Report

Thomas Fossøy Lyseggen, Git-repo INF3331-ThomasFossoy, thomaslyseggen@gmail.com

November 2, 2015

5.1 Python implementation of the heat equation

Well there is not really that much to say here. I have a main method that takes in a lot of parameters. These include all the variables that is needed to compute, and a few more. The extra ones are for are variables from the UI. Plot, verbose, test and save.

- plot - plots the data in *using pyplot.imshow()*.
- verbose - writes out what the program does.
- test - runs a comparison to the analytical approach. Need verbose as well to print the error.
- save - whether to save the plot or not.

Also have a heat source function that simply calculates the data. At last i have an analytic function that calculates the analytical data for comparison. This wil only be run if –test it given when running

```
python heat_equation.py
```

I have assumed that you should only run this, and all the other implementations through heat.py. If you need to run this alone, you will have to remove the

```
if __name__ == "__main__":
    print "use 'python heat_equation_ui.py' instead"
```

and add a

```
main()
```

at the end.

The evolvefunction computes the data. It iterates through the nested list(u), giving it values. It does this while counting from t0 to t1 with dt as timestep, and updates u. It's not much more to it, as the formula is given. If f is an int, it creates an array filled with that value. Could have just added the constant at the end, but at this point I do not have the time.

5.2 NumPy and C implementations

NumPy

Let's start with the numpy implementation. There is not really that much of a difference, other than it being numpyarrays instead of nested lists, and the performance, of course. It has the same extra parameters as the previous, and has the same meaing. Must run from heat.py or follow the instructions above.

The main function now only counts from t0 to t1 with dt as timestep. Using numpy arrays and vectorization I dont need the double for loops.

```
u_new[1:-1,1:-1]
```

This means essentially every element in the array except the first and the last.

C

Now for the c implementation. Unfortunately this is not all correct, and i have not been able to find the error. But at least it is faster, and the plot looks ok. I'm using weaves inline function to run it in c. This might not be the fastest option, but at least i made it work. To a degree at least.

It does everything the same as the numpy solution, except for the iterating of the data. This is done in c. Could have used weave to calculate analytic and heat source as well, but seem like a necessity. Also needs to be run from *heat.py*, and has the same extra parameters as the previous implementations.

All three implementations has the plot options. Just run:

```
python heat_equation.py -h
python heat_equation.py 50 100 0 1000 0.1 1 1 --test --plot -c
```

this will give you all the info you need. *–test* will calculate against the given heatformula.

5.3 Testing

The test is ran by following:

```
python heat_equation.py 50 100 0 1000 0.1 1 1 --test  -c
python heat_equation.py 50 100 0 1000 0.1 1 1 --test  -np
python heat_equation.py 50 100 0 1000 0.1 1 1 --test
```

for more information and options, just do:

```
python heat_equation.py -h
```

As you can see, the error is less than expected for the numpy approach. I did say that the error is too big in the C implementation, and it is. The normal implementation did pretty bad as well, altought I'm not sure if I should have tested that one.

5.4 Develop a user interface

The user interface. This one might look a little messy. And it is. The wrapper function is just there so I can use the timeit function for checking performance.

runis just a function for sending the arguments to the numpy implementations, and also checkes for a few things:

- time - if time is given as an option, I have to disable verbose and plot.
- -o - if this is given the program should save the resulting array in the given file.
- -i - if this is given the program should get the data from the file given.

This is the same for every implementation. running:

```
python heat_equation.py -h
```

gives us:

```
usage: heat_equation_ui.py [-h] [-n [n]] [-m [m]] [-t0 [t0]] [-t1 [t1]]
                           [-dt [dt]] [-nu [nu]] [-f [f]] [-t] [-np] [-c] [-p]
                           [-v] [-time] [-s] [-i IFILE] [-o OFILE]

Calculate the heat equation

optional arguments:
  -h, --help       show this help message and exit
  -n [n]           rectangle size x (default=50)
  -m [m]           rectangle size y (default=100)
  -t0 [t0]         Calcutation start time (default=0)
  -t1 [t1]         Calcutation stop time (default=1000)
  -dt [dt]         Calculation timestep (default=0.1)
  -nu [nu]         Calculation diffusion coefficient (default=1)
  -f [f]           Calculation heat source (default=1)
  -t, --test       Calculates a test against an analytic solution with a given
                   heat_source formula.
  -np, --numpy     numpy solution (default is very slow)
  -c, --c          c solution (default is very slow)
  -p, --plot       Plot (default=False)
  -v, --verbose    prints out what the program does (default=False)
  -time, --timeit  Uses timeit module to print the time used. Plotting wil be
                   disabled.
  -s, --save       Save the plot as a png(tmp_x.png). x being the solution
  -i IFILE         input file with data
  -o OFILE         output file with the final solution
```

Which is pretty self-explanatory. Since we're supposed to:

```
Option to specify the model parameters such as rectangle dimensions,
start-time, end-time, timestep, thermal diffusivity coefficient and constant
heat source.
```

I've done it like this. To give other dimension you do the following:

```
python heat_equation.py -n 100 -m 150
```

Because of lack of time, I had to implement a pretty bad 'read data from file' method. I have assumed that the file has to have all the data, and should have ',' as a delimiter. It is easy to crash though. I can now also see that this could have been done a lot easier with only one *run* function, and just done something like i did with the wrapper. But no time to implement that either.

5.7 Github activity plot

This task works,but its not great. The module is not very robust, nor a well written programme. But on my computer it did what is should do, and i tested it on ifi computers as well.

Running:

```
python github_activity.py -h
```

returns:

```
usage: github_activity.py [-h] [-s IFILE] f

Check git activity in given repo

positional arguments:
  f                     the gir-repo

optional arguments:
  -h, --help            show this help message and exit
  -s IFILE, --save IFILE
                        save the plot to the given file
```

Which is pretty self-explanatory.

I should also have given the plot a fixed size, because if you get it to plot, it looks a bit crammed. It also saves the plot to the git directory, not the working directory. I use datetime module to format the time, i a 'simple' regexpattern to find my authors and dates. I iterate and create a authordict with a list as value. This list contains the number of commits on different dates. It is sorted by index. so on the days an author did not commit, the index is 0. Uses a barplot to plot the activity.

Was allowed to commit Monday before 23:59, thats why its late.