Numerical Techniques 2022–2023

Student projects

Daan Degrauwe

daan.degrauwe@meteo.be

Postgraduate Studies in Weather and Climate Modeling

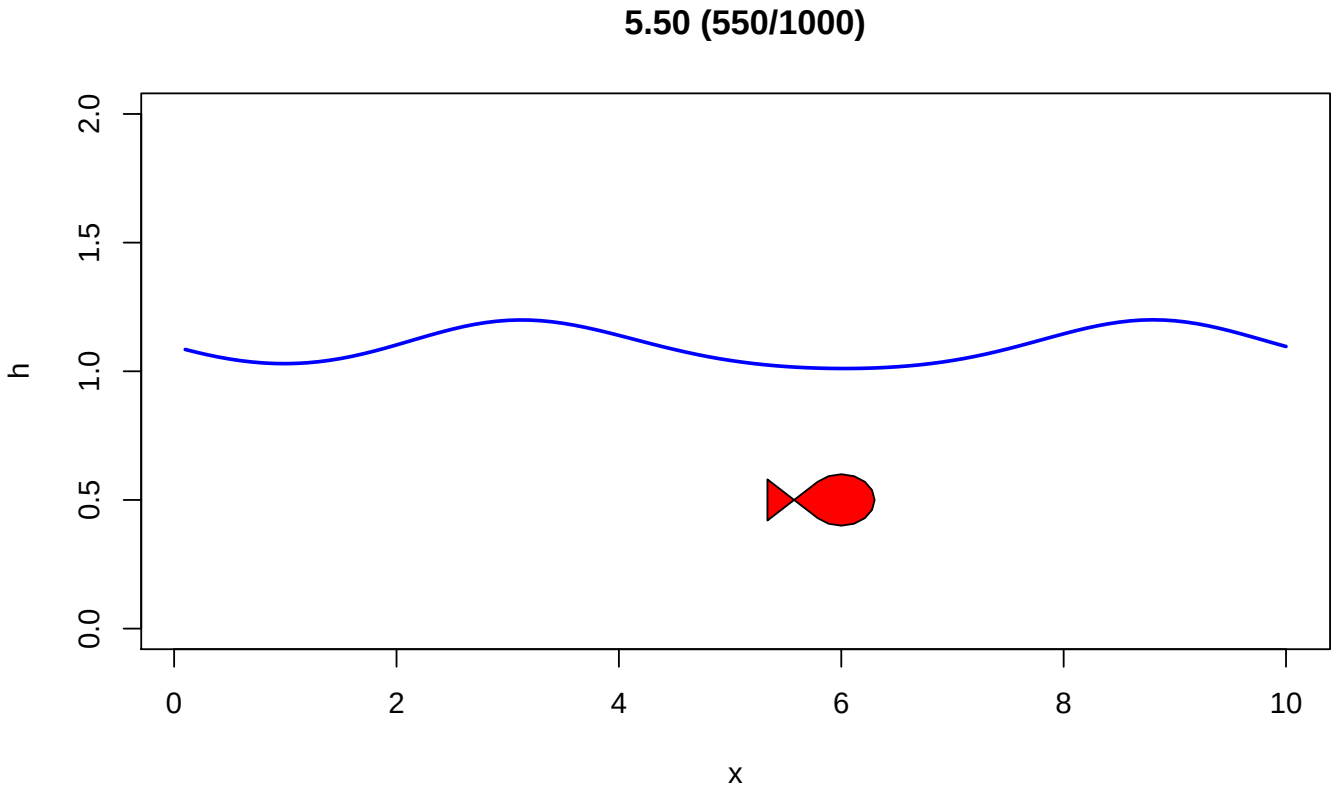
Ghent University

Starting from a simple model, apply some of the techniques from this course.

General remarks:

- no course on Fortran or Linux; no course on complex algebra; ask for help!
- it's allowed to use Matlab, Python, R, etc. if you prefer this instead of Fortran;
- open assignment: no exact solution + discussion is more important than results
- the purpose is you *learn* something
- try to trigger strange/unwanted phenomena, and discuss solutions.
- you can propose a topic yourself.

Shallow Water Equations



The linearized 1D shallow water equations (SWE) are given by:

$$\frac{\partial u}{\partial t} + U\frac{\partial u}{\partial x} + g\frac{\partial h}{\partial x} = 0$$
$$\frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} + U\frac{\partial h}{\partial x} = 0$$

where g is gravity, U and H are the constant basic-state velocity and water height, and $u(x,t)$ and $h(x,t)$ are the perturbations on the velocity and the water height.

The current model is very simple:

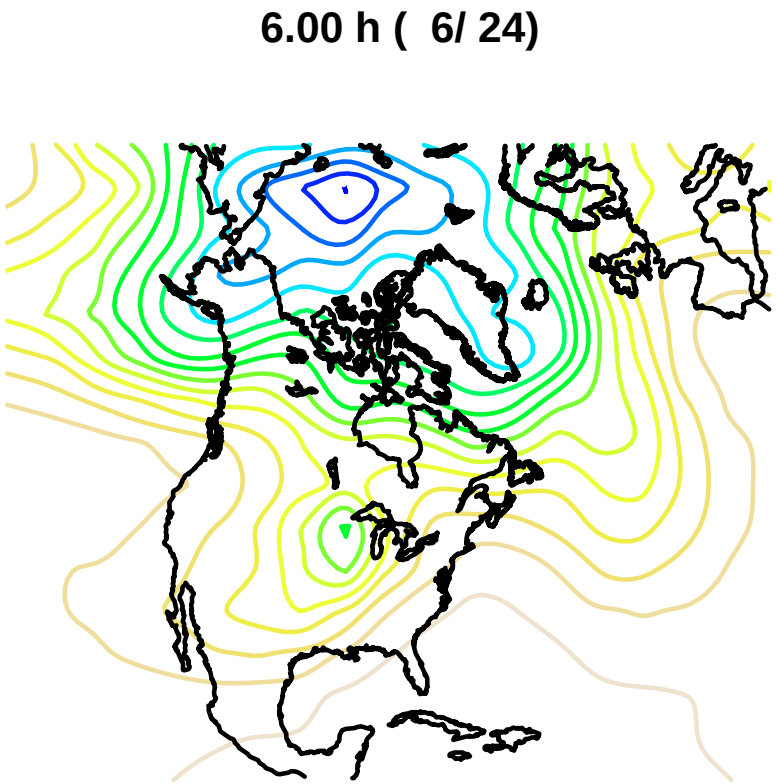
- Leapfrog time integration
- Second-order centered space differencing
- Periodic boundary conditions

Student projects:

1. Stagger u and h
2. Compare with nonlinear model
3. Transform into spectral model with Fourier decomposition
4. Transform into spectral model with Chebyshev decomposition

   J.P. Boyd, *Chebyshev and Fourier Spectral Methods*
5. Study Laplace transform integration

   Clancy and Lynch, 2011, QJRMS 137,
   Laplace transform integration of the shallow-water equations


6.00 h (6/ 24)

- The barotropic vorticity equation BVE is given by:

$$\frac{\partial \zeta}{\partial t} + \mathbf{u} \cdot \nabla \zeta = 0$$

where ζ is the vorticity, and $\mathbf{u}$ is the geostrophic wind, given by

$$\mathbf{u} = \mathbf{k} \times \nabla \psi \qquad\qquad \nabla^2 \psi = \zeta$$

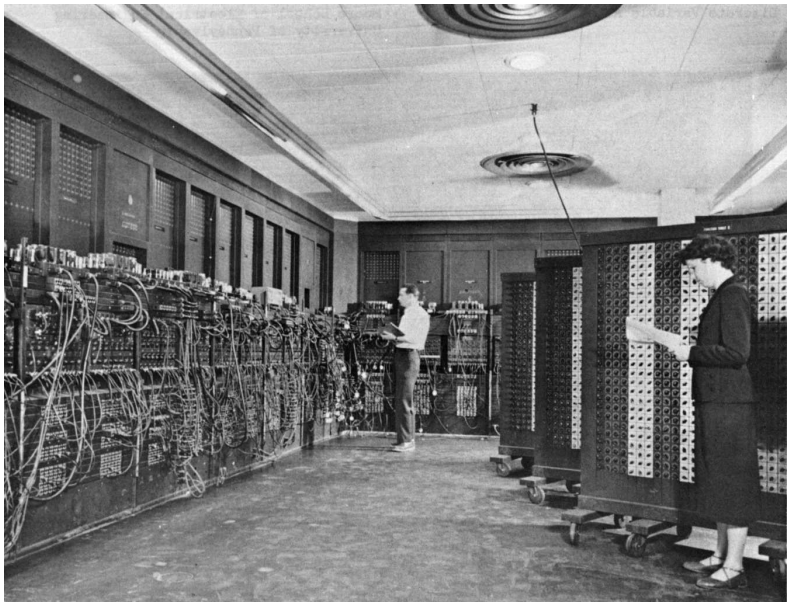
- Writing everything in terms of the streamfunction ψ, the BVE becomes

$$\frac{\partial \nabla^2 \psi}{\partial t} + J(\psi, \nabla^2 \psi) = 0$$

where J is the Jacobian operator:

$$J(p, q) = \frac{\partial p}{\partial x}\frac{\partial q}{\partial y} - \frac{\partial p}{\partial y}\frac{\partial q}{\partial x}$$

- This model was used for the first numerical weather prediction in 1950 on the ENIAC computer



- A 24-hour forecast took about 24h computing time

Original model

The original (1950) model is characterized by:

- a leapfrog time integration
- second-order centered space differences
- an expensive (pseudo-spectral) way to invert the Laplacian operator
- heuristic boundary conditions:
  - $\frac{\partial \psi}{\partial t} = 0$ on boundary
  - $\frac{\partial \nabla^2 \psi}{\partial t} = 0$ for entering fluid

Due to these boundary conditions and aliasing, the model is not stable.

The student's model is somewhat simplified:

- Spectral inversion of Laplacian
- Periodic boundary conditions
- Coriolis effect removed
- Projection impact removed

Improvements (student projects)

General remark: lots of interesting aspects in this model, but you'll have to dig deeper to find them.

Student projects:

6. Semi-Lagrangian scheme
7. High-resolution LAM nested in low-resolution LAM (coupling with Davies relaxation)
8. Spectral model and avoiding aliasing
9. Check energy cascade between large and small scales, and implement Arakawa Jacobian

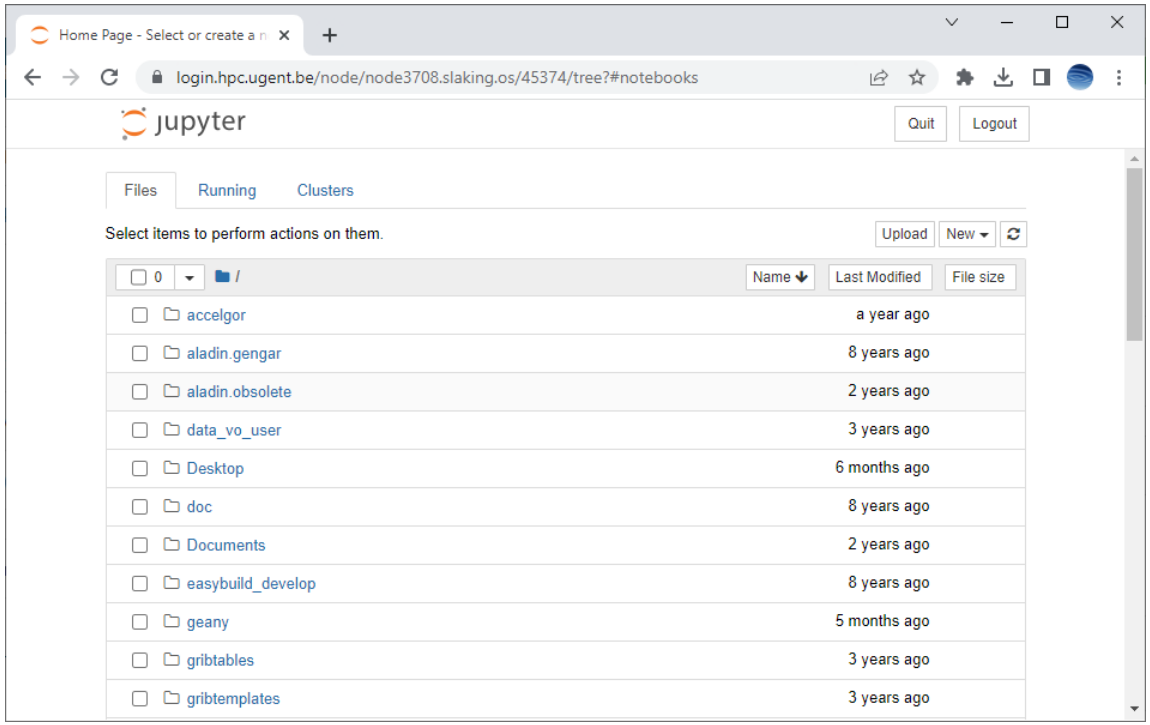
Practical aspects

- Groups of 2/3 persons
- Pick a single topic (e.g. SWE-spectral)
- More detailed background info (papers) on Ufora.
- Support sessions: 18 April, 14h30; come prepared!
- Report (say 5–10 pp.): deadline Sunday 24 April.
- Presentation for other students: Tuesday 25 April.

- Instead of working under Linux (previous years), we'll use a Jupyter notebook. Hopefully this is more student-friendly.
- However, the actual model is still written in Fortran, but compiling, launching and reviewing results will be done from the Jupyter notebook.
- To get started, take the following steps:
  1. login on the UGENT HPC infrastructure `https://login.hpc.ugent.be/`
  2. launch a Jupyter IPython Notebook (under Interactive Apps)
  3. **Make sure to use the following settings**:
     - ★ Set Cluster to 'slaking'
     - ★ Choose an appropriate value for the Time (you will be kicked off the system after this time)
     - ★ Set IPython version to '7.15.0 foss 2020a Python 3.8.2'
  4. Hit the 'Launch' button at the bottom
  5. After a few minutes, you should be able to connect to your Jupyter job (under My Interactive Sessions)

Getting started with the Jupyter notebook

- The previous steps should bring you to an interface like this



- This is not yet the actual notebook, but it allows you to easily navigate between files/folders, and to create new notebooks, folders or files.

Getting started with the Jupyter notebook

- Since file/folder manipulation is easier in a Linux terminal window, launch a terminal (under New → Terminal).

- Create a new folder and copy the necessary files with

  ```
  mkdir project_numtech
  cp -r /user/gent/407/vsc40744/ugent/projects/swe1D project_numtech/
  ```

  (replace `swe1D` with `barovort` if needed)
  You can close the terminal window now.

- Back in the Jupyter launch window, navigate to the folder you just created. When clicking a Fortran file, an editor will open so you can modify it.

- Finally, click on the file `swe1D.ipynb`. This will open the notebook where you can interactively launch experiments and review the results.

Getting started with the Jupyter notebook

- A notebook is organized in 'Cells' containing python code. The code in a cell is evaluated when hitting the Run button or pressing Ctrl-Enter.
- The SWE notebook contains 4 cells:
  - A cell to load the necessary functions from the module in the python file `swe1D_fun.py`
  - A cell to launch the model and retrieve the results (more on this later)
  - A cell to animate the water height in time
  - A cell to plot the water height at the last timestep
- What actually happens in the second cell is the following:
  1. the function `run_program` from is called from the `swe1D_fun` module.
  2. this function in turn launches a Linux script `run.sh`
  3. this script in turn performs the compilation with `gfortran` and runs the program
  4. the program generates an output file `output.dat`
  5. the function `run_program` reads the data from this output file and transforms it into numpy arrays, which can be easily plotted.

  While this looks quite combersome, *in principle* it shouldn't be modified. Only the Fortran code itself and the postprocessing in Jupyter need modifications.

  Should something look strange or unexpected (and it will!) don't hesitate to contact me!