# Hands-on: Python for Scientific Computing, a performance aspect

**Fábio Köpp & Ivan Girotto**

*fabio.kopp@ufrgs.br & igirotto@ictp.it*

4th Workshop on Advanced Techniques for Scientific Programming and Management of Open source Software Package
Trieste, Italy - ICTP


Thursday, 10 March 2016

# Summary

# Using the command time

The command time will show us the real time, user time and system time. This command can be used in different ways. That is, you can look for the time spent to run ls, for example. This can be done with the command: time xxxx
. Where xxxx could be ls or your code python filename.py In order to see the performance of different libraries, we will compare the native library's python vs numpy libraries.

Numpy example

```
import numpy as np
Nelements = 1000000
x = np.arange(Nelements)
s=np.sum(x)
print "The sum(numpy) of the sequence (1-1000000) is" , s
```

```
mav@fkopp:~/hands-on$ time python time11.py
The sum(numpy) of the sequence (1-1000000) is 499999500000

real    0m0.201s
user    0m0.100s
sys     0m0.044s
```

Native python library

```
N =1000000
x=range(N)
s=sum(x)
print "The sum(native library) of the sequence (1-1000000) is" , s
```

```
mav@fkopp:~/hands-on$ time python time00.py
The sum(native library) of the sequence (1-1000000) is 499999500000

real    0m0.126s
user    0m0.104s
sys     0m0.020s
```

# Example with time embedded in code

In this example, we just have to run the script run.sh provided in the zipfile.

```
from timeit import default_timer as timer
#mathnative
start = timer() #starting the clock

N =1000000
x=range(N)
s=sum(x)

end = timer()  # stopping the clock
print(end - start)  , "seconds [native library]"
```

```
mav@fkopp:~/hands-on/codes/time $ ls
run.sh  time0.py  time1.py
mav@fkopp:~/hands-on/codes/time $ ./run.sh
---------------------------
comparing numpy vs native without profiler
---------------------------
---------------------------
0.0467429161072 seconds [native library]
---------------------------
0.00434994697571 seconds [numpy library]
---------------------------
mav@fkopp:~/hands-on/codes/time $
```

# Example with the cProfiler

The command to run the profiler is:

python -m cProfile yourcode > yourcode.txt

Here, we are writting the output in yourcode.txt. In the example presented here, we compare the Romberg integration using two ways: coding all the Romberg integration and using scipy (library). The integral to be evaluated is $\int_{0.5}^{1} tan(x)dx$. The results are:

```
Romberg integration of <function vfunc at 0x7f9f6256ed70> from [0.5, 1.0]

 Steps  StepSize   Results
     1  0.500000  0.525928
     2  0.250000  0.495863  0.485841
     4  0.125000  0.487795  0.485105  0.485056
     8  0.062500  0.485734  0.485046  0.485043  0.485042
    16  0.031250  0.485215  0.485043  0.485042  0.485042  0.485042
    32  0.015625  0.485086  0.485042  0.485042  0.485042  0.485042  0.485042

The final result is 0.485042229943 after 33 function evaluations.
0.485042 0.842701
         |   43320 function calls (42375 primitive calls) in 0.163 seconds
```

```
Romberg integration of <built-in function tan> from (0.5, 1.0)

 Steps  StepSize   Results
     1  0.500000  0.525928
     2  0.250000  0.495863  0.485841
     4  0.166667  0.487795  0.485105  0.485056
     8  0.125000  0.485734  0.485046  0.485043  0.485042
    16  0.100000  0.485215  0.485043  0.485042  0.485042  0.485042
    32  0.083333  0.485086  0.485042  0.485042  0.485042  0.485042  0.485042

The final result is 0.485042229943 after 33 function evaluations.

Trapezoidal rule with 1000 function evaluations  = 0.485042274256
Theoretical result (correct to all shown digits) = 0.48504222994229

         1111 function calls in 0.001 seconds
```

Exercise

You must implement the time embedded method in the codes and compare with the previous result (cProfiler).

# Link for download the codes

Link for download the codes:

`https://drive.google.com/open?id=0B4CnBtk2FaZQRktKZHFDdkg1SDQ`