




 **fc bench** Octave package, User's Guide\*  
version 0.1.1

François Cuvelier<sup>†</sup>

December 19, 2019

**Abstract**

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

---

\* $\text{\LaTeX}$  manual, revision 0.1.1.a, compiled with Octave 5.1.0, and packages `fc-bench`[0.1.1], `fc-tools`[0.0.29].

<sup>†</sup>LAGA, UMR 7539, CNRS, Université Paris 13 - Sorbonne Paris Cité, Université Paris 8, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, [couvelier@math.univ-paris13.fr](mailto:couvelier@math.univ-paris13.fr)

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

# 0 Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	Automatic installation, all in one (recommended) . . . . .	8
2.2	Manual installation . . . . .	9
<b>3</b>	<b>Description</b>	<b>9</b>
3.1	fc_bench.bench function . . . . .	9
3.2	fc_bench.bdata object . . . . .	12
<b>4</b>	<b>Examples</b>	<b>12</b>
4.1	Matricial product examples . . . . .	12
4.2	LU factorization examples . . . . .	21

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

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

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

$$y(i) = P(x(i)).$$

These functions are

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

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

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

Listing 1: 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{:});
In = [ 3,100; 5,100; 7,100; 11,100; 3,500; 5,500; 7,500; 11,500];
fc_bench.bench(Lfun, setfun, 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: Octave
# release: 5.1.0
#-----
# 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:2019/12/19 08:37:52
#nbruns:5
#numpy: i4 i4 f4 f4 f4
#format: %d %d %.3f %.3f %.3f
#labels: m n Lagrange(s) lagint(s) polyLagrange(s)
100 3 0.013 0.012 0.013
100 5 0.027 0.018 0.034
100 7 0.045 0.024 0.045
100 11 0.098 0.036 0.098
500 3 0.067 0.063 0.065
500 5 0.132 0.091 0.132
500 7 0.227 0.121 0.225
500 11 0.491 0.181 0.521

```

To run benchmarks, the main tool is the `fc_bench.bench` function described in section 3.1 and with basic syntax:

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

So one has to set the three input datas.

- `Lfun` is a cell array of handle functions: one handle function by function to benchmark. So in our example we have:

```

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) };

```

Listing 2: setting Lfun

- `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  $k \in \llbracket 1, \text{size}(In,1) \rrbracket$ . 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:

```
function [Inputs,bDs]=setLagrange00(in,verbose,varargin)
```

The `in` parameter is a `[n,m]` value (given by `In(k,:)` for each benchmark). The returned `Inputs` (cell) contains all the inputs of the Lagrange functions in the same order: `Inputs={X,Y,x}`

The returned `bDs` (`bdata` cell array) contains the first columns of the printed result. In this example, given in Listing 3, we choose to print in first column the `m` values and in second column the `n` values.

---

```
function [Inputs,bDs,Errors]=setLagrange00(in,verbose,varargin)
    n=in(1); % degree of the interpolating polynomial
    m=in(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);

    Errors={};
    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.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 4. A complete script is given in Listing 5 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 3.1.

---

```

function [Inputs,bDs,Errors]=setLagrange(in,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.ffprintf;a=R.a;b=R.b;
    n=in(1); % degree of the interpolating polynomial
    m=in(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('#_m_is_the_order_of_the_Lagrange_polynomial\n');
        Fprintf('#_fun_function_is:%s\n',func2str(R.fun));
        Fprintf('#_m[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
    Errors={@(y) norm(y-R.fun(x))};
    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 4: fc\_bench.demos.setLagrange function

Listing 5: : 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) };
setfun=@(varargin) fc_bench.demos.setLagrange(varargin{:});
In = [ 3,100; 5,100; 7,100; 11,100; 3,500; 5,500; 7,500; 11,500];
compfun=@(o1,o2) norm(o1-o2,Inf);
fc_bench.bench(Lfun, setfun, In, 'compfun',compfun, 'info',true, ...
    'labelsinfo',true, 'a',-pi,'b',pi,'fun',@sin);

```

## 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: Octave
# release: 5.1.0
#-----
# 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
#-----
# 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)
#-----
# Comparative functions:
# comp[i-1], compares outputs of fun[0] and fun[i] by using
# @(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]
#-----
#date:2019/12/19 08:38:12
#nbruns:5
#numpy: i4 i4 f4 f4 f4 f4 f4 f4 ...
#format: %d %d %.3f %.3e %.3f %.3e %.3e %.3e ...
#labels: m n Lagrange(s) Error[0] lagint(s) Error[1] comp[0] polyLagrange(s) ...
Error[2] comp[1]
100 3 0.013 1.390e+00 0.012 1.390e+00 4.441e-16 0.013 ...
1.390e+00 0.000e+00
100 5 0.027 1.183e-01 0.019 1.183e-01 6.661e-16 0.027 ...
1.183e-01 0.000e+00
100 7 0.046 5.793e-03 0.025 5.793e-03 6.661e-16 0.045 ...
5.793e-03 0.000e+00
100 11 0.101 3.676e-06 0.037 3.676e-06 1.887e-15 0.098 ...
3.676e-06 0.000e+00
500 3 0.069 3.149e+00 0.062 3.149e+00 6.661e-16 0.065 ...
3.149e+00 0.000e+00
500 5 0.136 2.687e-01 0.093 2.687e-01 6.661e-16 0.132 ...
2.687e-01 0.000e+00
500 7 0.235 1.240e-02 0.125 1.240e-02 1.221e-15 0.225 ...
1.240e-02 0.000e+00
500 11 0.506 1.334e-05 0.183 1.334e-05 6.800e-15 0.490 ...
1.334e-05 0.000e+00

```

## 2 Installation

This package was tested on various OS with Octave releases:

Operating system	4.2.0	4.2.1	4.2.2	4.4.0	4.4.1	5.1.0
CentOS 7.7.1908	✓	✓	✓	✓	✓	✓
Debian 9.11	✓	✓	✓	✓	✓	✓
Fedora 29	✓	✓	✓	✓	✓	✓
OpenSUSE Leap 15.0	✓	✓	✓	✓	✓	✓
Ubuntu 18.04.3 LTS	✓	✓	✓	✓	✓	✓
MacOS High Sierra 10.13.6			✓	✓	✓	
MacOS Mojave 10.14.4			✓	✓	✓	
Windows 10 (1909)				✓	✓	✓

It is not compatible with Octave releases prior to 4.2.0.

## 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 <fc-bench> Octave package.
Copyright (C) 2018-2019 F. Cuvelier

1- Downloading and extracting the packages
2- Setting the <fc-bench> package
Write in ...
  ~/Octave/packages/fc-bench-full/fc_bench-0.1.1/configure_loc.m ...
3- Using packages :
  ->          fc-tools : 0.0.29
  with          fc-bench : 0.1.1
*** Using instructions
  To use the <fc-bench> package:
  addpath('~/Octave/packages/fc-bench-full/fc_bench-0.1.1')
  fc_bench.init()

  See ~/Octave/packages/tmp/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-0.1.1')
>> fc_bench.init()
Try to use default parameters!
Use fc_tools.configure to configure.
Write in ...
/home/cuvelier/tmp/fc-bench-full/fc_tools-0.0.29/configure_loc.m ...
Using fc_bench[0.1.1] with fc_tools[0.0.29].
```

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-0.1.1-full.tar.gz` and extract it in the `~/Octave/packages` directory:

```
wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave/fc-bench/0.1.1/fc-ben
tar xzf fc-bench-0.1.1-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-0.1.1/fc_bench-0.1.1')
>> addpath('~/Octave/packages/fc-bench-0.1.1/fc_tools-0.0.29')
```

## 3 Description

### 3.1 `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 cell array `Lfun`. So there are  $N$  functions `Lfun{i}`, for  $i \in \llbracket 1, N \rrbracket$ . The function handle `setfun` is used to set input datas to these functions. There is the imposed syntax:

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

The `In` variable is used to run  $n$  benchmarks of the functions contained in `Lfun`. For the  $k$ -th benchmark,  $k \in \llbracket 1, n \rrbracket$ , the `setfun` is used with first parameter `in` given as follows:

- if `In` is a cell, then  $n = \text{size}(\text{In}, 1)$  and  $\text{in} = \text{In}\{k, :\}$ ,
- if `In` is 1D-array, then  $n = \text{length}(\text{In})$  and  $\text{in} = \text{In}(k)$ ,
- if `In` is 2D-array, then  $n = \text{size}(\text{In}, 1)$  and  $\text{in} = \text{In}(k, :)$ .

For the  $k$ -th benchmark,  $k \in \llbracket 1, n \rrbracket$ , the  $N$  functions contains in `Lfun` are evaluated  $n$  times by the `tic-toc` command

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

where  $i \in \llbracket 1, N \rrbracket$  and `Inputs` is given by

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

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

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

- `'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 `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 displayed. Default is `false`.

- 'savefile', if value is a not empty string, then displayed results are saved in directory benches 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.
- 'before', value is a cell array of size 0 or N. if not empty, value{i} is an empty cell or a cell array of bdata objects. value{i} is used to set  $b_i$  columns datas before printing the Lfun{i} cputime column with  $b_i = \text{length}(\text{value}\{i\})$ . These columns datas are computed from the output of the Lfun{i} benchmarked function.
- 'after', as 'before' option except that the  $a_i = \text{length}(\text{value}\{i\})$  columns datas are printed after the Lfun{i} cputime column.
- 'compfun', value is a function handle or a cell array of function handles. This option can be used to display comparison between outputs of reference benchmarked function Lfun{1} and the others Lfun{i}. Each function handle must return a scalar.

In Figure 1, we represent how columns are constructed by the fc\_bench.bench function.



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.

## 3.2 `fc_bench.bdata` object

The `fc_bench.bdata` is used to described a column data of the bench. Class constructors are given by

```
bd = bdata();
bd = bdata(name);
bd = bdata(name, value);
bd = bdata(name, value, sformat);
bd = bdata(name, value, sformat, strlen);
```

where `name` is the name which appears in the column title (default ''), `value` is the value to be printed (default 0), `sformat` is the string format used to print value (default '') and `strlen` is the minimum number of characters printed (default 0).

This class must be improved in a future release.

## 4 Examples

### 4.1 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  $p$ .

#### 4.1.1 Square matrices: `fc_bench.demos.bench_MatProd00`<sup>2</sup> script

Let  $m = n = p$ .

```
function [Inputs,bDs,Errors]=setMatProd00(m,verbose,varargin)
    X=randn(m,m); Y=randn(m,m);
    Errors={};
    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 6: `fc_bench.demos.setMatProd00` function

The `fc_bench.demos.setMatProd00` function given in Listing 6 is used in `fc_bench.demos.bench_MatProd00` script

<sup>2</sup>file `+fc_bench/+demos/bench_MatProd00.m` of the package directory

Listing 7: : fc\_bench.demos.bench\_MatProd00 script

```

if ~exist('small'), small=false;end
Lfun=@(X,Y) mtimes(X,Y);
setfun=@(varargin) fc_bench.demos.setMatProd00(varargin{:});
if small, In=20:20:100;else, In=500:500:4000;end
fc_bench.bench(Lfun, setfun, 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: Octave
# release: 5.1.0
#-----
#date:2019/12/19 08:38:32
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
500 0.005
1000 0.013
1500 0.033
2000 0.069
2500 0.130
3000 0.209
3500 0.312
4000 0.444

```

4.1.2 Square matrices: fc\_bench.demos.bench\_MatProd01<sup>4</sup> script

Let  $m = n = p$ .

```

function [Inputs,bDs,Errors]=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
Errors={};
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 8: fc\_bench.demos.setMatProd01 function

The `fc_bench.demos.setMatProd01` function given in Listing 8 is used in `fc_bench.demos.bench_MatProd01` script:

<sup>4</sup>file `+fc_bench/+demos/bench_MatProd01.m` of the package directory

Listing 9: : fc\_bench.demos.bench\_MatProd01 script

```

if ~exist('small'), small=false;end
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{:});
if small, In=20:20:100;else, In=500:500:4000;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, ...
              'savefile','MadProd01.out');

```

## 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: Octave
# release: 5.1.0
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
#date:2019/12/19 08:38:43
#nbruns:5
#numpy: i4      f4
#format: %d      %.3f
#labels: m      mtimes(s)
      500      0.004
      1000     0.013
      1500     0.035
      2000     0.066
      2500     0.129
      3000     0.204
      3500     0.317
      4000     0.440

```

```

#-----
# 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: Octave
# release: 5.1.0
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
#date:2019/12/19 08:38:43
#nbruns:5
#numpy: i4      f4
#format: %d      %.3f
#labels: m      mtimes(s)
      500      0.004
      1000     0.013
      1500     0.035
      2000     0.066
      2500     0.129
      3000     0.204
      3500     0.317
      4000     0.440

```

Listing 10: Output file benches/MadProd01.out

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

### 4.1.3 Square matrices: fc\_bench.demos.bench\_MatProd02<sup>6</sup> script

Let  $m = n = p$ .

```
function [Inputs,bDs,Errors]=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
    Errors={};
    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 11: fc\_bench.demos.setMatProd02 function

The fc\_bench.demos.setMatProd02 function given in Listing 11 is used in fc\_bench.demos.bench\_MatProd02 script:

Listing 12: : fc\_bench.demos.bench\_MatProd02 script

```
if ~exist('small'), small=false;end
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{:});
if small, In=20:20:100;else, In=500:500:4000;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, ...
    'savefile','MadProd02.out');
```

#### 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: Octave
# release: 5.1.0
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
#date:2019/12/19 08:38:54
#nbruns:5
#numpy: i4 f4
#format: %d %f
#labels: m mtimes(s)
500 0.006
1000 0.013
1500 0.033
2000 0.078
2500 0.130
3000 0.213
3500 0.317
4000 0.443
```

<sup>6</sup>file +fc\_bench/+demos/bench\_MatProd02.m of the package directory

#### 4.1.4 Square matrices: `fc_bench.demos.bench_MatProd03`<sup>9</sup> and `04`<sup>10</sup> scripts

Let  $m = n = p$ . We want to compare computational 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 11 is used in `fc_bench.demos.bench_MatProd03` script

Listing 14: : `fc_bench.demos.bench_MatProd03` script

---

```
if ~exist('small'), small=false;end
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{:});
if small, In=20:20:100;else, In=200:200:1000;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment);
```

---

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: Octave
# release: 5.1.0
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# benchmarking matricial product functions
#-----
#date:2019/12/19 08:39:05
#nbruns:5
#numpy: i4      f4      f4      f4
#format: %d      %.3f   %.3f   %.3f
#labels: m      mtimes(s) @ (s) matprod02(s)
      200      0.001 0.000      0.593
      400      0.004 0.001      2.406
      600      0.005 0.002      5.521
      800      0.009 0.005     10.036
     1000      0.014 0.008     18.234
```

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.

<sup>10</sup>file `+fc_bench/+demos/bench_MatProd04.m` of the package directory



Listing 15: : fc\_bench.demos.bench\_MatProd04 script

```

if ~exist('small'), small=false;end
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'};
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In', 'comment', Comment, 'names', names, ...
    'info', false);

```

## Output

```

#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# benchmarking functions @(X,Y) mtimes(X,Y) and @(X,Y) X*Y
# where X and Y are m-by-m matrices
#-----
#date:2019/12/19 08:43:24
#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  0.129
200          0.001  0.001  0.587
300          0.003  0.001  1.355
400          0.003  0.001  2.419

```

#### 4.1.5 Square matrices: fc\_bench.demos.bench\_MatProd05<sup>12</sup> script

As previous section, we want to compare computational 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 11. 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 automatically print some informations on labels columns.

<sup>12</sup>file `+fc_bench/+demos/bench_MatProd05.m` of the package directory

Listing 16: : fc\_bench.demos.bench\_MatProd05 script

```

if ~exist('small'), small=false;end
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', ...
    '#comp[0] is the norm(A1-A2,Inf)', ...
    '#comp[1] is the norm(A1-A3,Inf)'};
compfun=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, 'names',names, ...
    'compfun',compfun, 'info',false);

```

Output

```

#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# 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
#   comp[0] is the norm(A1-A2,Inf)
#   comp[1] is the norm(A1-A3,Inf)
#-----
#date:2019/12/19 08:43:56
#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)   comp[0]  matprod02(s)  comp[1]
100      0.000  0.000  0.000e+00    0.140  3.517e-13
200      0.001  0.000  0.000e+00    0.576  1.221e-12
300      0.002  0.001  0.000e+00    1.348  2.058e-12
400      0.004  0.001  0.000e+00    2.407  3.382e-12

```

Listing 17: : fc\_bench.demos.bench\_MatProd05bis script

```

if ~exist('small'), small=false;end
Lfun=@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y) };
names={'mtimes(X,Y)', 'X*Y', ''};

compfun=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In', 'names',names, 'compfun',compfun, ...
    'info',false, 'labelsinfo',true);

```

## Output

```

#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# 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)
#-----
# Comparative functions:
# comp[i-1], compares outputs of fun[0] and fun[i] by using
# @(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]
#-----
#date:2019/12/19 08:44:28
#nbruns:5
#numpy: i4          f4          f4          f4          f4
#format: %d          %.3f       %.3f       %.3e       %.3f       %.3e
#labels: m  mtimes(X,Y)(s) X*Y(s)  comp[0]  matprod02(s)  comp[1]
      100          0.000      0.000      0.000e+00      0.129      3.255e-13
      200          0.001      0.000      0.000e+00      0.592      1.229e-12
      300          0.002      0.001      0.000e+00      1.364      2.115e-12
      400          0.004      0.001      0.000e+00      2.428      3.435e-12

```

#### 4.1.6 Non-square matrices: fc\_bench.demos.bench\_MatProd06<sup>14</sup> 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.

<sup>14</sup>file `+fc_bench/+demos/bench_MatProd06.m` of the package directory

---

```

function [Out,bDs,Errors]=setMatProd03(in,verbose,varargin)
    assert( ismember(length(in),[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);
    Ffprintf=R.ffprintf;
    if length(in)==1
        m=in;n=in;p=in; % square matrices
    else
        m=in(1);n=in(2);p=in(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
        Ffprintf('#_1st_input_parameter:_m-by-n_matrix_[%s]\n',name)
        if isreal(Y), name=class(Y); else, name=['complex_',class(Y)]; end
        Ffprintf('#_2nd_input_parameter:_n-by-p_matrix_[%s]\n',name)
    end
    Errors={};
    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

```

if ~exist('small'), small=false;end
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', ...
    '#_comp[0]_is_the_norm(A1-A2,Inf)', ...
    '#_comp[1]_is_the_norm(A1-A3,Inf)'};
compfun=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd03(varargin{:});
if small
    In=[ 100,50,100; 150,50,100; 200,50,100; 150,100,300 ]/5;
else
    In=[ 100,50,100; 150,50,100; 200,50,100; 150,100,300 ];
end
fc_bench.bench(Lfun, setfun, In, 'lcomplex',true, 'rclass','single', ...
    'comment',Comment, 'names',names, 'compfun',compfun, 'info',false);

```

## Output

```

#-----
# 1st input parameter: m-by-n matrix [complex double]
# 2nd input parameter: n-by-p matrix [single]
#-----
# 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
# comp[0] is the norm(A1-A2,Inf)
# comp[1] is the norm(A1-A3,Inf)
#-----
#date:2019/12/19 08:45:00
#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)   comp[0]   matprod01(s)   comp[1]
          100  50  100           0.000  0.000  0.000e+00       2.984  9.840e-05
          150  50  100           0.000  0.000  0.000e+00       4.446  1.082e-04
          200  50  100           0.000  0.000  0.000e+00       5.928  1.013e-04
          150  100 300           0.002  0.001  0.000e+00       26.674  4.963e-04

```

## 4.2 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

$$P * A = L * U$$

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

#### 4.2.1 `fc_bench.demos.bench_LU00`

We present a very simple benchmark, using the `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 output are given in Listing 22.

```
function [Out,bDs,Errors]=setLU00(in,verbose,varargin)
    p = inputParser;
    p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
    p.parse(varargin{:});
    R=p.Results;
    Ffprintf=R.ffprintf;
    m=in;
    A=randn(m,m);
    if verbose
        Ffprintf('#_input_parameter: m-by-m matrix [%s]\n',class(A))
    end
    Errors={};
    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

```
if ~exist('small'), small=false;end
Lfun=@(A) fc_bench.demos.permLU(A);
Comment='#_benchmarking fc_bench.demos.permLU function (LU...
    factorization)';
setfun=@(varargin) fc_bench.demos.setLU00(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, 'info',false);
```

#### Output

```
#-----
# input parameter: m-by-m matrix [double]
#-----
# benchmarking fc_bench.demos.permLU function (LU factorization)
#-----
#date:2019/12/19 08:49:41
#nbruns:5
#numpy:   i4      f4
#format:  %d      %.3f
#labels:  m      permLU(s)
          100    0.149
          200    0.604
          300    1.498
          400    2.450
```

#### 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:

$$\text{norm}(P*A-L*U, \text{Inf}).$$

The syntax of the `fc_bench.demos.permLU` function is

$$[L,U,P]=\text{fc\_bench.demos.permLU}(A).$$

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

$$\text{@}(L,U,P) \text{ norm}(L*U-P*A, \text{Inf});$$

This command is used in the initialization function to initialize the third output parameter `Errors` as

```
Errors{1} = @(L,U,P) norm(L*U-P*A,Inf)
```

The initialization function named `fc_bench.demos.setLU01` is provided in Listing 23.

---

```
function [Inputs,Bdatas,Errors]=setLU01(in,verbose,varargin)
    p = inputParser;
    p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
    p.parse(varargin{:});
    R=p.Results;
    Fprintf=R.fprintf;
    m=in;
    A=randn(m,m); % A is the input of the LU functions
    Errors{1}=@(L,U,P) norm(L*U-P*A,Inf);
    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(Errors{1}))
    end
    Bdatas{1}=fc_bench.bdata('m',m,'%d',7);
    Inputs={A};
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: it 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

```

if ~exist('small'), small=false;end
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{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, ...
    'names',names,'info',false);

```

## Output

```

#-----
# 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)
#-----
# benchmarking LU factorization functions
#-----
#date:2019/12/19 08:50:15
#nbruns:5
#numpy:   i4         f4         f4
#format:  %d         %.3f         %.3e
#labels:  m   permLU(s)   Error[0]
          100      0.152   9.068e-14
          200      0.668   3.075e-13
          300      1.417   7.342e-13
          400      2.461   1.292e-12

```

#### 4.2.3 fc\_bench.demos.bench\_LU02

We now want to add to previous example the computational 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. We also add a comparative function which compute

$$\|L_0 - L_i\|_\infty + \|U_0 - U_i\|_\infty + \|P_0 - P_i\|_\infty$$

where  $L_0, U_0, P_0$  are the three matrices returned by the



Listing 26: : fc\_bench.demos.bench\_LU02 script

```

if ~exist('small'), small=false;end
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';
compfun=@(o1,o2) ...
      norm(o1{o1}-o2{o1},Inf)+norm(o1{o2}-o2{o2},Inf)+norm(o1{o3}-o2{o3},Inf);
setfun=@(varargin) fc_bench.demos.setLU01(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, 'names',names, ...
              'compfun',compfun, 'info',false, 'labelsinfo',true);

```

## Output

```

#-----
# 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)
#-----
# 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)
#-----
# Comparative functions:
#   comp[i-1], compares outputs of fun[0] and fun[i] by using
#   @(o1, o2) norm (o1 {o1} - o2 {o1}, Inf) + norm (o1 {o2} - o2 {o2}, Inf) + norm (o1 {o3} - ...
#   o2 {o3}, Inf)
#   where
#   - 1st input parameter is the output of fun[0]
#   - 2nd input parameter is the output of fun[i]
#-----
#date:2019/12/19 08:50:49
#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]      comp[0]
100  0.001  6.004e-14  0.159  9.080e-14  1.010e-12
200  0.507  2.672e-13  0.603  3.476e-13  3.979e-12
300  0.271  6.158e-13  1.355  7.541e-13  1.068e-11
400  0.613  9.454e-13  2.434  1.261e-12  3.967e-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  $L^\infty$ ,  $L^1$  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 10 to 12 of Listing 27. Thereafter this function is used by the `fc_bench.demos.bench_LU03` script given in Listing 28.

```

1 function [Inputs,Bdatas,Errors]=setLU03(in,verbose,varargin)
2     p = inputParser;
3     p.addParamValue('fprintf',@(varargin) fprintf(varargin{:}));
4     p.parse(varargin{:});
5     R=p.Results;
6     Fprintf=R.fprintf;
7     m=in;
8     A=randn(m,m); % A is the input of the LU functions
9     ltd=fc_tools.utils.line_text_delimiter();
10    Errors={@(L,U,P) norm(L*U-P*A,Inf), ...
11            @(L,U,P) norm(L*U-P*A,1) , ...
12            @(L,U,P) norm(L*U-P*A,2)};
13    if verbose
14        Fprintf('#Prototype functions without wrapper: ...
15                [L,U,P]=fun(A)\n',class(A))
16        Fprintf('#Input parameter A: m-by-n matrix [%s]\n',class(A))
17        Fprintf('#Outputs are [L,U,P] such that P*A=L*U\n')
18        Fprintf(ltd);
19        for j=1:3
20            Fprintf('#Error [i,%d] computed with fun[i] outputs: \n# ...
21                    %s\n',j-1,func2str(Errors{j}))
22        end
23        Bdatas{1}=fc_bench.bdata('m',m,'%d',7);
24        Inputs={A};
25    end

```

Listing 27: `fc_bench.demos.setLU03` function

```

if ~exist('small'), small=false;end
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';
setfun=@(varargin) fc_bench.demos.setLU03(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, 'names',names, ...
    'info',false, 'labelsinfo',true);

```

Listing 28: `fc_bench.demos.bench_LU03` script

There is the output of the `fc_bench.demos.bench_LU03` script:

```

#-----
# 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,0] computed with fun[i] outputs :
# @ (L, U, P) norm (L * U - P * A, Inf)
# Error[i,1] computed with fun[i] outputs :
# @ (L, U, P) norm (L * U - P * A, 1)
# Error[i,2] computed with fun[i] outputs :
# @ (L, U, P) norm (L * U - P * A, 2)
#-----
# 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)
#-----
#date:2019/12/19 08:51:32
#nbruns:5
#numpy: i4 f4 f4 f4 f4 f4 f4 f4 f4
#format: %d %3f %3e %3e %3e %3f %3e %3e %3e
#labels: m lu(s) Error [0,0] Error [0,1] Error [0,2] permLU(s) Error [1,0] Error [1,1] Error [1,2]
100 0.000 6.199e-14 1.044e-13 1.862e-14 0.159 1.074e-13 1.220e-13 2.307e-14
200 0.001 2.567e-13 3.557e-13 5.103e-14 0.593 3.329e-13 4.049e-13 5.343e-14
300 0.003 5.917e-13 8.062e-13 9.187e-14 1.320 7.711e-13 8.967e-13 1.022e-13
400 0.004 9.450e-13 1.359e-12 1.313e-13 2.357 1.233e-12 1.474e-12 1.511e-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 func-

tion and the 'compfun' option of the `fc_bench.bench` function. The complete code is given by `fc_bench.demos.bench_LU04` script in Listing 29

Listing 29: : `fc_bench.demos.bench_LU04` script

```

if ~exist('small'), small=false;end
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';
compFuns=@(o1,o2) norm(o1{1}-o2{1},Inf), ...
          @(o1,o2) norm(o1{2}-o2{2},Inf), ...
          @(o1,o2) norm(o1{3}-o2{3},Inf);
setfun=@(varargin) fc_bench.demos.setLU01(varargin{:});
if small, In=10:10:50;else, In=100:100:400;end
fc_bench.bench(Lfun, setfun, In, 'comment',Comment, 'names',names, ...
              'compfun',compFuns, 'info',false, 'labelsinfo',true);

```

Output

```

#-----
# 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)
#-----
# 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)
#-----
# Comparative functions:
# comp[i-1,0], compares outputs of fun[0] and fun[i]
#   @(o1, o2) norm (o1 {1} - o2 {1}, Inf)
# comp[i-1,1], compares outputs of fun[0] and fun[i]
#   @(o1, o2) norm (o1 {2} - o2 {2}, Inf)
# comp[i-1,2], compares outputs of fun[0] and fun[i]
#   @(o1, o2) norm (o1 {3} - o2 {3}, Inf)
# For each comparative function:
# - 1st input parameter is the output of fun[0]
# - 2nd input parameter is the output of fun[i]
#-----
#date:2019/12/19 08:52:03
#nbruns:5
#numpy:  i4      f4      f4      f4      f4      f4      f4      f4
#format:  %d    %3f    %3e    %3f    %3e    %3e    %3e    %3e
#labels:  m    lu(s)  Error[0]  permLU(s)  Error[1]  comp[0,0]  comp[0,1]  comp[0,2]
         100  0.001  6.505e-14  0.144  9.947e-14  1.003e-13  8.312e-13  0.000e+00
         200  0.002  2.420e-13  0.583  3.389e-13  3.560e-13  3.116e-12  0.000e+00
         300  0.003  6.310e-13  1.325  7.013e-13  7.542e-13  1.010e-11  0.000e+00
         400  0.005  1.034e-12  2.357  1.308e-12  1.269e-12  2.356e-11  0.000e+00

```

## Informations for git maintainers of the bench Octave package

git informations on the packages used to build this manual

```
-----  
name : fc-bench  
tag : 0.1.1  
commit : 9892a2270513cc1105d32235fcda24729cf53175  
date : 2019-12-18  
time : 15-46-45  
status : 0  
-----
```

```
-----  
name : fc-tools  
tag : 0.0.29  
commit : a3251e64e77572bb3916e665ae01baade6fc5f1a  
date : 2019-12-15  
time : 11-23-10  
status : 0  
-----
```

git informations on the L<sup>A</sup>T<sub>E</sub>X package used to build this manual

```
-----  
name : fctools  
tag :  
commit : c73b7a2fbc6bfb1a5daf17097463a3a6f3541282  
date : 2019-03-22  
time : 12:57:26  
status : 1  
-----
```

Using the remote configuration repository:

```
url      ssh://lagagit/MCS/Cuvelier/Matlab/fc-config  
commit   400c9b81b3776ce7ba2b80e01ae2be2da4292836
```