Gnuplot, Python and automation

Abstract

In this assignment, the functionalities of both gnuplot and Python are exploited in order to automatize the process of plotting and fitting data.

Code development

Ex 1: The first exercise is focused on the integration of a Fortran program for the timing of various matrix-matrix multiplication algorithms (as a function of the size N) with a Gnuplot script, which plots the various measurements. The interaction is piloted by a Python script as follows.

At first, all previous instances of executables, results files, and input files with the sizes are erased, so that no conflict ensues. After this step, all executables are re-compiled with the appropriate optimization and the needed modules.

Then, a set of logarithmically[1] spaced integer values from a minimum N_{min} and a maximum of N_{max}, both asked to the user and properly checked to be positive, is saved on a list; for each element of this list, one at a time, an input file is created, and is then to be used by the Fortran program to time the various performances for that specific matrix size.

Since the Fortran program creates an output file called `results.txt` regardless of the given size, the Python script needs to rename this file right after it has been created, to avoid any eventual overwrite.

After all the results for all the sizes have been created, if the appropriate Gnuplot script exists (if that is not the case, a warning is issued to the user), it is launched by the Python script, and a plot for each optimization flag is created. To do so, the Gnuplot script uses a `for` loop to go through the 4 different output files, and the various input files are provided manually: since four different algorithms (two different implementations of a by column algorithm, one of a by row algorithm and the intrinsic `MATMUL`) were tested, their four names are manually written in the script.

Ex 2: The second exercise is instead centered around the automation of a fitting script. In particular, a single Gnuplot script was to be used to perform multiple fits on the different result files that were produced in the previous exercise.

The Gnuplot script was implemented in such a way that its parameters are obtained from an external `settings.txt` file. In this way, no modification of the Gnuplot script had to be performed by the Python script, and its only purpose was then to produce all the needed `settings.txt` files.

[1] This choice is motivated by the fact that the increase in times is not linear, and so a non linear choice better 'probes' the functional dependence.

To do so, the `os` module is used, as it allows to create a list of all the `.txt` and `.dat` files in the current directory. For each of these files, a `settings.txt` is created, containing its full name (to be used from Gnuplot at the time of loading the data) and the extension-less name (to be used in order to name the output plot, to title it and to name the logfile). Both these strings had to be enclosed in single quotes, for reasons that will be soon explained.

Once the `settings.txt` file is created, the Gnuplot script is capable of storing its contents into variables by means of the following command:

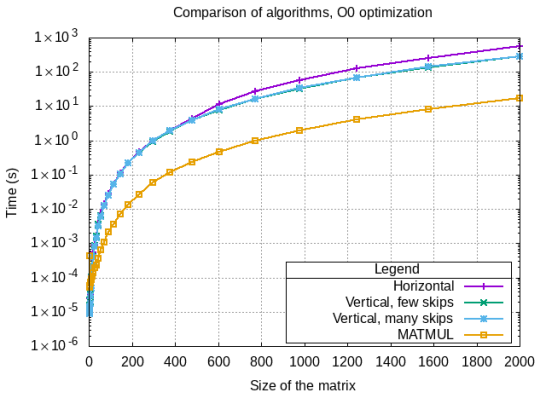
```
input_file = `sed -n 1p settings.txt`
```

Using backticks allows to store the output of the command they enclose; the result is that a single Gnuplot script is used multiple times.

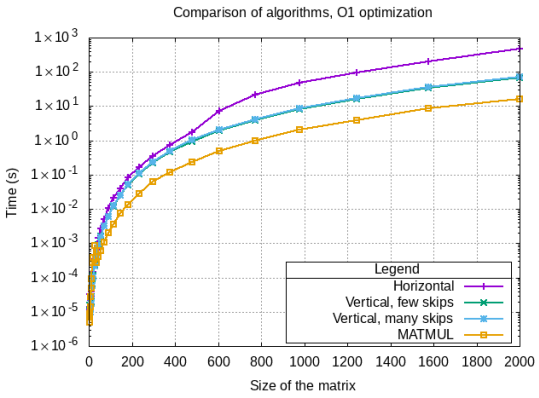
The single quotes in the file `settings.txt` are needed as what they enclose gets interpreted by Gnuplot as a string. A lack of single quotes would result in an error at runtime.

Results

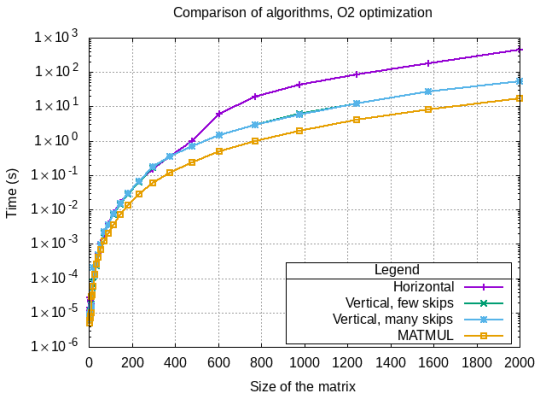
The automated scripts of the first exercise produced the following plots for the different optimization flags.



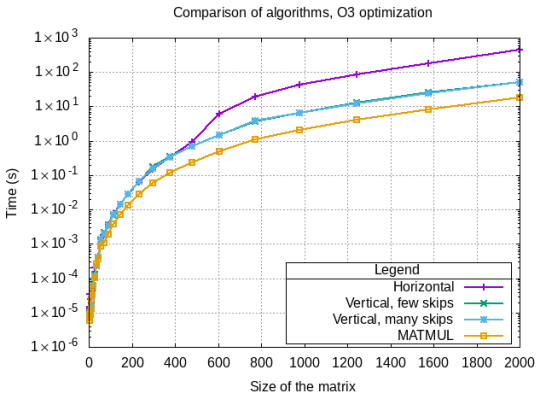
(a) No optimization.



(b) -O1 optimization.



(c) -O2 optimization.



(d) -O3 optimization.

Increasing the number of tested sizes showed that, for the 'naive' algorithm, a performance bottleneck appears for the optimized versions after some critical size (approximately 400-500) has been reached; this may be due to the algorithm exceeding the cache limit, thus slowing the whole process down.

The second exercise was also successfully completed; for each optimization and for each algorithm, a fit was performed using the function

$$f(x) = a + b \cdot x^c,$$

with a, b and c as fitting parameters[2], and the resulting function was printed together with the associated data. Also, the fitting function is shown in the legend. An example follows.

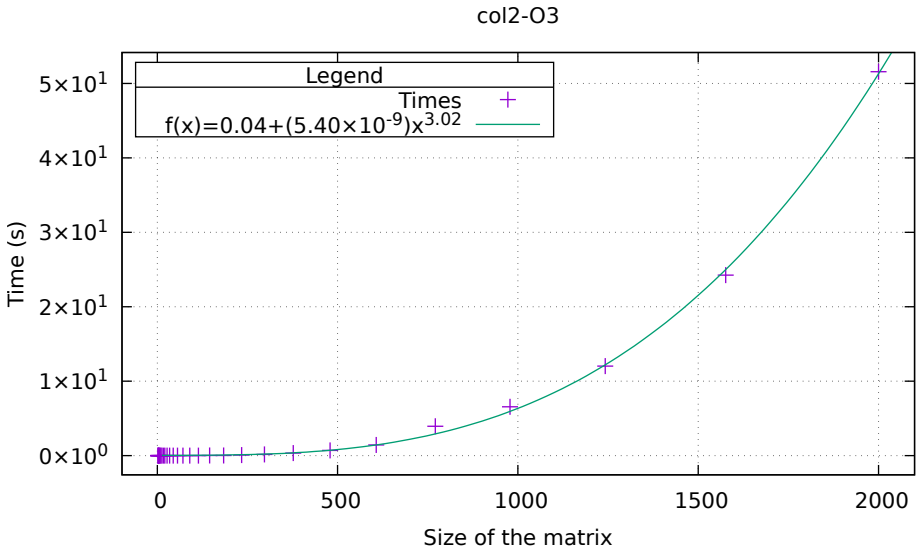


Figure 1: An example of the data and the fitting function for the 'many skips' vertical algorithm (jki ordering, cfr. the code in `matmul_performance.f90`).

All the pertinent parameters are also saved in the corresponding `.log` file:

```
*******************************************************************************
Mon Nov  4 20:37:58 2019


FIT:    data read from input_file using 1:2
        format = x:z
        #datapoints = 30
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
        f(x)=a+b*x**c
fitted parameters initialized with current variable values
```

[2]This choice allows to estimate the scaling parameter; perhaps unsurprisingly, $c \sim 3$ in almost all cases, with the exception of the naive algorithm, for which c is in the range 3.27-3.52.

```
 305 2.1397509037e+00  -2.11e-01  2.21e-06    4.161605e-02   5.403437e-09   3.022438e+00

After 305 iterations the fit converged.
final sum of squares of residuals : 2.13975
rel. change during last iteration : -2.11227e-06

degrees of freedom    (FIT_NDF)                        : 27
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.281514
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.07925

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.0416161        +/- 0.05721      (137.5%)
b               = 5.40344e-09      +/- 1.347e-09    (24.94%)
c               = 3.02244          +/- 0.03268      (1.081%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.332  1.000
c               0.327 -1.000  1.000
```

Comments and self evaluation

This week's assignment was an occasion for me to learn some of the functionalities of the various programming tools that I was not aware of. Beside the usage of Python modules such as `os` and `subprocess`, I also learned many useful Gnuplot commands, such as the ability to read external files outside of data representation, or to better control the output, either of fit or plots.

I consider these exercises to have been carried out appropriately. Among the possible improvements, a system to better keep track of all the different files that are produced could be considered; for instance, organizing the files in subdirectories and/or using more meaningful names.