Visualization of Thermally-driven Flow over the Alpine Region

Linue Wigger

Semester Thesis
July 2020

Prof. Dr. Markus Gross, Dr. Tobias Günther

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

This thesis addresses the development of a novel sample thesis. We analyze the requirements of a general template, as it can be used with the LaTeX text processing system. (And so on...) The abstract should not exceed half a page in size!

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung einer neuartigen Beispielausarbeitung. Wir untersuchen die Anforderungen, die sich für eine allgemeine Vorlage ergeben, die innerhalb der LaTeX-Textverarbeitungsumgebung verwendet werden kann. (Und so weiter und so fort...) Die Zusammenfassung sollte nicht länger als eine halbe Textseite sein! Sollte die wirklich auf deutsch und englisch sein? Ausserdem: äöu geht hier

Contents

List of Figures

List of Tables

1

Introduction

A common problem in meteorology is to find wind trajectories with certain properties, e.g. passing through a certain region, carrying particularly warm or humid air, etc. LAGRANTO is an existing tool for computing trajectories from wind velocity fields. We aim to create a similar program in a more modern programming language - C++ as opposed to Fortran. While reconstructing LAGRANTO is our base goal, we would like to get better performance and/or results as well. We succeeded in reproducing the results from LArsionGRANTO. Along the way, a few problems in the LAGRANTO code became apparent, allowing us to avoid thos vee in our.

2

Background

2.1 Data

We work on a set of NetCDF files containing assorted meteorological data. Most of the files contain data at a certain point in time and have names like "lfff00000000.nc". The number in the filename corresponds to the time past the reference time 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) which holds constant variables like the height of the surface.

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 $level$s 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 $level$s to actual height. Further details in section 2.3

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 northward component. All three have the same units and similar grids, but those grids are all staggered in different directions. 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 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 ($rlon$, $rlat$) coordinates. It is mainly used to prevent particles from leaving the domain through the ground.

Name	Description	dimensions	Constant	Staggering	Unit
U	$rlon$ component of velocity	3	no	$rlon$	m/s
V	$rlat$ component of velocity	3	no	$rlat$	m/s
W	vertical component of velocity	3	no	$level$	m/s
HHL	$level$-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*

The pressure P, temperature T and relative humidity $RELHUM$ are not relevant for the tracing, 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 given in staggered grids, recognizable by using $srlon$, $srlat$ and $level1$ for certain axes. The staggered grid coordinates lie halfway between the unstaggered grid points. Destaggering is done by averaging the two values and storing them at the grid position between them. Figure 2.1 shows how staggered ($srlon$, $rlat$) and ($rlon$, $srlat$) grids are converted to ($rlon$, $rlat$). The image also shows that the destaggered version of the grid has one row/column less than the staggered original.

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 given the (global) coordinates of the rotated north pole (λ_{pole}, ϕ_{pole}). Converting coordinates (λ_r, ϕ_r) in the rotated system to the global coordinates (λ_g, ϕ_g) is done as follows:

$$arg = \cos\phi_{pole} \cdot \cos\phi_r \cdot \cos\lambda_r + \sin\phi_{pole} \cdot \sin\phi_r; \tag{2.1}$$

$$\phi_g = \sin^{-1} arg; \tag{2.2}$$

$$c_1 = \sin\phi_{pole} \cdot \cos\lambda_r \cdot \cos\phi_r + \cos\phi_{pole} \cdot \sin\phi_r \tag{2.3}$$

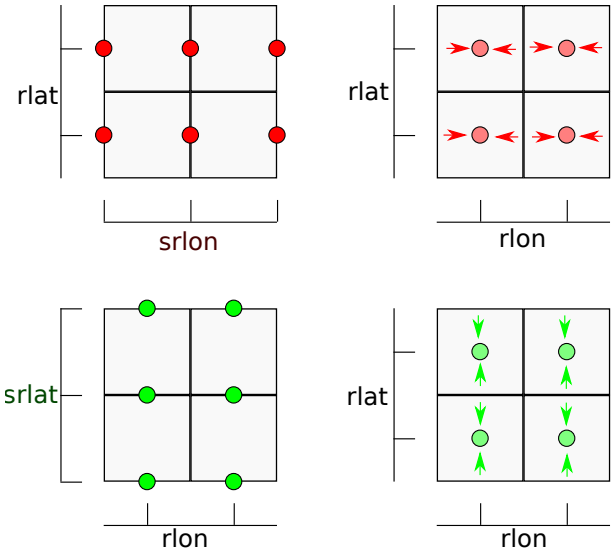

Figure 2.1: *Left: Points in staggered grids; Right: Destaggering by averaging two staggered points*

$$c_2 = \sin\lambda_r \cdot \cos\phi_r \tag{2.4}$$

$$zarg1 = \sin\lambda_{pole} \cdot c_1 - \cos\lambda_{pole} \cdot c_2 \tag{2.5}$$

$$zarg2 = \cos\lambda_{pole} \cdot c_1 + \cos\lambda_{pole} \cdot c_2 \tag{2.6}$$

$$\lambda_g = atan2(zarg1, zarg2) \tag{2.7}$$

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.

3

Method

3.1 Numeric integration

Solving differential equations of all types is a topic for itself, so here we have just the methods used to find the next point of a trajectory given the velocity field $UVW(p,t)$, the starting position p_{t_0} and a timestep of size δ_t.

LAGRANTO uses an iterative variant of Euler's method. The next point $p_{t_0}+\delta_t$ is computed using the average of the velocities at the original point p_{t_0} and the current guess for $p_{t_0}+\delta_t$. This method is related to the explicit trapezoidal rule: In the time-invariant case, stopping at q_2 is equivalent to using the explicit trapezoidal rule.

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 error should be smaller for the same timestep.

Iterative Euler

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

$$v_1 = UVW(p_{t_0}, t_0+\delta_t) \tag{3.2}$$

$$q_1 = p_{t_0} + \delta_t \frac{v_0+v1}{2} \tag{3.3}$$

$$v_2 = UVW(q_1, t_0+\delta_t) \tag{3.4}$$

$$q_2 = p_{t_0} + \delta_t \frac{v_0+v2}{2} \tag{3.5}$$

$$v_3 = UVW(q_2, t_0+\delta_t) \tag{3.6}$$

$$p_{t_0+\delta_t} = p_{t_0} + \delta_t \frac{v_0+v3}{2} \tag{3.7}$$

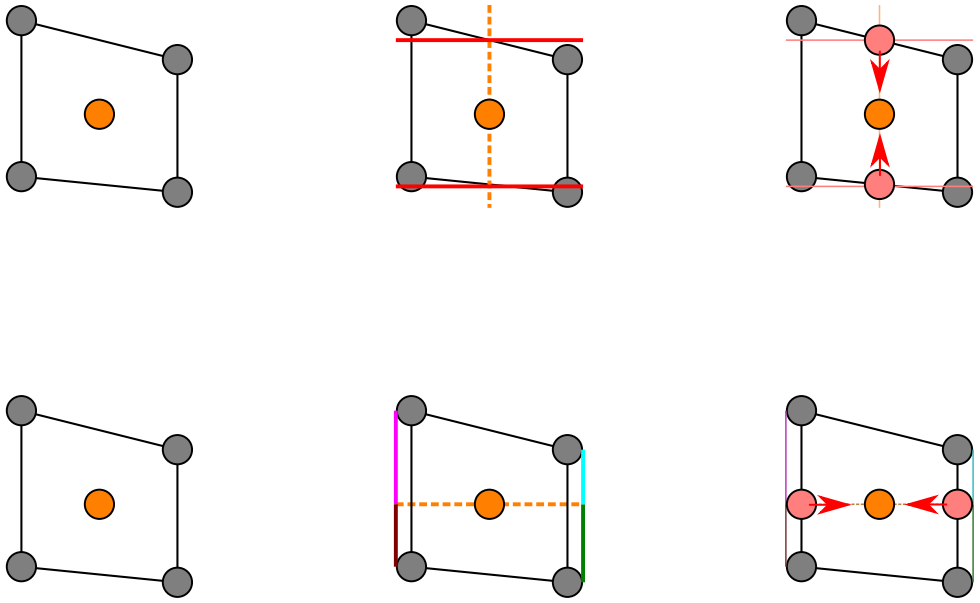

Figure 3.1: *Top: How LAGRANTO interpolates between levels; Bottom: How we do it TODO change it*

Classical Runge-Kutta

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

$$k_2 = UVW(p_{t_0} + k_1\frac{\delta_t}{2}, t_0 + \frac{\delta_t}{2}) \tag{3.9}$$

$$k_3 = UVW(p_{t_0} + k_2\frac{\delta_t}{2}, t_0 + \frac{\delta_t}{2}) \tag{3.10}$$

$$k_4 = UVW(p_{t_0} + k_3\delta_t, t_0 + \delta_t) \tag{3.11}$$

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

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 ($\deg rlon$, $\deg rlat$, m) some conversions are necessary.

3.2.1 How LAGRANTO does it

Mapping x and y to positions in the $rlon$-$rlat$-grid is simple enough. They are already in the right units and so the closest grid points can be found TODO(formula? picture?) The upper part of Figure 3.1 shows how LAGRANTO interpolates between levels: First two level heights for the upper and lower level are constructed (shown as red horizontal 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

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.

3.2.2 How I do it (TODO change subtitle)

Finding the grid coordinates of (x,y) and the bilinear interpolation weights along the $rlon$ and $rlat$ axes is the same as in the previous subsection.The lower part of figure 3.1 shows how we compute the interpolated value at the orange sample point. On each of the four (two in the picture) columns, we compute interpolation weights after a binary search on the equivalent column in HHL. The last step is bilinearly interpolating between those four values.

3.3 What happens when running

The tracing process starts by asking the user for initial points, start and end time, size of the timestep, and some optional parameters. 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 δ_t. Those that have left the domain are kept at their last positions while the others get positions for the next timestep based on the velocity at their current position.

3.3.1 Tracing output

The results from the particle tracing are written into a NetCDF file which stores: Time, position, extra variables. At each vertex. Each variable is stored in a separate array, time is the "big step", trajectory the "small step". That is how LAGRANTO does it.

3.4 Analysis

Qualitative: We visualize the trajectories using VTK (TODO reference). 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 HHL is also displayed to give a context beyond just the trajectory shape.

$rlon$ and $rlat$ 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 coordinates on the vertical axis z are rescaled by a factor of TODO to bring them more in line with the rest. Overall, the shapes in the picture are not entirely accurate, but they give a good idea of how things look like.

Quantitative: Given two trajectories, we want to measure how different they are by integrating the distance between their current positions over the tracing duration. Assuming two point sets (p_0, p_1, p_2, ..., p_N) and (q_0, q_1, q_2, ..., q_N), the distances will be ($|p_0 - q_0|$,$|p_1 - q_1|$,$|p_2 - q_2|$, ...,$|p_N - q_N|$). For approximating the integrated distance, we need times t_i associated with each point in addition to the spatial coordinates. The final trajectory distance is computed as $\Sigma_{i=0}^{N} w_i \cdot |p_i - q_i|$, where w_i is usually equal to the timestep δ_t (assuming a constant timestep). For the initial points p_0,q_0 as well as the final points p_N,q_N it is $\frac{\delta_t}{2}$ instead. More generally, w_i is equal to $\frac{t_{i+1} - t_{i-1}}{2}$ (assume $t_{-1} = t_0$ and $t_{N+1} = t_N$).

4

Results

4.1 Pictures and stuff

4.2 Comparison

Comparing results: I can reproduce LAGRANTO pretty well ... and which improvements do more or less?

4.3 Performance

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

First experiment: 4 trajectories, 7 input files, 60 iterations. Most of the work is reading the files. Second experiment: 459045 trajectories, 7 input files and 60 iterations again. This involves more computations.

LAGRANTO times are: 4 m 12 s for the first, 19 m 08 s for the second. My times are: 1 m 37 s for the first, 4 m 35 s for the second.