INF3331 - Assignment 5

Ina Vangen, ivvangen

November 1, 2015

5.5 Write a Latex report

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

How the code works

0. Setup

Operating system: Ubuntu 14.04 LTS
Python version: 2.7.6

Packages used: pickle, argparse, timeit, time, math, numpy, matplotlib, pyximport, os, re, random

Important: There was some troubles testing the script on the university-computers because matplotlib and cython were not installed on them. In order to make sure the code works, make sure all the packages above are installed and that you are using a compatible python version.

1. Introduction

The script is divided into several python (.py) -files. The heat_equation_ui.py is the file containing the argsparser (takes arguments from command line) and is the file that you are supposed to run to start the heat equation. You will find a short version on how to run it in the README.txt. Below follows more detailed section on how to run the script and how the core functions work.

In section 6-8, there will be a brief explanation on the different heat equation implemention as well as computation times. All computation times in the following section were done with the same values (default as in assignment), and we run on the from the same computer. The goal is to demonstrate the time differences of each implementation.

2. Running the script

To run script write: python *heat_equation_ui.py* in the command line.

The *heat_equation_ui.py* uses argsparse to let the user specify how the script should behave. The user are able to speficy the dimensions, start time, end time, timestep, thermal difusity, heat source, type of heat equation, as well as choosing whenether verbose, timit -mode should be on, or if the data should be read, written or saved as an image. All these values are set to a default value and doesn't need to be specified unless needed.

The most important part is to selec which type of heat equation (see section below detail on how each implementation works). To see how to specify the script using arpase, simply add the $--help$ command when running the script.

3. Heat equation

The script sets all variables to the desired values and then calls the seleced type (using argparse). For example will *heat_equation_numpy*() be called if the numpy type were choosen, and *heat_equation_cython*() if the cython type were choosen. This function return a list of data (*data_list*).

4. Plotting

The *plot_data*()-function what is found in *plotting.py* is based on matplotlib's pyplot and divides the section into several subplots within a grid using matplotlib's gridspec. *plot_data*() takes two list as a parameter. Left plot of a list that contains the initial values (*data_list_init*), and right plot of a list that contain the calculated data (*data_list*).

5. Testing

The user can specify if a test should run after the heat equation is performed. Testing is turned on by default. To turn off, add $-test$ in the command line when running the *heat_equation_ui.py*-script.

This function is to demonstrate that the implementation is mathematically correct. The *testing*()-function can be found in the *test_heat.py*-file. The script calculates a new heat source (f) using the *get_heat*()-function and an analytic_u using the *get_analytic*()-function. It then performs a *heat_equation_numpy*() using the new parameters (any heat equation type could have been choosen) and then compares the returned value (u) with the analytic_u. This is to see if there is an error value in it.

6. Python implementation

This implementation initalizes the necessary lists and uses a double for-loop to iterate over each object in the list. In each interation the equation is performed for each element in the 2-dimentional array. This is a very slow implementation (reasons follow in next sections).

Average computation time: 182.160234213 sec

- 185.69433784484863 sec
- 181.72458600997925 sec
- 181.0921471118927 sec
- 181.49347305297852 sec
- 180.79662704467773 sec

7. Numpy implementation

Numpy uses arrays of the same type as well as using vectorization and is therefore faster than the normal python implementation where the type is objects (which requires heavies computation time). Numpy also avoids a double for-loop which is consideres a very slow method to iterate through listsarrays

Average computation time: 12.9623294353 sec

- 13.0695321559906 sec
- 12.936105012893677 sec
- 12.955157995223999 sec
- 12.946208000183105 sec
- 12.904644012451172 sec

8. Cython implementation

Cython allows to write code that is similar to Python but translates it into C-code. If done correctly (optimized) this can be faster than numpy. This is likely because of some overhead in numpy. C code in general is faster than Python bacause it not interpreted which Python is. C also dont use obejct, but rather types (as with numpy) as well as having a better memory management.

Average computation time: 2.15159683228 sec

- 2.0980420112609863 sec
- 2.0902230739593506 sec
- 2.0964248180389404 sec
- 2.326665163040161 sec
- 2.1466290950775146 sec