7.30) and (7.31) need boundary conditions \( \rho(x_b, t) = \rho_b(x_b, t) \forall x_b \in \partial G \), and initial conditions \( \rho(x, t = 0) = \rho_0(x) \forall x \in G \), suitably chosen. In the following, we will not consider boundary conditions, so we assume that \( G = \mathbb{R}^d \).

Equation (7.31) in this form is not conservative, so we have to consider a source term with the divergence operator to obtain an equivalent to (7.30):
7 Discretization of Conservation Laws

\[
\frac{d\rho}{dt} = \rho \nabla \cdot \mathbf{v} \quad \text{on } \bar{G}.
\]  

(7.32)

In order to construct mass conserving semi-Lagrangian advection schemes, we want to consider the integral (conservation) form of the advection equation that can be obtained for example by the transport theorem A.0.5:

\[
\frac{d}{dt} \int_{V(t)} \rho \, dx = 0,
\]

(7.33)

with \( V(t) \) a reference volume moving with the flow. (7.33) will be discretized by the semi-Lagrangian method in the following section. We will derive definitions for local and global mass conservation from that.

The proposed algorithms either discretize the integrals by numerical quadrature schemes or by geometric intersection. Another approach, namely the MPSLM scheme, discretizes the integrals by a high finite number of mass cells, which essentially form a subdivision of grid cells.

### 7.4.1 Adaptive Semi-Lagrangian Scheme

In this section we give a brief introduction to the adaptive semi-Lagrangian scheme, introduced in [34]. We first consider equation (7.31) and use a time-centered discretization given by

\[
\frac{\rho(x, t + \Delta t) - \rho(x - 2\alpha(x), t - \Delta t)}{2\Delta t} = 0 \quad \text{in } \bar{G},
\]

(7.34)

where \( \Delta t \) is the discrete time step, \( \alpha(x) \) denotes the path to the upstream position corresponding to \( x \). \( \alpha(x) \) is given by a simple ordinary differential equation, namely

\[
\dot{x} = \frac{dx}{dt} = \mathbf{v}(x, t),
\]

(7.35)

where \( \mathbf{v} = \mathbf{v}(x, t) \) is the given velocity field as in (7.30). Solving (7.35) for \( t + \Delta t \) and taking the difference of the positions gives \( x(t + \Delta t) - x(t - \Delta t) = 2\alpha(x) \).

We end up with three steps of the so called three-time-level algorithm\(^2\) :

**Algorithm 7.4.1** (Basic semi-Lagrangian Algorithm)

1. Calculate \( \alpha(x) \) by solving (7.35).
2. Interpolate upstream value \( \rho(x - 2\alpha(x), t - \Delta t) \).
3. Update grid value \( \rho(x, t + \Delta t) \) using (7.34).

The given algorithm can be modified into a two-time-level algorithm by observing the fact that two decoupled calculations occur, when stepping forward in time: one acting on even, the other one on odd time steps. (7.34) is then modified to

\(^2\) The algorithm is called three-time-level algorithm, because it spans over three time levels, \( t + \Delta t, t, \) and \( t - \Delta t \)
\[ \rho(x, t) - \rho(x - \alpha(x), t - \Delta t) = 0 \Rightarrow \rho(x, t) = \rho(x - \alpha(x), t - \Delta t), \quad (7.36) \]

where we assume time level \( t \) to be unknown. Discretizing the ODE by a fixed point iteration, we obtain the discrete form of (7.35):

\[ \alpha^{k+1}(x) = \Delta t \cdot v \left( x - \frac{\alpha^{k}(x)}{2}, t + \frac{\Delta t}{2} \right). \quad (7.37) \]

There are many more possibilities to solve the ODE (7.35). A comprehensive comparison for this part of the computation can be found e.g. in [7]. In [281] another way of improving the computation of \( \alpha \) is demonstrated, while a semi analytical way to determine departure points is proposed in [276]. The convergence has been studied by McDonald [279, 280] and Falcone and Ferretti [142]. An analysis for the related Lagrange-Galerkin methods for advection-dominated flows can be found in [31]. Another study of the stability is presented in [170].

So far, we have not mentioned a mesh. In fact, this algorithm in principal works on arbitrary meshes and even in mesh-less situations [41]. However, we consider adaptively refined triangular and tetrahedral meshes in 2D and 3D here.

In order to formulate the adaptive semi-Lagrangian algorithm for equation (7.36), two different meshes have to be considered, one at time \( t \) which will be modified according to a suitable refinement criterion, and one fixed mesh at time \( t - \Delta t \). We denote the \( k \)-th iterate of the new mesh with \( T^{(k)}(t) \) and the old fixed mesh with \( T(t - \Delta t) \). We embed the semi-Lagrangian algorithm into an adaptive iteration to obtain:

**Algorithm 7.4.2 (Adaptive semi-Lagrangian Algorithm)**

1. Duplicate mesh \( T(t - \Delta t) \), obtaining an initial mesh for the adaptive iteration \( T^{(0)}(t) \).
2. \textbf{WHILE} \( T^{(k)}(t) \neq T^{(k-1)}(t) \) \( (k > 0) \):
3. \textbf{Perform the semi-Lagrangian algorithm for all} \( x \in T^{(k)}(t) \):
   a) Calculate \( \alpha(x) \) from (7.37).
   b) Calculate \( \rho(x - \alpha(x), t - \Delta t) \).
   c) Update \( \rho(x, t) \) according to (7.36).
4. Estimate the local error \( [\varepsilon]_{\tau} \) for each cell \( \tau \) of \( T^{(k)}(t) \).
5. Refine those elements \( \tau \), where \( [\varepsilon]_{\tau} > \theta_{\text{ref}} \cdot [\varepsilon]_{\max}, [\varepsilon]_{\max} = \max_{\tau \in T^{(k)}(t)} [\varepsilon]_{\tau}, \) and \( \theta_{\text{ref}} \) a given tolerance (see sect. 2.4). Coarsen the mesh analogously, to obtain \( T^{(k+1)}(t) \).
6. Set \( k \leftarrow k + 1 \).
7. \textbf{END WHILE}

This adaptive semi-Lagrangian method has been introduced in [34]. We use cubic spline interpolation at the upstream position in the 2D case and a modified quadratic Shepard method in the 3D case [338]. In order to achieve
monotonicity of the solution density, we apply a clipping algorithm, as described in [34]. Algorithm 7.4.2 is defined pointwise, where we evaluate $\rho(x,t)$ at grid points.

If we apply the same machinery to equation (7.33), we obtain

$$
\int_{V(t)} \rho(x,t) \, dx = \int_{V(t-\Delta t)} \rho(x,t-\Delta t) \, dx,
$$

(7.38)

where $V(t-\Delta t)$ is the upstream reference volume. So, instead of computing point values, we now need to define associated reference volumes $V(x,t)$ for each grid point.

One possibility is to choose the dual cell corresponding to $x$. The dual cell is the polyhedron, formed by the dual points $\xi_i(x)$, $i = 1:n(x)$, surrounding $x$ (see fig. 7.3). Here we take the surrounding cell’s barycenters as dual points.

According to this definition of the reference volume, the upstream reference volume $V(x-\alpha(x), t-\Delta t)$ is then formed by the polyhedron of all upstream dual points $\xi_i^{-} := \xi_i(x) - \alpha(\xi_i(x))$. It is intuitively clear that we have to require the trajectory pieces $\alpha(\xi_i(x))$ not crossing within one time-step, in order prevent volumes from collapsing and to guarantee stability of the method (Lipschitz-stability, see [371]). Now, we are able to define local and global mass conservation.

**Definition 7.4.3 (Global mass conservation)**

Let $\Psi(\sigma)^{T}_{0} : \rho(x-\alpha(x),t_{0}) \rightarrow \rho(x,t_{1})$ the discrete evolution equation implementing a numerical scheme denoted by $\sigma$, $x-\alpha(x)$ being the upstream position, and suppose $\mathcal{G} = \mathbb{R}^d$. Let furthermore $I = [0,T]$. Then we say that the scheme $\sigma$ is globally mass conserving, if

$$
\int_{\mathcal{G}} \rho(x,T) \, dx = \int_{\mathcal{G}} \Psi(\sigma)^{T}_{0}[\rho_{0}(x)] \, dx = \int_{\mathcal{G}} \rho_{0}(x) \, dx.
$$

**Remark 7.4.4** Note that the above definition can easily be extended to the case where $\mathcal{G} \subseteq \mathbb{R}^d$, $I = [t,T]$. 

Fig. 7.3. Dual nodes (■), forming a dual cell, corresponding to an original node (●)
Definition 7.4.5 (Local mass conservation)

With the assumptions from 7.4.3 and denoting a volume corresponding to the space-time position \((x,t)\) by \(V(x,t)\) we say that the scheme \(\sigma\) is locally mass conserving if

\[
\int_{V(x,t)} \rho(x,t) \, dx = \int_{V(x,t)} \Psi(\sigma)_{t-\Delta t}(\rho(x, t - \Delta t)) \, dx \\
= \int_{V(x-\alpha(x), t-\Delta t)} \rho(x, t - \Delta t) \, dx. \tag{7.39}
\]

Remark 7.4.6 It is easy to see that if a scheme is locally mass conserving, then it is globally mass conserving, while the converse is not necessarily true.

The locally mass conserving semi-Lagrangian schemes, described in the next section are all based on (7.39). So, they all discretize in one or the other way

\[
\int_{V(x,t)} \rho(x,t) \, dx = \int_{V(x-\alpha(x), t-\Delta t)} \rho(x, t - \Delta t) \, dx. \tag{7.40}
\]

Note that (7.40) is the discrete counterpart of (7.38).

7.4.2 Mass-Conserving Algorithms

This section is devoted to the description of several locally mass conserving semi-Lagrangian schemes. The introduction to the algorithms is given for the 2D versions for simplicity. Where applicable, extensions to 3D are mentioned.

Quasi-Conservative Semi-Lagrangian Algorithm by Priestley

We briefly review this method, introduced by Priestley [327] and applied to shallow water equations by Gravel and Staniforth [177]. It is built on top of a quasi-monotone semi-Lagrangian scheme proposed by Bermejo and Staniforth [58]. This method is not locally mass conserving, but only globally mass conserving; and it is not based on equation (7.40).

The algorithm is based on a blending mechanism. Given a high order (possibly non monotone) and a low order (monotone) interpolation of the upstream concentration \(\rho^H\) and \(\rho^L\) resp., the overall upstream concentration \(\rho^-\) is calculated by the following formula:

\[
\rho^-_k = \beta_k \rho^H_k + (1 - \beta_k) \rho^L_k
\]

for each grid point \(k = 1 : n\). \(\beta_k\) is maximized under the constraints \(0 \leq \beta_k \leq 1\) and \(\min([\rho_k, l], \rho^L_k) \leq \rho^-_k \leq \max([\rho_k, l], \rho^L_k)\), where \([\rho_k, l]\) is the set of the \(l\) neighboring concentration values. This describes essentially the algorithm of Bermejo and Staniforth.

Now, Priestley’s algorithm adds another constraint to the \(\beta_k\)'s, namely choose \(\gamma_k, 0 \leq \gamma_k \leq \beta_k\) with
\[ \rho_k^- = \gamma_k \rho_k^H + (1 - \gamma_k) \rho_k^L \]
such that
\[ \int_{\mathcal{G}} \rho^- (x) dx = \int_{\mathcal{G}} \rho^0 (x) dx = C, \]
where \( \rho^0 \) is the initial condition function. Priestley’s algorithm implements a linear programming approach and is very efficient. It has been implemented in the context of the adaptive semi-Lagrangian method in [34]. It adds a very small overhead to the computational work, while maintaining exact global conservation in the absence of convergent and divergent wind fields. However, it cannot maintain conservation properties in converging and accelerating wind.

**Cell-Integrated Semi-Lagrangian Algorithm**

The cell-integrated algorithm, introduced to the meteorology community by Machenhauer and Olk [271] and previously investigated by e.g. Semmazzi and Scroggs [360], is based on equation (7.40). In the mentioned approaches, based on a quadrilateral grid, each grid point is associated a cell. For each of the four cell corners the upstream position is determined. The concentration function is then integrated exactly over the upstream cell. Nair and Machenhauer and Nair, Scroggs and Semazzi give very efficient integration formulae based on geometric properties in [294, 295].

Here, we extend the algorithm to locally refined triangular meshes. For simplicity, we consider \( \rho (x, t - \Delta t) = \bar{\rho} \) to be constant on grid cells. The aim is to compute \( \rho (x, t) \) for all grid points \( x \). Consider the upstream dual cell \( V(x - \alpha (x), t - \Delta t) \) indicated by solid lines in fig. 7.4. The idea of the cell-integrated scheme is to geometrically intersect the upstream dual cell with the underlying fixed mesh. Then determine the mass contribution of each underlying cell to the (new) grid point value from the intersections.

In mathematical terms, we triangulate the upstream dual cell \( V(x - \alpha (x), t - \Delta t) \) into triangles \( t_i^{(x, t - \Delta t)} \), \( i = 1 : l_x, l_x \) being the number of
triangles forming the upstream dual cell (we use the index \( x \), since the coordinate determines uniquely the dual cell, the upstream dual cell, and their corresponding topological properties). Since we assume \( \rho \) to be constant on mesh cells \( \tau \) the mass contribution to each \( t_i^{(x,t-\Delta t)} \) is given by

\[
m_i = \sum_{\tau \in \mathcal{T}(t-\Delta t)} (|\tau \cap t_i^{(x,t-\Delta t)}| \cdot \bar{\rho}_{|\tau}).
\] (7.41)

\( \mathcal{T} \) is the mesh as in sect. 7.4.1. \( \rho(x,t) \) is then given by the formula

\[
\rho(x,t) = \frac{1}{|V(x,t)|} \sum_i m_i.
\] (7.42)

We are left with two problems:

1. to find those elements \( \tau \in \mathcal{T}(t-\Delta t) \), where \( \tau \cap t_i^{(x,t-\Delta t)} \neq 0 \), and
2. to exactly determine the intersection \( \tau \cap t_i^{(x,t-\Delta t)} \).

The first problem can be solved efficiently by a recursive algorithm, provided that a mesh refinement hierarchy is available (i.e. a nested refinement strategy like in sect. 3.3 is used). Starting on the coarsest level, a recursive procedure enters into all those cells \( \tau \) for which \( \tau \cap t_i^{(x,t-\Delta t)} \neq 0 \). Once the recursion reaches the finest level, the contributions are collected and summed in a backward recursion.

Problem 2 can be solved by standard computational geometry algorithms. Since we only need to intersect triangles with triangles, this can be accomplished very efficiently as for example described in [9]. In sect. 4.5 we reported on the intersection capability of \texttt{amatos}.

Since the crucial part of this algorithm is the intersection, for a 3D version, we would need the intersection of convex polyhedra. However, this is a non-trivial task. The intersection in 2D is computationally much more demanding than e.g. the linear programming problem in the previous algorithm 7.4.2. In 3D we did not attempt to implement a cell-integrated scheme, since we anticipated computational overhead that were not acceptable.

**Quadrature-Based Semi-Lagrangian Algorithm**

Reviewing (7.40), we see that we have to solve two integrals numerically. While the cell-integrated method of the previous section achieves this goal by geometrically intersecting the upstream cell with the underlying grid, the quadrature based method represents the integrals by numerical quadrature formulae. Morton et al. point to a possible instability issue when combining semi-Lagrangian schemes with inexact quadrature in [292].

The idea is to find a quadrature rule for

\[
\int_{V(x,t)} \rho(x,t) \, dx.
\]
Since in an adaptive mesh the reference volume $V(x, t)$ may be distorted, which is valid even more for the upstream volume $V(x - \alpha(x), t - \Delta t)$, we again triangulate the volume as shown in fig. 7.4.

Then, for each triangle in the dual cell, we can use a triangular quadrature scheme (see e.g. [138, 379]). For a first order approximation in each triangle, we can use

$$
\int_{t_i} \rho(x, t) \, dx \approx |t_i| \cdot \rho(\xi_i, t),
$$

where we denote $t_i, i = 1 : l_x$ the triangles forming the reference volume $V(x, t)$, $\xi_i$ the center of gravity of $t_i$. A second order approximation is given by values on the triangle’s edge centers, as indicated in fig. 7.5.

The quadrature based semi-Lagrangian scheme (with a first order quadrature scheme) is given by

$$
\rho(x, t) = \frac{1}{|V(x, t)|} \sum_{i=1 : l_x} |t_i^{(x, t-\Delta t)}| \cdot \rho(\xi_i - \alpha(\xi_i), t - \Delta t),
$$

(7.43)

where $t_i^{(x, t-\Delta t)}$ are the triangles forming the upstream dual cell. Higher order quadrature extends the above formula in an obvious way.

The quadrature-based algorithm by construction is not exactly (i.e. up to machine precision) mass conserving, because the values for $\rho$ at quadrature points are still interpolated and include error. However, this method is locally mass conserving up to the order of discretization, provided a quadrature rule of at least discretization order is used.

This scheme can be naturally extended to 3D. While the upstream dual cell now becomes a polyhedron in 3D, there is a triangulation into tetrahedral elements for which Newton-Cotes cubature formulae are given in [138]. The computational effort, however, is increased over the 2D case, since a lot more tetrahedra are needed to triangulate the upstream dual cell, resulting in an increased number of cubature points, and the interpolation at cubature points in 3D is computationally more expensive.
Cell-Weighted Semi-Lagrangian Algorithm

The idea for the cell-weighted semi-Lagrangian algorithm is based on a simplification of the first order quadrature scheme just introduced. Here, we simply approximate the integral by

$$\int_{V(x,t)} \rho(x,t) \, dx \approx |V(x,t)| \cdot \rho(x,t).$$

Doing so on both sides of (7.40) results in the following simple scheme

$$\rho(x,t) = \frac{|V(x - \alpha(x), t - \Delta t)|}{|V(x,t)|} \cdot \rho(x - \alpha(x), t - \Delta t). \quad (7.44)$$

Thus, the plain semi-Lagrangian scheme is modified by a factor $F = \frac{|V(x - \alpha(x), t - \Delta t)|}{|V(x,t)|}$, with $|\cdot|$ the area or volume, that recovers the conservation property.

This algorithm is very efficient, since additionally to the plain semi-Lagrangian method, it only needs to compute two areas/volumes. It can be extended easily to 3D. Note that we did not use any 2D specific notion in the formulation of the algorithm. However, as in the previous section, this algorithm is not exactly mass conserving (up to machine precision), since there is still an interpolation involved. For a constant, homogenous wind field, this algorithm reduces to the plain semi-Lagrangian algorithm, because $F = 1$ then. A similar approach can be found in [320].

Mass-Packet Based Semi-Lagrangian Algorithm

To describe the Mass-Packet Semi-Lagrangian Method (MPSLM) [285], we need to define mass packets. These are mass-volume units which are defined by their mass, volume and barycenter’s position. In general they are much smaller than the grid cells. They are used to subdivide the cells into smaller parts. The mass packets, created in each time step, will be advected according to the semi-Lagrangian idea. In this section, we will denote an item in the old mesh $T(t - \Delta t)$ by superscript $(-)$ while a new item at time $t$ is denoted by superscript $(+)$.  

First, the density $\rho^{(-)}$ must be transformed to mass values. Therefore, we define a mass attached at each node $N_j^{(-)}$ per cell $\tau_j^{(-)}$ by

$$m(N_j^{(-)}) = \rho(N_j^{(-)}) |A_j^{(-)}|,$$  

where we denote the volume by $|\cdot|$, $|A_j^{(-)}|$ is the corresponding area of $\tau_j^{(-)}$ as depicted in fig. 7.6.

Then the mesh cell $\tau_j^{(-)}$ is fully partitioned into $L$ disjoint pieces:  

$$\sum_{i=1}^{L} |M_j^i| = |\tau_j^{(-)}|$$  

where the $M_j^i$ are the $L$ mass packets of cell $\tau_j^{(-)}$ (see
Discretization of Conservation Laws

\begin{align*}
M_j^i & = \sum_{i=1}^{d+1} \lambda_i^i m(N_j^{i(-)} | M_j^i)
\end{align*}

Note that the barycentric coordinates have the property that \(\sum \lambda_i^i = 1\). Thus, keeping in mind the disjoint partition of cells in mass packets, summing over all mass packets in a cell, we obtain the exact mass assigned to each node. By this means the mass in mesh \(T(t - \Delta t)\) is virtually transformed into mass packets.

For each node of the new grid the upstream position is calculated following the plain SLM (7.36). Knowing the upstream grid cells and the mass packets on the old grid, a mapping step follows. The mapping consists of assigning each upstream (new) cell \(\tau_j^{(+)}\) the corresponding (old) mass packets. A mass packet is assigned to cell \(\tau_j^{(+)}\) if its barycenter is inside the upstream position of \(\tau_j^{(+)}\). It should be noted that this search algorithm represents the most time consuming part of the whole scheme. The mapping is illustrated in fig. 7.8.

After mapping the mass packets to new cells, the barycentric coordinates \(\mu_i^l\) with respect to the (new) nodes \(N_j^{l(+)}\) are computed. Finally, the nodes

\begin{align*}
\sum_{i=1}^{d+1} \lambda_i^i m(N_j^{i(-)} | M_j^i)
\end{align*}
Fig. 7.8. Mass packet mapping: The (solid) mass packets are associated to the upstream position of $\tau^{(+)}_j$

$N^{i(+)}_j$ get mass assigned corresponding to the mass packets found in cell $\tau^{(+)}_j$:

$$m(N^{i(+)}_j) = \sum_{l:M^l_j \in \tau^{(+)}_j} \mu^l_i m(M^l_j) \quad (7.47)$$

As for the $\lambda$’s it holds that $\sum_l \mu^l = 1$. Therefore, the mass is reassigned to the nodes without loss, except for mass packets lying outside the upstream (new) mesh.

Now that we know the mass at the (new) nodes, we can compute the density value

$$\rho(N^{i(+)}_j) = \frac{m(N^{i(+)}_j)}{|V^{i(+)}|} \quad (7.48)$$

where $V^{i(+)}$ is the control volume associated to node $N^{i(+)}_j$ composed of all $A^i_j$ in the patch of the node. This scheme guarantees that $\rho$ is non-negative. Details of the implementation and theoretical proofs of the scheme’s conservation properties can be found in [285]. Summing up, the MPSLM algorithm is given by

**Algorithm 7.4.7** (Mass Packet semi-Lagrangian Algorithm)

1. Compute nodal masses $m(N^{i(-)}_j)$ using (7.45).
2. Subdivide cells into mass packets and assign masses according to (7.46).
3. Compute upstream positions to $\tau^{(+)}_j$ and map mass packets to $\tau^{(+)}_j$.
5. Reconstruct nodal densities with (7.48).
Example Applications

This chapter is devoted to some example applications in adaptive atmospheric (and oceanic) modeling. Of course, there are many more examples of the application of adaptivity in atmospheric modeling. These examples represent projects that were realized by or in collaboration with the author. The first two examples in sections 8.1 and 8.2 are linear tracer transport examples, the third and fourth examples (section 8.3) are extracted from the research project PLASMA in which dynamical cores for a simplified climate model are under development. While these four examples share semi-Lagrangian time-discretization schemes which are well suited for adaptive mesh refinement (see [34]), the fifth example, demonstrating adaptive wave dispersion in section 8.4, employs a finite volume type discretization method for the shallow water equations.

8.1 Tracer Advection

Tracer advection – though very simple in it’s set-up – is a good testing environment for adaptive numerical schemes and for adaptive concepts. This is the reason, why some detailed investigation of this simple application is presented in this section. The example is a 2D tracer dispersion application in the arctic stratosphere. The following material is taken from a publication by the author and coworkers [39].

8.1.1 Filamentation of Trace Gas in the Arctic Polar Vortex

The perturbed chemistry within the polar stratospheric vortex has generated great interest in the respective dynamics. The atmospheric distribution of trace constituents in the lower stratosphere not only depends on chemical sources and sinks but also on the redistribution as a result of transport, induced by various dynamical processes [409].
The quasi-horizontal dispersion of a passive tracer on synoptic time scales is likely to occur through the formation of tracer filaments in the presence of a background wind shear. Plumb et al. showed that fine structures seen in aircraft data coincide with filamentary structures produced by a contour advection model [323]. Edouard et al. claim that the ozone depletion depends on the resolution of the model, as small scale structures influence the local depletion rate [136]. Although these results have been questioned in [361], it seems to be common sense that model resolution has an impact on density distribution and therefore on reaction rates. The scale cascade of tracers under the influence of shear and strain has also been investigated in detail by Haynes and Anglade [191].

Bregman et al. carried out in situ measurements of the trace gases $O_3$, $HNO_3$, and $N_2O$ in the Arctic lower stratosphere during February 1993 on board of a Cessna Citation aircraft during the first Stratosphere-Troposphere Experiment by Aircraft measurements (STREAM) campaign [72]. Strong variations in the concentrations and distributions of these trace gases were found. The time series with a resolution of minutes showed pronounced small-scale variations and large horizontal variations in the range of some dozen kilometers, indicating that the aircraft flew through air masses with different origins. Unfortunately these data do not allow the identification of horizontal scales below some dozen kilometers.

The horizontal resolution required for the reproduction of filaments cannot be achieved by general circulation models to date. Most approaches, therefore, use offline chemistry transport models (CTM) with high local resolution. When attempting to model filamentary structures, most other investigators use contour advection models (see [130] for a description). Other approaches utilize purely Lagrangian particle advection schemes (see e.g. [378]). In this section, we adopt an adaptive modeling technique, developed in [34], and [36], in order to achieve very high local resolution. (we go as far as approx. 5 km here).

Three main aspects are investigated.

- The influence of varying resolution of wind data;
- The applicability of very-high-resolution advection to coarse-resolution wind data;
- The influence of varying horizontal resolution of the transport model on the formation of small scale filamentary structures.

Other studies investigated the influence of resolution (see [408]).

We consider the horizontal transport of a conserved passive tracer in the lower polar stratosphere in an idealized configuration, using the wind components from a high-resolution regional climate model (HIRHAM) [119, 340]. HIRHAM is a sophisticated climate model, which resolves the arctic region with a horizontal resolution of 0.5°. It is forced at the lateral and lower boundaries by ECMWF reanalysis data. The simulated geopotential structure at 73.4 hPa (we call this the 70 hPa layer) during January 1990 with a cold low
is presented in fig. 8.1, well known as Polar Vortex with, in our situation two centers, one north of Greenland the other south-east of Novaya Zemlya.

8.1.2 Adaptive Tracer Transport Model

For the experiments, described in section 8.1.3, we use an adaptive tracer transport model. While a detailed description of the basic model ideas can be found in [34], we give a brief overview here. The model solves the advection equation in two dimensions:

\[
\partial_t \rho + \nabla \cdot (v \rho) = R \tag{8.1}
\]

The wind \(v = v(x,t)\) is given by the HIRHAM data (cf. section 8.1.1). \(R = R(x,t)\) is the right hand side of (8.1) which can contain additional forces, but is set to zero in our case (i.e. we simulate the advection of an idealized passive tracer without diffusion and without other sources and sinks). \(\rho = \rho(x,t)\) denotes the (scalar) concentration density of the tracer, while \(x = (x,y)\) is the horizontal coordinate in the 70 hPa layer.

The time dependent part of the equations is discretized by the conservative cell-weighted semi-Lagrangian method (see section 7.4.2). An adaptive grid generator (amatos) refines the triangular grid down to a lower bound, where the gradient of the tracer concentration \(\nabla \rho\) is large. On the other hand, in regions where \(\nabla \rho\) is small, the grid is coarsened up to an upper bound. The maximum-norm based refinement strategy is used (see algorithm 2.4.1 in section 2.4). We set \(\theta_{\text{ref}} = 0.2\) and \(\theta_{\text{crs}} = 0.1\). Table 8.1 gives the grid type according to the classification in section 3.2 and the corresponding horizontal grid resolution used in the experiments.
Table 8.1. Highest local grid resolution and corresponding grid type for the adaptive simulation

<table>
<thead>
<tr>
<th>grid type</th>
<th>horiz. resolution (km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNT[12]</td>
<td>55</td>
</tr>
<tr>
<td>CNT[4,15]</td>
<td>20</td>
</tr>
<tr>
<td>CNT[4,17]</td>
<td>10</td>
</tr>
<tr>
<td>CNT[4,19]</td>
<td>5</td>
</tr>
</tbody>
</table>

With this mechanism, we achieve very high local resolution without exhausting limited computing resources. An example of an adaptively refined grid is given in fig. 8.2.

8.1.3 Experiments and Results

In this sub-section we try to answer three main questions. The first one is of rather philosophical character, as we ask: why can we see small scale filaments in the tracer advection when these structures are not present (due to coarse resolution) in the driving wind data?\(^1\) There are other more in depth investigations of this question [323, 191], however we decided to take this example into consideration because of its simple and instructive character. The more practical aspects of our investigation will analyze the influence of resolution of wind data and model resolution to the formation of small scale structures.

Influence of wind gradient

In order to explore the first question, let us think of a simplified example: let \( \mathbf{v} \) in (8.1) be a constant circular wind with decreasing velocity towards the boundary, acting in a closed box: \( \mathbf{v}(x,y) = [u(x,y), v(x,y)] := [-y \cdot d, x \cdot \) \(d] \).

\[\text{Fig. 8.2. Adaptively refined grid, corresponding to the situation in fig. 8.6}\]

\(^1\) In fact, sub-grid features are present in the wind data as they are parameterized in the underlying model and thus contribute to the shape of the wind field!
Fig. 8.3. Initial configuration for the model problem with a circular wind field where $d = 4 \| (x_{center}, y_{center}) - (x, y) \|_2$. Let the initial concentration distribution be given by a rectangle with concentration 1 and zero everywhere else (see fig. 8.3).

After several time steps one will observe formation of small scale stirring in the transport model, while obviously no dynamical features are present in the underlying data. These small scale structures are caused by the presence of a gradient in the wind field acting orthogonally to the gradient of the concentration. In general, whenever a gradient in the wind field is non-tangential to the gradient of the concentration, stirring will occur. If this process can be resolved fine enough by the transport scheme, we will see small scale filamentation. On the other hand, if resolution is too coarse, filaments will not be visible due to numerical dissipation (see fig. 8.4).

It is therefore reasonable to use a very-high-resolution tracer transport scheme with data that are not equally highly resolved. The wind data contain sub-grid scale information, due to parameterized phenomena. The shape of the wind field is influenced by these parameterizations, resulting in filamentation by the above mentioned interaction of gradients.

Influence of Wind Data Resolution

In order to determine the influence of data resolution (i.e. the resolution of the given wind field) to the formation of small scale filaments, wind data are given on grids of $110 \times 100$, $55 \times 50$, and $22 \times 20$ grid-points respectively. This corresponds to a mesh size of approx. 50 km, 100 km, and 250 km respectively (or approx. $0.5^\circ \times 0.5^\circ$, $1^\circ \times 1^\circ$, and $2.5^\circ \times 2.5^\circ$ resp.). Minimal mesh size of the transport model is set to approx. 10 km, corresponding to the CNT[6,17] mesh. Here and in all the following experiments, the tracer concentration is initialized as shown in fig. 8.5. This position corresponds to the left “eye” of the Arctic Polar Vortex in January 1990.

Compared to the reference case ($110 \times 100$ data points), the $55 \times 50$ grid-point case shows no significant changes. However, reducing the data field to only $22 \times 20$ points results in a different behavior. On the one hand, wind structure is slightly different, resulting in a displaced path of the transport.
Fig. 8.4. Result of the advection with the model problem after some hours of model-time. With high (local) resolution, fine structures are visible (left) while numerical dissipation has led to heavy erosion with coarse (global) grid resolution (right). Note that the total mass of the tracer is conserved in both cases.

Fig. 8.5. Initial tracer distribution for the experiments

Fig. 8.6. Tracer concentration after 288 hours of model time, using highly resolved wind data on a grid of $110 \times 100$ grid-points (left), versus low-resolution $22 \times 20$ grid-points (right)
On the other hand, a more interesting behavior, namely the formation of small scale filaments, is deteriorated. When comparing both plots in fig. 8.6, one observes that fine stirring structures are present only in the high-resolution case.

Note that interpolation of the wind data set to the computational grid of the very-high-resolution transport model is a problem in itself. We tried bi-linear and bi-quadratic interpolation and found no significant difference. In the experiments we use bi-linear interpolation for efficiency and shape preserving reasons. This is in accordance with the choice in the contour advection simulations in [408].

**Influence of Model Resolution**

The next series of experiments aims at determining a necessary computational grid resolution for the formation of small scale filaments. Wind data are taken from the high-resolution data set (i.e. $110 \times 100$ data-points). The first case uses a uniformly refined grid with a resolution of approx. 55 km. This is almost the resolution of the given data set. Three additional model configurations are probed with local resolutions of approx. 20 km, 10 km, and 5 km respectively (see table 8.1). Note that all of these high-resolution tracer advection simulations can be easily computed on a standard workstation due to the adaptive refinement strategy.

As one would expect, finer resolution causes finer structures in the tracer concentration function. Figure 8.7 shows the tracer concentration for the different resolutions. While there are more and more second order filamentary structures visible, the main path and dispersion of the tracer field remain unchanged when increasing the resolution.

**8.2 Inverse Tracer Advection**

In air quality modeling and monitoring, one is often faced with the problem of reconstructing a source of pollution from measurement data. So, the inverse problem (of a forward directed simulation of tracer dispersion) can be formulated as follows:

\[
\text{Given a tracer (density) distribution } \rho(x, T) \text{ at some time } T, \text{ what was the distribution at time } t < T?\]

Note that the time advancement faces into the negative direction. This problem can be solved by the adjoint operator to the advection operator [137]. The adjoint or inverse modeling mode is sometimes called backward mode [362]. There is a large resource of meteorological and especially oceanographical literature on adjoint modeling or assimilation [260, 164, 357, 385]. Many of the adjoint models are derived by automatic differentiation techniques, so called adjoint compilers or pre-processors [99, 180]. Sirkes and Tziperman however,
argue that this (automated) approach might not be numerically stable, since it is not necessarily true that a stable forward discretization is also stable after automatic differentiation in the backward direction [366]. We do not comment on this result, but take it as a good reason for deriving the adjoint equations analytically and discretizing these equations by a numerically stable adaptive method. We shall briefly describe this technique here, referring to [331]. The reader is also referred to [386]. A similar strategy, utilizing a cost function and the generic finite element weak formulation of the model equations has been used by Dobrind for inversion of an ocean model [126].

Let $D$ be the linear differential operator, defined by

$$ D(\cdot) = \partial_t(\cdot) + \nabla \cdot (\mathbf{v}(\cdot)), \quad (8.2) $$

with a given velocity field $\mathbf{v}$. Then the advection equation, given by

$$ D(\rho) = r, \quad (8.3) $$

represents the transport of a tracer with a source term $r = r(x,t)$. For educational reasons and because in atmospheric simulation (lateral) boundary
conditions do not play a dominant role, we omit the detailed specification of adequate boundary and initial conditions and assume that these are suitably given. Furthermore, we assume that $\rho$ is given on a domain $G = G \times I \subset \mathbb{R}^d \times \mathbb{R}$, $I = [0,T]$. Let $H = H^1(G)$ be the Sobolev space of square integrable functions in $G$. Then let $D$ be defined on the subset $D(D) \subset H$ of functions that satisfy the boundary and initial conditions\(^2\).

The adjoint operator $D^*$ is defined by the duality relation

$$\langle D\phi, \psi \rangle = \langle \phi, D^* \psi \rangle, \quad (8.4)$$

where $\phi \in D(D)$ is an arbitrary function in the primal function space, $\psi \in D(D^*) \subset H$ is an arbitrary function in the dual space, and $\langle \cdot, \cdot \rangle$ is the inner product on $H$, given by

$$\langle \phi, \psi \rangle = \int_0^T \int_G \phi \psi \, dxdt,$$

with $\phi, \psi \in H$. Note that $D(D^*)$ is different from $D(D)$, and chosen such that the adjoint operator is well defined with appropriate boundary conditions. Now, for $\phi = \rho$ and $\psi = \rho^*$ we derive

$$\langle D\rho, \rho^* \rangle = \langle \rho, D^* \rho^* \rangle. \quad (8.5)$$

From (8.5) we deduce in analogy to (8.3) the adjoint advection equation given by

$$D^* \rho^* = q, \quad (8.6)$$

with $q$ representing a function of measurements, according to the problem formulation in the beginning of this section. By substitution of (8.6) and (8.3) into (8.5) it is easy to see that

$$\langle r, \rho^* \rangle = \langle \rho, q \rangle.$$

In order to derive an explicit formula for the adjoint equation to the advection equation, we proceed straightforward by substituting (8.2) into (8.5) to obtain

$$\langle D(\rho), \rho^* \rangle = \int_0^T \int_G \partial_t \rho \rho^* \, dxdt + \int_0^T \int_G \nabla \cdot (\mathbf{v} \rho) \rho^* \, dxdt.$$

After integration by parts and applying Green’s theorem, we get

$$\langle D(\rho), \rho^* \rangle = -\int_0^T \int_G \rho \partial_t \rho \rho^* \, dxdt - \int_0^T \int_G \rho \mathbf{v} \cdot \nabla \rho^* \, dxdt + F,$$

\(^2\) We deliberately omit the details of Hilbert spaces, specification of boundary and initial conditions, etc. and accept a great portion of sloppiness in this section. It is our intention to introduce to the basic idea of the adjoint and not to formulate the mathematical foundations of the derivation.
where $F$ represents further terms that arise from the integration by parts. Careful observation reveals that $F = 0$ for suitably chosen boundary conditions on $\rho^*$ and $\rho$. Therefore, we derive, using again the duality relation (8.4)

$$\langle \rho, D^*(\rho^*) \rangle = \int_0^T \int G \rho (-\partial_t \rho^* - \mathbf{v} \cdot \nabla \rho^*) \, dx dt. \quad (8.7)$$

From this, we deduce the adjoint operator to the advection operator given in (8.2):

$$D^*(\cdot) = -\partial_t (\cdot) - \mathbf{v} \cdot \nabla (\cdot). \quad (8.8)$$

Assuming a divergence free wind field ($\nabla \cdot \mathbf{v} = 0$), homogeneous forcing ($r \equiv 0$ and thus $q \equiv 0$), and a semi-Lagrangian advection scheme, where we discretize the material derivative

$$\frac{d \rho}{dt} \approx \rho(x,t) - \rho(x - \alpha, t - \Delta t) \over \Delta t$$

where $\alpha$ denotes the upstream particle path, we see that solving the adjoint equation is equivalent to solving the forward equation with an inverted (negative) wind field.

An academic example for the derived adjoint model is given in fig. 8.8. If we take the wind field for a forward computation as in the circular wind test case in section 8.1.2 above and the inverse wind being the negative of that. Then, we can

1. initialize the scalar field $\rho_{|t=0}$ by an arbitrary function (e.g. the logo "TUM");
2. compute the (discrete) solution at time $T$, $\rho_h_{|t=T}$ by a forward integration;
3. take $\rho_{|t=T}^{\text{inv}} = \rho_h_{|t=T}$ as initial data for the inverse (adjoint) model experiment,
4. integrate backwards in time to solve for $\rho_{|t=0}^{\text{inv}}$, which should correspond to $\rho_{|t=0}$.

We only show steps 3 and 4 in fig. 8.8. It can clearly be seen that adaptivity helps to reduce the diffusive error considerably. This is just a case study and further investigation needs to be conducted.

### 8.3 Shallow Water Equations

This section relies mostly on the results of the project PLASMA, which is a joint research project of the author’s group with two groups at the branches of Alfred-Wegener-Institute in Bremerhaven and Potsdam. Two paths have been opened for the development of a new adaptive dynamical core of an atmospheric model. A vorticity-divergence formulation and a velocity vector formulation. The first form has been tested extensively, while the second form has just been released (which is the reason for the relatively brief description).
8.3 Shallow Water Equations

Fig. 8.8. Initial field of tracer distribution (with grid) and final inverse modeling result after 48 hours of model time

8.3.1 Vorticity-Divergence Formulation

The implementation in vorticity-divergence form is called PLASMA-P. While the formulation of the model equations and their discretization has been developed mainly by Matthias Läuter [255, 258, 332], the adaptive mesh technology is provided by the package amatos [44], developed by the author’s group and the solution of the large linear systems of equations is comprised with the help of the solver library FoSSI by Frickenhaus and coworkers [156].

PLASMA-P in the current 2D version solves the spherical (rotational) shallow water equations (see appendix C) in vorticity-divergence formulation. The starting point for the derivation is given by the vectorial shallow water equations:

\[
\begin{align*}
\frac{d v}{dt} + \nabla_s \Phi + f_C n \times v + |v| <^2 n &= 0, \\
\frac{d \Phi}{dt} + (\Phi - \Phi_0) \nabla_s \cdot v - v \cdot \nabla_s \Phi_0 &= 0, \\
v \cdot n &= 0.
\end{align*}
\] (8.9)

Note that we used the tangential material derivative

\[
\frac{d}{dt} = \partial_t + v \cdot \nabla_s.
\]

These equations have been derived, starting from the shallow water equations as given in [100]. Remember that we denoted by \(\nabla_s\) the tangential (horizontal) \(\nabla\)-operator on the sphere (see appendix C). \(\Phi\) denotes the geopotential height and \(\Phi_0\) the orography, while \(v\) is the velocity and \(n\) the unit outward normal on the sphere.

We define the vorticity \(\zeta = \nabla_s \times v\) and the divergence \(\delta = \nabla_s \cdot v\). Applying the rotation and divergence operators to the momentum equation (first equation in (8.9)), a scalar formulation of the shallow water equations can be obtained, where we get additional equations from the Helmholtz decomposition.
\[
\frac{d\zeta}{dt} + \zeta \delta + \delta f_C + \mathbf{v} \cdot \nabla_s f_C = 0
\]
\[
\frac{d\delta}{dt} + \Delta_s \Phi - \zeta f_C + (\mathbf{n} \times \mathbf{v}) \cdot \nabla_s f_C + J(\mathbf{v}) = 0
\]
\[
\frac{d\Phi}{dt} + \delta(\Phi - \Phi_0) - \mathbf{v} \cdot \nabla_s \Phi_0 = 0
\]
\[
-\Delta_s \psi = \zeta
\]
\[
\Delta_s \chi = \delta
\]
\[
\nabla_s \times \psi + \nabla_s \chi = \mathbf{v}.
\]

The functions $\psi, \chi : S \times [0, T] \to \mathbb{R}$ are the stream function and velocity potential respectively, and $J$ is defined by

\[
J(\mathbf{v}) = (\nabla_s v_i)^T A^{i,j} (\nabla_s v_j) + \mathbf{v} \cdot \mathbf{v},
\]

with $A^{i,j}_{k,l} = \delta_{i,l} \delta_{j,k}$, and $\delta_{\mu\nu}$ the Dirac delta function. The first three equations in (8.10) are the actual scalar shallow water equations, while the last three equations represent the Helmholtz decomposition of the vector field $\mathbf{v}$. A thorough derivation and proof for the equivalence of (8.9) and (8.10) can be found in [256].

Equations (8.10) are discretized by an implicit Lagrange-Galerkin approach. However, the process is a little more complicated and is only outlined here. Again the details are given in [256]. In fact, PLASMA-P does not solve (8.10) but uses a stabilization by diminishing numerical diffusion. Thus, the advection operators $\mathbf{v} \cdot \nabla_s \varphi$, where $\varphi = \zeta, \delta, \Phi$ (hidden in the material derivative operators) are replaced by the operator $\mathbf{v} \cdot \nabla_s \varphi - \nu \Delta_s \varphi$ with suitably chosen diffusion parameter $\nu$.

A semi-discretization of (8.10) is given by the following system. Note that the mixed terms $\zeta \delta$ and $\Phi \delta$ have been treated by a time-averaging. Furthermore we have written $d_t \varphi = \frac{d\varphi}{dt}$.

\[
\begin{align*}
(d_t \zeta, b_h) + \nu_1 (\nabla_s \zeta, \nabla_s b_h) + (\zeta \delta, b_h) + (f_C \delta, b_h) &= (F_1, b_h) \\
(d_t \delta, b_h) + \nu_1 (\nabla_s \delta, \nabla_s b_h) + (\nabla_s \Phi, \nabla_s b_h) - (f_C \zeta, b_h) &= (F_2, b_h) \\
(d_t \Phi, \beta_h) + \nu_2 (\nabla_s \Phi, \nabla_s \beta_h) + (\delta \Phi, \beta_h) - (\Phi_0 \delta, \beta_h) &= (F_3, \beta_h) \\
(\nabla_s \psi, \nabla_s \gamma_h) &= (\zeta, \gamma_h) \\
(\nabla_s \chi, \nabla_s \gamma_h) &= - (\delta, \gamma_h) \\
(\nabla_s \times \psi + \nabla_s \chi, w_h) &= (\mathbf{v}, w_h).
\end{align*}
\]

Here, $b_h, \beta_h, \gamma_h, \text{and } w_h$ represent suitable variational test functions, while $(\cdot, \cdot)$ is the appropriate inner product; we also assumed suitably chosen Hilbert spaces for the equations to be well defined. Now the first three equations in (8.11) are further discretized in time by a semi-Lagrangian time integration method. Using quadrature rules to discretize the inner products above, finally leads to a large blocked system of equations, given by
\[
\begin{bmatrix}
M_1 + \Delta t(v_1S_1 + B_1) & \Delta t(C + B_2) & 0 \\
-\Delta tC & M_1 + \Delta t\nu_1S_1 & -\Delta tS_{12} \\
0 & B_4 & M_2 + \Delta t(v_2S_2 + N_3)
\end{bmatrix}
\begin{bmatrix}
\zeta_h \\
\delta_h \\
\Phi_h
\end{bmatrix}
= \begin{bmatrix}
F_1 \\
F_2 \\
F_3
\end{bmatrix}.
\] (8.12)

The vectors \(\zeta_h\), \(\delta_h\), and \(\Phi_h\) represent the coefficients of vorticity, divergence and geopotential height field in the Galerkin sense. For a variable \(\varphi\), \(\varphi_h = (\varphi_{h,1}, \ldots, \varphi_{h,n})\) denotes the vector of coefficients of the Galerkin representation

\[
\varphi(x) = \sum_{i=1:n} \varphi_{h,i} b_{h,i}(x)
\]

of the function \(\varphi\) corresponding to basis functions \(b_h\). Furthermore, we denote by \(M_i\) the mass matrix, \(S_i\) the stiffness matrix, \(C\) the Coriolis matrix with \(c_{i,j} = (f_{C}b_i, b_j)\), and \(B_i\) appropriately defined mixed variants of the mass matrix. Finally \(F_i\) are the right hand sides obtained by multiplication with the mass matrix. The Helmholtz decomposition related equations in (8.11 are discretized in a similar manner by the finite element method. To summarize, the solution process in PLASMA-P is given by the following algorithm.

\begin{algorithm}{8.3.1} (Discretization of PLASMA-P)
Let initial values for \(v\), and \(Ψ\) be given.
\begin{enumerate}
\item Initialize \(ζ\) and \(δ\) from the data.
\item Compute trajectories by solving the ordinary differential equation

\[
\dot{x} = v(x, t)
\]

for each grid point (see section 7.4.1).
\item Compute the right hand sides in (8.12), using the trajectories.
\item Solve the system of equations (8.12).
\item Solve the Poisson problems

\[
S_2\psi_h = M_{21}\zeta_h, \quad S_2\chi_h = M_{21}\delta_h,
\]

and derive the new velocity vector \(v_h\) from

\[
M_3v_h = R\psi_h + G\chi_h,
\]

where \(R\) is the rotation matrix and \(G\) the divergence matrix from the finite element discretization, resp.
\end{enumerate}
\end{algorithm}

In this scheme, the prognostic variables are the geopotential \(Φ\), the vorticity \(ζ\), and the divergence \(δ\). This is convenient, since these are all scalar values and secondly, the divergence and vorticity can immediately be used as refinement criteria in the following \(L^2\)-sense:

\[
\eta_r = \int_T \delta^2 + \zeta^2 \, dx.
\]
Fig. 8.9. Simulation of Rossby waves with PLASMA-P. A zonal jet, located at 30°N with wind maximum of 40 m s⁻¹ is disturbed by a 7 km high mountain, located at 30°N, 0°E. (Left), grid (right) geopotential height field (in m² s⁻², right) after 20 days (grid type CNT(1041 : 428) [km])

The rationale behind this choice is that a flow is less well approximated in areas of high turbulence. Now, turbulence occurs in areas of high vorticity or divergence (in first order approximation).

PLASMA-P has been validated mainly with the help of analytical test cases, described in section 8.5. Figure 8.9 shows the adaptively refined grid and a snapshot for a test case with a jet over an isolated mountain.

8.3.2 Velocity Formulation

The development branch building on the velocity formulation of the shallow water equations is called PLASMA-FEMmE, since it builds on the development described in [193, 194]. The basic formulation of the shallow water equations employed by PLASMA-FEMmE is given by

\[
\frac{dv_s}{dt} + \nabla_s \Phi + f_C (e_r \times v_s) = 0, \tag{8.13}
\]

\[
\frac{d\Phi}{dt} + \Phi \nabla_s \cdot v_s = 0. \tag{8.14}
\]

These equations are discretized in time by a semi-implicit semi-Lagrangian 2-time-level scheme introduced in [388]. This 2-time-level scheme has been derived from a 3-time-level scheme, therefore some of the terms have to be evaluated at the midpoint of the trajectory. We denote by:

- \( \varphi^+ \) a constituent \( \varphi \)'s value at a grid point at an unknown time \( t + \Delta t \),
- \( \varphi^- \) a constituent \( \varphi \)'s value at an upstream position at a known time \( t \), and
- \( \varphi^* \) a constituent \( \varphi \)'s value at a trajectory midpoint extrapolated to time \( t + \Delta t/2 \).

With this notation, and a splitting of the difference of free surface of geopotential height \( \Phi \) and orography \( \Phi_H \) (i.e. the effective geopotential height) into a constant basic state \( \Phi \) and a disturbance \( \hat{\Phi} \),

\[
\hat{\Phi}(x, t) + \Phi = \Phi(x, t) + \Phi_H(x),
\]
we write the semi-Lagrangian time discretization of (8.13) and (8.14) by
\[
\frac{\mathbf{v}^+ - \mathbf{v}^-}{\Delta t} = - \left[ \omega \nabla_s (\Phi^+ + \Phi_H^+) + (1 - \omega) \nabla_s (\Phi^- + \Phi_H^-) \right] - 
- [f_C(e_r \times \mathbf{v})]^*,
\]
\[
\frac{\hat{\Phi}^+ - \hat{\Phi}^-}{\Delta t} = - \hat{\Phi}^* \nabla_s \cdot \mathbf{v}^* - \hat{\mathbf{F}} \cdot \left[ \omega \nabla_s \cdot \mathbf{v} + (1 - \omega) \nabla_s \cdot \mathbf{v}^- \right].
\] (8.15)
(8.16)

The next step in the discretization process is to apply the tangential divergence operator to (8.16) and insert the result into (8.15), which gives an equation
\[
- \Delta_s \hat{\Phi}^+ + c_H \hat{\Phi}^* = \text{RHS}.
\] (8.17)

This is a Helmholtz-type equation with
\[
c_H = \frac{1}{\mathbf{F}(\omega \Delta t)^2},
\]
and \text{RHS} is a right hand side composed of known (and extrapolated) terms:
\[
\text{RHS} = \frac{1 - \omega}{\omega} \Delta_s \Phi^- + c_H \Phi^- + 
+ \Delta_s \Phi_H^+ + \frac{1 - \omega}{\omega} \Delta_s \Phi_H^- - 
- c_H \mathbf{F} \Delta t \nabla_s \cdot \mathbf{v}^* - c_H \Delta t \Phi^* \nabla_s \cdot \mathbf{v}^* + 
+ \frac{1}{\omega} \nabla_s \cdot [f_C(e_r \times \mathbf{v})]^*.
\]
\[\Delta_s\] is the Laplace-Beltrami operator on the sphere. Thus, we are left with discretizing a Helmholtz-problem on the sphere. This is accomplished by a finite element method with linear basis functions, yielding a second order spatial discretization suitable for the second order semi-Lagrangian time discretization described above.

The grid is maintained by the grid generation library \text{amatos}. The initial grid is the truncated icosahedron (see section 3.5). \text{amatos} is used to refine uniformly and locally according to a refinement criterion similar to that in PLASMA-P (previous section). Some preliminary validation has been carried out, using the Williamson test cases [415]. The scheme shows good conservation properties and does not need an artificial diminishing viscosity stabilization. The results reported here are personal communication by Thomas Heinze.

Williamson test case 2 consists of a quasi-stationary geostrophically balanced zonal flow (see fig. 8.10). It is not well suited for adaptive refinement, since it does not comprise any local features. However, it is interesting to observe that a well balanced refinement criterion effectively does not react and maintains a static grid for the whole simulation. This, together with time series of the $l^2$-error of relative vorticity and geopotential height is depicted in
fig. 8.11. The two different adaptively refined grids (of type CNT[7:11]) that remain almost unchanged during the computation are shown in fig. 8.12.

Test case 5 is well suited for adaptive refinement. An isolated mountain generates Rossby waves in an initially uniform jet-like zonal wind field. Figure 8.13 shows the geopotential height field and some wind vectors after 15 days of simulation. A corresponding grid is depicted in fig. 8.14. Again, similar conservation properties as in test case 2 can be observed.

8.4 Wave Dispersion

This section describes a finite element/finite volume discretization of the shallow water equations in spherical coordinates for wave dispersion (or tsunami modeling) in the ocean. The model accomplishing this task is based on the

![Fig. 8.11. Time series showing the accuracy of PLASMA-FEMmE with Williamson test case 2. (Upper left) number of vertices; (upper right) evolution of the $l^2$-error of geopotential height; (lower left) evolution of the $l^2$-error of relative vorticity; (lower right) evolution of the ratio of first moments (mass)]
Fig. 8.12. Adaptively refined meshes corresponding to the Williamson test case 2. (Left) initial grid; (right) grid after first (and only) adaptation step in time step 57. Shadings indicate different grid levels

shallow water equations, given below. In the description, we closely follow the presentation of the non-adaptive but unstructured grid version of the code by Hanert et al. [187] and Danilov [106].

We start with stating the equations for the surface elevation of a fluid (similar to (C.0.13)), the sea surface height (SSH) $h$ and the velocity $v$ in Eulerian form:

\[
\frac{\partial h}{\partial t} + \nabla \cdot [(h + H)v] = 0,
\]

\[
\frac{\partial v}{\partial t} + v \cdot \nabla v + f_C(\mathbf{e}_r \times v) = -g \nabla h.
\]

Fig. 8.13. Williamson test case 5 situation after 15 days; geopotential height as isolines/colors, wind as arrows
Fig. 8.14. A typical adaptively refined mesh corresponding to Williamson test case 5. Shadings indicate different refinement levels, contour lines represent the hill.

Here, $H$ denotes a reference water depth (layer height), and – as usual – $f_C$, $e_r$, $g$, and $\nabla$ are the Coriolis parameter, the outward unit vector on the sphere, the gravitational acceleration constant, and the 2D (spherical) gradient operator, resp.

Since discretization is achieved by a finite element method, we have to derive the variational formulation.

$$
\sum_l \int_{\tau_l} \left( \frac{\partial h}{\partial t} b_h - (h + H) \mathbf{v} \cdot \nabla b_h \right) \, d\mathbf{x} + \\
+ \sum_l \int_{\partial \tau_l} (h + H) b_h \mathbf{v} \cdot \mathbf{n}_l \, d\Gamma = 0 \quad (8.20)
$$

$$
\sum_l \int_{\tau_l} \left( \frac{\partial \mathbf{v}}{\partial t} b_{\mathbf{v}} - (\nabla \cdot (\mathbf{v} b_{\mathbf{v}})) + f_C (e_r \times \mathbf{v}) b_{\mathbf{v}} + g \nabla h b_{\mathbf{v}} \right) \, d\mathbf{x} + \\
+ \sum_l \int_{\partial \tau_l} (\mathbf{v} \cdot \mathbf{n}_l) b_{\mathbf{v}} \, d\Gamma + \sum_l \int_{\partial \tau_l} [\mathbf{v}][a(b_{\mathbf{v}})] \, d\Gamma = 0, \quad (8.21)
$$

where we used the following notation:

- $b_{\mathbf{v}}$, and $b_h$ represent the basis functions corresponding to the variables $\mathbf{v}$, and $h$.
- $\int_{\tau_l} (\cdot) \, d\mathbf{x}$ is the integral over an element $\tau_l$.
- $\int_{\partial \tau_l} (\cdot) \, d\Gamma$ is the boundary integral over the edges of $\tau_l$.
- $\mathbf{n}_l$ is the outward normal vector at each edge of $\tau_l$.
- $[\cdot]$ represents a jump condition at each edge of a cell $\tau_l$ in the following way: let $\tau_l$ and $\tau_k$ (let $l > k$) be the two adjacent cells of the edge in question, and let $g$ be a constituent. Then

$$
[g] = g|_{\tau_l} - g|_{\tau_k}.
$$
Thus, the last term in (8.21) represents a continuity condition on $v$ over edges.

The finite element spaces are chosen in the following way: $h$ (and consequently $H$) is discretized by a linear conforming Lagrange polynomial $P^1$ element, while $v$ is discretized by a non-conforming linear Lagrange polynomial $P^1_{nc}$ element. The node positions and an example basis function are depicted in fig. 8.15.

Time discretization is realized by a simple, explicit leap-frog (centered difference in time) scheme. Additionally, the equations are augmented by a viscosity term $\nabla \nu \nabla v$ partly for stabilization, partly because of more realistic results, especially when considering the reflection and propagation of short waves at boundaries. Additionally a bottom drag term $(r + C_d |v|) v / H$ is added in order to include realistic effects of tsunami run-up behavior. Here $r$ is a Rayleigh friction coefficient, and $C_d$ is the drag coefficient. So, the discretized equations read

$$v^+ = v^- + \Delta t f_v^-, \quad (8.22)$$
$$h^+ = h^- + \Delta t f_h^-, \quad (8.23)$$

where $f_v^-$ and $f_h^-$ are all known terms and can be interpreted as flux-terms. Filtering has to be applied in order to prevent splitting modes in the leap-frog scheme.

The adaptive mesh refinement is achieved differently to the approaches in the previous sections. In this method only one grid is employed. Constituents at known times have to be interpolated, once new nodes are inserted. Interpolation is performed at the (generic) order of accuracy provided by the finite element approximation.

The procedure is summarized by the following algorithm:
Fig. 8.16. The computational domain for the wave dispersion model. Contours show the depth of the domain

Algorithm 8.4.1 (Discretization of wave dispersion model)

Let initial values for \( \mathbf{v} \), and \( h \) be given, and let \( \mathcal{G} \) be the computational domain.

1. Triangulate the domain to obtain a triangulation \( \mathcal{T} = \bigcup_{l=1}^{M} \mathcal{T}_l \) with elements \( \tau_l, l = 1 : M \), and edges \( \gamma_k, k = 1 : L \).
2. Discretize surface height and velocity by \( P^1 \) conforming and \( P^1_{nc} \) non-conforming finite element expansions, respectively.
3. Initialize \( h^- \) and \( \mathbf{v}^- \) by initial values.
4. Compute \( h^+ \) and \( \mathbf{v}^+ \) by (8.22) and (8.23).
5. Determine a refinement criterion and adapt the mesh correspondingly.
6. IF mesh has changed, THEN:
   a) Interpolate \( h^- \) and \( \mathbf{v}^- \) at the new nodes
   b) GO TO step 4.

Note that with a fixed time step the algorithm is not unconditionally stable (in contrast to the semi-Lagrangian schemes, described in sections 8.1 and 8.3). Therefore the Courant number is controlled during the computation.

Some preliminary results from this algorithm are given below. The algorithm is applied to a wave dispersion problem in an idealized setting. A rectangular domain and a bottom with a slope form the computational domain (see fig. 8.16).

A radial sine shaped wave form is initialized. The wave initially propagates symmetrically radially. Lateral boundaries reflect the waves and the bottom topography leads to a non-symmetric wave form after some time steps. Figure 8.17 shows this behavior.
8.5 Test Cases

In this section we intend to collect a number of test cases either used in the examples or cited from other publications and especially useful for adaptive computations.

Test Cases for Linear Advection

We start our collection with some test cases for linear advection of a passive tracer in 2D and 3D. These test cases comprise constant divergence-free flow fields as well as divergent fields. They are especially suited for testing conservation properties of advection schemes.

We consider the (conservative) flux form of the advection equation for a mixing ratio or a density $\rho$:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0 \quad \text{in } \bar{\mathcal{G}},$$  \hspace{1cm} \text{(8.24)}

where the computational domain is defined by $(\mathbf{x}, t) \in \bar{\mathcal{G}} = \mathcal{G} \times I$, $\mathcal{G} \subset \mathbb{R}^d$, $d = 2, 3$, the spatial domain, $I = [0, T]$ the time interval, and $\mathbf{v} = \mathbf{v}(\mathbf{x}, t) \in \mathbb{R}^d$ is a given multi-dimensional flow. In order to be properly defined, (8.24) needs
additional specifications for the computational domain $\bar{G}$, for the boundary conditions $\rho_b(x,t)$, for the initial condition $\rho_0(x)$, and the wind $v(x,t)$. For all 2D tests we will assume

$$\bar{G} = \bar{G}_{2D} = \{(x,t) : x \in [-0.5, 0.5]^2; t \in [0,T]\},$$

where $T$ is specified in the test case. For the 3D tests, we assume

$$\bar{G} = \bar{G}_{3D} = \{(x,t) : x \in [0,1]^3; t \in [0,T]\}.$$ 

Additionally, in 3D we will denote with $\varphi_H$ the horizontal component of a vector-valued entity $\varphi \in \mathbb{R}^3$. Thus, if $\varphi = (\varphi_x, \varphi_y, \varphi_z)^T$ then $\varphi_H = (\varphi_x, \varphi_y)^T$.

We will assume all lateral boundaries to be outflow boundaries. That means, mass can escape from the computational domain, but cannot enter. It remains to be specified the initial condition, the wind, and the time interval.

**Example 8.5.1 (Diagonal Wind)**

*The 2D case:*

$$\rho_0(x,t) = \begin{cases} 
  1, & \text{in } \{x : |x - (-0.25, -0.25)| < 0.15\}, \\
  0, & \text{else.}
\end{cases}$$

$$v(x,t) = (1, 1)^T \cdot s, \text{ with a scaling factor } s = 0.36361^{-5}$$

$$T = 18,000 \text{ s}$$

*The 3D case:*

$$\rho_0(x,t) = \begin{cases} 
  1, & \text{in } \{x : |x - (0.25, 0.25, 0.5)| < 0.15\}, \\
  0, & \text{else.}
\end{cases}$$

$$v_H(x,t) = (1, 1)^T \cdot s, \text{ with a scaling factor } s = 0.36361^{-5}$$

$$v_z(x,t) = 0$$

$$T = 18,000 \text{ s}$$

**Example 8.5.2 (Converging Wind)**

*The 2D case:*

$$\rho_0(x,t) = \begin{cases} 
  1, & \text{in } \{x : |x - (-0.25, 0.0)| < 0.15\}, \\
  0, & \text{else.}
\end{cases}$$

$$v(x,t) = ((0.75, 0)^T - x^T) \cdot \frac{\omega}{2}, \text{ with } \omega = 0.36361^{-4}$$

$$T = 18,000 \text{ s}$$

*The 3D case:*

$$\rho_0(x,t) = \begin{cases} 
  1, & \text{in } \{x : |x - (0.25, 0.5, 0.5)| < 0.15\}, \\
  0, & \text{else.}
\end{cases}$$

$$v(x,t) = ((1.25, 0.5, 0.5)^T - x^T) \cdot \frac{\omega}{2}, \text{ with } \omega = 0.36361^{-4}$$

$$T = 18,000 \text{ s}$$
The following slotted cylinder test case has been adopted from a test case originally proposed in [425]. Note that the angular velocity factor \( \omega \) is chosen such that 96 time steps of 1,800 s are exactly one revolution. This test case represents a hard test for the numerical diffusion of an advection scheme, since it defines a non-smooth tracer function. Adaptive grids help to reduce numerical diffusion substantially while maintaining efficiency. Additionally, grid refinement increases the order of approximation near the steep gradients. High order approximations tend to show oscillation effects (over and under shooting) near these “singularities”.

**Example 8.5.3 (Slotted Cylinder Test Case)**

The 2D case:

\[
\rho_0(x, t) = \begin{cases} 
1, & \text{in } \{x: |x - (-0.25, 0.0)^T| < 0.15, \\
& \text{and } x \notin [-0.28, -0.22] \times [0, -0.5], \\
0, & \text{else.}
\end{cases}
\]

\[
v(x, t) = ((-y, x)^T \cdot \omega), \text{ with } \omega = 0.36361^{-4}
\]

\[T = 172,800 \text{ s}\]

The 3D case:

\[
\rho_0(x, t) = \begin{cases} 
1, & \text{in } \{x: |x - (0.25, 0.5, 0.5)^T| < 0.15, \\
& \text{and } x \notin [0.22, 0.28] \times [0.5, 1.0] \times [0, 1.0], \\
0, & \text{else.}
\end{cases}
\]

\[
v_H(x, t) = ((-y, x)^T \cdot \omega), \text{ with } \omega = 0.36361^{-4}
\]

\[
v_z(x, t) = 0
\]

\[T = 172,800 \text{ s}\]

The accelerating wind test case has been adopted and extended to 3D from [218]. Additionally, the scaling has been adjusted such that after 72 time steps of 1,800 s a full revolution is complete, and analytic error evaluation can be performed. As an example we give snapshots of both the 2D and 3D simulation of the accelerated wind field test case in figs. 8.18 and 8.19 resp.

**Example 8.5.4 (Accelerating Wind)**

The 2D case:

\[
\rho_0(x, t) = \begin{cases} 
1, & \text{in } \{x: |x - (-0.25, 0.0)^T| < 0.15, \\
& \text{and } x \notin [-0.28, -0.22] \times [0, -0.5], \\
0, & \text{else.}
\end{cases}
\]

\[
v(x, t) = \begin{cases} 
(-y, x)^T \cdot \omega, & \text{if } (x, y) \in \{x: x \leq 0\}, \\
(-y, x)^T \cdot \omega(1.5 \cos(2\phi) + 2.5), & \text{if } (x, y) \in \{x: x > 0, y > 0\}, \\
(-y, x)^T \cdot \omega(1.5 \cos(2\psi) + 2.5), & \text{if } (x, y) \in \{x: x > 0, y \leq 0\},
\end{cases}
\]

with \( \omega = 0.36361^{-4}, \phi = \arctan\left(\frac{y}{x}\right), \psi = \arctan\left(\frac{-y}{x}\right) \).

\[T = 129,600 \text{ s}\]
The 3D case:

\[
\rho_0(x, t) = \begin{cases} 
1, & \text{in } \{x : |x - (0.25, 0.5, 0.5)^T| < 0.15, \\
& \text{and } x \not\in [0.22, 0.28] \times [0.5, 1.0] \times [0, 1.0]\}, \\
0, & \text{else}.
\end{cases}
\]

\[
v_H(x, t) = \begin{cases} 
(-y, x)^T \cdot \omega, & \text{if } (x, y) \in \{x : x \leq 0\}, \\
(-y, x)^T \cdot \omega(1.5 \cos(2\phi) + 2.5), & \text{if } (x, y) \in \{x : x > 0, y > 0\}, \\
(-y, x)^T \cdot \omega(1.5 \cos(2\psi) + 2.5), & \text{if } (x, y) \in \{x : x > 0, y \leq 0\}, \\
\end{cases}
\]

with \(\omega = 0.36361^{-4}, \phi = \arctan\left(\frac{y}{x}\right), \psi = \arctan\left(\frac{-y}{x}\right)\).

\[
v_z(x, t) = 0
\]

\(T = 129,600 \text{ s}\)

Note that since we deal with linear advection, all the previous test cases can be solved analytically and, therefore, are very well suited for accuracy investigation. However, convergence tests need a smooth density distribution function, since even high order schemes usually do not achieve high order convergence near discontinuities in the data. Therefore, a fifth test case is given.

---

**Fig. 8.18.** Accelerating wind test case in 2D: Initial field of tracer distribution (left) intermediate state after 36 time steps (center) and final step after 72 time steps (right)

**Fig. 8.19.** Accelerating wind test case in 3D: Initial field of tracer distribution (left) intermediate state after 36 time steps (center) and final step after 72 time steps (right). The 0.95 iso-surface is shown
Fig. 8.20. Initial field of tracer distribution for convergence test case, contour lines in 0.1 unit steps

for convergence tests. We rotate the initial situation by just 1/4 revolution. The initial configuration is depicted in fig. 8.20.

**Example 8.5.5 (Convergence test case)**

*The 2D case:*

\[
\rho_0(x, t) = \begin{cases} 
4 \cos \left(\frac{r \pi}{2R}\right), & \text{in } \{x : r < R\} \quad \text{with } r = \|x - (-0.25, 0.0)^T\|, \ R = 0.15; \\
0, & \text{else.}
\end{cases}
\]

\[
v(x, t) = (-y, x)^T \cdot \omega, \ \text{with } \omega = 0.36361^{-4}
\]

\[T = 43,200 \ \text{s}\]

*The 3D case:*

\[
\rho_0(x, t) = \begin{cases} 
\cos \left(\frac{r \pi}{2R}\right), & \text{in } \{x : r < R\} \quad \text{with } r = \|x - (0.25, 0.5, 0.5)^T\|, \ R = 0.15; \\
0, & \text{else.}
\end{cases}
\]

\[
v_H(x, t) = (-y, x)^T \cdot \omega, \ \text{with } \omega = 0.36361^{-4}
\]

\[
v_z(x, t) = 0
\]

\[T = 43,200 \ \text{s}\]

**Test Cases for the Shallow Water Equations**

The shallow water equations serve as a first mile stone in the development of dynamical cores for atmospheric modeling systems. Most authors use the well known *Williamson test cases*, published in [415] with a set of references [222]. We are not going to repeat these test cases here.

One weakness of the Williamson test cases is that there is no time-dependent dynamical test case with an analytical solution. One such test case was proposed by Läuter, Handorf, and Dethloff recently (hereafter referred to as [LHD05]) [257]. Test cases for the spherical shallow water equations are grouped into four categories:
analytical solutions to simplified equations;
• steady state solutions to the complete equations;
• unsteady analytical solutions to the full equations with a prescribed solution forcing technique;
• close-to-reality initial data and comparison to high resolution model reference solutions.

[LHD05] add a new class of analytical test cases that does not need a forcing technique, but should converge under correct numerical treatment of the continuous equations. The derivation of test cases is based on the transformation of known solutions to the shallow water equations in non-rotating frames of reference to the rotating sphere. We cite two of the given test cases here, omitting the technique of transformation, which is described in detail in [LHD05] and has been adopted from [134, 311].

We start with repeating the formulation of the shallow water equations in a spherical rotating frame.

\[
\begin{align*}
\frac{dv}{dt} + \nabla s \Phi + f_C n \times v + \frac{|v|^2}{|x|} n &= 0, \\
\frac{d\Phi}{dt} + \Phi \nabla_s \cdot v - \nabla_s \cdot (v \Phi_0) &= 0, \\
v \cdot n &= 0.
\end{align*}
\] (8.25)

As usual, \( v \) denotes the tangential velocity, \( n \) the unit outward normal vector, \( f_C \) the Coriolis term, \( \Phi \) the geopotential height and \( \Phi_0 \) the orography. A simple test case is given by the axially symmetric flow:

**Example 8.5.6 (Unsteady axially symmetric flow)**

We first define

• \( a \in \mathbb{R}^3, |a| = 1 \) a fixed symmetry axis, and \( k_1, k_2 \) constants;
• \( u : [-1, 1] \rightarrow \mathbb{R} \) a smooth integrable velocity profile, i.e. \( u \) satisfies

\[
\int_{[-\frac{\pi}{4}, \frac{\pi}{4}]} |\tan xu^2(\sin x)| \, dx < \infty.
\]

• \( \varphi_t \) a coordinate transformation function from the non-rotating to the rotating frameset given by

\[
\varphi_t(x) = (x \cdot b_i)e_i, \quad \varphi_t^{-1}(y) = (y \cdot e_i)b_i,
\]

where

\[
b_1(t) = \cos(\Omega t)e_1 + \sin(\Omega t)e_2, \quad b_2(t) = -\sin(\Omega t)e_1 + \cos(\Omega t)e_2, \quad b_3(t) = e_3,
\]

with \( \Omega \) the earth's rotational velocity.
Then the pair \((v_A, \Phi_A)\) is a solution to the shallow water equations (8.25) with
\[
v_A(x, t) = u(\varphi_t(x) \cdot n(x)) \frac{\varphi_t(x) \times n(x)}{|\varphi_t(x) \times n(x)|} - \Omega \times x,
\]
\[
\Phi_A(x, t) = \int_0^{\arcsin(\varphi_t(x) \cdot n(x))} \tan y u^2(s) \, dy + \frac{1}{2}(\Omega \cdot x) + k_1,
\]
\[
\Phi_{A,0}(x) = \frac{1}{2}(\Omega \cdot x) + k_1.
\]
We used \(\Omega = \Omega e_3\), and \(\Phi_{A,0}\) denotes the orography. Note that \(k_1\) and \(k_2\) are not specified and can be chosen arbitrarily.

**Remark 8.5.7** With the analytical solution given in test case 8.5.6, the corresponding analytical functions for the vorticity \(\zeta\) and the divergence \(\delta\) can be given (\(r\) denotes the sphere’s radius):
\[
\zeta(x, t) = \frac{u(\varphi_t(x) \cdot n(x)) \varphi_t(x) \cdot n(x)}{r \sqrt{1 - (\varphi_t(x) \cdot n(x))^2}}
- \frac{1}{r} u'(\varphi_t(x) \cdot n(x)) \sqrt{1 - (u'(\varphi_t(x) \cdot n(x)))^2} - 2\Omega \cdot n(x),
\]
\[
\delta(x, t) = 0.
\]
The second test case cited from [LDH05] is a jet stream. A sketch of that test case is given in fig. 8.21.

**Example 8.5.8** (Unsteady jet stream)

We start with some definitions:

- Let \(c \in \mathbb{R}^3\) with \(|c| = 1\) a fixed rotation axis.
- Let \(u_0, u_1 \in \mathbb{R}\) flow velocities, \(\theta_0, \theta_1 \in [-\pi/2, \pi/2]\) latitudinal angles, and \(k_1, k_2 \in \mathbb{R}\) constants.
- Define the velocity and geopotential of a solid body rotation \(v_R\) and \(\Phi_R\) as
\[
v_R(x, t) = u_0 \varphi_t(c) \times n,
\]
\[
\Phi_R(x, t) = \frac{1}{2} \left[-(u_0 \varphi_t(c) \cdot n + \Omega \cdot x)^2 + (\Omega \cdot x)^2\right] + k_1.
\]
- With \(c_0 = u_1 \exp(4(\theta_0 - \theta_1)^{-2})\), let the jet velocity profile \(u_J\) be
\[
u_J(x) = \begin{cases} c_0 \exp((\arcsin(x - \theta_0) \arcsin(x - \theta_1))^{-1}) \text{ for } \theta_0 \leq \arcsin(x) \leq \theta_1, \\
0 \text{ else}. \end{cases}
\]
- Let \(a = \frac{u_0 c + r \Omega}{|u_0 c + r \Omega|}\), \(r\) the sphere’s radius, and a second velocity profile defined by
\[
u_S(x) = (u_0 c + r \Omega) \sqrt{1 - x^2}
\]
for \(x \in [-1, 1]\).
With these pre-definitions and the notation $s = \sin(y)$ the following system is a solution to the shallow water equations (8.25):

$$v(x, t) = v_R(x, t) + u_J(\varphi_t(a) \cdot n) \frac{\varphi_t(a) \times n}{|\varphi_t(a) \times n|},$$

$$\Phi(x, t) = \Phi_R(x, t) - \int_0^{\arcsin(\varphi_t(a) \cdot n)} \tan(y) \left( u_S(s) u_J(s) + u_J^2(s) \right) dy + k_2,$$

$$\Phi_0(x) = \frac{1}{2} (\Omega \cdot x)^2 + k_2.$$  \hspace{1cm} (8.26)

Note that with the definition of an analytical solution in test case 8.5.8 again vorticity and divergence can also be given. Indeed, they are identical to those in test case 8.5.6. Note further that [LDH05] gives also solutions to the shallow water equations, formulated in spherical coordinates for convenience. The constants used in the test cases are

- $k_1$: is the minimum geopotential height and should be chosen large enough to impose suitable layer of air above orography. In [LDH05] this is $k_1 = 2000$.  
- $k_2$: is the bottom geopotential height and usually set to $k_2 = 0$.  
- $u_0$: is the mean (westerly) velocity and in [LDH05] set to $u_0 = 20 [m s^{-1}]$.  
- $u_1$: is the maximum velocity in the jet and set to $u_1 = 38 [m s^{-1}]$ in accordance with the Williamson test cases.  
- $\theta_0$: is the southern latitude bounding the jet and set to $\theta_0 = \frac{\pi}{7}$.  
- $\theta_1$: is the northern latitude bounding the jet and set to $\theta_1 = \frac{\pi}{2} - \theta_0$.  

We close this section with a list of common test cases for the shallow water equations.

- Galewsky and coworkers propose a barotropic instability test case in [159].  
- We mentioned before the well known Williamson test suite of seven test cases including solid body rotation, quasi stationary waves (Rossby-Haurwitz waves), instationary flow and near realistic initial value problems with reference solutions [415, 222].  
- A balanced initial state of wave number 1, proposed by McDonald and Bates [282].  
- Solid body rotation test cases with an inclination angle [398, 414].  
- Zonal wind steady state solution by Browning and coworkers [74].
As an outlook, we mention the three dimensional test cases for baroclinic dynamical cores of future numerical modeling systems, proposed by Jablonowski and by Polvani and coworkers. These test cases are especially interesting, since they provide a frame of reference for full 3D models. The Jablonowski-Williamson test case describes the evolution of a baroclinic wave from a balanced (and perturbed) initial state [221]. It is well suited for adaptive methods, since the wave stays spatially local. A somewhat similar test case, also starting from an initially balanced flow field that is perturbed, has been proposed by Polvani and coworkers for primitive equation solvers [324]. Earlier test cases for 3D dynamical cores have been proposed by Held and Suarez [195] and Boer and Denis [64].

For fully featured models, the only way to evaluate their quality is by comparing modeling results to measurements. For an adaptive global atmospheric model such an evaluation is documented by Boybeyi and coauthors [69].
Conclusions

Adaptive atmospheric modeling is a truly interdisciplinary approach in scientific computing. We have seen several tools from applied mathematics, theoretical physics, computer science and even pure mathematics interacting in order to achieve efficiency, accuracy and robustness for adaptive algorithms in atmospheric modeling. This chapter aims in reviewing the methods and evaluating their respective usefulness. From this we try to derive a path for future research directions.

9.1 Evaluation of Methods

It is indeed challenging and fascinating at the same time how broad the spectrum of mathematical methods that help solving adaptive atmospheric modeling. We touched

- graph theory and computational geometry in the mesh generation part and parallel mesh partitioning;
- functional analysis with Sobolev spaces in the finite element discretization of differential equations;
- variational formulations of conservation laws;
- geometric and quadrature integration in the discretization of conservative semi-Lagrangian advection;
- classical analysis and set theory in the derivation of space-filling curves;
- complexity theory when deriving optimal search strategies and parallel algorithms.

Above all is the aim of applying adaptive multi-scale methods for true life applications.

In essence, adaptive methods have gained a technical state of maturity that allows for serious applications in atmospheric modeling. Efficient data structures for handling adaptively refined meshes are available and prove to
be useful. When applying adaptive mesh refinement, it is important to adhere to two principles

- hierarchical data structures, and
- neighborhood preservation.

There are powerful refinement and de-refinement strategies available, either for quadrilateral/hexahedral refinement and for triangular/tetrahedral refinement. The coupling with automatic mesh generation, however, has not yet been integrated. Most mesh handling and generating tools start from a given coarse grid.

Parallelization, though much more involved with adaptive methods, is possible and practical. Many tools show efficiency up to at least moderate numbers of processors. It is important to consider

- dynamic,
- parallelizable, and
- locality preserving

load balancing strategies.

The construction of differential operators for irregular mesh structures reveals some problematic issues. Mesh independency is not yet fully achieved. There remains some more research to be conducted in order to find a consistent way to deal with mesh irregularities. This is of great importance, since gradients in many discretizations of 3D atmospheric and oceanic circulation play a crucial role for the stability of the whole dynamical core.

The development of discretization schemes for conservation laws with adaptive mesh refinement has gained a state of maturity over the past 15 years that makes complex simulations feasible. We introduced finite volume schemes, discontinuous Galerkin schemes and conservative semi-Lagrangian schemes for solving the dynamic equations of atmospheric circulation. For adaptive methods to be useful, discretization techniques for conservation laws must be capable of implementing

- unstructured
- direction independent

discrete evolution operators. While the first two methods (FV and DG) are well established and proved their effective conservation properties, they lack the advantageous stability property making them truly mesh independent. Semi-Lagrangian schemes on the other hand are stable and therefore more mesh independent, but it is more cumbersome to achieve conservation.

The examples demonstrated in chap. 8 prove the effectiveness of adaptive methods in atmospheric modeling. However, it remains to be shown that adaptivity yields the same advantage in fully complex atmospheric applications. Some test cases, tailored for adaptive modeling, have been proposed. Yet, more interesting and challenging cases are to be developed. Adaptive methods can reveal their effectiveness in cases with
• localized features,
• large scale differences,
• time dependent local phenomena.

This work has tried to give a fair overview of the state of the art of adaptive atmospheric modeling. Some impressive results have been achieved. The interesting work has just begun. Now, we can start of thinking to solve problems that could not be tackled with the methods available so far. Studying fine-scale interaction with global processes, resolving orography realistically, including fully dynamic convective processes, to name a few, appear to be feasible options now.

9.2 Road Map for the Next Five Years

After more than 15 years from first papers on adaptive atmospheric modeling [368, 369, 125], this overview of the current state of adaptive methods for atmospheric modeling reveals: It is by far not a completed field of research, in contrary the real work just started. There are important tasks ahead of the community of adaptive atmospheric modelers and this section tries to outline a road map for the next five years. The reader is asked to be forbearing with the author, since even with some experience in adaptive modeling, there is no looking glass into the future. However, several options for further development are obvious.

Consistent Numerical Methods

There is still a great need for consistent numerical methods. That means the quest for the ultimate numerical scheme that is mesh independent, efficient, stable, accurate and preserves the structural properties of the continuous problem, is still open.

We came much closer to this goal in the recent years. Discontinuous Galerkin methods promise to preserve most of the continuous properties of the underlying conservation laws even for high order accuracy. However, there is still some work required to find a stable time integration scheme that fits well to the DG scheme and is stable for large Courant numbers. On the other hand, several authors presented (stable) semi-Lagrangian schemes that were modified such that conservation properties hold [360, 419, 421].

Moreover, it is necessary that the numerical schemes used are robust enough to handle the case of model changes. That means, when refining locally, the assumptions on approximations become more and more violated. In this case, the mathematical model has to be changed or in some cases, the characteristics of the model equations change (e.g. the low mach number flow, where some intermediate state between hyperbolic and parabolic equations is covered). The numerical method must be chosen such that no numerical artifacts result from this fact.
And finally, the numerical method should be able to hide grid effects even in large to very large ratios of refinement to de-refinement mesh size. In order to exploit adaptive methods, one wishes to be able to refine locally down to fine scales, while a global coarse background model generates the necessary background data. The numerical scheme should be stable for this kind of situations.

**Refinement Criteria**

Adaptive methods are only as good as the refinement criteria driving the adaptation strategy. This fact cannot be over exaggerated. So far, refinement is controlled by

- error proxies;
- physical criteria, like gradients, divergence or vorticity values;
- mathematical error estimates based on discretization or truncation error estimates for the dynamic equations.

However, what if there are several constituents that all play a role in the accuracy of the simulated situation, but react differently in distinct regions of the computational domain? How to treat equations, where the error can propagate in time? How to detect sub-grid scale processes that may trigger a refinement once they are relevant in magnitude (but cannot be resolved on a coarse mesh)?

Since atmospheric modeling is always concerned with multi-scale interaction of processes with a multitude of constituents involved, there is not much research from other communities that could serve as template for adoption. Research efforts will be necessary to solve the problem of finding rigorous efficient and robust refinement criteria.

**Sub-Grid Parametrization**

As mentioned above, atmospheric modeling is concerned with multi-scale phenomena, spanning a range of scales that cannot be resolved even with adaptive grid refinement. Therefore, some processes will always be captured by intelligent parametrization techniques. However, current parametrization is most often tuned to certain mesh resolutions. Thus, a re-design of parametrization schemes such that they become grid-independent or can be adapted to changing grid sizes in a functional way, is necessary.

This effort, as painful as it might be, promises some fruitful insight. Since such mesh-independent parametrization will probably be derived from first principles, a more consistent and physical approach may eventually be the result.

It is - on the other hand - naive to assume that after so many years of intense research activity in parameterizations that the community will easily take the challenge of rethinking many of the methods used so far. Therefore,
this research issue will probably remain on the agenda far beyond the next five years.

Additionally, the evolution of promising stochastic approaches to sub-grid parametrization are not prepared to adaptive mesh sizes. It may well be that stochastic modeling is much more mesh independent than the physics based parametrization, but this has not yet been investigated.

**Computational Efficiency**

Since adaptive methods almost always involve some kind of indirect addressing, irregularity in data structures and changing memory requirements, there is still a great demand in more efficient computational methods. Additionally, the first adaptive atmospheric models have to catch up with the vast optimization efforts that went into gaining speed in operational models and corresponding numerical cores. We outlined several methods in this work that try to address this issue, however, it still remains to be shown that adaptive methods really outperform well tuned uniform and regular grid based models.

**Realistic and Useful Applications**

Up to now, the state of the art in adaptive atmospheric modeling allowed for the development of dynamical cores and simplified models only. The only operational adaptive model OMEGA may be considered an exception, however, it also does not exploit the full potential of adaptive methods.

I can see applications in severe weather forecasting, where fronts of thunderstorms can be resolved accurately (without parameterizing the local convection). More generally, it is not hard to foresee that meso-scale convection cells can be detected by a sub-grid criterion and then adaptively resolved in a non-hydrostatic model.

For realistic assessment of orographic influence especially near mountainous regions (Föhn wind, lee waves, etc.) a locally refined mesh is an obvious option. Even in large scale models, this kind of refinement might yield much more realistic flow fields than a parameterized orographic forcing.

There are applications of adaptive methods for air quality and pollution modeling of localized emitters. However, in the inverse modeling situation, where one uses an adjoint model for solving the problem of determining a source from field values, adaptive methods can greatly increase the local accuracy, as shown in sect. 8.2. It remains to be shown, that this approach remains useful and applicable for realistic situations with reacting constituents, etc.

Scale-interaction is another promising field for adaptive methods. Large scale processes triggering small scale features that again have an important influence/feed-back to the large scale or small scale processes triggering large scale characteristic features play an important role in the atmosphere and oceans. For example, the tide induced small scale jets that are caused by coastal topography in the Weddel sea are supposedly responsible for significant
interaction and deep sea water formation in the Weddel eddy [315]. While
the tide wave is a truly global process with a characteristic length scale of
\( \frac{1}{2} r_{\text{Earth}} \) (\( r_{\text{Earth}} \) the earth’s radius), the jets have a diameter of some hundreds
of meters.

A realistic tsunami forecasting model will face the same ratio in magni-
tudes: The wave caused by deep sea seismic activity and the corresponding
characteristic length scale is probably a large fraction of an ocean basin or sev-
eral thousand kilometers. On the other hand the Tsunami unfolds its power
due to shallow water and an amplification of energy due to convergence. In
these regions, characteristic length scales of a few meters play a role.

**Rigorous Scale Analysis**

In order to be usefully solved by adaptive modeling techniques, it is necessary
to know the scale regions for which certain approximations are valid. Furth-
more, one can think of methods that extend the equations once they refine
below a certain level. In order to control the refinement criterion sensibly,
a rigorous scale analysis has to be conducted for the common mathematical
models of atmospheric dynamics [274].

**Coupling of Sub-System Models**

Adaptive grids pose additional difficulties for coupling several models. While
regular grids allow for more or less straight forward interpolation techniques
that have to adhere to conservation and structural properties of the underly-
ing equations, it is less obvious how to handle this problem with adaptively
refined, temporally changing grids. Interpolation is always an option, however,
the error has to be controlled.

More generally, what is the interface to an adaptive model? Up to now,
atmospheric modelers think in grid structures. This is not a useful concept
for adaptive methods. Here, we want to think in functional values, regardless
of the mesh. However, if a mesh is no longer explicitly provided, error bounds
need to be given together with interchanged values, since otherwise no useful
assessment of the overall accuracy is possible.

**Pre- and Post-Processing**

Most of the tools in atmospheric modeling that can be used for visualization,
archiving, data-base storage, distribution and interchange of data, etc., are
made for uniform and quadrilateral meshes. A first approach to reconnect
adaptive methods to the toolbox available for pre- and post-processing is
interpolation. In many cases this will suffice to yield satisfactory results (e.g.
in visualization, where the visual effect can be maintained even if the error
of interpolation is in the order of a few percent). However, if accurate data is
required, an interpolation might not be the right option. It will be necessary to re-design many of the pre- and post-processing tools for irregularly structured data.

Again, this is not necessarily a major drawback, since it might be worth to re-design several of the aged methods anyway. However, it is a time factor that has to be taken into account.

**Vision**

It is the author’s vision that in some years, we will not think in grids anymore. We will be talking about results, error measures, experiments. The numerical methods will be able to resolve the model equations accurately up to a user supplied tolerance. Results will be handled mesh-independently, more generally meshes will be what they ought to be, just tools for representing data. We will come back to the type or research in the early days of atmospheric sciences, that is we will be concerned about formulating model equations and well posed initial and boundary conditions instead of being concerned about the correct numerical solution of such equations. Adaptive methods will have gained a state of robustness and efficiency that they can be used in modular experimental software environments, allowing for interactive playful research.
Some Basic Mathematical Tools

This appendix contains a brief section on basic mathematical tools that are used throughout this book. We denote by \( x = x_{x_0,t_0}(t) \) the position at time \( t \) of a fluid particle that started at position \( x_0 \) at time \( t_0 \). We will often omit the subscripts. Let \( \ddot{x} = \ddot{x}_{x_0,t_0}(t) \) denote the trajectory representing the path of a particle that starts at position \( x_0 \) at starting time \( t_0 \) up to time \( t > t_0 \). In fact \( \ddot{x} \) marks the path between \( x_0 \) and \( x \). \( V(t) \) shall denote a material volume moving with the fluid and containing exactly the same particles over time.

Obviously, the derivative in time of a position in space is given by the velocity

\[
\frac{dx(t)}{dt} = v(x,t).
\]

The time derivative of a particle’s property \( \psi \) (the material property) is denoted by \( \frac{d\psi}{dt} \), called the material derivative. Since we assume a dense distribution of particles and the property \( \psi \) being defined for each particle, we obtain a scalar field \( \psi(x,t) \). For the material derivative, applying the chain rule and taking into consideration that the material derivative tracks individual particles, we have

\[
\frac{d\psi}{dt} = \frac{\partial \psi}{\partial t} + v \nabla \psi.
\] (A.1)

We will use the following theorems, that are recalled here for the reader’s convenience. Proofs can be found in [410].

**Theorem A.0.1 (Divergence Theorem)**

For any volume \( V \subset \mathbb{R}^d \) with piecewise smooth closed surface \( S \) and any differentiable vector field \( \mathbf{u} \) the following integral relation holds:

\[
\int_V \nabla \cdot \mathbf{u} \, dx = \int_S \mathbf{u} \cdot \mathbf{n} \, ds,
\]

where \( dx \) is the spatial differential, \( ds \) is the surface differential and \( \mathbf{n} \) is the outward unit normal on \( S \).
Theorem A.0.2 \textbf{(Stokes theorem)}

Let $C$ be a closed curve and $S$ a piecewise smooth surface bounded by $C$. Then, for any differentiable vector field $u$

\[
\int_C u \cdot dx = \int_S (\nabla \times u) \cdot n \; ds,
\]

where $dx$ is the curve differential along $C$, and $n$ as above the outward unit normal (right handed in traversal direction).

With Stoke’s theorem one has a representation of $u$ in terms of a gradient:

Corollary A.0.3 \textbf{(Potential)}

If $u$ is irrotational, i.e. $u$ satisfies $\nabla \times u = 0$, then there exists a scalar field $\psi$ such that

$u = \nabla \psi$.

$\psi$ is called the potential.

Another representation of a vector field $u$ is given by the

Theorem A.0.4 \textbf{(Helmholtz representation)}

For any bounded continuous vector field $u$ that vanishes at infinity there exists a scalar field $\psi$ and a divergence free vector field $w$ (i.e. $w$ satisfies $\nabla \cdot w = 0$), such that

$u = \nabla \psi + \nabla \times w$.

The following theorem is used in section 7.4.

Theorem A.0.5 \textbf{(Transport theorem)}

For any material volume $V(t)$ and differential scalar field $\psi$ we have ($v$ the velocity):

\[
\frac{d}{dt} \int_{V(t)} \psi \; dx = \int_{V(t)} \left( \frac{\partial \psi}{\partial t} + \nabla \cdot (v \psi) \right) \; dx. \tag{A.2}
\]
B

Metrics for Parallelizing Irregularly Structured Problems

Naturally, all the metrics for measuring the parallel performance of a non-adaptive program apply similarly to an adaptive program. For parallel metrics one may consult the standard literature (e.g. [205]). We recall the most important performance metrics for use in this book:

**Definition B.0.6 (Standard Metrics)**

1. The computing **time** for a single process is denoted by $t_p$, where $p$ is the process id. We omit the dependence of $t_p = t_p(N)$ on the problem size $N$ unless otherwise stated.

2. The **wall clock time** $t_{total}$ for a parallel process is defined by the maximum time among each individual process times$^1$:

$$ t_{total} := t_P := \max_{p=1:P} t_p. $$

3. The **speedup** $s_P$ for $P$ processors is defined as the inverse factor of decrease in computing time with parallel processors:

$$ s_P := \frac{t_1}{t_P}, $$

where $t_1$ and $t_P$ is the wall clock time on 1 and $P$ processors respectively.

4. The parallel **efficiency** $E_P$ for $P$ processors is defined as the percentage of real achieved speedup over theoretically optimal speedup. In other words, the efficiency is the speedup scaled by $\frac{1}{P}$.

$$ E_P := \frac{t_1}{P \cdot t_P}. $$

$^1$ Note that we assume that all processes start at the same time. More generally, the wall clock time is the time from program start to program termination with parallel execution in between these two points.
5. The **scale-up** $S_l$ (l scaling factor) measures the behavior of an algorithm with increasing number of processors $P$ and similarly increasing problem size $N$, such that each process maintains a constant problem size. The main influence limiting the scale-up is inter-process communication.

$$S_l := \frac{t_P(N)}{t_{l.P}(l \cdot N)}.$$ 

In order to measure the quality of partitions we need to establish metrics for the usefulness of a partitioning strategy. As mentioned in chapter 5, we need to fulfill several requirements. First, a partitioning strategy needs to be efficient. Since partitioning is not our main task, all the work that goes into calculating an (almost) optimal partitioning is overhead. We will account for this overhead. A good partitioner will cause as little overhead as possible.

**Definition B.0.7 (Partitioning Overhead)**

The partitioning overhead $O_{\text{part}}$ is the ratio of time for computing the partitioning $t_{\text{part}}$ over total time $t_{\text{total}}$, and is a number $0 < O_p < 1$.

$$O_{\text{part}} := \frac{t_{\text{part}}}{t_{\text{total}}}, \quad (B.1)$$

**Remark B.0.8** Note that $O_{\text{part}}$ can be a value that depends on the number of processors. Since the partitioning might scale with another slope as the overall program.

Since the overall performance is utterly dependent on good load balancing (note that the parallel wall clock time can only be as short as the largest individual process), we define the load balancing parameter:

**Definition B.0.9 (Load Balancing Parameter)**

Let $N_p$ be the problem size of processor $p$. The load balancing parameter is defined by

$$\mathcal{L} := \frac{\max_p N_p}{\min_p N_p}$$

Note that $\mathcal{L} \geq 1$ and that optimal load balancing is achieved when $\mathcal{L} = 1$.

But not only load balancing is crucial for efficient parallelization, data communication is also to be minimized. If inter-process communication has to take place, then we assume that it takes place over interfaces that have distinct processes on either side. Therefore, we introduce two metrics for evaluating the partitioning with communication issues in mind.

**Definition B.0.10 (Edge Cut)**

The edge cut $\mathcal{E}$ is defined as the number of edges (2D) or faces (3D) resp. $f_{pi}^{pj}$ that form a boundary between processors relative to the total number of edges/faces $f_{total}$:

$$\mathcal{E} := \frac{\# f_{pi}^{pj}}{\# f_{total}}$$
Finally, since re-partitioning has to take place frequently, we also have to consider the cost for moving cells between processors.

**Definition B.0.11 (Re-Partitioning Cost)**

Let $M$ be the total number of mesh cells and let $M_{ij}$ be the number of cells that are moved from processor $p_i$ to processor $p_j$. Then the re-partition cost $R$ is defined by

$$R := \frac{\sum_{i,j=1}^{P} M_{ij}}{M}.$$ 

Note that $0 \leq R \leq 1$, because every cell can be moved at most once (the net effect is counted). If the unknowns are associated to processors (instead of the cells) then the definition of $R$ has to be modified accordingly.

Some interesting comments on load balancing and metrics concerned with mesh partitioning can be found in [197, 198]. It is noteworthy that the edge cut is often a misleading measure for the quality of a partitioning method, since the computation is influenced by the true communication cost which is only weakly related to the edge cut.
C

Rotating Shallow Water Equations in Spherical Geometries

In sect. 7.1 we derived the shallow water equations from basic conservation principles. Here we extend the equations to be valid on the rotating sphere. Not all applications in atmospheric modeling need to account for the rotation, but since our emphasis is on global modeling, we have to consider it.

We consider a rotating frame of reference with angular velocity $\Omega$. An observer in an inertial frame of reference, which is the frame for the derived equations so far, and a second observer in a rotating frame of reference will have different views of changes to a vector $\xi = (\xi_1, \xi_2, \xi_3)$ in the following way:

$$\left[ \frac{d\xi}{dt} \right]_I = \left[ \frac{d\xi}{dt} \right]_R + \Omega \times \xi.$$

The subscripts $I$ and $R$ refer to the inertial and rotating frames resp. Using this relation, we see immediately that for the velocity $v$ we have that $v_I = v_R + \Omega \times x$. Some elementary transformations yield for the velocity

$$\left[ \frac{dv_I}{dt} \right]_I = \left[ \frac{dv_R}{dt} \right]_R + 2\Omega \times v_R + \Omega \times (\Omega \times x) + \frac{d\Omega}{dt} \times x,$$

where $x$ is the position vector, and the three additional terms correspond to the Coriolis acceleration, the centripetal acceleration and the acceleration due to changes in the angular velocity itself, resp. By assuming a constant angular velocity and transformations to treat the centripetal acceleration as a force potential, included in the interior force term, the only remaining acceleration is the Coriolis term. With this, the momentum equation (7.7) in a rotating coordinate frame can be written

$$\frac{dv_R}{dt} + 2\Omega \times v_R = -\frac{1}{\rho} \nabla p + \nabla \Phi + \mathcal{F},$$

where $\Phi$ is a force potential, and $\mathcal{F}$ represents the other forces as above. One can show, that all terms on the right hand side (under certain assumptions,
like Newtonian fluid, etc.) are independent of the frame of reference, and therefore, remain unchanged compared to (7.7).

The material derivatives of scalar quantities remain unchanged in a rotating frame of reference. However, the individual components of the material derivative are not independent. It holds

\[
\left[ \frac{d\eta}{dt} \right]_I = \left[ \frac{d\eta}{dt} \right]_R - (\Omega \times \mathbf{x}) \cdot \nabla \eta,
\]

and

\[
\mathbf{v}_I \cdot \nabla \eta = (\mathbf{v}_R + \Omega \times \mathbf{x}) \cdot \nabla \eta,
\]

for a scalar \( \eta \). Following the presentation in [194], we start deriving the shallow water equations for rotating spherical geometries from the following set of equations, the momentum equation equipped with the Coriolis acceleration (as above) and the continuity equation from sect. 7.1:

\[
\frac{d\mathbf{v}_R}{dt} = -\frac{1}{\rho} \nabla p - 2\Omega \times \mathbf{v}_R - \mathbf{g}, \tag{C.1}
\]

\[
\frac{dH}{dt} = -H \nabla \cdot \mathbf{v}_R. \tag{C.2}
\]

We have simplified the force terms above to the internal gravitational force and used the notation \( \mathbf{g} \) for the vector perpendicular to the earth’s surface (in each point) of length \( g \), the gravitational constant. With \( \mathbf{e}_r \) the outward normal unit vector on the sphere, we can write \( -\mathbf{g} = \mathbf{e}_r g \).

We use the following results for vector calculus on spherical manifolds [135]. Let \( \xi = (\xi_1, \xi_2, \xi_3) \) be a vector in \( \mathbb{R}^3 \), then \( \xi_s \) denotes the tangential components of \( \xi \) at each point \( \mathbf{x} \in S \subset \mathbb{R}^3 \), where \( S \) is the 2D sphere in 3D space. We have

\[
\xi_s = \xi - \mathbf{e}_r (\mathbf{e}_r \cdot \xi).
\]

Furthermore, for the tangential gradient and the tangential divergence we have

\[
\nabla_s \eta = \nabla \eta - \mathbf{e}_r (\mathbf{e}_r \cdot \nabla) \eta,
\]

\[
\nabla_s \cdot \xi = \nabla \cdot \xi - \mathbf{e}_r \cdot (\mathbf{e}_r \cdot \nabla) \xi.
\]

The radial differential for a scalar \( \eta \) is defined by \( \frac{\partial \eta}{\partial r} = \mathbf{e}_r \cdot \nabla \eta \). Using these notations we can immediately transform the material derivative for the tangential components of vectors on the sphere to obtain

\[
\frac{d\xi_s}{dt} = \frac{\partial \xi_s}{\partial t} + (\mathbf{v} \cdot \nabla)\xi_s.
\]

We define the geopotential height \( \Phi \) by

\[
\Phi = g(h - h_b) = gH.
\]
With these tools at hand, we obtain the spherical form of the continuity equation (C.2):
\[
\frac{d\Phi}{dt} + \Phi \nabla_s \cdot \mathbf{v}_s = 0.
\]  
(C.3)

In order to derive the tangential components of the spherical momentum equation we have to transform the Coriolis acceleration first. Additionally we use the Coriolis parameter \( f_C = 2|\Omega| \sin \phi \), where \( \phi \) is the latitude of the fluid element’s position, in order to reflect the position dependent Coriolis acceleration on the surface of the sphere. After some additional scaling arguments and transformations that are not repeated here, we obtain the momentum equation of the spherical shallow water equations:
\[
\frac{d\mathbf{v}_s}{dt} + \nabla_s \Phi + f_C (\mathbf{e}_r \times \mathbf{v}_s) = 0.
\]  
(C.4)

There are a number of different formulations of the spherical shallow water equations. We will report on some for reference and convenience. Williamson and coworkers [415] give several of the shallow water equations which we do not want to repeat here, since that reference is well known. Their formulations include the flux form SWE, the advective form of the SWE and the vorticity divergence form of the SWE, a stream function based formulation and a constrained 3D form. We want to mention some of the more recent formulations here. Note, that all of the mentioned formulations are pairwise equivalent. So, the choice of formulation depends very much on the numerical discretization scheme used.

We start with repeating the above result as a first form of the shallow water equations (SWE). In the following examples we will always put the momentum equation first.

**Definition C.0.12** (SWE with tangential operators and geopotential height)
\[
\frac{d\mathbf{v}_s}{dt} + \nabla_s \Phi + f_C (\mathbf{e}_r \times \mathbf{v}_s) = 0, \\
\frac{d\Phi}{dt} + \Phi \nabla_s \cdot \mathbf{v}_s = 0
\]

Heinze and Hense combine this form of the SWE with a splitting of \( \Phi \) into a constant term and a fluctuation [194]. The second form follows the presentation in [273], and uses the fluid layer height instead of the geopotential height.

**Definition C.0.13** (SWE with fluid layer height)
\[
\frac{d\mathbf{v}_s}{dt} + g \nabla h + f_C (\mathbf{e}_r \times \mathbf{v}_s) = 0, \\
\frac{dH}{dt} + H \nabla \cdot \mathbf{v}_s = 0.
\]
The conservative form of the shallow water equations as in [273]:

**Definition C.0.14 (SWE in conserving form according to Majda)**

\[
\frac{\partial (Hv_s)}{\partial t} + \nabla \cdot \left( Hv_s \otimes v_s + \frac{g}{2} H^2 I \right) = -(f_C e_r \times v_s + g \nabla h_B) H, \\
\frac{\partial H}{\partial t} + \nabla \cdot (Hv_s) = 0.
\]

We have used \( \otimes \) as the tensor product. If we assume a flat bottom, a plane domain, and no rotation, the right hand side of the momentum equation in definition C.0.14 becomes zero and the shallow water equations are also called *isentropic fluid flow equations*.

Giraldo uses a 3D cartesian coordinate formulation of the shallow water equations and introduces Lagrange multipliers to restrict the velocity vectors to the sphere’s surface in [167]. We cite his compact form of the shallow water equations:

**Definition C.0.15 (SWE in Cartesian coordinates according to Giraldo)**

The SWE are given as a system of equations, defined by

\[
\frac{\partial \Phi}{\partial t} + \nabla \cdot (\Phi v^T) = S(\Phi),
\]

with

\[
\Phi = \begin{bmatrix} \Phi \\ \Phi u \\ \Phi v \\ \Phi w \end{bmatrix}, \quad v = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \quad S(\Phi) = -\Phi \nabla \Phi - f_C (e_r \times \Phi u) + \mu e_r.
\]

The last term in \( S(\Phi) \) is a force that restricts the 3D velocity vector to the sphere’s surface with a Lagrange multiplier \( \mu \) determined in the discretized equations.

Ringler and Randall use a formulation that includes the kinetic energy and absolute vorticity [339]:

**Definition C.0.16 (SWE with kinetic energy and absolute vorticity)**

\[
\frac{\partial v}{\partial t} + \zeta e_r \times v + \nabla [k + g (h + h_s)] = 0, \\
\frac{\partial h}{\partial t} + \nabla \cdot (hv) = 0,
\]

with \( h_s \) the fluid surface height, \( h \) the fluid layer thickness, \( k \) the kinetic energy, and \( \zeta \) the absolute vorticity,

\[
\zeta = f_C + e_r \cdot \nabla \times v.
\]

A scalar formulation is also given in [339]:
Definition C.0.17 (Scalar SWE with kinetic energy and absolute vorticity)

\[
\frac{\partial \delta}{\partial t} - \mathbf{e}_r \cdot \nabla \times (\zeta \mathbf{v}) + \Delta [k + g(h + h_s)] = 0, \\
\frac{\partial \zeta}{\partial t} + \nabla \cdot (\zeta \mathbf{v}) = 0, \\
\frac{\partial h}{\partial t} + \nabla \cdot (h \mathbf{v}) = 0,
\]

with \( \delta = \nabla \cdot \mathbf{v} \) the divergence, and \( \zeta \) the absolute vorticity. The velocity vector has to be reconstructed by a Helmholtz decomposition.

A scalar form using the vorticity-divergence formulation with a Helmholtz decomposition is used by Läuter [255] (see sect. 8.3).

Definition C.0.18 (SWE in vorticity divergence form with tangential operators)

\[
\frac{d \zeta}{dt} + \zeta \delta + \delta f_C + \mathbf{v} \cdot \nabla s f_C = 0, \\
\frac{d \delta}{dt} + \Delta_s \Phi - \zeta f_C + (\mathbf{e}_r \times \mathbf{v}) \cdot \nabla s f_C + J(\mathbf{v}) = 0, \\
\frac{d \Phi}{dt} + \delta(\Phi - \Phi_0) - \mathbf{v} \cdot \nabla s \Phi_0 = 0.
\]

Note that the operators \( \nabla s \), and \( \Delta_s \) together with the material derivative \( \frac{d}{dt} \) are tangential operators. \( J \) is a matrix specified in sect. 8.3, \( \zeta = \nabla \times \mathbf{v} \) is the vorticity and \( \delta = \nabla \cdot \mathbf{v} \) the divergence, \( \Phi_0 \) the orography.

Yet another scalar formulation with the fluid height, the potential vorticity and the divergence is used by Pesch [316].

Definition C.0.19 (SWE in height divergence form with tangential operators)

\[
\frac{\partial \delta}{\partial t} + \nabla \left[ h \zeta \mathbf{e}_r \times \mathbf{v} + \nabla \left( g(h + h_s) + \frac{1}{2} \mathbf{v} \cdot \mathbf{v} \right) \right] = 0, \\
\frac{\partial h}{\partial t} + \nabla \cdot (h \mathbf{v}) = 0, \\
\frac{\partial h \zeta}{\partial t} + \nabla \cdot (h \zeta \mathbf{v}) = 0.
\]

Pesch uses the potential vorticity \( \zeta \).

Blikberg uses the shallow water equations in spherical geometry in the following compact form [63]:

Definition C.0.20 (SWE in spherical compact form)

\[
\frac{\partial \mathbf{q}}{\partial t} + \partial_\phi \mathbf{f} + \partial_\lambda \mathbf{g} = \mathbf{s},
\]
where \((\lambda, \phi)\) are the latitudinal and longitudinal coordinate pair, \(\partial_\nu\) is the partial derivative in \(\nu\) direction and

\[
\mathbf{q} = \begin{bmatrix} \hat{\lambda} \\ \hat{\mu} \\ \hat{\nu} \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} a(\hat{\mu}u) \\ a(\hat{\mu}^2) + b(\hat{\lambda})^2 g \\ a(\hat{\mu}uv) \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} c(\hat{\mu}v) \\ c(\hat{\mu}uv) \\ c(\hat{\mu}^2) + d(\hat{\lambda})^2 g \end{bmatrix},
\]

with \(\hat{\lambda} = \lambda \cos \phi\), \(a = (r_{\text{earth}} \cos \lambda)^{-1}\), \(b = (2r_{\text{earth}} \cos^2 \lambda)^{-1}\), \(c = r_{\text{earth}}^{-1}\), and \(d = (2r_{\text{earth}} \cos \lambda)^{-1}\). The right hand side is given by

\[
\mathbf{s} = \begin{bmatrix} 0 \\ (f_C + c u \tan \lambda) \hat{\mu}v \\ -(f_C + c v \tan \lambda) \hat{\mu}u - d(\hat{\lambda})^2 g \tan \lambda \end{bmatrix}.
\]

Note that this set of equations does not involve orography.
## List of Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>First occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{I} = [0, T]$</td>
<td>time interval</td>
<td>2.3</td>
</tr>
<tr>
<td>$\mathcal{G} \in \mathbb{R}^d$, $\mathcal{G}_h \in \mathbb{R}^d$</td>
<td>continuous and discrete computational domain</td>
<td>2.3</td>
</tr>
<tr>
<td>$\rho_h$</td>
<td>discrete (scalar) variable</td>
<td>2.3</td>
</tr>
<tr>
<td>$\tau$</td>
<td>element of a triangulation</td>
<td>2.3</td>
</tr>
<tr>
<td>$h$</td>
<td>mesh size parameter, also used as discrete subscript</td>
<td>2.4</td>
</tr>
<tr>
<td>$\eta, \eta_r$</td>
<td>(local) refinement criterion</td>
<td>2.3</td>
</tr>
<tr>
<td>$\theta_{\text{ref}}, \theta_{\text{crs}}$</td>
<td>refinement/coarsening tolerance</td>
<td>2.4</td>
</tr>
<tr>
<td>$\mathcal{T}$</td>
<td>a triangulation</td>
<td>2.4</td>
</tr>
<tr>
<td>$\mathcal{T}_1 \prec \mathcal{T}_2$</td>
<td>hierarchically refined triangulations</td>
<td>3.1.7</td>
</tr>
<tr>
<td>$\vartheta$</td>
<td>inner angle of triangulation</td>
<td>3.1</td>
</tr>
<tr>
<td>$v_i$</td>
<td>vertex of a triangulation</td>
<td>3.3</td>
</tr>
<tr>
<td>$\tau_{3D}$</td>
<td>element/tetrahedron in 3D triangulation</td>
<td>3.4</td>
</tr>
<tr>
<td>$(\lambda, \phi, r)$</td>
<td>spherical coordinates</td>
<td>3.5</td>
</tr>
<tr>
<td>$\partial_x, \partial_t$</td>
<td>first order partial differential operators</td>
<td>4.2.1</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>Laplace’s operator</td>
<td>4.2.2</td>
</tr>
<tr>
<td>$\mathcal{L}, \mathcal{E}, \mathcal{R}$</td>
<td>Load balancing, edge cut, and re-partitioning cost</td>
<td>B</td>
</tr>
<tr>
<td>$G(V,E)$</td>
<td>graph with vertex set $V$ and edges $E$</td>
<td>5.1</td>
</tr>
<tr>
<td>$\Delta x, \Delta t$</td>
<td>spatial and time step</td>
<td>3.5, 4.2.1</td>
</tr>
<tr>
<td>$\cdot</td>
<td>x$, $\cdot</td>
<td>y$</td>
</tr>
<tr>
<td>$b, b_i$</td>
<td>usually a basis function</td>
<td>6.1.3</td>
</tr>
<tr>
<td>$\mathcal{V}, \mathcal{V}_h$</td>
<td>continuous and discrete function space</td>
<td>6.1.3</td>
</tr>
<tr>
<td>$\varepsilon, [\varepsilon]$</td>
<td>true error and error estimate</td>
<td>2.4, 2.6.3</td>
</tr>
<tr>
<td>$\delta_{ij}$</td>
<td>Dirac delta function</td>
<td>6.1.3</td>
</tr>
<tr>
<td>$\mathcal{D}, \mathcal{D}^*$</td>
<td>abstract differential operator, and adjoint</td>
<td>6.1.3, 8.2</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>radial basis function</td>
<td>6.1.4</td>
</tr>
</tbody>
</table>
### List of Notations

#### Notation continued:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>First occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{x} )</td>
<td>trajectory or particle path</td>
<td>A</td>
</tr>
<tr>
<td>( V(t), V )</td>
<td>material volume, control volume</td>
<td>A, 7.2.2</td>
</tr>
<tr>
<td>( \frac{d}{dt} )</td>
<td>material derivative</td>
<td>A</td>
</tr>
<tr>
<td>( \mathbf{n} )</td>
<td>outward unit normal</td>
<td>A.0.1</td>
</tr>
<tr>
<td>( \sigma_{\text{surf}} )</td>
<td>surface stress tensor</td>
<td>7.1.2</td>
</tr>
<tr>
<td>( \mathcal{F} )</td>
<td>force term</td>
<td>7.1.2</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>angular velocity</td>
<td>C</td>
</tr>
<tr>
<td>( \mathbf{v} )</td>
<td>velocity (vector)</td>
<td>C</td>
</tr>
<tr>
<td>( \Phi )</td>
<td>geopotential height</td>
<td>C</td>
</tr>
<tr>
<td>( f_C )</td>
<td>Coriolis parameter</td>
<td>C</td>
</tr>
<tr>
<td>( \nabla_s, \Delta_s )</td>
<td>tangential gradient/divergence, and Laplacian</td>
<td>C, 8.3.1</td>
</tr>
</tbody>
</table>
References

32. J. Behrens, *amatos – Adaptive mesh generator for atmospheric and oceanic simulation*,
   http://www.amatos.info.
44. ——, *amatos: Parallel adaptive mesh generator for atmospheric and oceanic simulation*, Ocean Modelling **10** (2005), no. 1–2, 171–183.


156. S. Frickenhaus, W. Hiller, and M. Best, FoSSI: The family of simplified solver interfaces for the rapid development of parallel numerical atmosphere and ocean models, Ocean Modelling 10 (2005), 185–191.


References


Index

amatos 61
amatos-application 68

abstract vector notation see vector notation
acceleration
  angular velocity 167
centripetal 167
Coriolis 167
adaptive mesh 111
adaptive semi-Lagrangian method 112
adjoint operator 129, 131
admissible patch 29
advection equation
  adjoint 131
  Eulerian flux form 111
  integral form 112
  Lagrangian form 111
advection-dominated problem 110
Akima see gradient
ALBERTA 60
algebraic connectivity 75
algorithm
  adaptive semi-Lagrangian 113
  basic adaptive 14
  basic non-adaptive 13
  finite volume 102
  MPSLM 121
  QR 82
  semi-Lagrangian 112
amatos 133
AMR 15
AMRCLAW 61

AMROC 61
angular acceleration see acceleration
angular momentum 92
angular velocity see velocity
API (application programming interface) 65
atomic mass unit 97
automatic differentiation 129
Avogadro’s number 97
a posteriori 12
a priori 12
balancing patch size 60
bandwidth 57
barotropic flow 98
basis function 83
binary tree see data structure
bisection see refinement
blue refinement see refinement
body forces 94
boundary conditions
  Dirichlet 82
  periodic 62
buckyball see icosahedron
capped sphere 46
ceiling function 73
cell 23, see data structure
cell interface 50
cell-integrated method 116
cell-weighted SLM 119, 125
centripetal acceleration see acceleration
CFL condition 106
chain rule 161
chemistry transport model 124
CHOMBO 61
circum-circle 25
classification see triangulation 26
CLAWPACK 61
closure
  quadrilateral 32
  quadrilateral patch 34
  triangular 30
coarsening
  regular triangular 31
  triangular bisection 29
communication
  data 164
computational complexity 90
  tree search 54
computational geometry 66, 117, 153
computing time see metrics
cconductivity 96
conforming mesh see triangulation
connectivity 51
connectivity matrix 56, see graph
conservation
  differential form 93, 95
  of mass 91, 93
  of momentum 91
conservation law 91, 153
conservation of mass 92
conservation of momentum 95
consistency 104
consistent numerical method 155
continuity eq. see conservation
continuity equation 91
  spherical 169
contour advection method 124
contractive operator 105
ccontrol volume 102
convergence 104
  h-refinement 15
Coriolis acceleration see acceleration
Coriolis parameter 169
coupling 158
Courant number 106, 155
CTM see chemistry transport model
cubature 118
cubed sphere 47
CUBIT 60
Dahlquist’s equivalence theorem 104
data dependency 55
data locality 55
data structure
  binary tree 53, 62
  cell 50
  derived 62
  edge 50
  face 50
  grid patch 59
  index list 62
  list oriented 51
  node 50
  object oriented 52, 62
  oct tree 53
  quad tree 53
  tree 52
  vectorized 54, 63
degree 72
Delaunay
  graph 36
density 92
DG method see discontinuous Galerkin
difference plot 88
diminishing diffusion 134
Dirac delta function 84, 134
Dirichlet boundary cond. see boundary conditions
discontinuous Galerkin method 107, 154
dispersion of tracer 124
divergence 133
tangential 168
divergence free 162
divergence theorem 91, 161
dodecahedron 44
DOF (degree of freedom) 66
domain partitioning 57
dry air 97
dual cell 114
duality relation 131
dynamical core 123
ECMWF 124
edge see data structure
edge cut see metrics
edge flux 103
edge weight see graph
Index 203

efficiency see metrics
eigenvalue 75
elements spectral 61
ELLAM see Eulerian-Lagrangian energy internal 95 kinetic 95 total 95
energy transport advective 97 diffusive 97
equations of state 97 equivalence theorem 104
error h-refinement 15 error in $l^2$-norm 88
Eulerian-Lagrangian method 110 evolution operator 104
face see data structure filament 124
finite difference 55
finite element 65, 66, 140 conforming 84 non-conforming 84 signature 66
finite element method 84
finite volume method 100, 154
flux operator 59 Fortran 90 62
FORTRAN77 51 FoSSI 133
Fourier’s law 96 frame of reference inertial 167 rotating 167
Galerkin approach 83
Galerkin method 82
Galerkin representation 135
gas constant 97
gather 63
Gaussians 87
geopotential height 133, 168
ghost cell 59
global error 105

global mass conservation see mass conservation
globally marked edge see tetrahedron
Godunov’s method 103
gradient 64, 79, 154
Akima’s method 81 convergence 88
finite difference approximation 80
Galerkin method 82
interpolatory method 85
Klucwicz’s method 80
least squares method 82
radial basis function method 86
tangential 168
test case 87
timing result 90
graph 72
associated matrix 74 coarse 77
connectivity matrix 74 degree 72
edge weight 75
fine 77
incidence matrix 74
Laplacian 74 Laplacian matrix 74
matrix properties 75
maximal matching 77
of finite difference mesh 73
of finite element mesh 73
self edge 73
undirected 72
vertex 73
graph theory 153
greedy algorithm 76
green refinement see refinement
Green’s theorem 131
grid atoms 50
grid generator 125
grid patch see data structure

h-refinement 15
$\infty$-norm based 15, 125
equi-distribution 16
halo region 71
hanging node 24
hash table 52
hat function 84
heat 96
heat equation 98
heat flux 96
heating
dissipative 97
external 97
rate 96
Helmholtz decomposition 133, 171
Helmholtz representation 162
hexahedral refinement see refinement
hierarchical basis 18
hierarchical data structure 154
HIRHAM 124
horizontal resolution 124
hp-refinement 18
hydrostatic assumption 99
hyperbolic conservation law 103
icosahedral triangulation see sphere
icosahedron 42
truncated 44, 137
in situ measurement 124
incidence matrix see graph
incompressibility 99
incompressible flow 94
index list see data structure
inner circle 25
integration by parts 107, 131
internal energy 95
internal forces 94
interpolation 64
interpolation problem 86
intersection 66, 116
area 67
lines 67
inverse model 157
inverse multi-quadrics 87
irregular mesh see unstructured
irrotational 162
isentropic equations 170
iterator 59
k-simplex 23
kinetic energy 95
Klucwicz see gradient
Lagrange-Galerkin method 61, 111, 134
Lagrangian particle advection method 124
Laplace operator 55
Laplace-Beltrami operator 137
Laplacian matrix see graph
lat-lon grid 46
reduced 46
Lax equivalence theorem 104
Lax-Richtmyer stability 105
least squares approach 82
length scale 99
Lin-Rood scheme 106
linear programming 116
linked list 52
Lipschitz stability criterion 106
Lipschitz-stability 114
list oriented grid see data structure
load balancing 71, 154, 164
local mass conservation see mass conservation
local support 84
longest edge 25, see refinement
manifold 168
mapping step 120
marked edge see refinement
mass 92
mass conservation
global 114
local 115
mass flux
time averaged 101
mass packet SLM 119
material derivative 161
material volume 161
MATLAB 60
matrix reordering see reordering
maximal matching see graph
mesh-less method 113
metrics 163
dge cut 76, 164
efficiency 163
load balancing 164
partitioning overhead 164
re-partitioning cost 165
scale-up 164
speedup 163
time 163
wall clock 163
molecular weight 97
momentum eq. see conservation
momentum equation 91, 167
MPI 61
MPSLM see mass packet SLM
multi-quadrics 87
multiplicity of eigenvalue 75

neighborhood preservation 154
Neumann stability see von Neumann node see data structure
numerical quadrature 117
numerical trace function 108

object oriented grid see data structure
oct tree see data structure
octahedral triangulation see sphere
octahedron 41
one-step error see truncation error
optimal patch size see balancing
optimization 157
\quad vectorization 54
ordinary differential equation 107
over-determined system 82
ozone depletion 124

p-norm 16
p-refinement 17
parallel computing 71, 107
parametrization 156
particle 161
particle path see trajectory
partitioning
\quad Kernighan-Lin 72, 75
\quad multi-level 72
\quad multi-level method 76
\quad overhead 164
\quad space-filling curves 72
\quad spectral bisection 72, 74
\quad strategy 164
partitioning problem 73
patch of cells 33
perfect gas 97
periodic boundary conditions see boundary conditions
PLASMA 132
Poisson’s equation 82
polar stratosphere 124
Polar Vortex 125
pole singularity 46
polygon

convex 67
oriented 67
potential 162
potential temperature 98
programming interface see API

QMG 60
QR-algorithm see algorithm
quad tree see data structure
quadrature see data structure
quadrature based method 117
quadrature based SLM 118
quadrilateral grid 116
quadrilateral refinement see refinement
quasi-conservative SLM 115
quasi-monotone SLM 115
radial basis function 65, 85
rate of change 93
ratio
\quad volume to surface 71
re-partitioning cost 165
recursive algorithm 117
reduced grid see lat-lon grid refinement
\quad blue 31
\quad general quadrilateral 31
\quad green 30
\quad hexahedral 40
\quad index insertion 37
\quad longest edge 28
\quad marked edge 28
\quad quadrilateral 154
\quad quadrilateral patch 33
\quad regular quadrilateral 32
\quad regular triangular 30
\quad tetrahedral bisection 39
\quad triangular 154
\quad triangular bisection 28
re-partitioning criterion 135, 156
reordering 55, 56
\quad Cuthill-McKee 56
\quad minimum degree 56, 57
resolution
\quad horizontal 124
Riemann problem 103
Rossby wave 138
rotating sphere 97, 167
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runge-Kutta Discontinuous Galerkin method</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>SAMRAI</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>scale interaction</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>scale-up</td>
<td></td>
<td>see metrics</td>
</tr>
<tr>
<td>scatter</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>scattered data</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>self edge</td>
<td></td>
<td>see graph</td>
</tr>
<tr>
<td>self-adaptation</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>semi-Lagrangian method</td>
<td>61, 110, 154</td>
<td></td>
</tr>
<tr>
<td>SFC</td>
<td>57, 153</td>
<td>see space-filling curve</td>
</tr>
<tr>
<td>shallow water equations</td>
<td>98, 100</td>
<td></td>
</tr>
<tr>
<td>scalar formulation</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>signature</td>
<td></td>
<td>see finite element</td>
</tr>
<tr>
<td>simplicial cell</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>SLM</td>
<td></td>
<td>see semi-Lagrangian method</td>
</tr>
<tr>
<td>slope</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>slope limiter</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Sobolev space</td>
<td>83, 131, 153</td>
<td></td>
</tr>
<tr>
<td>soccer ball</td>
<td></td>
<td>see icosahedron</td>
</tr>
<tr>
<td>solenoidal velocity</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>space-filling curve</td>
<td>57, 153</td>
<td></td>
</tr>
<tr>
<td>sparse matrix</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>spectral elements</td>
<td></td>
<td>see elements 61</td>
</tr>
<tr>
<td>speedup</td>
<td></td>
<td>see metrics</td>
</tr>
<tr>
<td>sphere</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icosahedral triangulation</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>octrahedral triangulation</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>rotating</td>
<td>167</td>
<td></td>
</tr>
<tr>
<td>spherical geodesic grid</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>spherical triangle</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>stability</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>CFL criterion</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>stabilization</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>standard position</td>
<td></td>
<td>see tetrahedron</td>
</tr>
<tr>
<td>state variable</td>
<td>95, 97</td>
<td></td>
</tr>
<tr>
<td>stirring</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>Stokes theorem</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>stream function</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>stress tensor</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>structure preservation</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>structured mesh</td>
<td></td>
<td>see triangulation</td>
</tr>
<tr>
<td>sub stepping</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>surface forces</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>tangential component</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>tangential divergence</td>
<td></td>
<td>see divergence</td>
</tr>
<tr>
<td>tangential gradient</td>
<td></td>
<td>see gradient</td>
</tr>
<tr>
<td>Tcl/Tk</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>test case</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>accelerating wind</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>convergence</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>converging wind</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>diagonal wind</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>Jablonowski-Williamson</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>slotted cylinder</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>unsteady axially symmetric flow</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Williamson</td>
<td>137, 147</td>
<td></td>
</tr>
<tr>
<td>test case for gradient</td>
<td></td>
<td>see gradient</td>
</tr>
<tr>
<td>test function</td>
<td>83, 107</td>
<td></td>
</tr>
<tr>
<td>tetrahedron</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bisection</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>globally marked edge</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>standard position</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>thermal conductivity</td>
<td></td>
<td>see conductivity</td>
</tr>
<tr>
<td>thermodynamics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>first law</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>thin plate spline</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>three-time-level method</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>total energy</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>trace function</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>tracer transport</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>trajectory</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>Lagrangian</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>transport theorem</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>tree</td>
<td></td>
<td>see data structure</td>
</tr>
<tr>
<td>tree search</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>triangulation</td>
<td>15, 24</td>
<td></td>
</tr>
<tr>
<td>3D bisection</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>non-admissible</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>admissible</td>
<td>24, 26</td>
<td></td>
</tr>
<tr>
<td>bisection</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>classification of</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>conforming</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Delaunay</td>
<td>27, 36</td>
<td></td>
</tr>
<tr>
<td>hierarchical</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>naming convention</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>orthogonality</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>quadrilateral</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>regular</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>regularly refined</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>structured</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>unstructured</td>
<td>26, 37</td>
<td></td>
</tr>
<tr>
<td>truncation error</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>tsunami</td>
<td>138, 158</td>
<td></td>
</tr>
<tr>
<td>two-time-level method</td>
<td>112</td>
<td></td>
</tr>
</tbody>
</table>
undirected graph see graph
unstructured mesh see triangulation
upwind scheme 104
Vandermonde matrix 82
variational formulation 83
vector data structure see data structure
vector field 161
vector notation 55
vectorization see optimization
velocity 161
angular 167
velocity potential 134
vertex see graph
vertex insertion see refinement
von Neumann stability analysis 106
Voronoi
diagram 36
Voronoi cell 36
vorticity 133
wall clock time see metrics
wave dispersion 138
weak formulation 108
Wedell sea 157
Williamson test cases see test case
wind shear 124
work 96
Yin-Yang grid 47
1. Volumes in the following three categories will be published in LNCSE:
   i) Research monographs
   ii) Lecture and seminar notes
   iii) Conference proceedings

Those considering a book which might be suitable for the series are strongly advised to contact the publisher or the series editors at an early stage.

2. Categories i) and ii). These categories will be emphasized by Lecture Notes in Computational Science and Engineering. Submissions by interdisciplinary teams of authors are encouraged. The goal is to report new developments – quickly, informally, and in a way that will make them accessible to non-specialists. In the evaluation of submissions timeliness of the work is an important criterion. Texts should be well-rounded, well-written and reasonably self-contained. In most cases the work will contain results of others as well as those of the author(s). In each case the author(s) should provide sufficient motivation, examples, and applications. In this respect, Ph.D. theses will usually be deemed unsuitable for the Lecture Notes series. Proposals for volumes in these categories should be submitted either to one of the series editors or to Springer-Verlag, Heidelberg, and will be refereed. A provisional judgment on the acceptability of a project can be based on partial information about the work: a detailed outline describing the contents of each chapter, the estimated length, a bibliography, and one or two sample chapters – or a first draft. A final decision whether to accept will rest on an evaluation of the completed work which should include
   – at least 100 pages of text;
   – a table of contents;
   – an informative introduction perhaps with some historical remarks which should be accessible to readers unfamiliar with the topic treated;
   – a subject index.

3. Category iii). Conference proceedings will be considered for publication provided that they are both of exceptional interest and devoted to a single topic. One (or more) expert participants will act as the scientific editor(s) of the volume. They select the papers which are suitable for inclusion and have them individually refereed as for a journal. Papers not closely related to the central topic are to be excluded. Organizers should contact Lecture Notes in Computational Science and Engineering at the planning stage.

In exceptional cases some other multi-author-volumes may be considered in this category.

4. Format. Only works in English are considered. They should be submitted in camera-ready form according to Springer-Verlag’s specifications.

Electronic material can be included if appropriate. Please contact the publisher.

Technical instructions and/or LaTeX macros are available via
http://www.springer.com/east/home/math/math+authors?SGWID=5-40017-6-71391-0.

The macros can also be sent on request.
General Remarks

Lecture Notes are printed by photo-offset from the master-copy delivered in camera-ready form by the authors. For this purpose Springer-Verlag provides technical instructions for the preparation of manuscripts. See also Editorial Policy.

Careful preparation of manuscripts will help keep production time short and ensure a satisfactory appearance of the finished book.

The following terms and conditions hold:

Categories i), ii), and iii):
Authors receive 50 free copies of their book. No royalty is paid. Commitment to publish is made by letter of intent rather than by signing a formal contract. Springer-Verlag secures the copyright for each volume.

For conference proceedings, editors receive a total of 50 free copies of their volume for distribution to the contributing authors.

All categories: Authors are entitled to purchase further copies of their book and other Springer mathematics books for their personal use, at a discount of 33.3% directly from Springer-Verlag.

Addresses:

Timothy J. Barth
NASA Ames Research Center
NAS Division
Moffett Field, CA 94035, USA
e-mail: barth@nas.nasa.gov

Michael Griebel
Institut für Numerische Simulation
der Universität Bonn
Wegelerstr. 6
53115 Bonn, Germany
e-mail: griebel@ins.uni-bonn.de

David E. Keyes
Department of Applied Physics
and Applied Mathematics
Columbia University
200 S. W. Mudd Building
500 W. 120th Street
New York, NY 10027, USA
e-mail: david.keyes@columbia.edu

Risto M. Nieminen
Laboratory of Physics
Helsinki University of Technology
02150 Espoo, Finland
e-mail: rni@fyslab.hut.fi

Dirk Roose
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
3001 Leuven-Heverlee, Belgium
e-mail: dirk.roose@cs.kuleuven.ac.be

Tamar Schlick
Department of Chemistry
Courant Institute of Mathematical Sciences
New York University
and Howard Hughes Medical Institute
251 Mercer Street
New York, NY 10012, USA
e-mail: schlick@nyu.edu

Mathematics Editor at Springer: Martin Peters
Springer-Verlag, Mathematics Editorial IV
Tiergartenstrasse 17
D-69121 Heidelberg, Germany
Tel.: *49 (6221) 487-8185
Fax: *49 (6221) 487-8355
e-mail: martin.peters@springer.com
Lecture Notes in Computational Science and Engineering


For further information on these books please have a look at our mathematics catalogue at the following URL: [www.springer.com/series/3527](http://www.springer.com/series/3527)

---

**Monographs in Computational Science and Engineering**


For further information on this book, please have a look at our mathematics catalogue at the following URL: [www.springer.com/series/7417](http://www.springer.com/series/7417)
Texts in Computational Science and Engineering


For further information on these books please have a look at our mathematics catalogue at the following URL: www.springer.com/series/5151