fc bench Octave package, User's Guide *

François Cuvelier†

October 14, 2018

Abstract

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

*Compiled with Octave 4.2.1, with packages `fc-bench`[0.0.5] and `fc-tools`[0.0.23]

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

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

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

`y=fun(X,Y,x)`

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

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

These functions are

- `fc_bench.demos.Lagrange`, a simplistic writing;
- `fc_bench.demos.lagint`, an optimized writing ;
- `fc_bench.demos.polyLagrange`, using `polyfit` and `polyval` Octave functions.

Their source codes are in directory `+fc_bench\+demos` of the package.

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.

```
function [Inputs,bDs]=setLagrange00(N,verbose,varargin)
  n=N(1); % degree of the interpolating polynomial
  m=N(2); % number of interpolate values
  a=0;b=2*pi;
  X=a:(b-a)/n:b;
  Y=cos(X);
  x=a+(b-a)*rand(1,m);

  bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
  bDs{2}=fc_bench.bdata('n',n,'%d',5); % second column in bench output
  Inputs={X,Y,x}; % is the inputs of the matricial product functions
end
```

Listing 1: `fc_bench.demos.setLagrange00` function

Listing 2: : fc_bench.demos.bench_Lagrange00 script

```
Lfun={@(X,Y,x) fc_bench.demos.Lagrange(X,Y,x), ...
      @(X,Y,x) fc_bench.demos.lagint(X,Y,x), ...
      @(X,Y,x) fc_bench.demos.polyLagrange(X,Y,x) };

setfun=@(varargin) fc_bench.demos.setLagrange00(varargin{:});
n=[5,9,15]; m=[100,500,1000];
[N,M]=meshgrid(n,m);
LN=[N(:),M(:)];
fc_bench.bench(Lfun, setfun,'LN',LN, 'labelsinfo',true);
```

Output

```
#--------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#--------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0], Lagrange: @(X, Y, x) fc_bench.demos.Lagrange (X, Y, x)
#  fun[1],  lagint: @(X, Y, x) fc_bench.demos.lagint (X, Y, x)
#  fun[2], polyLagrange: @(X, Y, x) fc_bench.demos.polyLagrange (X, Y, x)
#--------------------------------------------------------------------------
#--------------------------------------------------------------------------
#date:2018/10/14 09:42:21
#nbruns:5
#numpy:  i4   i4          f4        f4             f4
#format: %d   %d        %.3f      %.3f           %.3f
#labels:  m    n   Lagrange(s)  lagint(s)  polyLagrange(s)
        100    5       0.024      0.016          0.024
        500    5       0.118      0.081          0.118
       1000    5       0.236      0.163          0.236
        100    9       0.062      0.027          0.061
        500    9       0.307      0.134          0.306
       1000    9       0.613      0.269          0.613
        100   15       0.154      0.043          0.153
        500   15       0.766      0.214          0.763
       1000   15       1.533      0.428          1.531
```

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.

```
function [Inputs,bDs]=setLagrange(N,verbose,varargin)
  p = inputParser;
  p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
  p.addParamValue('a',0,@isscalar);
  p.addParamValue('b',2*pi,@isscalar);
  p.addParamValue('fun',@cos);
  p.parse(varargin{:});
  R=p.Results;
  Fprintf=R.fprintf;a=R.a;b=R.b;
  n=N(1); % degree of the interpolating polynomial
  m=N(2); % number of interpolate values
  X=a:(b-a)/n:b; Y=R.fun(X);
  x=a+(b-a)*rand(1,m);
  if verbose
    Fprintf('#␣Setting␣inputs␣of␣Lagrange␣polynomial␣functions:␣...
       y=LAGRANGE(X,Y,x)\n')
    Fprintf('#␣where␣X␣is␣a:(b-a)/n:b,␣Y=fun(X)␣and␣x␣is␣random␣values␣on␣...
       [a,b]\n')
    Fprintf('#␣␣␣n␣is␣the␣order␣of␣the␣Lagrange␣polynomial\n')
    Fprintf('#␣␣␣fun␣function␣is:␣%s\n',func2str(R.fun))
    Fprintf('#␣␣␣[a,b]=[%g,%g]\n',a,b)

    Fprintf('#␣␣␣X:␣1-by-(n+1)␣array\n')
    Fprintf('#␣␣␣Y:␣1-by-(n+1)␣array\n')
    Fprintf('#␣␣␣x:␣1-by-m␣array\n')
  end

  bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
  bDs{2}=fc_bench.bdata('n',n,'%d',5); % second column in bench output
  Inputs={X,Y,x}; % is the inputs of the matricial product functions
end
```

Listing 3: `fc_bench.demos.setLagrange` function

Listing 4: : fc_bench.demos.bench_Lagrange script

```
Lfun={@(X,Y,x) fc_bench.demos.Lagrange(X,Y,x), ...
      @(X,Y,x) fc_bench.demos.lagint(X,Y,x), ...
      @(X,Y,x) fc_bench.demos.polyLagrange(X,Y,x) };

error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setLagrange(varargin{:});
n=[5,9,15]; m=[100,500,1000];
[N,M]=meshgrid(n,m);
LN=[N(:),M(:)];
fc_bench.bench(Lfun, setfun,'LN',LN,  'error',error, 'info',false, ...
    'labelsinfo',true, 'a',-pi,'b',pi,'fun',@sin);
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0], Lagrange: @(X, Y, x) fc_bench.demos.Lagrange (X, Y, x)
#  fun[1],  lagint: @(X, Y, x) fc_bench.demos.lagint (X, Y, x)
#  fun[2], polyLagrange: @(X, Y, x) fc_bench.demos.polyLagrange (X, Y, x)
# cmpErr[i], error between fun[0] and fun[i] outputs computed with function
#    @(o1, o2) norm (o1 - o2, 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 a:(b-a)/n:b, Y=fun(X) and x is random values on [a,b]
#   n is the order of the Lagrange polynomial
#   fun function is: sin
#   [a,b]=[-3.14159,3.14159]
#   X: 1-by-(n+1) array
#   Y: 1-by-(n+1) array
#   x: 1-by-m array
#------------------------------------------------------------------------------
#date:2018/10/14 09:43:25
#nbruns:5
#numpy:  i4   i4          f4          f4          f4               f4           f4
#format: %d   %d        %.3f        %.3f        %.3e             %.3f         %.3e
#labels:  m    n  Lagrange(s)  lagint(s)   cmpErr[1]  polyLagrange(s)    cmpErr[2]
        100    5        0.024      0.017   5.551e-16            0.025    0.000e+00
        500    5        0.118      0.082   7.772e-16            0.118    0.000e+00
       1000    5        0.239      0.165   6.661e-16            0.236    0.000e+00
        100    9        0.063      0.027   2.220e-15            0.062    0.000e+00
        500    9        0.308      0.135   2.554e-15            0.307    0.000e+00
       1000    9        0.616      0.270   3.414e-15            0.625    0.000e+00
        100   15        0.156      0.044   5.063e-14            0.153    0.000e+00
        500   15        0.768      0.215   6.950e-14            0.774    0.000e+00
       1000   15        1.544      0.450   5.973e-14            1.544    0.000e+00
```

2 Installation

This package was tested on

Windows 10 (1803): with Octave versions 4.2.x ($x = 0, 1, 2$) and 4.4.x ($x = 0, 1$)

macOS High Sierra 10.13.4: with Octave 4.2.2 (installed with homebrew)

Ubuntu 18.04 LTS: with Octave versions 4.2.x ($x = 0, 1, 2$) and 4.4.x ($x = 0, 1$)

It is not compatible with Octave 4.0.x and previous.

2.1 Automatic installation, all in one (recommended)

For this method, one just has to get/download the install file ofc_bench_install.m

or to get it on the dedicated web page. Thereafter, one runs it under Octave. This script downloads, extracts and configures the *fc-bench* and the required package *fc-tools* in the current directory.

For example, to install this package in `~/Octave/packages` directory, one has to copy the file ofc_bench_install.m in the `~/Octave/packages` directory. Then in a Octave terminal run the following commands

```
>> cd ~/Octave/packages
>> ofc_bench_install
```

The optional `'dir'` option can be used to specify installation directory:

`ofc_bench_install('dir',dirname)`

where `dirname` is the installation directory (string).
This is the output of the `ofc_bench_install` command on a Linux computer:

```
Parts of the GNU Octave <fc-bench> package.
Copyright (C) 2018 Francois Cuvelier <cuvelier@math.univ-paris13.fr>

1- Downloading and extracting the packages
2- Setting the <fc-bench> package
Write in ...
     ~/Octave/packages/fc-bench-full/fc_bench-0.0.5/configure_loc.m ...
3- Using toolboxes :
   ->              fc-tools : 0.0.23
   ->              fc-bench : 0.0.5
*** Using instructions
   To use the <fc-bench> toolbox:
   addpath('~/Octave/packages/fc-bench-full/fc_bench-0.0.5')
   fc_bench.init()

   See ~/Octave/packages/ofc_bench_set.m
```

The complete package (i.e. with all the other needed packages) is stored in the directory `~/Octave/packages/fc-bench-full` and, for each Octave session, one have to set the package by:

```
>> addpath('~/Octave/packages/fc-bench-full/fc-bench-0.0.5')
>> fc_bench.init()
Using fc_bench[0.0.5] with fc_tools[0.0.23].
```

For **uninstalling**, one just has to delete directory

`~/Octave/packages/fc-bench-full`

2.2 Manual installation

- Download one of **full archives** which contains all the needed toolboxes: *fc-tools* and *fc-bench*.
- Extract the archive in a folder.

- Set Octave path by adding path of needed packages.

For example under Linux, to install this package in `~/Octave/packages` directory, one can download fc-bench-0.0.5-full.tar.gz and extract it in the `~/Octave/packages` directory:

```
wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave/fc-bench/0.0.5/fc-ben
tar zxf fc-bench-0.0.5-full.tar.gz -C ~/Octave/packages
```

For each Octave session, one has to set the package by adding paths of all packages:

```
>> addpath('~/Octave/packages/fc_bench-0.0.5')
>> addpath('~/Octave/packages/fc_tools-0.0.23')
```

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 cell array `Lfun`. The function handle `setfun` is used to set input datas to these functions. There is the imposed syntax:

```
function [Inputs,Bdatas]=setfun(N,verbose,varargin)
  ...
end
```

By default, for all `N` in `5:5:20`, computational time in second of each function in `Lfun` is evaluated by `tic-toc` command:

`t=tic(); out=Lfun{i}( Inputs{:} ); tcpu=toc(t);`

where `Inputs` is given by

`[Inputs,Bdatas]=setfun(N,verbose,varargin{:})`

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

Some optional `key`/`value` pairs arguments are available with `key`:

- `'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` as follows
  - if `value` is an `n`-by-`1` or `1`-by-`n` array, `value(i)` is used,
  - if `value` is an `n`-by-`m` array, `value(i,:)` is used,
  - if `value` is an `n`-by-`m` cell array, `value{i,:}` is used,

  By default, `value` is `5:5:20` .
- `'names'` , set the names that will be displayed during the benchmarks to name each of the functions of `Lfun` . By default `value` is the empty cell and all the names are guessed from the handle functions of `Lfun` . Otherwise, `value` is a cell array 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 handle 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 cell of strings displayed before running the benchmarks. If `value` is a cell of strings, after printing the `valuei`, 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 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 directory `value` is where file is saved.
- `'error'` , to use when compative errors between the various functions are desired when displaying. In this case an handle function must be given which returns error (as scalar) between the output of the first function `Lfun{1}` and one of the others.

3.1 Matricial product examples

Let `X` be a `m`-by-`n` matrix and `Y` be a `n`-by-`p` matrix We want to measure efficiency of the matricial product `mtimes(X,Y)` (function version) or `X*Y` (operator function) with various values of `m`, `n` and `p`.

3.1.1 Square matrices: `fc_bench.demos.bench_MatProd00` script

Let `m` = `n` = `p`.

```
function [Inputs,bDs]=setMatProd00(m,verbose,varargin)
  X=randn(m,m); Y=randn(m,m);
  bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
  Inputs={X,Y}; % is the inputs of the matricial product functions
end
```

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 script (file +fc_bench/+demos/bench_MatProd00.m of the package directory)

Listing 6: : fc_bench.demos.bench_MatProd00 script

```
Lfun={@(X,Y) mtimes(X,Y)};
setfun=@(varargin) fc_bench.demos.setMatProd00(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',500:500:4000);
```

Output

```
#------------------------------------------------------------------------------
#   computer: cosmos-ubuntu-18-04
#     system: Ubuntu 18.04.1 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: Octave
#    release: 4.2.1
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#date:2018/10/14 09:44:29
#nbruns:5
#numpy:  i4        f4
#format: %d       %.3f
#labels:  m   mtimes(s)
         500      0.002
        1000      0.008
        1500      0.018
        2000      0.037
        2500      0.070
        3000      0.114
        3500      0.169
        4000      0.243
```

3.1.2 Square matrices: fc_bench.demos.bench_MatProd01 script

Let m = n = p.

```
function [Inputs,bDs]=setMatProd01(m,verbose,varargin)
  X=randn(m,m); Y=randn(m,m);
  if verbose
    fprintf('# 1st input parameter: m-by-m matrix\n')
    fprintf('# 2nd input parameter: m-by-m matrix\n')
  end
  bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
  Inputs={X,Y}; % is the inputs of the matricial product functions
end
```

Listing 7: fc_bench.demos.setMatProd01 function

The fc_bench.demos.setMatProd01 function given in Listing 7 is used in fc_bench.demos.bench_MatProd01 script (file +fc_bench/+demos/bench_MatProd01.m of the package directory)

Listing 8: : fc_bench.demos.bench_MatProd01 script

```
Lfun={@(X,Y) mtimes(X,Y)};
Comment={'#␣benchmarking␣function␣@(X,Y)␣mtimes(X,Y)', ...
          '#␣where␣X␣and␣Y␣are␣m-by-m␣matrices'};
setfun=@(varargin) fc_bench.demos.setMatProd01(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',500:500:4000, 'comment',Comment, ...
    'savefile','MadProd01.out');
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#------------------------------------------------------------------------------
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#------------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd01.out
#date:2018/10/14 09:44:36
#nbruns:5
#numpy:  i4         f4
#format: %d       %.3f
#labels:  m   mtimes(s)
        500      0.002
       1000      0.008
       1500      0.018
       2000      0.038
       2500      0.071
       3000      0.114
       3500      0.170
       4000      0.243
```

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#------------------------------------------------------------------------------
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd01.out
#date:2018/10/14 09:44:36
#nbruns:5
#numpy:  i4         f4
#format: %d       %.3f
#labels:  m   mtimes(s)
        500      0.002
       1000      0.008
       1500      0.018
       2000      0.038
       2500      0.071
       3000      0.114
       3500      0.170
       4000      0.243
```

Listing 9: Output file `benchs/MadProd01.out`

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

3.1.3 Square matrices: `fc_bench.demos.bench_MatProd02` script

Let $\mathtt{m} = \mathtt{n} = \mathtt{p}$.

```
function [Inputs,bDs]=setMatProd02(m,verbose,varargin)
  p = inputParser;
  p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
  p.parse(varargin{:});
  Fprintf=p.Results.fprintf;
  X=randn(m,m); Y=randn(m,m);
  if verbose
    Fprintf('# 1st input parameter: m-by-m matrix\n')
    Fprintf('# 2nd input parameter: m-by-m matrix\n')
  end
  bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
  Inputs={X,Y}; % is the inputs of the matricial product functions
end
```

Listing 10: `fc_bench.demos.setMatProd02` function

The `fc_bench.demos.setMatProd02` function given in Listing 10 is used in `fc_bench.demos.bench_MatProd02` script (file `bench_MatProd02.m` of the `+fc_bench/+demos` package directory)

Listing 11: : `fc_bench.demos.bench_MatProd02` script

```
Lfun={@(X,Y) mtimes(X,Y)};
Comment={'# benchmarking function @(X,Y) mtimes(X,Y)', ...
         '# where X and Y are m-by-m matrices'};
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',500:500:4000, 'comment',Comment, ...
   'savefile','MadProd02.out');
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#------------------------------------------------------------------------------
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#------------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#------------------------------------------------------------------------------
#benchfile: benchs/MadProd02.out
#date:2018/10/14 09:44:43
#nbruns:5
#numpy:  i4        f4
#format: %d       %.3f
#labels:  m   mtimes(s)
        500      0.002
       1000      0.008
       1500      0.018
       2000      0.037
       2500      0.072
       3000      0.115
       3500      0.171
       4000      0.243
```

```
#---------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#---------------------------------------------------------------------------
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#---------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#---------------------------------------------------------------------------
#benchfile: benchs/MadProd02.out
#date:2018/10/14 09:44:43
#nbruns:5
#numpy:  i4        f4
#format: %d       %.3f
#labels:  m   mtimes(s)
        500      0.002
       1000      0.008
       1500      0.018
       2000      0.037
       2500      0.072
       3000      0.115
       3500      0.171
       4000      0.243
```

Listing 12: Output file benchs/MadProd02.out

3.1.4 Square matrices: fc_bench.demos.bench_MatProd03 and 04 scripts

Let m = n = p. We want to compare computationnal times between the mtimes(X,Y) function, the X*Y command and the fc_bench.demos.matprod01 function given in Listing 13.

```
function C=matprod01(A,B)
  [n,m]=size(A);[p,q]=size(B);
  assert( m==p )
  C=zeros(n,q);
  for i=1:n
    for j=1:q
      S=0;
      for k=1:m
        S=S+A(i,k)*B(k,j);
      end
      C(i,j)=S;
    end
  end
end
```

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 script (file bench_MatProd03.m of the +fc_bench/+demos package directory)

Listing 14: : fc_bench.demos.bench_MatProd03 script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y)};
Comment='#␣benchmarking␣matricial␣product␣functions';
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment);
```

Output

```
#---------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.1 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: Octave
#     release: 4.2.1
#---------------------------------------------------------------------------
# benchmarking matricial product functions
#---------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#---------------------------------------------------------------------------
#date:2018/10/14 09:44:50
#nbruns:5
#numpy:  i4          f4     f4            f4
#format: %d        %.3f   %.3f          %.3f
#labels:  m   mtimes(s)   @(s)  matprod02(s)
        100       0.000  0.000         0.111
        200       0.001  0.000         0.517
        300       0.002  0.002         1.230
        400       0.004  0.001         2.154
```

As the second handle function in `Lfun` has no name, the guess name is `@`. 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'` cell then a guessed name is used.

Listing 15: : fc_bench.demos.bench_MatProd04 script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y) };
names={'mtimes(X,Y)','X*Y',''};
Comment={'#␣benchmarking␣functions␣@(X,Y)␣mtimes(X,Y)␣and␣@(X,Y)␣...
    X*Y', ...
           '#␣where␣X␣and␣Y␣are␣m-by-m␣matrices'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'names',names, 'info',false);
```

Output

```
#---------------------------------------------------------------------------
# benchmarking functions @(X,Y) mtimes(X,Y) and @(X,Y) X*Y
# where X and Y are m-by-m matrices
#---------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#---------------------------------------------------------------------------
#date:2018/10/14 09:45:19
#nbruns:5
#numpy:  i4               f4       f4            f4
#format: %d             %.3f     %.3f          %.3f
#labels:  m   mtimes(X,Y)(s)  X*Y(s)  matprod02(s)
        100            0.000    0.000         0.119
        200            0.001    0.001         0.514
        300            0.002    0.001         1.214
        400            0.003    0.001         2.209
```

3.1.5 Square matrices: `fc_bench.demos.bench_MatProd05` script

As previous section, we want to compare computationnal times between the `mtimes(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 is the reference one and errors are always computed by using output of this reference function and output of the functions.

Two examples, using the `fc_bench.bench` function with `'error'` option to display comparative errors, are proposed. 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 manual writing to print some informations on labels columns. The second one given in Listing 17 uses the `'labelsinfo'` option to automaticaly print some informations on labels columns.

Listing 16: : `fc_bench.demos.bench_MatProd05` script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y) };
names={'mtimes(X,Y)','X*Y',''};
Comment={'# Benchmarking functions:' ...
          '#     A1=mtimes(X,Y) (reference)', ...
          '#     A2= X*Y', ...
          '#     A3= fc_bench.demos.matprod02(X,Y)', ...
          '# where X and Y are m-by-m matrices', ...
          '# cmpErr[1] is the norm(A1-A2,Inf)', ...
          '# cmpErr[2] is the norm(A1-A3,Inf)'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'names',names, 'error',error, 'info',false);
```

Output

```
#------------------------------------------------------------------------------
# Benchmarking functions:
#    A1=mtimes(X,Y) (reference)
#    A2= X*Y
#    A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are m-by-m matrices
# cmpErr[1] is the norm(A1-A2,Inf)
# cmpErr[2] is the norm(A1-A3,Inf)
#------------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#------------------------------------------------------------------------------
#date:2018/10/14 09:45:48
#nbruns:5
#numpy:  i4             f4       f4           f4            f4           f4
#format: %d           %.3f     %.3f         %.3e          %.3f         %.3e
#labels:  m   mtimes(X,Y)(s)  X*Y(s)    cmpErr[1]  matprod02(s)    cmpErr[2]
        100           0.000    0.000    0.000e+00         0.119    3.360e-13
        200           0.001    0.001    0.000e+00         0.548    1.210e-12
        300           0.002    0.001    0.000e+00         1.241    2.185e-12
        400           0.003    0.001    0.000e+00         2.207    3.492e-12
```

Listing 17: : fc_bench.demos.bench_MatProd05bis script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
   fc_bench.demos.matprod02(X,Y) };
names={'mtimes(X,Y)','X*Y',''};

error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'names',names, ...
   'error',error, 'info',false, 'labelsinfo',true);
```

Output

```
#--------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0], mtimes(X,Y): @(X, Y) mtimes (X, Y)
#  fun[1],     X*Y: @(X, Y) X * Y
#  fun[2], matprod02: @(X, Y) fc_bench.demos.matprod02 (X, Y)
# cmpErr[i], error between fun[0] and fun[i] outputs computed with function
#    @(o1, o2) norm (o1 - o2, Inf)
# where
#    - 1st input parameter is the output of fun[0]
#    - 2nd input parameter is the output of fun[i]
#--------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#--------------------------------------------------------------------------
#date:2018/10/14 09:46:17
#nbruns:5
#numpy:  i4            f4      f4          f4           f4          f4
#format: %d          %.3f    %.3f        %.3e         %.3f        %.3e
#labels:  m  mtimes(X,Y)(s)  X*Y(s)    cmpErr[1]  matprod02(s)   cmpErr[2]
        100          0.000   0.000   0.000e+00        0.110   3.230e-13
        200          0.001   0.000   0.000e+00        0.525   1.205e-12
        300          0.001   0.001   0.000e+00        1.266   2.121e-12
        400          0.003   0.001   0.000e+00        2.242   3.582e-12
```

3.1.6 Non-square matrices: `fc_bench.demos.bench_MatProd06` script

As previous section, we want to compare computationnal times between the `mtimes(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.

```
function [Out,bDs]=setMatProd03(N,verbose,varargin)
  assert( ismember(length(N),[1,3]) )
  p = inputParser;
  %p.KeepUnmatched=true;
  p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
  p.addParamValue('lclass','double');
  p.addParamValue('rclass','double');
  p.addParamValue('lcomplex',false,@islogical);
  p.addParamValue('rcomplex',false,@islogical);
  p.parse(varargin{:});
  R=p.Results;
  R.lclass=lower(R.lclass);R.rclass=lower(R.rclass);
  Fprintf=R.fprintf;
  if length(N)==1
    m=N;n=N;p=N; % square matrices
  else
    m=N(1);n=N(2);p=N(3);
  end

  X=genMat(m,n,R.lclass,R.lcomplex);
  Y=genMat(n,p,R.rclass,R.rcomplex);

  if verbose
    if isreal(X), name=class(X); else, name=['complex␣',class(X)]; end
    Fprintf('#␣1st␣input␣parameter:␣m-by-n␣matrix␣[%s]\n',name)
    if isreal(Y), name=class(Y); else, name=['complex␣',class(Y)]; end
    Fprintf('#␣2nd␣input␣parameter:␣n-by-p␣matrix␣[%s]\n',name)
  end

  bDs{1}=fc_bench.bdata('m',m,'%d',7);
  bDs{2}=fc_bench.bdata('n',n,'%d',7);
  bDs{3}=fc_bench.bdata('p',p,'%d',7);
  Out={X,Y};
end

function V=genMat(m,n,classname,iscomplex)
  V=randn(m,n,classname);
  if iscomplex,  V=complex(V,randn(m,n,classname));end
end
```

Listing 18: `fc_bench.demos.setMatProd03` function

The `fc_bench.demos.setMatProd03` function given in Listing 18 is used in `fc_bench.demos.bench_MatProd06` script (file `bench_MatProd06.m` of the `+fc_bench/+demos` package directory)

Listing 19: : fc_bench.demos.bench_MatProd06 script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
   fc_bench.demos.matprod01(X,Y) };
names={'mtimes(X,Y)','X*Y',''};
Comment={'#␣benchmarking␣functions:' ...
         '#␣␣␣␣␣A1=mtimes(X,Y)␣(reference)', ...
         '#␣␣␣␣␣A2=␣X*Y', ...
         '#␣␣␣␣␣A3=␣fc_bench.demos.matprod02(X,Y)', ...
         '#␣where␣X␣and␣Y␣are␣m-by-m␣matrices', ...
         '#␣Error[1]␣is␣the␣norm(A1-A2,Inf)', ...
         '#␣Error[2]␣is␣the␣norm(A1-A3,Inf)'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd03(varargin{:});
LN=[ 100,50,100; 150,50,100; 200,50,100; 150,100,300 ];
fc_bench.bench(Lfun, setfun, 'LN',LN, 'lcomplex',true, ...
   'rclass','single', ...
   'comment',Comment, 'names',names, 'error',error, 'info',false);
```

Output

```
#----------------------------------------------------------------------------
# benchmarking functions:
#     A1=mtimes(X,Y) (reference)
#     A2= X*Y
#     A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are m-by-m matrices
# Error[1] is the norm(A1-A2,Inf)
# Error[2] is the norm(A1-A3,Inf)
#----------------------------------------------------------------------------
# 1st input parameter: m-by-n matrix [complex double]
# 2nd input parameter: n-by-p matrix [single]
#----------------------------------------------------------------------------
#date:2018/10/14 09:46:47
#nbruns:5
#numpy:    i4    i4    i4            f4       f4          f4            f4          f4
#format:   %d    %d    %d          %.3f     %.3f        %.3e          %.3f        %.3e
#labels:    m     n     p  mtimes(X,Y)(s)  X*Y(s)   cmpErr[1]  matprod01(s)   cmpErr[2]
          100    50   100         0.000    0.000   0.000e+00         2.752   9.588e-05
          150    50   100         0.000    0.000   0.000e+00         4.132   1.161e-04
          200    50   100         0.000    0.000   0.000e+00         5.503   1.101e-04
          150   100   300         0.001    0.001   0.000e+00        24.672   4.937e-04
```

3.2 LU factorization examples

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

$$\mathtt{P} * \mathtt{A} = \mathtt{L} * \mathtt{U}$$

Its header is given in Listing 20.

```
function [L,U,P]=permLU(A)
% FUNCTION fc_bench.demos.permLU
% -- [L,U,P]=permLU(A)
%      Computes permuted LU factorization of A.
%      L, U and P are respectively the lower triangular matrix with unit
%      diagonal, the upper triangular matrix and the permutation matrix
%      so that
%          P*A = L*U.
```

Listing 20: Header of the `fc_bench.demos.permLU` function

3.2.1 fc_bench.demos.bench_LU00

We present a very simple benchmark, using the fc bench package, of the fc_bench.demos.permLU function. The fc_bench.demos.setLU00 function given in Listing 21 is used in the script fc_bench.demos.bench_LU00 (file bench_LU00.m of the +fc_bench/+demos package directory). The source code and the printed ouput are given in Listing 22.

```
function [Out,bDs]=setLU00(N,verbose,varargin)
  p = inputParser;
  p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
  p.parse(varargin{:});
  R=p.Results;
  Fprintf=R.fprintf;
  m=N;
  A=randn(m,m);
  if verbose
    Fprintf('#␣input␣parameter:␣m-by-n␣matrix␣[%s]\n',class(A))
  end
  bDs{1}=fc_bench.bdata('m',m,'%d',7);
  Out={A};
end
```

Listing 21: fc_bench.demos.setLU00 function

Listing 22: : fc_bench.demos.bench_LU00 script

```
Lfun={ @(A) fc_bench.demos.permLU(A)};
Comment='#␣benchmarking␣fc_bench.demos.permLU␣function␣(LU␣...
    factorization)';
setfun=@(varargin) fc_bench.demos.setLU00(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'info',false);
```

Output

```
#---------------------------------------------------------------------------
# benchmarking fc_bench.demos.permLU function (LU factorization)
#---------------------------------------------------------------------------
# input parameter: m-by-n matrix [double]
#---------------------------------------------------------------------------
#date:2018/10/14 09:51:07
#nbruns:5
#numpy:    i4         f4
#format:   %d        %.3f
#labels:    m   permLU(s)
          100      0.126
          200      0.510
          300      1.175
          400      2.137
```

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:

norm(P*A-L*U,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 depends on the outputs (with same order)

```
Error=@(L,U,P) norm(L*U-P*A,Inf);
```

This command is written in the initialization function (after initialization of returned input datas) and the handle function `Error` is appended at the end of the `Inputs` cell array. The initialization function named `fc_bench.demos.setLU01` is provided in Listing 23.

```
function [Inputs,Bdatas]=setLU01(N,verbose,varargin)
  p = inputParser;
  p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
  p.parse(varargin{:});
  R=p.Results;
  Fprintf=R.fprintf;
  m=N;
  A=randn(m,m); % A is the input of the LU functions
  Error=@(L,U,P) norm(L*U-P*A,Inf); % A is known
  if verbose
    Fprintf('#␣Prototype␣functions␣without␣wrapper:␣...
       [L,U,P]=fun(A)\n',class(A))
    Fprintf('#␣Input␣parameter␣A:␣m-by-n␣matrix␣[%s]\n',class(A))
    Fprintf('#␣Outputs␣are␣[L,U,P]␣such␣that␣P*A=L*U\n')
    Fprintf('#␣Error[i]␣computed␣with␣fun[i]␣outputs␣:\n#␣␣␣...
       %s\n',func2str(Error))
  end
  Bdatas{1}=fc_bench.bdata('m',m,'%d',7);
  Inputs={A,Error}; % Adding Error function handle
end
```

Listing 23: `fc_bench.demos.setLU01` function

The `fc_bench.demos.permLU` function returns multiple outputs, so we need to write a wrapper function for using it as input function in `fc_bench.bench` function. This wrapper function is very simple: its converts the three outputs `[L,U,P]` of the `fc_bench.demos.permLU` in a 1-by-3 cell array `{L,U,P}` . We give in Listing 24 an example of a such function for a generic LU factorization function given by a function handle named `fun` .

```
function R=wrapperLU(fun,A)
% wrapper of LU factorization functions (needed by fc_bench.bench function)
  [L,U,P]=fun(A);
  R={L,U,P};
end
```

Listing 24: `fc_bench.demos.wrapperLU` function

Listing 25: : fc_bench.demos.bench_LU01 script

```
Lfun={ @(A) fc_bench.demos.wrapperLU(@(X) fc_bench.demos.permLU(X),A) };
names={'permLU'}; % Cannot guess name of the function , so one give it
Comment='#␣benchmarking␣LU␣factorization␣functions';
setfun=@(varargin) fc_bench.demos.setLU01(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'names',names,'info',false);
```

Output

```
#---------------------------------------------------------------------------
# benchmarking LU factorization functions
#---------------------------------------------------------------------------
# Prototype functions without wrapper: [L,U,P]=fun(A)
# Input parameter A: m-by-n matrix [double]
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
#   @(L, U, P) norm (L * U - P * A, Inf)
#---------------------------------------------------------------------------
#date:2018/10/14 09:51:35
#nbruns:5
#numpy:    i4         f4         f4
#format:   %d       %.3f       %.3e
#labels:    m   permLU(s)   Error[0]
          100      0.132   1.045e-13
          200      0.540   3.441e-13
          300      1.244   7.071e-13
          400      2.267   1.278e-12
```

3.2.3 fc_bench.demos.bench_LU02

We now want to add to previous example the computationnal times of the `lu` Octave function. This function accepts various number of inputs and outputs but the command

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

must give the same results as the `fc_bench.demos.permLU` function. So we can use the same initialization and wrapper functions

Listing 26: : fc_bench.demos.bench_LU02 script

```
Lfun={@(A) fc_bench.demos.wrapperLU(@lu,A), ...
      @(A) fc_bench.demos.wrapperLU(@(X) fc_bench.demos.permLU(X),A) };
names={'lu','permLU'};
Comment='#␣benchmarking␣LU␣factorization␣functions';
error=@(o1,o2) ...
   norm(o1{1}-o2{1},Inf)+norm(o1{2}-o2{2},Inf)+norm(o1{3}-o2{3},Inf);
setfun=@(varargin) fc_bench.demos.setLU01(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
   'names',names, 'error',error, 'info',false, 'labelsinfo',true);
```

Output

```
#------------------------------------------------------------------------------
# benchmarking LU factorization functions
#------------------------------------------------------------------------------
# Benchmarking functions:
#  fun[0],      lu: @(A) fc_bench.demos.wrapperLU (@lu, A)
#  fun[1],  permLU: @(A) fc_bench.demos.wrapperLU (@(X) fc_bench.demos.permLU (X), A)
# cmpErr[i], error between fun[0] and fun[i] outputs computed with function
#    @(o1, o2) norm (o1 {1} - o2 {1}, Inf) + norm (o1 {2} - o2 {2}, Inf) + norm (o1 {3} - o2 ...
    {3}, Inf)
# where
#    - 1st input parameter is the output of fun[0]
#    - 2nd input parameter is the output of fun[i]
#------------------------------------------------------------------------------
# Prototype functions without wrapper: [L,U,P]=fun(A)
# Input parameter A: m-by-n matrix [double]
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
#   @(L, U, P) norm (L * U - P * A, Inf)
#------------------------------------------------------------------------------
#date:2018/10/14 09:52:04
#nbruns:5
#numpy:    i4     f4          f4         f4          f4          f4
#format:   %d    %.3f        %.3e       %.3f        %.3e        %.3e
#labels:    m   lu(s)    Error[0]  permLU(s)    Error[1]   cmpErr[1]
          100   0.003   6.385e-14      0.134   8.257e-14   9.697e-13
          200   0.002   2.592e-13      0.530   3.270e-13   3.661e-12
          300   0.002   6.091e-13      1.250   7.018e-13   8.543e-12
          400   0.004   1.038e-12      2.222   1.318e-12   3.355e-11
```

Informations for git maintainers of the fc bench Octave package

git informations on the packages used to build this manual

```
-------------------------------------------------
   name : fc-bench
    tag : 0.0.5
 commit : 6793192b41aedd7635b7d0cc23f0f4d54fdfdddb
   date : 2018-06-20
   time : 12-00-14
 status : 0
-------------------------------------------------
   name : fc-tools
    tag : 0.0.23
 commit : 5728a827d9e6b883bb8ba8005a83a1a3f7d16be8
   date : 2018-05-14
   time : 14-32-51
 status : 0
-------------------------------------------------
```

git informations on the LaTeX package used to build this manual

```
-------------------------------------------------
   name : fctools
    tag :
 commit : c511471ae802d82c31520e0fe881e5cc66c087dd
   date : 2018-09-14
   time : 07:17:53
 status : 1
-------------------------------------------------
```

Using the remote configuration repository:

url	`ssh://lagagit/MCS/Cuvelier/Matlab/fc-config`
commit	`1f59292dd5c454f2dd42922906cc95c91abd9084`