Peer-review of assignment 5 for *INF3331-Ina*

Thomas Oddsund, Git-repo INF3331-ThomasStenberg, thomaco@student.matnat.uio.no
Herman Loennechen, Git-repo INF3331-Herman, hermanlo@student.matnat.uio.no
Andre Kaasen, Git-repo INF3331-Andre, andrekaa@student.matnat.uio.no

Deadline: Tuesday, 10. November 2015, 23:59:59.

1 Feedback

The program is well structured and organized in an understandable fashion. All the variables and functions have sensible names. Below follows some simple suggestions to improvements.

2 Review

Python 2.7.6 on a Linux Mint 17.2 Rafaela was used for reviewing the code.

Assignment 5.1: Python implementation of the heat equation

The python implementation works fine. Placing the double for-loop in an additional function wouldn't hurt and the line break in the update formula is a must, like this:

```
def calculate():
        for i in range(1, m-1):
                for j in range(1, n-1):
                        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])

while (t0 < t1):
        calculate()
        if verbose: print "\rTimestep: {}".format(t0),
        t0 += dt
        u = u_new
```

Assignment 5.2: NumPy and C implementations

NumPy

The same calculate function as in Python implementation above might be a thing for the NumPy implementation as well. That being said, both the NumPy and the Python implmentation looks very similar, so you might want to make some kind of class hierarchy as suggested in the lectures, which will involve addition of class signatures to the code e.g.:

```
class Python_solver(object):

# Some code ...

        def calculate():
                for i in range(1, m-1):
                        for j in range(1, n-1):
                                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])

        while (t0 < t1):
                calculate()
                if verbose: print "\rTimestep: {}".format(t0),
                t0 += dt
                u = u_new

# More code, maybe ...
```

Cython (C)

The Cython implementation is seriously neat. Well done.

Assignment 5.3: Testing

Your test seems to work fine, however, you don't, as far as we can see and the assignment ask for, extend you testing to a bigger material i.e. you do not check whether the "err" value **decreases** if the size of the material **increases**. When running the Cython implementation with n=100, m=200 and t1=2000, we get exacly the results the assignment text ask for. The point is that the assignment text ask you explicitly to check results for a bigger rectangle and since the program runs testing as default it would be natural to test both "regular" rectangle as well as the increased one.

Assignment 5.4: Develop a user interface

The ui is clear and orderly. The only thing syntax-wise would be line breaks from

```
parser.add_argument('-type', choices=['python', 'numpy', 'c'], default='c', help="Choose heat equation mode")
```

to somthing like:

```
parser.add_argument('-type', choices=['python', 'numpy', 'c'], default='c',
        help="Choose heat equation mode")
```

But this is rather trivial, isn't it? Something that might be (a little more) worth mentioning is that since the program runs Cython and testing on default you don't really need to list Cython as an option in the ui i.e. a user have only two options. Furthermore, since testing is on by default the test flag:

```
parser.add_argument('-test', action="store_true", default=False,
        help="Turn on test-mode")
```

do not turn on test-mode but:

```
parser.add_argument('-test', action="store_true", default=False,
        help="Turn OFF test-mode")
```

This is stated in the report, so it is not of great importance.

Assignment 5.5: Write a Latex report

The report is thorough and gives a good overview of the program, especially of the performance of the different implementations. There is reflections conserning different runtimes and use of data structures, and good descriptions of how the program is structured.

Assignment 5.7: Github activity plot

The activity plot is implemented well. It has all the required features, the division of labour between functions is intuitive and it outputs a nice diagram.

3 Advice and comments

You have a lot of attribute values that are passed to instantiations of functions/objects like this:

```
data_list = heat_equation_numpy(t0, t1, dt, n, m, nu,
                        data_list_init, f, verbose)
```

And one huge return statement in the argparser function:

```
return args.n, args.m, args.t0, args.t1, args.dt, args.nu, args.hs, \
        args.v, args.t, args.img, args.input, args.output, args.type, args.test
```

The first mentioned kinds of calls are quite frequent in your code and it's no problem, but you might want to consider using the kwargs dictionaries like this:

```
kwargs = {"n"=args.n, "m"=args.m, "t0"=args.t0, "t1"=args.t1, "dt"=args.dt, "nu"=args.nu, \
         "hs"=args.hs,"v"=args.v, "t"=args.t, "img"=args.img, "input"=args.input, \
         "output"=args.output,"type"=args.type, "test"=args.test}
```

Then return the dictionary and then maybe swap some values around and pass all the attribute values like this:

```
data_list = heat_equation_numpy(**kwargs_new)
```

This, at least, saves some space.