fc bench Matlab toolbox, User's Guide*
version 0.0.6

François Cuvelier†

March 6, 2019

Abstract

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

---

0 Contents

1 Introduction

The 🍊 bench Matlab toolbox 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 $\leqslant$ `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 optimized writing ;

- `fc_bench.demos.polyLagrange`, using `polyfit` and `polyval` Matlab functions.

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

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

3

Listing 2: : `fc_bench.demos.bench_Lagrange00` script

```
if ~exist('small'), small=false;end
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{:});
if small
  n=[5,9,15]; m=[100,500,1000];
else
  n=[3,5,7]; m=[10,30,70];
end
[N,M]=meshgrid(n,m);
LN=[N(:),M(:)];
fc_bench.bench(Lfun, setfun,'LN',LN, 'labelsinfo',true);
```

Output

```
#------------------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#         RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
# MaxThreads: 12
#------------------------------------------------------------------------------
# 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:2019/03/05 11:43:19
#nbruns:5
#numpy:  i4  i4        f4        f4              f4
#format: %d  %d      %.3f      %.3f            %.3f
#labels:  m   n  Lagrange(s) lagint(s) polyLagrange(s)
        10   3       0.001     0.002           0.001
        30   3       0.002     0.002           0.002
        70   3       0.004     0.005           0.002
        10   5       0.001     0.001           0.001
        30   5       0.002     0.003           0.002
        70   5       0.004     0.006           0.004
        10   7       0.001     0.001           0.001
        30   7       0.002     0.004           0.002
        70   7       0.005     0.008           0.005
```

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.

4

```
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

```matlab
if ~exist('small'), small=false;end
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{:});
if small
  n=[5,9,15]; m=[100,500,1000];
else
  n=[3,5,7]; m=[10,30,70];
end
[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:2019/03/05 11:43:33
#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]
        10   3      0.001     0.001   2.220e-16      0.001   0.000e+00
        30   3      0.001     0.001   4.441e-16      0.001   0.000e+00
        70   3      0.002     0.003   4.441e-16      0.002   0.000e+00
        10   5      0.000     0.001   3.331e-16      0.000   0.000e+00
        30   5      0.001     0.002   4.441e-16      0.001   0.000e+00
        70   5      0.003     0.005   6.661e-16      0.003   0.000e+00
        10   7      0.001     0.001   2.220e-16      0.001   0.000e+00
        30   7      0.002     0.003   5.551e-16      0.002   0.000e+00
        70   7      0.004     0.007   6.661e-16      0.004   0.000e+00
```

2 Installation

This toolbox was tested on

Windows 10 (1803): with Matlab R2015b to R2018b

macOS High Sierra 10.13.6: with Matlab R2015b to R2018b

macOS Mojave 10.14: with Matlab R2018b

Ubuntu 18.04 LTS: with Matlab R2015b to R2018b

centOS 7.6.1810: with Matlab R2015b to R2018b

Fedora 29: with Matlab R2015b to R2018b

OpenSUSE Leap 15.0: with Matlab R2015b to R2018b

Debian 9.8: with Matlab R2015b to R2018b

It is not compatible with Matlab R2015a and previous.

2.1   Automatic installation, all in one (recommended)

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

<div align="center">mfc_bench_install.m</div>

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

For example, to install this toolbox in `~/Matlab/toolboxes` directory, one has to copy the file mfc_bench_install.m in the `~/Matlab/toolboxes` directory. Then in a Matlab terminal run the following commands

```
>> cd ~/Matlab/toolboxes
>> mfc_bench_install()
```

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

<div align="center">mfc_bench_install('dir',dirname)</div>

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

```
Parts of the <fc-bench> Matlab toolbox.
Copyright (C) 2018-2019 F. Cuvelier

1- Downloading and extracting the toolboxes
2- Setting the <fc-bench> toolbox
Write in ...
    ~/Matlab/toolboxes/fc-bench-full/fc_bench-0.0.6/configure_loc.m ...
3- Using toolboxes :
   ->              fc-tools : 0.0.25
*** Using instructions
   To use the <fc-bench> toolbox:
   addpath('~/Matlab/toolboxes/fc-bench-full/fc_bench-0.0.6')
   fc_bench.init()

   See ~/Matlab/toolboxes/mfc_bench_set.m
```

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

```
>> addpath('~/Matlab/toolboxes/fc-bench-full/fc-bench-0.0.6')
>> fc_bench.init()
Using fc_bench[0.0.6] with fc_tools[0.0.25].
```

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

<div align="center">~/Matlab/toolboxes/fc-bench-full</div>

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 Matlab path by adding path of the needed toolboxes.

For example under Linux, to install this toolbox in `~/Matlab/toolboxes` directory, one can download fc-bench-0.0.6-full.tar.gz and extract it in the `~/Matlab/toolboxes` directory:

```
wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/Matlab/fc-bench/0.0.6/fc-ben
tar zxf fc-bench-0.0.6-full.tar.gz -C ~/Matlab/toolboxes
```

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

```
>> addpath('~/Matlab/toolboxes/fc-bench-0.0.6/fc_bench-0.0.6')
>> addpath('~/Matlab/toolboxes/fc-bench-0.0.6/fc_tools-0.0.25')
```

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{:})
```

8

```
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
```

<div align="center">Listing 5: <code>fc_bench.demos.setMatProd00</code> function</div>

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 toolbox directory)

<div align="center">Listing 6: : <code>fc_bench.demos.bench_MatProd00</code> script</div>

```
if ~exist('small'), small=false;end
Lfun={@(X,Y) mtimes(X,Y)};
setfun=@(varargin) fc_bench.demos.setMatProd00(varargin{:});
if small, LN=20:20:100;else, LN=500:500:4000;end
fc_bench.bench(Lfun, setfun, 'LN',LN);
```

<div align="center">Output</div>

```
#-----------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#         RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
#  MaxThreads: 12
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
#date:2019/03/05 11:43:46
#nbruns:5
#numpy:  i4          f4
#format: %d        %.3f
#labels:  m    mtimes(s)
       500      0.003
      1000      0.020
      1500      0.063
      2000      0.118
      2500      0.132
      3000      0.420
      3500      0.622
      4000      0.790
```

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
```

<div align="center">Listing 7: <code>fc_bench.demos.setMatProd01</code> function</div>

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 toolbox directory)

<div align="center">Listing 8: : <code>fc_bench.demos.bench_MatProd01</code> script</div>

```
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, LN=20:20:100;else, LN=500:500:4000;end
fc_bench.bench(Lfun, setfun, 'LN',LN, 'comment',Comment, ...
   'savefile','MadProd01.out');
```

<div align="center">Output</div>

```
#-----------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#         RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
#  MaxThreads: 12
#-----------------------------------------------------------------------
# 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:2019/03/05 11:44:16
#nbruns:5
#numpy:  i4          f4
#format: %d        %.3f
#labels:  m   mtimes(s)
        500       0.003
       1000       0.021
       1500       0.063
       2000       0.083
       2500       0.130
       3000       0.321
       3500       0.484
       4000       0.621
```

```
#-----------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#         RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
#  MaxThreads: 12
#-----------------------------------------------------------------------
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
#benchfile: benchs/MadProd01.out
#date:2019/03/05 11:44:16
#nbruns:5
#numpy:  i4          f4
#format: %d        %.3f
#labels:  m   mtimes(s)
        500       0.003
       1000       0.021
       1500       0.063
       2000       0.083
       2500       0.130
       3000       0.321
       3500       0.484
       4000       0.621
```

<div align="center">Listing 9: Output file <code>benchs/MadProd01.out</code></div>

As we can see the information print in <code>fc_bench.demos.setMatProd01</code> function are missing in output file <code>benchs/MadProd01.out</code>. 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 $m = n = p$.

```
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` toolbox directory)

Listing 11: : `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, LN=20:20:100;else, LN=500:500:4000;end
fc_bench.bench(Lfun, setfun, 'LN',LN, 'comment',Comment, ...
    'savefile','MadProd02.out');
```

Output

```
#--------------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#         RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
#  MaxThreads: 12
#--------------------------------------------------------------------------
# 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:2019/03/05 11:44:45
#nbruns:5
#numpy:  i4          f4
#format: %d        %.3f
#labels:  m   mtimes(s)
       500        0.004
      1000        0.021
      1500        0.064
      2000        0.150
      2500        0.186
      3000        0.420
      3500        0.521
      4000        0.867
```

```
#-----------------------------------------------------------------------
#    computer: rhum-ubuntu18-04
#      system: Ubuntu 18.04.2 LTS (x86_64)
#   processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#             (2 procs/6 cores by proc/2 threads by core)
#        RAM: 62.9 Go
#    software: Matlab
#     release: 2018b
# MaxThreads: 12
#-----------------------------------------------------------------------
# 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:2019/03/05 11:44:45
#nbruns:5
#numpy:  i4          f4
#format: %d        %.3f
#labels:  m   mtimes(s)
       500      0.004
      1000      0.021
      1500      0.064
      2000      0.150
      2500      0.186
      3000      0.420
      3500      0.521
      4000      0.867
```

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` toolbox directory)

```matlab
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, LN=20:20:100;else, LN=500:500:4000;end
fc_bench.bench(Lfun, setfun, 'LN',LN, 'comment',Comment);
```

Output

```
#------------------------------------------------------------------------------
#     computer: rhum-ubuntu18-04
#       system: Ubuntu 18.04.2 LTS (x86_64)
#    processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
#              (2 procs/6 cores by proc/2 threads by core)
#          RAM: 62.9 Go
#     software: Matlab
#      release: 2018b
#   MaxThreads: 12
#------------------------------------------------------------------------------
# benchmarking matricial product functions
#------------------------------------------------------------------------------
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#------------------------------------------------------------------------------
#date:2019/03/05 11:45:17
#nbruns:5
#numpy:  i4        f4     f4          f4
#format: %d       %.3f   %.3f        %.3f
#labels:  m    mtimes(s) @(s)   matprod02(s)
      500        0.003 0.003       0.767
     1000        0.013 0.011      10.561
     1500        0.067 0.065      38.561
     2000        0.129 0.088     104.012
     2500        0.133 0.129     225.277
     3000        0.264 0.222     406.594
     3500        0.345 0.397     460.587
     4000        0.597 0.495     663.746
```

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
─────────────────────────────────────────────────────────────────────
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'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
if small, LN=10:10:50;else, LN=100:100:400;end
fc_bench.bench(Lfun, setfun, 'LN',LN, '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:2019/03/05 15:28:49                                             │
│ #nbruns:5                                                             │
│ #numpy:  i4            f4       f4          f4                        │
│ #format: %d           %.3f     %.3f        %.3f                       │
│ #labels: m   mtimes(X,Y)(s) X*Y(s)  matprod02(s)                      │
│        100        0.001    0.000       0.010                          │
│        200        0.000    0.000       0.048                          │
│        300        0.001    0.001       0.121                          │
│        400        0.002    0.001       0.280                          │
└─────────────────────────────────────────────────────────────────────┘
```

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.

```
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', ...
         '#␣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{:});
if small, LN=10:10:50;else, LN=100:100:400;end
fc_bench.bench(Lfun, setfun, 'LN',LN, '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:2019/03/05 15:29:04
#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        0.001    0.000   0.000e+00        0.011     0.000e+00
        200        0.000    0.000   0.000e+00        0.051     0.000e+00
        300        0.001    0.001   0.000e+00        0.117     2.631e-12
        400        0.001    0.001   0.000e+00        0.276     4.229e-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',''};

error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
if small, LN=10:10:50;else, LN=100:100:400;end
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:2019/03/05 15:29:20
#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         0.001    0.000  0.000e+00        0.010   0.000e+00
         200         0.000    0.000  0.000e+00        0.052   0.000e+00
         300         0.001    0.001  0.000e+00        0.117   2.631e-12
         400         0.002    0.001  0.000e+00        0.278   4.229e-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` toolbox 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', ...
         '#␣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{:});
if small
  LN=[ 100,50,100; 150,50,100; 200,50,100; 150,100,300 ]/5;
else
  LN=[ 100,50,100; 150,50,100; 200,50,100; 150,100,300 ];
end
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:2019/03/05 15:29:35
#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       0.050   8.613e-05
          150     50    100       0.000   0.000   0.000e+00       0.074   1.348e-04
          200     50    100       0.000   0.000   0.000e+00       0.101   9.515e-05
          150    100    300       0.000   0.000   0.000e+00       0.355   7.543e-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

$$P * A = L * U$$

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

3.2.1 `fc_bench.demos.bench_LU00`

We present a very simple benchmark, using the `fc bench` toolbox, 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`
toolbox directory). The source code and the printed ouput are given in List-
ing 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
```

<div align="center">Listing 21: <code>fc_bench.demos.setLU00</code> function</div>

<div align="center">Listing 22: : <code>fc_bench.demos.bench_LU00</code> script</div>

```
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, LN=10:10:50;else, LN=100:100:400;end
fc_bench.bench(Lfun, setfun, 'LN',LN, 'comment',Comment, 'info',false);
```

<div align="center">Output</div>

```
#----------------------------------------------------------------------------
# benchmarking fc_bench.demos.permLU function (LU factorization)
#----------------------------------------------------------------------------
# input parameter: m-by-n matrix [double]
#----------------------------------------------------------------------------
#date:2019/03/05 15:29:52
#nbruns:5
#numpy:   i4        f4
#format:  %d        %.3f
#labels:   m   permLU(s)
          100      0.032
          200      0.124
          300      0.299
          400      0.584
```

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:

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

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

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

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

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

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

```
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, LN=10:10:50;else, LN=100:100:400;end
fc_bench.bench(Lfun, setfun, 'LN',LN, '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:2019/03/05 15:30:11
#nbruns:5
#numpy:    i4         f4         f4
#format:   %d       %.3f       %.3e
#labels:    m   permLU(s)   Error[0]
           100      0.037   8.186e-14
           200      0.125   2.653e-13
           300      0.311   6.683e-13
           400      0.598   1.118e-12
```

3.2.3 `fc_bench.demos.bench_LU02`

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

$$\texttt{[L,U,P]=lu(A)}$$

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

22

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';
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{:});
if small, LN=10:10:50;else, LN=100:100:400;end
fc_bench.bench(Lfun, setfun, 'LN',LN, '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:2019/03/05 15:30:32
#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   0.001  8.715e-14    0.033  8.186e-14  6.752e-13
          200   0.001  3.167e-13    0.131  2.653e-13  4.739e-12
          300   0.001  7.861e-13    0.325  6.683e-13  1.399e-11
          400   0.002  1.203e-12    0.632  1.118e-12  3.031e-11
```

23

Informations for git maintainers of the fc bench Matlab toolbox

```
                git informations on the toolboxes used to build this manual
  -------------------------------------------------
    name : fc-bench
     tag : 0.0.6
  commit : 6d2d0b0e832ff242027e4e3a5c0a5601a94a74d8
    date : 2019-03-05
    time : 10-12-19
  status : 0
  -------------------------------------------------
    name : fc-tools
     tag : 0.0.25
  commit : 23b7513a4a1b0619f56aa612a487bb6acd81ad2a
    date : 2019-03-03
    time : 10-07-17
  status : 0
  -------------------------------------------------
```

```
                git informations on the LaTeX package used to build this manual
  -------------------------------------------------
    name : fctools
     tag :
  commit : 03c061dee7b753948620eb7d36636287d93d0ed4
    date :
    time :
  status : 1
  -------------------------------------------------
```

Using the remote configuration repository:

```
  url      ssh://lagagit/MCS/Cuvelier/Matlab/fc-config
  commit   3d72161dd649e2601996f078bd5e602646e22551
```