Thomas Orgis

Lagrangian Tracer Advection in a Coupled Global Climate Model

Thesis for the degree of Diplom-Physiker, presented to Institute of Physics, University of Potsdam in 2007
English translation (main text, excepting some figure headings and labels) by the author of the German original.
February 12, 2009

Φ

A physicist has data or creates some.

Then he looks at it.
Then he thinks a while.
Then he talks about it.

If he is lucky, someone is paying attention.

Perhaps he writes his gained insights down to paper, so that others with bigger Δt or Δx can follow.

I am a physicist. I am not sure about the insight part, but I am going to write something down.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The main topic of this work is the tracer advection in the wind field of a coupled global climate model. Essential is the global aspect: The domain of interest is the atmosphere of the earth.

As a first approximation. this is the space between two concentric spheres, while the distance between these is small compared to the diameter. Simulation runs of the climate model produce multivariate[1] time series of various quantities (three-dimensional wind, geo potential, temperature, chemical concentrations, ...) on a geographical grid, which is a non-rectangular one, when looking at division of the "real" earth surface. The vertical direction is given as a pressure coordinate whose relation to the height in meters actually varies in time.

My focus is on the transport properties of the spatially and temporally highly variable three-dimensional wind field. The means for the investigation is the computed transport of virtual passive tracer particles, relating to represented air mass. Because of the complex structure of the fields (temporally varying, big gradients with low resolution, long time scale) the individual trajectory is numerically very unreliable. It sensitively depends on parameters and the detailed implementation of the algorithm as well as compiler options and choice of floating point accuracy. Therefore the Ansatz to use a large number of trajectories in an ensemble and reach sensible statements that way.

The movement of large ensembles shall deliver information about transport barriers and important mixing points via statistical analysis, specifically of densities and mixing rations derived from time series.

The work presented here covers the first steps of transport analysis, the basic approach that computes three-dimensional transport of passive tracers based on the modeled wind fields. The basic method is not new. The numerical integration of trajectories in given wind fields is not new.

New is possibly the scale and ignorance of the endeavor. I want to compute global transport in longitude, latitude and height, solely based on the wind fields. There is no consideration of conservation of fluid dynamical quantities (like described in [Stohl1998]) or even general limitation to air layers that are considered to be separated. The further physics as well es the problem of the differing character of horizontal and vertical wind data is put aside consciously. I investigate an uniform global scheme for the transport in the atmosphere which also works at the poles. It is about the question how far such a "blindfolded"[2] approach can lead to meaningful conclusions through large numbers of individual trajectories, using today's computing power.

[1]With the actually used grid that means time series of vectors with $48 \cdot 96 \cdot 23 = 105984$ components, or even 317952 when taking the full wind vector.

[2]☺ Blind in the non-biological sense of being without prejudice – *close-eyes-and-charge* blind, not *against-the-wall* blind or *no-forest-because-of-the-trees* blind.

I will present my method of computing the transport of large particle ensembles, together with the implemented computer software package with the working title PEP-Tracer (available at [PEP-Tracer]). Both represent a in some respects (yet) primitive approach with current tools. A basic feature of the software is the extensibility[3]. The "current tools" are powerful computers – networked to clusters – in big numbers and with plenty of memory and storage[4] as well as object oriented programming, to manage the growth of the software, not just the growth of the hardware.

After presenting the method in detail, I will analyze first long time runs for consistency with the GCM[5]. The consistency is searched in the conservation of the mass distribution of the model (determined via pressure, temperature, moisture and physical volume of regions) during numerical transport from globally distributed starting positions.

I use mainly three languages:

1. English in the main text.
2. Mathematical syntax and symbols in plain formulas as well as in formal representations of algorithms.
3. ANSI/ISO[6] C++ for examples of implementation (excerpts from the software package associated with this work). Embedded comments are in English, like (nearly) all symbol names. The C++ Standard Template Library is also used.

It will occur that I display the same content in all three languages, what hopefully will be seen as insightful consideration from different directions instead of unnecessary redundancy.

1.1 Symbols and Syntax

Not all mathematical syntax that I use is obvious through established convention. Therefore, Table 1.1 is listing several syntactical elements that may need explanation.

mimicking the the ternary operator in C++, I use the following symbolism for Boolean expressions:

$$(\text{[condition]}\ \textbf{?}\ \text{[if-yes-expression]}\ \textbf{:}\ \text{[if-no-expression]})$$

$$\text{[condition]}\left\|\begin{array}{cl} & \text{[if-yes-expression 1]}\\ \textbf{?} & \text{[if-yes-expression 2]}\\ & \dots \\ \hline & \text{[if-no-expression 1]}\\ \textbf{:} & \text{[if-no-expression 2]}\\ & \dots \end{array}\right.$$

The first variant is embedded into other expressions, avoiding space wastage. It is to be read as follows: At its place [if-yes-expression] is used if [condition] is true, otherwise [if-

[3]What I consider to be a *positive* feature.

[4]For today that means multiple GiB of RAM and disk storage in the TiB range. I am sure that these numbers will rather amuse than impress in a few years.

[5]Global Climate/Circulation Model

[6]ISO/IEC 14882:1998 or 14882:2003, also named ANSI C++ ; Important here is the usage of the portable standard language, not a specific compiler and its extensions.

$[a;b]$	closed interval between a and b
$[a;b)$	interval between a and b, open at b
$(a;b]$	interval between a and b, open at a
$(a;b)$	open interval between a and b
$\{a;b\}$	Set with elements a and b
$\mathrm{ganz}\,(x)$	integer part of x ($\mathrm{ganz}\,(3,14) = 3;\ \mathrm{ganz}\,(-3,14) = -3$)
$\mathrm{rund}\,(x)$	value of x rounded to the next integer ($\mathrm{rund}\,(3,14) = 3;\ \mathrm{rund}\,(3,5) = 4$)
$i,\ j,\ k,\ n$	integer numbers, usually used as indices; without further note $i, j, k, n \in \mathbb{Z}$
$a := b$	Assign the value b to the variable a. This notation is used when the value of a can be changed by further assignments and so $a = b$ would not be correct in the mathematical context. It is a kind of temporary "=".

Table 1.1: Definition of additional mathematical Syntax

no-expression] is used. The second variant is just the enlargement of the first, covering whole blocks of expressions. The condition itself is written using usual Boolean syntax, like $(x = 0 \wedge y = 0) \vee z = 0$ ("if x and y are equal to zero or z is equal to zero").

As a matter of principle, I make use of the SI system of units. Some mathematical symbols have consistent meaning throughout this work, Table 1.2 showing these.

There are some common symbols used in the program texts, not coming from the C++ standard. They shall be described here: The data type **pep_t** is used to abstract the basic floating point type, so usually identical to **double** or **float**.

Important named constants are to be found in Table 1.3. These constants are multiply redundant, but support the abstraction[7] and show in a **for** loop immediately if it is about coordinates in space or speeds.

Furthermore, the definition of some vector data types (see Table 1.4) eases the work with winds and positions in the three-dimensional space or the four-dimensional space-time. With these types, also respective vector operations are used, those are listed in Table 1.5

The values of angles, usually for the geographic coordinates λ and β, will be sometimes given in $gradN$ or °O to keep it vivid, but in the mathematics they will be used in radians almost exclusively. Mentioning 90°O in the text is equal to $\lambda = \frac{\pi}{2}$. Also does $\beta \in \left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$ just mean that the latitude is between north and south pole.

1.2 Computer systems that have been used

Computations have been completed on a set of computer system with differing configuration. Hardware as well as Software does influence the results, most importantly the time needed to achieve these. In the text, I will refer to systems named in Table 1.6 by their given name

[7] ☺ as well as reasonably easy porting of the programs to 17 or 42 dimensions

λ, β	longitude (λ) and latitude (β), in degree or radians
h	global height coordinate (meters)
p	global pressure coordinate (Pascal) – most often in place of a height coordinate
T	absolute temperature (Kelvin)
t	time
R	radius of the idealized Earth sphere
$\vec{r} = \begin{pmatrix} \lambda \\ \beta \\ p \end{pmatrix}$	coordinate vector of a particle in the global system (longitude, latitude, pressure)
$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	coordinate vector of a particle in the local system
$(x, y, z)_{\mathrm{lok}}$	explicit denotation of coordinates in the local system (if not given through context)
$\dot{\vec{r}} = \frac{\mathrm{d}}{\mathrm{d}t}\vec{r}$	velocity vector in global system
$\dot{\vec{x}} = \frac{\mathrm{d}}{\mathrm{d}t}\vec{x}$	velocity vector in local system
$r_{\mathrm{b}}(\beta) = R\cos\beta$	radius of a latitude circle at β ($r_{\mathrm{b}}(0) = R$; $r_{\mathrm{b}}(\frac{\pi}{2}) = 0$)
o, a, e	partition parameters for l**o**ngitude, l**a**titude, l**e**vel (height)
η	mass conservation ratio
γ	gravitation constant
g_0	gravitational acceleration at sea level
M	earth mass
m	mass

Table 1.2: Symbols with fixed meaning

Constant	Value	Meaning
`DIMS`	4	number of space-time dimensions (3 space + 1 time)
`SPACEDIMS`	3	number of space dimensions
`LON`	0	index of longitude coordinate
`LAT`	1	index of latitude coordinate
`LEV`	2	index of height/pressure
`TIME`	3	index of time
`SPEEDS`	3	number of directions for wind speeds, in principle identical to `SPACEDIMS`
`U`	0	index of u
`V`	1	index of v
`W`	2	index of ω
`X`	0	index of x or $\dot{x}$
`Y`	1	index of y or $\dot{y}$
`Z`	2	index of z or $\dot{z}$

Table 1.3: Named constants
for enhanced source code readability

Definition	Meaning
typedef pep_t **place**[`DIMS`];	coordinates in space and time, $(\vec{x}, t)$, $(\vec{r}, t)$
typedef size_t `dataindex`[`DIMS`];	indices in the coordinate grid
typedef pep_t **wind**[`SPEEDS`];	wind... $\dot{\vec{x}}$, $\dot{\vec{r}}$
typedef pep_t **spaceplace**[`SPACEDIMS`];	coordinates without time, $\vec{x}$, $\vec{r}$

Table 1.4: Vector data types
for the coordinates and winds

Signature	Meaning
`null_dataindex(n); null_place(n); null_space(n); null_wind(n)`	zero out indices, places and winds $\vec{n} := \vec{0}$
`copy_dataindex(a, b); copy_place(a, b); copy_space(a, b); copy_wind(a, b)`	copy indices, places and winds: $\vec{b} := \vec{a}$
`add_space(a, b, c); add_flatspace(a, b, c); add_place(a, b, c); add_wind(a, b, c)`	addition of places and winds: $\vec{c} := \vec{a} + \vec{b}$
`sub_place(a, b, c); sub_space(a, b, c); sub_flatspace(a, b, c); sub_wind(a, b, c)`	subtraction of places and winds: $\vec{c} := \vec{a} - \vec{b}$
`sadd_space(a, s, c)`	addition of scalar value to each component: $\vec{c} := \vec{a} + s \cdot \vec{1}$
`ssub_space(a, s, c)`	subtraction of scalar value from each component: $\vec{c} := \vec{a} - s \cdot \vec{1}$
`smul_space(a, s, c)`	multiplication with scalar: $\vec{c} := s \cdot \vec{a}$
`smul_flatspace(a, s, c)`	like the above, but only in the first two dimensions (λ, β or x, y)
`sdiv_space(a, s, c)`	division by scalar value: $\vec{c} := \frac{1}{s}\vec{a}$
`sdiv_flatspace(a, s, c)`	like the above, but only in the first two dimensions (λ, β or x, y)

Table 1.5: Vector operations defined as functions. Actually, they are defined as preprocessor macros. A more advanced, but also more complex alternative is the more consequent usage of the object oriented programming paradigm with (template) classes and operators instead.

instead of explaining the configuration each time. A common feature of all systems is that they are operated (mostly) with Free software[8].

- **Adams**: Compaq XP1000 AlphaStation with DEC 21264A (EV67) 64 Bit CPU, 667MHz and 1GiB PC100 RAM (two channels of 256 bit width, with 4MiB still largest L2 cache in this list). Software: Source Mage GNU/Linux with Kernel 2.6.17.1 and gcc-4.1.1 .
- **Neuling**: Laptop IBM ThinkPad X31, 32 bit Pentium-M Processor of the first generation (Banias, 1,4GHz, 1MiB cache), Intel 855PM chip set with 512MiB DDR266 RAM. Software: Source Mage GNU/Linux with Kernel 2.6.20.7 and gcc-4.1.2
- **Atlas**: Opteron 2210 (two cores with 1,8GHz), Broadcom HT1000 chip set, 2GiB DDR2 RAM. Software: openSUSE 10.2 GNU/Linux (x86-64), Kernel 2.6.18.2-34 and gcc-4.1.2 20061115. Main function of this machine is to provide ca. 3,7TiB of disk storage on 15 SATA II hard disks in RAID configuration via NFS to the Grotrian cluster.
- **Grotrian cluster**: Gigabit-Ethernet cluster of nine dual core Opteron systems (AMD Opteron 248, AMD chip set, 2GiB DDR RAM) and six systems with two dual core CPUs each (AMD Opteron 275, nVidia chip set, 2GiB DDR RAM). Altogether there are 42 Processors with clock speed of 2.2GHz. Software: SuSE Linux 10.0, kernel 2.6.13-15, gcc-4.0.2 20050901.
- **Grotrian**: Control node of the cluster with two AMD Opteron 248 CPUs and 2GiB DDR RAM. Software: The same SuSE Linux 10.0 as the rest of the cluster.

Table 1.6: Computer systems

[8] in the sense of freedom, not free beer

2 Data source: ECHO-GiSP GCM

2.1 Model

In the course of the Pole-Equator-Pole project (see [PEP]) my work is closely related with the development of the ECHO-GiSP[1] GCM[2] (Figure 2.1) by Sascha Brand at the Alfred Wegener Institute Potsdam (Article submitted, furthermore presented at the EGU General Assembly: [Brand2007]). Existing components (ECHO-G) are extended with a module for interactive feedback with stratospheric chemistry (computed by MECCA/iSP). Among general statements about tracer transport in the GCM, the question about the influence of the chemistry feedback does play a role in my work.

Some model parameters:

- 39 model pressure levels, including stratosphere (logarithmic distance)
- 40 variable tracers (chemical constituents), e.g. $H_2O, NO_2, OH, HCHO$
- 117 chemical reaction equations for the stratosphere
- Time steps:
  - atmosphere: 15min
  - ocean: 2h
  - coupling ocean/atmosphere: 1d

Inside the chemistry module, semi-Lagrangian transport is used: The densities on grid points are advected for a time step with the wind and then the dislocated densities are interpolated back onto the grid points. To preserve continuity, the result is treated with a correction factor – to prevent loss (or gain) of mass through the interpolation alone.

Sascha Brand provided me with data from two model runs: One with interactively coupled chemistry and one reference run without that feedback. Said feedback mechanism actually consists of changes in the energy influx for the dynamical ore (adsorption/reflection in different air layers depending on the chemical constituents)

2.2 Data

My topic is not the model itself, but the data it produces. Foremost, of course, the wind fields. But other data sets are of interest, too. All data reaching me from ECHO-GiSP shares a common format; from the spectral model the fields are extracted using a grid

[1](**ECH**AM & H**O**PE-**G** **i**ncluding **S**tratosphere by AWI Research Unit **P**otsdam)

[2]Global Circulation/Climate Model, a global model for the atmosphere, here actually AOGCM, meaning a model of atmosphere coupled with ocean (and sea ice)

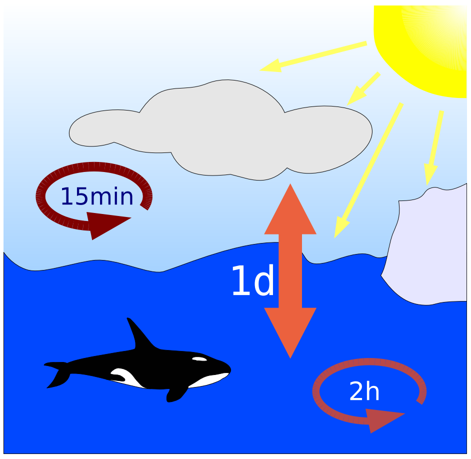


Figure 2.1: Simplified scheme of the ECHO-GiSP AOGCM
Atmosphere dynamics are integrated with a time step of 15 Minutes, Ocean dynamics with 2 hours. The coupling is done each day. The extraction of data for my use is done half-daily.

- 96 longitudes with fixed spacing of 3.75°O (ca. 417km at the equator) and
- 48 latitudes with varying spacing around $3,71$°N, on interpolated to
- 23 pressure levels (in Pascal, Pa). The extraction of fields happens with
- $0,5$ days temporal resolution.

The wind fields are given on each grid points as

- west wind component (towards the east) u in meters per second (m/s)
- south wind component (towards the north) v in meters per second (m/s)
- vertical wind component ω in Pascal per second (Pa/s).

The preliminary final data set for me consists of 95 years each for a reference run and a run with interactive chemistry. The efficient work with the amount of data is a technical challenge: One time series of the multivariate wind fields over 95 years needs 83GiB of storage.

The access to the data is abstracted through the `data` class (shortened definition in Listing 2.1). One aspect of this class is to hide the access to potentially many (hundreds, thousands) of files that make up one data set. Another aspect is to hide the detailed structure of the grid and contained variables

Nowhere in my programs the grid resolution in space or time is fixed. This information is collected from the self-describing data sets in the open and platform independent NetCDF format ([NetCDF]). Without such a format, the design of a flexible and portable software package for data analysis would be very questionable.

Through the common data format it is relatively easy for me to adapt my programs for the work with additional, or just different, data sets. For example, I initially just needed the three-dimensional wind data and created the programs accordingly, but was able to quickly integrate changes to enable a co-worker to analyze ECHO-GiSP data for geopotential and chemical concentrations (used in [Chandra2007]).

Listing 2.1: Core elements of the common data interface

```
class data
{
  public:
  vector<string> varnames;
  vector<string> dimnames;
  size_t vars; size_t dims; // convenience shortcuts to .size()
  pep_t timeunit; //seconds per time unit in file (p.ex. days in
     file: 86400)
  pep_grid grid;
  bool ready;

  data();
  virtual ~data(){ destruct("data"); };
  // placeholders to be implemented in special interface
  virtual bool init(){ return false; }; // to-be useful
  virtual bool init(char** filenames, const int count){ return
     false; }; // to-be useful

  // generic data access via index
  virtual void get_vector(dataindex index, pep_t *vals) = 0;
  // coordinate access
  virtual void get_vector(const place p, pep_t *w);

  // omitted here: utility functions for finding a point in the
     grid, etc.
};
```

Meanwhile, there does exist a series of special classes that can be used through the `data` interface[3]:

- `echog_pressure`, which defines the common grid dimensions of ECHO-GiSP data,
- `echog_pressure_cached`, which implements the for the quasi-random access indispensable[4] data cache in RAM,
- `egp_wind` as specialization on the wind components u, v and ω,
- `egp_geopoth` for accessing the geopotential, including a special method to convert to height meters and
- `egp_density`, which works with temperature and humidity data and computes the density.

[3]via C++ polymorphism

[4]Without this, the computer is fully occupied with searching through the hard disk and the main processor is essentially waiting all the time.

3 From the wind field to the transported ensemble

The central building block of the Lagrangian view on transport is the single trajectory, the movement of a single particle. It is about the computation of the trajectory of the position vector

$$\vec{x}(t)$$

driven by the given instationary wind field $\dot{\vec{r}}$

$$\dot{\vec{r}} = f(\vec{r}, t);\ \vec{r}(t_0) = \vec{r}_0 \tag{3.1}$$

This is a classic initial value problem of first order. With analytic treatment with a given function $\dot{\vec{r}}(\vec{r}, t)$, the solution is obvious by principle[1]. I will show you a simple (unphysical) example with $\vec{r} = (x);\ f(\vec{r}, t) = (3x + 1)t$ in dimensionless form:

$$\begin{aligned} \dot{x} = \frac{\mathrm{d}}{\mathrm{d}t}x &= (3x+1)t \qquad (3.2) \\ \overset{x>=0}{\Leftrightarrow} \frac{1}{3x+1}\mathrm{d}x &= t\mathrm{d}t \\ \Leftrightarrow \int_0^x \frac{1}{3x+1}\mathrm{d}x &= \frac{1}{2}t^2 \end{aligned}$$

Using the substitution

$$\begin{aligned} y &= 3x+1 \\ \frac{\mathrm{d}y}{\mathrm{d}x} &= 3 \Rightarrow \mathrm{d}x = \frac{\mathrm{d}y}{3} \end{aligned}$$

the integral can be solved like that

$$\begin{aligned} \int_1^{3x+1} \frac{1}{3}\cdot\frac{1}{y}\mathrm{d}y &= \frac{1}{2}t^2 \\ \Leftrightarrow \frac{1}{3}\ln y|_1^{3x+1} = \frac{1}{3}\ln(3x+1) &= \frac{1}{2}t^2 \\ \Leftrightarrow \ln(3x+1) &= \frac{3}{2}t^2 \\ 3x+1 &= e^{\frac{3}{2}t^2} \\ \Rightarrow x(t) &= \frac{e^{\frac{3}{2}t^2}-1}{3} \end{aligned}$$

[1]☺ as long as the differential equation is easy to integrate...

This fulfills the differential equation Equation 3.2:

$$\begin{aligned}
\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}x &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{e^{\frac{3}{2}t^2}-1}{3}\right) \\
&= 2\frac{3}{2}t\frac{e^{\frac{3}{2}t^2}}{3} = t\cdot 3\frac{e^{\frac{3}{2}t^2}}{3} \\
&= t\cdot\left(3\frac{e^{\frac{3}{2}t^2}-1}{3}+3\frac{1}{3}\right) \\
&= (3x+1)t
\end{aligned}$$

Now, in this work, I do not deal with a simple differential equation, but a climate model, that gives me values of the function $f(\vec{x},t)$ from Equation 3.2 in form of the wind components u, v and ω at discrete points in time on discrete points of a spatial grid in λ, β and p.

The intention stays the same: I want to compute the trajectory $\vec{r}(t)$ out of given $\dot{\vec{r}} = f(\vec{r},t)$. But an analytical treatment does not provide the answer. Numerical integration in the given discrete wind field (Indices o, a, e for dimensions longitude, latitude, height and i for the time) does:

$$\dot{\vec{x}} = \begin{pmatrix} u_{o,i} \\ v_{a,i} \\ \omega_{e,i} \end{pmatrix} \; ; \; o,a,e,i \in Z^+ \tag{3.3}$$

A basic question comes to mind regarding the horizontal wind in m/s and the horizontal coordinates in the geographic grid, (λ,β). Those need to be brought together somehow. The vertical speed is given in Pa/s, which basically fits the pressure levels in Pa. Under the assumption of spherical geometry and (local) neglecting of the height variation of the pressure levels this pressure speed is also orthogonal to the horizontal wind. This property is a precondition for the application of a integration scheme like Runge-Kutta in multiple dimensions[2].

The question of the horizontal coordinates and the accompanying winds has to be resolved. One has to make a decision, between

1. either computation in the geographical grid, for that local conversion from m/s into °O/s and °N/s **or**
2. computation in a local space, using meters, after conversion of the °O and °N coordinates into m as well as transformation of the wind direction in the chosen projection.

The first variant is without doubt the simplest – one just has to scale the values of the wind components and there is no further need to bother with geometry. A disadvantage, besides systematic error, is the situation at the poles. Directly at the pole in the (λ,β) coordinate system, the wind is not defined at all. There is just one degenerated direction towards the opposite pole (at the north pole, you can use “south” synonymous with “any direction”, the other way round at the south pole) and “east” or “west” is not defined at all. Some people just artificially define values at the poles, or extend the values for each longitude to the final latitude of the pole, to be able to totally work in the geographic grid. I want to approach

[2]Without orthogonality of the dimensions the coordinate changes of the sub steps would not be independent; movement in x direction would also change p.

this problem directly. I want to work with properly defined winds and directions. Therefore, I will investigate the second method that works in a local Cartesian coordinate system (with flat coordinates in meters). But before I give the simpler computation in the geographic grid a treatment, too.

Before going into detail about the transport computations, I want to present the generic algorithms for interpolation and integration, that are in use for my work up to now[3].

3.1 Linear interpolation in N Cartesian dimensions

The simplest method to interpolate between points in a *Cartesian* grid is the linear interpolation inside a grid cell[4]. In detail:

The objective is the linear interpolated value of a variable y that depends on coordinates $x_n;\ n \in [1;N]$: $y(\vec{x});\ \vec{x} \in \mathbb{R}^N$. Given are values of y on a hyper cuboid in $\mathbb{R}^N$, called "grid cell", that encloses the point $\vec{x}$. The grid cell is defined by intervals in each of the N dimensions, enclosing the respective components of $\vec{x}$: $x_{n,0} \leq x_n \leq x_{n,1}$.

In the literature, I usually find just one display of the variant in two dimensions, with the lapidary note that it is analogous in higher dimensions. Sometimes, this is accompanied with the sketch of the idea of recursion using one-dimensional linear interpolations. The direct algorithms for bilinear or trilinear (2D and 3D) interpolation are in widespread use and described. But what I was unable to find, is a definite formula and corresponding algorithm for the N-dimensional interpolation. I often read that the generalization to arbitrary dimensions is obvious. But despite – or because of? – that, I was unable to find something definite. It is true that I actually just need the interpolation in 4 dimensions, but I think it is appropriate to derive this special case from some generic rule. Because I cannot find a description of this "obvious" generic rule, I have to construct it beforehand. The difficulty is not in the concept[5], but rather in getting the rule into the right shape. This is not impossible, though, given the right tool – what I am about to show with the following pages.

3.1.1 A generic recursion formula

The concept of linear interpolation is very simple. What complicates the issue for arbitrary dimensions, is the enumeration of the corners of the grid cell. The coordinate vector of each grid point in the role of the grid cell corner consists of a combination of lower and upper interval boundaries in the individual dimensions. The x_n coordinate of a corner is either $x_{n,0}$ or $x_{n,1}$. A certain corner is thus defined through a series of N choices of lower border (0) or upper border (1) for the interval in each dimension. A natural hierarchy of the corner points is established when this selection of lower or upper border in dimension n is interpreted as n-th digit of a binary number. To work with this number as a whole and the individual digits at the same time, I use the term of the vector in the binary vector

[3]It is one aim of program design to keep the algorithms replaceable. It shall be possible to add and evaluate different approaches.

[4]Different from the general interpolation problem that searches a function for the *whole* grid.

[5]☺ Which is obvious, remember?

space $\mathbb{B}^k$; $\mathbb{B} = \{0; 1\}$. Every element $\vec{b}$ in this space corresponds to the binary encoding of a number[6] $b \in \left[0; 2^k - 1\right] \subset \mathbb{Z}$:

$$b = \sum_{n=1}^{k} b_n 2^{n-1} \tag{3.4}$$

E.g. for $k = 3$, $\vec{b} = (0; 1; 1)$ is the vector corresponding to the decimal number $b = 6$, according to its representation in binary[7]: 110.

This assignment is nothing different than the representation of a number in the one or the other system of numbers, therefore it is bijective, unambiguous. With the definition of of $\vec{b} \in \mathbb{B}^k$ I can reach all corners of the k-dimensional grid cell through simple iteration in b and the directly connected index vector $\vec{b}$. The k-dimensional coordinate vectors $\vec{x}^{(k)}$ are index ted via the plain number b, what, using the relation to $\vec{b}$, includes the indices for all components $x_n^{(k)} = xn, b_n$ (reminder: $x_{n,0}$ and $x_{n,1}$ are the given interval borders):

$$\vec{x}_b^{(k)} = (x_{1,b_1}, \ldots, x_{k,b_k}) \tag{3.5}$$

I treated k separately from the dimension count N, to be able to represent the k-dimensional grid cell contained in the N-dimensional space, appearing as intermediate result in the course of the recursive interpolation The complete coordinate vector $\vec{x}_b$ of a corner of the k-dimensional grid cell in the N-dimensional space is described by

$$k \in [0; N] : \; \vec{x}_b = (\vec{x}_b^{(k)}, x_{k+1}, \ldots, x_N) = (x_{1,b_1}, \ldots, x_{k,b_k}, x_{k+1}, \ldots, x_N) \tag{3.6}$$

(Components of $\vec{x}_b^{(k)}$ are inserted into $\vec{x}_b$.).

A remark about the hierarchy of the indices b shall help understanding the following recursion mechanism. Between the coordinates and indices of adjacent recursion steps, there holds the relation

$$\begin{array}{rcl} \left(\vec{x}_b^{(k-1)}, x_{k,\mathbf{0}}\right) & = & x_b^{(k)} \\ \left(\vec{x}_b^{(k-1)}, x_{k,\mathbf{1}}\right) & = & x_{b+2^{k-1}}^{(k)} \end{array} \tag{3.7}$$

That becomes clear when looking at the index vector. For $x_b^{(k-1)}$, this is $\vec{b} = (b_1, \ldots, b_{k-1})$, the index value $\sum_{n=1}^{k-1} b_n 2^{n-1} = b$. When $\vec{b}$ is extended by one dimensions, attaching $b_k = 0$, the result is $\vec{b'} = (b_1, \ldots, b_{k-1}, 0)$ and the index value does not change: $b' = b + 0 \cdot 2^{k-1} = b$. But when a 1 is attached, the resulting index value of $\vec{b''} = (b_1, \ldots, b_{k-1}, 1)$ is $b'' = b + 1 \cdot 2^{k-1}$. The extension of the index vector causes a doubling of the used indices, from 2^{k-1} to $2 \cdot 2^{k-1} = 2^k$. All index vectors that got a 0 attached, have an index value $b' \in \left[0; 2^{k-1} - 1\right]$, all vectors that got a 1, lie in the upper half of the enlarged interval: $b'' \in \left[2^{k-1}, 2^k - 1\right]$.

[6] b is a valid norm of the vector $\vec{b}$

[7] In the binary notation, the first digit (the least significant bit) is written at the *right* side, while it appears on the *left* side in the vector notation, according to common convention.

Using this indexing, a clear generic formulation of the linear interpolation in a grid cell in N dimensions is possible, usually only being indicated verbally or in two dimensions:

$$\begin{aligned}
\forall n \in [1;N]: \qquad & d_{n,0} = \frac{x_{n,1} - x_n}{x_{n,1} - x_{n,0}};\; d_{n,1} = 1 - d_{n,0} && (3.8)\\
\forall k \in [1;N]: \qquad & \forall b \in \left[0; 2^{k-1} - 1\right]: \\
y_b^{(k-1)}(\vec{x}_b^{(k-1)}, x_k, \ldots, x_N) \;=\; & d_{k,0} \cdot y_b^{(k)} \left(\vec{x}_b^{(k-1)}, x_{k,\mathbf{0}}, x_{k+1}, \ldots, x_N\right) \\
& + d_{k,1} \cdot y_{b+2^{(k-1)}}^{(k)} \left(\vec{x}_b^{(k-1)}, x_{k,\mathbf{1}}, x_{k+1}, \ldots, x_N\right)
\end{aligned}$$

Corresponding to the location of x_n between $x_{n,0}$ and $x_{n,1}$, the weights $d_{n,0}$ and $d_{n,1}$ are fixed – just like in the one-dimensional case. The recursion derives the interpolation in the dimension n from the values resulting from the interpolation in dimensions N to $n+1$. That way, the values $y_b^{(N)}$ at the 2^N corners are interpolated to the final $y_0^{(0)}$. The first and last step of the recursion are given by

$$\begin{aligned}
y(x_1, \ldots, x_N) &= y_0^{(0)}(x_1, \ldots, x_N) && (3.9)\\
&= d_{1,0} \cdot y_0^{(1)} (x_{1,\mathbf{0}}, x_2, \ldots, x_N) \\
&\quad + d_{1,1} \cdot y_1^{(1)} (x_{1,\mathbf{1}}, x_2, \ldots, x_N) \\
\forall b \in \left[0; 2^{N-1} - 1\right]: & \\
y_b^{(N-1)} \left(\vec{x}_b^{(N-1)}, x_N\right) &= d_{k,0} \cdot y_b^{(N)} \left(\vec{x}_b^{(N-1)}, x_{N,\mathbf{0}}\right) \\
&\quad + d_{k,1} \cdot y_{b+2^{N-1}}^{(N)} \left(\vec{x}_b^{(N-1)}, x_{N,\mathbf{1}}\right) \\
\forall b \in \left[0; 2^N - 1\right]: \; y_b^{(N)} &= y_b^{(N)} \left(\vec{x}_b^{(N)}\right)
\end{aligned}$$

Following Equation 3.9 and Equation 3.9, a C++ function can be written in a very direct way, to be seen in Listing 3.1. Due to the intended application, this function does not just compute a single value, but a vector $\vec{y}$ of multiple variables, these components being independent and thus not bothering the basic algorithm.

3.1.2 Reversal of the recursion

I must confess that the recursive implementation in Listing 3.1 may be the first variant in a didactic sense, but occurred to me as the last one. The inverted work flow appeared more natural to me: not top-down via recursion but rather bottom-up in an iterative way. Generally, the reformulation of a recursive algorithm to yield an iterative algorithm is deemed beneficial, though it is not guaranteed that this will result in increased computing efficiency in the real world application.

After some thinking, such a reformulation to iterative computation can be found. The $d_{n,i}$ from Equation 3.8 stay as given. When you look at Equation 3.9, you can also read the iterative algorithm instead of the recursive one! I work in a loop over dimensions from high k and compute, for every step, the set of $y_b^{(k-1)}$ out of the $y_b^{(k)}$, or $y_{b+2^{(k-1)}}^{(k)}$, respectively. The formula is the same, but the program in Listing 3.2 does not need recursion; just **for** loops.

Listing 3.1: N-dimensional linear Interpolation by recursion

```c
void int_linear_recursive( size_t dims, size_t vars,
                           pep_t *here, pep_t *corners,
                           pep_t *cornval, pep_t *result,
                           size_t k = 1, size_t b = 0 )
{
  // work on indices b and b+2^(k-1)
  size_t b1 = b + ((size_t)1<<(k-1))*vars;
  xdebug("k=%zu;  b=%zu;  b1=%zu", k, b, b1);
  // calculate weigths
  pep_t d0 =
    (corners[dims+k-1] != corners[k-1])
    ? ((corners[dims+k-1] - here[k-1]) / (corners[dims+k-1] -
       corners[k-1]))
    : 1;
  pep_t d1 = 1 - d0;

  if(k < dims)
  {
    pep_t b1result[vars];
    // recursion to get y_b and y_(b+2^(k-1))
    int_linear_recursive(dims, vars, here, corners, cornval,
       result, k+1, b);
    int_linear_recursive(dims, vars, here, corners, cornval,
       b1result, k+1, b1);
    // ...and the final weighted sum
    for(size_t v = 0; v < vars; ++v)
    result[v] = d0*result[v] + d1*b1result[v];
  }
  else
  {
    // bottom case: k=dims
    for(size_t v = 0; v < vars; ++v)
    result[v] = d0*cornval[b+v] + d1*cornval[b1+v];
  }
}
```

Listing 3.2: N-dimensional linear interpolation by iteration

```c
void int_linear_reverse(size_t dims, size_t vars,
                        pep_t *here, pep_t *corners[2],
                        pep_t *cornval, pep_t *result  )
{
  // eliminating every dimension
  for(size_t d = 0; d < dims; ++d)
  {
    // weights
    pep_t s0 =
      (corners[1][d] != corners[0][d])
      ? ( (corners[1][d] - here[d])
          / (corners[1][d] - corners[0][d]) )
      : 1;
    pep_t s1 = 1 - s0;
    // the 2^(dims-d) points left after eliminating d dimensions
    for(size_t corn=0; corn < ((size_t)1<<dims)>>d; corn+=2)
    {
      // we compute the values for p0 and p1 together into the place of p0
      // next iteration uses these values (when in doubt, think of memory layout)
      size_t p0_offset = (corn<<d)*vars;
      size_t p1_offset = ((corn+1)<<d)*vars;
      // now interpolating in dimension d (from behind, actually) for every variable
      for(size_t v = 0; v < vars; ++v)
      cornval[p1_offset+v] = s1*cornval[p1_offset+v]
                           + s0*cornval[p0_offset+v];
    }
  }

  if(result != NULL)
  memcpy(result, cornval, sizeof(pep_t)*vars);
}
```

In practice, for fixed N, neither Listing 3.1, nor Listing 3.2 will be put to service. To compute the actually needed interpolation in maximal four dimensions efficiently, I derive the explicit computation according to the complete recursion of Equation 3.9. The view of the resulting formula inspires a much more direct formulation of the N-dimensional interpolation, which will complete this little collection of formulas and algorithms.

3.1.3 Explicit representation in four dimensions

With the ECHO-GiSP data I have to deal with the four dimensions λ (longitude), β (latitude), p (pressure, height) and t (time). In four dimensions it is fairly workable to execute the complete recursion. This may be a bit lengthy, but at least memorable. Instead of the anonymous enumeration of x_n, I use the definite symbols of the dimensions; we look for $y(t, p, \beta, \lambda)$ inside the grid cell spanned by $(t_0, p_0, \beta_0, \lambda_0)$ and $(t_1, p_1, \beta_1, \lambda_1)$. So: $x_1 \to t$; $x_2 \to p$; $x_3 \to \beta$; $x_4 \to \lambda$.

The weights are given by Equation 3.8:

$$\begin{aligned}
d_{\lambda,0} &= \frac{\lambda_1 - \lambda}{\lambda_1 - \lambda_0}; \; d_{\lambda,1} = 1 - d_{\lambda,0} \\
d_{\beta,0} &= \frac{\beta_1 - \beta}{\beta_1 - \beta_0}; \; d_{\beta,1} = 1 - d_{\beta,0} \\
d_{p,0} &= \frac{p_1 - p}{p_1 - p_0}; \; d_{p,1} = 1 - d_{p,0} \\
d_{t,0} &= \frac{t_1 - t}{t_1 - t_0}; \; d_{t,1} = 1 - d_{t,0}
\end{aligned} \tag{3.10}$$

The dimensions can be interpolated "away" now step by step. The numbers i, j, k play the part of $\vec{b}$, or b in the recursion steps.

$$\begin{aligned}
y_0^{(0)}(t, p, \beta, \lambda) &= d_{t,0} \cdot y_0^{(1)}(t_0, p, \beta, \lambda) \\
&\quad + d_{t,1} \cdot y_1^{(1)}(t_1, p, \beta, \lambda) \\
y_i^{(1)}(t_{i_1}, p, \beta, \lambda) &= d_{p,0} \cdot y_i^{(2)}(t_{i_1}, p_0, \beta, \lambda) \\
&\quad + d_{p,1} \cdot y_{i+2}^{(2)}(t_{i_1}, p_1, \beta, \lambda) \\
y_j^{(2)}(t_{i_1}, p_{j_2}, \beta, \lambda) &= d_{\beta,0} \cdot y_j^{(3)}(t_{i_1}, p_{j_2}, \beta_0, \lambda) \\
&\quad + d_{\beta,1} \cdot y_{j+4}^{(3)}(t_{i_1}, p_{j_2}, \beta_1, \lambda) \\
y_k^{(3)}(t_{i_1}, p_{j_2}, \beta_{k_3}, \lambda) &= d_{\lambda,0} \cdot y_k^{(4)}(t_{i_1}, p_{j_2}, \beta_{k_3}, \lambda_0) \\
&\quad + d_{\lambda,1} \cdot y_{k+8}^{(4)}(t_{i_1}, p_{j_2}, \beta_{k_3}, \lambda_1)
\end{aligned} \tag{3.11}$$

After replacements and multiplication of terms, we have all $2^4 = 16$ of them.

$$\begin{aligned}
y(t,p,\beta,\lambda) \quad = \quad & d_{t,0}d_{p,0}d_{\beta,0}d_{\lambda,0}y(t_0,p_0,\beta_0,\lambda_0) \\
& +d_{t,0}d_{p,0}d_{\beta,0}d_{\lambda,1}y(t_0,p_0,\beta_0,\lambda_1) \\
& +d_{t,0}d_{p,0}d_{\beta,1}d_{\lambda,0}y(t_0,p_0,\beta_1,\lambda_0) \\
& +d_{t,0}d_{p,0}d_{\beta,1}d_{\lambda,1}y(t_0,p_0,\beta_1,\lambda_1) \\
& +d_{t,0}d_{p,1}d_{\beta,0}d_{\lambda,0}y(t_0,p_1,\beta_0,\lambda_0) \\
& +d_{t,0}d_{p,1}d_{\beta,0}d_{\lambda,1}y(t_0,p_1,\beta_0,\lambda_1) \\
& +d_{t,0}d_{p,1}d_{\beta,1}d_{\lambda,0}y(t_0,p_1,\beta_1,\lambda_0) \\
& +d_{t,0}d_{p,1}d_{\beta,1}d_{\lambda,1}y(t_0,p_1,\beta_1,\lambda_1) \\
& +d_{t,1}d_{p,0}d_{\beta,0}d_{\lambda,0}y(t_1,p_0,\beta_0,\lambda_0) \\
& +d_{t,1}d_{p,0}d_{\beta,0}d_{\lambda,1}y(t_1,p_0,\beta_0,\lambda_1) \\
& +d_{t,1}d_{p,0}d_{\beta,1}d_{\lambda,0}y(t_1,p_0,\beta_1,\lambda_0) \\
& +d_{t,1}d_{p,0}d_{\beta,1}d_{\lambda,1}y(t_1,p_0,\beta_1,\lambda_1) \\
& +d_{t,1}d_{p,1}d_{\beta,0}d_{\lambda,0}y(t_1,p_1,\beta_0,\lambda_0) \\
& +d_{t,1}d_{p,1}d_{\beta,0}d_{\lambda,1}y(t_1,p_1,\beta_0,\lambda_1) \\
& +d_{t,1}d_{p,1}d_{\beta,1}d_{\lambda,0}y(t_1,p_1,\beta_1,\lambda_0) \\
& +d_{t,1}d_{p,1}d_{\beta,1}d_{\lambda,1}y(t_1,p_1,\beta_1,\lambda_1)
\end{aligned} \tag{3.12}$$

The sum is expressed using four short loops in the associated C++ program (Listing 3.3) – it is up to the compiler to optimize these away for maximum efficiency[8].

3.1.4 Generalization to plain sum in N dimensions

Looking at Equation 3.12, especially at the compressed loop in the program, provokes the thought that the general interpolation in N dimensions can be expressed as a plain sum, too. That is the case! Using the binary vector, the linear interpolation in N dimensions can indeed be expressed compactly as a sum over all possible values of b and the corresponding $\vec{b} \in \mathbb{B}^N$ – the assignments of 0 or 1 to the indices b_n.

$$y(\vec{x}) \quad = \quad \sum_{b=0}^{2^N-1} \left(\prod_{n=1}^{N} d_{n,b_n} \right) \cdot y\left(\vec{x}_{\vec{b}}\right) \tag{3.13}$$

This sum encompasses all 2^N corners of the grid cell and weights them with the appropriate product of the $d_{n,i}$. In theory, that leads to the same result as when using Equation 3.9 – the formula corresponds to the worked-out recursion, as shown for four dimensions. In practice, you can get differences due to the limited accuracy of computers combined with the lowered number of mathematical operations. The theoretical equivalence to Equation 3.9 can be proven generally by induction:

I begin with the assumption, that the interpolation of $y(x'_1, \ldots, x'_N)$ is given by the plain sum (as shown for $N = 4$). Each interpolation over N' dimensions can be seen as an incomplete

[8]... which is probably appropriate here, but excessive "unrolling" of loops can have a negative impact due to increased code size.

Listing 3.3: definite program code for the interpolation in four dimensions

```c
pep_t interpolate_4d(pep_t grid[DIMS][GRIDSIZE],
  size_t index[2][DIMS], pep_t position[DIMS], pep_t val
     [2][2][2][2])
{
  pep_t m = 0;    // the value at point of interest
  pep_t d[DIMS][2]; // d and (1-d) for 4 coordiantes

  for(int i = 0; i < DIMS; ++i)
  {
    d[i][0] = (grid[i][ index[1][i] ] - position[i])
              / (grid[i][ index[1][i] ] - grid[i][ index[0][i] ]);
    d[i][1] = 1-d[i][0];
  }

  for(int t = 0; t < 2; ++t)
  for(int p = 0; p < 2; ++p)
  for(int beta = 0; beta < 2; ++beta)
  for(int lambda = 0; lambda < 2; ++lambda)
  m += d[0][t] * d[1][p] * d[2][beta] * d[3][lambda]
        * val[t][p][beta][lambda]

  return m;
}
```

interpolation in the $(N = N' + 1)$ - dimensional space. The application of the finale step from Equation 3.9 will show that the result can, again, be represented as a plain sum. To be able to apply Equation 3.9, I add the interval border $x_{1,i}$ as new first coordinate for the embedding in the N - dimensional space, renumbering the following indices $x_n = x'_{n-1}$. Now the two interpolated values for the upper and lower border of the interval $[x_{1,0}; x_{1,1}]$ of the first, added, dimension are covered:

$$i \in \{0;1\}: \; y_i(x'_1, \ldots, x'_N) = y(x_{1,i}, x'_1, \ldots, x'_N) = y(x_{1,i}, x_2, \ldots, x_N)$$

The direct formula for y needs some change because of the newly introduced coordinate with its new index. But this index is fixed at 0 or 1, while the old indices are shifted in position by one. The sum is still carried out over $b \in \left[0; 2^{N'} - 1\right]$ elements, but the actual work index is modified: $b(i) = 2b' + i$. The multiplication with 2 moves the indices (binary bits) by one position, the addition of i fixes the newly created first[9] index to $i = 0$ or $i = 1$.

$$b(i) = 2b' + i: \; y(x_{1,i}, x_2, \ldots, x_N) \;=\; \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=2}^{N} d_{n,(b(i))_n} \right) \cdot y(\vec{x}_{\vec{b}(i)}) \qquad (3.14)$$

Now I can unify that form with Equation 3.13 with the final step of the recursion from Equation 3.9:

$$\begin{aligned} y(x_1, \ldots, x_N) \quad &= \quad d_{1,0} \cdot \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=2}^{N} d_{n,(b(0))_n} \right) \cdot y(\vec{x}_{\vec{b}(0)}) && (3.15) \\ &\quad +d_{1,1} \cdot \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=2}^{N} d_{n,(b(1))_n} \right) \cdot y(\vec{x}_{\vec{b}(1)}) \\ &\overset{(b(0))_1=0}{\underset{(b(1))_1=1}{=}} \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=1}^{N} d_{n,(b(0))_n} \right) \cdot y(\vec{x}_{\vec{b}(0)}) && (3.16) \\ &\quad + \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=1}^{N} d_{n,(b(1))_n} \right) \cdot y(\vec{x}_{\vec{b}(1)}) \end{aligned}$$

The global factors $d_{1,0}$ and $d_{1,1}$ move into the product inside the sum, so that that product can be extended by the first index with $(b(i))_1 = i$.

What still is missing for the identification with Equation 3.13, is the unification of the two sums. A view on the binary indices shows that this is actually already a given: Together, both partial sums cover all $b(i) \in \left[0; 2^N - 1\right]$ via $b' = 2^{N-1} - 1 \Rightarrow b(1) = 2(2^{N-1} - 1) + 1 = 2^N - 1$. The first sum contains all even $b(i)$ (because of the least significant bit being zero) and the second all odd $b(i)$. The derived index $b(i)$ can be replaced by a direct b in the unified sum over $b \in \left[0; 2^N - 1\right]$, which proves that

$$\begin{aligned} i \in \{0;1\}: \; y(x_{1,i}, x_2, \ldots, x_N) \;&=\; \sum_{b'=0}^{2^{N-1}-1} \left(\prod_{n=2}^{N} d_{n,(b(i))_n} \right) \cdot y(\vec{x}_{\vec{b}(i)}) && (3.17) \\ \Rightarrow y(x_1, x_2, \ldots, x_N) \;&=\; \sum_{b=0}^{2^N-1} \left(\prod_{n=1}^{N} d_{n,b_n} \right) \cdot y\left(\vec{x}_{\vec{b}}\right) \end{aligned}$$

[9]the first, or rather *zeroth*, bit – according to the value $2^0 = 1$

To summarize: From the validity of Equation 3.13 for $N-1$ dimensions it follows, with Equation 3.9 as special case of Equation 3.9, the validity of Equation 3.13 for N dimensions. I derived the direct formula from the laid-out recursion in four dimensions, so it is proved that it is valid for $N > 4$. In principle, the proof for $N < 4$ is missing, but already a glance at $N = 1$ show the equivalence for this case.

Having secured its validity, the formula explicitly using the components of $\vec{b}$ can be implemented by aid of the bit operators in C++ in Listing 3.4. The combination of bitshift to the left or right[10] as well as bitwise "and" facilitates the access of individual bits out of the binary representation of an integer number. ((corn & (1<<d))>>d) yields 1 if bit d (from the right) is set and 0 if not.

Listing 3.4: N-dimensional linear interpolation using plain sum

```
void int_linear_direct(size_t dims, size_t vars, pep_t *here,
   pep_t *corners[2], pep_t *cornval, pep_t *result)
{
  pep_t d[dims][2];    // weight factors for dims directions
  for(size_t dim = 0; dim < dims; ++dim)
  {
    d[dim][0] =
      (corners[1][dim] != corners[0][dim])
      ? ( (corners[1][dim] - here[dim])
          / (corners[1][dim] - corners[0][dim]) )
      : 1;
    d[dim][1] = 1 - s[dim][0];
  }

  for(size_t i=0; i != vars; ++i) result[i] = 0; // init

  // sum up the parts, we have 2^dims corners
  for(size_t corn=0; corn < ((size_t)1)<<dims; ++corn)
  {
    // compute weight of corner point
    pep_t weight = 1;
    for(size_t dim = 0; dim < dims; ++dim)
    weight *= d[dim][(corn & (((size_t)1)<<dim))>>dim];

    // now add the weighted values
    for(size_t v=0; v < vars; ++v)
    result[v] += cornval[corn*vars+v]*weight;
  }
}
```

[10]left: in direction of higher bits; right: in direction of lower bits

3.1.5 Efficiency comparison

You may ask the question about why one needs three different algorithms for one and the same (sort of simple) computation. One answer is delivered by an investigation of the needed runtime depending on dimensionality and number of variables, up to 20 each. Figure 3.1 shows a mixed picture: Every variant has its realm, where it is the best performer for the task. Listing 3.2 is dominating on Neuling for 10 to 16 dimensions and low variable count, while for higher values of both parameters it is inferior to both other variants. But these relations depend sensitively on the machine in use – especially the memory architecture decides if it is about computing time of the processor or waiting time for memory access. Therefore, the main message of the measurements is that the question for the generally best-performing algorithm cannot be answered clearly. At the edge of the investigated parameter space, the plain sum seems to win for high dimension and variable count, but that can change with further scaling of parameters or a different hardware platform.

One rule hold true for all methods shown here: The complexity of the linear interpolation using all corners of the N-dimensional grid cell is governed by 2^N, the number of corner points. The computing effort raises exponentially with the number of dimensions – no algorithm that incorporates all corners can change that. A possibility for the linear interpolation with significantly reduced complexity employs the selection of a subset of corner points to be used – like shown in [Rovatti1998]. There, the simplex of $N+1$ corner points encompassing the interpolation point is determined, the interpolation then only using these $N+1$ points. That is indeed a definite saving compared to the 2^N corner points of the complete cell, but at the price of neglecting more of your available data.

In the end, for my applications with interpolation in two or four dimensions, a fixed variant like Listing 3.3 will be employed. But it does not hurt to be aware of the background – and to have treated the N-dimensional linear interpolation in a more thorough way than usual, for once.

3.2 Interpolation using distance-weighted sum / Shepard interpolation

The linear interpolation forms a sum over the values at encompassing grid points, weighted with the product of distances in the individual dimensions. That works in a Cartesian grid with rectangular cells, but not when the grid points are irregularly spaced, or the grid is just no rectangular one. When you loop at the (λ, β) geographical grid points not in the abstract coordinate space but on the earth surface, the grid cells are not rectangular anymore.

A simple method for the interpolation in arbitrary grids has been presented by Shepard about forty years ago in [Shepard1968]. He formulated it for two dimensions, yet the number of dimensions is actually irrelevant for the basic idea. This basic idea is also known as "distance-weighted sum" of the neighboring, or even all, grid points. I only consider a local Shepard interpolation: I am limiting the sum to the corner points of the encompassing grid cell in advance. This cell is not Cartesian anymore but still regular in (λ, β), so that the concerning

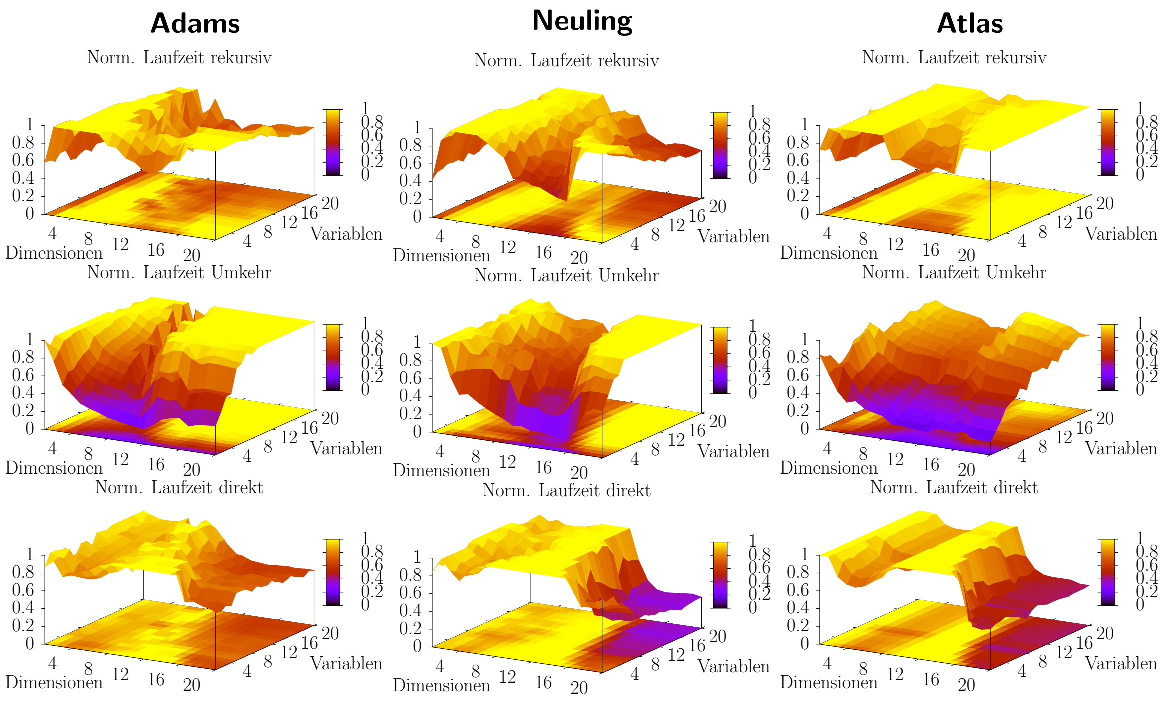


Figure 3.1: Relative runtime of the linear interpolation of up to 20 variables in up to 20 dimensions with the three different approaches on three different computer systems. The depicted value is the runtime for the particular machine, normalized to the maximum of the three runs. A value of 1 means that the variant needed the most time, a value below 1 shows how much faster the concerned variant was compared to the slowest. Despite the indication of some common characteristics of the algorithms concerning the parameters, it is clear that e.g. Atlas generally prefers the memory-intensive iterative approach over the recursion, while the other computer systems (with slower memory access) prefer the recursion for higher variable and dimension count. The differing balance between memory access / transfer and the actual processing performance is always an important factor.

corner points are quickly identified. Using the distance function $d(\vec{x}, \vec{x}_i)$, the computation after Shepard is given by

$$y(\vec{x}) = \frac{\sum\limits_{i=1}^{n} \frac{1}{d(\vec{x},\vec{x}_i)^k} y(\vec{x}_i)}{\sum\limits_{i=1}^{n} \frac{1}{d(\vec{x},\vec{x}_i)^k}} \tag{3.18}$$

One is free to choose an exponent k, but already in [Shepard1968], $k = 2$ is suggested as optimal value, also in Cartesian space computable with minimal effort[11]: With the euclidean metric, the distance weight is simply derived from $d(\vec{x}, \vec{x}_i)^2 = \left(\sqrt{(\vec{x} - \vec{x}_i)^2}\right)^2 = (\vec{x} - \vec{x}_i)^2$. The code in Listing 3.5 shows the essential interpolation method as used in the class `world`.

3.3 four-step Runge-Kutta integration

The established Runge-Kutta scheme of fourth grade is to be found schematically in the literature, usually for the single coordinate $x(t)$. The scheme essentially stays the same for higher (Cartesian!) dimensions and can be applied directly here after generalizing to vectors of variables. Out of $\vec{x}$ at time t_0, $\vec{x}$ at time $t_0 + \Delta t$ is being computed:

$$\begin{aligned}
\vec{x}_0 &= \vec{x}(t_0) \\
\vec{k}_1 &= \Delta t \cdot \dot{\vec{x}}(\vec{x}_0, t_0) \\
\vec{k}_2 &= \Delta t \cdot \dot{\vec{x}}(\vec{x}_0 + \frac{1}{2}\vec{k}_1, t_0 + \frac{1}{2}\Delta t) \\
\vec{k}_3 &= \Delta t \cdot \dot{\vec{x}}(\vec{x}_0 + \frac{1}{2}\vec{k}_2, t_0 + \frac{1}{2}\Delta t) \\
\vec{k}_4 &= \Delta t \cdot \dot{\vec{x}}(\vec{x}_0 + \vec{k}_3, t_0 + \Delta t) \\
\Rightarrow \vec{x}(t_0 + \Delta t) &= \vec{x}_0 + \frac{1}{6}\left(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4\right) \qquad (3.19)
\end{aligned}$$

For further details concerning this standard procedure, I defer to [Bronstein2000], where the essential issues are covered.

3.4 3D integration in the geographical

The first approach works completely in the geographical grid and basically ignores the curvature of dimensions. Values for $\dot{\vec{x}}$ between grid points are computed using linear interpolation in all four dimensions of the grid. But the winds extracted from ECHO-GiSP are given in

[11] the local Cartesian coordinate system, where the grid cell, seen as Cartesian in global coordinate space, is not Cartesian anymore

Listing 3.5: Prototype of the applied Shepard-interpolation

```
size_t n;                // input: number of grid cell points
coordinates *system;  // input: local system
place np[n];          // input: grid cell places
wind  nw[n];          // input: grid cell winds
wind w;               // output: resulting local wind

wind lw[n];  // local grid cell winds
pep_t weightsum = 0;
size_t i;

for(i = 0; i < n; ++i) // weights, sum of
{
  // global -> local coordinates
  system->lwind(np[i], nw[i] ,lw[i]);
  system->lplace(np[i],workplace);
  // lmetric_2d = 2D (squared) distance from system base
  if( (weight[i] = system->lmetric_2d(workplace)) != 0 )
  weightsum += (weight[i] = 1/weight[i]);
  else break; // hit a grid point
}

if(i < n) // unlikely exact hit of a point
copy_wind(lw[i],w);
else // sum winds with normalized weights
{
  for(i = 0; i < n; ++i) weight[i] /= weightsum;

  for(size_t speed = 0; speed < SPEEDS; ++speed)
  {
    w[speed] = 0;
    for(i = 0; i < n; ++i)
    w[speed] += weight[i] * lw[i][speed];
  }
}
```

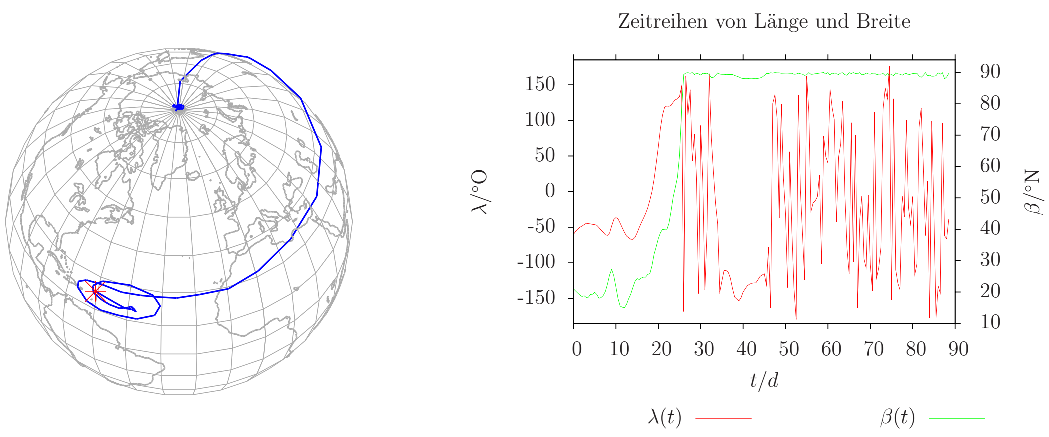


Figure 3.2: The polar trap for geographical grid integration
An early result using geographical grid integration without vertical transport (movement on a pressure surface): The particle stays trapped at the north pole instead of going to the other side.

m/s. To be able to execute the integration in the geographic grid, v and u need to be locally converted from m/s into °/s:

$$u'(\lambda,\beta,p) \quad = \quad u(\lambda,\beta,p)\cdot\frac{180°}{\pi R\cdot cos(\beta)} \tag{3.20}$$

$$v'(\lambda,\beta,p) \quad = \quad v(\lambda,\beta,p)\cdot\frac{180°}{\pi R} \tag{3.21}$$

Together with the unchanged vertical wind ω in the locally adapted velocity, we have

$$\dot{\vec{x}}'(\vec{r}) = \dot{\vec{x}}'(\lambda,\beta,p) \quad = \quad \begin{pmatrix} u' \\ v' \\ \omega \end{pmatrix} \tag{3.22}$$

Instead of $\dot{\vec{r}}$, the converted $\dot{\vec{x}}'$ is used in 3.19.

That is the complete calculation, actually! Truly simple, but also problematic.

My first prototype[12] uses this procedure, limited to horizontal computation corresponding to the first test data. Independent of the latter restriction, a fundamental problem shows itself in vicinity of the poles: Without special treatment, particles can become trapped at the pole, see Figure 3.2. One can see from Figure 3.3 and Figure 3.4 that, especially at the north pole, the stagnation of particles at one place is highly unrealistic, taking those wind characteristics into account.

3.5 The earth is round

In the implementation following the prototype, the bilinear interpolation in λ, β has been replaced by the transformation in a local coordinate system with more flexible distance-weighted sum. This approach of a local coordinate system follows out of treatment of the

[12]written in Perl using PDL to read NetCDF data and do vector operations, just 370 lines of code

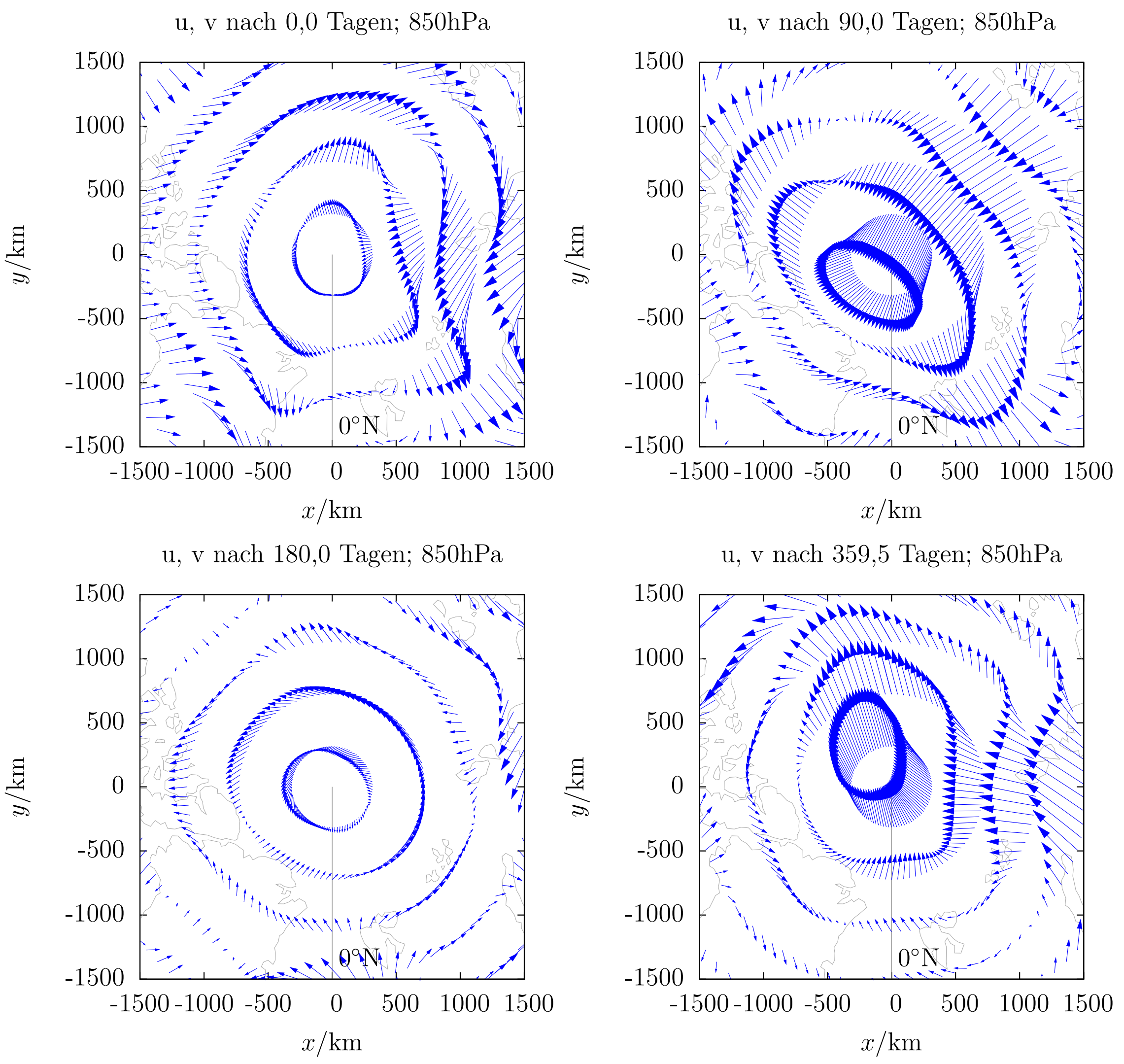


Figure 3.3: North polar wind in 850hPa (orthographic), interactive run
Especially at the pole, there is a hefty wind blowing over it, not just into. On the display is data from the interactive run of year 1965. In the middle of the picture, 30 kilometers length of wind vector correspond to 1m/s wind speed.

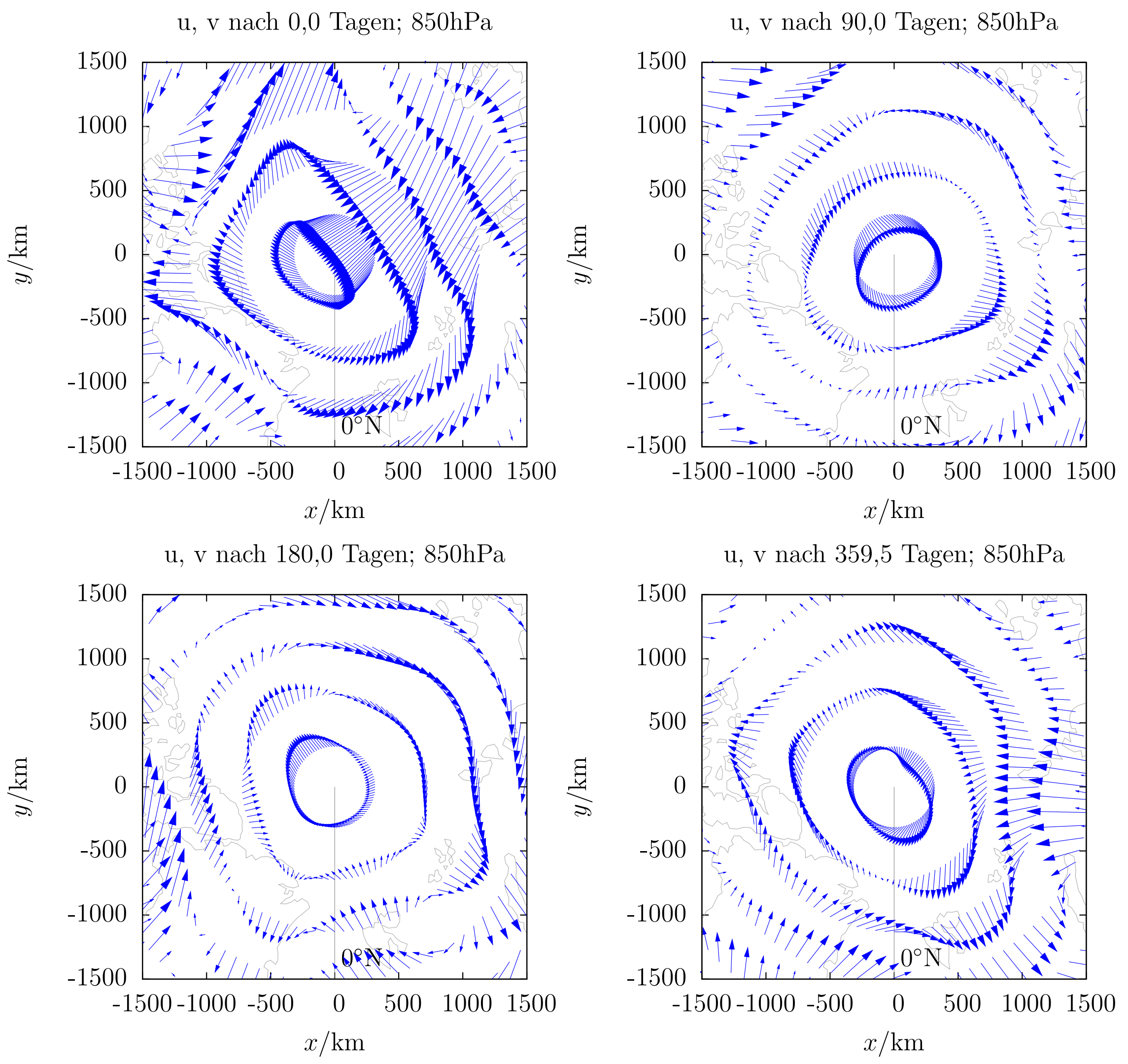


Figure 3.4: North polar wind in 850hPa (orthographic), reference run
Especially at the pole, there is a hefty wind blowing over it, not just into. On the display is data from the reference run of year 1965. In the middle of the picture, 30 kilometers length of wind vector correspond to 1m/s wind speed.

fact that the earth is round, wile one uses an integration scheme that has been designed for Cartesian space.

A multi-dimensional Runge-Kutta scheme (like other common schemes) assumes the different spacial dimensions to be Cartesian, so that the coordinate axes are rectangular to each other. Indeed, that is *locally* the case in the longitude-latitude-height system[13]. But already the corners of grid cells have differing north and east directions and the individual integration steps are ambiguous when you move on the surface according to the velocities given in m.

When I say differing north and east directions, then I better explain that. Of course, the directions are properly defined[14]. But the relation of directions at different places is not that simple on the curved geometry. For me standing on the earth surface, it appears flat at first. I think of myself being in a Cartesian coordinate system, with *one* north and, rectangular to that, *one* east direction. The impression lies, of course, we know that the earth is round and the coordinate system built by the longitudes and latitudes not Cartesian. But still, that local view and the associated separation of the geographical directions bears some sense.

A thought experiment may serve as an example: I have got two points on the earth surface, sufficiently far away from each other and not both on the equator or a common longitude. One is the starting point, the other the final point. At the starting point, I point with a compass[15] towards the final point and note the position of the needle, that is pointing towards north. Then, I move to the final point. having arrived, I point the compass back towards the starting point, then turn it 180 degrees, so that it again points into the same direction as before, in my humble opinion[16]. I note the current position of the needle...

The position of the needs at the final point is differing from the position noted at the starting point! Why? Well, I moved in my “flat” world, without regard for the geographic grid. “Along a straight line” means that I went on my way to the final point along a great circle – and that I also point the compass along that circle. If I fixed a course (e.g. strictly east) and followed that with the help of the compass, I would not have gone on a straight line, but along a circle around the pole[17].

The difference of the north directions should be apparent in Figure 3.5. At the final point, the direction that pointed to north at the starting point, does not point to the north anymore – same for the east direction, which is always locally orthogonal to the north.

One can show it in a less subtle way, observe Figure 3.6: Looking from far away in space at the close neighborhood of the north pole, it is more than obvious that a wind towards the north can come from greatly differing directions.

It is apparent as well, that, if am arbitrary number of people from arbitrary places start going towards the north, they will meet in the end at the north pole. In planar geometry, the parallel ways would never cross. On the other hand, two people can start on the same latitude, each going into the direction of the other, east or west. If both would keep their initial leaving direction, without “correcting” along the way using a compass, and thus

[13]It is more interesting when you also take into account the non-trivial pressure gradient.

[14]Except for the tiny problem at the poles, where no direction but the direction to the opposite pole is defined, being totally degenerated on the plane.

[15]an ideal compass that indeed always indicates geographical north, or, alternatively, a real one that is always tuned to the correct declination

[16]as ant on the earth surface, not knowing what north actually is supposed to mean on the sphere

[17]which is only the same at the equator

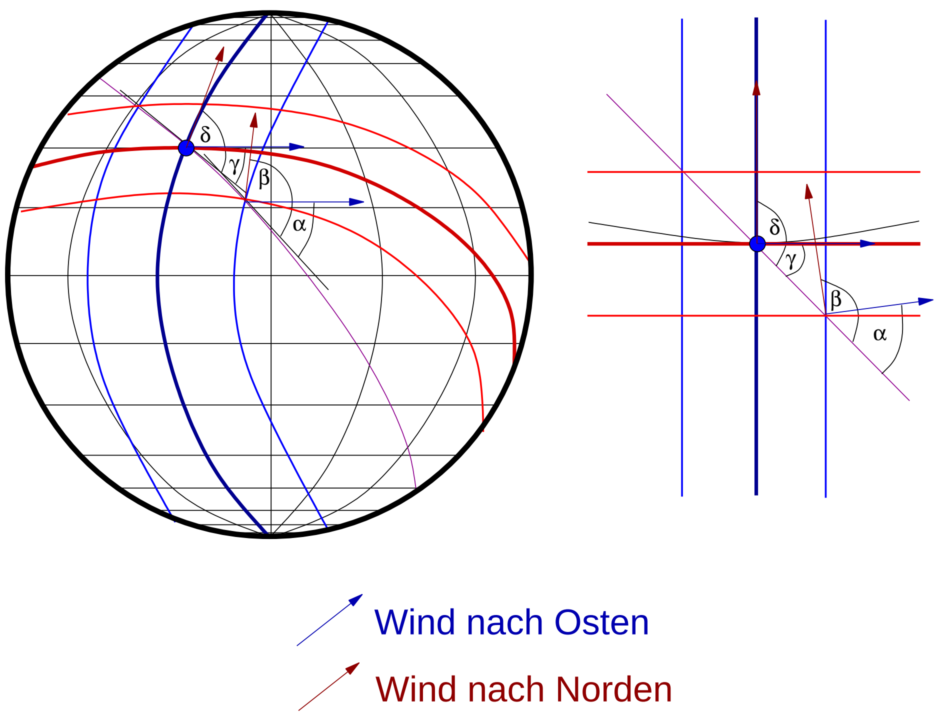


Figure 3.5: local pseudo-Cartesian coordinate system in global perspective
A coordinate system with great circles as main axes and the coordinates on the parallel small circles, corresponding to the local view of a continued north and east direction. It visualizes the changing meaning of the directions in geo coordinates. One the left side, the view on the earth sphere, on the right a display in the individual Cartesian system.

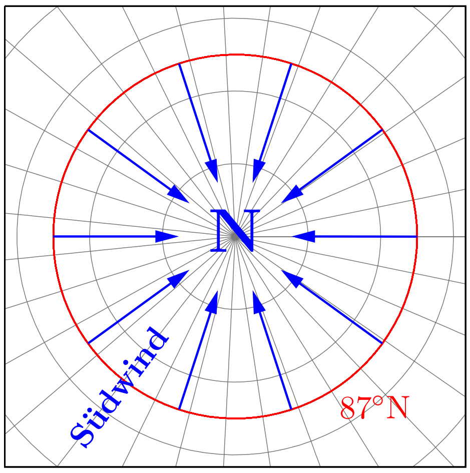


Figure 3.6: North is north?
All imagined south winds blow towards the north in this orthographic view... and they clearly point in different directions!

moving along a great circle, they could be lucky and meet at the two crossing points of their great circles, if they matched their pace closely. A meeting is not guaranteed at all.

3.6 The local transformation

The interpolation and integration in the geographical grid is no valid option for me. Therefore, the main focus is on computing the integration steps in a local coordinate system. The first incarnation of this system is an orthographic projection around the current position of the particle under consideration (sketched in Figure 3.7). Though the projection is neither correct for area size nor for angles on a global scale, it is very close to that in the local vicinity. Also, circles around the starting position keep their shape; distortions of the projection are the same in each direction.

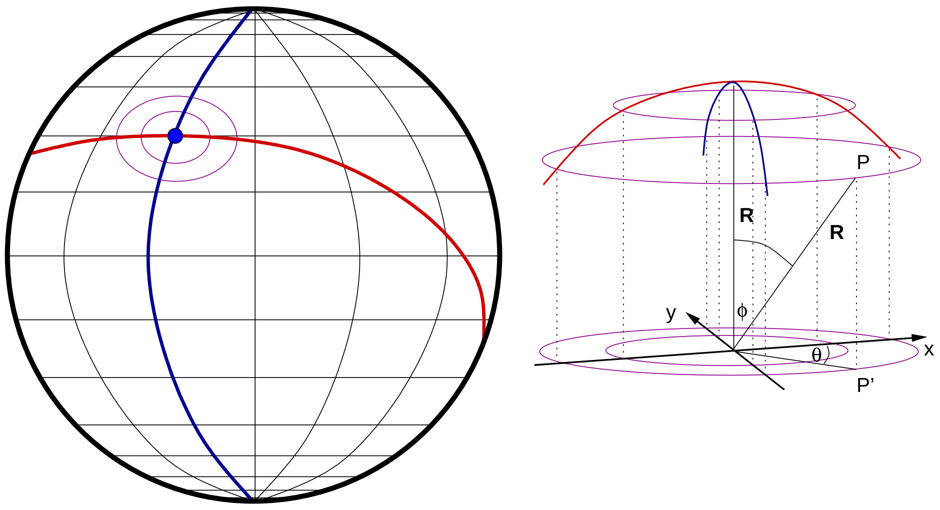


Figure 3.7: Sketch of the local orthographic projection.

In the following passages, the transformation on the surface – the vertical pressure coordinate is kept unchanged – is described. I want to express a point (λ, θ) in the coordinates (x, y) of the orthographic projection over a specific point $P = (\lambda_p, \beta_p)$.

Precursor of the actual projection is a transformation of the geographic coordinates $(\lambda, \beta$ on the earth surface (as ideal sphere) into Cartesian (x, y, z) with the z axis leading through the earth core and the point P. This z coordinate has its meaning for fulfilling the spherical geometry and is used for the back transformation, while the actual projection is nothing else than replacing that z with the pressure coordinate. The thickness of the atmosphere is neglected compared to the earth radius[18]

[18] A more advanced projection *may* include the height over the earth surface, but that does not seem sensible in this case here.

3.6.1 Equivalent representation in the rotated Cartesian system

The translation of a point on the sphere into the Cartesian system oriented towards P is executed in three stages. Beginning with the initial coordinates (λ, β) of the to-be translated point as well as the wind directions $\vec{e}_u$ and $\vec{e}_v$ in the geographic system, the intermediate values of stage i (not necessarily changed) are denoted (λ_i, β_i) or $\vec{x}_i = (x_i, y_i, z_i)$; the wind directions $\vec{e}_{u,i}$ and $\vec{e}_{v,i}$. At the end of the view in the rotated Cartesian system we have

$$\vec{x}_3 = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} \; ; \; \vec{e}_{u,3} = \begin{pmatrix} u_{x,3} \\ u_{y,3} \\ u_{z,3} \end{pmatrix} \; ; \; \vec{e}_{v,3} = \begin{pmatrix} v_{x,3} \\ v_{y,3} \\ v_{z,3} \end{pmatrix}$$

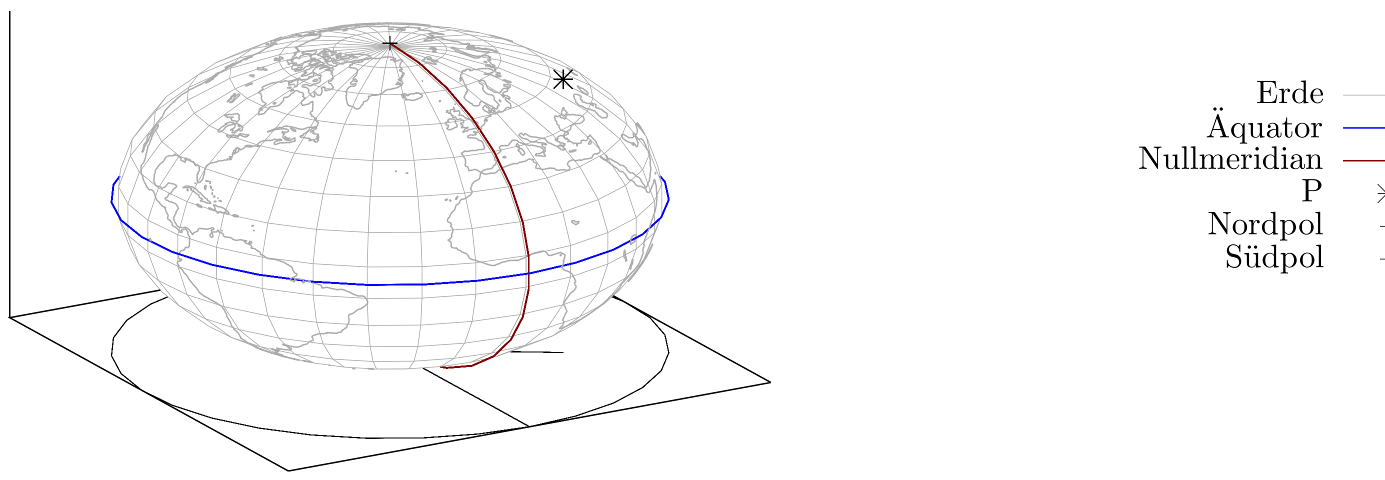


Figure 3.8: Local transformation, stage 0
This is the initial situation in the geographic system with the arbitrary point P and emphasized prime meridian and equator.

The stages themselves consist of a first rotation in stage 1 (aligning the to-be-defined x axis), the change to a Cartesian system in stage 2 and, at last, the rotation of that Cartesian system around the x axis, to direct the z axis through P.

Stage 1: Rotation around polar axis, new prime meridian

The first rotation revolves simply around the polar axis and ensures that the final rotation will be along a coordinate axis (the x one). Let

$$\begin{aligned} \lambda_1(\lambda, \beta) &= \lambda - \lambda_p - \frac{\pi}{2} \\ \beta_1(\lambda, \beta) &= \beta \end{aligned} \tag{3.23}$$

The wind directions are untouched by this, $\vec{e}_{u,1} = \vec{e}_u$; $\vec{e}_{v,1} = \vec{e}_v$. In the system of stage 1, P has the coordinates $(-\frac{\pi}{2}, \beta_p, R)$.

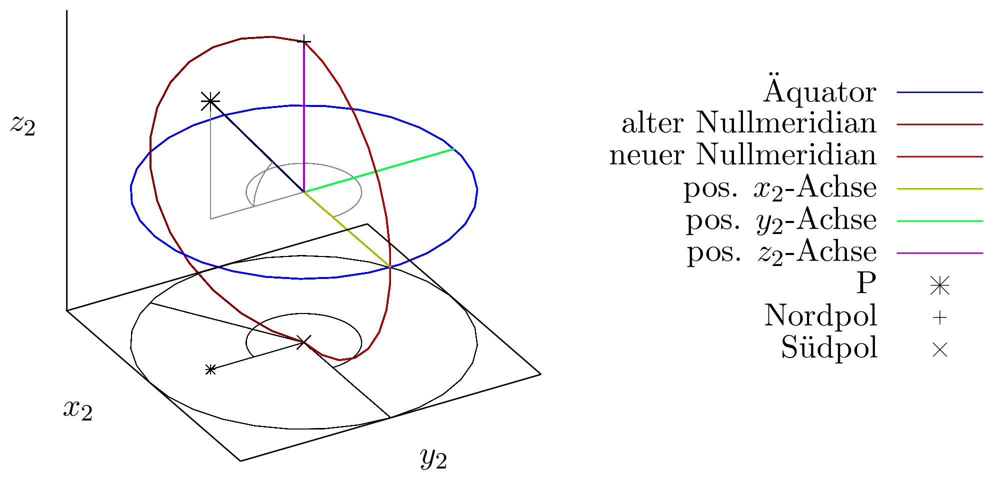


Figure 3.9: Local transformation, stage 1 and 2
Here, the first rotation (stage 1) and the definition of Cartesian coordinates (stage 2) are shown together.

Stage 2: Definition eines kartesischen Systems für die Kugel

Before performing the second rotation, I convert the spherical coordinates into three-dimensional Cartesian coordinates (see Figure 3.9).

$$\begin{aligned} x_2(\lambda_1,\beta_1) &= R\cdot\cos\beta_1\cdot\cos\lambda_1 \\ y_2(\lambda_1,\beta_1) &= R\cdot\cos\beta_1\cdot\sin\lambda_1 \\ z_2(\lambda_1,\beta_1) &= R\cdot\sin\beta_1 \end{aligned} \tag{3.24}$$

The origin is the earth center, the positive x_2 axis runs through the longitude 0 in the equator plane, the positive y_2 axis through the longitude $\frac{\pi}{2}$ (90°) and the positive z_2 axis through the north pole.

Now look at the wind directions is due; north and east direction need to be projected onto the Cartesian axes, depending on location. I need $\vec{e}_{u,1}(\lambda_1,\beta_1)$ and $\vec{e}_{v,1}(\lambda_1,\beta_1)$ in the Cartesian system to be able to transform them with the coordinates in the next stage.

The east direction $\vec{e}_u$ is always parallel to the equator, thus the z_2 component is equal to zero, since the x_2 axis is identical to the polar axis. Subsequently, the east direction in the Cartesian system of stage 2 is given by

$$\vec{e}_{u,2}(\lambda_1,\beta_1) = \begin{pmatrix} -\sin\lambda_1 \\ \cos\lambda_1 \\ 0 \end{pmatrix} \tag{3.25}$$

This is a normed unit vector: $|\vec{e}_{u,2}| = \sqrt{sin^2\lambda_1 + \cos^2\lambda_1} = \sqrt{1} = 1$

The north direction on the sphere is not that simple. There are components for all three Cartesian directions. But one can approach that step-by-step. Looking at the north unit vector...

$$\begin{aligned} e_{v,z,2} &= \cos\beta_1 \\ e_{v,xy,2} &= -\sin\beta_1 \end{aligned}$$

($e_{v,xy,2} < 0$ means: pointing towards the origin)

The xy component is portioned out depending on λ_1:

$$\begin{aligned} e_{v,x,2} &= \cos\lambda_1 e_{v,xy,2} = -\cos\lambda_1 \sin\beta_1 \\ e_{v,y,2} &= \sin\lambda_1 e_{v,xy,2} = -\sin\lambda_1 \sin\beta_1 \end{aligned}$$

As a whole:

$$\vec{e}_{v,2}(\lambda_1, \beta_1) = \begin{pmatrix} -\cos\lambda_1 \sin\beta_1 \\ -\sin\lambda_1 \sin\beta_1 \\ \cos\beta_1 \end{pmatrix} \tag{3.26}$$

I check the norm of this unit vector:

$$\begin{aligned} |\vec{e}_{v,2}| &= \cos^2\lambda_1 \sin^2\beta_1 + \sin^2\lambda_1 \sin^2\beta_1 + \cos^2\beta_1 \\ &= (1 - \sin^2\lambda_1)\sin^2\beta_1 + \sin^2\lambda_1 \sin^2\beta_1 + \cos^2\beta_1 \\ &= \sin^2\beta_1 + \cos^2\beta_1 \\ &= 1 \end{aligned}$$

The unit vectors for the wind directions are defined now; the horizontal wind blows along

$$\vec{v}(\lambda_1, \beta_1) = u \cdot \vec{e}_{u,2}(\lambda_1, \beta_1) + v \cdot \vec{e}_{v,2}(\lambda_1, \beta_1) \tag{3.27}$$

Stufe 3: Drehung um x-Achse

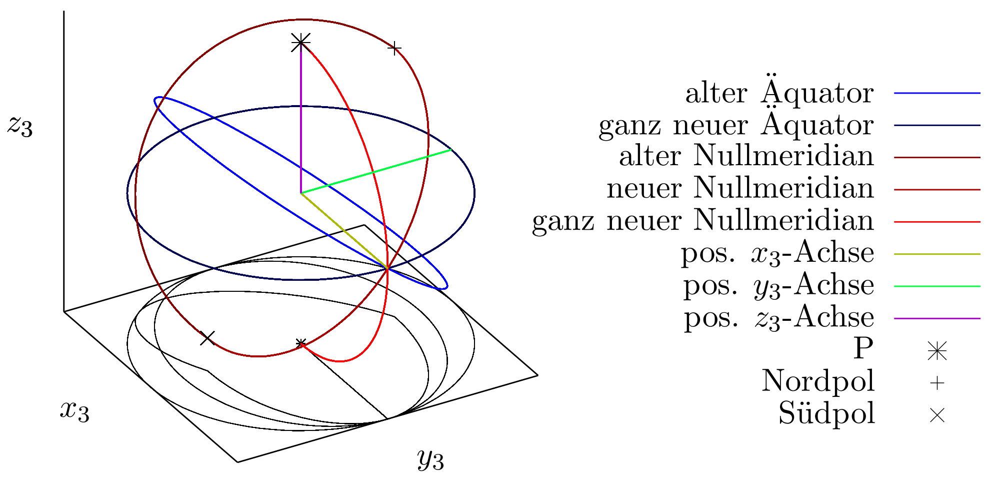


Figure 3.10: Local transformation, stage 3
The last rotation around the x_2 axis (identical with x_3 axis) leads to stage 3. The "ganz neue" prime meridian is included to help the orientation — strictly speaking, it does not exist in the system of stage 3, which is only given in Cartesian coordinates.

At last, the system need to be rotated towards P, by the angle $\frac{\pi}{2} - \beta_p$ around the x axis. Common knowledge knows the rotation matrix D_α in $\mathbb{R}^3$ for the rotation with the angle α around the x axis anti-clockwise and the inverse $D_{-\alpha}$ for the clockwise rotation.

$$D_\alpha = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \tag{3.28}$$

$$D_{-\alpha} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(-\alpha) & -\sin(-\alpha) \\ 0 & \sin(-\alpha) & \cos(-\alpha) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix} \tag{3.29}$$

I want to rotate the coordinate *system* from the north pole to P – *anti*-clockwise. That means, I rotate the *coordinates clockwise*. The absolute value of the angle is the latitude difference from the north pole to P, $\alpha = \frac{\pi}{2} - \beta_p$. For the simplification of the rotation matrix, the following identities are helpful:

$$\begin{aligned} \cos\left(\frac{\pi}{2} - \gamma\right) &= \sin\gamma \\ \sin\left(\frac{\pi}{2} - \gamma\right) &= \cos\gamma \end{aligned} \tag{3.30}$$

It follows the rotation matrix for the stage 3, $D_{-(\pi 2 - \beta_p)}$:

$$\begin{aligned} D_3 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2} - \beta_p) & \sin(\frac{\pi}{2} - \beta_p) \\ 0 & -\sin(\frac{\pi}{2} - \beta_p) & \cos(\frac{\pi}{2} - \beta_p) \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sin\beta_p & \cos\beta_p \\ 0 & -\cos\beta_p & \sin\beta_p \end{pmatrix} \end{aligned} \tag{3.31}$$

Together, those steps build the transformation of the global coordinates (λ, β) (on the earth surface) into the local coordinates (x, y, z) with the earth center as origin.

$$\begin{aligned}
\vec{x}_3 &= \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} = D_3 \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sin\beta_p & \cos\beta_p \\ 0 & -\cos\beta_p & \sin\beta_p \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \qquad (3.32) \\
&= \begin{pmatrix} x_2 \\ y_2 \sin\beta_p + z_2 \cos\beta_p \\ -y_2 \cos\beta_p + z_2 \sin\beta_p \end{pmatrix} = \begin{pmatrix} R\cdot \cos\beta_1 \cdot \cos\lambda_1 \\ R\cdot \cos\beta_1 \cdot \sin\lambda_1 \sin\beta_p + R\cdot \sin\beta_1 \cos\beta_p \\ -R\cdot \cos\beta_1 \cdot \sin\lambda_1 \cos\beta_p + R\cdot \sin\beta_1 \sin\beta_p \end{pmatrix} \\
&= R \begin{pmatrix} \cos\beta \cdot \cos(\lambda - \lambda_p - \frac{\pi}{2}) \\ \cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta_p + \sin\beta \cos\beta_p \\ -\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \cos\beta_p + \sin\beta \sin\beta_p \end{pmatrix} \\
\vec{e}_{u,3} &= D_3 \vec{e}_{u,2} \qquad (3.33) \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sin\beta_p & \cos\beta_p \\ 0 & -\cos\beta_p & \sin\beta_p \end{pmatrix} \begin{pmatrix} -\sin\lambda_1 \\ \cos\lambda_1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\sin\lambda_1 \\ \sin\beta_p \cos\lambda_1 \\ -\cos\beta_p \cos\lambda_1 \end{pmatrix} \\
&= \begin{pmatrix} -\sin(\lambda - \lambda_p - \frac{\pi}{2}) \\ \sin\beta_p \cos(\lambda - \lambda_p - \frac{\pi}{2}) \\ -\cos\beta_p \cos(\lambda - \lambda_p - \frac{\pi}{2}) \end{pmatrix} \\
\vec{e}_{v,3} &= D_3 \vec{e}_{v,2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sin\beta_p & \cos\beta_p \\ 0 & -\cos\beta_p & \sin\beta_p \end{pmatrix} \begin{pmatrix} -\cos\lambda_1 \sin\beta \\ -\sin\lambda_1 \sin\beta \\ \cos\beta \end{pmatrix} \qquad (3.34) \\
&= \begin{pmatrix} -\cos\lambda_1 \sin\beta \\ -\sin\beta_p \sin\lambda_1 \sin\beta + \cos\beta_p \cos\beta \\ \cos\beta_p \sin\lambda_1 \sin\beta + \sin\beta_p \cos\beta \end{pmatrix} \\
&= \begin{pmatrix} -\cos(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta \\ -\sin\beta_p \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta + \cos\beta_p \cos\beta \\ \cos\beta_p \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta + \sin\beta_p \cos\beta \end{pmatrix}
\end{aligned}$$

I consider it advisable to dwell here for a moment. An error can easily creep into such a derivation; a sine instead of a cosine or just a plus instead of a minus[19].

[19]☺ The human intellect is just not good at hard yes/no questions. Often you guesstimate and arrive at no, while you actually mean yes.The human answer to “Yes or no?” can, realistically, only be “Maybe?”.

Test: Spherical geometry

Up to now, only coordinate transformations have been carried out, without the projection. Accordingly, $\vec{x}_3$ should be of length R – in case the transformation is correct. I test this:

$$\begin{aligned}
|\vec{x}_3| &= \sqrt{x_3^2 + y_3^2 + z_3^2} = R \cdot \left(\left(\frac{x_3}{R}\right)^2 + \left(\frac{y_3}{R}\right)^2 + \left(\frac{z_3}{R}\right)^2 \right)^{-\frac{1}{2}} \\
\Rightarrow \left(\frac{|\vec{x}_3|}{R}\right)^2 &= \left(\cos\beta \cdot \cos(\lambda - \lambda_p - \frac{\pi}{2})\right)^2 \\
&\quad + \left(\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta_p + \sin\beta\cos\beta_p\right)^2 \\
&\quad + \left(-\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \cos\beta_p + \sin\beta\sin\beta_p\right)^2 \\
&= \cos^2\beta \underbrace{\left(\cos^2(\lambda - \lambda_p - \frac{\pi}{2}) + \sin^2(\lambda - \lambda_p - \frac{\pi}{2})(\sin^2\beta_p + \cos^2\beta_p)\right)}_{=1} \\
&\quad +2\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta_p \sin\beta\cos\beta_p \\
&\quad -2\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2}) \cos\beta_p \sin\beta\sin\beta_p \\
&\quad + \sin^2\beta(\sin^2\beta_p + \cos^2\beta_p) \\
&= \cos^2\beta + \sin^2\beta = 1
\end{aligned}$$

It is the case. Good to see. I will cover some further special test cases at P and its counter pole $P' = (\lambda_p + \pi, -\beta_p)$. This counter pole is outside the scope of the following orthographic projection, but needs to be correctly handled by the globally valid coordinate transformation.

Tests at P and counter pole P'

I consider the transformation of place and wind at P with

$$\begin{aligned}
&\lambda = \lambda_p; \ \beta = \beta_p \\
&\lambda_1 = \lambda_p - \lambda_p - \tfrac{\pi}{2} = -\tfrac{\pi}{2}
\end{aligned}$$

and at the counter pole P' with

$$\begin{aligned}
&\lambda = \lambda_p + \pi; \ \beta = -\beta_p \\
&\lambda_1 = \lambda_p + \pi - \lambda_p - \tfrac{\pi}{2} = \tfrac{\pi}{2}
\end{aligned}$$

The expected values for the $\vec{x}_3$ coordinates of the two points as well as the corresponding wind directions are obvious without explicit computation. Furthermore, $\lambda_1 = \pm\frac{\pi}{2}$ facilitates quick resolution of the angular functions via $\cos\left(\pm\frac{\pi}{2}\right) = 0$ and $\sin\left(\pm\frac{\pi}{2}\right) = \pm 1$.

Location of P For the transformation of the location of P I expect the $\vec{x}_3$ coordinates $(0, 0, R)$.

$$\begin{aligned}
\vec{x}_3(\lambda_p, \beta_p) &= R \begin{pmatrix} \cos\beta_p \cos\left(-\frac{\pi}{2}\right) \\ \cos\beta_p \sin\left(-\frac{\pi}{2}\right) \sin\beta_p + \sin\beta_p \cos\beta_p \\ -\cos\beta_p \sin\left(-\frac{\pi}{2}\right) \cos\beta_p + \sin\beta_p \sin\beta_p \end{pmatrix} \\
&= R \begin{pmatrix} 0 \\ -\cos\beta_p \sin\beta_p + \sin\beta_p \cos\beta_p \\ \cos\beta_p \cos\beta_p + \sin\beta_p \sin\beta_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ R \end{pmatrix}
\end{aligned}$$

South-wind at P The north direction at P should be translated to the y component of the transformed vector.

$$\begin{aligned}
\Rightarrow \vec{e}_{v,3}(\lambda_p, \beta_p) &= \begin{pmatrix} -\cos\left(-\frac{\pi}{2}\right) \sin\beta_1 \\ -\sin\beta_p \sin\left(-\frac{\pi}{2}\right) \sin\beta_p + \cos\beta_p \cos\beta_p \\ \cos\beta_p \sin\left(-\frac{\pi}{2}\right) \sin\beta_p + \sin\beta_p \cos\beta_p \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ \sin\beta_p \sin\beta_p + \cos\beta_p \cos\beta_p \\ -\cos\beta_p \sin\beta_p + \sin\beta_p \cos\beta_p \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}
\end{aligned}$$

West-wind at P The east direction at P should be translated to the x component of the transformed vector.

$$\vec{e}_{u,3}(\lambda_p, \beta_p) = \begin{pmatrix} -\sin\left(-\frac{\pi}{2}\right) \\ \sin\beta_p \cos\left(-\frac{\pi}{2}\right) \\ -\cos\beta_p \cos\left(-\frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Location of P' For the transformation of the location of P' I expect the coordinates of P with inverted z_3.

$$\begin{aligned}
\vec{x}_3(\lambda_p + \pi, -\beta_p) &= R \begin{pmatrix} \cos(-\beta_p) \cos\frac{\pi}{2} \\ \cos(-\beta_p) \sin\frac{\pi}{2} \sin\beta_p + \sin(-\beta_p) \cos\beta_p \\ -\cos(-\beta_p) \sin\frac{\pi}{2} \cos\beta_p + \sin(-\beta_p) \sin\beta_p \end{pmatrix} \\
&= R \begin{pmatrix} 0 \\ \cos\beta_p \sin\beta_p - \sin\beta_p \cos\beta_p \\ -\cos\beta_p \cos\beta_p - \sin\beta_p \sin\beta_p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -R \end{pmatrix}
\end{aligned}$$

South-wind at P' The north direction at P' should, just like the one at P, be translated to the y component of the transformed vector.

$$\begin{aligned}
\vec{e}_{v,3}(\lambda_p + \pi, -\beta_p) &= \begin{pmatrix} -\cos\frac{\pi}{2} \sin(-\beta_p) \\ -\sin\beta_p \sin\frac{\pi}{2} \sin(-\beta_p) + \cos\beta_p \cos(-\beta_p) \\ \cos\beta_p \sin\frac{\pi}{2} \sin(-\beta_p) + \sin\beta_p \cos(-\beta_p) \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ \sin\beta_p \sin\beta_p + \cos\beta_p \cos\beta_p \\ -\cos\beta_p \sin\beta_p + \sin\beta_p \cos\beta_p \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}
\end{aligned}$$

West-wind at P' The east direction at P' should be translated to the x component of the transformed vector, but with negative sign, because east points to the exact opposite direction on the other side of the earth.

$$\vec{e}_{u,3}(\lambda_p + \pi, -\beta_p) = \begin{pmatrix} -\sin\frac{\pi}{2} \\ \sin\beta_p \cos\frac{\pi}{2} \\ -\cos\beta_p \cos\frac{\pi}{2} \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$$

3.6.2 Back-transformation into the geographical system

After performing the integration step in the local Cartesian system, the newly computed location needs to be mapped back into the global geographical system. For that, I retrace the stages backwards, starting with the rotation of stage 3. I must rotate back by the angle $\frac{\pi}{2} - \beta_p$, which is equivalent to the inversion of D_3. The resulting matrix for $\alpha = \frac{\pi}{2} - \beta_p$ is written down in Equation 3.28.

$$\begin{aligned} \vec{x}_2 &= D^{-1}\vec{x}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2} - \beta_p) & -\sin(\frac{\pi}{2} - \beta_p) \\ 0 & \sin(\frac{\pi}{2} - \beta_p) & \cos(\frac{\pi}{2} - \beta_p) \end{pmatrix} \vec{x}_3 \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sin\beta_p & -\cos\beta_p \\ 0 & \cos\beta_p & \sin\beta_p \end{pmatrix} \vec{x}_3 = \begin{pmatrix} x_3 \\ y_3 \sin\beta_p - z_3 \cos\beta_p \\ y_3 \cos\beta_p + z_3 \sin\beta_p \end{pmatrix} \end{aligned} \tag{3.35}$$

After the backwards rotation, stage 2 needs to be reverted: The transformation from Cartesian into geographic spherical coordinates. The system Equation 3.24 needs to be resolved for λ_1 and β_1.

$$\begin{aligned} G1: &\quad \frac{x_2}{R} = \cos\beta_1 \cdot \cos\lambda_1 \\ G2: &\quad \frac{y_2}{R} = \cos\beta_1 \cdot \sin\lambda_1 \\ G3: &\quad \frac{z_2}{R} = \sin\beta_1 \end{aligned} \tag{3.36}$$

The solution is found over the tangent function; I look for

$$\begin{aligned} \tan\lambda_1 &= \frac{\sin\lambda_1}{\cos\lambda_1} \\ \tan\beta_1 &= \frac{\sin\beta_1}{\cos\beta_1} \end{aligned}$$

what is quickly successful for λ_1:

$$G2/G1: \quad \frac{\sin\lambda_1}{\cos\lambda_1} = \frac{y_2}{x_2}$$

For β_1 we need a bit more work:

$$\begin{aligned}
G1^2: &\quad \frac{x_2^2}{R^2} = \cos^2\beta_1 \cos^2\lambda_1 \\
G2^2: &\quad \frac{y_2^2}{R^2} = \cos^2\beta_1 \sin^2\lambda_1 \\
G1^2 + G2^2: &\quad \frac{1}{R^2}(x_2^2 + y_2^2) = \cos^2\beta_1 \underbrace{(\sin^2\lambda_1 + \cos^2\lambda_1)}_{=1} = \cos^2\beta_1 \\
\frac{G3}{\sqrt{G1^2 + G2^2}}: &\quad \frac{z_2}{\sqrt{x_2^2 + y_2^2}} = \frac{\sin\beta_1}{\cos\beta_1}
\end{aligned}$$

So, both tangent values are found and the angles are given by:

$$\begin{aligned}
\tan\lambda_1 = \frac{\sin\lambda_1}{\cos\lambda_1} &= \frac{y_2}{x_2} \\
\tan\beta_1 = \frac{\sin\beta_1}{\cos\beta_1} &= \frac{z_2}{\sqrt{x_2^2 + y_2^2}}
\end{aligned} \tag{3.37}$$

In the literature (see [Bronstein2000]), one can find the conversion from Cartesian to "mathematical" spherical coordinates, differing from geographical coordinates:

$$\begin{aligned}
\tan\lambda &= \frac{y}{x} \\
\tan\beta &= \frac{\sqrt{x^2 + y^2}}{z}
\end{aligned} \tag{3.38}$$

This can be identified with the geographical coordinates used here, by invoking the definition of the latitude β and using Equation 3.30:

$$\begin{aligned}
&\beta_{\text{geo}} = \frac{\pi}{2} - \beta_{\text{math}} \\
\Rightarrow \quad &\sin\beta_{\text{geo}} = \sin\left(\frac{\pi}{2} - \beta_{\text{math}}\right) = \cos\beta_{\text{math}} \\
&\cos\beta_{\text{geo}} = \cos\left(\frac{\pi}{2} - \beta_{\text{math}}\right) = \sin\beta_{\text{math}} \\
\Rightarrow \quad &\tan\beta_{\text{geo}} = \frac{\sin\beta_{\text{geo}}}{\cos\beta_{\text{geo}}} = \frac{\cos\beta_{\text{math}}}{\sin\beta_{\text{math}}} = (\tan\beta_{\text{math}})^{-1}
\end{aligned}$$

The tangent inversion The arctan function will produce angles again, out of the Cartesian coordinates. By definition, that function is only definite when restricted to a certain interval. Since I aim for the work with electronic computing devices, I do not concentrate just on theoretical (albeit sensible) conventions, but rather ask the machine directly. That results in[20]:

```
shell$ rechne 'atan(0)'
0
shell$ rechne 'atan(1000)'
```

[20]`rechne` is a Perl program that evaluates the given formula and prints out the result. It is using the same C library functions as my trajectory program.

```
1.56979632712823
shell$ rechne 'atan(-1000)'
-1.56979632712823
```

The practically chosen interval is thus

$$\arctan \quad : \quad (-\infty;\infty) \to \left[-\frac{\pi}{2};\frac{\pi}{2}\right] \tag{3.39}$$

This is also exactly what the ISO C90 standard[21] specifies[22]

Out of

$$\lambda_1 = \arctan\frac{y_2}{x_2} \tag{3.40}$$

$$\beta_1 = \arctan\frac{z_2}{\sqrt{x_2^2+y_2^2}} \tag{3.41}$$

I directly get values for β_1 inside the desired interval ($\beta_1 < 0$ for $z < 0$ and $\beta_1 > 0$ for $z > 0$) and for λ_1, I need to additionally check, if $x_2 < 0$ ti decide whether the value out of $\left[-\frac{\pi}{2};\frac{\pi}{2}\right]$ needs to be shifted to the other side by π.

The trouble points: While you can define arctan for $\pm\infty$ to $\pm\frac{\pi}{2}$ as well as here $\frac{0}{0}=0$ and get sensible values also for borderline $\vec{x}_2$, it is advisable to treat the corner cases specifically regarding the programming of a computer. The computation Equation 3.41 is becoming problematic when x_2 and y_2 are both zero. But then β_1 is properly defined to be valued $\frac{\pi}{2}$ if z_2 is positive and $-\frac{\pi}{2}$ if z_2 is negative. Equation 3.40 becomes problematic if x_2 equals zero. Then, λ equals $\frac{\pi}{2}$ for positive y_2 and $-\frac{\pi}{2}$ for negative y. In case both x_2 and y_2 being equal to zero, the value of λ_1 is irrelevant.

Well, having arrived back in the system of stage 1 with λ_1 and θ_1, then all it needs is

$$\begin{aligned}\beta &= \beta_1 \\ \lambda &= \lambda_1 + \lambda_p + \frac{\pi}{2}\end{aligned} \tag{3.42}$$

to arrive at the normal geographic coordinates.

[21] ISO/IEC 9899-1990, Programming Languages - C

[22] One should neither blindly depend on the standard nor a specific implementation. That helps to prevent nasty surprises.

An algorithm for the complete back-transformation of the place from $\vec{x}_3$ to (λ, β), including the corner cases, is laid out in Equation 3.43:

$$
\begin{gathered}
\begin{aligned}
x_2 &= x_3\\
y_2 &= y_3 \sin\beta_p - z_3 \cos\beta_p\\
z_2 &= y_3 \cos\beta_p + z_3 \sin\beta_p
\end{aligned}
\\[1em]
\mathbf{x_2 = 0} \left\| \begin{array}{ll}
 & \lambda_1 = \left(y_2 > 0 \;\mathbf{?}\; \frac{\pi}{2} \;\mathbf{:}\; -\frac{\pi}{2}\right) \\
\mathbf{?} & \mathbf{y_2 = 0} \left\| \begin{array}{l}
\mathbf{?}\;\; \beta_1 = \left(z_2 > 0 \;\mathbf{?}\; \frac{\pi}{2} \;\mathbf{:}\; -\frac{\pi}{2}\right) \\ \hline
\mathbf{:}\;\; \beta_1 = \arctan \frac{z_2}{\sqrt{x_2^2 + y_2^2}} = \arctan \frac{z_2}{y_2}
\end{array}\right. \\ \hline
\mathbf{:} & \begin{array}{l}
\lambda_1 = \arctan \frac{y_2}{x_2} + (x_2 < 0 \;\mathbf{?}\; \pi \;\mathbf{:}\; 0) \\
\beta_1 = \arctan \frac{z_2}{\sqrt{x_2^2 + y_2^2}}
\end{array}
\end{array}\right.
\\[1em]
\begin{aligned}
\lambda &= \lambda_1 + \lambda_p + \tfrac{\pi}{2}\\
\beta &= \beta_1
\end{aligned}
\end{gathered}
\tag{3.43}
$$

For $x_2 = 0$, the longitude is set to $\frac{\pi}{2}$ for positive y_2 and $-\frac{\pi}{2}$ for negative y_2. If also $y_2 = 0$, we are located at one of the poles, where the longitude can be arbitrarily chosen and the latitude is fixed at $\frac{\pi}{2}$ for positive z_2 (north pole) and $-\frac{\pi}{2}$ for negative z (south pole).

Even though IEEE floating point math includes some treatment of division by 0, it is safer to treat the obvious cases explicitly. Especially, 0/0 would lead to `NaN` otherwise, which is not helpful.

In addition, there is the correct determination of λ_1 – for $x_2 < 0$ it is the counter angle on the other side of the globe.

3.6.3 Orthographic projection, transformation forwards and backwards

The actual orthographic projection is the simple ignoring of the z_3 coordinate (projection from infinite distance onto the $x_3 y_3$ plane), the geometric view of the local movement on a xy tangential plane around the center of interest, P. Instead of z_3 the pressure coordinate in the athmosphere, which has been treated as flat in the transformation, is used[23].

The work coordinates are

$$\vec{x} \quad = \quad \begin{pmatrix} x \\ y \\ p \end{pmatrix} \tag{3.44}$$

with the pressure p as third coordinate, with the velocity

$$\dot{\vec{x}} \quad = \quad \begin{pmatrix} u e_{u,x} + v e_{v,x} \\ u e_{u,y} + v e_{v,y} \\ w \end{pmatrix} \tag{3.45}$$

An integration step results in new coordinates x, y and p, while p stays unchanged during the back-transformation into the global system and x, y lead to λ, β. A z is needed for that

[23]Update: This is different in more recent versions of the PEP-Tracer software.

back-transformation in an indirect way: It is preserving the fixed distance from the earth center, the earth radius R.

$$\begin{aligned} z^2 &= R^2 - (x^2 + y^2) \\ z &= +\sqrt{(R^2 - (x^2 + y^2))} \end{aligned} \tag{3.46}$$

This is used instead of the "real" z_3. Behind the choice of the positive root hides the limitation on the hemisphere around P, only dealing with locations that are at most a quarter earth circumference away from P. Since this is about a *local* coordinate system, that poses no problem – I will not consider computations that include movements of thousands of kilometers for one local integration step.

To summarize: The transformation for the basis point $P(\lambda_p, \beta_p)$ and the coordinates $\vec{r} = (\lambda, \beta, p)$ in the global system and $\vec{x} = (x, y, p)$ in the local system.

Location – forward It begins with a rotation around the polar axis, introducing the angles λ_1 and β_1 (while $\beta_1 = \beta$). Then the reinterpretation of the world in Cartesian coordinates follows: $\vec{x}_2$. Last step is the rotation of the system in those coordinates by the angle $\frac{\pi}{2} - \beta_p$: $\vec{x}_3$. In the end, we have the local coordinates $\vec{x}$, which consist of the first two components of $\vec{x}_3$ and the unchanged pressure coordinate p.

$$\vec{x} = \begin{pmatrix} R\cos\beta \cdot \cos(\lambda - \lambda_p - \frac{\pi}{2}) \\ R\left(\cos\beta \cdot \sin(\lambda - \lambda_p - \frac{\pi}{2})\sin\beta_p + \sin\beta\cos\beta_p\right) \\ p \end{pmatrix} \tag{3.47}$$

Location – backward One needs to keep in mind that, here, instead of z the spherical condition 3.46 is to be used.

$$\begin{aligned} \lambda &= \arctan\left(\frac{y\sin\beta_p - \sqrt{(R^2 - (x^2 + y^2))}\cos\beta_p}{x}\right) + \lambda_p + \frac{\pi}{2} \\ &\quad + (x < 0 \,\mathbf{?}\, \pi \,\mathbf{:}\, 0) \end{aligned} \tag{3.48}$$

$$\beta = \arctan\left(\frac{y\cos\beta_p + \sqrt{(R^2 - (x^2 + y^2))}\sin\beta_p}{\sqrt{(x)^2 + \left(y\sin\beta_p - \sqrt{(R^2 - (x^2 + y^2))}\cos\beta_p\right)^2}}\right) \tag{3.49}$$

Wind Also here, we do not let the horizontal wind (global: u and v) blow in z (or rather p) direction. The local 3D wind consists of projections of u and v together with the unchanged vertical component ω.

$$
\begin{aligned}
\vec{e}_u &= \begin{pmatrix} -\sin(\lambda - \lambda_p - \frac{\pi}{2}) \\ \sin\beta_p \cos(\lambda - \lambda_p - \frac{\pi}{2}) \\ 0 \end{pmatrix} \\
\vec{e}_v &= \begin{pmatrix} -\cos(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta \\ -\sin\beta_p \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta + \cos\beta_p \cos\beta \\ 0 \end{pmatrix} \\
\vec{e}_w &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
\dot{\vec{x}} &= u\vec{e}_u + v\vec{e}_v + w\vec{e}_w \qquad (3.50) \\
&= \begin{pmatrix} -u\sin(\lambda - \lambda_p - \frac{\pi}{2}) - v\cos(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta \\ u\sin\beta_p \cos(\lambda - \lambda_p - \frac{\pi}{2}) + v\left(-\sin\beta_p \sin(\lambda - \lambda_p - \frac{\pi}{2}) \sin\beta + \cos\beta_p \cos\beta\right) \\ w \end{pmatrix}
\end{aligned}
$$

3.7 3D integration in the local system

Now I have Equation 3.47 up to Equation 3.50 to work in a local system around point $P_0 = (\lambda_0, \beta_0, p_0)$. What do I do with that? I want to get from P_0 to P_1 in an integration step (time step Δt). The basic algorithm is laid out in three stages:

1. Change into the local Cartesian system: $P_0 = (\lambda_0, \beta_0, p_0) \to (0, 0, p_0)_{\text{lok}}$
2. Integration step to $(x_1, y_1, p_1)_{\text{lok}} = (\Delta x, \Delta y, p + \Delta p)_{\text{lok}}$
3. Change to global system: $(x_1, y_1, p_1)_{\text{lok}} \to (\lambda_1, \beta_1, p_1) = P_1$

In the course of the integration I will need interpolated wind speeds at different locations $Q = (x, y, p)_{\text{lok}}$ in the local system. For that I need $Q = (\lambda(x, y), \beta(x, y), p)$ in the global system, to find the encompassing grid points[24] $G_n(\lambda_n, \beta_n, p_n)$ in the coordinate space of the wind data source as well as read out the winds $\dot{\vec{r}}_n$ themselves. In the local system those are $G_n = (x_n(\lambda_n, \beta_n), y_n(\lambda_n, \beta_n))_{\text{lok}}$ and $\dot{\vec{x}}_n$. Due to the coordinate transformation, at least the horizontal grid cells are not rectangular anymore and in vicinity of the poles they are resembling circles instead. Thus, the simple linear interpolation from Equation 3.13 can only be used in height and time. In the xy plane its place is taken over by the distance-weighted sum out of Equation 3.18, which can work with arbitrary distribution of points[25].

The function `ensemble_rk4::single_step()` in Listing 3.6 unifies these building steps and shows some moves (of simulated air parcels). The coordinate system is hidden behind an abstraction. It can also be just the identity with the global system, so that this routine can be employed for the plain integration in the geographical grid, too.

[24] usually the 16 corners of the space-time grid cell, at the poles all points with the northernmost or southernmost latitude

[25] An envisioned improvement is the inclusion of the closeness of the grid points to each other.

Listing 3.6: Single RK4 integration step

```
void ensemble_rk4::single_step(place pos)
{
  place workplace;  // scratch variable for place (x)
  place localstart; // start place in local system
  spaceplace k[4];  // the RK4 coefficients
  spaceplace half;  // scratch var for k/2
  home->system->set_base(pos); // move local system
  home->system->lplace(pos, localstart); // get local coordinates
  // k1 = h*f(x0, y0)
  if(home->get_lwind(localstart,k[0])) // get local wind
  {
    smul_space(k[0], timestep, k[0]);
    // k2 = h*f(x0+h/2, y0+k1/2)
    sdiv_space(k[0], 2, half);
    add_space(localstart, half, workplace)
    workplace[TIME] = localstart[TIME]+timestep/2;
    if(home->get_lwind(workplace, k[1]))
    {
      smul_space(k[1], timestep, k[1]);
      // k3 = h*f(x0+h/2, y0+k2/2)
      sdiv_space(k[1], 2, half);
      add_space(localstart, half, workplace);
      // still: workplace[TIME] = localstart[TIME]+timestep/2;
      if(home->get_lwind(workplace, k[2]))
      {
        smul_space(k[2], timestep, k[2]);
        // k4 = h*f(x0+h, y0+k3)
        add_space(localstart, k[2], workplace);
        workplace[TIME] = localstart[TIME]+timestep;
        if(home->get_lwind(workplace, k[3]))
        {
          smul_space(k[3], timestep, k[3]);
          // x1 = x0 + (k0 + 2k1 + 2 k2 + k3)/6
          for(size_t i = 0; i < SPACEDIMS; ++i)
          workplace[i] = localstart[i]
                       + (k[0][i] + k[1][i]*2 + k[2][i]*2 + k[3][i
                         ])/6;

          home->system->gplace(workplace, pos);
} } } } }
```

This routine is embedded in a class `ensemble_rk4`, which represents a sub-ensemble of particles together. Using the classes `egp_wind` as introduced in section 2.2 (behind the abstraction of the `data` interface), `ensemble_rk4` (behind the `ensemble` interface), the local coordinate transformation in `flat_pressure` (`coordinates`) and `world` (default implementation of the `world` interface) to connect data, coordinates and interpolation, few program code lines are enough to execute a large-scale numerical tracer experiment. (Listing 3.7).

A remark to the large scale: You can scale properly by dividing the overall ensemble into partial ensembles (blocks) to facilitate serial (to fit into computer memory, manage smaller files) or parallel (using a cluster) work. Block sizes in use for my experiments are between 1000 and 100,000 particles; the performance optimum depends on parameters like the CPU cache size.

Listing 3.7: Prototype of a tracer experiment

```
// parameters
pep_t begin   = START_TIME;
pep_t end     = END_TIME;
pep_t step    = TIMESTEP;
long nsteps   = (long) ((end-begin)/step);
pep_t members = NUMBER_OF_PARTICLES;
// compute/set the starting positions somehow
spaceplace *start = GET_START_POS(NUMBER_OF_PARTICLES);
// data source for ECHO-GiSP NetCDF files
data *pepper = new egp_wind; // could be some different source
    ...
pepper->init(datafiles, number_of_files);
world *w = new world; // the default world with lokal
    coordinates
coordinates *s = new flat_pressure; // the coordinate
    transformations
// connect the world with coordinates and data source
w->set_system(s);
w->set_source(pepper);
// create the ensemble and place it into our world
ensemble *p = new ensemble_rk4;
p->set_home(w);
p->set_timestep(step);
p->set_starttime(begin)
p->set_members(members);
p->beam(start); // set the starting positions
do
{
  p->step(); // advance the ensemble one timestep
  do_something_with(p->position);
}
while(p->steps < nsteps);
```

4 Global ensembles

4.1 Starting positions

After having resolved *how* the movement happens, the question about*what* to move arises. This question is reduced to the starting position at a certain time for the passive tracers[1] under consideration. For the ensemble: many starting positions in a predefined range. When I want to undertake quantitative investigations of mixing of air masses of differing origin or the distribution of air from a limited starting region, I need to associate some weight / represented air mass with the particles (sub-ensembles) or simply manage to make them all represent the same mass so that simple counts have a direct meaning.

In approximation, one can treat the air density in one pressure level as constant. But one cannot ignore the vertical density gradient (barometric formula!).

The biggest possible picture concerns ensembles distributed over whole air layers or over the whole world (as covered by the data source). Later investigations will probably be dealing with more local scope (limited starting regions), e.g. to achieve higher resolution and more reliable mass relations. The global view, once worked out, provides a basis that can be simplified on demand. Indeed I begin with global runs, using ensembles distributed throughout the whole atmosphere, within the limits of available storage space, to evaluate global transport features; as basic evaluation of the method. The global evolution of the mass distribution by the global transport enables a comparison with the density that can be derived from ECHO-GiSP data. Ideally, the climate model and the Lagrangian transport computation should agree over the distribution of air mass – in reality that means that they should show a reasonable ratio that stays valid for a certain time.

4.1.1 Global starting positions in a level (dh)

In the horizontal, evenly distributed starting positions on the surface form a sensible ansatz. Neglecting the variation in density on the level, that ensures the same weight for each particle. The relations of particle counts in a region indicate the relations of air masses. Even if density variations are to be considered in future, the even distribution starting positions according to a mean density is a good starting point.

A simple deterministic positioning on the surface follows from the fixation of starting positions on latitude circles while scaling the number of positions with the circumference of these circles. The identifying parameter for this positioning is the even number h, determining the number of starting positions on the equator. I use the naming scheme dh for the positioning with parameter h. So, "d180" denotes the positioning with 180 positions on the equator and

[1]The term of the passive tracer means means a specific air mass that resides at a specific place (or its vicinity, see section 5.2) at the starting time.

"d1000" the positioning with 1000 of these. Two formulas for β and λ define the individual positions.

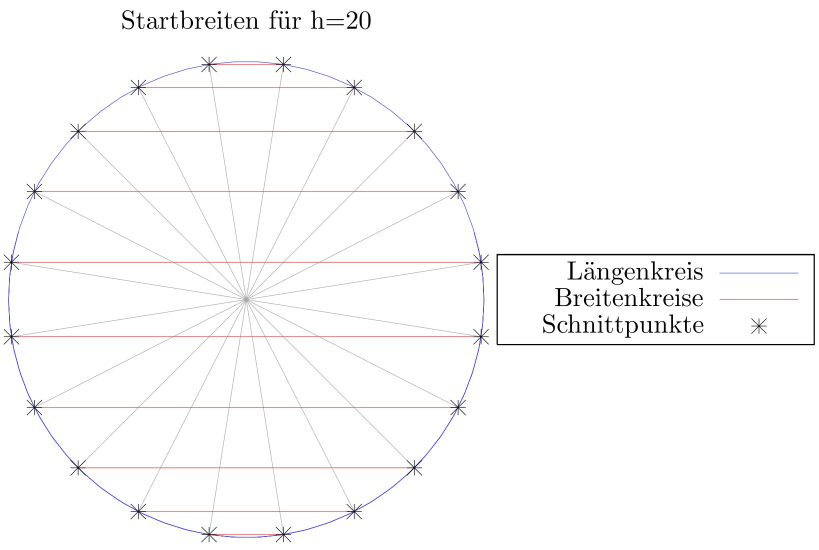


Figure 4.1: Even partition of latitudes

$$\beta_a = \pi\left[\left(a+\frac{1}{2}\right)\frac{2}{h}-\frac{1}{2}\right];\ a \in \left[0; \frac{h}{2}-1\right] \tag{4.1}$$

$$\lambda_{a,o} = o \cdot \frac{2\pi}{\operatorname{ganz}\left(h \cdot cos\beta_a\right)} - \pi;\ o \in [0; \operatorname{ganz}\left(h\cos\beta_a\right)-1]$$

$(\lambda \in [-\pi;\pi)\ \lambda \in [-180°; 180°)$, h even number)

I distribute $\frac{h}{2}$ latitude circles spaced by $\frac{2\pi}{h}$ from each other.

$$\left[-\frac{\pi}{2}+\frac{\pi}{2h}; \frac{\pi}{2}-\frac{\pi}{2h}\right]$$

That means that the prolongation of a meridian to a full circle has h crossings with these latitude circles with uniform distance of $R\frac{2\pi}{h}$. (in meters) – also over the poles, like depicted in Figure 4.1. On these latitude circles, positions λ_o are set, spaced via

$$r_{\mathrm{b}}(\beta_a)\frac{2\pi}{\operatorname{ganz}\left(h \cdot cos\beta_a\right)} = R\cos\beta_a\frac{2\pi}{\operatorname{ganz}\left(h \cdot cos\beta_a\right)}$$

For big h, where $\operatorname{ganz}\left(h \cdot cos\beta_a\right) \approx h \cdot cos\beta_a$, this spacing is the same as the spacing in latitude direction:

$$R\cos\beta_a\frac{2\pi}{\operatorname{ganz}\left(h \cdot cos\beta_a\right)} \overset{h>>1}{\approx} R\cos\beta_a\frac{2\pi}{h \cdot cos\beta_a} = R\frac{2\pi}{h} \tag{4.2}$$

This positioning provides, for not too small h, starting positions with the same distance – in meters – to the next neighbor. With different wording: The positions are uniformly distributed on the sphere surface and I begin with an approximately constant area density of particles.

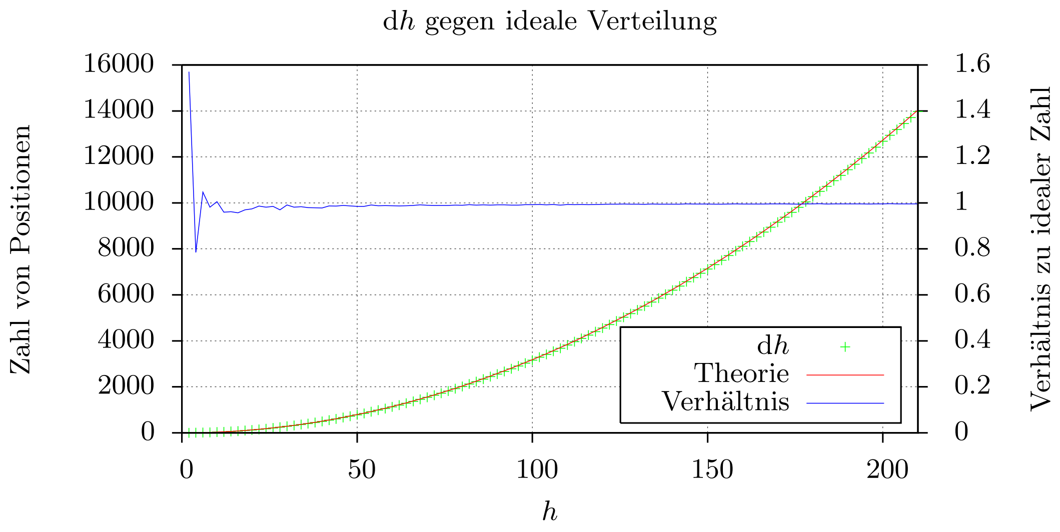


Figure 4.2: Comparison of number of positions in dh with $\frac{h^2}{\pi}$

The value of h is a direct measure for the area density. It is defined as the one-dimensional density of starting positions on the equator, h positions on one earth circumference, ruling the one-dimensional density $h\,(2\pi R)^{-1}$. The positioning is constructed to achieve the same one-dimensional density in latitude direction, so that the area density, for sufficiently large h, is given by $h^2\,(2\pi R)^{-2}$. A simple test of the uniformity of the dh positioning consists of the comparison of the total number of particles on the surface compared with the expected analytical value for uniform density, $h^2\,(2\pi R)^{-2} \cdot 4\pi R^2 = h^2\pi^{-1}$. Figure 4.2 shows appropriate agreement[2].

4.1.2 Starting positions in whole atmosphere (dh-v)

In the vertical direction one would need to lower the density of starting positions according to the exponentially decreasing air density. But this is not practical. Either you have no particles at all or just too few of them in higher levels to be able to trace the dynamics, or you have to compute just too many particles in the lower levels.

My implemented approach for a global positioning is based on the prescribed pressure levels from the ECHO-GiSP data and sub levels placed according to a parameter v. The notation dh-v means the the distribution of v starting levels of the positioning dh for each of the n_p pressure levels of the model data (but not above the topmost level).

$$p_{n_p,1} = p_{n_p};\ \forall k \in [1, n_p - 1]:\ \forall i \in [1; v]:\ p_{k,i} = p_k + (i-1)\frac{p_{k+1} - p_k}{v}$$

This placement developed out of the simplest one that just took the model levels. In future, this will probably be replaced by a consistent logarithmic placement of levels[3].

The whole positioning scheme with $n_p \cdot v$ layers gets name dh-v. In the next chapter, I will analyze d210-3 and d1000-3 runs, meaning each 3 starting levels per model level and 210

[2]Exact agreement is not possible as I do not place *fractions* of particles but only whole particles.
[3]Update: This is the case in later PEP-Tracer versions.

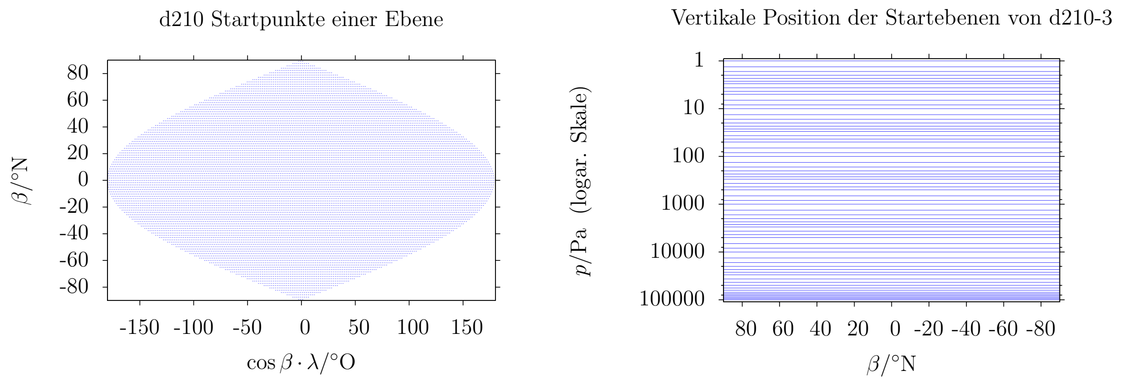


Figure 4.3: View on the horizontal and vertical positioning of d210-3
Depending on print quality, the horizontal start positioning on the left side should appear as many uniformly distributed particles or just as a colored area.

and 1000 particles per great circle, respectively. An impression of d210-3 shall be delivered by Figure 4.3 geben.

4.2 10 days of to transport

Now it has been settled *how* and by which *means what* is transported.
How: A method for integration of trajectories has been developed, accumulated in the program `ensemble_run`, which in essence shares the work flow with Listing 3.7. *Means:* I have access to the wind fields from ECHO-GiSP through the interface of `egp_wind`. *What:* With dh-v, a system for the creation of global and local particle ensembles with homogeneous horizontal density in a pressure level[4] is given.

To demonstrate the scheme, I show views on some test runs[5], that I am creating on my laptop Neuling just now, while writing this. This is no problem for the chosen particle counts (up to 8744) and the number of integration intervals (200 with time step of 0.05 days and ten days whole duration). The computing time is just a few minutes[6].

The first views dwell on the development of two global d160 positionings on one pressure level in the troposphere (850hPa, Figure 4.4) and one in he stratosphere (10hPa, Figure 4.5), respectively. The snapshots in horizontal view at least indicate the differing characteristics of the dynamics in these air layers. The movement out of 10hPa is dominated by the polar vortex, while out of 850hPa there is rather small-scale mixing. But for both the zonal channeling of the winds and thus the equator as transport barrier is strongly evident, at least for the time scale of up to ten days.

[4]In disregard of the actually varying air density in one pressure level. I will come back to that in the next chapter.
[5]The computations run during the first days of the model year 1965, with wind data from the interactive run.
[6]Quoting `ensemble_run`: "Statistics: 8110 trajectories of 200 steps in 157 seconds (0.0193588s per trajectory, 10331.2ss/s)", so, below three minutes for 8110 trajectories of 200 time steps.

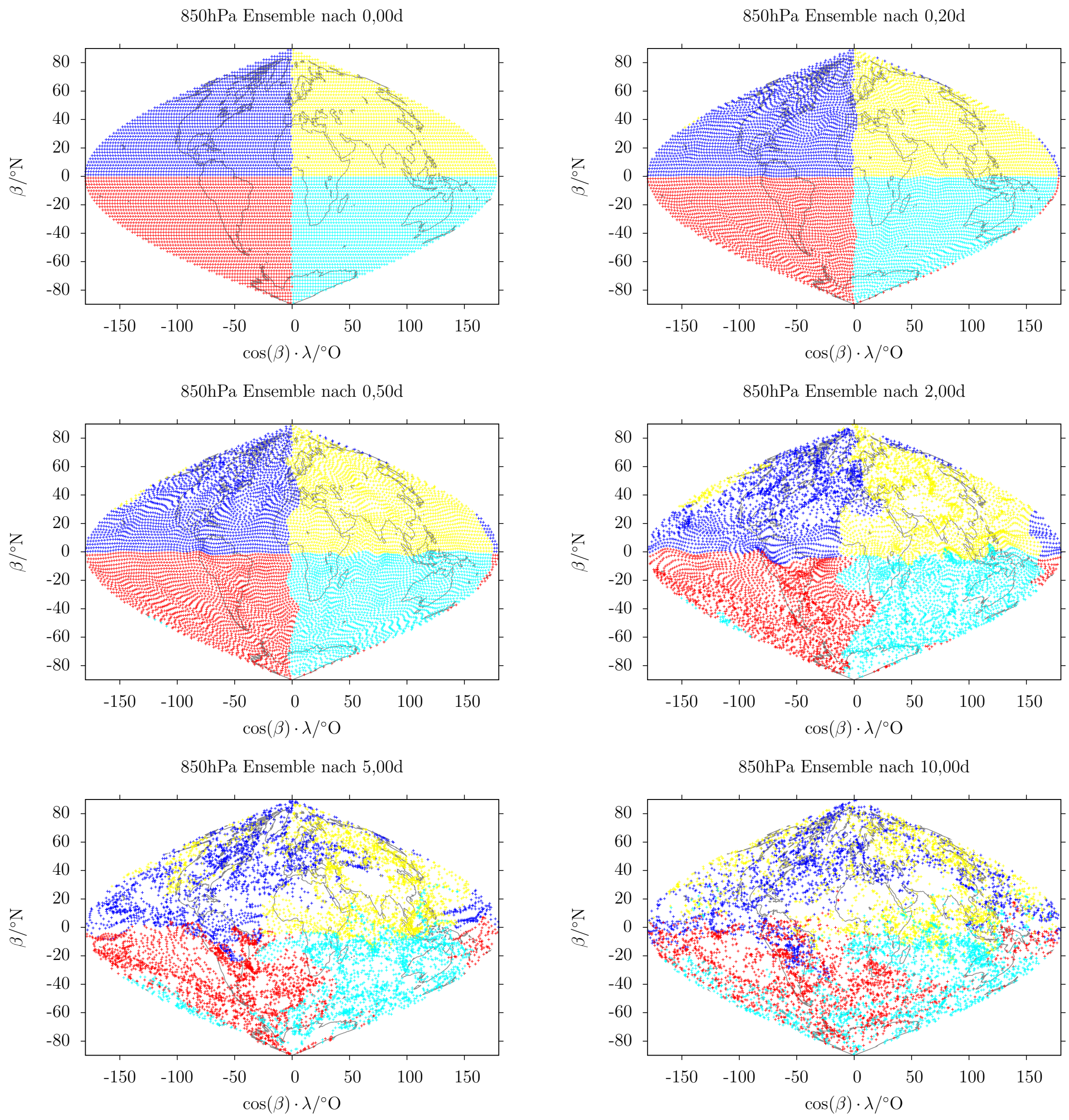

850hPa Ensemble nach 0,00d
850hPa Ensemble nach 0,20d
850hPa Ensemble nach 0,50d
850hPa Ensemble nach 2,00d
850hPa Ensemble nach 5,00d
850hPa Ensemble nach 10,00d
β/°N
cos(β) · λ/°O

Figure 4.4: 10 days of transport of d160 in 850hPa
Each particle of the ensemble (8110 altogether) is depicted by a colored symbol. The color of the symbol encodes its origin in one of the quadrants (northwest, northeast, southwest and southeast). This coloring provides a simple but effective visualization of the mixing dynamics in north-south as well as east-west direction.
Please keep in mind that the 850hPa denote only the *starting* level of the ensemble — the transport is executed in three dimensions. Consequently, this is no mixing of particles *in* 850hPa, but particles *out of* 850hPa.

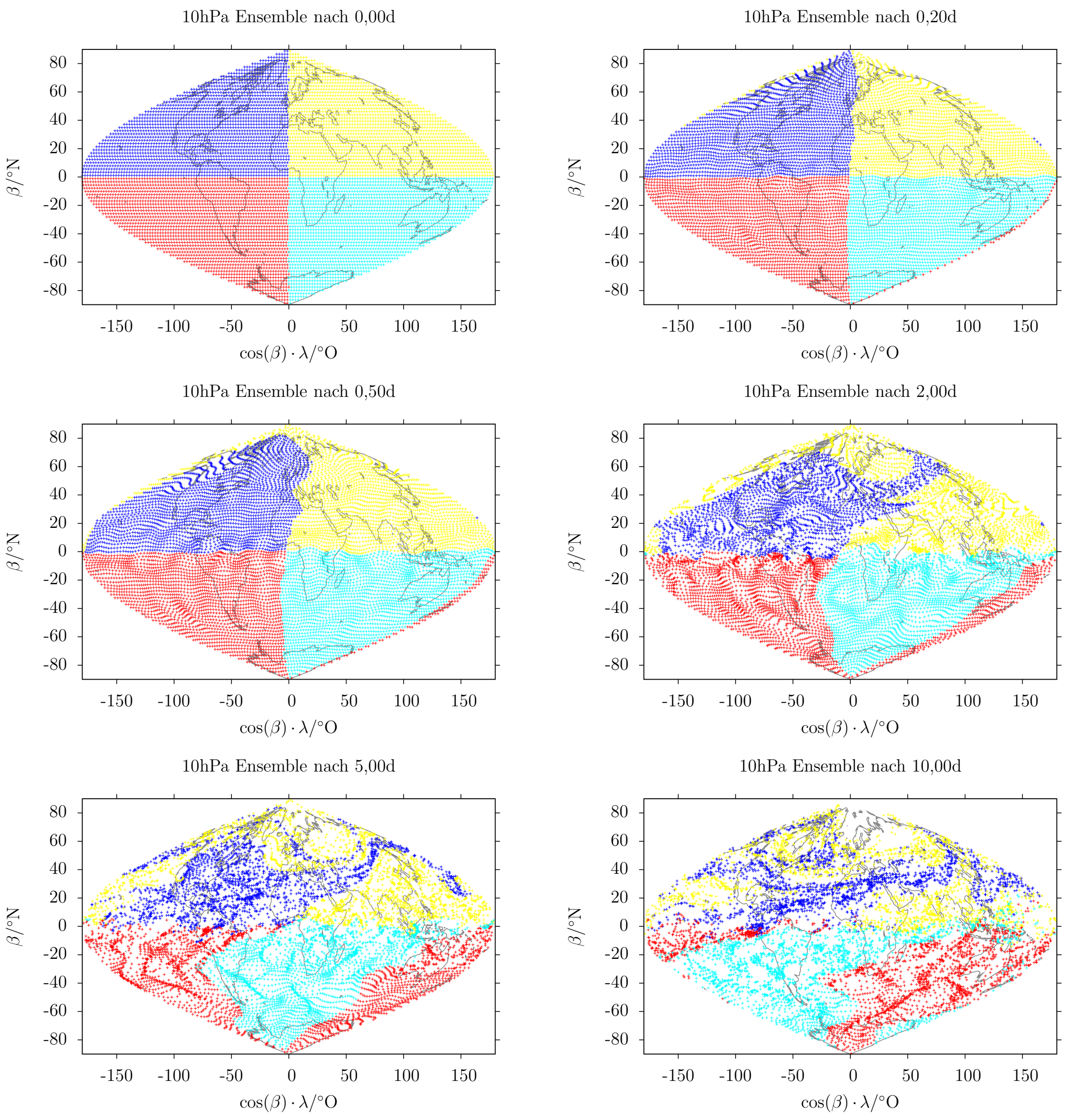

10hPa Ensemble nach 0,00d
10hPa Ensemble nach 0,20d
10hPa Ensemble nach 0,50d
10hPa Ensemble nach 2,00d
10hPa Ensemble nach 5,00d
10hPa Ensemble nach 10,00d
β/°N
cos(β) · λ/°O

Figure 4.5: 10 days of transport of d160 in 10hPa
Each particle of the ensemble (8110 altogether) is depicted by a colored symbol. The color of the symbol encodes its origin in one of the quadrants (northwest, northeast, southwest and southeast). This coloring provides a simple but effective visualization of the mixing dynamics in north-south as well as east-west direction.
Please keep in mind that the 10hPa denote only the *starting* level of the ensemble — the transport is executed in three dimensions. Consequently, this is no mixing of particles *in* 10hPa, but particles *out of* 10hPa. The next figure, Figure 4.6, definitely shows the existence of vertical transport.

The convective up-transport at the equator is the topic of Figure 4.6, where an excerpt of a d400 ensemble in 850hPa between latitudes of $-10°$N and $10°$N is advected. It is plain to see that particles are driven up towards the tropopause.

The insights are preliminary concluded by two horizontally local, but extended over the height from 1000hPa up to 0.01, ensembles, each inside a circle of $5°$N around the north or south pole, respectively, in Figure 4.7 and Figure 4.8. The plots show the in-mixing of polar air into lower latitudes without distinction for source height. The differing situation on the north and south side are discernible. From the south pole, the distribution is formed by a symmetric vortex, while the asymmetry in the north is evident from the beginning.

Those are some qualitative impressions, that mainly testify that the implemented Lagrangian advection scheme works so far. Known transport structures in the atmosphere are reproduced — I am on the right path.

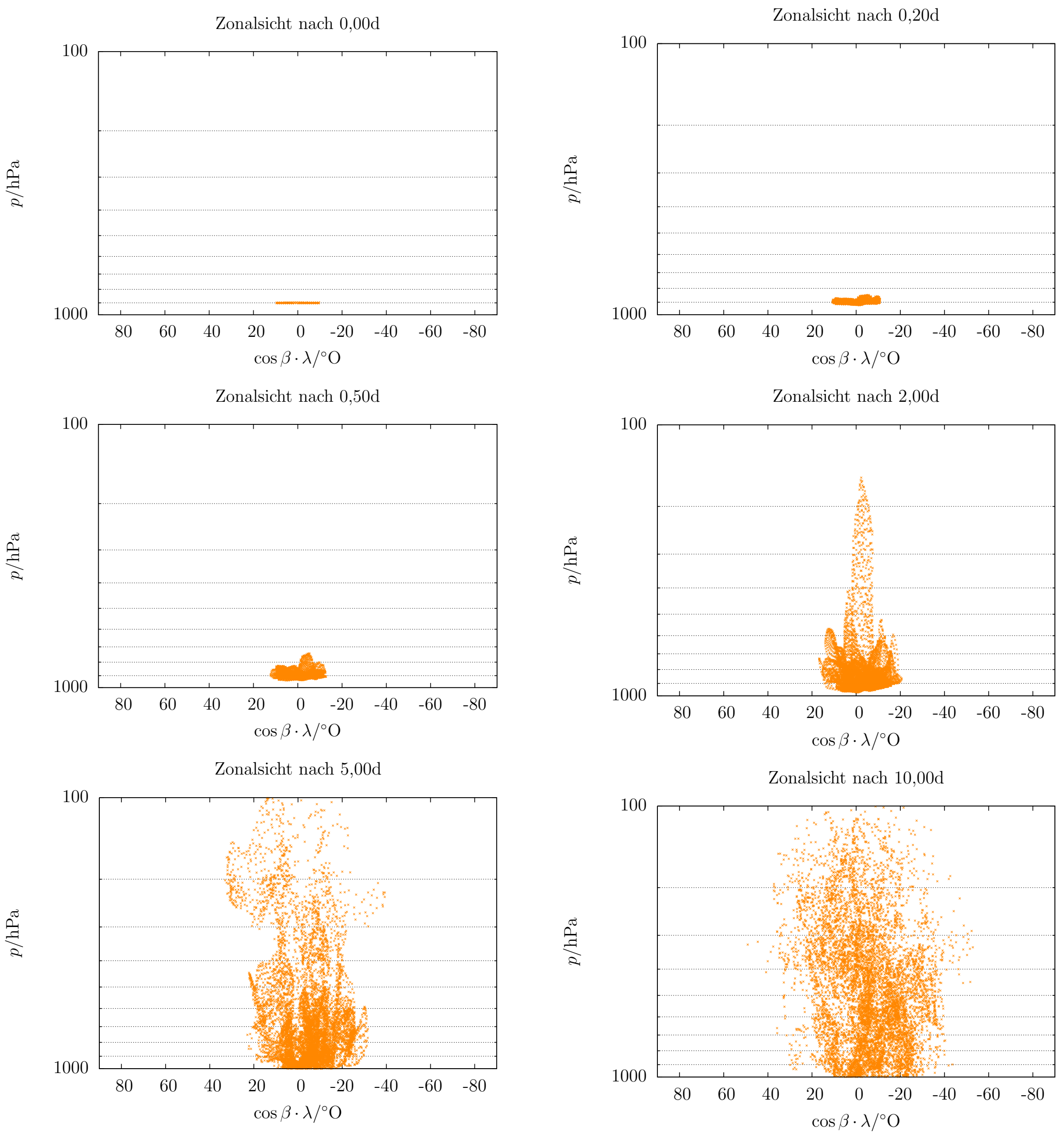


Figure 4.6: Convective up-welling in the tropics
The ensemble of 8744 particles starts at $p = 850\text{hPa}$ and $\beta \in [-10; 10]$ (distributed over all longitudes) and is subject to convective up-welling in the central tropics.

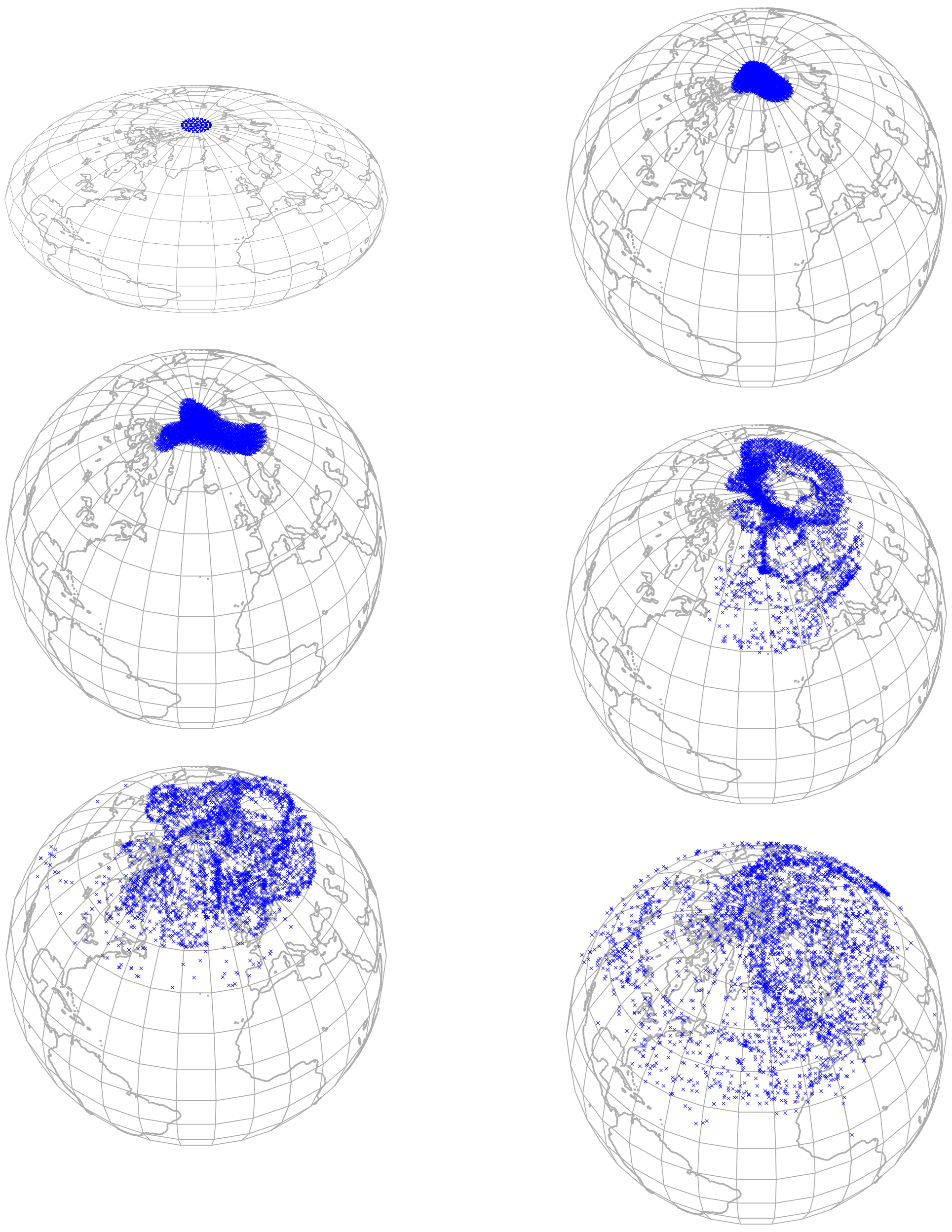


Figure 4.7: Distribution from north pole in 10 days
3216 particles from all layers of the climate model experience distribution out of the close vicinity of the north. The temporal order is the same as before: form left to right, from top to down: beginnig, after 0.2, 0.5, 2, 5 and 10 days.

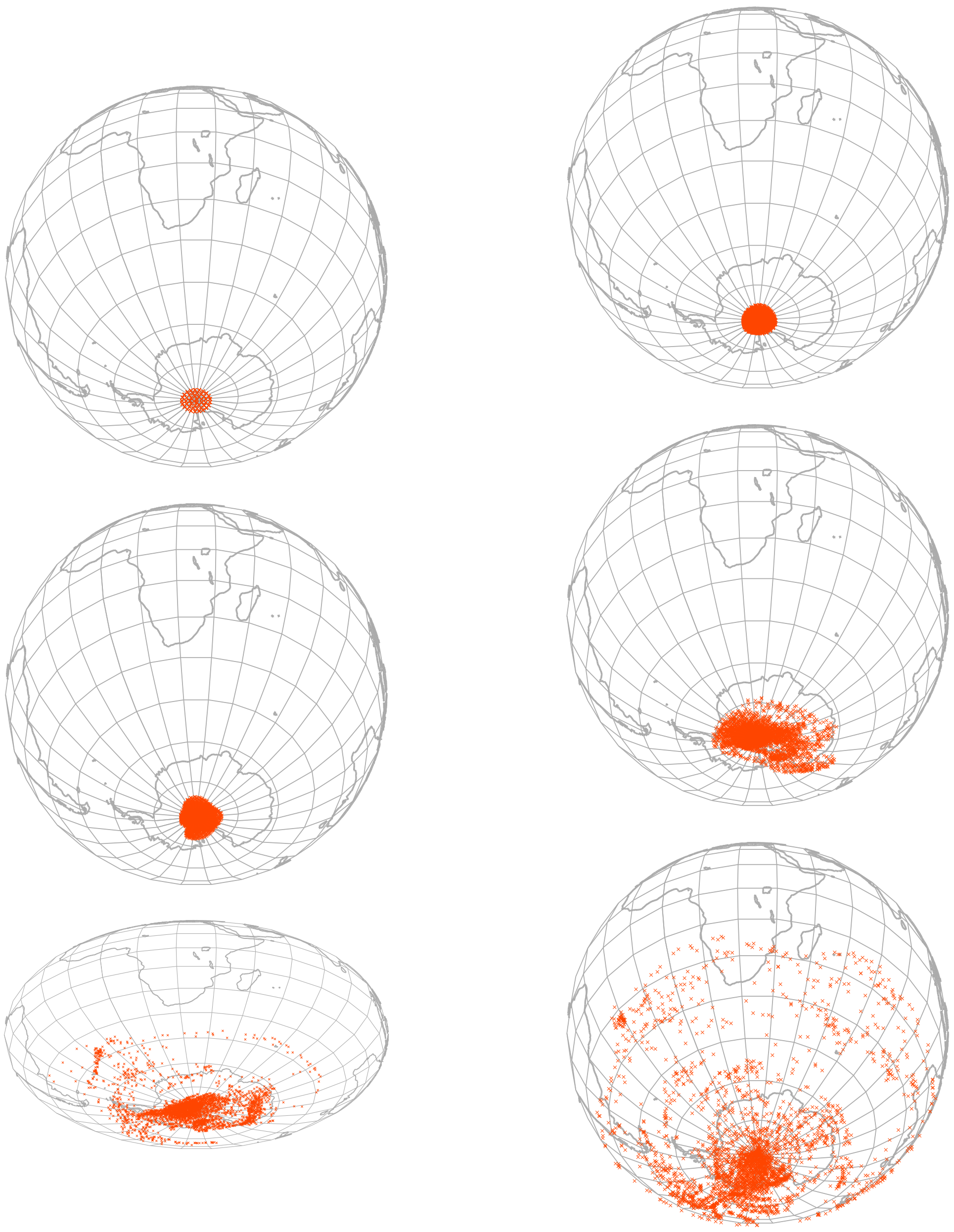


Figure 4.8: Distribution from south pole in 10 days
3216 particles from all layers of the climate model experience distribution out of the close vicinity of the south. The temporal order is the same as before: form left to right, from top to down: beginning, after 0.2, 0.5, 2, 5 and 10 days.

5 Mass consistency with the climate model

5.1 First global runs up to 80 years

The long-term goal of my work is the analysis of long-time transport structures — barriers, mixing ratios, distant connection. A necessary step in this direction are complete global computations with low (achievable) resolution in space and time. Before entering the details, I want to have drawn the global picture. I chose the 80-year time span from 1965 to 2044 out of the 80 years of available data for this long-time picture. The first 15 years (1950 to 1964) I leave as transitive phase[1]. Now, there are in all four global transport runs, consisting of

- 80 years with d210-3 (ca. $1 \cdot 10^6$ particles) with a time step of 0.1d and storage of positions each month[2] and
- 1 year with d1000-3 (ca. $21 \cdot 10^6$ particles), also with a time step of 0.1d, but storage every day,

each once driven by wind fields of the ECHO-GiSP reference run and once by those of the interactively coupled one. These are really quite rough overview runs for a start. Also, I intend to undertake future computations with smaller time step – a tenth of a day is rather large.

The d1000-3 runs became possible through the investment in the RAID server Atlas, which provides the necessary storage space for a year of d1000-3 with daily snapshots. That space amounts to 356GiB. Actually, this number does not look as scary now as it did just a few years ago. A sign of the times is also, that we can handle these amounts of data on relatively cheap ATA hard disks; there is no need for an elaborate and slower tape archive system[3]

The question of sensible visualization is posed after computing those ensemble. A first approach are plots with projected dots (or point-like symbols) like in Figure 4.5, or the zonal view like Figure 4.6. Coloration of points depending on their origin can provide an impression of the mixing. But especially for the important zonal view (vertical and north-south transport), that is very problematic. Two aspects prohibit quantitative statements:

- Particles from different source regions, especially different heights, represent different air mass. A dot is a dot — there is no room for relative weighting. Without some equalization, the upper air masses look deserted very quickly, because of many — light — particles move into the lower levels and few — but heavy — particles come up from there.

[1]The time span from the start of the model that is needed for the “spin-up” towards a usable state.
[2]each 30 days, according to the 360 model days per year
[3]That may be needed for long-time storage, but for the mid-term storage and especially active usage, hard disks are in advantage.

- The aggregation of all points on a latitude leaves the impression that air mass is concentrated from the poles to the equator. Better is a view that provides longitude means according to the density. For the same mass density, you need more particles in the tropics, because they have to fill a bigger volume. In the end, I deem it sensible to depict the density and not absolute mass.

A historic plot (Figure 5.1), which resembles one I presented as first example of the three-dimensional integration a year ago at a working meeting, is subject to both issues: The atmosphere concentrates above the tropic surface.

I see, that for a meaningful plot you need more than just distribution of many colored points on the canvas. Also, I do not just want to make plots, I want to *analyze* the data quantitatively. I compute the large number of trajectories to compute statistical measures from them. The measure that is currently missing, is the weight of one particle, of one sub-ensemble — the represented air mass. After assigning air mass to particles or groups of them, respectively, I can arrive at meaningful numbers for mixing rations. The computation of the represented air mass and the consistency comparison that is made possible through that are the topics, with which I will conclude this Diploma thesis. I will not continue to treat the 80 year long d210-3 run, but instead concentrate on the better resolution of the d1000-3. Both runs contributed to [Orgis2007], though.

5.2 Quantification: Partition & Weight

For computing mixing rations out of the many individual trajectories, I need some definition of source and target zones[4]. I must divide the atmosphere. In the defined parts, I can compute measures from the number of particles that entered there. I can cover questions of distribution of air from a certain source region during a certain time. Or the inverted question: From which source region does which part of the air mass in a target region originate?

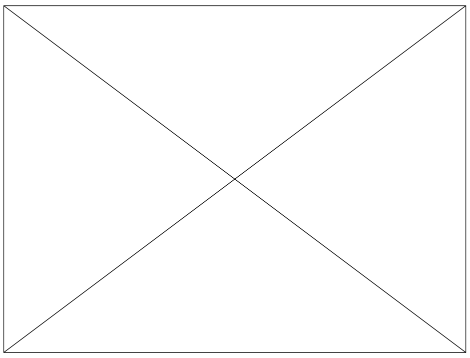
To be able to quantitatively compare the air mass represented by particles at all, I need to attach some weight to the particles. But even an exact density value (not available in practice anyway) at the starting point would not be enough, because it is not practical to distribute starting points vertically according to density. Like mentioned before, and later visualized in Figure 5.7, the exponential density gradient prohibits a useful coverage of the atmosphere in the upper layers, or just means too many particles in the lower layers. A possible approach is the attachment of some weight by dividing the mass in zone among the contained particles.

I invest some time in the question about the partitioning of the atmosphere that should be used. The main interesting aspect is the sphere surface[5] with the inhomogeneous coordinates. In the vertical direction, there is little to say against the logarithmic division into layers, but in the horizontal, the issue is less clear, when you consider some requirements.

The basic requirements for me is to separate the treatment of mass/density and mixing from the systematic tightening of the coordinates towards the poles. When simply using cells of

[4]I use a generic term “zone” here for naming an arbitrary region in the atmosphere in 3D, or 2D on the surface. This is not about fixed regions like subtropics or polar (climate) zone.

[5]Yes, *idealized* as sphere...



The particles have been started on the marked layers; the color of a dot tells the source layer.

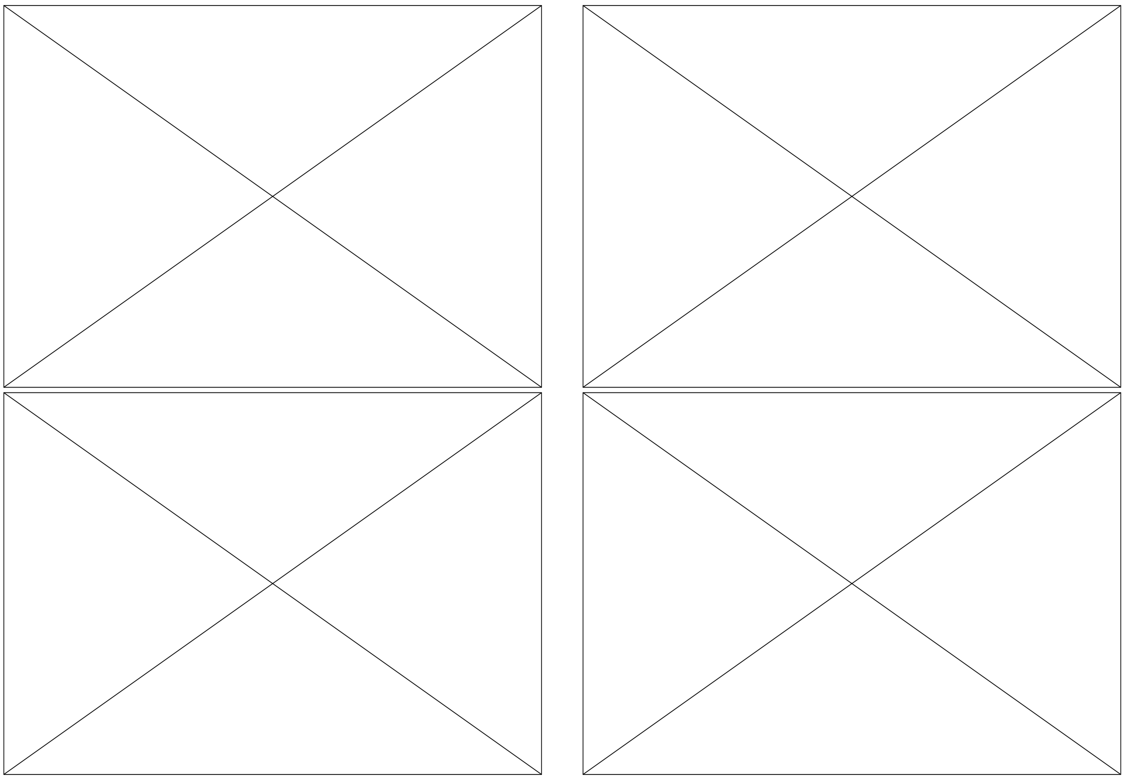


Figure 5.1: Historic view on 180 days of an early 3D transport run
...with indication of stratospheric intrusion (arrow).
Every particle in the ensemble is depicted as a dot, without considering any difference in the represented mass. Furthermore, all particles on one latitude are drawn together in this zonal view. That, naturally, causes the visual concentration towards the equator — together with the concentration on lower air layers because missing weight equalization.

a grid in (λ, β) with fixed grid point distances $\Delta\lambda$ and $\Delta\beta$ (see Figure 5.2), the area of a cell near the poles is much smaller of one near the equator[6] This is problematic, because that is a *systematic* difference in the uncertainty when computing air mass from counted particles in the cell. Air density computed out of the particle ensemble in a cell near the pole relies on a much lower number of particles in the smaller area. It is not sensible to do statistical comparisons (like computation of mass density from particle number) on such a differing basis. The saying advises against comparing apples and oranges — the comparison of densities from differing cells of the simple grid is more like comparing apples to peas!

The difference between apples and peas lies mainly in the size. Oranges, however, are shaped a bit differently and also feature a different taste[7], but they are of similar size and thus a definite improvement, compared to the peas. I can replace the differently shaped and very differently sized zones by still differently shaped but uniformly sized zones using a more flexible partition. Disregarding the shape, I can now calm down and compare apples with oranges. They are not that different after all.

The goal is the partition of the earth surface into zones of same size. I will now present two approaches for that, the second of these being the optimum that I will use later on. After establishing the partition of the atmosphere in zones from the surface partition and logarithmic partition of the height (pressure), I will come back to the question of assigning air mass to these zones.

5.2.1 Embeddable Equipartition

An obvious possibility of areal equipartition of the surface is offered by the variation of the latitude division. Instead of the fixed interval $\Delta\beta$ we use a variable latitude spacing — towards the poles the latitude range covered by a zones is enlarged to compensate for cell area.

On the sphere, the area of a zone with $\lambda \in [\lambda_0; \lambda_1]$ and $\beta \in [\beta_0; \beta_1]$ is given by

$$\begin{aligned} A_{\text{real}} &= \int\limits_{\beta_0}^{\beta_1} (\lambda_1 - \lambda_0) R \cos\beta \cdot R \mathrm{d}\beta \\ &= (\lambda_1 - \lambda_0) R^2 \sin\beta \big|_{\beta_0}^{\beta_1} \end{aligned} \tag{5.1}$$

R is a global constant and $(\lambda_1 - \lambda_0)$ is the same for all zones for the fixed interval under consideration. Because of that, I can normalize to the area of a hemisphere[8] (or a segment of it between λ_0 and λ_1) and simply consider

$$A(\beta_0, \beta_1) = \sin\beta_1 - \sin\beta_0 \tag{5.2}$$

The area follows the sine function between β_0 and β_1.

[6]The area between two latitude circles scales down according to the down-scaling of the latitude circumference with $\cos\beta$. At $\Delta\beta = 10°\text{N}$, the area of a cell with $\beta \in [0°\text{N}; 10°\text{N}]$ has $11{,}4$ times the size as a cell in $\beta \in [80°\text{N}; 90°\text{N}]$.

[7]☺ Better? I will not decide that here...

[8]Normierte Gesamtoberfläche der Kugel ist gleich 2.

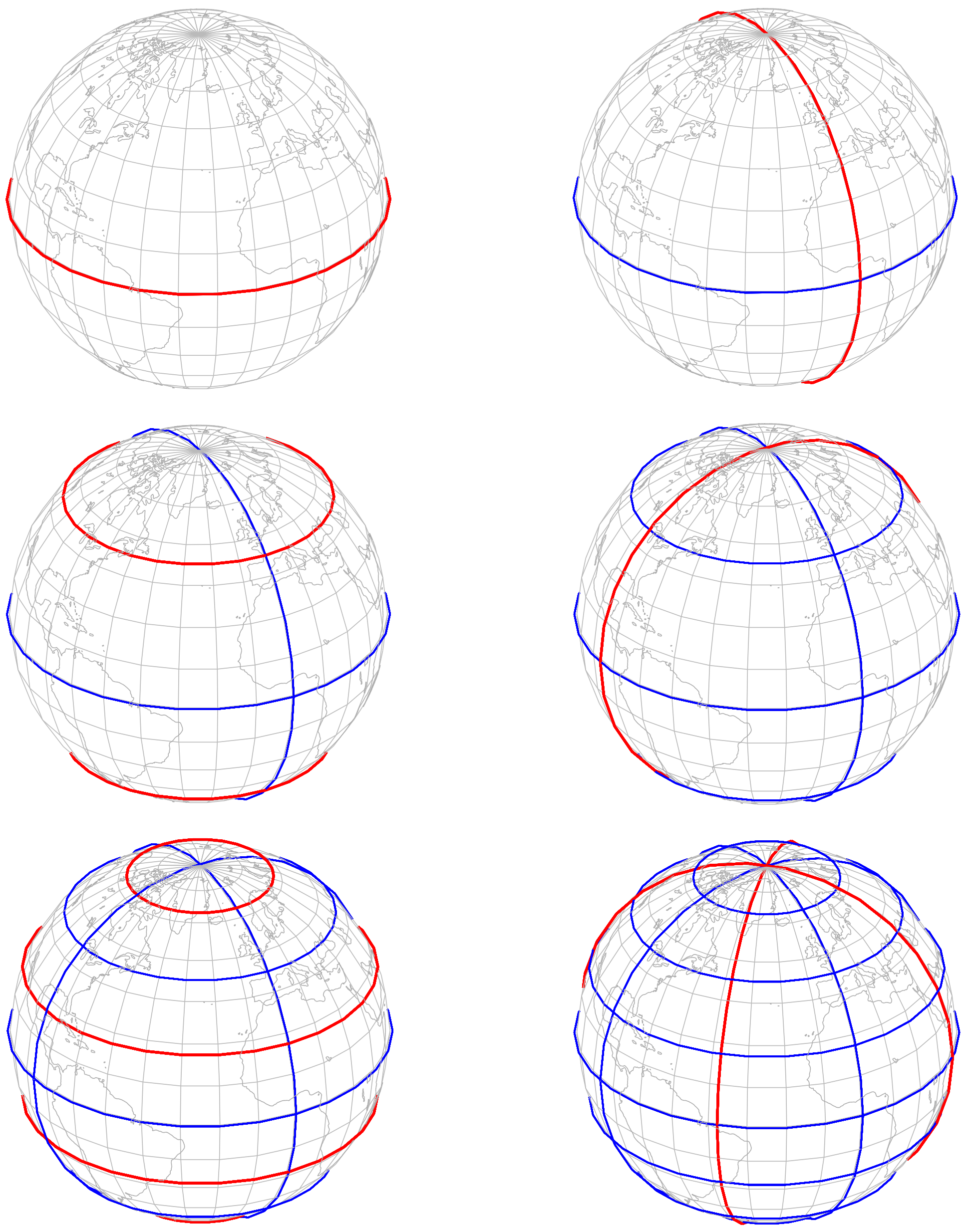


Figure 5.2: Division of earth surface into unequal regions with fixed grid ...doubling division with 2, 4, 8, 16, 32 and 64 regions. Zones near the poles are distinctively smaller than those near the equator (beginning at third picture).

Following that insight, the partition into latitude regions of same area is achieved by a fixed increment in the sine of latitude. When I intend to divide the latitudes between B_0 and B_1 ($B_0 < B_1$) in a ranges, the borders β_i; $i \in [0; a]$ are given by

$$\beta_i \quad = \quad \operatorname{asin}\left(\sin B_0 + \frac{i}{a}(\sin B_1 - \sin B_0)\right) \tag{5.3}$$

Especially for the whole sphere with $B_0 = -\frac{\pi}{2}$ and $B_1 = \frac{\pi}{2}$, this is

$$\beta_i \quad = \quad \operatorname{asin}\left(-1 + 2\frac{i}{a}\right) \tag{5.4}$$

The longitude partition of the range from Λ_0 to Λ_1 in o parts is given directly by a fixed interval

$$\lambda_i \quad = \quad \Lambda_0 + \frac{i}{o}(\Lambda_1 - \Lambda_0) \tag{5.5}$$

Again for the whole sphere from 0 to 2π:

$$\lambda_i \quad = \quad 2\frac{i}{o}\pi \tag{5.6}$$

This grid defines $a \cdot o$ zones on the sphere surface, each featuring the same area.

$$A_{\text{real}} = \frac{1}{o}(2\pi - 0) \cdot R^2 \cdot \frac{1}{a}\left(\sin\frac{\pi}{2} - \sin\left(-\frac{\pi}{2}\right)\right) = \frac{4\pi}{oa}R^2$$

One aspect inherited from the simple grid with constant $\Delta\beta$ is the embeddability of higher resolved partitionings. when o or a are increased to a multiple, all formerly defined borders β_i and λ_i are still in use and there are just additional borders between these. This property is used in Figure 5.3 to build the partition with $o = a = 8$ iteratively.

The property of same area per cell is given, but the adaption using only the latitude ranges has a catch: The zones get thicker in latitude towards the poles. In the important zonal view of the whole atmosphere ($p - \beta$ plot, mean over λ). this means a loss in resolution, and subsequently information, towards the poles. The poles are important — also in regard of the fact that my work stands in the frame of the Pole - Equator - Pole project!

5.2.2 Balanced Equipartition (Zo-a)

The adaption of the zone area via the latitude thickness is not satisfying. Inclusion of the longitude partition is the next step. I will give up the simple structure of homogeneous longitude division, and thus the global grid. A variable number of longitude divisions (as single zones) for the individual latitude range joins the the variation of latitude borders. Less zones are put into the latitude ranges towards the poles, keeping the latitude extend low while ensuring the uniform zone area. That maintains a higher resolution near the poles, and overall more homogeneous at that, in the zonal view.

I name this partition of surface Zo-a, corresponding the number of latitude ranges a and the maximum number (in the latitude range next to the equator) o of longitude ranges.

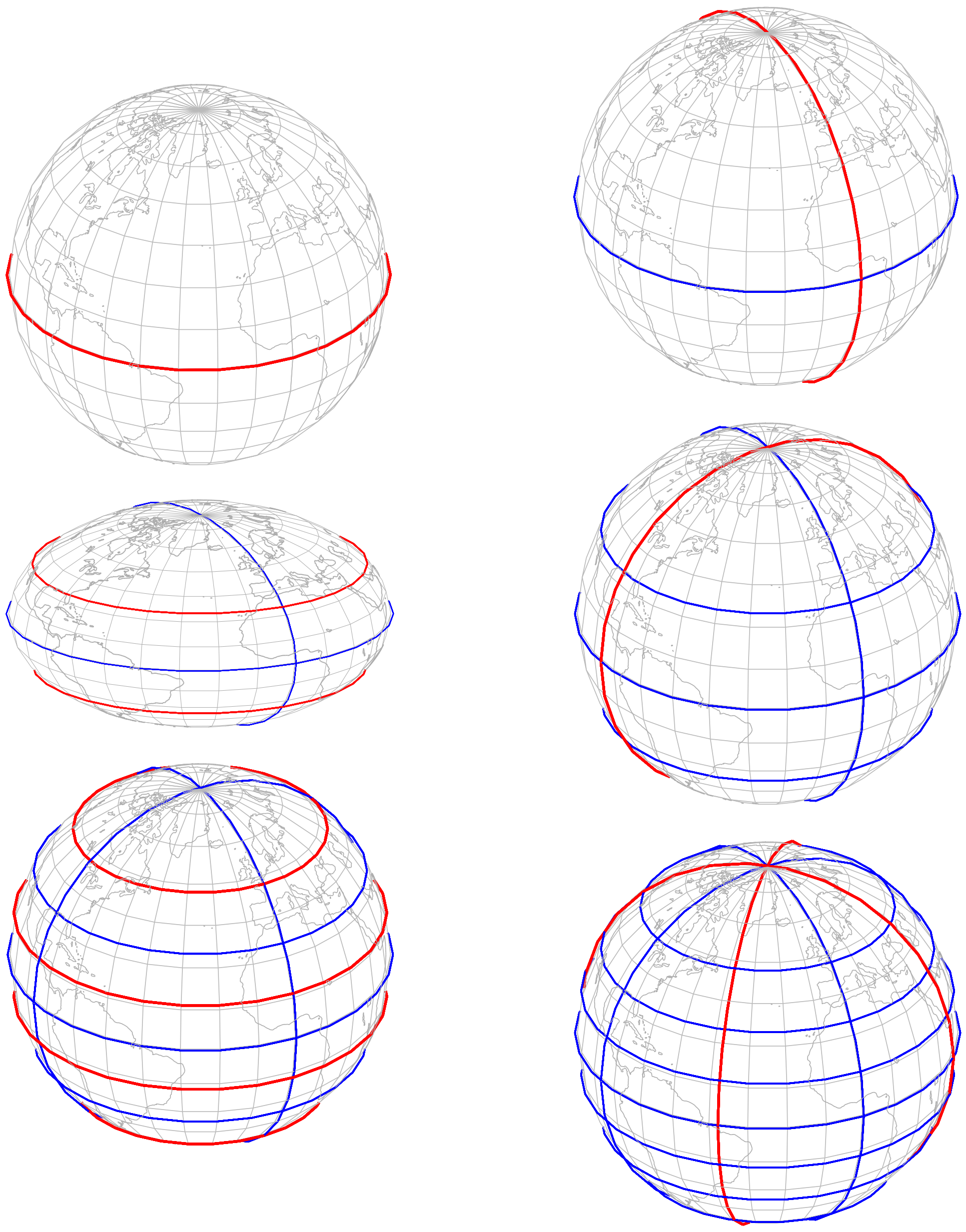


Figure 5.3: The first partitioning of the earth surface in zones of equal area ...with zone count doubling to 2, 4, 8, 16, 32 and 64 zones. The Equipartition is achieved by sacrificing resolution near the poles.

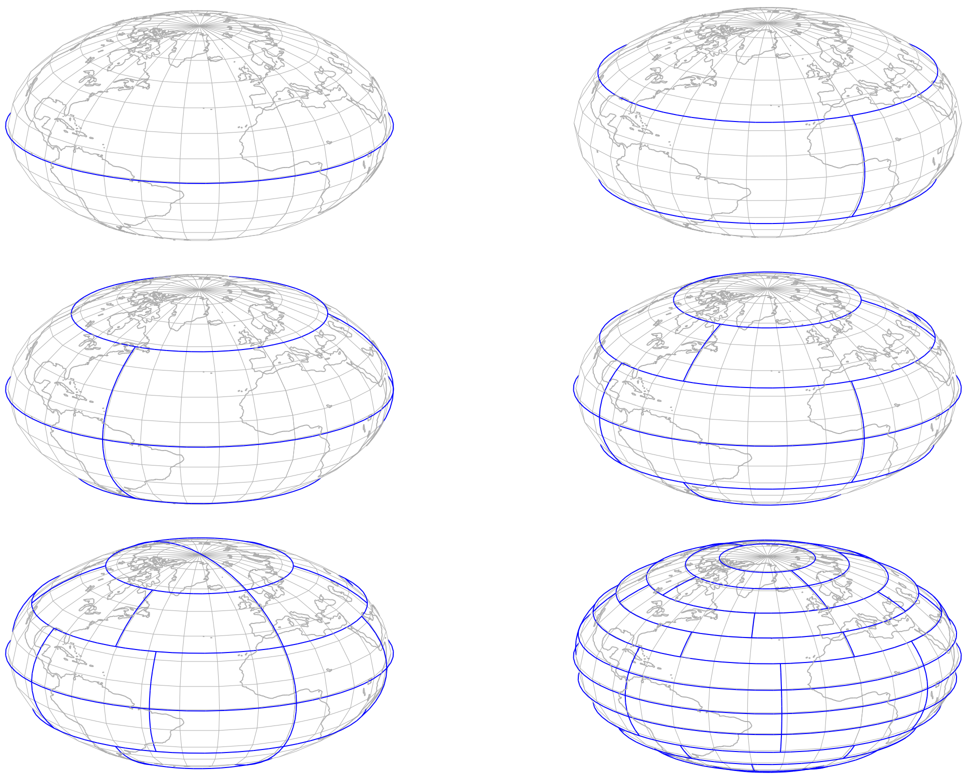


Figure 5.4: Balanced Equipartition
...to 2, 4, 8, 16, 32 and 64 zones. The chosen parameters (left to right, top to down) are (o, a): $(1, 2)$, $(2, 3)$, $(3, 4)$, $(4, 6)$, $(8, 6)$, $(7, 14)$. This partitioning combines equal area with approximately uniform latitude partition, at the loss of the simple partition into global longitude intervals.

A remaining variable is the possible limitation of the overall longitude and latitude region (Borders $B_{0/1}$ and $\Lambda_{0/1}$). When these border are not mentioned, then the partition of the whole surface is meant ($B_{0/1} = \pm\frac{\pi}{2}$ and $\Lambda_0 = 0 \wedge \Lambda_1 = 2\pi$). In the (implicitly) given overall region the parameters o and a define a partition unambiguously. In detail, the partition consists of the latitude borders $\beta_i \; \forall i \in [0; a]$ and the longitude borders $\lambda_{i,j} \; \forall i \in [0; a) \wedge j \in [1; z_i]$ for each latitude range.

The algorithm for the partition Zo-a

1. There exists the prescribed range: $\beta_0 = B_0 < \beta_a = B_1$ and $\Lambda_0 <= \lambda_{i,k} <= \Lambda_1$
2. Starting point is the ideally uniform partition of latitudes.
$$\Delta\beta = \frac{\beta_a - \beta_0}{a}; \quad \forall i \in [1;a): \ \beta_i := \beta_0 + i \cdot \Delta\beta$$
3. The relative[9] gauge area A^* for a single zone is determined by the partition of the largest (in terms of surface area) latitude range out of this pattern.
$$A^* := \frac{1}{o} \max\{(\sin\beta_{i+1} - \sin\beta_i) : i \in [0;a)\}$$
Note: The global restriction of longitudes is irrelevant here, is is about relative weighting of latitude ranges.
4. Initialization of the total zone count: $Z := 0$
5. The number z_i of zones in the latitude range i is determined by A^*, the total count Z being incremented iteratively:
$$\begin{array}{lrcl} \forall i \in [0;a): & L_i & := & Z \\ & z_i & := & \mathrm{rund}\left(\frac{1}{A^*}(\sin\beta_{i+1} - \sin\beta_i)\right) \\ & z_i = 0 \ ? \ z_i & := & 1 \\ & Z & := & Z + z_i \end{array}$$
6. The actual relative single-zone area is computed from Z:
$$A^\dagger = \frac{\sin\beta_a - \sin\beta_0}{Z}$$
7. Using that value and the z_i (L_i are summed z_i), the latitude borders are finally fixed to new values[10].
$$\forall i \in [1;a): \beta_i := \mathrm{asin}(\sin\beta_0 + L_i \cdot A^\dagger)$$
8. At last, the longitude partition for each, now fixed, latitude range is executed. The outer borders result from the computation in theory and one could spare the two individual assignments. Instead, the possible error of four unnecessary computing operations is intercepted by defining the edge of the world exactly.
$$\begin{array}{lrcl} \forall i \in [0;a): & \Delta\lambda & := & \frac{\Lambda_1 - \Lambda_0}{z_i} \\ & \lambda_{i,0} & := & \Lambda_0 \\ & \lambda_{i,z_i} & := & \Lambda_1 \\ & \forall j \in [1;z_i): & & \lambda_{i,j} := \Lambda_0 + j \cdot \Delta\lambda \end{array}$$

This routine is followed closely in the program (fragment in Listing 5.1) – with the exception that the program first stores the sine values in β and replaces those with angles later.

[9] relative to the restricted longitude range; $A^*/A^*_{\mathrm{real}} = 2\pi/(\Lambda_1 - \Lambda_0)$

[10] The deviation from the ideal latitude partition is minimized by using large enough o

Listing 5.1: Zo-*a* partition

```
pep_t step = (B1-B0)/a; // D beta
#define latsin latb
#define latarea(i) (latsin[i+1]-latsin[i])
latsin[0] = sin(B0); latsin[a] = sin(B1);
for(size_t i = 1; i < a; ++i) latsin[i] = sin(B0 + i*step);
pep_t onezone = 0;
for(size_t i = 0; i < a; ++i) // find biggest ideal region
{
  pep_t area = latarea(i);
  if(area > onezone) onezone = area;
}
onezone /= o;       // A^*
zones_per_lev = 0; // Z
for(size_t i = 0; i < a; ++i)
{
  latindex[i] = zones_per_lev; // index of first zone of band
  // z_i: how many zones of about standard size to put in this latitude band
  zones_per_lat[i] = (size_t) rint(latarea(i)/onezone);
  if(zones_per_lat[i] == 0) zones_per_lat[i] = 1; // > 0 !
  zones_per_lev += zones_per_lat[i];
}
// Now it's the real deal - A^dagger
onezone = (latsin[a]-latsin[0])/zones_per_lev;
// place the latitudes area-even
// actually switching to latitudes here
for(size_t i = 1; i < a; ++i)
latb[i] = asin(latsin[0] + latindex[i]*onezone);
#undef latarea
#undef latsin
latb[0] = B0; latb[a] = B1;
for(size_t i  = 0; i < a; ++i) // longitudes at last
{
  lonb[i] = new pep_t[zones_per_lat[i]+1];
  pep_t step = (L1-L0)/zones_per_lat[i];
  lonb[i][0] = L0; lonb[i][zones_per_lat[i]] = L1;
  for(size_t j = 1; j < zones_per_lat[i]; ++j)
  lonb[i][j] = L0 + j*step; // lambda_i,j
}
```

5.2.3 Global partition in 3D (Z*o*-*a*-*e*)

For the global view I combine a partition Z*o*-*a* with the logarithmic partition of the whole vertical range in *e* layers. This combined, whole, partition into balanced zones is referred to as Z*o*-*a*-*e*. An alternative to the logarithmic spacing of the layers, given by the computer program, is usage of the predefined list of pressure borders, e.g. to resolve the tropopause[11] more accurately.

Examples of the global partition are shown in Figure 5.5.

5.2.4 Weight from standard atmosphere

My first attempt to make particle counts comparable was the application of the US standard atmosphere 1976[12], shown in Table 5.1, based on the pressure levels.

Using that density table, I can assign a pseudo mass to a pressure interval via interpolation and integration of density. “Pseudo”, because I compute not kg out of the kg/m^3, but a scale value with the unit Pakg/m^3 – without using the spatial expends in m. Thus, the more abstract term “weight” seems more appropriate to me. The computation of the weight $g(p_a, p_b)$ of an air layer executes the following scheme:

- Given: $p_a > p_b$ (begin and end of the layer, pressure falling to the top).
- Extract a sub-table out of the atmosphere table (also using inter/extrapolation), (p_i, ρ_i) with n entries, so that $p_1 = p_a$ and $p_n = p_b$.
- Weight is computed via mean density in the sub-intervals.

$$g(p_a, p_b) = \sum_{i=2}^{n} \frac{1}{2}(\rho_{i-1} + \rho_i) \cdot (p_{i-1} - p_i) \tag{5.7}$$

With this rough weighting I can demonstrate now, why the distribution of particles of same weight (representing the same air mass) in the whole atmosphere is problematic. In Figure 5.7 you see how well the stratosphere is covered by two hundred layers in the whole atmosphere — basically not at all. One would need a *very* high number of layers and thus an *extremely* high number of particles.otwendig.

The association of a weight with an air layer (pressure range) is a simple possibility to give weight to the zones of the three-dimensional partition (based on Z*o*-*a*). All zones are equal for their surface area by design[13]. That way, the relative weight of zones form different air layers is defined solely by the upper and lower borders of their layers in the pressure coordinate. Especially, zones in the same layer have the same weight, which eases investigations of horizontal mass exchange.

But, like showed later on in the analysis of global computations, this weighting is not useful for transport analysis. Actually, it is not the main point that the table has not enough data points to describe the model atmosphere. The main point is that you cannot ignore

[11]the border/transition between troposphere and stratosphere

[12]The specific table choice is not important. It will be evident that the approach is problematic enough to void the question about the choice of a norm atmosphere table.

[13]Remember: The thickness of the atmosphere is neglected against the earth radius.

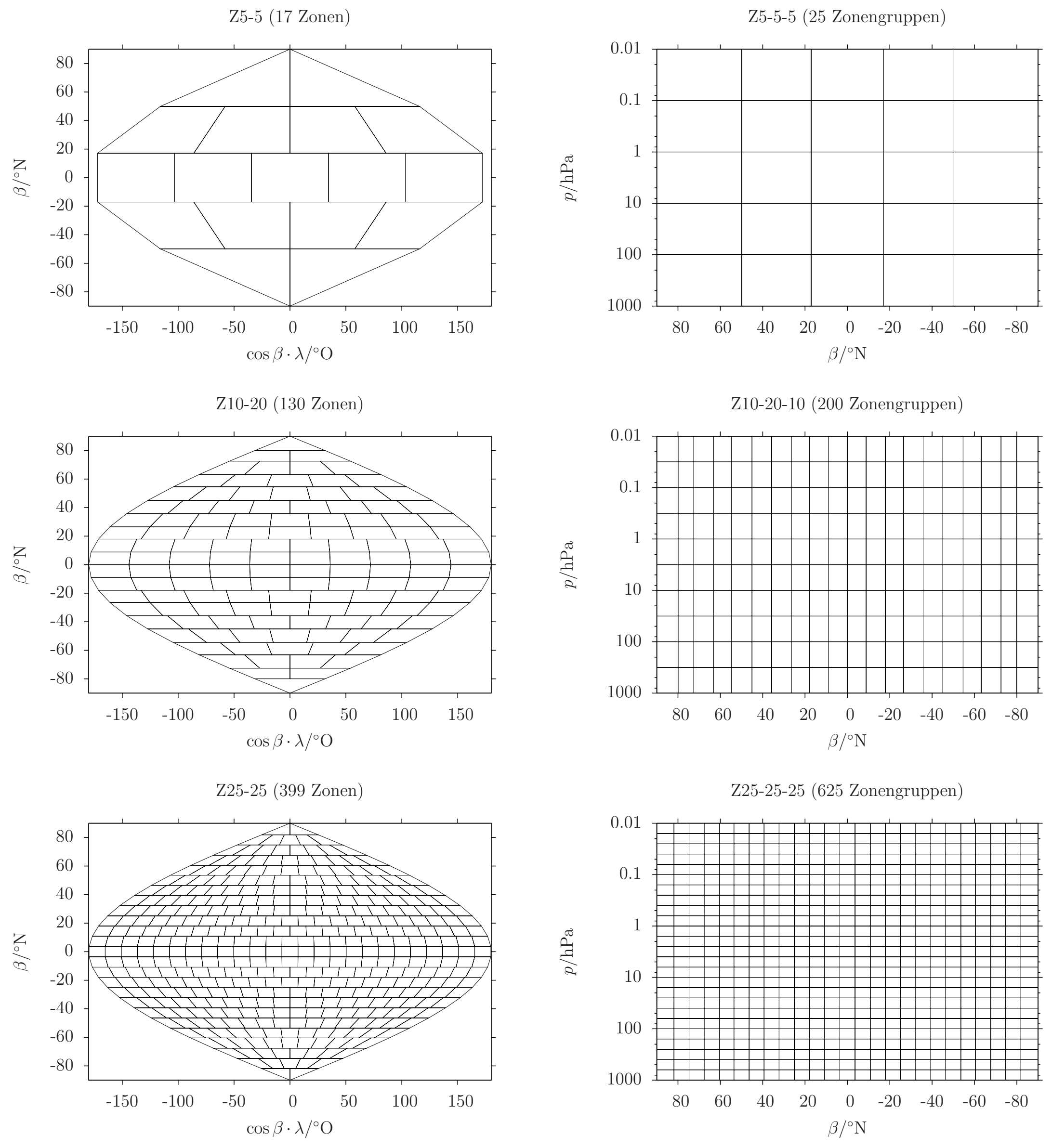


Figure 5.5: Three examples for Z*o*-*a* and Z*o*-*a*-*e*

geopotential Height h/m	geometric height z/m	temperature $T/°\mathrm{C}$	air pressure p/Pa	density $\rho/\frac{\mathrm{kg}}{\mathrm{m}^3}$
0	0	15	$1,0132 \cdot 10^{+05}$	$1,2250 \cdot 10^{+00}$
11000	11019	$-56,5$	$2,2632 \cdot 10^{+04}$	$3,6393 \cdot 10^{-01}$
20000	20063	$-56,5$	$5,4749 \cdot 10^{+03}$	$8,8037 \cdot 10^{-02}$
32000	32162	$-44,5$	$8,6802 \cdot 10^{+02}$	$1,3225 \cdot 10^{-02}$
47000	47350	$-\ \ 2,5$	$1,1091 \cdot 10^{+02}$	$1,4276 \cdot 10^{-03}$
51000	51413	$-\ \ 2,5$	$6,6939 \cdot 10^{+01}$	$8,6163 \cdot 10^{-04}$
71000	71802	$-58,5$	$3,9564 \cdot 10^{+00}$	$6,4212 \cdot 10^{-05}$
84852	86000	$-86,2$	$3,7340 \cdot 10^{-01}$	$6,9582 \cdot 10^{-06}$

Table 5.1: Standard atmosphere 1976 with density
Derived from [WikiNorm].

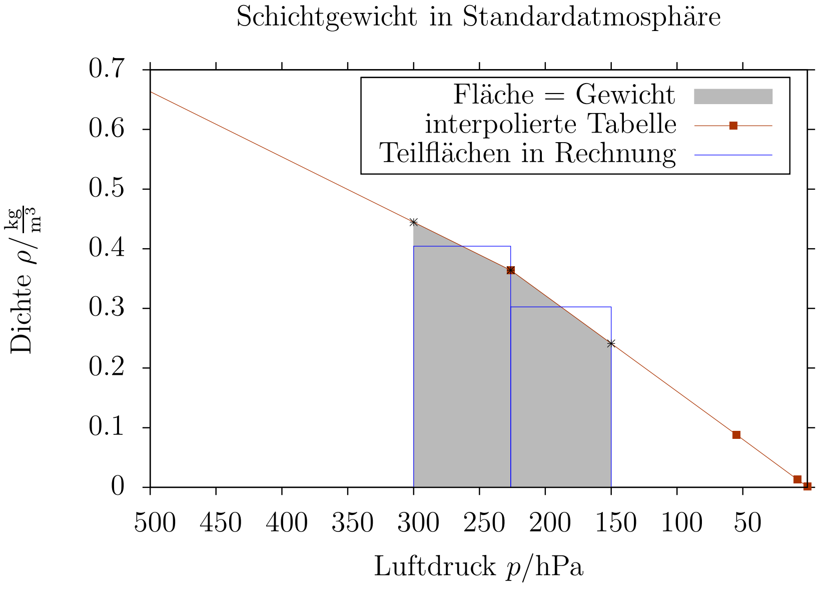


Figure 5.6: Computation of the weight of an air layer
from 300hPa to 150hPa in the density table from the standard atmosphere. It is the simples integration using mean values in the sub-intervals defined by the table points.

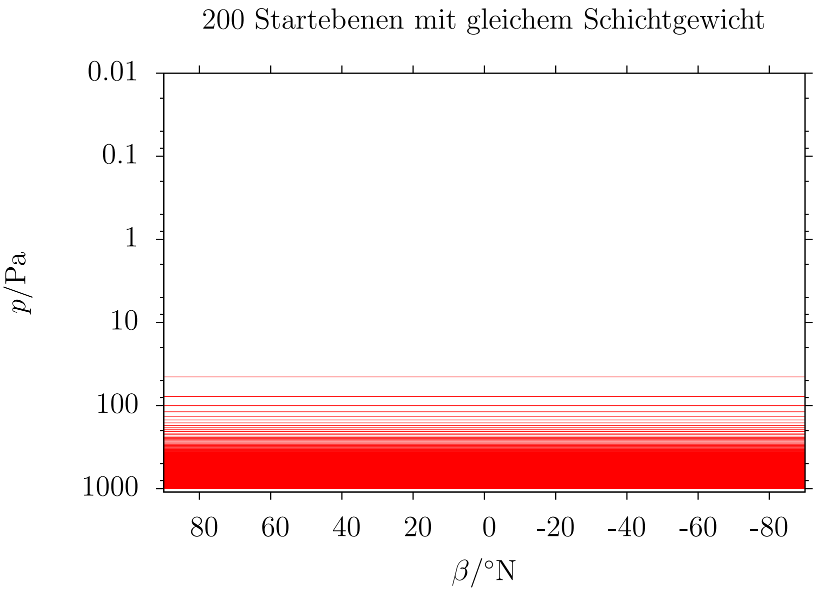


Figure 5.7: 200 layers of same weight according to standard atmosphere
Unsurprisingly, the air mass concentrates at the bottom. That is realistic, but with start-points distributed along that pattern, there is rarely any information about stratospheric dynamics.

the dynamics of air mass distribution in the analysis of transport computations. The mass density in the model shows spatial[14] and temporal variability.

5.2.5 Dynamic density from the climate model

I do not need a table like Table 5.1 to attach mass to the transported particles. The wind fields are extracted from a whole climate model after all! The air density is not one of the variables given by the model runs directly, but it can be derived. Fields of temperature $T(\lambda, \beta, p)$ and relative humidity $s(\lambda, \beta, p)$ are provided. Together with the gas constant $R = 287,1\mathrm{J/kgK}$ for dry air I can compute the density ρ:

$$\begin{aligned} p &= \rho R T(1 + 0.61s) \\ \Leftrightarrow \rho &= \frac{p}{RT(1 + 0.61s)} \end{aligned} \tag{5.8}$$

From this formula I get the field of density $\rho(\lambda, \beta, o)$ for every point in time. In Figure 5.8 one can see, that the logarithmic profile is still the main pattern as expected, but the field has horizontal and well as temporal variation. It is not static at all. The same holds true for the relation of the pressure coordinate to the height in meters, which I need for the application of the density to zones from the partition, see Figure 5.9. The vertical extend of a zone in m together with the base area in m^2 enables the computation from density in $\mathrm{kg/m}^3$ to a mass — no pseudo mass, but a definite figure in kg.

[14]also longitudinal and latitudinal, not just height

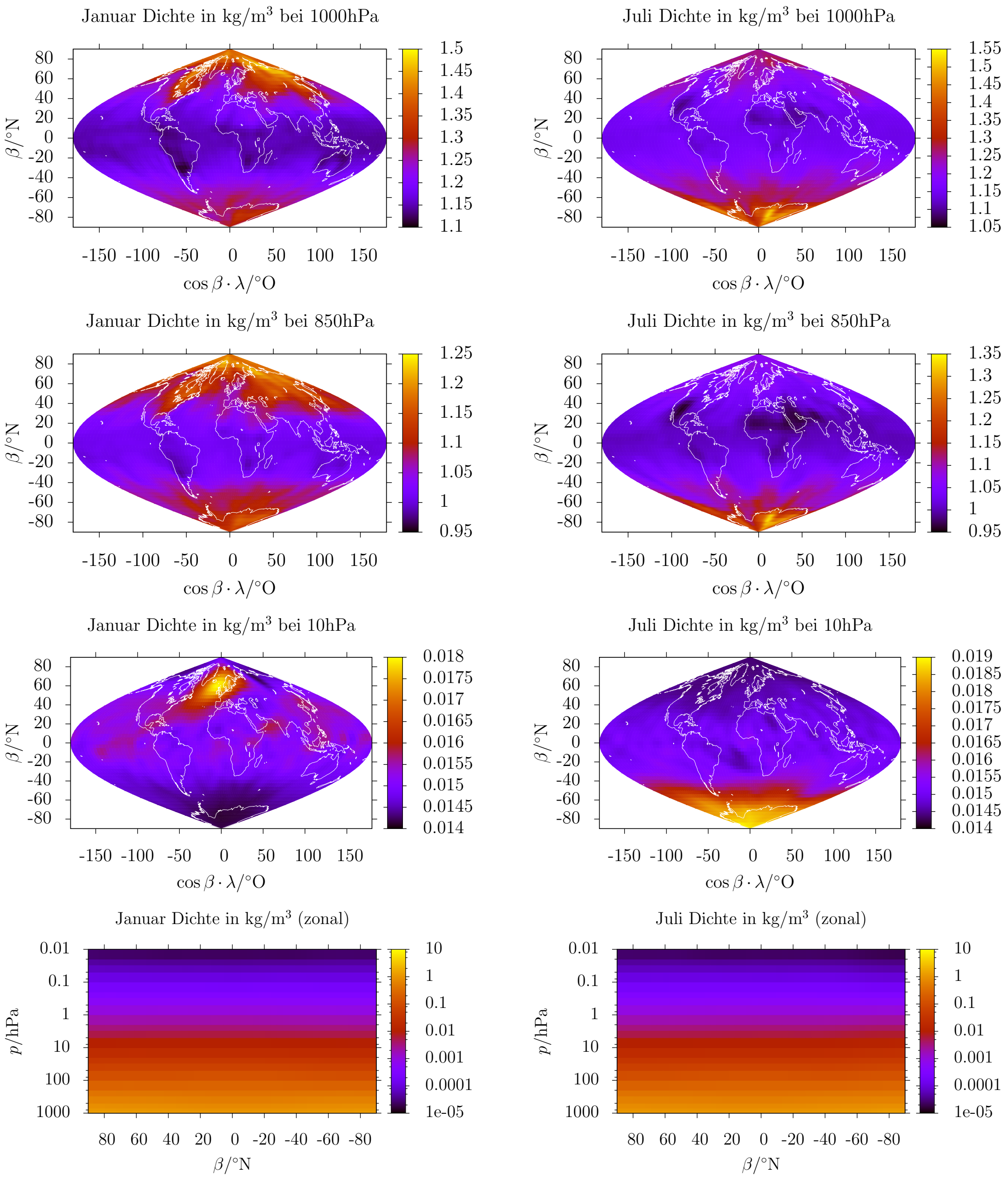

Januar Dichte in kg/m³ bei 1000hPa
Juli Dichte in kg/m³ bei 1000hPa
Januar Dichte in kg/m³ bei 850hPa
Juli Dichte in kg/m³ bei 850hPa
Januar Dichte in kg/m³ bei 10hPa
Juli Dichte in kg/m³ bei 10hPa
Januar Dichte in kg/m³ (zonal)
Juli Dichte in kg/m³ (zonal)
β/°N
cos β · λ/°O
p/hPa

Figure 5.8: Air density from ECHO-GiSP in January and July 1965
Despite the density profile looking flat in the zonal view and only the gradient in height shows, the horizontal and temporal variability is existent.

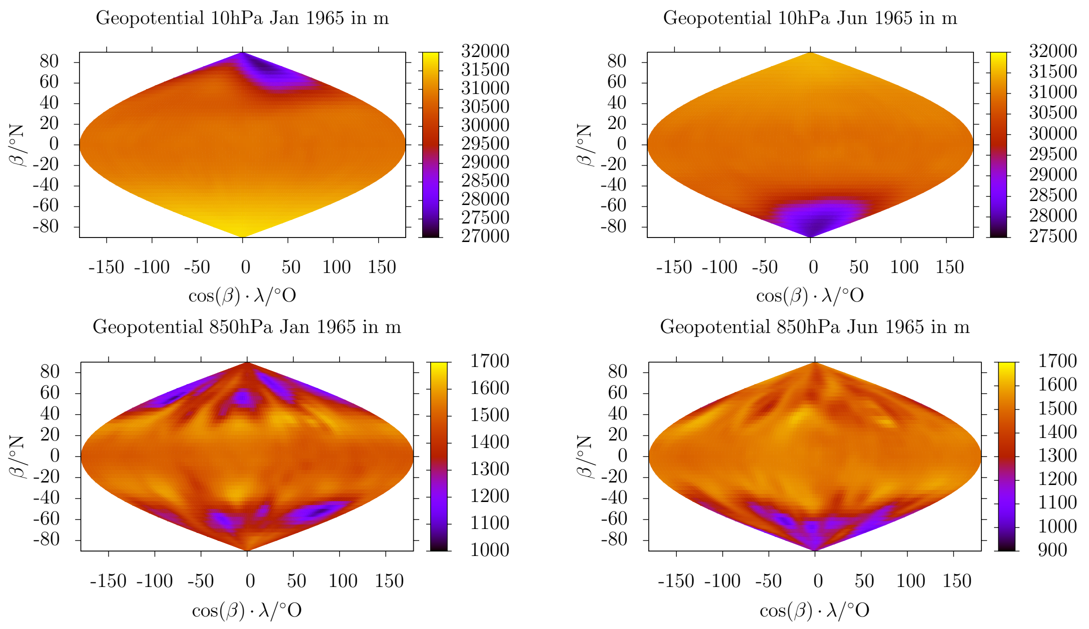

Geopotential 10hPa Jan 1965 in m
Geopotential 10hPa Jun 1965 in m
Geopotential 850hPa Jan 1965 in m
Geopotential 850hPa Jun 1965 in m
β/°N
$\cos(\beta)\cdot\lambda$/°O
32000
31500
31000
30500
30000
29500
29000
28500
28000
27500
27000
1700
1600
1500
1400
1300
1200
1100
1000
900

Figure 5.9: Geopotential height of 850hPa and 10hPa in January and July 1965
The height of pressure levels is neither spatially nor temporally constant.

I compute the height $h(\lambda, \beta, p)$ of a point with pressure coordinate using the field of geopotential[15] $h_p(\lambda, \beta, p)$ that is also provided by the model.

$$\begin{aligned} g(h)\mathrm{d}h &= g_0 \mathrm{d}h_p \qquad (5.9) \\ \Rightarrow \Delta E_{pot} = \int_0^h mg(h)\mathrm{d}h &= mg_0 \cdot h \end{aligned}$$

A direct conversion between geopotential and "real" meters is given by the definition of $g(h)$, respectively $g_0 = g(0)$, through the ratio $g(h)/g_0$.

$$\begin{aligned} g(h) &= \gamma \frac{M}{(R+h)^2} \\ \Rightarrow \frac{g(h)}{g_0} &= \frac{\gamma \frac{M}{(R+h)^2}}{\gamma \frac{M}{R^2}} = \left(\frac{R}{R+h}\right)^2 \end{aligned}$$

[15]Please keep in mind that this is all about time-varying quantities, also, when I do not mention the parameter t explicitly.

In the differential equation for h_p:

$$\begin{aligned}
\mathrm{d}h_p &= \frac{g(h)}{g_0}\mathrm{d}h = \left(\frac{R}{R+h}\right)^2 \mathrm{d}h \\
\Leftrightarrow h_p &= \int\limits_0^h \left(\frac{R}{R+h}\right)^2 \mathrm{d}h = -\frac{R^2}{R+h}\Big|_0^h \\
&= -\frac{R^2}{R+h} + R = \frac{-R^2 + R(R+h)}{R+h} = \frac{-R^2 + R^2 + Rh}{R+h} \\
&= \frac{Rh}{R+h}
\end{aligned}$$

The conversion between h_p and h follows: [16]

$$h_p = \frac{Rh}{R+h} \Leftrightarrow h \quad \Leftrightarrow \quad h = \frac{Rh_p}{R-h_p} \tag{5.10}$$

With the fields $\rho(\lambda,\beta,p)$ and $h(\lambda,\beta,p)$ I got all the data, to attach a more justified mass to a three-dimensional partition zones. The method **pep_t** `zone_mass::get_mass`(**place** &`bottom`, **place** &`top`) combines the density from `egp_density::get_density( dataindex index)` and the height information from `egp_geopoth::get_height( dataindex index)` as well as the base area as defined for all zones by the partitioning scheme to compute the air mass of the zone specified via `bottom` and `top` (interval borders for λ, β, p as well a certain point in time, t), in the following manner:

1. It defines the intervals in the grid of the source data that completely contain `top` UN `bottom`. This includes the choice of all grid points inside the zone as well as neighboring points, when the zone border is between those and the included grid points.
2. A mean density is computed from the values out of `egp_density::get_density( dataindex index)` on these points. The mean is taken linearly through a weighted sum and subsequent normalization over all points, where the weight is ruled by $\cos\beta$ – according to the increase of density of grid points in higher latitudes. There is no special weighting in the height (still pressure coordinates) because the linear mean with logarithmically spaced pressure levels already contains the exponential aspect of the density and also there are only 23 pressure levels in the data; in practice there will not be much need to sum over a bigger number of levels anyway. Otherwise some special weighting may be appropriate.
3. Using `egp_geopoth::get_height( dataindex index)` (and linear interpolation to p_{top} and p_{bottom}), a mean height is computed in an analogous way of summing over all (λ_i, β_i).
4. This mean height is multiplied with the stored base area of a zone to produce the associated volume. That finally provides the mass for the given zone when multiplied with the mean density.

A test of this procedure is the comparison of the overall sum of zone masses with an expected mass for the whole atmosphere. [NCAR-Masse] claims a mean whole mass of $5,1480 \cdot 10^{18}$kg

[16] For the calming of skeptic minds when looking at $(R-h_p)^{-1}$: $h_p = R$ is only possible as limit $\lim_{h\to\infty} \frac{Rh}{R+h} = R\lim_{h\to\infty} \frac{1}{\frac{R}{h}+1} = R$

with deviations in the range of $1 \cdot 10^{15} kg$ (changing content of water vapor). With partition Z20-20-30 I reach $5,197 \cdot 10^{18}$kg, without horizontal partition, Z1-1-30 yields $5,206 \cdot 10^{18}$kg. Using only 10 layers, Z1-1-10 is a bit more over the top; this partition leads to $5,729 \cdot 10^{18}$kg. The values are outside the deviation interval given by [NCAR-Masse], but the magnitude matches. Also, there is the question how close the atmospheric mass implied by ECHO-GiSP really has to match the value from literature. The reproduction of global circulation patterns is more in focus than the reproduction of the mass sum.

Also concerning the consistency of mass transport, the computation of the partitioned mass using temperature, humidity and geo potential is at least an important step forward compared to the rough weighting from the last section. It is evident that the inclusion of model data is absolutely necessary when you intend to investigate transport and mixing between different air layers. The comparison of the mass conservation ratio will show this — after I have defined this term.

5.3 Mass conservation ratio, consistency with climate model

I define the mass conservation ration in the model atmosphere that has been partitioned into Z zones (like shown in subsection 5.2.3) with volumes V_i, $i \in [1; Z]$.

$$V = \sum_{i=1}^{Z} V_i \tag{5.11}$$

One needs to consider that the volumes V_i are differing, even varying in time. Because of that – and actually more so because of the huge difference in air density[17] – scalars of the Lagrangian Transport in each zone represent a different air mass, even if thee same number of particles is started in each zone.

The tracer experiment with the mentioned partitioning of the atmosphere consists of the fixed distribution of particles at the beginning and the tracing of the number of particles from zone j in zone i.

$$\text{number of } j\text{-particles in zone} i\text{:} \qquad n_{ij}(t) \tag{5.12}$$

For further considerations and as a basis for comparisons one needs the air mass $\mu_i(\vec{P}(t))$ of one zone. This is derived from the time-varying physical parameters $\vec{P}$ of the climate model (temperature, humidity, zone size via geopotential of the pressure levels).

$$\text{model mass} \qquad \mu_i(\vec{P}(t)) \tag{5.13}$$

A particle from the Zone j represents a fraction of the mass, that the zone contained at time t_0: $\mu_j(\vec{P}(t_0))/n_{jj}(t_0)$. That defines the represented mass of all particles from zone j in zone i:

$$\text{represented mass } m_{ij}(t) \quad = \quad n_{ij} \cdot \frac{\mu_j(\vec{P}(t_0))}{n_{jj}(t_0)} \tag{5.14}$$

[17]Mainly due to the vertical gradient, but also due to horizontal variation.

Summing of these partial masses for each source zone j yields the *transport mass*, meaning the mass, that the zone i now contains due to the transport computation.

$$\text{transport mass } M_i(t) \quad = \quad \sum_{j=1}^{Z} m_{ij}(t) \tag{5.15}$$

This is one aspect, Another is the mass $\mu_i(\vec{P}(t))$, that should be contained in zone i according to the model data at the time t. Ideally, these masses should match. That being the case, my transport computation driven by the wind fields out of the model would be consistent with that model's state (concerning the mass distribution). That would be a confirmation for my method, but also for the climate model itself, since the generated wind is supposed to reflect the actual dynamics of air masses. Well, the measure for investigating that consistency is the

$$\text{mass conservation ratio}\eta(t) \quad = \quad \frac{M_i(t)}{\mu_i(\vec{P}(t))} \tag{5.16}$$

This ratio is equal to 1 at time t_0 by definition, and is expected to vary, more or less, during the numerical transport experiment. The comparison between model run with interactive chemistry and the reference is interesting here, because the generated winds only show a feedback into the climate model (via the chemistry module) in the interactive run. ECHO-GiSP computes semi-Lagrangian transport of the concentrations of the various chemical constituents and the the changed distribution of chemical concentrations influences the model dynamics through changes in the incoming radiation (absorption/reflection of sunlight). In the reference run, the winds are used in the chemistry module, but there is no feedback to the core dynamics.

The first analysis of the mass conservation ratio has been conducted with the assignment of masses via the rough table of the standard atmosphere. Actually, this was what told me clearly that this approach for the air weight is not accurate enough. You can follow that conclusion after looking at Figure 5.10 — the numbers, especially in the higher levels, are so different from the ideal 1, that they do not even fit into their boxes on the plot! In the lowest four layers you could imagine that tuning of other parameters influencing the accuracy of the computations would yield some usable result, but for higher levels, the values are utterly unacceptable. A density table with more entries than Table 5.1 may help a bit, but for sure the dynamical approach with data from the model is better than any static table.

The mass conservation under usage of the model data, in Figure 5.11, draws a far more pleasing picture. The values of η are not free of deviations, but these stay in one order of magnitude and can serve as a basis for discussing the real problems of my method, or the model data.

I close this chapter with a comparison of the mass conservation of the d1000-3 run with wind from the interactive ECHO-GiSP and from the reference[18] , to be seen inFigure 5.12. Grave differences are not discernible; for a more detailed analysis, I also prefer to investigate larger ensembles. Around one billion of particles (without storing trajectories explicitly) should be

[18]These d1000-3 stem from earlier data as shown before, because a re-computation after error correction in a detail has not yet been finished. Because of the the plots are not directly comparable to Figure 5.11, but still comparable among each other.

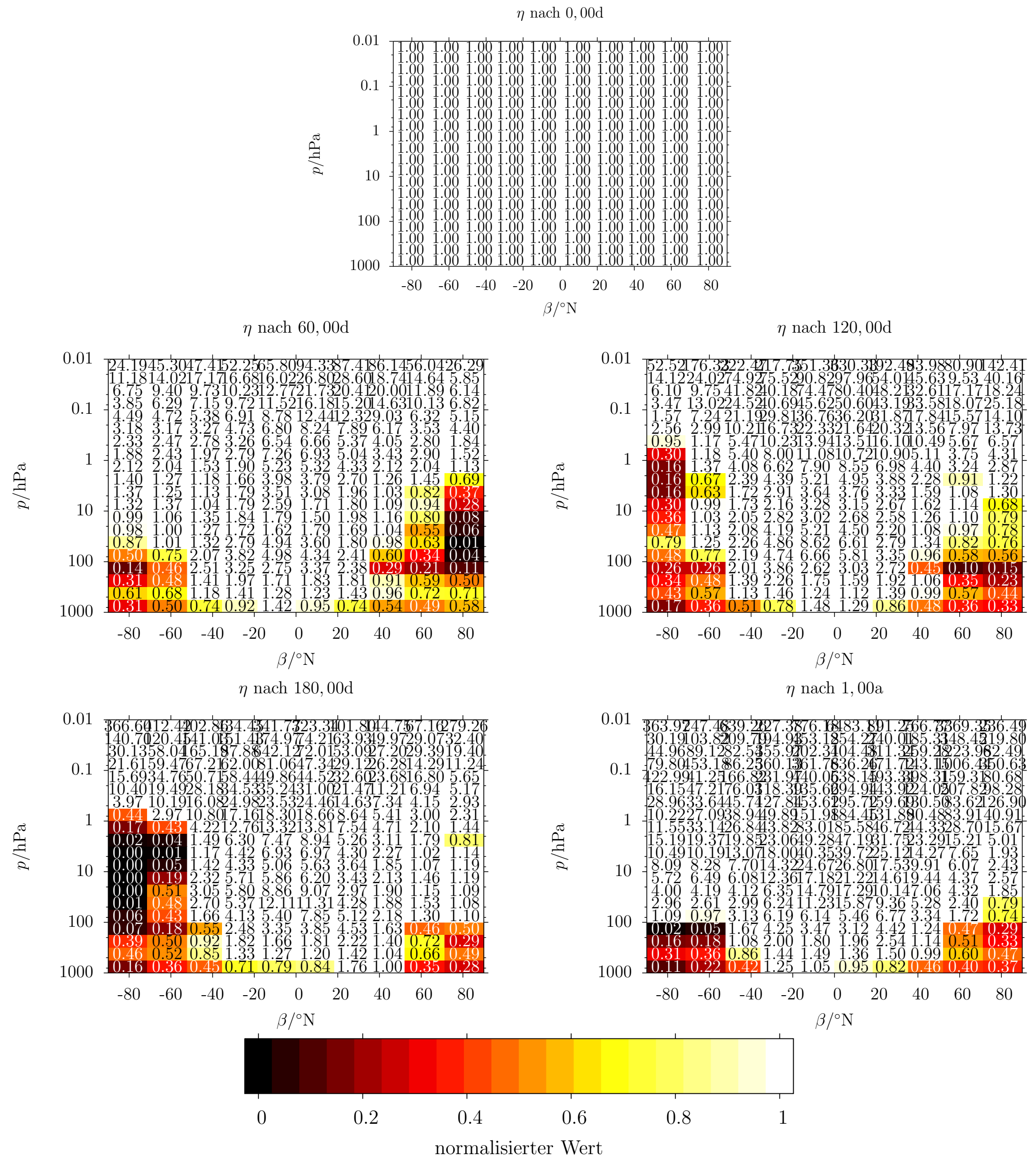


Figure 5.10: Mass conservation ratio d1000-3, standard atmosphere
The first — frightening — incarnation of the mass comparison according to subsection 5.2.4 for the year 1965 of the interactive ECHO-GiSP data as covered by the d1000-3 run. The values of η in the respective zone (the mean over all zones in a latitude/pressure region that are stacked in this zonal view) are given via color coding and the written numbers. The color coding itself is only meaningful from 0 to 1.

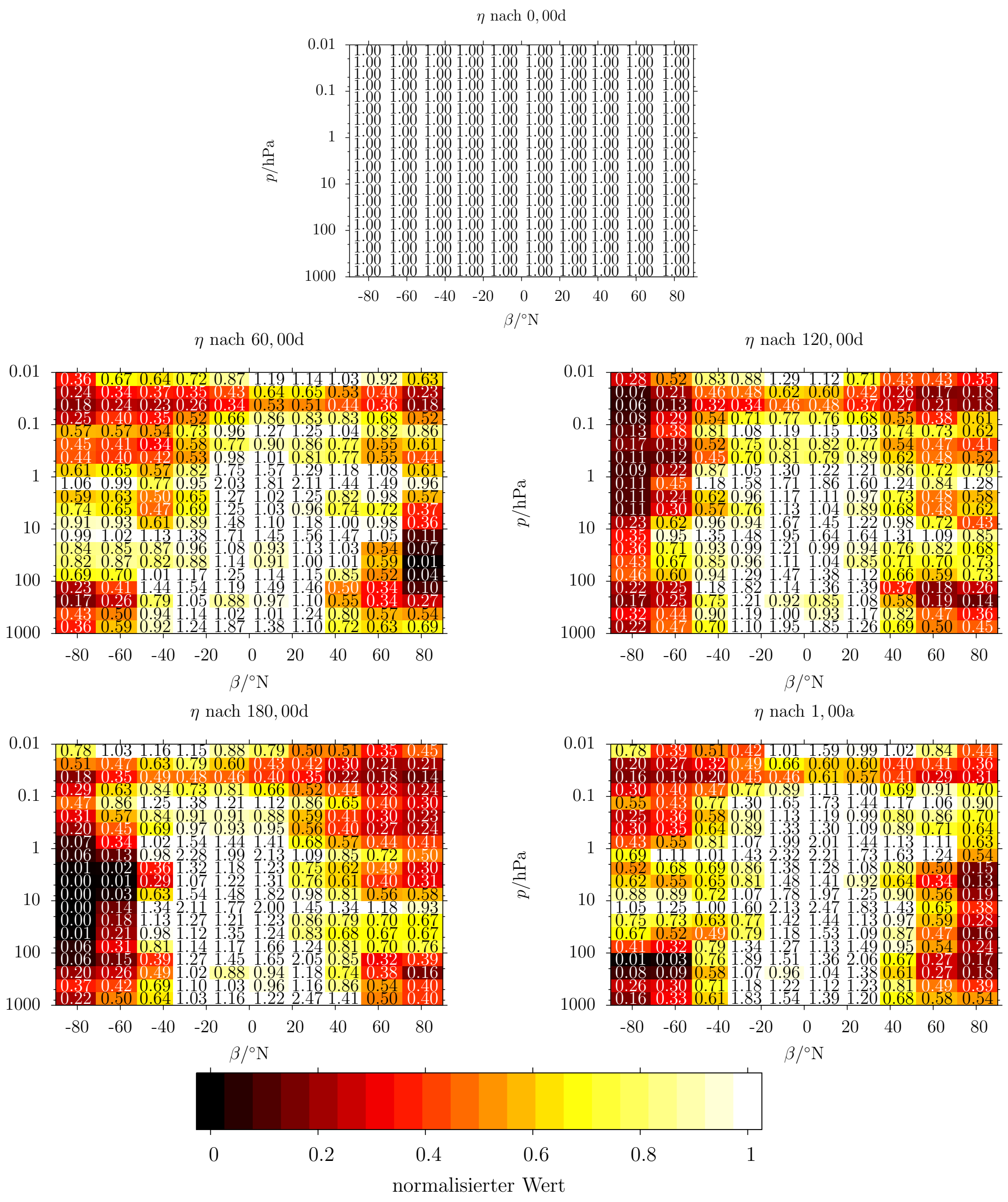


Figure 5.11: Mass conservation ratio d1000-3, dynamic model weight
Computed from the same trajectory ensemble as Figure 5.10, this shows a more balanced picture, thanks to the density data from the model. But the picture is not all bright: Especially at the poles there seems to be a problem of keeping the mass. It has to be determined in future, in how far that is the fault of the trajectory computation and thus could be improved there or the (partial) fault of the climate model itself.

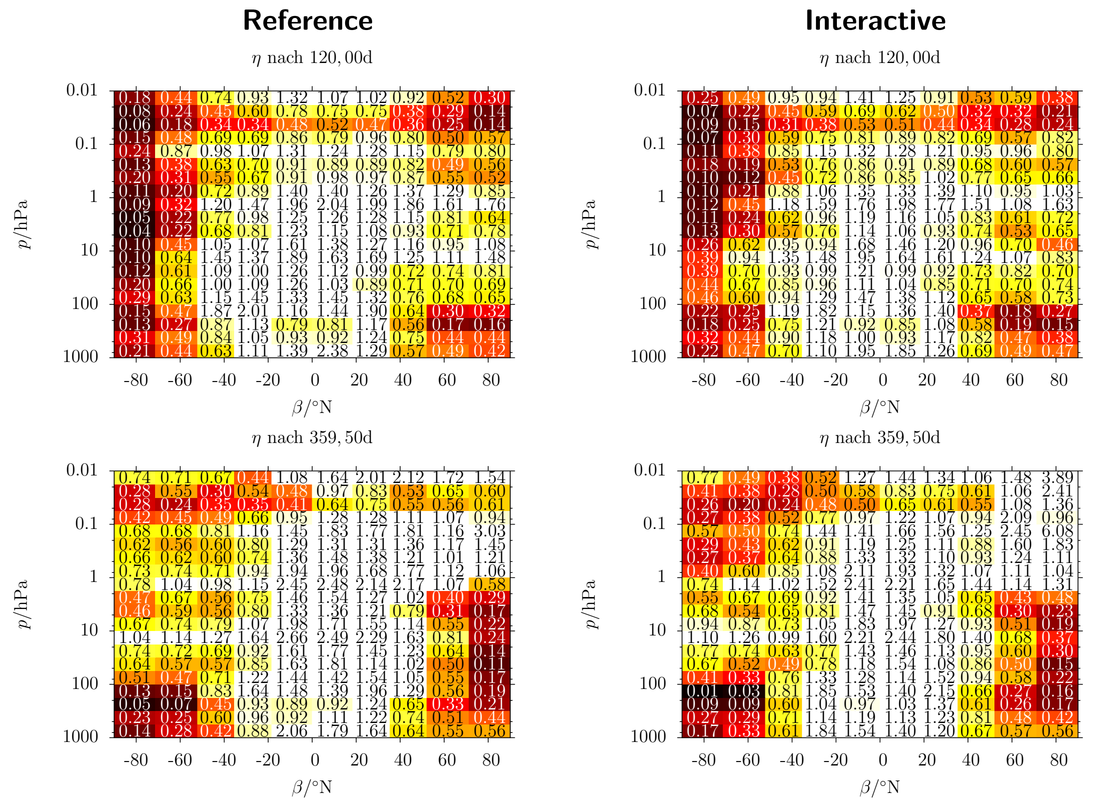


Figure 5.12: Comparison of mass conservation, reference and interactive

...of two earlier d1000-3 runs. You can see some difference in details (exact values of the numbers), while the global picture is very similar.

possible with few weeks / months of computing time on the Grotrian cluster, and are being undertaken.

6 Summary

In this diploma thesis, I developed a numerical approach for the three-dimensional Lagrangian transport in the atmosphere, driven by wind fields from a climate model. The computation of a single integration step in local coordinates circumvents the problem of the geographical coordinates near the poles, thus makes possible really global computation of particle trajectories with a uniform scheme. The software architecture, outlined here and available with complete source code at [PEP-Tracer], has not just been designed, but is currently able to handle large data sets as input and output. The modular construction facilitates the consequent improvement and inclusion of advanced analysis schemes and alternative approaches, e.g. for interpolation and integration.

The transport of ensembles of several million particles over a time range up to 80 years has been computed. Computing time for that on the Grotrian cluster was below a week for one run. The efficiency can be significantly improved, for sure, because performance optimizations so far only occurred in form of the buffer in `echog_pressure_cached`.

The weighting of particles using a local air density derived from the model data provides the theoretical possibility of quantitative investigations on transport and mixing also over large scales in the vertical direction (from the earth surface to the stratosphere and back). A balanced partition of the sphere surface with cells of equal base area, while keeping approximately uniform latitudinal resolution, provides a basic approach to avoid statistical and numerical problems of the smaller cells of a usual grid towards the poles. Practical limits of the mass ratio computation need to be established, while the method needs to be tuned with the mass conservation ratio in mind.

At several occasions, the results up to now have been presented. First steps with two-dimensional transport using simplest integration have been shown in form of an animation at the Potsdamer Wissenschaftsmarkt in the course of the Einstein year 2005 ([WissMarkt]) as well as on a Poster at the IEEE NDES 2005 ([NDES2005]) and the EGU general assembly 2006 ([EGU2006A] and [EGU2006N]). The three-dimensional investigations were presented at the EGU general assembly of this year, in form of a poster ([Orgis2007]). Furthermore, the developed program package has been used for other work on the ECHO-GiSP data, presented in a talk at the EGU general assemble of this year ([Chandra2007]).

In future, even larger ensembles shall be computed for the more detailed evaluation of the method. The analysis methods are actively extended. Produced trajectories need more statistical methods as well as methods of Nonlinear Dynamics, to cover further questions of atmospheric physics, in the end.

More a by-product – but still an important one for me – is the treatment of the linear interpolation in a grid cell of arbitrary dimensions, more thorough than anything I have been able to find in the literature[1].

[1] I am talking of Equation 3.9 *and* Equation 3.13

Bibliography

[Brand2007] Sascha Brand, K. Dethloff, D. Handorf: *Influence of stratospheric ozone chemistry on the atmospheric variability in a coupled atmosphere-ocean-sea ice model*, J. Geophys. Res. (Atm.) submitted 2007 (Poster-Präsentation auf EGU General Assembly: `http://www.cosis.net/abstracts/EGU2007/10114/EGU2007-J-10114.pdf`)

[Bronstein2000] I.N. Bronstein, K.A.Semendjajew, G. Musiol und H. Mühlig: *Taschenbuch der Mathematik*, Verlag Harri Deutsch, Thun und Frankfurt am Main, 5. Aufl., 2000/2001

[Chandra2007] Komalapriya Chandrasekaran, Th. Orgis, U. Schwarz, J. Kurths, S. Brand, K. Dethloff: *Recurrence plots for investigation of nonlinear low frequency variability in atmosphere*, `http://www.cosis.net/abstracts/EGU2007/07719/EGU2007-J-07719.pdf` (Vortrag auf EGU General Assembly 2007)

[EGU2006A] Thomas Orgis, S. Brand, U. Schwarz, J. Kurths, K. Dethloff: *Tracer advection in ECHO-GiSP GCM*, `http://www.cosis.net/abstracts/EGU06/03189/EGU06-J-03189.pdf` (Poster zu EGU General Assembly 2006, als Bilddatei: `http://www.agnld.uni-potsdam.de/%7Eorgis/pep/2006/egu-atmo-poster.png`)

[EGU2006N] Thomas Orgis, S. Brand, U. Schwarz, J. Kurths, K. Dethloff: *Saddle Nodes in Wind Fields*, `http://www.cosis.net/abstracts/EGU06/03193/EGU06-J-03193.pdf` (Poster zu EGU General Assembly 2006, als Bilddatei: `http://www.agnld.uni-potsdam.de/%7Eorgis/pep/2006/egu-nlp-poster.png`)

[NCAR-Masse] Kevin E. Trenberth and Lesley Smith (National Center for Atmospheric Research): *The Mass of the Atmosphere: a Constraint on Global Analyses*, http://www.cgd.ucar.edu/cas/abstracts/files/kevin2003_6.html

[NDES2005] Thomas Orgis, S. Brand, K. Dethloff, J. Kurths: *Atmospheric Tracer Advection*, `http://www.agnld.uni-potsdam.de/%7Eorgis/pep/2005/ndes-poster.png` (Poster zur Konferenz Nonlinear Dynamics in Electronic Systems 2005 (Bilddatei))

[NetCDF] *Network Common Data Form*, `http://www.unidata.ucar.edu/software/netcdf/` (Definition und Software zum NetCDF Datenformat)

[Orgis2007] Thomas Orgis, S. Brand, U. Schwarz, J. Kurths, K. Dethloff: *3D Tracer advection in ECHO-GiSP GCM*, `http://www.cosis.net/abstracts/EGU2007/02313/EGU2007-J-02313.pdf` (Poster zu EGU General Assembly 2007, als Bilddatei: `http://www.agnld.uni-potsdam.de/~orgis/pep/2007/egu.png`)

[PEP] *Pole - Equator - Pole (PEP) — Variability of atmospheric trace constituents along a North-South Transect*, `http://www.awi-potsdam.de/www-pot/atmo/pep/` (Interdisziplinäres Forschungsprojekt mit Teilnahme von HFG-Forschungszentren (AWI, FZK) und Universitäten (Bremen, Potsdam, Karlsruhe))

[PEP-Tracer] Thomas Orgis: *PEP-Tracer Programmpaket zur Analyse von ECHO-GiSP Daten (vorläufiger Arbeitstitel)*, `http://www.agnld.uni-potsdam.de/~orgis/diplom/`, eMail: orgisagnld.uni-potsdam.de (Unter der URL finded man die SVN-Revision 1543 im Unterverzeichnis pep-tracer oder im Tarball pep-tracer-r1543_20070521.tar.bz2; für aktuellere bitte anfragen. In Zukunft evtl. auch über TOCSY: `http://tocsy.agnld.uni-potsdam.de/`)

[Rovatti1998] Riccardo Rovatti, Michele Borgatti, Roberto Guerrieri: *A Geometric Approach to Maximum-Speed n-Dimensional Continuous Linear Interpolation in Rectangular Grids*, IEEE Transactions on Computers 8/47Aug.1998, S. 894-899

[Shepard1968] Donald Shepard: *A two-dimensional interpolation function for irregularly-spaced data*, Proceedings of the 1968 ACM National Conference, 1968, S. 517-524

[Stohl1998] A. Stohl, P. Seibert: *Accuracy of trajectories as determined from the convervation of meteorological tracers*, Q.J.R. Meteorol. Soc. 1241998, S. 1465-1484

[WikiNorm] Wikipedia: *Normatmosphäre (22:26 Uhr, 15. Mai 2007)*, `http://de.wikipedia.org/w/index.php?title=Normatmosph%C3%A4re&oldid=31879482` (Ursprüngliche Datenquelle ist NCAR.)

[WissMarkt] Thomas Orgis: *Film der 2D Bewegung von 10000 Partikeln über 3 Monate*, `http://www.agnld.uni-potsdam.de/%7Eorgis/pep/2005/wissmarkt.mpeg` (erstellt für den Wissenschaftsmarkt anlässlich des Einstein-Jahres in Potsdam)

7 Thanks & Testimony

Thanks

I thank Jürgen Kurths and Udo Schwarz for supervising this diploma thesis.
I thank Sascha Brand and Klaus Dethloff of the Alfred Wegener Institute for the data that gave my computations a meaning.
I thank Timo Felbinger for his advanced LaTeXknowledge (and that he is sharing it with me).
I thank the world for Free Software that paves the way for the free thoughts of a scientist.
I am thankful for numerous inspired conversations and gained insights.

Special thanks go to all who had patience with me.

Testimony

Hereby I testify that I have written this work on my own, using the named sources, to the best of my knowledge. There is no content pulled from other sources than the named ones.

Thomas Orgis