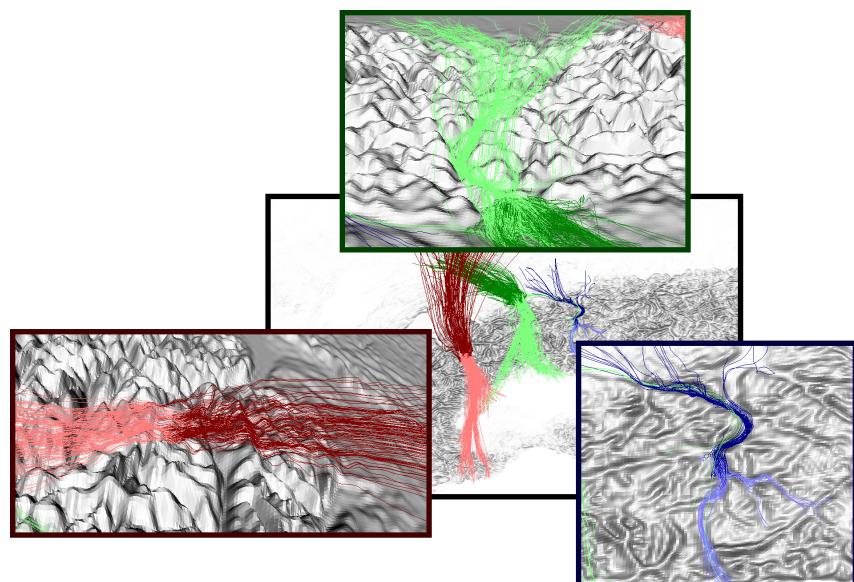


Visualization of Thermally-driven Flow over the Alpine Region



Linus Wigger

Semester Thesis
July 2020

Prof. Dr. Markus Gross, Dr. Tobias Günther



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



computer graphics laboratory

Abstract

The goal of this thesis was to create a tool for computing and displaying wind trajectories. This was done in collaboration with meteorologists from the Institute of Atmospheric and Climate Science. Our main contribution is a C++ analogue of the Lagrangian Analysis Tool LAGRANTO.

To evaluate our results we looked at trajectories obtained from both programs using the same input data. We found our implementation to have better speed and accuracy, but we lack some of the features.

While the focus lay on visualization and analysis of results initially, it later shifted towards reproducing trajectories from the template and improving on that. This worked in the sense that we can obtain trajectories that are practically the same. The visualization became more of a debugging tool than a user-friendly application in the end.

Zusammenfassung

Das Ziel dieser Arbeit war es, ein Tool zur Berechnung und Darstellung von Windtrajektorien zu erstellen. Dies wurde in Zusammenarbeit mit Meteorologen vom Institut für Atmosphäre und Klima getan. Unsere Hauptleistung ist ein in C++ geschriebenes Programm mit ähnlichen Funktionen wie das Lagrangian Analysis Tool LAGRANTO.

Um unsere Resultate auszuwerten haben wir Trajektorien angesehen, die von beiden Programmen mit den gleichen Eingabedaten ausgerechnet wurden. Dabei fanden wir heraus, dass unsere Implementierung besser ist in Sachen Geschwindigkeit und Genauigkeit, dafür fehlen einige der Features.

Obwohl der Schwerpunkt ursprünglich auf dem Visualisieren und Analysieren von Resultaten lag, änderte er sich später, und es wurde mehr darauf geachtet, dass die Trajektorien aus der Vorlage reproduziert und verbessert werden können. Dies funktionierte in der Hinsicht, dass wir praktisch identische Trajektorien produzieren können. Der Visualisierungsteil wurde daher eher zu einem Debuggingwerkzeug als zu einer benutzerfreundlichen Anwendung.

Contents

List of Figures	vii
List of Tables	ix
1 Introduction	1
2 Background	3
2.1 Data	3
2.2 Destaggering	4
2.3 Conversion between coordinate systems	5
3 Method	7
3.1 Numeric integration	7
3.2 Sampling	8
3.2.1 Using local level heights	8
3.2.2 Using adjacent level heights	9
3.2.3 Converting horizontal velocity	9
3.3 Implementation	10
3.3.1 Tracing output	10
3.4 Analysis	11
3.4.1 Qualitative	11
3.4.2 Quantitative	11
4 Results	13
4.1 Qualitative evaluation	13
4.2 Quantitative evaluation	16
4.3 Performance	18

Contents

5 Conclusion	21
Bibliography	23

List of Figures

2.1	Left: Points in staggered grids; Right: Destaggering by averaging two staggered points	5
3.1	Interpolation procedure for sampling at the orange point: The order of operations and interpolation weights (α and β for r_{lon} and r_{lat} , various γ for the vertical axis) depend on the method used.	8
3.2	Structure of the output data: $x_{(i,j)}$ is the value of variable x on trajectory i after the j th timestep	10
4.1	Color according to: Trajectory index(top left), temperature(top right), pressure(bottom left), relative humidity(bottom right)	14
4.2	Colors according to the same criteria as in figure 4.1 above. The colors in the upper left case also depend on the trajectory set: Purple/blue for the forward trajectories, orange/black backward	14
4.3	A closer look of the top left part of figure 4.1 (view from the west)	15
4.4	The relative humidity is high below and low above (view from the west)	15
4.5	Average distance between LAGRANTO trajectories and ours, timestep is $h = 1\text{min}$ on top, $h = 2\text{min}$ in the middle, $h = 5\text{min}$ on the bottom	17
4.6	A few of the trajectories that feed into the plots of figure 4.5. The left side shows roughly 1000 trajectories with $h = 1\text{min}$ per method, on the right it is one tenth of that with $h = 5\text{min}$. The colors are similar to those in the plots, with magenta for the LAGRANTO trajectories	18
4.7	Times for tracing a variable number of particles over 60 timesteps and 7 data files	19

List of Tables

2.1	Important variables	4
-----	-------------------------------	---

1

Introduction

A common problem in meteorology is to find wind trajectories with certain properties, for example passing through a certain region, carrying particularly warm or humid air, or possessing a certain type of shape. Data is more often available in the form of fields containing wind velocity, air pressure, temperature, and any number of other parameters for a certain point in time. To go from fields to trajectories it is necessary to initialize air particles somewhere and calculate how their positions change over time when traveling according to the velocity field.

This project was done in close collaboration with Dr. Michael Sprenger and Lukas Jansing from the Institute of Atmospheric and Climate Science at ETH Zurich. They provided and explained the data. Other contributions include helping decide the next steps and giving feedback on results.

LAGRANTO ([WD97] and [SW15]) is an existing Fortran-based program for computing trajectories from wind velocity fields. LAGRANTO takes as its input a set of initial points (given by their positions and starting time), settings for the integration (reference time, duration, step size), and appropriate data files. Given those, it computes trajectories starting from the initial points and writes them down in an output file.

We aim to create a similar program in C++. It should be able to produce more or less the same results as LAGRANTO given equivalent inputs. While reconstructing LAGRANTO is our base goal, we would like to get better performance and/or results as well. Running certain parts of the code in parallel should help with the speed.

There are LAGRANTO variants for different types of input data. We work with the COSMO version because all our input data is in the COSMO model (see also [SDS⁺03] or [BSF⁺11]). The data was obtained from a local ETH simulation setup similar to MeteoSwiss. It covers four days in November 2016 where Föhn winds from the south cross the Alps.

The trajectories we obtain as our solution are compared to the results from LAGRANTO. We visualize trajectories using VTK, the Visualization Toolkit ([SML06]). Looking at the trajectory

1 Introduction

shapes in 3D allows us to search for errors in the code and interesting features of the data.

Our viewer should be able to load and render different sets of trajectories. Filtering and coloring trajectories according to user-defined criteria are also important. In the best case, all of this can be done at runtime using a GUI we create for that purpose.

2

Background

2.1 Data

We work on a set of NetCDF files containing assorted meteorological data in the COSMO model which were simulated and kindly provided by Lukas Jansing. Most of the files contain data at a certain point in time and have names along the lines of "lfff01234000.nc". The number in the filename corresponds to the time past the reference date using the format DDHHMMSS, so for example "lfff00015000.nc" would contain the data at one hour and fifty minutes. In addition, there is a file "lfff00000000c.nc" (note the c at the end), called the constants file, which holds constant variables like the height of the surface at a certain point. The reference time for our data is 2016, Nov, 21, 00:00 and the files go from "lfff00000000.nc" (reference date) to "lfff03225000.nc" (3 days 22 hours 50 minutes later) with 10 minutes between files.

Most of the important variables are stored as three-dimensional arrays. The three dimensions are called *rlon*, *rlat* and *level*. *rlon* and *rlat* are coordinates in a rotated geographical coordinate system. The *levels* correspond to the vertical position of a point, but it is not a simple linear transformation. Instead, the constants file holds the necessary information to convert *levels* to actual height. Further details can be found in section 2.3. Unless noted otherwise, the grid size for our data is always $1158 \times 774 \times 80$. The domain ranges from -6.8° to 4.77° in *rlon* and from -4.4° to 3.33° in *rlat*. The uppermost level ends at a height of 22km.

Table 2.1 gives an overview of the most interesting variables. The three variables *UVW* define the velocity field: *U* is the eastward (in the rotated system) component of the wind, *V* the northward component, and *W* the upward component. All three have the same units (*m/s*) and similar but not equal grids. The grids of *UVW* are staggered: All vertices in one grid are translated by half a cell size in one direction. Section 2.2 describes how the staggered grids are handled.

HHL maps the *level* of a grid point to a physical height. Like *W*, it is staggered in the vertical

2 Background

Name	Description	Dimensions	Time-invariant	Staggering	Unit
U	$r\text{lon}$ component of velocity	3	no	$r\text{lon}$	m/s
V	$r\text{lat}$ component of velocity	3	no	$r\text{lat}$	m/s
W	vertical component of velocity	3	no	level	m/s
HHL	$\text{level-to-height map}$	3	yes	level	m
$HSURF$	height of surface	2	yes	none	m
P	pressure	3	no	none	Pa
T	temperature	3	no	none	K
$RELHUM$	relative humidity	3	no	none	$\%$

Table 2.1: Important variables

direction and needs to be destaggered before it can be used with most other variables. HHL is important because the particle positions have a real height in meters as their third component and there needs to be a way to find grid coordinates from the particle position.

$HSURF$ contains the height of the surface for given ($r\text{lon}$, $r\text{lat}$)-coordinates. It is mainly used to prevent particles from leaving the domain through the ground.

The pressure P , temperature T , and relative humidity $RELHUM$ are not relevant for the tracking, but they work well as examples of the kind of data one may wish to track along the trajectories.

2.2 Destaggering

U , V , W , and HHL are stored in staggered grids, recognizable by using the names $srlon$, $srlat$ and level1 for certain axes. The staggered grid coordinates lie halfway between the unstaggered grid points. Destaggering is done by averaging the values at two vertices that are adjacent in the staggering direction, then storing the result at the grid position between those vertices. Figure 2.1 shows how staggered ($srlon$, $r\text{lat}$)- and ($r\text{lon}$, $srlat$)-grids are converted to ($r\text{lon}$, $r\text{lat}$). The image also shows that the destaggered version of the grid has one row/column less than the staggered original.

The dimensions of the UVW grid are effectively $1157 \times 773 \times 80$. Compared to the default size, this is one element less in $r\text{lon}$ and $r\text{lat}$. The number of levels remains at 80 because the staggered axis level1 has size 81.

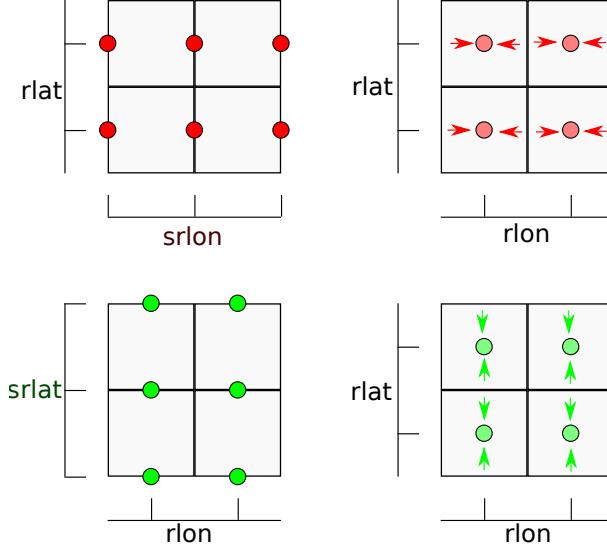


Figure 2.1: Left: Points in staggered grids; Right: Destaggering by averaging two staggered points

2.3 Conversion between coordinate systems

The velocities U, V, W , as well as other variables like temperature, are defined on a regular grid with axes corresponding to $(rlon, rlat, level)$.

$rlon$ and $rlat$ can be converted into lon and lat if given the (global) coordinates of the rotated north pole $(\lambda_{pole}, \phi_{pole})$. In our data, ϕ_{pole} is always 43° and λ_{pole} is -170° . Converting coordinates (λ_r, ϕ_r) in the rotated system to the global coordinates (λ_g, ϕ_g) is done as follows:

$$\phi_g = \sin^{-1}(\cos(\phi_{pole}) \cdot \cos(\phi_r) \cdot \cos(\lambda_r) + \sin(\phi_{pole}) \cdot \sin(\phi_r)); \quad (2.1)$$

$$c_1 = \sin(\phi_{pole}) \cdot \cos(\lambda_r) \cdot \cos(\phi_r) + \cos(\phi_{pole}) \cdot \sin(\phi_r) \quad (2.2)$$

$$c_2 = \sin(\lambda_r) \cdot \cos(\phi_r) \quad (2.3)$$

$$c_3 = \sin(\lambda_{pole}) \cdot c_1 - \cos(\lambda_{pole}) \cdot c_2 \quad (2.4)$$

$$c_4 = \cos(\lambda_{pole}) \cdot c_1 + \cos(\lambda_{pole}) \cdot c_2 \quad (2.5)$$

$$\lambda_g = \text{atan2}(c_3, c_4) \quad (2.6)$$

The reverse transformation to get (λ_r, ϕ_r) from (λ_g, ϕ_g) is done as follows:

$$\phi_r = \sin^{-1}(\cos(\phi_{pole}) \cdot \cos(\phi_g) \cdot \cos(\lambda_g - \lambda_{pole}) + \sin(\phi_{pole}) \cdot \sin(\phi_g)); \quad (2.7)$$

$$c_1 = -\sin(\lambda_g - \lambda_{pole}) \cdot \cos(\phi_g) \quad (2.8)$$

$$c_2 = -\sin(\phi_{pole}) \cdot \cos(\phi_g) \cdot \cos(\lambda_g - \lambda_{pole}) + \cos(\phi_{pole}) \cdot \sin(\phi_g) \quad (2.9)$$

$$\lambda_r = \text{atan2}(c_1, c_2) \quad (2.10)$$

Not included in the above equations are the steps to ensure that all λ and ϕ remain in the intervals $[-180, 180]$ and $[-90, 90]$ respectively. The λ switch from one end of the interval to the other while the ϕ are simply held at $\pm 90^\circ$

2 Background

The vertical coordinates z are given in meters above sea level and need to be mapped to grid levels. To that purpose, we have the time-invariant scalar field HHL which maps (staggered) levels at specific grid points to their height.

The fact that the values are stored in a regular grid that corresponds to an irregular real shape means that one needs to be careful when interpolating values given at coordinates between grid points. Two possible methods are discussed in the following chapter in section 3.2.

3

Method

3.1 Numeric integration

Solving differential equations of all types numerically is a topic for itself. In this section we limit ourselves to describing the methods we use to find the next point of a trajectory given the time-dependent velocity field $UVW(p, t)$, the starting position p_{t_0} , and a timestep of size h .

LAGRANTO uses an iterative variant of Euler's method, described in [WD97] as a variant of Pettersen's method ([Pet56]). The next point p_{t_0+h} is computed using the average of the velocities at the original point p_{t_0} and the current guess for p_{t_0+h} . The guess starts at p_{t_0} and is recomputed three times.

One step using the Iterative Euler method goes as follows:

$$v_0 = UVW(p_{t_0}, t_0) \quad (3.1)$$

$$v_1 = UVW(p_{t_0}, t_0 + h) \quad (3.2)$$

$$q_1 = p_{t_0} + h \frac{v_0 + v1}{2} \quad (3.3)$$

$$v_2 = UVW(q_1, t_0 + h) \quad (3.4)$$

$$q_2 = p_{t_0} + h \frac{v_0 + v2}{2} \quad (3.5)$$

$$v_3 = UVW(q_2, t_0 + h) \quad (3.6)$$

$$p_{t_0+h} = p_{t_0} + h \frac{v_0 + v3}{2} \quad (3.7)$$

We preferred to use the classical Runge-Kutta integration scheme. It uses four samples of UVW per iteration like the iterative Euler method, but the iterative Euler method has an error $\mathcal{O}(h^3)$ per step whereas the Runge-Kutta method has a lower error $\mathcal{O}(h^5)$ (see for example [PTVF92]).

3 Method

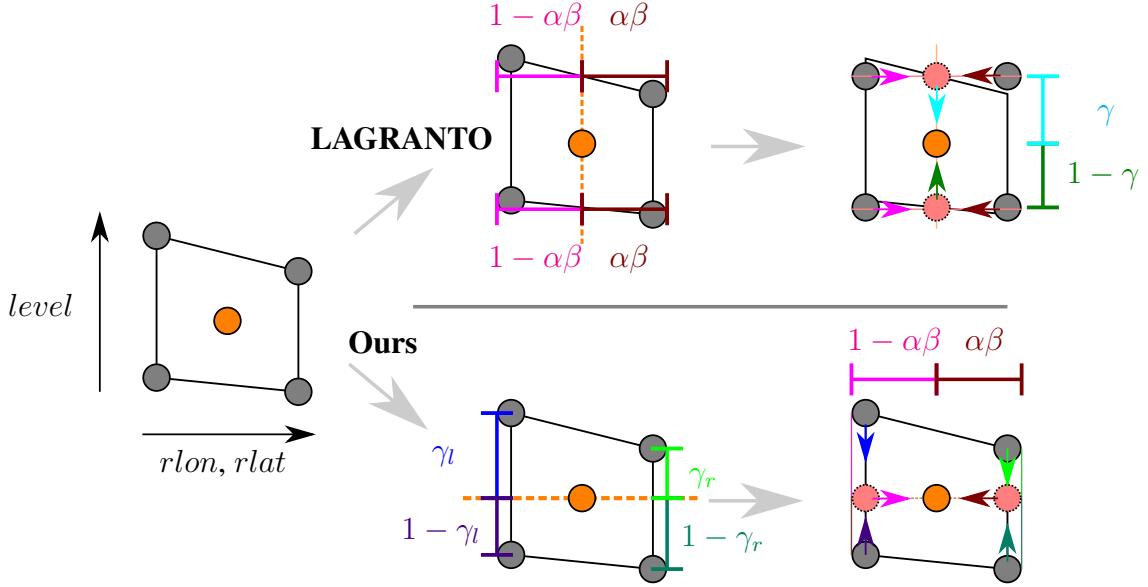


Figure 3.1: Interpolation procedure for sampling at the orange point: The order of operations and interpolation weights (α and β for $rlon$ and $rlat$, various γ for the vertical axis) depend on the method used.

The steps for the classical Runge-Kutta Method are:

$$k_1 = UVW(p_{t_0}, t_0) \quad (3.8)$$

$$k_2 = UVW\left(p_{t_0} + k_1 \frac{h}{2}, t_0 + \frac{h}{2}\right) \quad (3.9)$$

$$k_3 = UVW\left(p_{t_0} + k_2 \frac{h}{2}, t_0 + \frac{h}{2}\right) \quad (3.10)$$

$$k_4 = UVW(p_{t_0} + k_3 h, t_0 + h) \quad (3.11)$$

$$p_{t_0+h} = p_{t_0} + (k_1 + 2k_2 + 2k_3 + k_4) \frac{h}{6} \quad (3.12)$$

3.2 Sampling

Sampling the velocity field UVW at a certain position (x,y,z) and time t is a common operation during particle tracing. Because UVW is defined in m/s on a $(rlon, rlat, level)$ -grid and the position is given in $(^{\circ}, ^{\circ}, m)$ some conversions are necessary.

3.2.1 Using local level heights

Mapping x and y to positions in the $(rlon, rlat)$ -grid is done using the fact that the grid is rectangular and regular: Assuming (λ_0, ϕ_0) are the $(rlon, rlat)$ -coordinates of the grid point $(0,0)$ and the distance to the next vertex is Δ_λ (in $rlon$ -direction) or Δ_ϕ (in $rlat$ -direction), the

grid coordinates are obtained from the real coordinates (x,y) as $(\frac{x-\lambda_0}{\Delta_\lambda}, \frac{y-\phi_0}{\Delta_\phi})$. By rounding those grid coordinates up or down we get the coordinates of the nearest grid points.

The upper part of figure 3.1 shows how LAGRANTO interpolates between levels: In a first step, two level heights for the upper and lower level are constructed (shown as split magenta and dark red lines). This requires a binary search to locate two levels for the z -coordinate of the sampling point. LAGRANTO essentially performs trilinear interpolation in a box-shaped cell whose exact position and height depends on (x,y) . Notice how in the third step the corner points have been moved slightly up or down: The differing real heights of the grid points only matter when determining the local level heights. For the final interpolation, all four corner points on one level are considered to be at the same height.

There are three interpolation weights (α, β, γ) for the axes $(rlon, rlat, level)$. They are computed as $(\frac{x-x_0}{x_1-x_0}, \frac{y-y_0}{y_1-y_0}, \frac{z-z_0}{z_1-z_0})$, where x_0 and y_0 are the coordinates of the western and southern grid points and z_0 the (interpolated) height of the lower level. Accordingly, (x_1, y_1, z_1) is the position of the upper northeastern corner.

3.2.2 Using adjacent level heights

The grid coordinates of (x,y) and the bilinear interpolation weights α and β along the $rlon$ and $rlat$ axes are computed the same way as in the previous subsection.

The lower part of figure 3.1 shows how we compute the interpolated value at the sample point. On each of the four (two in the picture) grid columns nearby, we compute interpolation weights $\gamma_i = \frac{z-z_{0i}}{z_{1i}-z_{0i}}$ after finding lower and upper heights z_{0i} and z_{1i} with a binary search on column i in HHL . The last step is bilinearly interpolating between those four values. As mentioned, the weights α and β for the horizontal interpolation are the same ones that LAGRANTO uses. For the vertical interpolation, LAGRANTO uses only one set of weights (γ and $1 - \gamma$). Our version has different weights on each column (the pairs for γ_l and γ_r are visible in the picture), making the sampling process slightly more complicated and hopefully accurate.

3.2.3 Converting horizontal velocity

The horizontal velocities U and V are given in m/s but the corresponding particle coordinates are $(rlon, rlat)$, so U and V need to be converted into $^\circ/s$ before the integration step. Given (U, V) at a point with coordinates (λ, ϕ, z) , we use the formula:

$$(U', V') = \left(\frac{U}{111200 \cos(\phi)}, \frac{V}{111200} \right) \quad (3.13)$$

The constant 111200 corresponds to the length in meters of one degree (in either direction) at the equator. The complete velocity vector (U', V', W) has units $(^\circ/s, ^\circ/s, m/s)$ and is used for integrating later.

3 Method

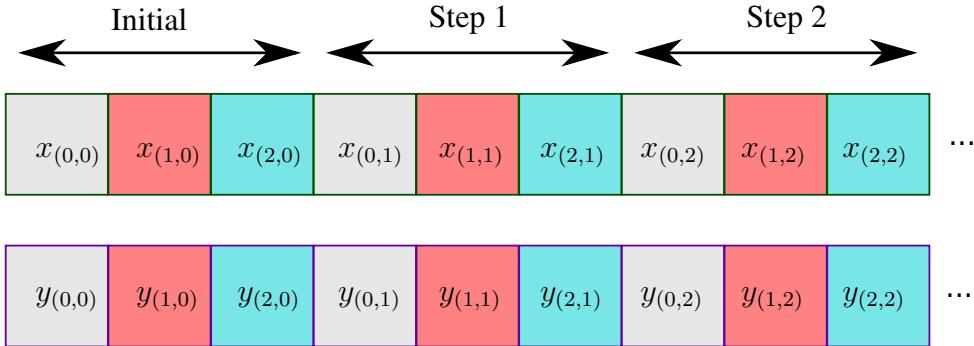


Figure 3.2: Structure of the output data: $x_{(i,j)}$ is the value of variable x on trajectory i after the j th timestep

3.3 Implementation

The tracing process starts by asking the user for initial points, start and end time, size of the timestep, and additional settings like which variables to track, what type of integrator to use, plus a few other options that matter for debugging and comparing to LAGRANTO (mostly concerning how UVW is sampled). After allocating space for the output data, the UVW fields are extracted from the first three appropriate files. As the simulation runs, the oldest field is regularly replaced by new UVW from the next file in line, minimizing the memory needed at runtime.

At each step, all trajectories have to be advanced by h . Those that have left the domain are kept at their last positions while the others get positions for the next time step based on the velocity at their current position. The iteration over all trajectories was parallelized with OpenMP.

Particles that leave the domain need to be handled in a special way. The exact procedure depends on the settings. In LAGRANTO, points outside the allowed $(rlon,rlat)$ -intervals have their coordinates fixed at $(-999, -999, -999)$ for all remaining iterations. In our version they are simply left where they are. Points that leave the domain on the z -axis can be made to jump back by setting an optional flag (the flag is set by default). Jumping is done by translating the point in question by a fixed amount (usually $10m$) in vertical direction.

3.3.1 Tracing output

The results from the particle tracing are written into a NetCDF file which contains an array for each variable. Time, coordinates in both (lon,lat) and $(rlon,rlat)$, and height are always stored. Other variables like temperature or pressure need to be included in the initial user input.

The number of elements per array is $N_{tra} \cdot (N_{steps} + 1)$, where the number of trajectories is N_{tra} and N_{steps} is the number of integration steps. The arrays are ordered according to timestep first and trajectory second. Figure 3.2 shows an example with 2 arrays (x and y), 3 trajectories (gray, red, blue), and 5 timesteps (plus the initial state), for a total of 18 elements per array.

This format matches the output file of LAGRANTO, allowing us to compare the results directly. It should be noted that while LAGRANTO uses $(rlon,rlat)$ for the computations, the output

only contains coordinates in (lon, lat) by default.

3.4 Analysis

The following subsections describe how we evaluate our output trajectories. We first confirm that their shapes look roughly the same as in LAGRANTO, then we use more exact measurements to determine the quality of our implementation.

3.4.1 Qualitative

We visualize the trajectories using VTK ([SML06]). The trajectories are loaded from an output file and drawn in 3D. The user can move the camera to get a better view. A surface obtained from *HSURF* is also displayed to give a context beyond just the trajectory shape.

r_{lon} and r_{lat} or lon and lat (depending on the settings) correspond to the x and y axes of the renderer. For comparing the results to those from LAGRANTO, the global coordinates (lon, lat) are used because LAGRANTO includes only those in its output. The rendered points need to have their z components rescaled because the horizontal and vertical coordinates have vastly different scales (a difference of 1 is a lot on the x - and y -axes but 100 on the z -axis is barely noticeable). All heights are rescaled by an arbitrary factor of $5 \cdot 10^{-5}$. While this rescaling does not lead to exact proportions (one unit in horizontal direction does not correspond to the same distance as one unit in the vertical direction), it helps make the shapes recognizable.

3.4.2 Quantitative

We compare different integrators by giving them the same input and measuring the average distance between their outputs over time. We split the distance into a horizontal and vertical component because the units are different and the total distance would be dominated by the much more chaotic vertical part otherwise.

We use the output from LAGRANTO as a reference and look at how the difference to our method evolves over time.

Regarding the performance, we run simulations with the same integrator settings and initial points using both LAGRANTO and our method. We measure the time needed for several differently-sized sets of initial points.

The total time is expected to increase linearly with the number of integration steps and the number of files needed. The number of files read depends only on the duration of the simulation and is assumed to be constant. The number of integration steps on the other hand is generally proportional to both the number of trajectories and the number of timesteps. In our tests we only vary the former. Special cases where many trajectories leave the domain and require more or less work afterward (depending on the settings) are ignored.

4

Results

4.1 Qualitative evaluation

Figures 4.1 and 4.2 show examples of trajectories we computed and visualized. In the top left part, each trajectory has a constant color which depends on its index: Starting at blue for the first trajectory and ending at purple. Figure 4.2 includes another color scheme (black-orange) for the trajectories that were computed with a negative timestep. The trajectories in the upper right part are colored according to temperature: Blue is cold, white is 0°C , red is warm. The lower two sections visualize pressure (red at high values, bright at low ones) and relative humidity (yellow at 0% and blue at 100%).

Figure 4.1 shows a set of trajectories starting on the south side of the Alps. The particles start at 13:00 of the second day (22.Nov2016) at a latitude of 45.2°N and are traced for 5 hours as they move north. The initial points are spread across $7.5 - 10.5^{\circ}\text{E}$ in *lon*-direction and 2500 – 3000m in height. The three variables temperature, pressure, and humidity appear to be strongly correlated: The colors have similar patterns, starting cold/high/humid and becoming warmer/lower/drier after passing the mountains.

Figure 4.2 shows several trajectories passing over Chur (around 9.53°E 46.85°N). The starting time for the simulation is midnight between the 22nd and 23rd of November. The integration time is 3 hours in both directions (so from 21:00 to 3:00). Temperature and pressure behave as expected, becoming lower at higher altitudes. The relative humidity seems to have two wet and dry regions each.

Figure 4.3 shows another view of the trajectories from figure 4.1. One can see how trajectories ascend and descend with the surface, resulting in wave shapes over the mountains. Less visible are the jumps that happen when the trajectory goes through the ground. There are small hiccups on some of the curves, for example one in the upper left corner of the zoomed-in section.

4 Results

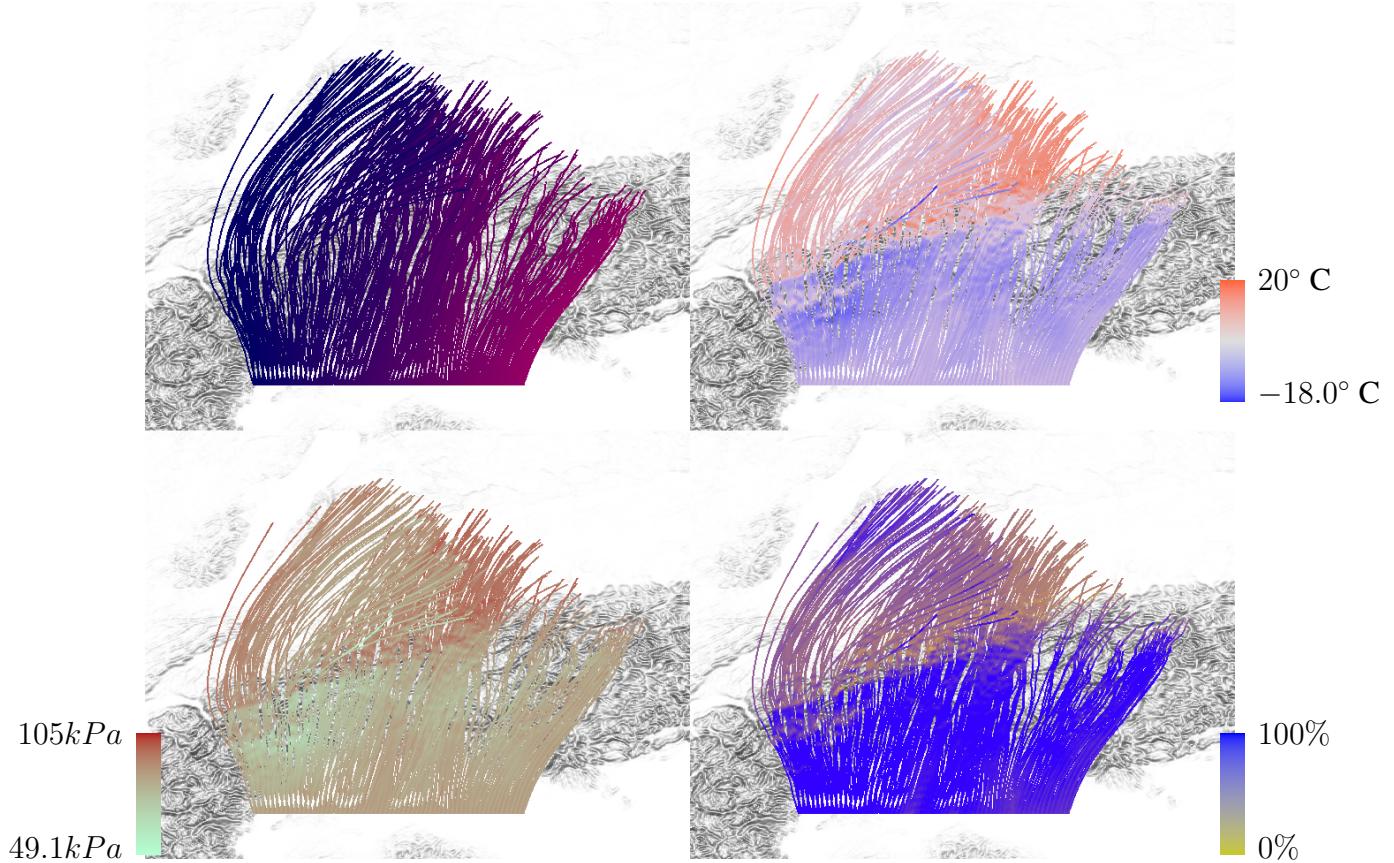


Figure 4.1: Color according to: Trajectory index (top left), temperature (top right), pressure (bottom left), relative humidity (bottom right)

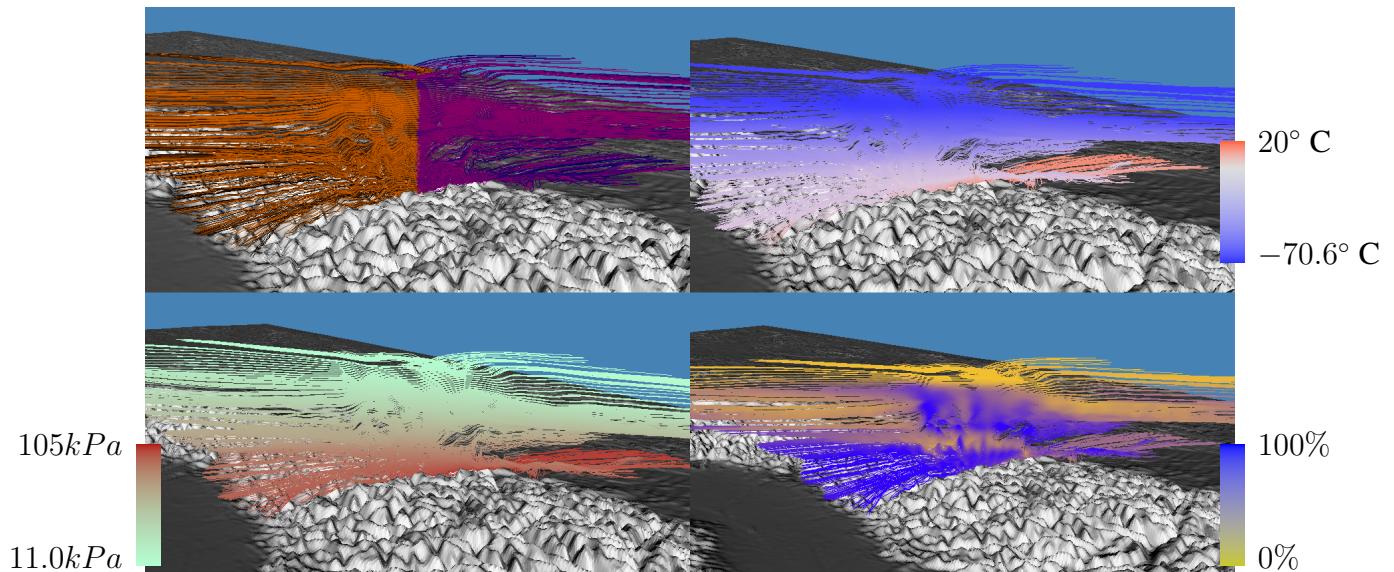


Figure 4.2: Colors according to the same criteria as in figure 4.1 above. The colors in the upper left case also depend on the trajectory set: Purple/blue for the forward trajectories, orange/black backward

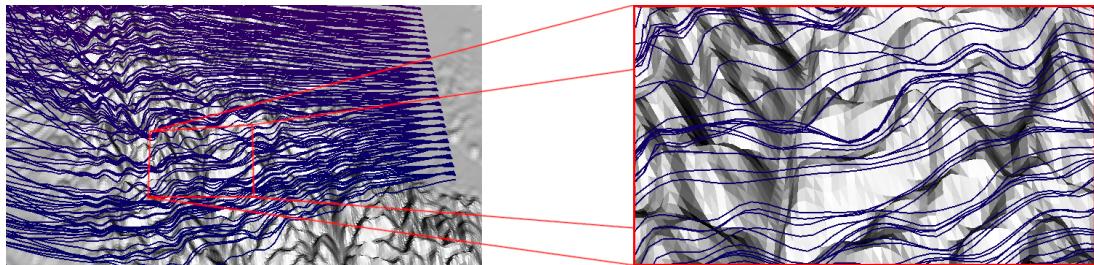


Figure 4.3: A closer look of the top left part of figure 4.1 (view from the west)

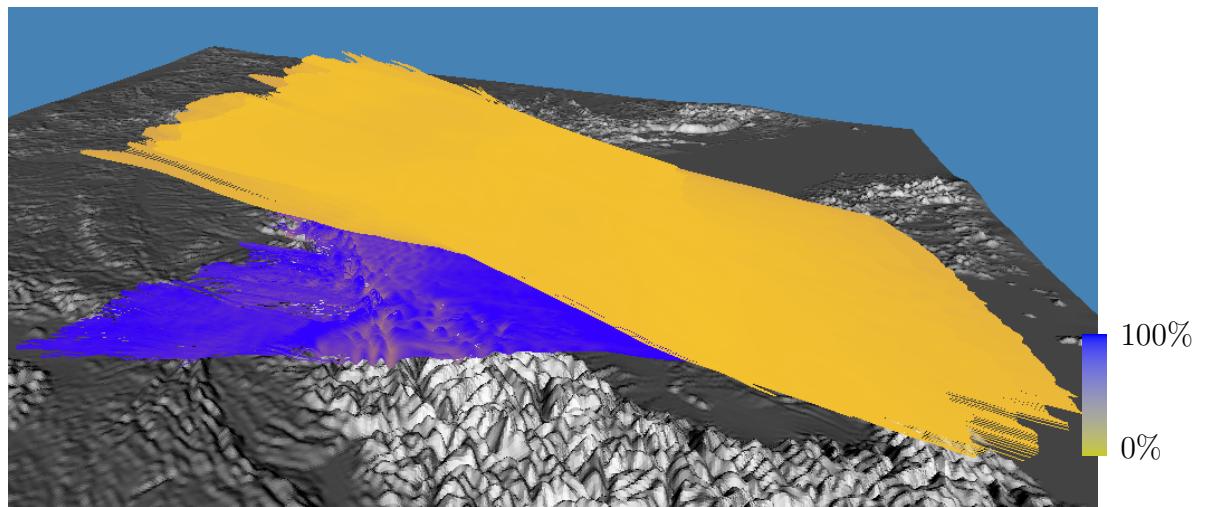


Figure 4.4: The relative humidity is high below and low above (view from the west)

4 Results

Figure 4.4 shows that the air high above ($14 - 16\text{km}$) has low relative humidity, especially compared to the lower trajectories (which start around height 4km). The trajectories in figure 4.4 run from 3:00 to 9:00 on the third day of the data (November 23) and actually consist of a forward and backward component starting at 6:00 each. The overall shape of the upper and lower trajectories is different. The upper trajectories move slightly more to the east and cover a larger area, implying faster speeds for winds that are never blocked by mountains.

4.2 Quantitative evaluation

The plots in figure 4.5 plot the difference between trajectories computed by LAGRANTO and five variants of our tracing algorithm. All values are averaged over around 7000 trajectories (some of which are pictured in figure 4.6). The five variants are:

- Copying LAGRANTO: Settings to perform almost the same operations as LAGRANTO. The only intended differences are near the ground.
- Sample W correctly: Examining the customized LAGRANTO code that we received showed that the vertical velocity W was being sampled on a staggered grid even after it had been destaggered. This is an error.
- Level interpolation on 4 columns: Use the procedure described in section 3.2.2.
- Runge-Kutta instead of Iterative Euler: Changes the ODE solver.
- All improvements: Combines the three variants above.

The test trajectories should not leave the domain because we handle particles outside in a different way than LAGRANTO and here we want to measure only the differences from integrating. As seen in figure 4.6, all trajectories start and end at acceptable (lon, lat)-coordinates. Collisions with the ground are avoided by initializing all particles at heights of at least 7km . There is a very small number of trajectories that reach the surface when the time step is $h = 5\text{min}$. This is unfortunate because it distorts the plots, even though the effect is lessened by averaging with several thousand unproblematic trajectories.

What can be seen in the plots of figure 4.5 is that the black line (representing the most LAGRANTO-like setting) stays at a very low value. There are small variations, but apart from the boundary cases, we can reproduce the LAGRANTO results almost exactly.

The right side which measures the average vertical distance looks very chaotic and is not very useful for gathering information. The reason for this is unclear. It is possible that slight changes in coordinates have major effects on the value of W , more so than for U and V .

The choice of integrator does not matter that much for small timesteps but in the case of $h = 5\text{min}$, the blue Runge-Kutta curve dominates the left plot. This is expected because the Euler integrator works worse with larger timesteps. Curiously, the solid green curve ("All improvements") stays closer to the other methods, even though it also uses the Runge-Kutta integrator.

It appears that the choice of how to sample across *levels* makes more of a difference than

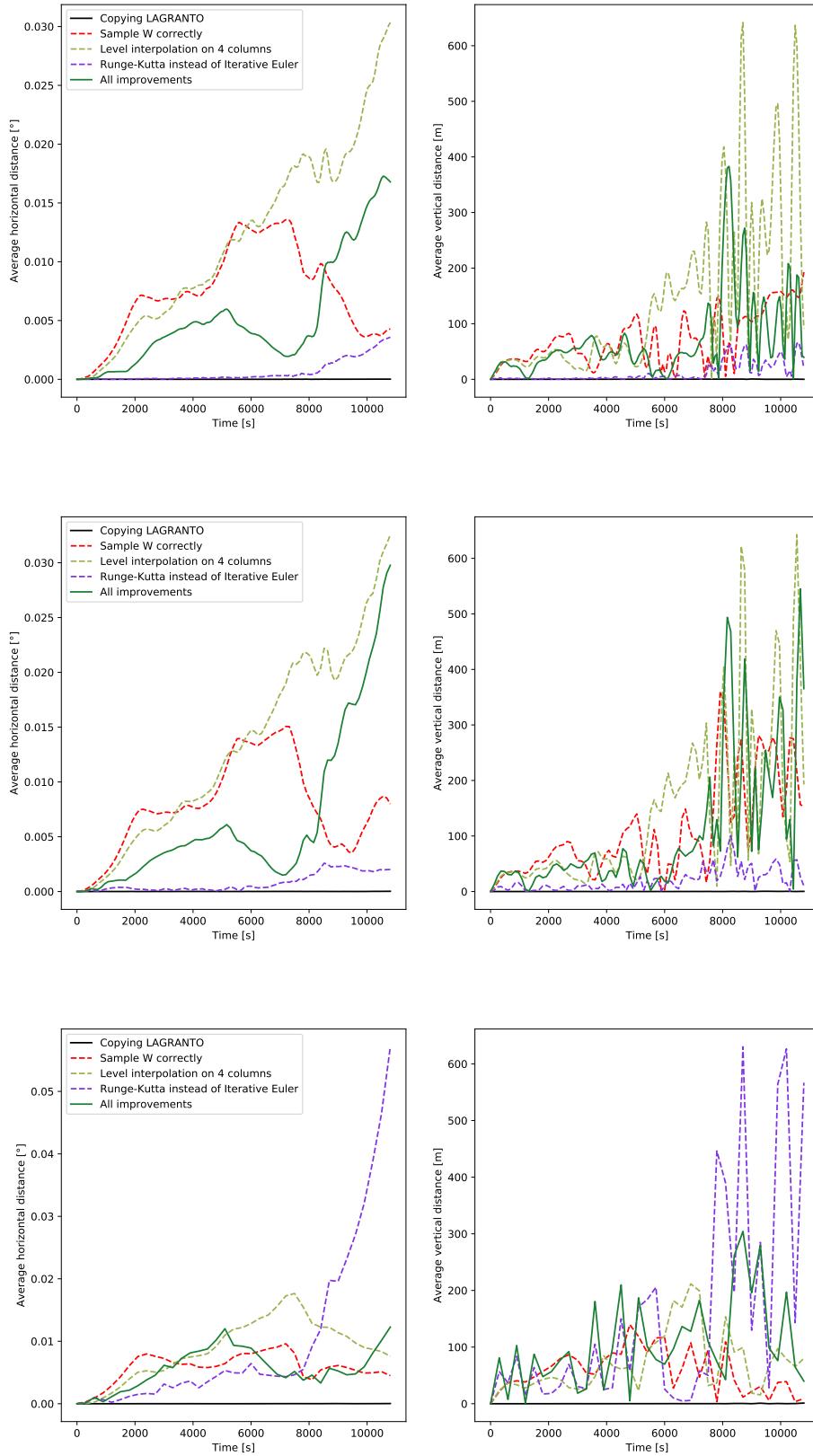


Figure 4.5: Average distance between LAGRANTO trajectories and ours, timestep is $h = 1\text{min}$ on top, $h = 2\text{min}$ in the middle, $h = 5\text{min}$ on the bottom

4 Results

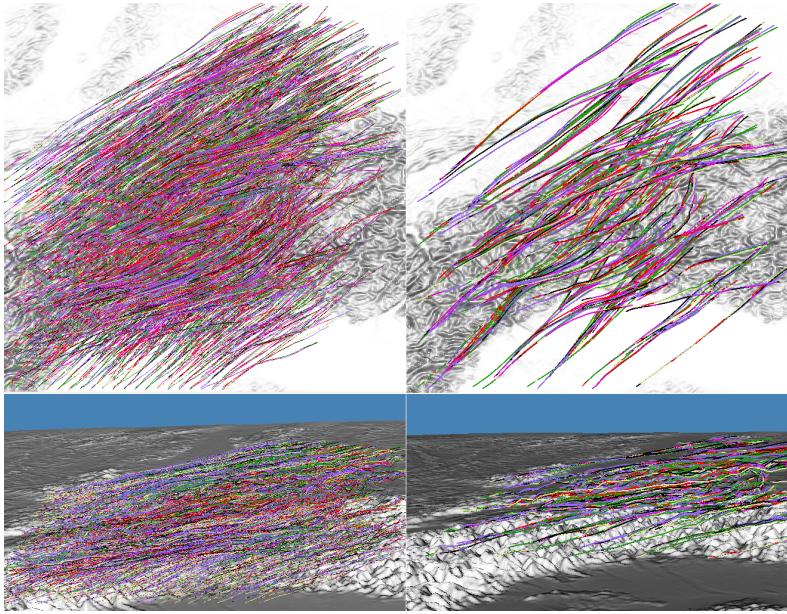


Figure 4.6: A few of the trajectories that feed into the plots of figure 4.5. The left side shows roughly 1000 trajectories with $h = 1\text{min}$ per method, on the right it is one tenth of that with $h = 5\text{min}$. The colors are similar to those in the plots, with magenta for the LAGRANTO trajectories

correcting the sampling of W . The fact that both of those affect the z -axis may be another factor in the irregular curves on the right side.

4.3 Performance

There are two main performance bottlenecks: Reading the data and doing computations on each particle and timestep. We added a simple OpenMP parallelization for iterating over all trajectories during the simulation phase and hope to at least match the speed of LAGRANTO with that.

Figure 4.7 plots the measured times for a test case of computing different numbers of trajectories. As expected, the time scales linearly in the number of trajectories. Our method is faster than LAGRANTO in all cases. In the early part, where the reading of the data makes up almost all of the time, our version does in roughly 1.5 minutes what LAGRANTO does in 4 minutes. Increasing the number of trajectories, it becomes evident that our version also has a better scaling, even when unparallelized.

The rather slow speed of LAGRANTO might be a consequence of not selecting the optimal compiler options for the chosen problem and the computer used.

All time measurements were taken on the same machine: It has an Intel® Core™ i5-3427U CPU with 4 cores running at 1.80 GHz with 7.7 GiB of memory.

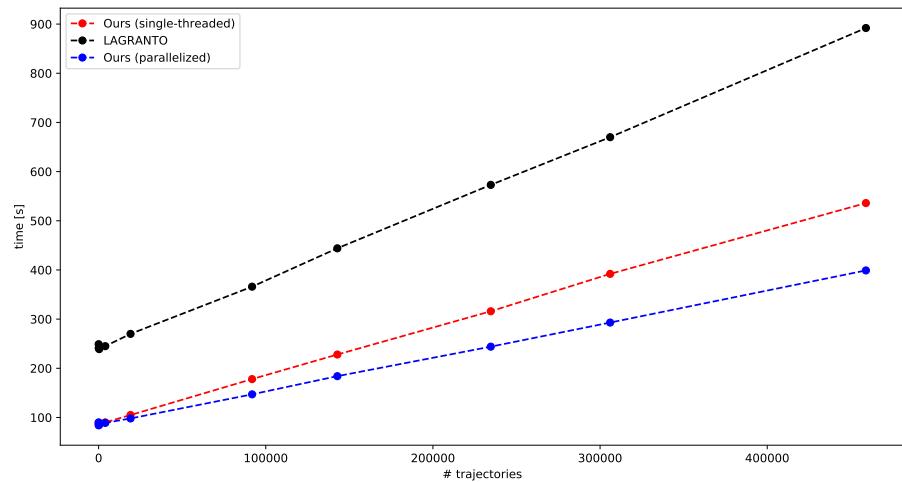


Figure 4.7: Times for tracing a variable number of particles over 60 timesteps and 7 data files

5

Conclusion

The basic functionality of computing trajectories according to input data works well. The visualization and analysis parts ended up rather limited due to time constraints.

When comparing to LAGRANTO, better performance and arguably better results have been achieved. The plots in section 4.2 were not particularly helpful for anything beyond confirming that we can produce the same results. Intuitively, removing a bug from LAGRANTO and using an integrator with lower error bounds should only improve the results. A better analysis would be possible with a test setup using a completely artificial UVW with a known ground truth.

There are some LAGRANTO features that are missing but would be quite simple to add: LAGRANTO uses a file to read the initial points. We let the user define them at the start and also write them into a separate file that can be used for LAGRANTO later. LAGRANTO also includes options like writing down only a subset of the computed points (useful for small timesteps), changing the timestep between files, or writing the output in a different format (for example as a text file). As mentioned in the introduction, LAGRANTO can work with input data that is not in the COSMO format.

Even though we can reproduce the LAGRANTO trajectories almost exactly for the most part, there are still some differences: Points outside the domain are not handled the same way as in LAGRANTO. Another issue is that LAGRANTO uses fixed units for certain variables (e.g. temperature in Celsius) whereas our code simply uses the values from the data files.

While the particle tracer works, it can still be extended. Because the region of interest for this project was relatively small and compact, the solver can not handle trajectories that reach the boundaries of the coordinate system (poles, date line). Some complications already occur at the borders of the local domain. Particles near the ground and outside the $(rlon, rlat)$ -domain are handled differently between LAGRANTO and our code and both methods lead to undesirable results sometimes. The parallelization of the solver is currently very simple and could most likely be improved in more than one way.

5 Conclusion

The visualization with VTK is limited. While it is possible to read trajectories and show how they pass over the landscape, most settings are hardcoded. Displaying different trajectories or changing the variables requires changing the code. Optimally this would have been done using a GUI. Currently all loaded trajectories are displayed at once and there is no way to, for example, select only those that pass through a certain region or those with a certain average temperature.

Although it is possible to obtain trajectories and look at them, we did not look into the non-technical results very much. The search for interesting events would become much easier if the visualization tool was extended first. For the time being, we mostly found common features like wave-shaped trajectories.

Bibliography

- [BSF⁺11] Michael Baldauf, Axel Seifert, Jochen Förstner, Detlev Majewski, Matthias Raschendorfer, and Thorsten Reinhardt. Operational Convective-Scale Numerical Weather Prediction with the COSMO Model: Description and Sensitivities. *Monthly Weather Review*, 139(12):3887–3905, December 2011.
- [Pet56] S. Petterssen. *Weather analysis and forecasting. Volume I, Motion and motion systems*, volume 1. McGraw-Hill, 1956.
- [PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. Cambridge University Press, USA, 1992.
- [SDS⁺03] J. Steppeler, G. Doms, U. Schättler, H. W. Bitzer, A. Gassmann, U. Damrath, and G. Gregoric. Meso-gamma scale forecasts using the nonhydrostatic model lm. *Meteorology and Atmospheric Physics*, 82:75–96, 2003.
- [SML06] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.
- [SW15] M. Sprenger and Heini Wernli. The lagranto lagrangian analysis tool - version 2.0. *Geoscientific Model Development*, 8, 08 2015.
- [WD97] Heini Wernli and Huw C. Davies. A lagrangian-based analysis of extratropical cyclones. i: The method and some applications. *Quarterly Journal of the Royal Meteorological Society*, 123(538):467–489, 1997.