Multi-run script & Automated Fits

Umberto Maria Tomasini
Quantum Information Course

06/11/2018

Abstract

In this report we will take the code of the exercise 3 of the first week -which was about matrix- matrix multiplication via three different methods- and we will modify it in order to read the dimension of the matrix from a file. Then we eill write a script which executes that program in a certain range of matrix's dimension, plotting the resulting Cpu_time over those dimensions. Lastly we will write another script which will automatically fit the data, compute and plot absolute residuals, relative residuals and their logarithm, via Gnuplot.

Theory

Code Development

Program Ex.3 modified

The code of the third exercise was modified as follows:

- the multiplication squared matrix- squared matrix via three different methods (row by row, column by column and by the intrinsic function $matmul$) is done for only one dimension. That dimension is taken as imput by the file "MatDimension.txt".
- the three outputs (the cpu_times spent on each method) are printed in three different file: "Results-1" for the row by row method, "Results-2" for the column by column method and "Results-3" for the intrinsic function method (the access to file is "append"). The formatting of the output files was modified: write(unit=unit,"(I4,4X,E16.9)", iostat=my_stat, iomsg=my_msg). The file is such that in every line there is an integer of max 4 figures (the matrix dimension) and another number which is written in scientific notation with nine significant figures (the cpu_time), separated by four spaces.
- In order to make more clear the execution of the script, we have chosen to turn the value of the variable DEBUG to ".false." (the program was alredy executed without flaws).
- The computation of the total time spent on debugging is removed, as well as the printing of the istant of time.

The complete code is in the last section.

Scripting: making fits and plots

The following script executes the above program from a dimension nmin to a dimension nmax, with a certain step. The results are three file called "Results-1", "Results-2" and "Results-3". For every file of resulting data considered, the script gives as output:

- A linear fit, in a file called "linearfit-%d.pdf", where "%d" is the number of the "Results-" file considered. The fit is done taking the log of the resulting data. This is linear. In fact for high matrix dimension the computational cost of each method should be: $\Delta t \approx a \cdot n^3$, discarding the non-leading terms. Taking the logarithm: $\log(\Delta t) \approx 3\log(n) + \log(a)$, which is linear in $\log(n)$.
- The absolute residuals $|\Delta f = f_{th} - f_{exp}|$, in a file called "absres-%d.pdf".
- The relative residuals $|\frac{\Delta f = f_{th} - f_{exp}}{f_{th}}|$, in a file called "relres-%d.pdf".
- The relative residuals $\log\left(|\frac{\Delta f = f_{th} - f_{exp}}{f_{th}}|\right)$, in a file called "logrelres-%d.pdf".
- The data of each fit, contained inside "parametersfit-%d."

Moreover all three sets of data are plotted on the same pdf, called "rawplot.pdf". The last output is another pdf, called "all3linearfit.pdf" which contains the plots of all three fitting functions (of the three methods). In the same figure the resulting data are plotted or, more precisely, their logarithm (in order to make the comparison with the fitting function reasonable). The parameters of those fits are contained in the file called "all3parametersfit.txt".

Listing 1: Scripting: making fits and plots

```
import numpy as np
import sys
import os
import subprocess



nmin=100
#lower bound dimension matrix
nmax=2525
#upper bound dimension matrix
stepp=25
#step of the increasing dimension

#########################################################
#cycle to re-iterate the launch of the program
#that multply two matrices in three different method
#and compute the computational cpu time needed
#with the dimension of the matrix that goes from
#nmin to nmax with step
#the result are saved in three files:
#"Results-1": multiplication row by row.
#"Results-2": multiplication by col by col.
#"Results-3": multiplication by intrinsic matmul.

#########################################################

for ii in range(nmin,nmax,stepp):
f = open("MatDimension.txt","w")
#opening the file with the dimension of the matrix
f.write(str(ii))
f.close()
#write the dimension on the opened file
subprocess.call("./a.out")
#execute the fortran program


#########################################################
#PLOTTING the three sets of data on the same pdf: "rawplot.
```

```
   pdf"

#Within gnuplot the list of commands contained in "rawplot"
   is executed
#On the x-axis there is the dimension N of the matrix
#On the y-axis there is the Cpu time
#the x range is from nmin to nmax (to be inserted manually
   inside "rawplot")
#Results 1: by row. Results 2: by col. Results 3: intrinsic
   matmul.
#########################################################

os.system("gnuplot␣rawplot")
#Results 1: by row. Results 2: by col. Results 3: intrinsic
   matmul.

#########################################################
#AUTOMATIC FITTING of the sets of data, separately.

#For every "Results-%d" file, the list of commands within "
   fitting" executes:
#linear fit (taking the log of the cubic law), inside "
   linearfit-%d.pdf"
#absolute residuals (\Delta f), inside "absres-%d.pdf"
#relative residuals (\Delta f/f), inside "relres-%d.pdf"
#logarithm of relative residuals (log(\Delta f/f(), inside "
   logrelres-%d.pdf"
#the x range is from nmin to nmax (to be inserted manually
   inside "rawplot")
#On the x-axis there is log(N), with N the dimension of the
   matrix
#The data of the fit are contained inside "parametersfit-%d."

#Logic of the script: the data contained in the considered "
   Results-%d" file
#are copied inside a temporary file "resultstemp". On that
   file are done the
#plots and the fit listed above. The results are stored in
   file with a
#temporary name, which are changed in the names listed above,
    which contain
#the number of the "Results-%d" file. This applies also for
   the data on the fit.
```

```
#This procedure is valid for a general number of "Results-%d"
    file,
#contained in the same directory of the script.
#This number is saved in the file called "numresults.txt",
#from which it is read.
##########################################################

os.system("find . -name \"Results-*\" | wc -l > numresults.
   txt")
#prints the number of result files on a txt

num = open("numresults.txt","r")
nn= int (num.readlines()[0])
num.close()
#read number of result files

for ii in range(1,nn+1):
#cycle over the "Results-%d" file
res = open("Results-%d" % ii,"r")
reslines = res.readlines()
#reading data from result file number ii

temp = open("resultstemp","w")
for jj in reslines:
temp.write(jj)
#copying data in temporary file
res.close()
temp.close()
#closing files

os.system("gnuplot fitting")
#linear fit of data, absolute residuals,
#relative residuals, log of realtive residuals

newnamelinear= ("linearfit-%d.pdf" %ii)
os.rename("templinearfit.pdf",newnamelinear)
#renaming the file of the linear fit

newnameabs= ("absres-%d.pdf" %ii)
os.rename("tempabsres.pdf",newnameabs)
#renaming the file of the absolute residuals

newnamerel= ("relres-%d.pdf" %ii)
os.rename("temprelres.pdf",newnamerel)
#renaming the file of the relative residuals
```

```
newnamelog= ("logrelres-%d.pdf" %ii)
os.rename("templogrelres.pdf",newnamelog)
#renaming the file of the log of the relative residuals

os.remove("resultstemp")
#removing the temporary file of results

fitlog = open("fit.log","r")
fitlines = fitlog.readlines()
#reading parameters of the fit from "fit.log"

fitlogd = open("parametersfit-%d" % ii,"w")
for jj in fitlines:
fitlogd.write(jj)
#copying parameters on "parametersfit-%d"
fitlog.close()
fitlogd.close()
#closing files

os.remove("fit.log")
#removing "fit.log"


#########################################################
#FITTING the three sets of data all together.

#linear fit (taking the log of the cubic law)
#Within gnuplot the list of commands contained in "
   all3fitting" is executed
#On the x-axis there is log(N), with N the dimension of the
   matrix
#On the y-axis there is log(Cpu time)
#the x range is from nmin to nmax (to be inserted manually
   inside "all3fitting")
#Results 1: by row. Results 2: by col. Results 3: intrinsic
   matmul.
#########################################################

os.system("gnuplot␣all3fitting")
#Results 1: by row. Results 2: by col. Results 3: intrinsic
   matmul.

flog = open("fit.log","r")
all3fitlines = flog.readlines()
```

```
#reading parameters of the fit from "fit.log"

all3fitlog = open("all3parametersfit","w")
for jj in all3fitlines:
all3fitlog.write(jj)
#copying parameters on "all3parametersfit"

flog.close()
all3fitlog.close()
#closing files

os.remove("fit.log")
#removing "fit.log"

exit()
```

Gnuplot commands

In the script above are called via the command *os.system* three different lists of gnuplot commands.

Listing 2: Plotting the three sets of data together

```
set term pdf

set output "rawplot.pdf"

#name of the output file

set xr [100:3900]

#settin xrange from nmin to nmax

set xlabel "N"

set ylabel "Cpu␣time␣[sec]"

#on the x axis there is the dimension of the matrix
#on the y axis there is the cpu time

set title "Results-1:␣by␣row.␣Results-2:␣by␣col.␣Results-3:␣
   intrinsic␣matmul.␣"

set key top left

#the table of the contents is moved to the figure's top-left
```

```
plot "Results-1" u 1:2 w p lt rgb "blue", "Results-2" u 1:2 w
    p lt rgb "red", "Results-3" u 1:2 w p lt rgb "green"

#Results-1 (by row): blue
#Results-2 (by col): red
#Results-3 (by intrinsic): green

set term wxt

#set to normal terminal
```

Listing 3: Automatic fitting and residuals

```
#LINEAR FIT

set term pdf

set output "templinearfit.pdf"

#name of the output pdf

f(x)= a+b*x

fit  f(x) "resultstemp" u (log($1)):(log($2)) via a,b

#fitting the data taking the logarithm of both columns
#via a linear function

set xr [log(100):log(3900)]

#setting xrange

set xlabel "log(N)"

set ylabel "log(Cpu time)"

plot "resultstemp" u (log($1)):(log($2)) w p lt rgb "blue"
   notitle, f(x) notitle

#plotting of the data (the log(data)) and
#the interpoling function


#ABSOLUTE RESIDUALS
```

```
set output "tempabsres.pdf"

#name of the output pdf

p(x,y)=abs(f(x)-y)

#this function computes absolute residuals

set xr [log(100):log(3900)]

set xlabel "log(N)"

set ylabel "|{/Symbol␣D}f|" enhanced

plot "resultstemp" u (log($1)):(p(log($1),log($2))) w p lt
   rgb "green" notitle
#the p function accept as imput the two columns of data

#RELATIVE RESIDUALS

set output "temprelres.pdf"

#name of the output pdf

p(x,y)=abs((f(x)-y)/f(x))

#this function computes relative residuals

set xr [log(100):log(3900)]

set xlabel "log(N)"

set ylabel "|{/Symbol␣D}f/f|" enhanced

plot "resultstemp" u (log($1)):(p(log($1),log($2))) w p lt
   rgb "brown" notitle
#the p function accept as imput the two columns of data

#LOGARITHM OF RELATIVE RESIDUALS

set output "templogrelres.pdf"

#name of the output pdf

p(x,y)=log(abs((f(x)-y)/f(x)))
```

```
#this function is the logarithm of relative residuals

set xr [log(100):log(3900)]

set xlabel "log(N)"

set ylabel "log(|{/Symbol␣D}f/f|)" enhanced

plot "resultstemp" u (log($1)):(p(log($1),log($2))) w p lt
   rgb "red" notitle
#the p function accept as imput the two columns of data


set term wxt
```

Listing 4: Plotting the three fits together

```
set term pdf

set output "all3linearfit.pdf"

#setting the output pdf

f(x)= a+b*x

g(x)= c+d*x

h(x)= e+f*x

#three different linear function

fit  f(x) "Results-1" u (log($1)):(log($2)) via a,b

fit  g(x) "Results-2" u (log($1)):(log($2)) via c,d

fit  h(x) "Results-3" u (log($1)):(log($2)) via e,f

#three different linear fits

set xr [log(100):log(3900)]

set xlabel "log(N)"

set ylabel "log(Cpu␣time)"
```

```
set title "Results-1:␣by␣row.␣Results-2:␣by␣col.␣Results-3:␣
   intrinsic␣matmul.␣"

set key top left

#the table of the contents is moved to the figure's top-left

plot "Results-1" u (log($1)):(log($2)) w p lt rgb "blue", f(x
   ) lt rgb "blue" notitle, "Results-2" u (log($1)):(log($2))
    w p lt rgb "red", g(x) lt rgb "red" notitle, "Results-3"
   u (log($1)):(log($2)) w p lt rgb "green", h(x) lt rgb "
   green" notitle

#Results-1 and fitting f(x)(by row): blue
#Results-2 and fitting g(x)(by col): red
#Results-3 and fitting h(x)(by intrinsic): green

set term wxt
```

Results

Estimating N_{max}

In order to decide the range of the matrices dimension (from nmin to nmax) e the step of the increasing dimension, it was decided firstly to go from 100 to 2525 with step=25. The execution took more or less three hours and an half. The parameters of the three fits obtained were:

	Intercept	Slope
By row	-23.6514	3.75172
By col	-23.6811	3.7648
By matmul	-21.1094	2.95827

Using those parameters, the following Python code was implemented. Its goal is to estimate the time needed for further computations.

Listing 5: Estimating the total cpu time

```
import numpy as np

nmin=2600
nmax=4000
step=100
accum=0
```

```
for ii in range(nmin,nmax,step):
accum= accum+ np.exp(-23.6514)*(ii)**3.75172 + np.exp
   (-23.6811 )*(ii)**3.7648 +np.exp(-21.1094  )*(ii)**2.95827
#the parameters of the fits are used


print(accum)
#number of seconds
print(accum/3600)
#number of hours
```

Obviusly this program takes into account only the contribution $\approx O(n^3)$ of the multiplications, disregarding the minor contributions of the rest of the program. Deciding a priori that the maximum range of time given to further computations would be 7 hours/ 7 hours and half (a night's sleep), the range was set from 2600 to 4000 with a step of 100 (the 4000 was not computed).

Combining the results obtained from 100 to 2525 with a step of 25 and from 2600 to 4000 with a step of 100, the results are the following.

Results from 100 to 3900

Plot of the three sets of data

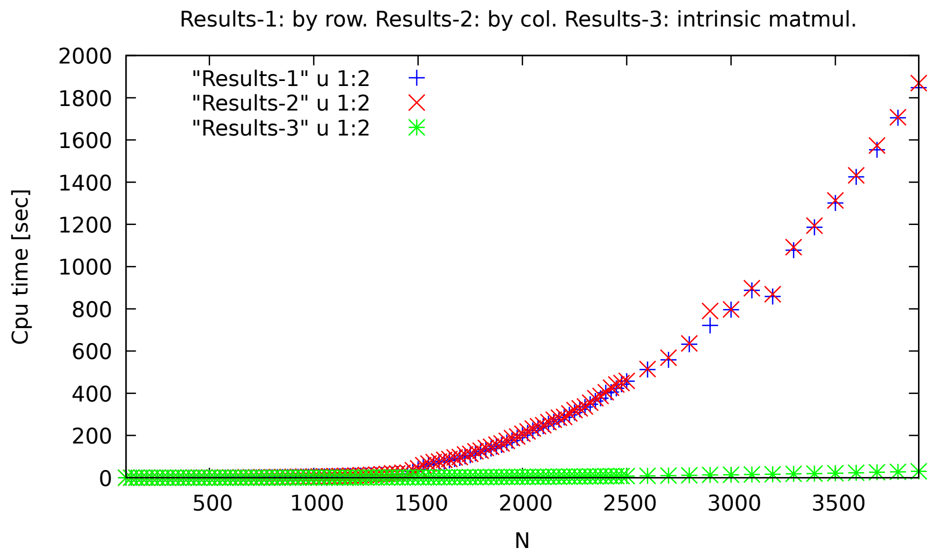

Figure 1: Plot of the three sets of data

Fit and residuals: row by row method

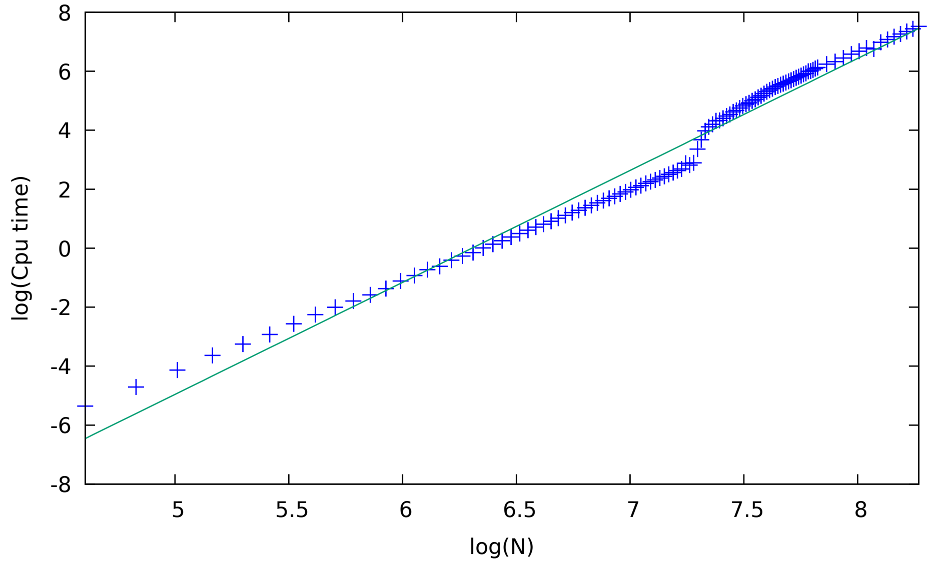

Figure 2: Linear fit: row by row

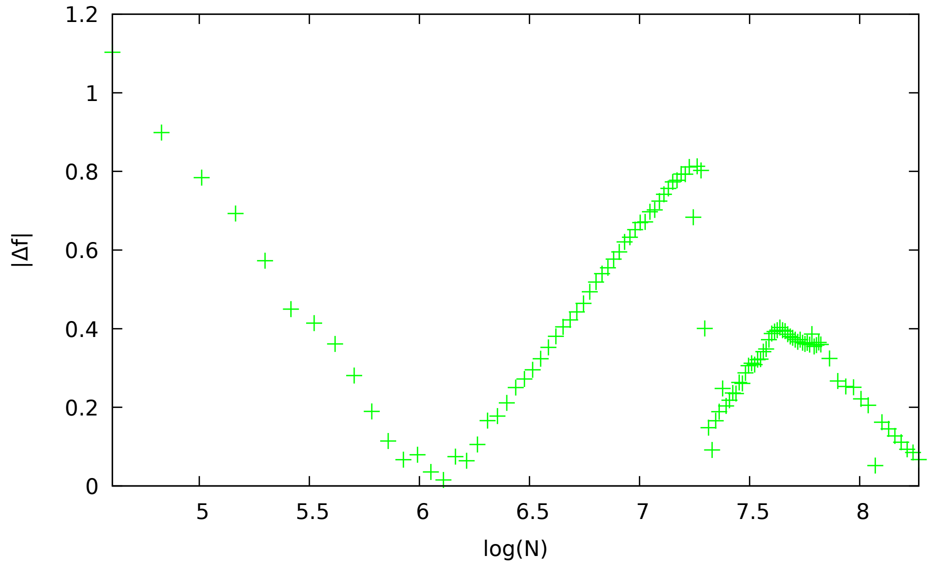

Figure 3: Absolute residuals: row by row

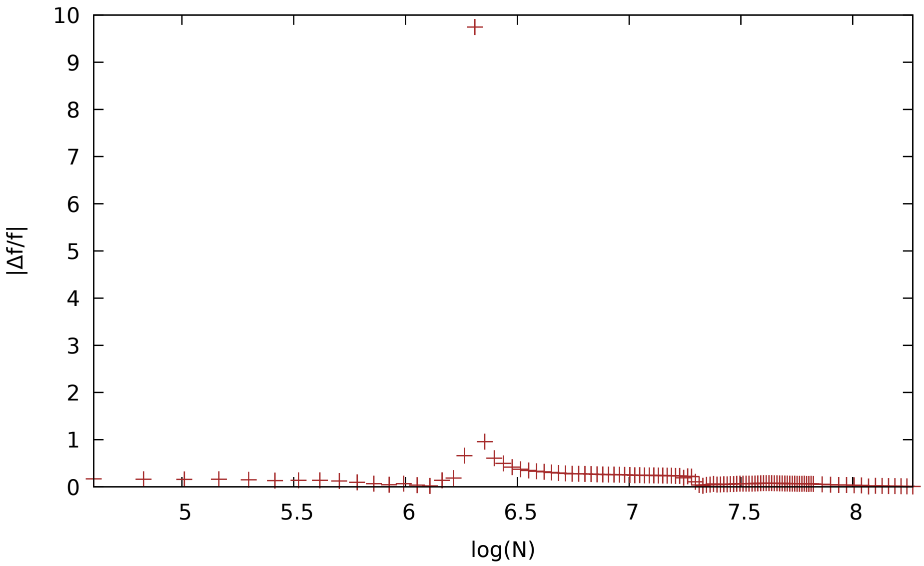

Figure 4: Relative residuals: row by row

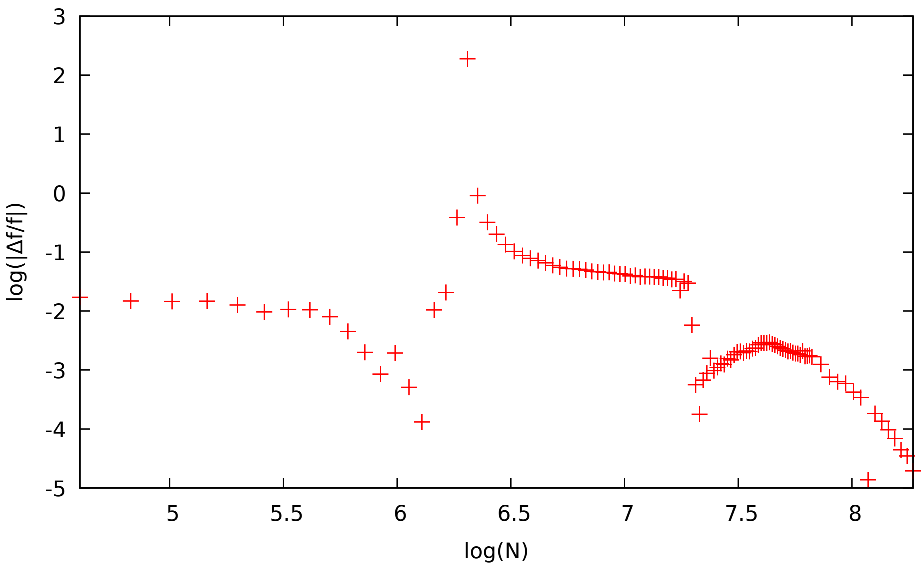

Figure 5: Log of relative residuals: row by row

Fit and residuals: column by column method

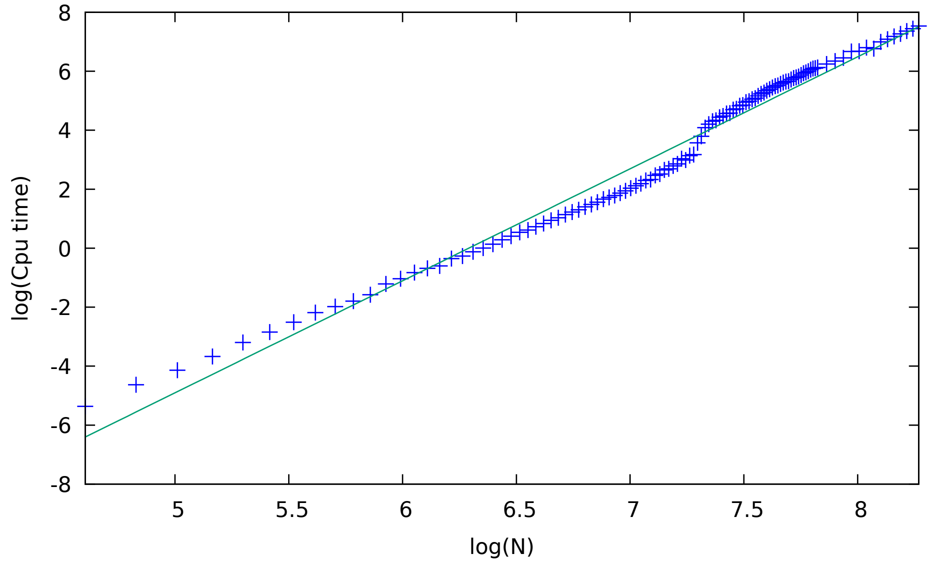

Figure 6: Linear fit: column by column

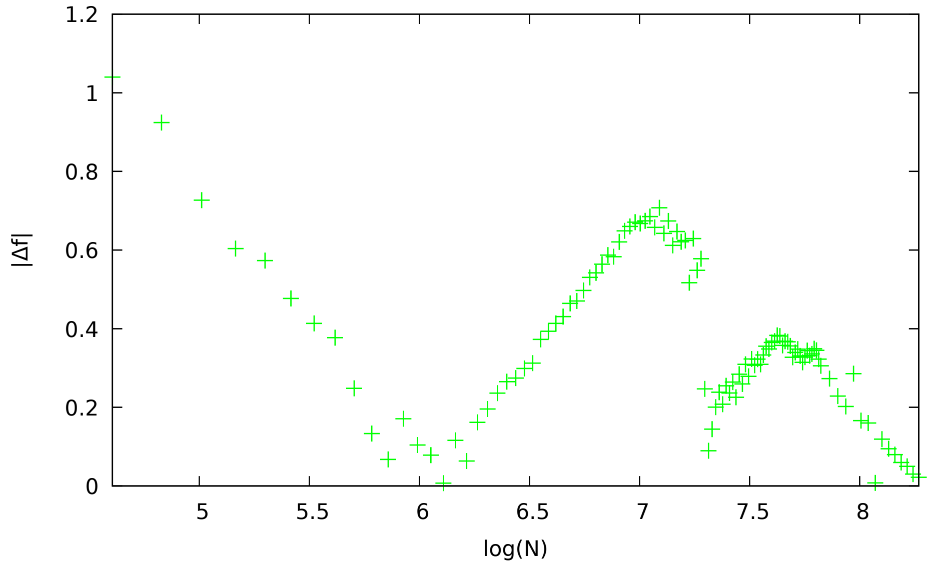

Figure 7: Absolute residuals: column by column

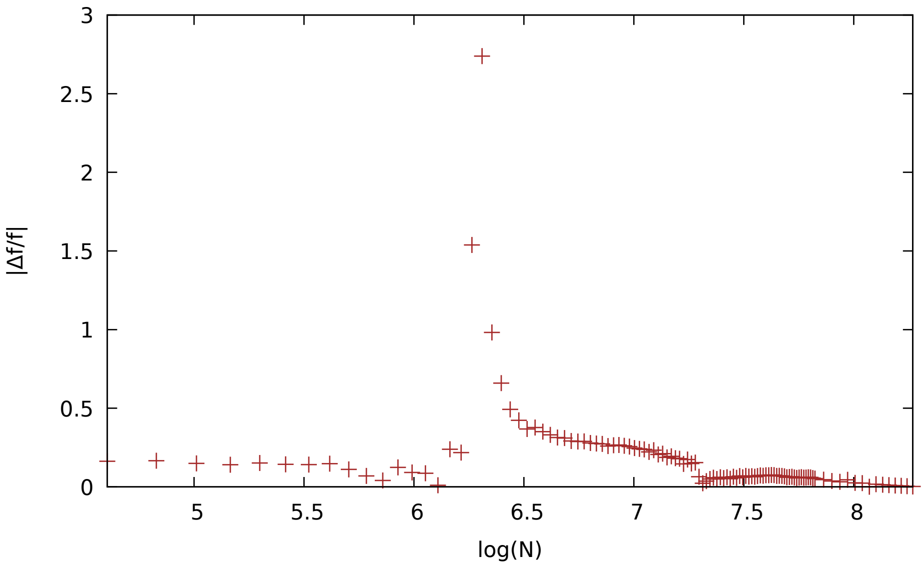

Figure 8: Relative residuals: column by column

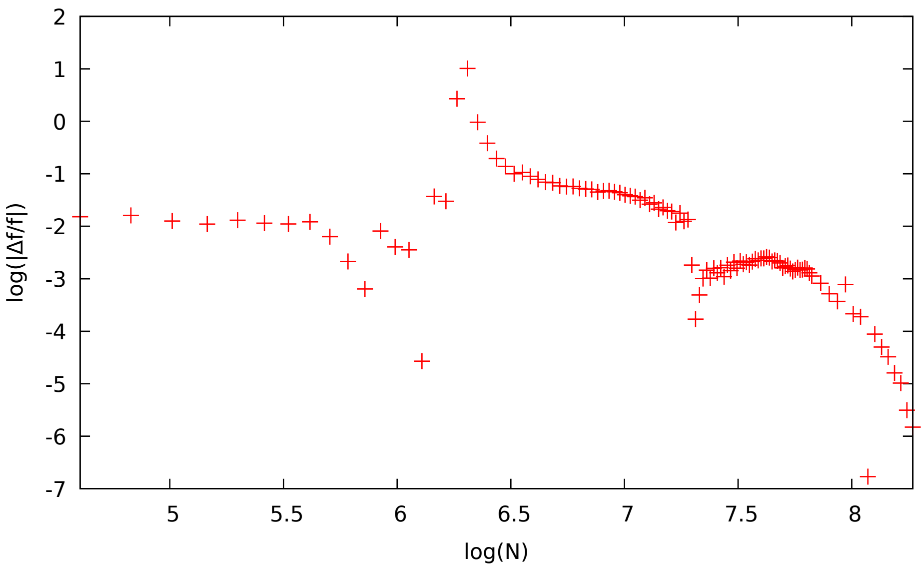

Figure 9: Log of relative residuals: column by column

Fit and residuals: intrinsic function method

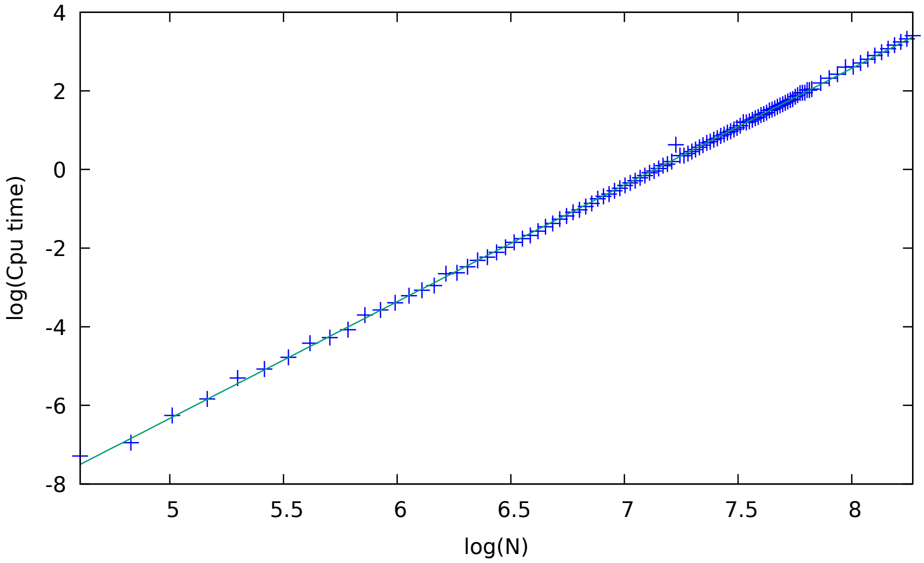

Figure 10: Linear fit: intrinsic function

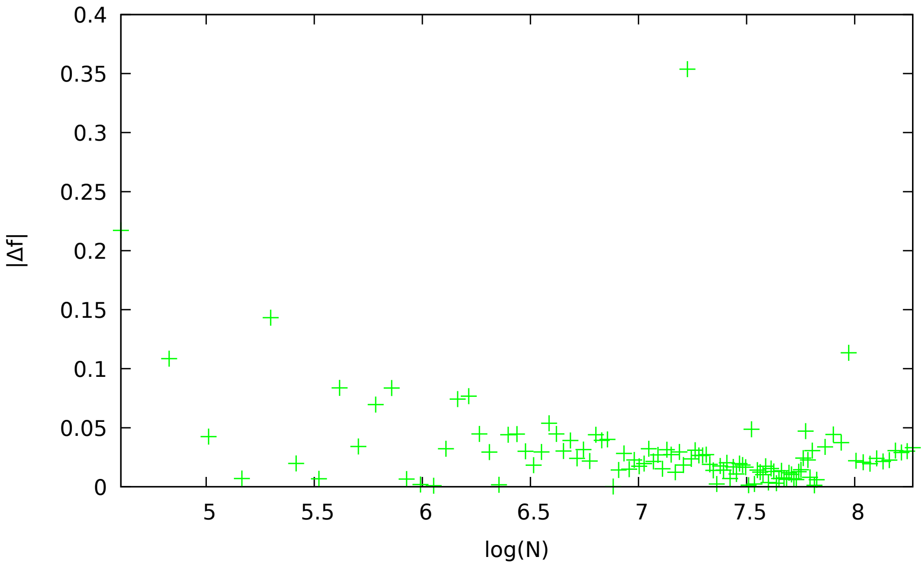

Figure 11: Absolute residuals: intrinsic function

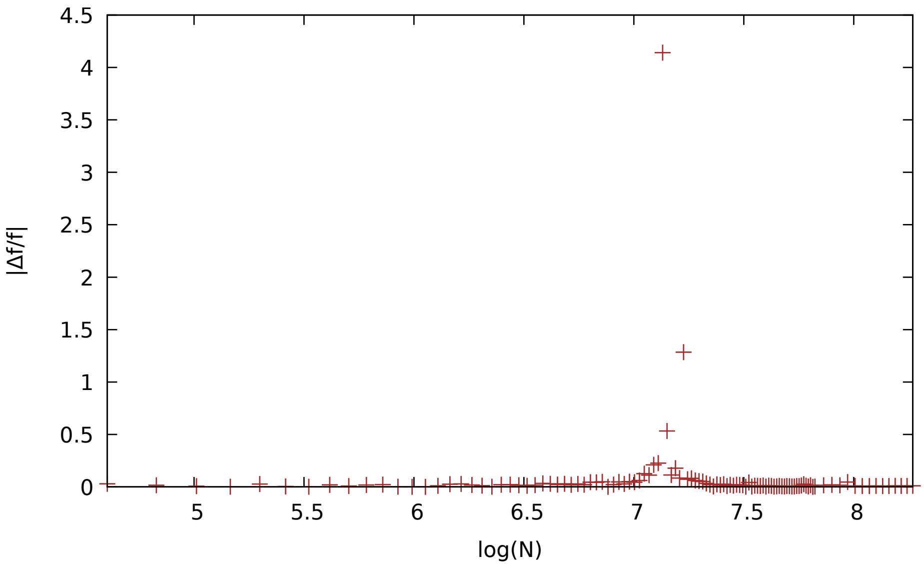

Figure 12: Relative residuals: intrinsic function

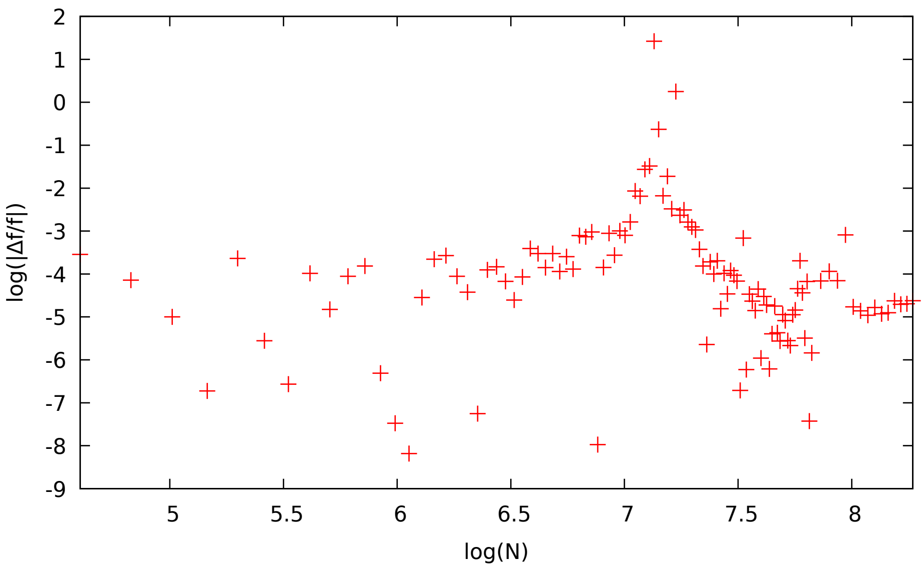

Figure 13: Log of relative residuals: intrinsic function

Plot of the three fitting function

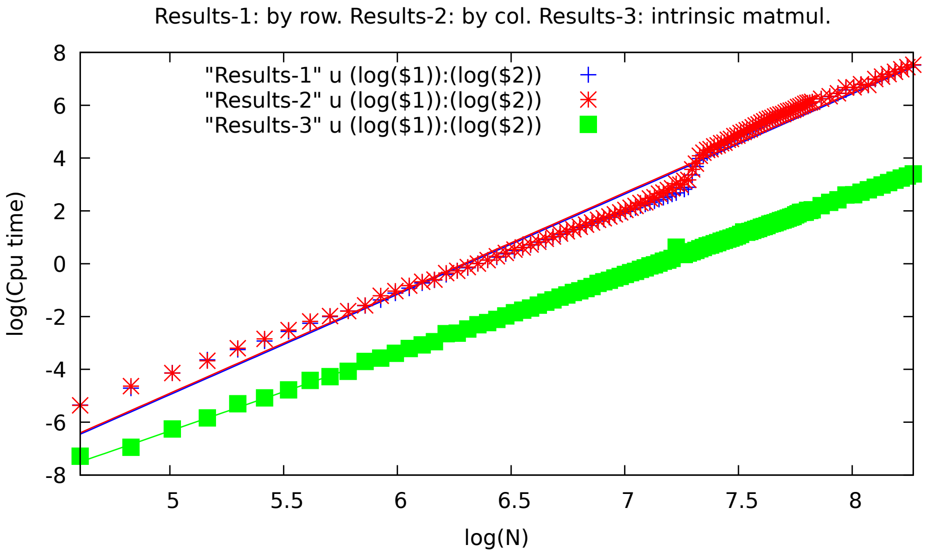

Figure 14: Plot of the three fitting function

Parameters of the three fits (the complete fit.log is below).

	Intercept	Slope
By row	-23.9417	3.797
By col	-23.8943	3.79811
By matmul	-21.1703	2.96775

Motivated by the fact that in the linear fits of the first two methods two different linear trends can be recognized, we repetead the analysis on a first zone from 100 to 1400 and on a second zone from 1800 to 3900. For sake of completeness we also repeated the anylisis on the third method, although there is only one main trend.

Results from 100 to 1400

Plot of the three sets of data. From 100 to 1400

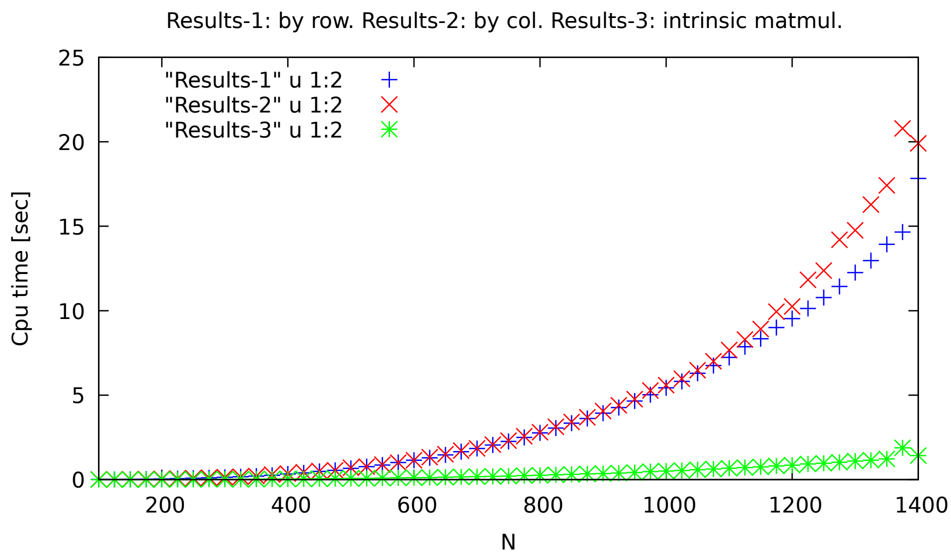

Figure 15: Plot of the three sets of data. From 100 to 1400

Fit and residuals: row by row method. From 100 to 1400

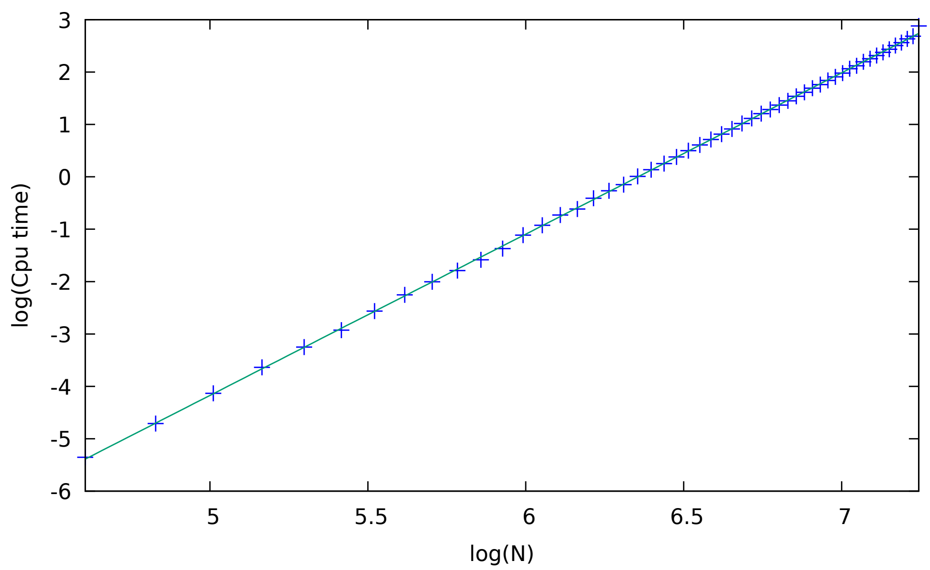

Figure 16: Linear fit: row by row. From 100 to 1400

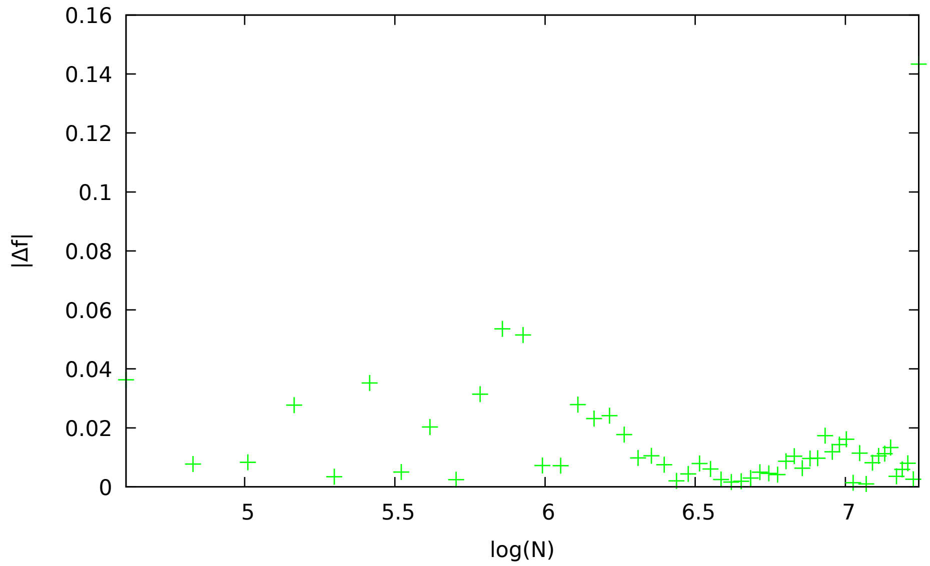

Figure 17: Absolute residuals: row by row. From 100 to 1400

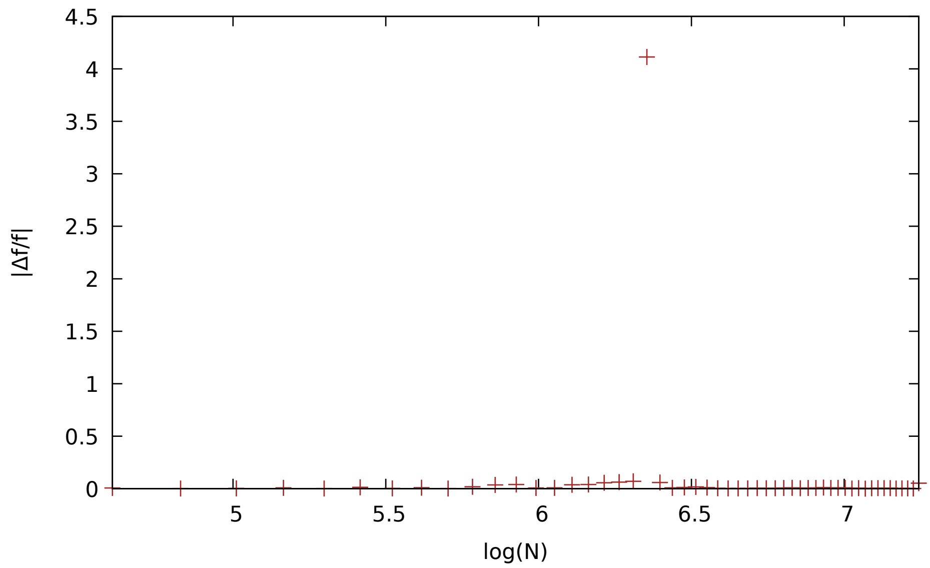

Figure 18: Relative residuals: row by row. From 100 to 1400

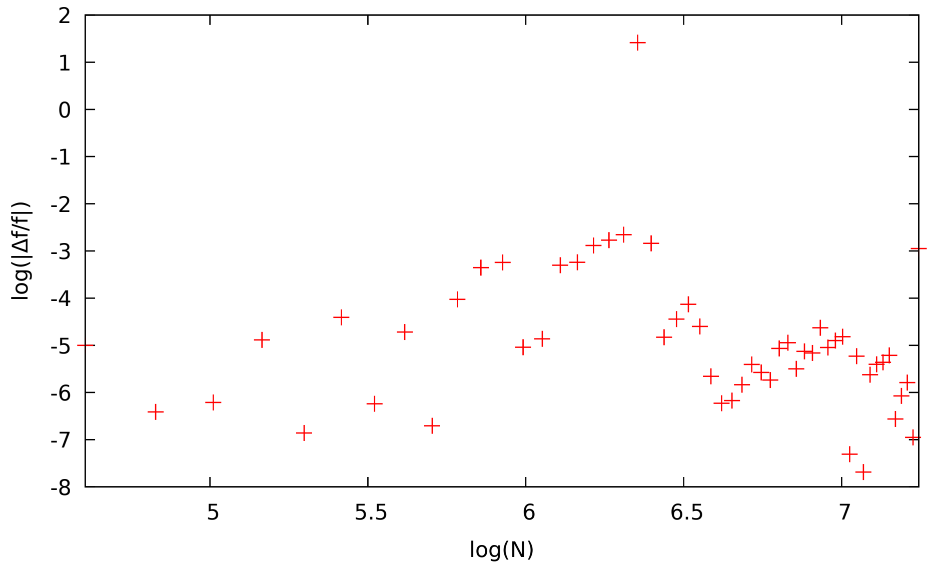

Figure 19: Log of relative residuals: row by row. From 100 to 1400

Fit and residuals: column by column method. From 100 to 1400

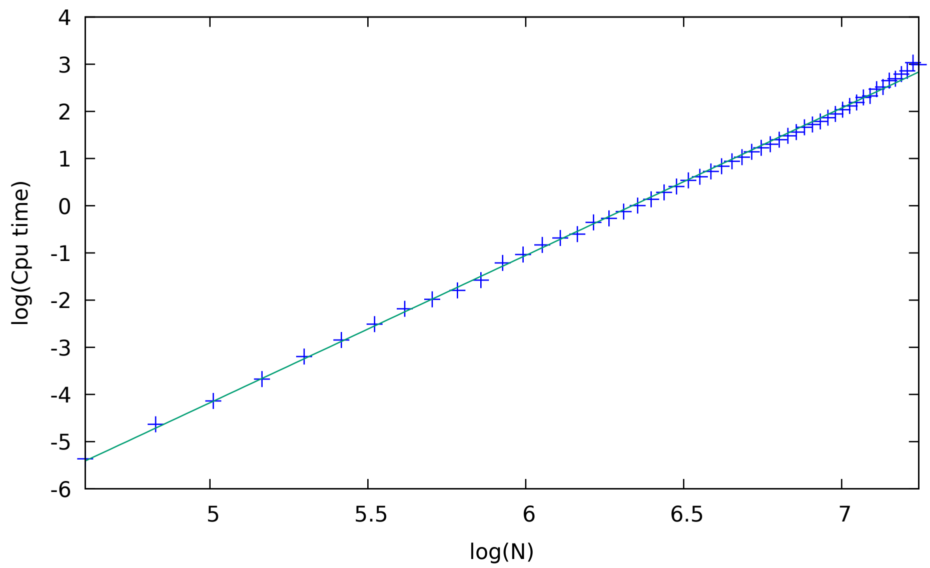

Figure 20: Linear fit: column by column. From 100 to 1400

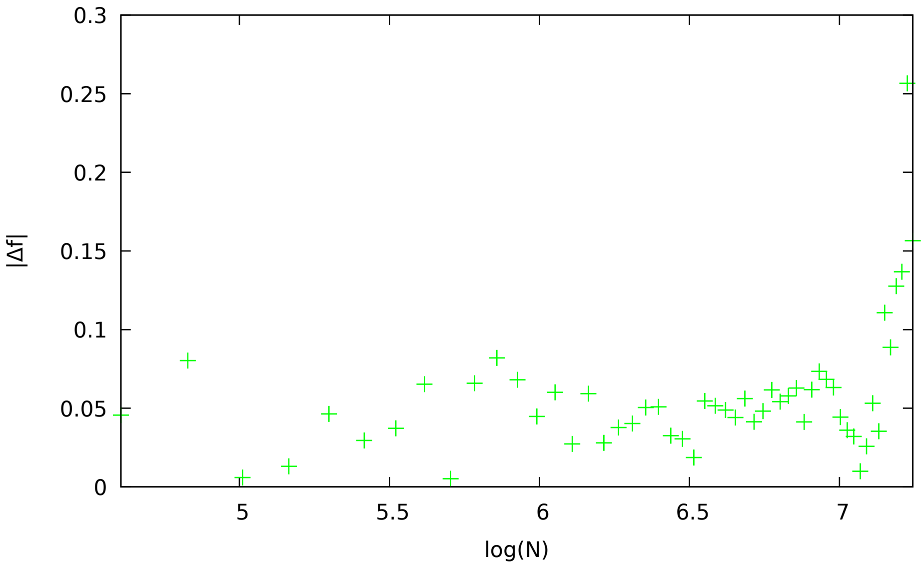

Figure 21: Absolute residuals: column by column. From 100 to 1400

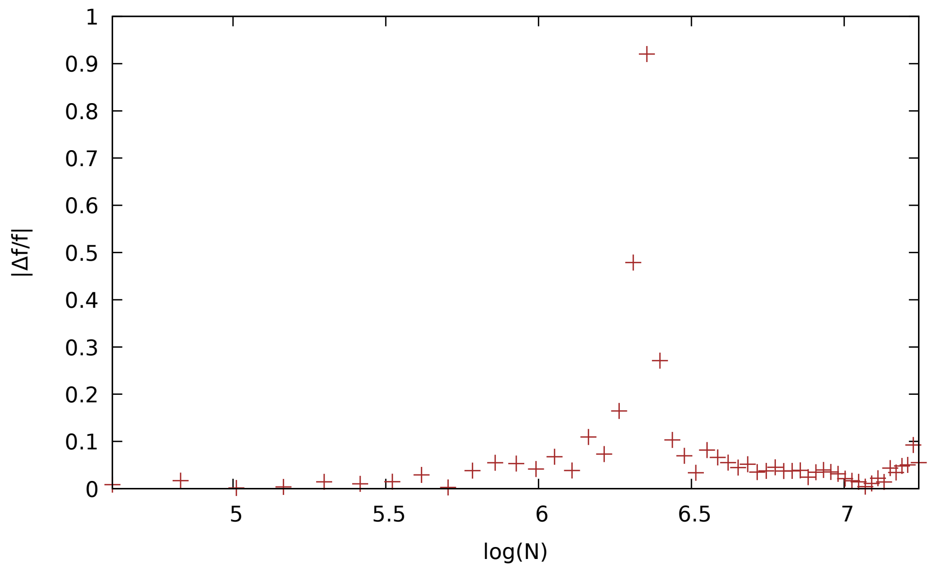

Figure 22: Relative residuals: column by colum. From 100 to 1400n

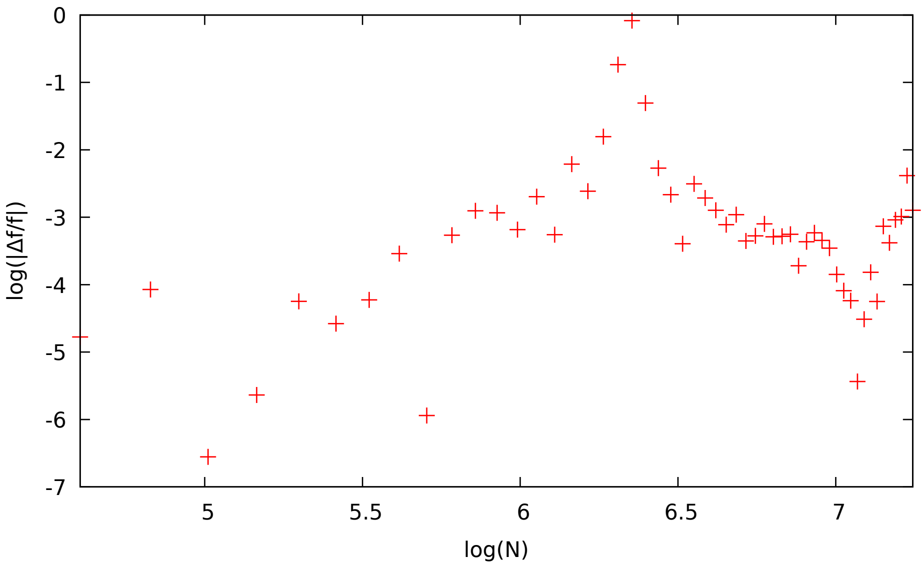

Figure 23: Log of relative residuals: column by column. From 100 to 1400

Fit and residuals: intrinsic function method. From 100 to 1400

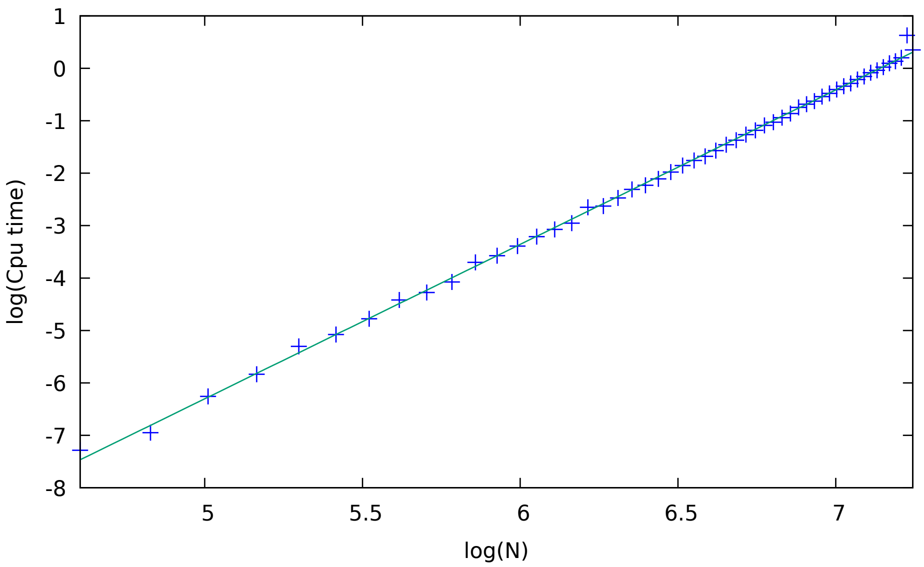

Figure 24: Linear fit: intrinsic function. From 100 to 1400

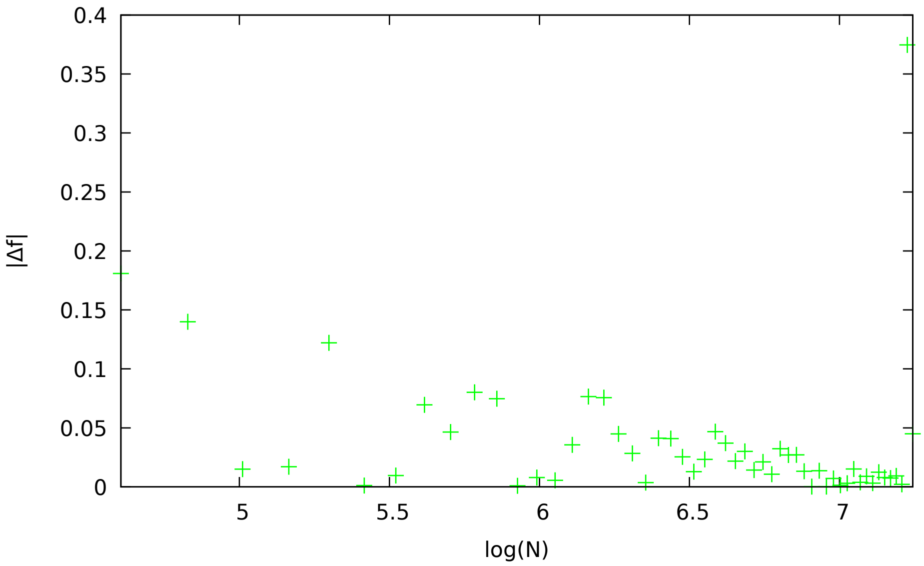

Figure 25: Absolute residuals: intrinsic function. From 100 to 1400

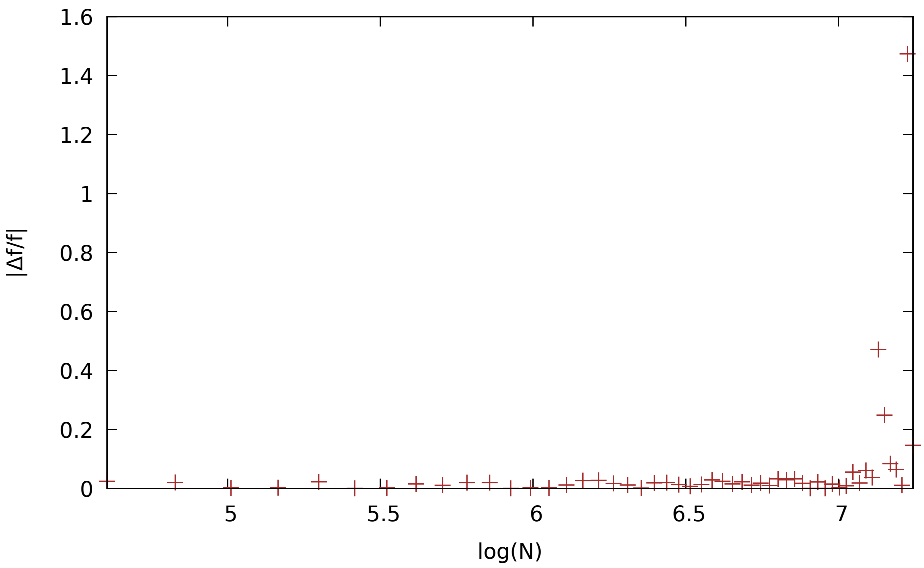

Figure 26: Relative residuals: intrinsic function. From 100 to 1400

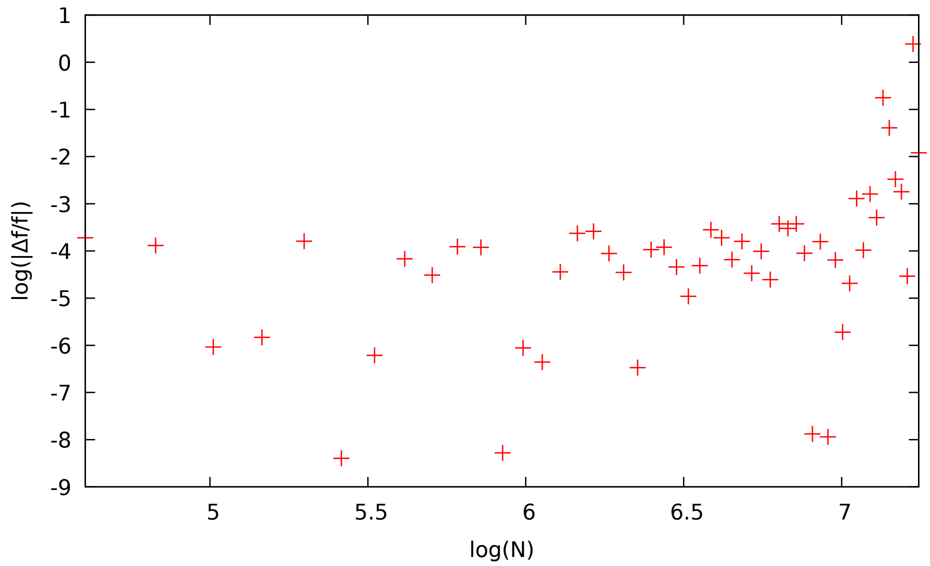

Figure 27: Log of relative residuals: intrinsic function. From 100 to 1400

Plot of the three fitting function. From 100 to 1400

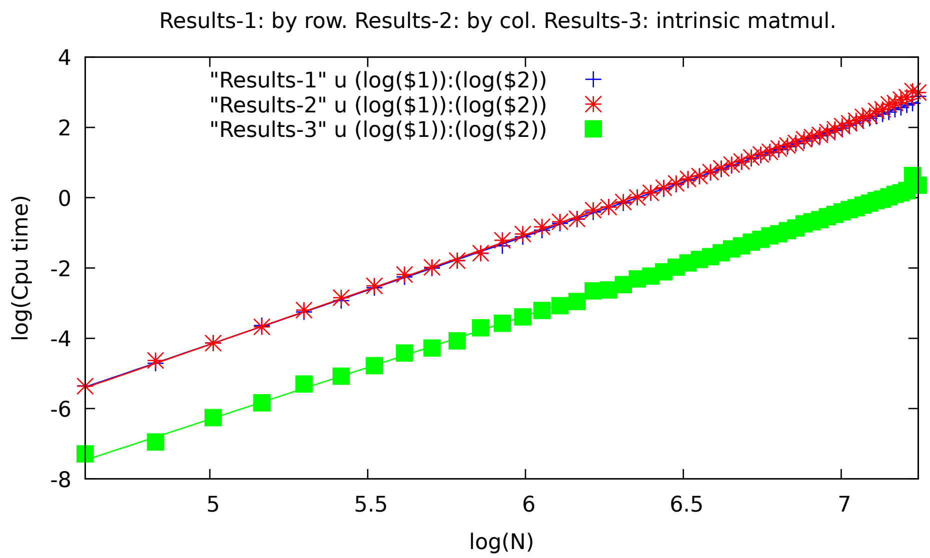

Figure 28: Plot of the three fitting function. From 100 to 1400

Parameters of the three fits (from 100 to 1400 (matrix dim)). The complete data are in the below subsection.

	Intercept	Slope
By row	-19.5703	3.07942
By col	-19.7937	3.1236
By matmul	-21.0333	2.94588

Results from 1800 to 3900

Plot of the three sets of data. From 1800 to 3900

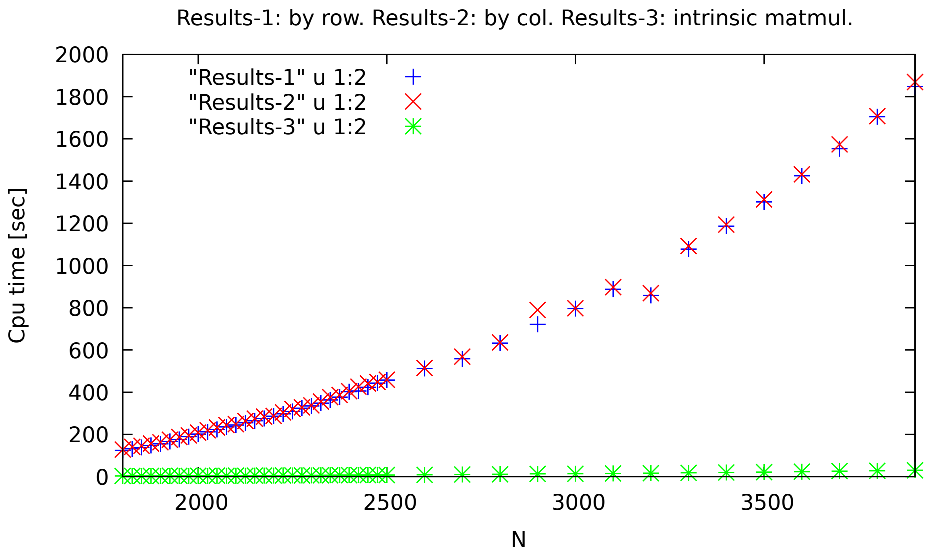

Figure 29: Plot of the three sets of data. From 1800 to 3900

Fit and residuals: row by row method. From 1800 to 3900

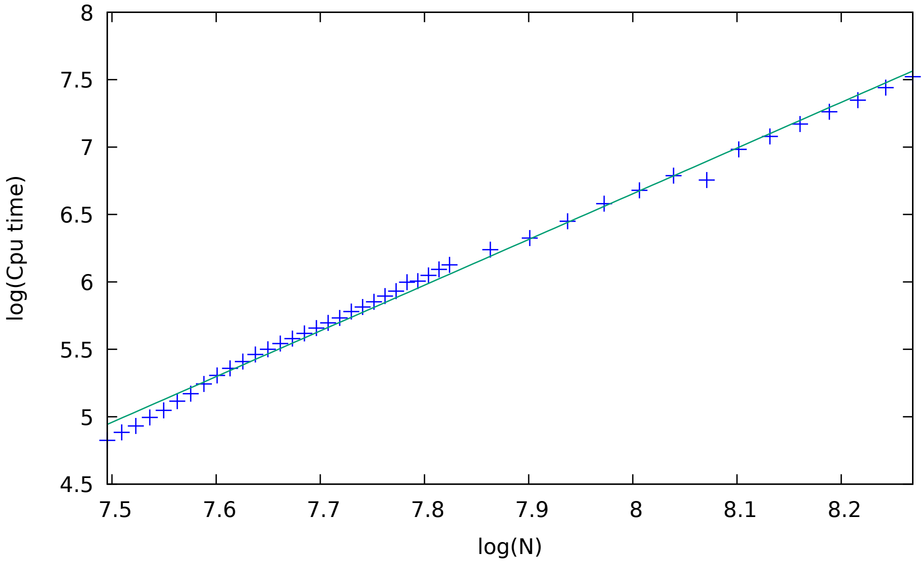

Figure 30: Linear fit: row by row. From 1800 to 3900

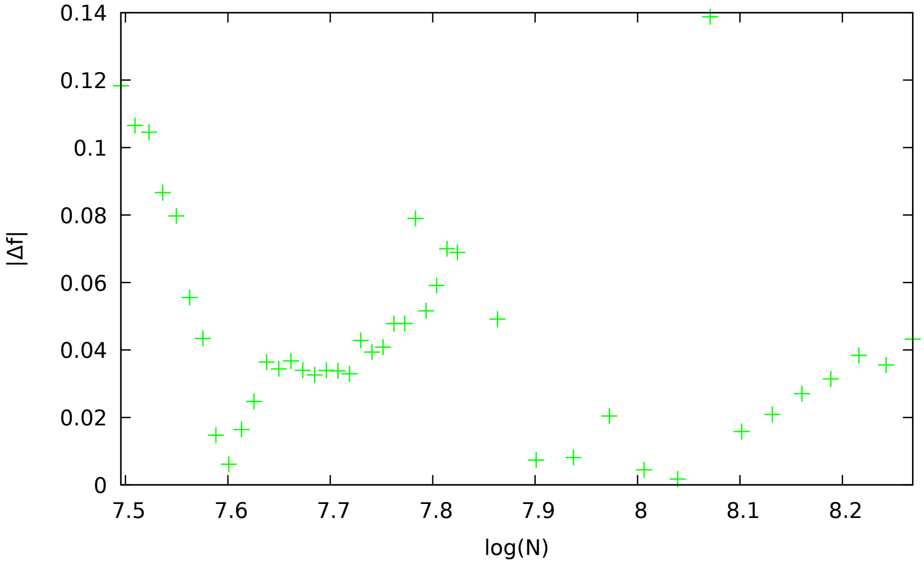

Figure 31: Absolute residuals: row by row. From 1800 to 3900

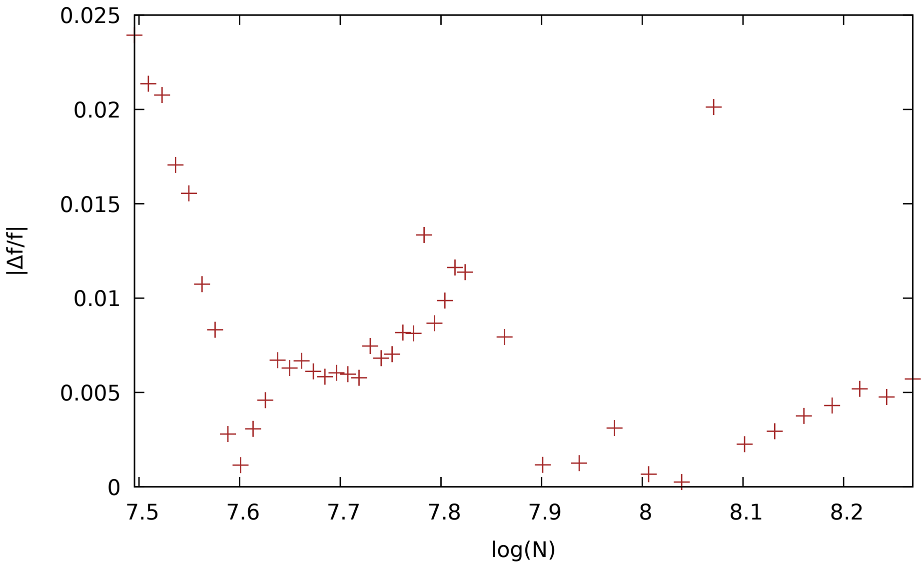

Figure 32: Relative residuals: row by row. From 1800 to 3900

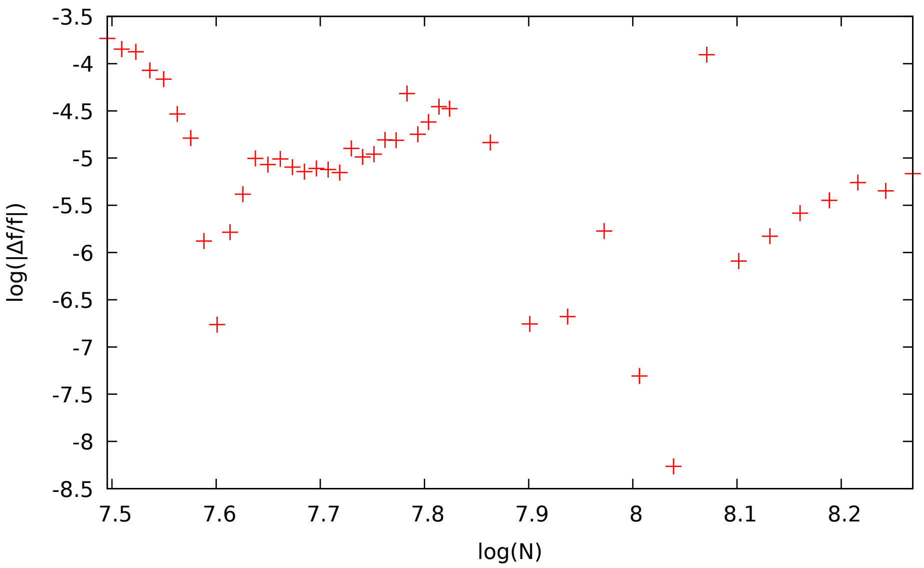

Figure 33: Log of relative residuals: row by row. From 1800 to 3900

Fit and residuals: column by column method. From 1800 to 3900

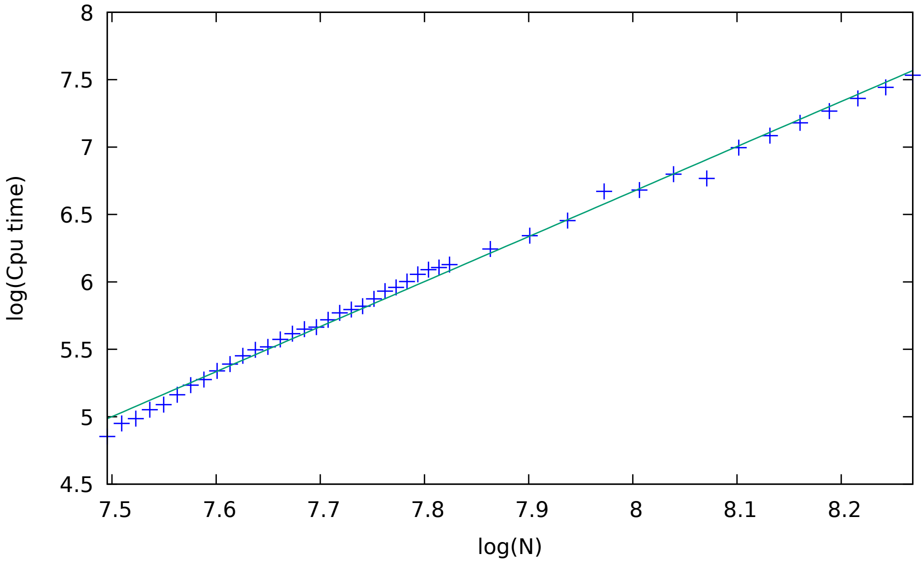

Figure 34: Linear fit: column by column. From 1800 to 3900

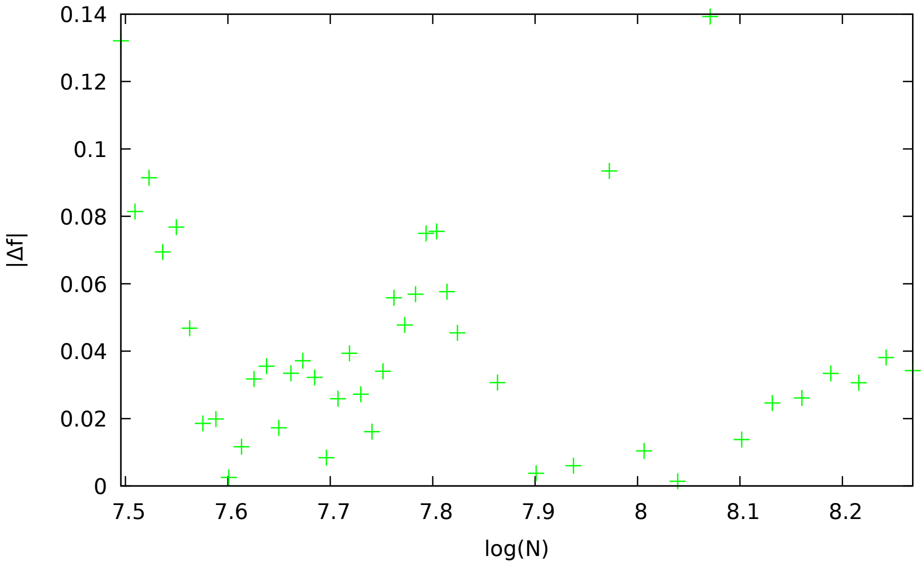

Figure 35: Absolute residuals: column by column. From 1800 to 3900

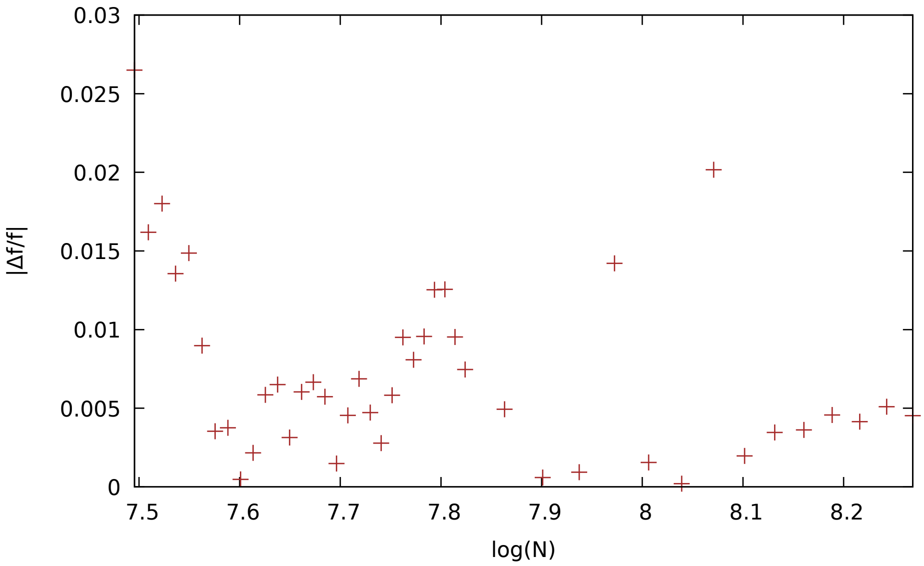

Figure 36: Relative residuals: column by colum. From 1800 to 3900

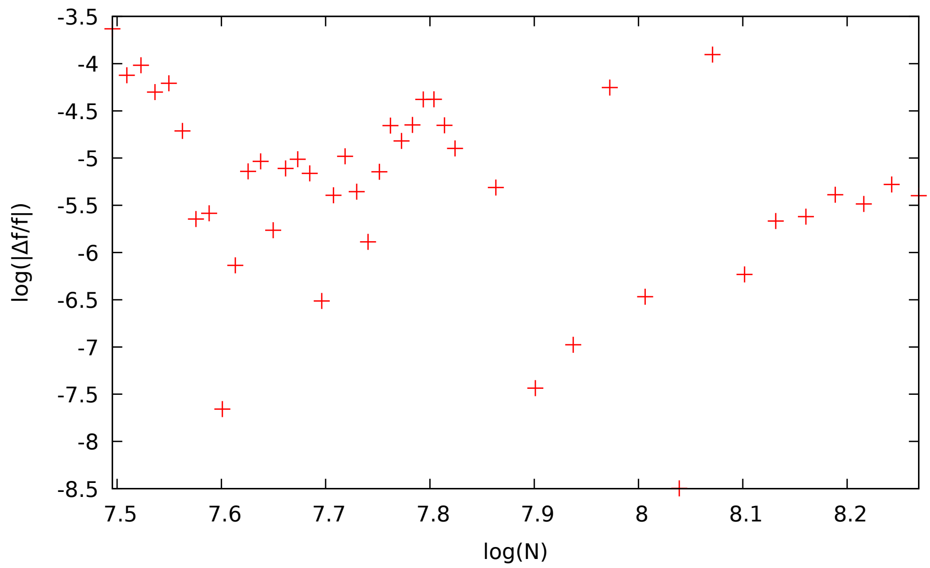

Figure 37: Log of relative residuals: column by column. From 1800 to 3900

Fit and residuals: intrinsic function method. From 1800 to 3900

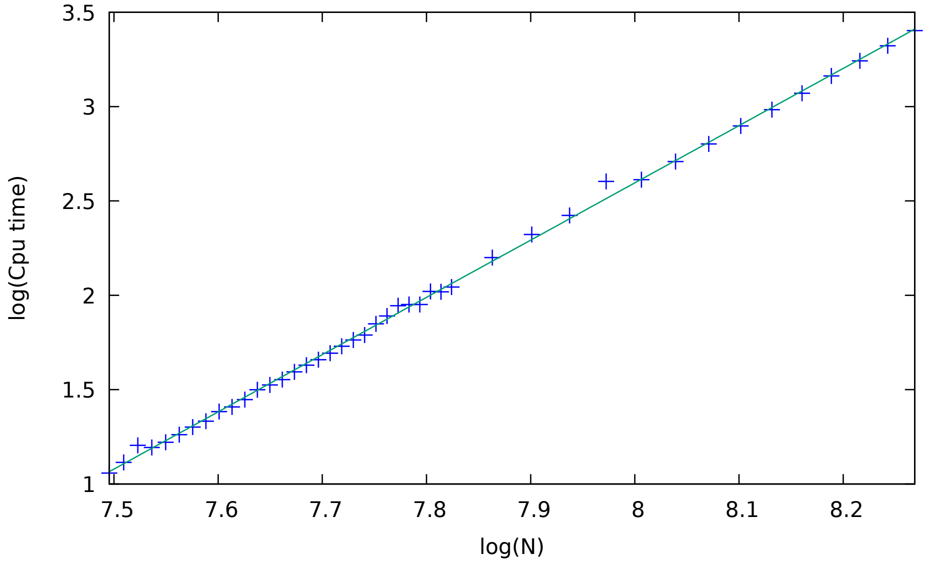

Figure 38: Linear fit: intrinsic function. From 1800 to 3900

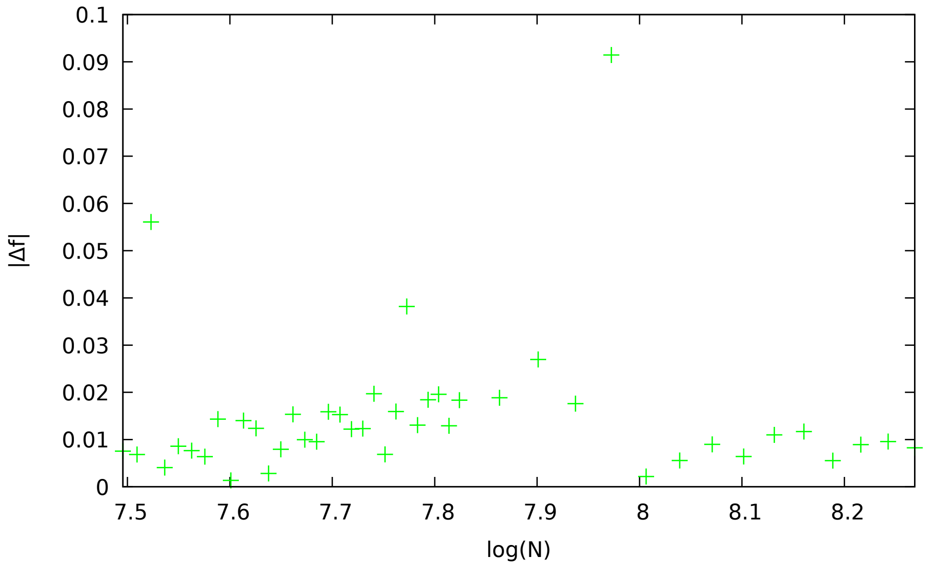

Figure 39: Absolute residuals: intrinsic function. From 1800 to 3900

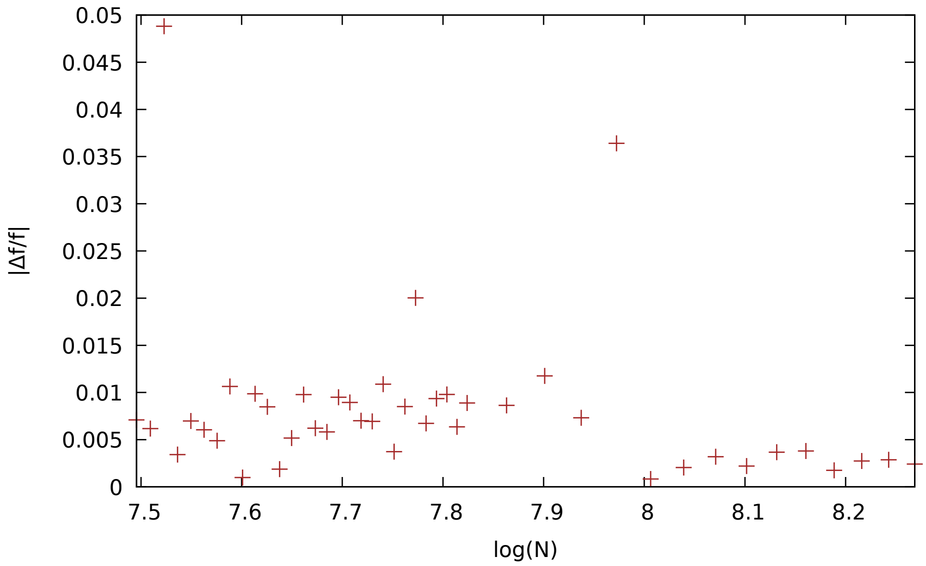

Figure 40: Relative residuals: intrinsic function. From 1800 to 3900

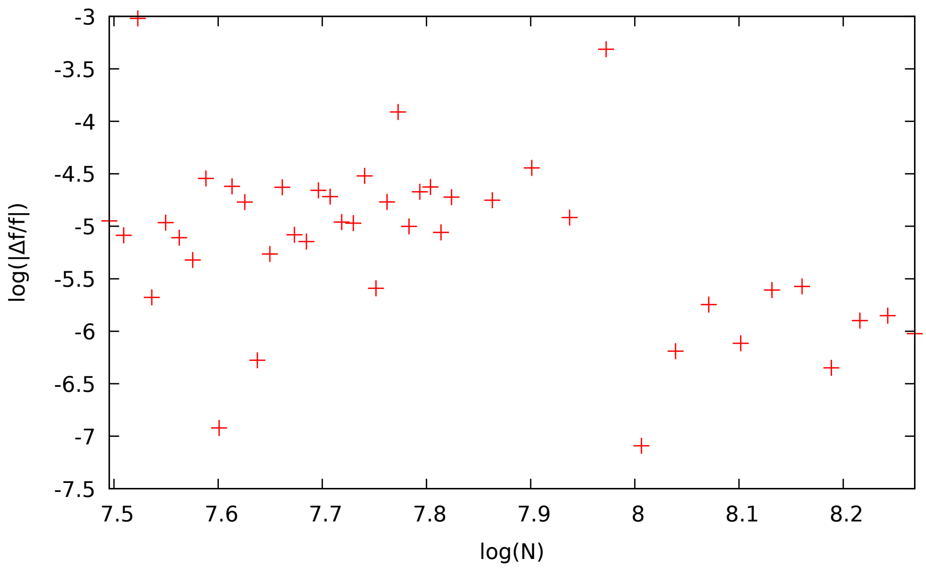

Figure 41: Log of relative residuals: intrinsic function. From 1800 to 3900

Plot of the three fitting function. From 1800 to 3900

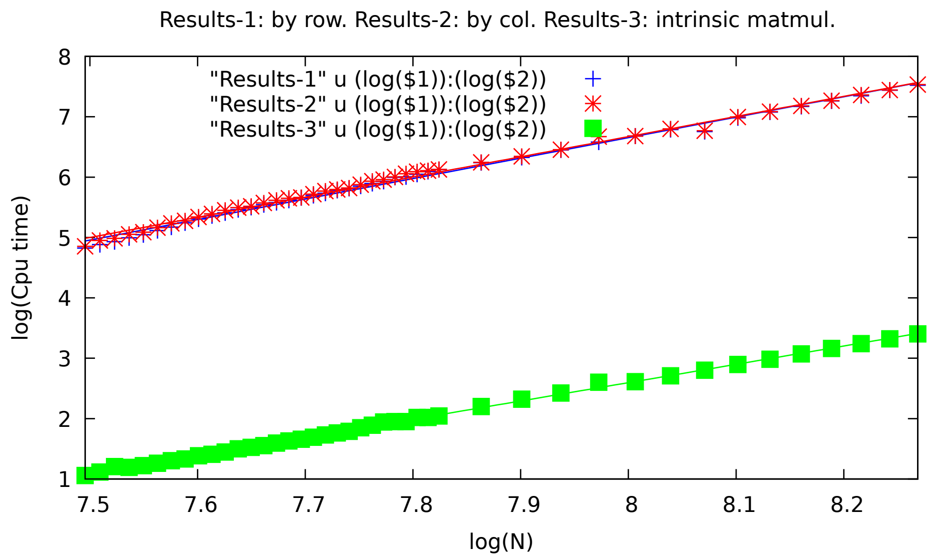

Figure 42: Plot of the three fitting function. From 1800 to 3900

Parameters of the three fits (from 1800 to 3900 (matrix dim)). The complete data are in the below subsection.

	Intercept	Slope
By row	-20.4728	3.39082
By col	-20.0473	3.33968
By matmul	-21.6692	3.03309

Self-Evaluation

Comments on the Results

The most striking features of the plot of the three sets of data is the similirity between the first and the second method (this is reasonable: the two methods are very similar) and the difference between those two and the third method, which is much faster than the other.

The liner fits of the first two methods on the whole range [100,3900] present the following trend: until some point between 7 and 7.5 (1100 and 1800 in matrix dimension) there is one linear trend, then there is a transient and then another linear trend, from 7.5 (more or less). This observation is motivated by tha fact that the absolute residuals, the relative residuals and the logarithm of relative residuals present a discontinuity in the so called transient zone. Moreover, the absolute residuals in the zone from 100 to the transient zone suggests that the points are aligned along a line which is incident with the fitting line, with the intersection around 6 (400 in matrix dimension, more or less). After the transient there is another descent, which recalls the one in the first zone. As a consequence, we can infer there is another alignement of the point in that high-dimension zone. As a consequence of those alignments, the slopes are very different from the theorical ones: around 3.8.

On the other hand, the analysis on the third method presents a good linear fit, supported by low absolute residuals, relative residuals and logarithms of relative residuals.

In all the three methods, in the relative residuals and in the logarithm of relative residuals we can see a small zone which is much higher than the others. In that smal zone the fitting function in near to 0, so the relative residuals and their logarithm are very high, respect to the others.

The analysis between 100 and 1400 shows us a good linear fit, for all the three methods, supported by low absolute residuals, relative residuals and logarithms of relative residuals. It is noticeable that the slopes are closer to the theorical 3 respect to the ones of the overall fit.

In the absolute residuals, the relative residuals and the logarithm of relative residuals of the fits from 1800 to 3900 we can see a discontinuity between 7.8 and 7.9. This is due to the fact that before 2500 the steps between dimensions wes 25, then 100. As a consequence, there are a lot more points before 2500. Nevertheless, the residuals are low and therefore the fit is good. On the other hand, the obtained slopes are different from the expected ones for the first two methods: 3.3 or 3.4 instead of 3.

What we have learnt:

- Writing a Python Script which communicates automatically with the terminal and Gnuplot, in addition to automatically executing programs.
- Performing via Gnuplot linear fits, plotting absolute residuals, relative residuals and their logarithm.
- Commenting and thinking on the results.
- Estimating the time needed for high-cost computations.

What can be done in the future:

- Understand why the points align along two different lines, one in the low-dimension zone and one in the high-dimension zone.

Data of the three fits: from 100 to 3900

****

```
Mon Oct 29 09:45:06 2018


FIT:    data read from "Results-1" u (log($1)):(log($2))
format = x:z
#datapoints = 111
residuals are weighted equally (unit weight)

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

iter      chisq       delta/lim  lambda   a             b
0 3.4755286224e+03   0.00e+00  5.10e+00    1.000000e+00   1.000000e+00
4 2.2068480154e+01  -3.12e-01  5.10e-04   -2.394167e+01   3.796996e+00

After 4 iterations the fit converged.
final sum of squares of residuals : 22.0685
rel. change during last iteration : -3.12165e-06

degrees of freedom    (FIT_NDF)                        : 109
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.449959
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.202463

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = -23.9417         +/- 0.3789       (1.583%)
b               = 3.797            +/- 0.05308      (1.398%)

correlation matrix of the fit parameters:
a      b
a               1.000
b              -0.994  1.000


*******************************************************************************
Mon Oct 29 09:45:06 2018


FIT:    data read from "Results-2" u (log($1)):(log($2))
format = x:z
#datapoints = 111
residuals are weighted equally (unit weight)

function used for fitting: g(x)
```

```
g(x)= c+d*x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   c             d
0 3.4109617706e+03   0.00e+00  5.10e+00    1.000000e+00   1.000000e+00
4 1.9350825095e+01  -3.55e-01  5.10e-04   -2.389430e+01   3.798111e+00

After 4 iterations the fit converged.
final sum of squares of residuals : 19.3508
rel. change during last iteration : -3.5468e-06

degrees of freedom    (FIT_NDF)                        : 109
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.421344
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.177531

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c               = -23.8943         +/- 0.3548       (1.485%)
d               = 3.79811          +/- 0.0497       (1.309%)

correlation matrix of the fit parameters:
c      d
c               1.000
d              -0.994  1.000


*************************************************************************
Mon Oct 29 09:45:06 2018


FIT:    data read from "Results-3" u (log($1)):(log($2))
format = x:z
#datapoints = 111
residuals are weighted equally (unit weight)

function used for fitting: h(x)
h(x)= e+f*x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   e             f
0 7.7673995350e+03   0.00e+00  5.10e+00    1.000000e+00   1.000000e+00
5 3.1049987943e-01  -5.61e-09  5.10e-05   -2.117031e+01   2.967751e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.3105
```

```
rel. change during last iteration : -5.61369e-14

degrees of freedom    (FIT_NDF)                        : 109
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0533725
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00284862

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
e               = -21.1703         +/- 0.04494      (0.2123%)
f               = 2.96775          +/- 0.006296     (0.2122%)

correlation matrix of the fit parameters:
e      f
e               1.000
f              -0.994  1.000
```

Data of the three fits: from 100 to 1400

```
*******************************************************************************
Wed Oct 31 09:33:48 2018


FIT:    data read from "Results-1" u (log($1)):(log($2))
format = x:z
#datapoints = 53
residuals are weighted equally (unit weight)

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

iter      chisq       delta/lim  lambda   a             b
0 2.8403361635e+03   0.00e+00  4.63e+00    1.000000e+00   1.000000e+00
5 3.5812761533e-02  -1.02e-06  4.63e-05   -1.957031e+01   3.079415e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.0358128
rel. change during last iteration : -1.02136e-11

degrees of freedom    (FIT_NDF)                        : 51
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0264993
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000702211
```

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = -19.5703         +/- 0.03495      (0.1786%)
b               = 3.07942          +/- 0.005399     (0.1753%)

correlation matrix of the fit parameters:
a      b
a               1.000
b              -0.995  1.000


*************************************************************************
Wed Oct 31 09:33:48 2018


FIT:    data read from "Results-2" u (log($1)):(log($2))
format = x:z
#datapoints = 53
residuals are weighted equally (unit weight)

function used for fitting: g(x)
g(x)= c+d*x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   c             d
0 2.7987669617e+03   0.00e+00  4.63e+00    1.000000e+00   1.000000e+00
5 2.5970853675e-01  -1.44e-07  4.63e-05   -1.979373e+01   3.123602e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.259709
rel. change during last iteration : -1.44256e-12

degrees of freedom    (FIT_NDF)                        : 51
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0713605
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00509232

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c               = -19.7937         +/- 0.09411      (0.4754%)
d               = 3.1236           +/- 0.01454      (0.4655%)

correlation matrix of the fit parameters:
c      d
c               1.000
```

```
d               -0.995  1.000


*******************************************************************************
Wed Oct 31 09:33:48 2018


FIT:    data read from "Results-3" u (log($1)):(log($2))
format = x:z
#datapoints = 53
residuals are weighted equally (unit weight)

function used for fitting: h(x)
h(x)= e+f*x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   e             f
0 4.8823868816e+03   0.00e+00  4.63e+00    1.000000e+00   1.000000e+00
5 2.5916307452e-01  -1.61e-07  4.63e-05   -2.103325e+01   2.945878e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.259163
rel. change during last iteration : -1.60795e-12

degrees of freedom    (FIT_NDF)                        : 51
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0712855
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00508163

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
e               = -21.0333         +/- 0.09401      (0.447%)
f               = 2.94588          +/- 0.01453      (0.4931%)

correlation matrix of the fit parameters:
e      f
e               1.000
f              -0.995  1.000
```

Data of the three fits: from 1800 to 3900

```
*******************************************************************************
Wed Oct 31 17:56:51 2018
```

```
FIT:    data read from "Results-1" u (log($1)):(log($2))
format = x:z
#datapoints = 43
residuals are weighted equally (unit weight)

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

iter      chisq       delta/lim  lambda   a             b
0 3.5393020278e+02   0.00e+00  5.56e+00    1.000000e+00   1.000000e+00
5 1.2662992118e-01  -6.01e-02  5.56e-05   -2.047284e+01   3.390820e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.12663
rel. change during last iteration : -6.01076e-07

degrees of freedom    (FIT_NDF)                        : 41
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0555746
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00308853

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = -20.4728         +/- 0.3003       (1.467%)
b               = 3.39082          +/- 0.03847      (1.135%)

correlation matrix of the fit parameters:
a      b
a               1.000
b              -1.000  1.000


*******************************************************************************
Wed Oct 31 17:56:51 2018


FIT:    data read from "Results-2" u (log($1)):(log($2))
format = x:z
#datapoints = 43
residuals are weighted equally (unit weight)

function used for fitting: g(x)
g(x)= c+d*x
```

```
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   c             d
0 3.4701109840e+02   0.00e+00  5.56e+00    1.000000e+00   1.000000e+00
5 1.1751538879e-01  -6.22e-02  5.56e-05   -2.004734e+01   3.339684e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.117515
rel. change during last iteration : -6.22253e-07

degrees of freedom    (FIT_NDF)                        : 41
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0535372
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00286623

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c               = -20.0473         +/- 0.2893       (1.443%)
d               = 3.33968          +/- 0.03706      (1.11%)

correlation matrix of the fit parameters:
c      d
c               1.000
d              -1.000  1.000


*******************************************************************************
Wed Oct 31 17:56:51 2018


FIT:    data read from "Results-3" u (log($1)):(log($2))
format = x:z
#datapoints = 43
residuals are weighted equally (unit weight)

function used for fitting: h(x)
h(x)= e+f*x
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   e             f
0 2.0010699954e+03   0.00e+00  5.56e+00    1.000000e+00   1.000000e+00
5 1.9261685541e-02  -4.38e-01  5.56e-05   -2.166919e+01   3.033094e+00

After 5 iterations the fit converged.
final sum of squares of residuals : 0.0192617
rel. change during last iteration : -4.38012e-06
```

```
degrees of freedom    (FIT_NDF)                        : 41
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0216748
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000469797

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
e               = -21.6692         +/- 0.1171       (0.5405%)
f               = 3.03309          +/- 0.015        (0.4947%)

correlation matrix of the fit parameters:
e      f
e               1.000
f              -1.000  1.000
```

Complete Code of the Program

Listing 6: Fortran code Ex.3 modified

```
        \brief \b Multiplication matrix-matrix and its
           computational time
        C>
        C>   =========== DOCUMENTATION ===========
        C>   Definition:
        C>  ===========
        C>
        C>       SUBROUTINE ROWROWMATMUL (nn,m1,m2,mris1,
           timetot1)
        C>
        C>        .. Scalar Arguments ..
        C>        INTEGER*2            nn
        C>        REAL                timetot1
        C>        ..
        C>       .. Array Arguments ..
        C>        INTEGER*2           m1(nn,nn)
        C>        INTEGER*2           m2(nn,nn)
        C>        INTEGER*2           mris1(nn,nn)
        C>       ..
        C>
        C>   Purpose
        C>   =======
        C>
        C>\details \b Purpose:
        C>\verbatim
```

```
C>
C> ROWROWMATMUL does a matrix-matrix multiplication
   between two matrices
C> given as input: m1 and m2. The resulting matrix is
    mris1. ROWROWMATMUL
C> deals with INTEGER*2 squared matrices, with
   numbers of rows and columns equal C> to nn. In the
    entry (i,j) of mris1 there is the scalar product
   between the C> i-th row of m1 and the j-th column
   of m2.
C> The  multiplication is row by row because while i
   is fixed j
C> goes from 1 to nn. In other words the j cycle is
   inside the i cycle.
C> Calling the function CPU_TIME at the beginning of
   that moltiplication, we assign the initial time to
    REAL variable start1. The same is done for the
   ending C> time, in REAL variable finish1. The
   total time timetot1 is the difference between the
   finish time and the ending time.

C>\endverbatim
C>
C>  Arguments:
C>  ==========
C>

C> \param[in] nn
C> \verbatim
C>          nn is INTEGER*2
C>          The dimension of the squared array m1, m2
    and mris1
C> \endverbatim
C> \param[in] m1
C> \verbatim
C>          m1 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[in] m2
C> \verbatim
C>          m2 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[out] m2
C> \verbatim
```

```
C>           mris1 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C>
C> \param[out] timetot1
C> \verbatim
C>           timetot1 is REAL
C> \endverbatim
C>
C>  Authors:
C> ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018
C>
C>
C>        SUBROUTINE COLCOLMATMUL (nn,m1,m2,mris2,
   timetot2)
C>
C>        .. Scalar Arguments ..
C>        INTEGER*2            nn
C>        REAL              timetot2
C>        ..
C>       .. Array Arguments ..
C>        INTEGER*2         m1(nn,nn)
C>        INTEGER*2         m2(nn,nn)
C>        INTEGER*2         mris2(nn,nn)
C>        ..
C>
C>  Purpose
C>  =======
C>
C>\details \b Purpose:
C>\verbatim
C>
C> COLCOLMATMUL does a matrix-matrix multiplication
   between two matrices
C> given as input: m1 and m2. The resulting matrix is
    mris2. COLCOLMATMUL
C> deals with INTEGER*2 squared matrices, with
   numbers of rows and columns equal C> to nn. In the
    entry (j,i) of mris1 there is the scalar product
   between the j-th row of m1 and the i-th column of
   m2.
```

```
C> The  multiplication is row by row because while i
   is fixed j
C> goes from 1 to nn. In other words the j cycle is
   inside the i cycle.
C> Calling the function CPU_TIME at the beginning of
   that moltiplication, we assign the initial time to
    REAL variable start2. The same is done for the
   ending time, in REAL variable finish2. The total
   time timetot2 is the difference between the finish
    time and the ending time.

C>\endverbatim
C>
C>  Arguments:
C>  ==========
C>

C> \param[in] nn
C> \verbatim
C>          nn is INTEGER*2
C>          The dimension of the squared array m1, m2
    and mris2
C> \endverbatim
C> \param[in] m1
C> \verbatim
C>          m1 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[in] m2
C> \verbatim
C>          m2 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[out] m2
C> \verbatim
C>          mris2 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C>
C> \param[out] timetot2
C> \verbatim
C>          timetot2 is REAL
C> \endverbatim
C>
C>  Authors:
```

```
C>  ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018
C>
C>
C>        SUBROUTINE MATMULINTRINSIC (nn,m1,m2,mris1,
   timetot1)
C>
C>        .. Scalar Arguments ..
C>        INTEGER*2            nn
C>        REAL              timetot3
C>        ..
C>        .. Array Arguments ..
C>        INTEGER*2           m1(nn,nn)
C>        INTEGER*2           m2(nn,nn)
C>        INTEGER*2           mris3(nn,nn)
C>        ..
C>
C>  Purpose
C>  =======
C>
C>\details \b Purpose:
C>\verbatim
C>
C> MATMULINTRINSIC does a matrix-matrix
   multiplication between two matrices
C> given as input: m1 and m2. The resulting matrix is
    mris1. MATMULINTRINSIC
C> deals with INTEGER*2 squared matrices, with
   numbers of rows and columns equal C> to nn.
C> The  multiplication is done via the intrinsic
   function mris3=matmul(m1,m2).
C> Calling the function CPU_TIME at the beginning of
   that moltiplication, we  assign the initial time
   to REAL variable start3. The same is done for the
   ending C> time, in REAL variable finish3. The
   total time timetot3 is the difference between the
   finish time and the ending time.

C>\endverbatim
C>
C> Arguments:
C>  ==========
```

```
C>

C> \param[in] nn
C> \verbatim
C>          nn*2 is INTEGER
C>          The dimension of the squared array m1, m2
    and mris3
C> \endverbatim
C> \param[in] m1
C> \verbatim
C>          m1 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[in] m2
C> \verbatim
C>          m2 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[out] m2
C> \verbatim
C>          mris3 is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C>
C> \param[out] timetot1
C> \verbatim
C>          timetot3 is REAL
C> \endverbatim
C>
C> Authors:
C>  ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018
C>
C>       SUBROUTINE CHECKDIM(nn,nncheck,debug)
C>
C>       .. Scalar Arguments ..
C>       INTEGER*2           nn
C>       INTEGER*2           nncheck
C>       LOGICAL             debug
C>       ..
C>       .. Array Arguments ..
C>       INTEGER*2         m(nn,nn)
```

```
C>        INTEGER*2          mcheck(nn,nn)
C>        ..
C>   Purpose
C>   =======
C>
C>\details \b Purpose:
C>\verbatim
C>
C> CHECKDIM checks if the dimension of the matrix are
     correct. Firstly checks
C> if the the dimension INTEGER*2 is above 10000. If
   it is so, it prints a WARNING
C> message because it takes too much time. Secondly,
   it checks if nn is inferior
C> to 1, printing a WARNING message. Lastly, it
   checks if nn is actually nncheck
C> as it should be. If it is not true, it prints a
   WARNING message, stating which
C> should be the dimension and the actual one. If
   everything goes well, it prints "okay:␣right␣
   dimensions␣matrices",
C> and the actual dimension.

C>\endverbatim
C>
C> Arguments:
C>  ==========
C>

C> \param[in] nn
C> \verbatim
C>          nn is INTEGER*2
C>          The dimension of the squared arrays
C> \endverbatim
C> \param[in] qq
C> \verbatim
C>          qq is INTEGER*2
C>          the number of the cycle, it should be nn=
   qq*100
C> \endverbatim
C>
C> \param[inout] debug
C> \verbatim
C>          debug is LOGICAL
C> \endverbatim
```

```
C>
C> Authors:
C>  ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018

C>       SUBROUTINE CHECKMAT(nn,m,mcheck,debug)
C>
C>       .. Scalar Arguments ..
C>       INTEGER*2             nn
C>       LOGICAL               debug
C>       ..
C>       .. Array Arguments ..
C>       INTEGER*2           m(nn,nn)
C>       INTEGER*2           mcheck(nn,nn)
C>       ..
C>  Purpose
C>
C>  Purpose
C>  =======
C>
C>\details \b Purpose:
C>\verbatim
C>
C> CHECKMAT checks if the two imput matrices m and
   mcheck are equal. This is done
C> via a do cycle which checks the equality between
   the two matrices element
C> by element. For each entry which is not equal, the
    internal variable INTEGER*4 accum
C> increseas by 1 (starting from 0). If accum in the
   end is bigger than 0,
C> a WARNING message states how much entries between
   the two matrices are
C> different. Otherwise, it is printed "Same␣matrices
   ".
C>\endverbatim
C>
C> Arguments:
C>  ==========
C>

C> \param[in] nn
```

```
C> \verbatim
C>          nn is INTEGER*2
C>          The dimension of the squared arrays
C> \endverbatim
C>
C> \param[in]
C> \verbatim
C>          m is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[in]
C> \verbatim
C>          mcheck is INTEGER*2 ARRAY (nn,nn)
C> \endverbatim
C>
C> \param[inout] debug
C> \verbatim
C>          debug is LOGICAL
C> \endverbatim
C>
C> Authors:
C>  ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018

C>
C>       PROGRAM test_performance_mulmat

C>  Purpose
C>  =======
C>
C>\details \b Purpose:
C>\verbatim
C>
C>This program implements matrix-matrix
   multiplication in three different method: row by
   row, column by column and by intrinsic function.
   For each method an apposite subroutine is called:
   ROWROWMATMUL, COLCOLMATMUL and MATMULINTRINSIC.
   For each method the CPU_TIME is computed. This is
   done for squared matrix, which dimension is taken
   as imput from a file, called "MatDimension.txt".
   The first matrix is such that at the entry (i,j)
```

```
          there is the value i+j, whilst the second matrix
          is such that at the at the entry (i,j) there is
          the value i*j. At the beginning, the matrices are
          allocated and in the ending are deallocated. The
          results are printed on three file, one for each
          method, called "Results-1" for the row by row
          method, "Results-2" for the col by col method and
          "Results-3" for the intrinsic function method. The
           format is the following: 4 figures for one
          integer, four spaces, and lastly a real number
          written in scientific notation, with nine
          significant figures.
C>
C> DEBUGGING of the program. This is controlled with
          a CHARACTER*1 variable called CHOICE. The
          programmer can choose to give the choice to do or
          not the debugging to the user keeping or changing
          the value of CHOICE. Keeping its default value "X"
          , the user can choose at the beginning of the
          program to do the debug, via a printed choice on
          the terminal.
C> If he says no, the logical variable DEBUG turns .
          FALSE. and all the debugging routines do not work.
C> If he says yes, the logical variable DEBUG turns .
          FALSE. and therefore all the debugging procedures
          below are executed. If one of the them reports an
          error, a warning message is printed.
C> Otherwise, if the programmer changes the value of
          CHOICE in "n", the user has not this choice and
          the debugging is not done (DEBUG is .FALSE.).
          Lastly, changing the value of CHOICE in "y", the
          user has not the choice and the debugging is done
          (DEBUG is .TRUE.).
C> The debugging is done printing various checkpoints
           throughout the program, in specific points: at
          the beginning of each cycle, initialization of the
           input matrices, calling the subroutines of the
          three methods, checking the resulting matrices.
C> Moreover the subroutines CHECKDIM and CHECKMAT are
           used a lot , interfaced via the operator CHECK.
          CHECKDIM is used every time a subroutine is called
          . CHECKMAT checks after the calling of the three
          method that the imput matrices are still the same
          (after each one). Moreover, it checks if the
          resulting matrices mris1, mris and mris3 are the
```

```
   same at the end of each cycle.
C> Lastly, error handling commands are added for the
   functions OPEN, WRITE, ALLOCATE and DEALLOCATE.
   They print which error the functions reports and
   its relative error message.
C> If one of the these procedures reports an error, a
    warning message is printed.

C>\endverbatim
C>  Authors:
C>  ========
C>
C> \author Univ. of Padua
C>
C> \date 23 October 2018

C>
   ================================================================

!MODULE MAT-MULTPLICATIONS
module matmultiplications

contains
subroutine rowrowmatmul(nn,m1,m2,mris1,timetot1)
!first method:row by row
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn):: mris1
real :: finish1, start1,timetot1
call cpu_time (start1)!initial time
do ii=1,nn
do jj=1,nn
do kk=1,nn
mris1(ii,jj)=mris1(ii,jj)+m1(ii,kk)*m2(kk,jj)
!SCALAR PRODUCT BTW
!ii-th ROW OF m1 AND j-th COLUMN OF m2
!IT GOES IN THE ENTRY (i,j) OF mris1
end do
end do
end do
call cpu_time (finish1)!finish time
timetot1=finish1-start1
end subroutine rowrowmatmul
```

```fortran
subroutine colcolmatmul(nn,m1,m2,mris2,timetot2)
!second method:col by col
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn) :: mris2
real :: finish2, start2,timetot2
call cpu_time (start2)!initial time
do ii=1,nn
do jj=1,nn
do kk=1,nn
mris2(jj,ii)=mris2(jj,ii)+m1(jj,kk)*m2(kk,ii)
!SCALAR PRODUCT BTW
!jj-th ROW OF m1 AND ii-th COLUMN OF m2
!IT GOES IN THE ENTRY (j,i) OF mris2
end do
end do
end do
call cpu_time (finish2)!finish time
timetot2=finish2-start2
end subroutine colcolmatmul


subroutine matmulintrinsic(nn,m1,m2,mris3,timetot3)
!third method: intrisic matmul
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn) :: mris3
real :: finish3, start3,timetot3

call cpu_time (start3)!initial time
mris3= matmul(m1,m2)
!matmul DOES A MATRIX-MATRIX MULTIPLICATION m1*m2
call cpu_time (finish3)!finish time
timetot3= finish3-start3
end subroutine matmulintrinsic

end module matmultiplications
```

```
!MODULE DEBUGGING
module debugging

interface check
module procedure checkdim,checkmat
end interface

contains
subroutine checkdim(nn,nncheck,debug)!checking
   dimensions nn
implicit none
integer*2 :: nn, nncheck
logical :: debug

if(debug.eqv..TRUE.) then

if(nn>10000) then !is the dim too large?
print*, "WARNING:␣too␣large␣dimension:"
print*, nn
else if(nn<1) then !is the dim minor than 1?
print*, "WARNING:␣dimension␣minor␣than␣1"
else if((nn-nncheck)>0.5) then !is the dim wrong?
print*, "the␣dimension␣is␣wrong,␣it␣should␣be:"
$                                   , nncheck
print*, "but␣is", nn
print*, "␣"
else
print*, "okay:␣right␣dimensions␣matrices",nn
end if
end if
end subroutine checkdim

subroutine checkmat(nn,m,mcheck,debug)
implicit none
!checking if the input matrixes are equal
integer*2, dimension(nn,nn) :: m, mcheck
integer*2 :: tt,ss,nn
integer*4 accum
logical debug

if(debug.eqv..TRUE.) then
accum=0
do tt=1,nn
do ss=1,nn
if(m(tt,ss)/=mcheck(tt,ss)) then
```

```
accum=accum+1
end if
end do
end do

if(accum>0) then
print*, "The␣two␣matrices␣have"
$                          , accum, "different␣entries"
else
print*, "Same␣matrices"
end if

end if
end subroutine checkmat

end module debugging


program test_performance_mulmat
!This program implements matrix-matrix multiplication
!in three different method: column by column, row by
   row
!and by intrinsic function. For each method the
   CPU_TIME
!is computed. This is done for squared matrix, from
!100x100 to 1000x1000, with a step=100.
!The results are printed on three different files,
   according
!to the used method.

!DECLARATION VARIABLES

use matmultiplications
use debugging
implicit none

integer*2, dimension(:,:), allocatable :: m1
integer*2, dimension(:,:), allocatable :: m2
integer*2, dimension(:,:), allocatable :: mcheck1,
   mcheck2
integer*2, dimension(:,:), allocatable :: mris1,
   mris2, mris3
integer*2  ii, jj, nn, nncheck
real :: ttime
real :: timetot1, timetot2, timetot3
```

```fortran
real :: t1,t2, accumtime, t3,t4
logical :: debug
character*1 :: choice
integer :: my_stat
character (256) :: my_msg
call cpu_time(t1)!INITIAL TIME ALL PROGRAM

accumtime=0.0
choice="n" !Initialization of choice variable
!if it is "X", it asks the debug
!otherwise the programmer can choose between
!doing the debug or not writing "y" or "n"


if(choice=="y") then
debug=.TRUE.
else if(choice=="n") then
debug=.FALSE.
else
print*, "Not␣understood"
stop
end if


!OPENING THE 3 "Result" FILE

!First method:row by row
open(unit = 90, file = "Results-1",
$        status = "unknown", access="append",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-Results-1␣failed␣with␣stat␣=␣'
$                              , my_stat, '␣msg␣=␣'//trim(
   my_msg)
end if

!Second method:col by col
open(unit =80, file = "Results-2",
$        status = "unknown", access="append",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-Results-2-␣failed␣with␣stat␣=␣'
```

```
$                         , my_stat, '␣msg␣=␣'//trim(
   my_msg)
end if

!Third method:intrinsic matmul
open(unit = 70, file = "Results-3",
$        status = "unknown", access="append",
$   iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-Results-3␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(
   my_msg)
end if


!HERE THE MATRIX DIMENSION IS TAKEN AS IMPUT FROM
!FILE "MatDimension.txt"

open(unit = 30, file = "MatDimension.txt",
$        status = "unknown",
$   iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-MatDimension-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(
   my_msg)
end if

read(30,*) nn
nncheck=nn

!ALLOCATION MATRICES
!DEBUG
if(debug.eqv..TRUE.) then!DEBUG
print*, "␣"
print*, "MATRICES␣OF␣ORDER␣", nn
print*, "␣"
end if

if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if
```

```
allocate(m1(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating m1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣m1␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
allocate(m2(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating m2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣m2␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
allocate(mris1(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris1␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(mris2(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris2␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
allocate(mris3(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris3
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris3␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg
   )
end if

allocate(mcheck1(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mcheck1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mcheck1␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
allocate(mcheck2(nn,nn), stat=my_stat, errmsg=my_msg)
```

```
!allocating mcheck2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mcheck2␣with␣stat␣=␣'
$                 , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if



!INITIALIZATION m1: in the entry (i,j)
!the value i+j is assigned
do ii=1,nn
do jj=1,nn
m1(ii,jj)=ii+jj
mcheck1(ii,jj)= m1(ii,jj)
end do
end do
!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "INITIALIZATION␣m1"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if


!INITIALIZATION m2, in the entry (i,j)
!the value i*j is assigned
do ii=1,nn
do jj=1,nn
m2(ii,jj)=ii*jj
mcheck2(ii,jj)= m2(ii,jj)
end do
end do

!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "INITIALIZATION␣m2"
end if
if(debug.eqv..TRUE.) then !checkdim
call check(nn,nncheck,debug)
end if
```

```
if(debug.eqv..TRUE.) then !check matrices
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if

!FIRST METHOD: row by row in resulting matrix


call rowrowmatmul(nn,m1,m2,mris1,timetot1)


write(90,"(I4,4X,E16.9)",iostat=my_stat, iomsg=my_msg
   )
$                 nn, timetot1

if(my_stat /= 0) then
print*, 'Write␣-Results-1-␣failed␣with␣stat␣=␣'
$                  , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time

!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "FIRST␣METHOD:␣row␣by␣row"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if

if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if

!SECOND METHOD: column by column in resulting matrix



call colcolmatmul(nn,m1,m2,mris2,timetot2)
```

```
write(80,"(I4,4X,E16.9)",iostat=my_stat, iomsg=my_msg
   )
$              nn, timetot2

if(my_stat /= 0) then
print*, 'Write␣-Results-2-␣failed␣with␣stat␣=␣'
$                  , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time

!DEBUG

if(debug.eqv..TRUE.) then
print*, "␣"
print*, "SECOND␣METHOD:␣col␣by␣col"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if
if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if


!THIRD METHOD: intrinsic function matmul


call matmulintrinsic(nn,m1,m2,mris3,timetot3)


write(70,"(I4,4X,E16.9)",iostat=my_stat, iomsg=my_msg
   )
$              nn, timetot3

if(my_stat /= 0) then
print*, 'Write␣-Results-3-␣failed␣with␣stat␣=␣'
$                  , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time

!DEBUG
```

```
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "THIRD␣METHOD:␣intrinsic␣matmul"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if
if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if

!the resulting matrices are different?
!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "CHECKING␣RESULTING␣MATRICES"
end if
if(debug.eqv..TRUE.) then!checkdim
print*, "Checking␣first␣and␣second␣method"
call check(nn,mris1,mris2,debug)
print*, "Checking␣third␣and␣second␣method"
call check(nn,mris2,mris3,debug)
print*, "Checking␣first␣and␣third␣method"
call check(nn,mris1,mris3,debug)
end if


!DEALLOCATION matrices:

deallocate(m1, stat=my_stat, errmsg=my_msg)
!deallocating m1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣m1␣with␣stat␣=␣'
$                        , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if


deallocate(m2, stat=my_stat, errmsg=my_msg)
!deallocating m2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣m2␣with␣stat␣=␣'
```

```
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if

deallocate(mris1, stat=my_stat, errmsg=my_msg)
!deallocating mris1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris1␣with␣stat␣=␣'
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if
deallocate(mris2, stat=my_stat, errmsg=my_msg)
!deallocating mris2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris2␣with␣stat␣=␣'
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if

deallocate(mris3, stat=my_stat, errmsg=my_msg)
!deallocating mris3
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris3␣with␣stat␣=␣'
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if

deallocate(mcheck1, stat=my_stat, errmsg=my_msg)
!deallocating mcheck1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mcheck1␣with␣stat␣=␣'
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if

deallocate(mcheck2, stat=my_stat, errmsg=my_msg)
!deallocating mcheck2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mcheck2␣with␣stat␣=␣'
$                              , my_stat, '␣and␣msg␣=␣'//
   trim(my_msg)
end if


stop
```

```
        end program test_performance_mulmat
```