fc bench Python package, User's Guide *

François Cuvelier†

May 8, 2018

Abstract

The fc bench Python package allows to benchmark functions and much more

*Compiled with Python 3.6.5, packages `fc_bench`-0.0.2rc0 and `fc_tools`-0.0.16

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

This work was partially supported by the ANR project DEDALES under grant ANR-14-CE23-0005.

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 ($\mathtt{X(j)}, \mathtt{Y(j)}$) 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 scipy.interpolate.barycentric_interpolate Python function.

Their source codes are in file demos.py of the package. The last function needed the scipy package [3] to be installed.

To run benchmarks, the main tool is fc_bench.bench function described in section 3. To use it, you must first write a function to initialize the input datas of the Lagrange function: it is given in Listing 1. Then this function is used as second argument of the fc_bench.bench function while the first one contains the three handle functions to benchmark. A complete script is given in Listing 2 with its displayed output.

```python
def setLagrange00(N,verbose,**kwargs):
  from fc_bench.bench import bData
  n=N[0] # degree of the interpolating polynomial
  m=N[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)
  bDs=[]
  bDs.append(bData('{:>8}'.format('m'),m,'{:5}'))
  bDs.append(bData('{:>8}'.format('n'),n,'{:5}'))
  return ((X,Y,x),bDs)
```

Listing 1: fc_bench.demos.setLagrange00 function

Listing 2: : fc_bench.demos.bench_Lagrange00 function

```python
def bench_Lagrange00():
  import fc_bench
  import numpy as np
  Lfun=[Lagrange,lagint,scipyLagrange]
  n=[5,9,15]; m=[100,500,1000];
  N,M=np.meshgrid(n,m)
  LN=np.vstack((N.flatten(),M.flatten())).T
  fc_bench.bench(Lfun,setLagrange00,LN=LN)
```

Output

```
#------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------
#------------------------------------------------------------------------
#date:2018/05/08 06:30:38
#nbruns:5
#numpy:       i4       i4          f4          f4                 f4
#format:     {:5}     {:5}     {:13.3f}     {:11.3f}           {:18.3f}
#labels:       m        n     Lagrange(s)   lagint(s)   scipyLagrange(s)
              100       5          0.012       0.015              0.000
              100       9          0.021       0.024              0.000
              100      15          0.037       0.039              0.000
              500       5          0.058       0.074              0.000
              500       9          0.105       0.143              0.000
              500      15          0.215       0.229              0.000
             1000       5          0.135       0.172              0.000
             1000       9          0.243       0.286              0.000
             1000      15          0.373       0.396              0.000
```

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(N,verbose,**kwargs):
  import fc_tools
  a=kwargs.pop('a',0)
  b=kwargs.pop('b',2*np.pi)
  sfun=kwargs.pop('fun','lambda x: np.cos(x)')
  fun=eval(sfun)
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  n=N[0] # degree of the interpolating polynomial
  m=N[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,Error),bDs)
```

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']
  error=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  n=[5,9,15]; m=[100,500,1000];
  N,M=np.meshgrid(n,m)
  LN=np.vstack((N.flatten(),M.flatten())).T
  fc_bench.bench(Lfun,setLagrange,LN=LN,error=error, names=names,
     info=False,labelsinfo=True, a=-1,b=1,fun='lambda x: np.sin(x)')
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0],          Lag: fc_bench.demos.Lagrange
#  fun[1],       lagint: fc_bench.demos.lagint
# cmpErr[i], error between fun[0] and fun[i] outputs computed with function
#    lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
# where
#    - 1st input parameter is the output of fun[0]
#    - 2nd input parameter is the output of fun[i]
#------------------------------------------------------------------------------
# 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)
#------------------------------------------------------------------------------
#date:2018/05/08 06:30:57
#nbruns:5
#numpy:     i4    i4       f4         f4          f4         f4          f4
#format: {:>5} {:>5}  {:8.3f}  {:10.3e}    {:11.3f}  {:10.3e}    {:11.3e}
#labels:     m     n   Lag(s)   Error[0]   lagint(s)   Error[1]    cmpErr[1]
           100     5    0.012  1.163e-05       0.015  1.163e-05    4.441e-16
           100     9    0.021  2.848e-10       0.025  2.848e-10    9.992e-16
           100    15    0.037  1.121e-14       0.039  1.121e-14    2.398e-14
           500     5    0.058  1.163e-05       0.074  1.163e-05    5.551e-16
           500     9    0.104  2.902e-10       0.124  2.902e-10    1.332e-15
           500    15    0.186  1.887e-14       0.199  1.887e-14    1.932e-14
          1000     5    0.116  1.163e-05       0.150  1.163e-05    6.661e-16
          1000     9    0.210  2.902e-10       0.252  2.902e-10    1.887e-15
          1000    15    0.372  2.154e-14       0.398  2.154e-14    2.476e-14
```

2 Installation

The fc bench Python package was tested on various OS and Python versions:

- **CentOS 7**[7]
  - with Python 2.7.14 compiled from source [8]
  - with Python 3.5.5 compiled from source [8]
  - with Python 3.6.4 compiled from source [8]
- **Fedora 27**[6]
  - with Python 2.7.14 compiled from source [8]
  - with Python 3.5.5 compiled from source [8]
  - with Python 3.6.4 compiled from source [8]

- **MacOS High Sierra**[1]
  - with Miniconda Python 2.7 distribution (release 2.7.14) [9]
  - with Miniconda Python 3.6 distribution (release 3.6.4) [9]
  - with Python 2.7.14 [8]
  - with Python 3.5.4 [8]
  - with Python 3.6.5 [8]
- **openSUSE Leap 42.3 7**[4]
  - with Python 2.7.14 compiled from source [8]
  - with Python 3.5.5 compiled from source [8]
  - with Python 3.6.4 compiled from source [8]
- **Ubuntu 18.04 LTS, 17.10, 16.04 LTS** [10]
  - with Python 2.7.x (x=13,14,15) compiled from source [8]
  - with Python 3.5.5 compiled from source [8]
  - with Python 3.6.x (x=3,4,5) compiled from source [8]
- **Windows 10** [5]
  - with Miniconda Python 2.7 distribution (release 2.7.14) [9]
  - with Miniconda Python 3.6 distribution (release 3.6.4) [9]
  - with Python 2.7.15 [8]
  - with Python 3.5.4 [8]
  - with Python 3.6.5 [8]

Installation : The fcbench Python package can be installed by using **pip** (or **pip3**) command [2].

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

Description

`fc_bench.bench(Lfun, setfun)`

Runs benchmark for each function given in the list `Lfun`. The function `setfun` is used to set input datas to these functions. There is the imposed syntax:

```
def setfun(N,verbose,**kwargs):
  ...
  return ((In1,In2,...),bDs)
```

By default, for all `N` in `[5,10,15]`, 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]=setfun(N,verbose,**kwargs)
```

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

Some optional `key` are available:

- `LN`, to set values of the first input of the `setfun` function of the `n` benchmark to be run. For `i`-th benchmark, the `setfun` function is used with the `i`-th `value` given by `value[i]`. `value` can be a list of length `n` or a numpy array with shape `(n,)` or `(n,m)` By default, `value` is `[5,10,15]`.

- `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`.
- `error` , to use when compative errors between the various functions are desired when displaying. In this case a function must be given which returns error (as scalar) between the output of the first function `Lfun[0]` and one of the others.

3.1 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.

3.1.1 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(m,verbose,**kwargs):
  from fc_bench.bench import bData
  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]
  LN=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd00,LN=LN)
```

Output

```
#--------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#--------------------------------------------------------------------------
#--------------------------------------------------------------------------
#date:2018/05/08 06:31:14
#nbruns:5
#numpy:       i4         f4
#format:    {:>8}    {:11.3f}
#labels:        m    matmul(s)
              500        0.002
             1000        0.005
             1500        0.016
             2000        0.040
             2500        0.074
             3000        0.123
             3500        0.192
             4000        0.268
```

3.1.2 Square matrices: fc_bench.demos.bench_MatProd01 function

Let m = n = 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(m,verbose,**kwargs):
  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 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']
  LN=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd01,LN=LN, comment=comment,
     savefile='MadProd01.out')
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd01.out
#date:2018/05/08 06:31:24
#nbruns:5
#numpy:        i4          f4
#format:    {:>8}    {:11.3f}
#labels:        m    matmul(s)
              500        0.003
             1000        0.005
             1500        0.017
             2000        0.036
             2500        0.075
             3000        0.126
             3500        0.191
             4000        0.274
```

```
#------------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd01.out
#date:2018/05/08 06:31:24
#nbruns:5
#numpy:        i4          f4
#format:    {:>8}    {:11.3f}
#labels:        m    matmul(s)
              500        0.003
             1000        0.005
             1500        0.017
             2000        0.036
             2500        0.075
             3000        0.126
             3500        0.191
             4000        0.274
```

Listing 9: Output file benchs/MadProd01.out

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.

3.1.3 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 10. This function is then used in `fc_bench.demos.bench_MatProd02` given in Listing 11.

```
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 10: `fc_bench.demos.setMatProd02` function

Listing 11: : `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']
  LN=np.arange(500,4001,500)
  fc_bench.bench(Lfun,setMatProd02,LN=LN, comment=comment,
     savefile='MadProd02')
```

Output

```
#------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------
#benchfile: benchs/MadProd02.out
#date:2018/05/08 06:31:34
#nbruns:5
#numpy:        i4          f4
#format:    {:>8}    {:11.3f}
#labels:        m   matmul(s)
              500       0.003
             1000       0.005
             1500       0.017
             2000       0.038
             2500       0.078
             3000       0.123
             3500       0.190
             4000       0.271
```

```
#------------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are (m,m) Numpy arrays
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd02.out
#date:2018/05/08 06:31:34
#nbruns:5
#numpy:         i4          f4
#format:     {:>8}    {:11.3f}
#labels:         m   matmul(s)
               500       0.003
              1000       0.005
              1500       0.017
              2000       0.038
              2500       0.078
              3000       0.123
              3500       0.190
              4000       0.271
```

Listing 12: Output file `benchs/MadProd02.out`

3.1.4 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 13.

```
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 13: `fc_bench.demos.matprod01` function

The `fc_bench.demos.setMatProd02` function given in Listing 10 is used in `fc_bench.demos.bench_MatProd03` function (file `demos_op.py`).

Listing 14: : `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']
  LN=np.arange(50,201,50)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,LN=LN,
      comment=comment)
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are m-by-m Numpy arrays
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#date:2018/05/08 06:31:44
#nbruns:5
#numpy:        i4          f4            f4             f4
#format:    {:>8}    {:11.3f}      {:13.3f}       {:14.3f}
#labels:        m   matmul(s)   <lambda>(s)   matprod01(s)
               50       0.000         0.000          0.062
              100       0.000         0.000          0.489
              150       0.000         0.001          1.639
              200       0.000         0.000          3.890
```

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 15. When empty value is set in `names` list then a guessed name is used.

Listing 15: : 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']
  LN=np.arange(50,201,50)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,LN=LN,
     comment=comment, names=names)
```

Output

```
#------------------------------------------------------------------------------
#   computer: cosmos
#     system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------------
# Benchmarking function numpy.matmul
# where X and Y are m-by-m Numpy arrays
#------------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#------------------------------------------------------------------------------
#date:2018/05/08 06:32:27
#nbruns:5
#numpy:        i4              f4       f4              f4
#format:    {:>8}        {:16.3f}  {:8.3f}        {:14.3f}
#labels:        m    matmul(X,Y)(s)  X@Y(s)   matprod01(s)
               50           0.000    0.000           0.064
              100           0.000    0.000           0.501
              150           0.000    0.000           1.683
              200           0.000    0.000           3.981
```

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

3.1.5 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 13. 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 10. The first one given in Listing 16 uses the `comment` option and a manual writing to print some informations about labels columns. The second one given in Listing 17 uses the `labelsinfo` option to automatically print some informations about labels columns.

Listing 16: : 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]
  error=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',
           '# cmpErr[1] is the norm(A1-A2,Inf)',
           '# cmpErr[2] is the norm(A1-A3,Inf)']
  LN=np.arange(100,401,100)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,LN=LN,
     comment=comment, names=names, info=False, error=error)
```

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
# cmpErr[1] is the norm(A1-A2,Inf)
# cmpErr[2] is the norm(A1-A3,Inf)
#--------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#--------------------------------------------------------------------------
#date:2018/05/08 06:33:11
#nbruns:5
#numpy:       i4               f4       f4          f4             f4          f4
#format:    {:>8}       {:16.3f}  {:8.3f}    {:11.3e}       {:14.3f}    {:11.3e}
#labels:       m   matmul(X,Y)(s)  X@Y(s)   cmpErr[1]  matprod02(s)   cmpErr[2]
             100            0.000    0.000  0.000e+00          0.008   3.579e-13
             200            0.000    0.000  0.000e+00          0.035   1.447e-12
             300            0.001    0.001  0.000e+00          0.080   2.333e-12
             400            0.002    0.001  0.000e+00          0.168   4.202e-12
```

Listing 17: : 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]
  error=lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
  names=['matmul(X,Y)','X@Y','']
  LN=np.arange(100,401,100)
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd02,LN=LN, names=names,
     info=False,labelsinfo=True, error=error)
```

Output

```
#--------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0],   matmul(X,Y): numpy.core.multiarray.matmul
#  fun[1],           X@Y: lambda X,Y: X@Y
#  fun[2],     matprod02: fc_bench.demos.matprod02
# cmpErr[i], error between fun[0] and fun[i] outputs computed with function
#    lambda o1,o2: np.linalg.norm(o1-o2,np.inf)
# where
#    - 1st input parameter is the output of fun[0]
#    - 2nd input parameter is the output of fun[i]
#--------------------------------------------------------------------------
# 1st input parameter: (m,m) Numpy array
# 2nd input parameter: (m,m) Numpy array
#--------------------------------------------------------------------------
#date:2018/05/08 06:33:13
#nbruns:5
#numpy:       i4              f4       f4          f4             f4          f4
#format:    {:>8}        {:16.3f}  {:8.3f}    {:11.3e}       {:14.3f}    {:11.3e}
#labels:       m    matmul(X,Y)(s)   X@Y(s)   cmpErr[1]   matprod02(s)   cmpErr[2]
             100           0.000    0.000   0.000e+00          0.008   3.714e-13
             200           0.000    0.000   0.000e+00          0.034   1.376e-12
             300           0.001    0.001   0.000e+00          0.081   2.353e-12
             400           0.002    0.001   0.000e+00          0.165   4.101e-12
```

3.1.6 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 13 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(N,verbose,**kwargs):
  assert len(N)==3 or len(N)==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(N)==1:
    m=n=p=N
  else:
    [m,n,p]=N

  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 18: `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 19: `fc_bench.demos.genMat` function

The `fc_bench.demos.setMatProd03` function given in Listing 18 is used in `fc_bench.demos.bench_MatProd06` function.

Listing 20: : `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]
  error=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',
           '# cmpErr[1] is the norm(A1-A2,Inf)',
           '# cmpErr[2] is the norm(A1-A3,Inf)']
  LN=[ [100,50,100],[150,50,100],[200,50,100],[150,100,300]]
  fc_bench.bench(Lfun,fc_bench.demos.setMatProd03,LN=LN, lcomplex=True,
     rtype=np.dtype('f4'),
                 comment=comment, names=names, info=False, error=error)
```

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
# cmpErr[1] is the norm(A1-A2,Inf)
# cmpErr[2] is the norm(A1-A3,Inf)
#--------------------------------------------------------------------------
# 1st input parameter: (m,n) Numpy array [complex128]
# 2nd input parameter: (n,p) Numpy array [float64]
#--------------------------------------------------------------------------
#date:2018/05/08 06:33:16
#nbruns:5
#numpy:       i4       i4       i4               f4        f4          f4            f4          f4
#format:   {:>7}    {:>7}    {:>7}          {:16.3f}  {:8.3f}    {:11.3e}      {:14.3f}    {:11.3e}
#labels:       m        n        p    matmul(X,Y)(s)   X@Y(s)   cmpErr[1]   matprod02(s)   cmpErr[2]
             100       50      100             0.000     0.000   0.000e+00          0.012   2.637e-13
             150       50      100             0.000     0.000   0.000e+00          0.018   2.702e-13
             200       50      100             0.000     0.000   0.000e+00          0.023   2.600e-13
             150      100      300             0.001     0.000   0.000e+00          0.058   1.430e-12
```

3.2 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 21.

```
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 21: `fc_bench.demos.permLU` function

3.2.1 `fc_bench.demos.bench_LU00`

We present a very simple benchmarkof the `fc_bench.demos.permLU` function. The `fc_bench.demos.setLU00` function given in Listing 22 is used in the `fc_bench.demos.bench_LU00` function (file `demos.py`). The source code and the printed ouput are given in Listing 23.

```
def setLU00(m,verbose,**kwargs):
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  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 22: `fc_bench.demos.setLU00` function

Listing 23: : 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)']
  LN=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU00,LN=LN, comment=comment)
```

Output

```
#------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#------------------------------------------------------------------------
# Benchmarking function fc_bench.demos.permLU function (LU factorization)
#------------------------------------------------------------------------
# input parameter: (m,m) Numpy array
#------------------------------------------------------------------------
#date:2018/05/08 06:33:18
#nbruns:5
#numpy:        i4          f4
#format:    {:>8}    {:11.3f}
#labels:        m   permLU(s)
              100       0.016
              200       0.060
              300       0.138
              400       0.251
```

3.2.2 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 24.

```
def setLU01(m,verbose,**kwargs):
  import fc_tools
  Print=kwargs.get('Print',lambda s: print(s))
  from fc_bench.bench import bData
  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,Error),bDs)
```

Listing 24: fc_bench.demos.setLU01 function

Listing 25: : 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)']
  LN=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU01,LN=LN, comment=comment)
```

Output

```
#---------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#---------------------------------------------------------------------------
# Benchmarking function fc_bench.demos.permLU function (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);
#---------------------------------------------------------------------------
#date:2018/05/08 06:33:22
#nbruns:5
#numpy:        i4          f4         f4
#format:    {:>8}   {:11.3f}   {:10.3e}
#labels:        m    permLU(s)   Error[0]
              100        0.014  9.545e-14
              200        0.057  3.486e-13
              300        0.129  9.289e-13
              400        0.240  1.458e-12
```

3.2.3 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 26. Thereafter these two functions are used in `fc_bench.demos.bench_LU02` function given in Listing 27 with its output.

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

Listing 26: `fc_bench.demos.scipyLU` function

Listing 27: : `fc_bench.demos.bench_LU02` function

```
def bench_LU02():
  import fc_bench
  import numpy as np
  Lfun=[scipyLU,permLU]
  comment=['# Benchmarking functions(LU factorization):',
           '#   fun[1]:  fc_bench.demos.scipyLU',
           '#   fun[1]:  fc_bench.demos.permLU']
  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)
  LN=np.arange(100,401,100)
  fc_bench.bench(Lfun,setLU01,LN=LN, comment=comment,error=cmpErr)
```

Output

```
#---------------------------------------------------------------------------
#    computer: cosmos
#      system: Ubuntu 17.10 (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: 3.6.5
#---------------------------------------------------------------------------
# Benchmarking functions(LU factorization):
#   fun[1]: fc_bench.demos.scipyLU
#   fun[1]: fc_bench.demos.permLU
#---------------------------------------------------------------------------
# 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:2018/05/08 06:33:26
#nbruns:5
#numpy:        i4           f4         f4          f4         f4          f4
#format:     {:>8}     {:12.3f}   {:10.3e}    {:11.3f}   {:10.3e}    {:11.3e}
#labels:        m    scipyLU(s)   Error[0]   permLU(s)   Error[1]   cmpErr[1]
              100         0.000  6.666e-14       0.020  6.666e-14   9.695e-13
              200         0.001  2.656e-13       0.059  2.656e-13   5.592e-12
              300         0.003  6.864e-13       0.136  6.864e-13   1.441e-11
              400         0.004  1.283e-12       0.237  1.283e-12   2.542e-11
```

3 References

[1] Apple. **macOS** high sierra. https://www.apple.com/macos/high-sierra/.

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

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

[4] SUSE LLC. **openSUSE** linux distribution. https://www.opensuse.org/.

[5] Microsoft. **Windows 10**. https://www.microsoft.com/windows.

[6] Fedora Project. **Fedora** linux distribution. https://getfedora.org/.

[7] The CentOS Project. **CentOS** linux distribution. https://www.centos.org/.

[8] Python.org. Python. http://www.python.org/, 2013.

[9] Conda team. Miniconda, minimal anaconda distribution. https://conda.io/miniconda.html, 2017.

[10] Canonical Ltd Ubuntu Foundation. **Ubuntu** linux distribution. https://www.opensuse.org/.

Informations for developpers/maintainers of the fc bench Python package

git informations on the packages used to build this manual

```
  name: fc-bench
   tag: 0.0.2rc0
commit: 9dedc7a3673efec522559acd1e95752afe0d3560
  date: 2018-05-07
  time: 18-10-05
status: 0


  name: fc-tools
   tag: 0.0.16
commit: fbe68454862f64c1b1822eb6f5f82c7e19317ad2
  date: 2018-05-06
  time: 04-50-39
status: 0
```

git informations on the LaTeX package used to build this manual

```
  name: fctools
   tag:
commit: f75855ddf94c6a449248f570db5b0a8186bb7e5f
  date: 2018-04-17
  time: 09-53-32
status: True
```

git informations on the fc_config package used to build this distribution

```
  name: fc-config
   tag:
commit: 58247fd89beb6b790f09af3c02504030e9b24579
  date: 2018-05-08
  time: 06-24-12
status: False
```