Assignment 5 (week8)

Deadline: Nov 1st, 23:59

Important: The scripts of this assignment should be saved in a subdirectory named `week8` of your INF3331 github repository. Points will be given on correctness of the solutions, the documentation and the code readibility.

5.0 Introduction

In exercises 5.1 - 5.5 we will implement a model for simulating the dissipation of heat within a material in a given domain over time. For simplicity, we assume that the simulation domain is a 2-dimensional rectangle.

The heat dissipation in a material are described by a partial differential equation, the heat equation:

$$\frac{\partial u}{\partial t} - \nu \Delta u = f \tag{1}$$

Here, ν (`nu`) is the (real-valued) material-specific thermal diffusivity, u is the space- and time-dependent temperature within the material, and f is a heat source. The Laplace operator is defined as $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, and $\frac{\partial}{\partial}$ denotes the partial derivative. Two snapshots of the solution of heat equation at different times are shown in figure 1.

5.1 Python implementation of the heat equation (8 points)

Implement a solver for the heat equation 1 in Python. This implementation should only use Python objects, such as lists, tuples and dictionaries, but not numpy arrays (numpy and C versions will be implemented in the next exercise).

For a rectangle of size $n \times m$, the temperature at any given time is stored as a two-dimensional list `u` of shape (n, m). The value of `u[i][j]` shall represent the temperature at the coordinates i, j.

The model starts with a user-specified, initial temperature distribution at time $t0$. At any time t during the simulation, the model performs an update step to proceed to time $t + dt$, with a user-defined timestep dt. The update formula is:

```
u_new[i][j] = u[i][j]
            + dt*(nu*u[i-1][j] + nu*u[i][j-1] - 4*nu*u[i][j]
            + nu*u[i][j+1] + nu*u[i+1][j] + f[i][j])
```

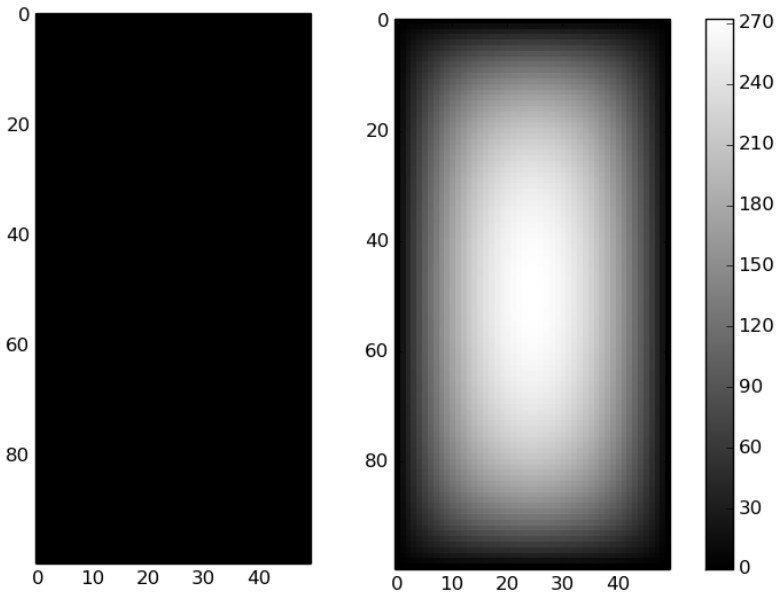

Figure 1: The temperature at time 0.0 (left) and 1000.0 (right).

This update step is repeated until the end of simulation time is reached. We assume that the temperature values at the boundaries are fixed during time - that is the update step should not update the boundary values.

Your implementation should be parametrised with the start time `t0`, the end time `t1`, the time step `dt`, the rectangle dimensions `n, m`, the initial temperature distribution `u` (an $n \times m$ list), a heat source function `f` (also a $n \times m$ list) and the thermal diffusivity `nu` (a material specific parameter, a float).

Run the simulation on a 50×100 large rectangle, starting with $u = 0$ at $t0 = 0$ until $t1 = 1000$ with a timestep $dt = 0.1$, $\nu = 1$ (i.e. `nu=1`) and f set to 1 everywhere. Plot the results using matplotlib (for example with `pyplot.imshow` and `pyplot.colorbar` or `scitools` on UiO computers) and compare them with figure 1.

File: `heat_equation.py`

5.2 NumPy and C implementations (8 points)

Re-implement exercise 5.1 in two variations: one using numpy arrays and one based on a mixed C/Python implementation. The numpy solution should make use of vectorisation and avoid copying of arrays if possible. For the mixed implementation, only the numerical part (the update step) needs to be implemented in C, so that the model can still benefit from the plotting tools in Python. You may choose which strategy to use for integrating the C code in Python (we recommend `weave`)

Files: `heat_equation_numpy.py` and `heat_equation_X.py` (replace X with the tool you used for the C solution)

5.3 Testing (4 points)

Demonstrate that your implementation is mathematically correct by solving the heat equation with following heat source term:

```
f[i][j] = nu*((2*pi/(m-1))**2 + (2*pi/(n-1))**2) \
          *sin(2*pi/(n-1)*i)*sin(2*pi/(m-1)*j)
```

Here the i index goes from 0 until n-1, and the j index from 0 until m-1. For large simulation times, the solution of the heat equation converges to the analytical solution

```
analytic_u[i][j] = sin(2*pi/(n-1)*i)*sin(2*pi/(m-1)*j)
```

Use the remaining parameters (rectangle size, initial u, t0, t1, dt and ν) from in 5.1, but with the new heat source and compute the difference between final solution time and the analytical solution. Then print the component of the difference with the maximum absolute value:

```
err = (abs(u - analytic_u)).max()
```

This error should be below 0.0012. Check that the error decreases if you increase the size of the rectangle (you might have to increase the final time for this test).

5.4 Develop a user interface (4 points)

Implement a *common user interface* for your Python solvers as well as for the C-solution using the `argparse` module. This user-interface should offer at least the following command line options:

- Option to specify the model parameters such as rectangle dimensions, start-time, end-time, timestep, thermal diffusivity coefficient and constant heat source;
- Name of an output file in which the program stores the solution at the final time, or an input file which is loaded and used as the initial temperature[1];
- Option to save a plot of final temperature as an image;
- Option to switch between the three implementations (pure Python, Numpy, C);
- Option to activate verbose mode, which prints what the program does.
- A switch for turning on the `timeit` module, which then reports important timings at the end of the program.

Give the options useful name and description/help messages.

Files: `heat_equation_ui.py`

5.5 Write a Latex report (6 points)

Describe what you have done in a Latex report. In addition, add a runtime comparison for the different implementations and explain the differences.

File: `report.tex` and `report.pdf`

[1] Use for example the 'pickle' module. The input/output options can be used to checkpoint/restart the simulation, e.g. on supercomputers.

5.6 (INF4331 students only) More interfaces (10 points)

Implement 2 more implementations of the heat equation solver using different C/C++ integration tools (of your choice, for example weave, swig, or cython). Together with 5.2 you will have 3 mixed C implementations.

Files: 2 x `heat_equation_X.py` (replace X with the tool you used for the C solution)

5.7 Github activity plot (10 bonus points :)

Write a Python script that takes a path to a github directory. The script inspects the timestamps of all commits in that repository and generates a histogramm plot that visualises the activity of the repository over time[2].

The script should:

1) Check that the given filepath is provided and valid and that it is a github repository (checking e.g. the exit code of the command `git rev-parse --git-dir`). If the input does not exist or is invalid print a useful error message. Use `argparse` for handling the command lines options.
2) Write a function that takes in the git directory as a parameter. The function then executes `git log` and extracts the authors and dates for all commits. Use a regular rexpression to obtain these information and return the results.
3) Use matplotlib or scitools to visualise the commit activity of that repository over time. The exact type of plot is up to you. The different authors should also be visible in this plot, for example by using different colors and a legend. the plot as a pdf file.
4) Add command line options for saving the plot as a PDF or an image.

File: `github_activity.py`

Total points (INF3331): 30 + 10 bonus points

Total points (INF4331): 40 + 10 bonus points

[2]Similar to https://github.com/OpenTidalFarm/OpenTidalFarm/graphs/contributors