bench Python package, User's Guide[*]

version 0.2.0

François Cuvelier[†]

December 23, 2019

Abstract

The bench Python package allows to benchmark functions and much more

[*]LaTeX manual, revision 0.2.0, compiled with Python 3.7.6, and packages `fc-bench`[0.2.0], `fc-tools`[0.0.24],

[†]LAGA, UMR 7539, CNRS, Université Paris 13 - Sorbonne Paris Cité, Université Paris 8, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr

0 Contents

1 Introduction

The fc bench Python package aims to perform simultaneous benchmarks of several functions performing the same tasks but implemented in different ways.

We will illustrate its possibilities on an example. This one will focus on different ways of coding the Lagrange interpolation polynomial. We first recall some generalities about this polynomial.

Let `X` and `Y` be 1-by-($\mathtt{n}+1$) arrays where no two `X(j)` are the same. The Lagrange interpolating polynomial is the polynomial $P(t)$ of degree $\leqslant \mathtt{n}$ that passes through the ($\mathtt{n}+1$) points (`X(j)`, `Y(j)`) and is given by

$$P(t) = \sum_{j=1}^{\mathtt{n}+1} Y(j) \prod_{k=1,k\neq j} \frac{t - \mathtt{X(k)}}{\mathtt{X(j)} - \mathtt{X(k)}}.$$

Three different functions have been implemented to compute this polynomial. They all have the same header given by

```
def fun(X,Y,x):
```

where `x` is a 1-by-`m` array and the returned value `y` is a 1-by-`m` so that

$$\mathtt{y(i)} = P(\mathtt{x(i)}).$$

These functions are

- `fc_bench.demos.Lagrange`, a simplistic writing;
- `fc_bench.demos.lagint`, an optimized writing ;
- `fc_bench.demos.scipyLagrange`, using the `barycentric_interpolate` function of the `scipy.interpolate` module.

Their source codes are in file `demos.py` of the package. The last function needed the `scipy` package [2] to be installed.

Firstly we give a complete script in Listing 1 with its displayed output. Then we quickly give some explanations on how to use the fcbench package.

Listing 1: : `fc_bench.demos.bench_Lagrange00` function

```
def bench_Lagrange00():
  import fc_bench
  import numpy as np
  Lfun=[Lagrange,lagint,scipyLagrange]
  In=[[3,100],[5,100],[7,100],[11,100],[3,500],[5,500],[7,500],[11,500]]
  fc_bench.bench(Lfun,setLagrange00,In,labelsinfo=True)
```

Output

```
#---------------------------------------------------------------------------
#   computer: cosmos-ubuntu-18-04
#     system: Ubuntu 18.04.3 LTS (x86_64)
#  processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#        RAM: 62.6 Go
#   software: Python
#    release: 3.7.6
#---------------------------------------------------------------------------
#---------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0],      Lagrange: fc_bench.demos.Lagrange
#  fun[1],        lagint: fc_bench.demos.lagint
#  fun[2], scipyLagrange: fc_bench.demos.scipyLagrange
#---------------------------------------------------------------------------
# Comparative functions:
#   For each comparative function:
#     - 1st input parameter is the output of fun[0]
#     - 2nd input parameter is the output of fun[i]
#---------------------------------------------------------------------------
#date:2019-12-23_08-02-20
#nbruns:5
#numpy:                              f8         f8               f8
#format:    {:5}     {:5}     {:11.3f}    {:9.3f}         {:16.3f}
#labels:        m        n  Lagrange(s)  lagint(s)  scipyLagrange(s)
              100        3        0.010      0.012             0.000
              100        5        0.015      0.018             0.000
              100        7        0.020      0.024             0.000
              100       11        0.034      0.036             0.000
              500        3        0.046      0.061             0.000
              500        5        0.073      0.092             0.000
              500        7        0.103      0.122             0.000
              500       11        0.169      0.183             0.000
```

To run benchmarks, the main tool is `fc_bench.bench` function described in section 3 and with basic syntax:

`fc_bench.bench(Lfun,setfun,In)`

So one has to set the three input datas.

- `Lfun` is the list of functions to be benchmarked. So in our example we have:

  `Lfun=[Lagrange,lagint,scipyLagrange]`

- `In` is used to set `m` (number of interpolate values) and `n` (degree of the interpolating polynomial) values to produce one row on the printed output. One has `n=In(k,1)` and `m=In(k,2)`, for each `k` $\in [\![1, \texttt{size(In,1)}]\!]$. For example, we can take:

  `In=[[3,100],[5,100],[7,100],[11,100],[3,500],[5,500],[7,500],[11,500]]`

- `setfun` is a function handle. For this example, the corresponding function is called `setLagrange00`. The prototype of this function is imposed:

  `def setLagrange00(datas,verbose,**kwargs)`

  and returns a tuple with 3 elements:

  `(Inputs,bDs,Errors)`

  The `datas` parameter is a `[n,m]` value (given by `In[k]` for each benchmark). The returned `Inputs` (tuple) contains all the inputs of the Lagrange functions in the same order: `Inputs=(X,Y,x)`
  The returned `bDs` (`bdata` list) contains the first columns of the printed result. In this example, given in Listing 2, we choose to print in first column the `m` values and in second column the `n` values. The last returned `Errors` (function or list of functions) is used to compute error(s) from outputs of benchmarked functions. In this example `Errors` is an empty list.

```
def setLagrange00(datas,verbose,**kwargs):
  from fc_bench.bench import bData
  n=datas[0] # degree of the interpolating polynomial
  m=datas[1] # number of interpolate values
  a=0;b=2*np.pi
  X=np.linspace(a,b,n+1)
  Y=np.cos(X)
  x=a+(b-a)*np.random.rand(m)
  Inputs=(X,Y,x)
  bDs=[]
  bDs.append(bData('{:>8}'.format('m'),m,'{:5}')) # 1st column
  bDs.append(bData('{:>8}'.format('n'),n,'{:5}')) # 2nd column
  Errors=[]
  return (Inputs,bDs,Errors)
```

Listing 2: `fc_bench.demos.setLagrange00` function

We now propose a slightly more elaborate version of the initialization function that allows to display some informations and to choose certain parameters when generating inputs datas. This new version named `fc_bench.demos.setLagrange` is given in Listing 3. A complete script is given in Listing 4 with its displayed output. In this script some options of the `fc_bench.bench` function are used `'error'`, `'info'`, `'labelsinfo'`, jointly with those of the `fc_bench.demos.setLagrange`: `'a'`, `'b'` and `'fun'`. One must be careful not to take as an option name for the initialization function one of those used in `fc_bench.bench` function. More details are given in section 3.

```
def setLagrange(datas,verbose,**kwargs):
  import fc_tools
  a=kwargs.get('a',0)
  b=kwargs.get('b',2*np.pi)
  sfun=kwargs.get('fun','lambda x: np.cos(x)')
  fun=eval(sfun)
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  n=datas[0] # degree of the interpolating polynomial
  m=datas[1] # number of interpolate values
  X=np.linspace(a,b,n+1)
  Y=fun(X)
  x=a+(b-a)*np.random.rand(m)
  Error=lambda y: np.linalg.norm(y-fun(x),np.inf)
  if verbose:
    Print('# Setting inputs of Lagrange polynomial functions:
       y=LAGRANGE(X,Y,x)')
    Print('# where X is numpy.linspace(a,b,n+1), Y=fun(X) and x is random
       values on [a,b]')
    Print('#   n is the order of the Lagrange polynomial')
    Print('#   fun function is: %s'%sfun)
    Print('#   [a,b]=[%g,%g]'%(a,b))
    Print('#   X: (n+1,) numpy array')
    Print('#   Y: (n+1,) numpy array')
    Print('#   x: (m,)   numpy array')
    Print('#   Error[i] computed with fun[i] output:')
    Print('#     %s'%fc_tools.others.func2str(Error))

  bDs=[]
  bDs.append(bData('{:>5}'.format('m'),m,'{:>5}'))
  bDs.append(bData('{:>5}'.format('n'),n,'{:>5}'))
  return ((X,Y,x),bDs,Error)
```
Listing 3: `fc_bench.demos.setLagrange` function

Listing 4: : `fc_bench.demos.bench_Lagrange` function

```
def bench_Lagrange():
  import fc_bench
  import numpy as np
  Lfun=[Lagrange,lagint]
  names=['Lag','lagint']
  compfun=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  n=[3,5,7,11]; m=[100,500];
  N,M=np.meshgrid(n,m)
  In=np.vstack((N.flatten(),M.flatten())).T
  fc_bench.bench(Lfun,setLagrange,In,compfun=compfun, names=names,
     info=False,labelsinfo=True, a=-1,b=1,fun='lambda x: np.sin(x)')
```

Output

```
#------------------------------------------------------------------------------
# Setting inputs of Lagrange polynomial functions: y=LAGRANGE(X,Y,x)
# where X is numpy.linspace(a,b,n+1), Y=fun(X) and x is random values on [a,b]
#   n is the order of the Lagrange polynomial
#   fun function is: lambda x: np.sin(x)
#   [a,b]=[-1,1]
#   X: (n+1,) numpy array
#   Y: (n+1,) numpy array
#   x: (m,)  numpy array
#   Error[i] computed with fun[i] output:
#     lambda y: np.linalg.norm(y-fun(x),np.inf)
#------------------------------------------------------------------------------
# Benchmarking functions:
#   fun[0],          Lag: fc_bench.demos.Lagrange
#   fun[1],       lagint: fc_bench.demos.lagint
#------------------------------------------------------------------------------
# Comparative functions:
#   comp[i-1,0], compares outputs of fun[0] and fun[i]
#       lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
#    For each comparative function:
#      - 1st input parameter is the output of fun[0]
#      - 2nd input parameter is the output of fun[i]
#------------------------------------------------------------------------------
#date:2019-12-23_08-02-30
#nbruns:5
#numpy:    <i8   <i8      f8          f8         f8          f8         f8
#format: {:>5} {:>5} {:6.3f}   {:10.3e}    {:9.3f}   {:10.3e}    {:9.3e}
#labels:     m     n  Lag(s)    Error[0]  lagint(s)    Error[1]    comp[0]
           100     3   0.010   1.218e-03      0.012   1.218e-03  2.220e-16
           100     5   0.015   1.161e-05      0.018   1.161e-05  4.441e-16
           100     7   0.020   6.970e-08      0.024   6.970e-08  5.551e-16
           100    11   0.033   8.762e-13      0.037   8.757e-13  2.998e-15
           500     3   0.045   1.218e-03      0.061   1.218e-03  3.331e-16
           500     5   0.071   1.163e-05      0.091   1.163e-05  5.551e-16
           500     7   0.101   7.005e-08      0.122   7.005e-08  6.661e-16
           500    11   0.167   8.746e-13      0.182   8.749e-13  3.997e-15
```

2 Installation

This toolbox was tested on various OS with Python releases (from python.org):

Linux	**2.7.17**	**3.5.9**	**3.6.10**	**3.7.6**	**3.8.1**
CentOS 7.7.1908	✓	✓	✓	✓	✓
Debian 9.11	✓	✓	✓	✓	✓
Fedora 29	✓	✓	✓	✓	✓
OpenSUSE Leap 15.0	✓	✓	✓	✓	✓
Ubuntu 18.04.3 LTS	✓	✓	✓	✓	✓

Apple Mac OS X	**2.7.17**	**3.5.4**	**3.6.8**	**3.7.6**	**3.8.1**
MacOS High Sierra 10.13.6	✓	✓	✓	✓	✓
MacOS Mojave 10.14.4	✓	✓	✓	✓	✓

Microsoft Windows	**2.7.17**	**3.5.4**	**3.6.8**	**3.7.6**	**3.8.1**
Windows 10 (1909)	✓	✓	✓	✓	✓

Installation : The fcbench Python package can be installed by using **pip** (or **pip3**) command [1].

- For an installation which isolated to the current user, one can do:

  ```
  $ pip install -U --user fc_bench
  ```

- For an installation for all users, one can do:

  ```
  $ sudo pip install -U fc_bench
  ```

An other way is to download the required archive and to make installation from the downloaded file.

- For an installation which isolated to the current user, one can do:

  ```
  $ pip install <PATH_TO_FOLDER>/fc_bench-<VERSION>.tar.gz --user -U
  ```

  where `<PATH_TO_FOLDER>` will be replaced by the path to the saved archive and `<VERSION>` by the version of the archive.

- For an installation for all users, one can do:

  ```
  $ sudo pip install <PATH_TO_FOLDER>/fc_bench-<VERSION>.tar.gz -U
  ```

Uninstall : To uninstall this package, you only have to execute one of these commands depending on the type of installation performed

```
$ pip uninstall fc_bench
```

or

```
$ sudo pip uninstall fc_bench
```

3 `fc_bench.bench` function

The `fc_bench.bench` function run benchmark

Syntaxe

```
fc_bench.bench(Lfun,setfun,In )
fc_bench.bench(Lfun,setfun,In, key=value, ...)
R=fc_bench.bench(Lfun,setfun,In)
R=fc_bench.bench(Lfun,setfun,In, key=value,
    ...)
```

Description

`fc_bench.bench(Lfun, setfun, In)`

Runs benchmark for each function given in the list `Lfun`. So there are N functions `Lfun[i]`, for `i` $\in [\![0,$ `N` $[\![$. The function `setfun` is used to set input datas to these functions. There is its imposed syntax:

```
def setfun(datas,verbose,**kwargs):
  ...
  return (Inputs,bDs,Errors)
```

The `In` input variable is used to run `n` benchmarks of the functions contained in `Lfun`. For the `k`-th benchmark, `k` $\in [\![0,$ `n` $[\![$, the `setfun` is used with first parameter `datas` given as follows:

- if `In` is a list, then `n=len(In)` and `datas=In[k]`;
- if `In` is a Numpy array, then `n=In.shape[0]` and `datas=In[k]`.

Computational time in second of each function in `Lfun` is evaluated by `time.time()` function:

```
t=time.time(); out=Lfun[i]( *Inputs ); tcpu=time.time()-t
```

where `Inputs` is given by

```
Inputs,Bdatas,Errors=setfun(datas,verbose,**kwargs)
```

`fc_bench.bench(Lfun, setfun, In, key=value, ...)`

Some optional `key` are available:

- `names`, set the names that will be displayed during the benchmarks to name each of the functions of `Lfun`. By default `value` is `None` and all the names are guessed from the functions of `Lfun`. Otherwise, `value` is a list with same length as `Lfun` such that `value[i]` is the string name associated with `Lfun[i]` function. If `value[i]` is the empty string, then the name is guessed from the function `Lfun[i]`.
- `nbruns`, to set number of benchmark runs for each case and the mean of computational times is taken. Default `value` is `5`. In fact, `value+2` benchmarks are executed and the two worst are forgotten (see `fc_bench.mean_run` function)

- `comment` , string or list of strings displayed before running the benchmarks. If `value` is a list of strings, after printing the `value[i]`, a line break is performed.
- `info` , if `value` is `True`(default), some informations on the computer and the system are displayed.
- `labelsinfo` , if `value` is `True`, some informations on the labels of the columns are automaticaly displayed. Default is `False`.
- `savefile` , if `value` is a not empty string, then displayed results are saved in directory `benchs` with `value` as filename. One can used `savedir` option to change the directory.
- `savedir` , if `value` is a not empty string, then when using `savefile` , the file is saved in directory `value`.
- `before` , `value` is a list of size 0 or `N`. if not empty, `value[i]` is None or a list of `bdata` objects. `value[i]` is used to set b_i columns datas **before** printing the `Lfun[i]` cputime column with $b_i =$ `len(value[i])`. These columns datas are computed from the output of the `Lfun[i]` benchmarked function.
- `after` , as `before` option except that the $a_i =$ `len(value[i])` columns datas are printed **after** the `Lfun[i]` cputime column.
- `compfun` , `value` is a function or a list of functions. This option can be used to display comparison between outputs of reference benchmarked function `Lfun[0]` and the others `Lfun[i]`. Each function must return a scalar.

In Figure 1, we represent how columns are constructed by the `fc_bench.bench` function.

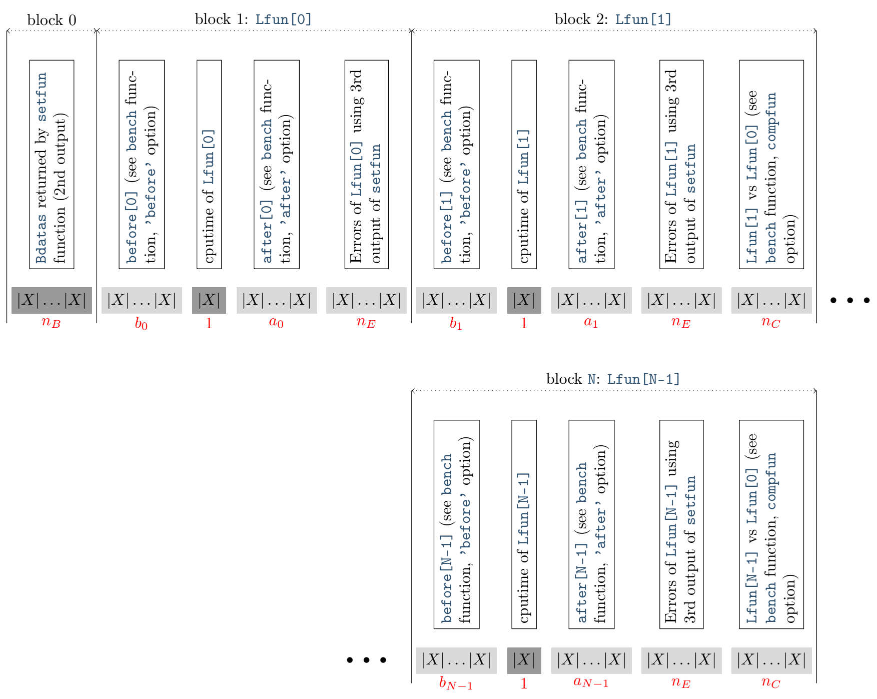
block 0
block 1: Lfun[0]
block 2: Lfun[1]
Bdatas returned by setfun function (2nd output)
before[0] (see bench function, 'before' option)
cputime of Lfun[0]
after[0] (see bench function, 'after' option)
Errors of Lfun[0] using 3rd output of setfun
before[1] (see bench function, 'before' option)
cputime of Lfun[1]
after[1] (see bench function, 'after' option)
Errors of Lfun[1] using 3rd output of setfun
Lfun[1] vs Lfun[0] (see bench function, compfun option)
$|X|\dots|X|$
$|X|$
n_B
b_0
1
a_0
n_E
b_1
a_1
n_C
block N: Lfun[N-1]
before[N-1] (see bench function, 'before' option)
cputime of Lfun[N-1]
after[N-1] (see bench function, 'after' option)
Errors of Lfun[N-1] using 3rd output of setfun
Lfun[N-1] vs Lfun[0] (see bench function, compfun option)
b_{N-1}
a_{N-1}

Figure 1: Description of columns of the bench. Each `|X|` represents one column. Columns with light gray background are optionals. Below each subblock (boxes with gray background), the number of columns is given.

3.1 `bData` object

The `bData` object is used to described and print the columns of the `bench` output. Its contructor is:

Syntaxe

```
bData(name,value,sformat,key=value)
```

where `name` is the label of the column which appears in line `#labels: ...` of the `bench` output, `value` is the corresponding value to print (it can be a numerical value or a function) and `sformat` is the format used for formatting the column by using the `str.format()` method. The string print is `sformat.format(value)`. The `strlen` optional key is used to set the minimum number of characters printed (default `0`) otherwise this number is automaticaly computed.
The `numpy` optional key is used to set the column which appears in line `#numpy: ...` of the `bench` output: its a `str` object (default `''`) that can be `'i4'` for int32, `'i8'` for int64, `'f8'` for float32, ... (see `numpy.dtype`, for example `np.dtype(np.int16).str` returns `'<i2'`

4 Examples

4.1 Matricial product examples

Let `X` be a `(m,n)` numpy array and `Y` be a `(n,p)` numpy array. We want to measure efficiency of the matricial product by using functions that we have written or `numpy.matmul(X,Y)` (function version) or `X@Y` (operator version for Python >= 3.5) with various values of `m`, `n` and `p`. When using the fc bench package with Python version < 3.5, efficiency is mesured only for the function `numpy.matmul` and the ones we wrote.

4.1.1 Square matrices: `fc_bench.demos.bench_MatProd00` function

Let `m` = `n` = `p`. As a first step, we want to measure the performance of the function `numpy.matmul(X,Y)` .

```
def setMatProd00(datas,verbose,**kwargs):
  from fc_bench.bench import bData
  m=datas
  X=np.random.randn(m,m)
  Y=np.random.randn(m,m)
  bDs=[bData('{:>8}'.format('m'),m,'{:>8}')]
  return ((X,Y),bDs,[])
```

Listing 5: `fc_bench.demos.setMatProd00` function

The `fc_bench.demos.setMatProd00` function given in Listing 5 is used in `fc_bench.demos.bench_MatProd00` function (file `demos.py` of the package directory)

Listing 6: : `fc_bench.demos.bench_MatProd00` function

```
def bench_MatProd00():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul]
  In=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd00,In)
```

Output

```
#---------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#---------------------------------------------------------------------------
#---------------------------------------------------------------------------
#date:2019-12-23_08-02-38
#nbruns:5
#numpy:       <i8         f8
#format:    {:>8}    {:9.3f}
#labels:        m   matmul(s)
              500      0.004
             1000      0.007
             1500      0.018
             2000      0.038
             2500      0.074
             3000      0.120
             3500      0.181
             4000      0.256
```

4.1.2 Square matrices: `fc_bench.demos.bench_MatProd01` function

Let $\mathtt{m} = \mathtt{n} = \mathtt{p}$.
Some improvements are made to the `fc_bench.demos.setMatProd00` function (Listing 5) by using `verbose` flag to display some information: this gives the `fc_bench.demos.setMatProd01` function presented in Listing 7. From the previous example, we also add the use of the options `comment` and `savefile` to the function `fc_bench.bench`: this gives the function `fc_bench.demos.bench_MatProd01` presented in Listing 8 that uses the `fc_bench.demos.setMatProd01` function.

```
def setMatProd01(datas,verbose,**kwargs):
  from fc_bench.bench import bData
  m=datas
  X=np.random.randn(m,m)
  Y=np.random.randn(m,m)
  if verbose:
    print('# 1st input parameter: (m,m) Numpy array')
    print('# 2nd input parameter: (m,m) Numpy array')
  bDs=[bData('{:>8}'.format('m'),m,'{:>8}')]
  return ((X,Y),bDs,[])
```

Listing 7: `fc_bench.demos.setMatProd01` function

Listing 8: : `fc_bench.demos.bench_MatProd01` function

```
def bench_MatProd01():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul]
  comment=['# Benchmarking function numpy.matmul',
           '# where X and Y are (m,m) Numpy arrays']
  In=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd01,In, comment=comment,
     savefile='MatProd01.out')
```

Output

```
#-----------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#-----------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#-----------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#-----------------------------------------------------------------------------
#benchfile: benchs/MatProd01.out
#date:2019-12-23_08-02-47
#nbruns:5
#numpy:       <i8         f8
#format:    {:>8}    {:9.3f}
#labels:        m  matmul(s)
              500      0.003
             1000      0.008
             1500      0.018
             2000      0.037
             2500      0.075
             3000      0.120
             3500      0.181
             4000      0.257
```

As we can see information print from the `fc_bench.demos.setMatProd01` function are missing in output file `benchs/MadProd01.out`. We will see how to print them also in the output file.

4.1.3 Square matrices: `fc_bench.demos.bench_MatProd02` function

Let `m` = `n` = `p`.
To have information print from the `fc_bench.demos.setMatProd01` also saved in the output file we just add the following line:

```
Print=kwargs.get('Print',lambda s: print(s))
```

This gives the `fc_bench.demos.setMatProd02` function presented in Listing 9. This function is then used in `fc_bench.demos.bench_MatProd02` given in Listing 10.

```
def setMatProd02(m,verbose,**kwargs):
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  X=np.random.randn(m,m)
  Y=np.random.randn(m,m)
  if verbose:
    Print('# 1st input parameter: (m,m) Numpy array')
    Print('# 2nd input parameter: (m,m) Numpy array')
  bDs=[bData('{:>8}'.format('m'), m, '{:>8}')]
  return ((X,Y),bDs,[])
```

Listing 9: `fc_bench.demos.setMatProd02` function

Listing 10: : `fc_bench.demos.bench_MatProd02` function

```
def bench_MatProd02():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul]
  comment=['# Benchmarking function numpy.matmul',
           '# where X and Y are (m,m) Numpy arrays']
  In=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd02,In, comment=comment,
     savefile='MatProd02')
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#benchfile: benchs/MatProd02.out
#date:2019-12-23_08-02-56
#nbruns:5
#numpy:       <i8        f8
#format:    {:>8}    {:9.3f}
#labels:        m   matmul(s)
              500      0.004
             1000      0.007
             1500      0.018
             2000      0.038
             2500      0.076
             3000      0.121
             3500      0.181
             4000      0.255
```

4.1.4 Square matrices: fc_bench.demos.bench_MatProd03 and 04 functions

Let m = n = p.
We want to compare computationnal times between the `numpy.matmul(X,Y)` function, the `X@Y` command (@ operator is defined since version 3.5 of Python) and the `fc_bench.demos.matprod01` function given in Listing 11.

```
def matprod01(A,B):
  import numpy as np
  (n,m)=A.shape
  (p,q)=B.shape
  assert m==p,'shapes %s and %s not aligned: %d (dim 1) != %d (dim
     0)'%(str(A.shape),str(B.shape),A.shape[1],B.shape[0])
  C=np.zeros((n,q))
  for i in np.arange(n):
    for j in np.arange(q):
      for k in np.arange(m):
        C[i,j]+=A[i,k]*B[k,j]
  return C
```

Listing 11: fc_bench.demos.matprod01 function

The `fc_bench.demos.setMatProd02` function given in Listing 9 is used in `fc_bench.demos.bench_MatProd03` function (file `demos_op.py`).

Listing 12: : fc_bench.demos.bench_MatProd03 function

```
def bench_MatProd03():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul,lambda X,Y: X@Y,fc_bench.demos.matprod01]
  comment=['# Benchmarking function numpy.matmul',
           '# where X and Y are m-by-m Numpy arrays']
  In=np.arange(50,201,50)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,In, comment=comment)
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are m-by-m Numpy arrays
#------------------------------------------------------------------------------
#   computer: cosmos-ubuntu-18-04
#     system: Ubuntu 18.04.3 LTS (x86_64)
#  processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#        RAM: 62.6 Go
#   software: Python
#    release: 3.7.6
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#date:2019-12-23_08-03-05
#nbruns:5
#numpy:      <i8        f8           f8            f8
#format:    {:>8}    {:9.3f}     {:11.3f}      {:12.3f}
#labels:        m   matmul(s)  <lambda>(s)  matprod01(s)
               50      0.000        0.000         0.089
              100      0.000        0.000         0.689
              150      0.000        0.000         2.374
              200      0.000        0.000         5.398
```

As the second function in `Lfun` has no name, the guess name is `<lambda>`. One can set a more convenient name by using the `names` option: this is the object of Listing 13. When empty value is set in `names` list then a guessed name is used.

Listing 13: : `fc_bench.demos.bench_MatProd04` function

```
def bench_MatProd04():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul,lambda X,Y: X@Y,fc_bench.demos.matprod01]
  names=['matmul(X,Y)','X@Y','']
  comment=['# Benchmarking function numpy.matmul',
           '# where X and Y are m-by-m Numpy arrays']
  In=np.arange(50,201,50)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,In, comment=comment,
     names=names)
```

Output

```
#----------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are m-by-m Numpy arrays
#----------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#----------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#----------------------------------------------------------------------------
#date:2019-12-23_08-04-06
#nbruns:5
#numpy:       <i8            f8      f8           f8
#format:    {:>8}       {:14.3f} {:6.3f}      {:12.3f}
#labels:        m   matmul(X,Y)(s)  X@Y(s)  matprod01(s)
               50          0.000   0.000         0.090
              100          0.000   0.000         0.703
              150          0.000   0.000         2.342
              200          0.000   0.000         5.608
```

With Python version < 3.5, the `lambda X,Y: X@Y` function must be omited in `Lfun`.

4.1.5 Square matrices: `fc_bench.demos.bench_MatProd05` function

As previous section, we want to compare computationnal times between the `numpy.matmul(X,Y)` function, the `X@Y` command and the `fc_bench.demos.matprod01` function given in Listing 11. In addition, we also want to display errors between the outputs of the functions. The first function given in `Lfun` is the reference one and errors are always computed by using output of this reference function and output of the other functions.

Two examples are proposed that use the `fc_bench.bench` function with `error` option to display comparative errors. They both use the `fc_bench.demos.setMatProd02` function given in Listing 9. The first one given in Listing 14 uses the `comment` option and a manual writing to print some informations about labels columns. The second one given in Listing 15 uses the `labelsinfo` option to automatically print some informations about labels columns.

Listing 14: : `fc_bench.demos.bench_MatProd05` function

```
def bench_MatProd05():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul,lambda X,Y: X@Y ,fc_bench.demos.matprod02]
  compfun=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  names=['matmul(X,Y)','X@Y','']
  comment=['# Benchmarking functions:',
           '#     A1=numpy.matmul(X,Y) (reference)',
           '#     A2= X@Y',
           '#     A3= fc_bench.demos.matprod02(X,Y)',
           '# where X and Y are m-by-m Numpy arrays',
           '# comp[0] is the norm(A1-A2,Inf)',
           '# comp[2] is the norm(A1-A3,Inf)']
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,In, comment=comment,
     names=names, info=False, compfun=compfun)
```

Output

```
#---------------------------------------------------------------------------
# Benchmarking functions:
#     A1=numpy.matmul(X,Y) (reference)
#     A2= X@Y
#     A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are m-by-m Numpy arrays
# comp[0] is the norm(A1-A2,Inf)
# comp[2] is the norm(A1-A3,Inf)
#---------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#---------------------------------------------------------------------------
#date:2019-12-23_08-05-08
#nbruns:5
#numpy:      <i8             f8      f8        f8           f8        f8
#format:    {:>8}       {:14.3f} {:6.3f}   {:9.3e}      {:12.3f}   {:9.3e}
#labels:       m   matmul(X,Y)(s)  X@Y(s)    comp[0]  matprod02(s)   comp[1]
             100          0.000   0.000  0.000e+00        0.019  3.888e-13
             200          0.000   0.000  0.000e+00        0.052  1.378e-12
             300          0.001   0.000  0.000e+00        0.125  2.373e-12
             400          0.001   0.001  0.000e+00        0.239  4.182e-12
```

Listing 15: : `fc_bench.demos.bench_MatProd05bis` function

```python
def bench_MatProd05bis():
  import fc_bench
  import numpy as np
  # with labelsinfo , <lambda> function must be alone on a code line
  # (otherwise not well  guessed)
  f=lambda X,Y: X@Y
  Lfun=[np.matmul,f,fc_bench.demos.matprod02]
  compfun=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  names=['matmul(X,Y)','X@Y','']
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,In, names=names,
     info=False,labelsinfo=True, compfun=compfun)
```

Output

```
#---------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#---------------------------------------------------------------------------
# Benchmarking functions:
#   fun[0],  matmul(X,Y):
#   fun[1],          X@Y: lambda X,Y: X@Y
#   fun[2],    matprod02: fc_bench.demos.matprod02
#---------------------------------------------------------------------------
# Comparative functions:
#   comp[i-1,0], compares outputs of fun[0] and fun[i]
#      lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
#    For each comparative function:
#     - 1st input parameter is the output of fun[0]
#     - 2nd input parameter is the output of fun[i]
#---------------------------------------------------------------------------
#date:2019-12-23_08-05-11
#nbruns:5
#numpy:       <i8             f8      f8         f8           f8         f8
#format:    {:>8}       {:14.3f} {:6.3f}    {:9.3e}      {:12.3f}    {:9.3e}
#labels:        m matmul(X,Y)(s)  X@Y(s)    comp[0]  matprod02(s)    comp[1]
              100          0.000   0.000  0.000e+00         0.019  3.796e-13
              200          0.000   0.000  0.000e+00         0.053  1.486e-12
              300          0.000   0.000  0.000e+00         0.127  2.469e-12
              400          0.001   0.001  0.000e+00         0.239  4.209e-12
```

4.1.6 Non-square matrices: `fc_bench.demos.bench_MatProd06` function

As previous section, we want to compare computationnal times between the `numpy.matmul(X,Y)` function, the `X@Y` command and the `fc_bench.demos.matprod01` function given in Listing 11 but this time with non-square matrices. In addition, we also want to display errors between the outputs of the functions. The first function is the reference one and errors are always computed by using output of this reference function and output of the functions.

```
def setMatProd03(datas,verbose,**kwargs):
  assert len(datas)==3 or len(datas)==1
  Print=kwargs.get('Print',lambda s: print(s))
  ldtype=kwargs.get('ldtype', )
  rdtype=kwargs.get('rdtype', )
  lcomplex=kwargs.get('lcomplex',False)
  rcomplex=kwargs.get('rcomplex',False)

  from fc_bench.bench import bData
  if len(datas)==1:
    m=n=p=datas
  else:
    [m,n,p]=datas

  X=genMat(m,n,ldtype,lcomplex)
  Y=genMat(n,p,rdtype,rcomplex)

  if verbose:
    Print('# 1st input parameter: (m,n) Numpy array [%s]'%str(X.dtype))
    Print('# 2nd input parameter: (n,p) Numpy array [%s]'%str(Y.dtype))
  bDs=[bData('{:>7}'.format('m'), m, '{:>7}')]
  bDs.append(bData('{:>7}'.format('n'), n, '{:>7}'))
  bDs.append(bData('{:>7}'.format('p'), p, '{:>7}'))
  return ((X,Y),bDs,[])
```

Listing 16: `fc_bench.demos.setMatProd03` function

```
def genMat(m,n,dtype,isComplex):
  M=0;
  if isComplex:
    M=1j*np.random.randn(m,n)
  M+=np.random.randn(m,n)
  return M
```

Listing 17: `fc_bench.demos.genMat` function

The `fc_bench.demos.setMatProd03` function given in Listing 16 is used in `fc_bench.demos.bench_MatProd06` function.

Listing 18: : fc_bench.demos.bench_MatProd06 function

```
def bench_MatProd06():
  import fc_bench
  import numpy as np
  Lfun=[np.matmul,lambda X,Y: X@Y,fc_bench.demos.matprod02]
  compfun=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  names=['matmul(X,Y)','X@Y','']
  comment=['# Benchmarking functions:',
           '#      A1=numpy.matmul(X,Y) (reference)',
           '#      A2= X@Y',
           '#      A3= fc_bench.demos.matprod02(X,Y)',
           '# where X and Y are respectively (m,n) and (n,p) Numpy
              arrays',
           '# comp[0] is the norm(A1-A2,Inf)',
           '# comp[1] is the norm(A1-A3,Inf)']
  In=[ [100,50,100],[150,50,100],[200,50,100],[150,100,300]]
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd03,In, lcomplex=True,
     rtype=np.dtype('f4'),
                 comment=comment, names=names, info=False,
                    compfun=compfun)
```

Output

```
#--------------------------------------------------------------------------
# Benchmarking functions:
#     A1=numpy.matmul(X,Y) (reference)
#     A2= X@Y
#     A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are respectively (m,n) and (n,p) Numpy arrays
# comp[0] is the norm(A1-A2,Inf)
# comp[1] is the norm(A1-A3,Inf)
#--------------------------------------------------------------------------
# 1st input parameter: (m,n) Numpy array [complex128]
# 2nd input parameter: (n,p) Numpy array [float64]
#--------------------------------------------------------------------------
#date:2019-12-23_08-05-16
#nbruns:5
#numpy:                                        f8      f8         f8           f8          f8
#format:   {:>7}    {:>7}    {:>7}        {:14.3f} {:6.3f}    {:9.3e}      {:12.3f}     {:9.3e}
#labels:       m        n        p   matmul(X,Y)(s)  X@Y(s)    comp[0]  matprod02(s)     comp[1]
             100       50      100            0.001   0.000  0.000e+00         0.019   2.727e-13
             150       50      100            0.000   0.000  0.000e+00         0.026   2.751e-13
             200       50      100            0.000   0.000  0.000e+00         0.034   2.865e-13
             150      100      300            0.000   0.000  0.000e+00         0.082   1.460e-12
```

4.2 LU factorization examples

Let `A` be `(m,m)` Numpy array. The function `fc_bench.demos.permLU` computes the permuted LU factorization of `A` and returns the three `(m,m)` Numpy array `L` , `U` and `P` which are respectively a lower triangular matrix with unit diagonal, an upper triangular matrix and a permutation matrix so that

$$\texttt{P@A} == \texttt{L@U}$$

Its header is given in Listing 19.

```
def permLU(A):
  (m,n)=A.shape
  assert m==n
  U=A.copy()
  p=np.arange(n)
  L=np.eye(m)
  for k in np.arange(m-1):
    mu=np.argmax(abs(U[k:,k]))+k
    if abs(U[mu,k])> 1e-15:
      if mu!=k:
        U[[k,mu],k:]=U[[mu,k],k:]
        L[[k,mu],:k]=L[[mu,k],:k]
        p[[k,mu]]=p[[mu,k]]
      for j in np.arange(k+1,m):
        L[j,k]=U[j,k]/U[k,k]
        U[j,k:]+=-L[j,k]*U[k,k:]
  return L,U,permInd2Mat(p)
```

Listing 19: `fc_bench.demos.permLU` function

4.2.1 fc_bench.demos.bench_LU00

We present a very simple benchmark of the `fc_bench.demos.permLU` function. The `fc_bench.demos.setLU00` function, given in Listing 20, is used in the `fc_bench.demos.bench_LU00` function (file `demos.py`). The source code and the printed ouput are given in Listing 21.

```
def setLU00(datas,verbose,**kwargs):
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  m=datas
  A=np.random.randn(m,m)
  if verbose:
    Print('# input parameter: (m,m) Numpy array')

  bDs=[bData('{:>8}'.format('m'),m,'{:>8}')]
  return ((A),bDs,[])
```

Listing 20: `fc_bench.demos.setLU00` function

Listing 21: : fc_bench.demos.bench_LU00 function

```
def bench_LU00():
  import fc_bench
  import numpy as np
  Lfun=[permLU]
  comment=['# Benchmarking function fc_bench.demos.permLU function (LU
     factorization)']
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU00,In, comment=comment)
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking function fc_bench.demos.permLU function (LU factorization)
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#------------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#date:2019-12-23_08-05-18
#nbruns:5
#numpy:       <i8         f8
#format:    {:>8}    {:9.3f}
#labels:        m   permLU(s)
              100      0.016
              200      0.062
              300      0.142
              400      0.257
```

4.2.2 fc_bench.demos.bench_LU01

We return to the previous benchmark example to which we want to add for each m value the error committed:

np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf).

The syntax of the fc_bench.demos.permLU function is

L,U,P=fc_bench.demos.permLU(A).

So we can defined, for each input matrix A , an Error function which only depands on the outputs (with same order)

```
Error=lambda L,U,P: np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf)
```

This command is written in the initialization function (after initialization of returned input datas) and the Error function is appended at the end of the Inputs tuple. The initialization function named fc_bench.demos.setLU01 is provided in Listing 22.

```
def setLU01(datas,verbose,**kwargs):
  import fc_tools
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  m=datas
  A=np.random.randn(m,m)
  Error=lambda L,U,P: np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf);
  if verbose:
    Print('# input parameter: (m,m) Numpy array')
    Print('# Outputs are [L,U,P] such that P*A=L*U')
    Print('# Error[i] computed with fun[i] outputs :\n#
       %s'%fc_tools.others.func2str(Error,source=False))
  bDs=[bData('{:>8}'.format('m'),m,'{:>8}')]
  return ((A),bDs,Error)
```

Listing 22: `fc_bench.demos.setLU01` function

Listing 23: : `fc_bench.demos.bench_LU01` function

```
def bench_LU01():
  import fc_bench
  import numpy as np
  Lfun=[permLU]
  comment=['# Benchmarking function fc_bench.demos.permLU function (LU
     factorization)']
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU01,In, comment=comment)
```

Output

```
#---------------------------------------------------------------------------
# Benchmarking function fc_bench.demos.permLU function (LU factorization)
#---------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.3 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#              (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Python
#     release: 3.7.6
#---------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
#    np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf);
#---------------------------------------------------------------------------
#date:2019-12-23_08-05-22
#nbruns:5
#numpy:       <i8         f8          f8
#format:    {:>8}    {:9.3f}    {:10.3e}
#labels:        m   permLU(s)    Error[0]
              100      0.016   9.341e-14
              200      0.064   3.152e-13
              300      0.145   7.247e-13
              400      0.258   1.231e-12
```

4.2.3 `fc_bench.demos.bench_LU02`

We now want to add to previous example the computationnal times of the `lu` Scipy function provides by `scipy.linalg` module. This function accepts various number of inputs and outputs but the command

`P,L,U=lu(A)`

returns the three (m, m) Numpy arrays where

`A` $==$ `P@L@U`

where `P` is the transpose of that obtained by the function `fc_bench.demos.permLU`. So we must write a wrapper function to fit with the `Error` function and order of parameters: this is done in `fc_bench.demos.scipyLU` function given in Listing 24. Thereafter these two functions are used in `fc_bench.demos.bench_LU02` function given in Listing 25 with its output.

We also add a comparative function, by using the `compfun` option, which compute

$$\|\mathbb{L}_0 - \mathbb{L}_i\|_\infty + \|\mathbb{U}_0 - \mathbb{U}_i\|_\infty + \|\mathbb{P}_0 - \mathbb{P}_i\|_\infty$$

where $\mathbb{L}_0$, $\mathbb{U}_0$, $\mathbb{P}_0$ are the three matrices returned by the first function `Lfun[0]` and $\mathbb{L}_i$, $\mathbb{U}_i$, $\mathbb{P}_i$ are the three matrices returned by the function `Lfun[i]`.

```
def scipyLU(A):
  from scipy.linalg import lu
  P,L,U=lu(A)
  return L,U,P.T
```

Listing 24: `fc_bench.demos.scipyLU` function

Listing 25: : `fc_bench.demos.bench_LU02` function

```
def bench_LU02():
  import fc_bench
  import numpy as np
  Lfun=[scipyLU,permLU]
  comment='# Benchmarking functions(LU factorization)'
  cmpErr=lambda o1,o2:
     np.linalg.norm(o1[0]-o2[0],np.inf)+np.linalg.norm(o1[1]-o2[1],np.inf)+np.linalg.norm(o1[2]-o2[2],np.inf
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU01,In,
     comment=comment,compfun=cmpErr,labelsinfo=True,info=False)
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking functions(LU factorization)
#------------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
#    np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf);
#------------------------------------------------------------------------------
# Benchmarking functions:
#   fun[0],       scipyLU: fc_bench.demos.scipyLU
#   fun[1],        permLU: fc_bench.demos.permLU
#------------------------------------------------------------------------------
# Comparative functions:
#   comp[i-1,0], compares outputs of fun[0] and fun[i]
#       lambda o1,o2:
    np.linalg.norm(o1[0]-o2[0],np.inf)+np.linalg.norm(o1[1]-o2[1],np.inf)+np.linalg.norm(o1[2]-o2[2],np.inf)
#    For each comparative function:
#      - 1st input parameter is the output of fun[0]
#      - 2nd input parameter is the output of fun[i]
#------------------------------------------------------------------------------
#date:2019-12-23_08-05-27
#nbruns:5
#numpy:       <i8         f8          f8         f8          f8         f8
#format:    {:>8}    {:10.3f}    {:10.3e}    {:9.3f}    {:10.3e}    {:9.3e}
#labels:       m    scipyLU(s)    Error[0]  permLU(s)    Error[1]    comp[0]
             100        0.004   7.194e-14      0.023   9.875e-14  6.839e-13
             200        0.014   2.762e-13      0.070   3.214e-13  7.067e-12
             300        0.025   5.854e-13      0.158   7.361e-13  1.059e-11
             400        0.038   9.567e-13      0.284   1.219e-12  3.317e-11
```

4.2.4 `fc_bench.demos.bench_LU03`

We now want to change, to previous example, the error computation. We want to display the L^∞, L^1 and L^2 norms of `L*U-P*A`. So in a new initialization function, `fc_bench.demos.setLU03`, we set the third output variable `Errors` as given in lines 9 to 12 of Listing 26. Thereafter this function is used by the `fc_bench.demos.bench_LU03` script given in Listing 27.

```python
def setLU03(datas,verbose,**kwargs):
  import fc_tools
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  m=datas
  A=np.random.randn(m,m)
  # Set separately error functions!
  # otherwise can't guess and print functions.
  Einf=lambda L,U,P: np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf)
  E1=lambda L,U,P: np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),1)
  E2=lambda L,U,P: np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),2)
  Errors=[Einf,E1,E2]
  if verbose:
    Print('# input parameter: (m,m) Numpy array')
    Print('# Outputs are [L,U,P] such that P*A=L*U')
    for j in range(3):
      Print('# Error[i,%d] computed with fun[i] outputs :\n#
          %s'%(j,fc_tools.others.func2str(Errors[j],source=False)))
  bDs=[bData('{:>8}'.format('m'),m,'{:>8}')]
  return ((A),bDs,Errors)
```

Listing 26: `fc_bench.demos.setLU03` function

Listing 27: : `fc_bench.demos.bench_LU03` function

```python
def bench_LU03():
  import fc_bench
  import numpy as np
  Lfun=[scipyLU,permLU]
  comment='# Benchmarking functions(LU factorization)'
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU03,In,
     comment=comment,labelsinfo=True,info=False)
```

Output

```
#---------------------------------------------------------------------------
# Benchmarking functions(LU factorization)
#---------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
# Outputs are [L,U,P] such that P*A=L*U
# Error[i,0] computed with fun[i] outputs :
#   np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf)
# Error[i,1] computed with fun[i] outputs :
#   np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),1)
# Error[i,2] computed with fun[i] outputs :
#   np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),2)
#---------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0],       scipyLU: fc_bench.demos.scipyLU
#  fun[1],        permLU: fc_bench.demos.permLU
#---------------------------------------------------------------------------
# Comparative functions:
#   For each comparative function:
#     - 1st input parameter is the output of fun[0]
#     - 2nd input parameter is the output of fun[i]
#---------------------------------------------------------------------------
#date:2019-12-23_08-05-32
#nbruns:5
#numpy:      <i8         f8           f8           f8           f8        f8           f8           f8           f8
#format:   {:>8}   {:10.3f}     {:12.3e}     {:12.3e}     {:12.3e}   {:9.3f}     {:12.3e}     {:12.3e}     {:12.3e}
#labels:       m  scipyLU(s)   Error[0,0]   Error[0,1]   Error[0,2] permLU(s)   Error[1,0]   Error[1,1]   Error[1,2]
             100       0.005    6.634e-14    7.570e-14    1.563e-14     0.022    8.462e-14    8.961e-14    1.870e-14
             200       0.014    2.736e-13    3.402e-13    4.475e-14     0.070    3.274e-13    3.597e-13    5.105e-14
             300       0.026    5.830e-13    7.253e-13    8.293e-14     0.157    8.061e-13    7.820e-13    9.797e-14
             400       0.039    9.242e-13    1.426e-12    1.272e-13     0.276    1.261e-12    1.377e-12    1.467e-13
```

4.2.5 `fc_bench.demos.bench_LU04`

As an other example, we want to compare the three matrices `L`, `U` and `P` given by the two functions. So we use the `fc_bench.demos.setLU01` initialization function and the `compfun` option of the `fc_bench.bench` function. The complete code is given by `fc_bench.demos.bench_LU04` script in Listing 28

Listing 28: : `fc_bench.demos.bench_LU04` function

```python
def bench_LU04():
  import fc_bench
  import numpy as np
  Lfun=[scipyLU,permLU]
  comment='# Benchmarking functions(LU factorization)'
  # Define each error functions on one line!
  # otherwise can't guess and print functions.
  C1=lambda o1,o2: np.linalg.norm(o1[0]-o2[0],np.inf)
  C2=lambda o1,o2:np.linalg.norm(o1[1]-o2[1],np.inf)
  C3=lambda o1,o2: np.linalg.norm(o1[2]-o2[2],np.inf)
  compfun=[C1,C2,C3]
  In=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU01,In,
      comment=comment,compfun=compfun,labelsinfo=True,info=False)
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking functions(LU factorization)
#------------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
#    np.linalg.norm(np.matmul(L,U)-np.matmul(P,A),np.inf);
#------------------------------------------------------------------------------
# Benchmarking functions:
#   fun[0],      scipyLU: fc_bench.demos.scipyLU
#   fun[1],       permLU: fc_bench.demos.permLU
#------------------------------------------------------------------------------
# Comparative functions:
#   comp[i-1,0], compares outputs of fun[0] and fun[i]
#       lambda o1,o2: np.linalg.norm(o1[0]-o2[0],np.inf)
#   comp[i-1,1], compares outputs of fun[0] and fun[i]
#       lambda o1,o2:np.linalg.norm(o1[1]-o2[1],np.inf)
#   comp[i-1,2], compares outputs of fun[0] and fun[i]
#       lambda o1,o2: np.linalg.norm(o1[2]-o2[2],np.inf)
#    For each comparative function:
#      - 1st input parameter is the output of fun[0]
#      - 2nd input parameter is the output of fun[i]
#------------------------------------------------------------------------------
#date:2019-12-23_08-05-37
#nbruns:5
#numpy:      <i8          f8         f8        f8         f8          f8          f8          f8
#format:    {:>8}    {:10.3f}   {:10.3e}   {:9.3f}   {:10.3e}    {:11.3e}    {:11.3e}    {:11.3e}
#labels:       m  scipyLU(s)   Error[0] permLU(s)   Error[1]   comp[0,0]   comp[0,1]   comp[0,2]
             100       0.005  6.341e-14     0.023  9.680e-14   9.497e-14   9.020e-13   0.000e+00
             200       0.015  2.400e-13     0.093  3.155e-13   4.833e-13   6.467e-12   0.000e+00
             300       0.026  5.623e-13     0.157  7.167e-13   1.029e-12   1.060e-11   0.000e+00
             400       0.040  8.551e-13     0.353  1.250e-12   1.667e-12   4.124e-11   0.000e+00
```

4 References

[1] Python Software Foundation. Pypi, the python package index. `https://pypi.python.org/`, 2003–.

[2] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001–.

Informations for developpers/maintainers of the bench Python package

git informations on the packages used to build this manual

```
  name: fc-bench
   tag: 0.2.0
commit: 56ba4901391836cc91e2ce64c6f15699b966fdd5
  date: 2019-12-23
  time: 07-53-49
status: 0


  name: fc-tools
   tag: 0.0.24
commit: 2ae83c0d581962971179c005d0f88ab33286725c
  date: 2019-12-21
  time: 11-34-49
status: 0
```

git informations on the LaTeX package used to build this manual

```
  name: fctools
   tag:
commit: c73b7a2fbc6bfb1a5daf17097463a3a6f3541282
  date: 2019-03-22
  time: 12-57-26
status: True
```

git informations on the fc_config package used to build this distribution

```
  name: fc-config
   tag:
commit: 76c1d56aa81c2e6a4d532eebb37c3b3768cf29f8
  date: 2019-12-22
  time: 10-07-31
status: True
```