



 Matlab toolbox, User's Guide*
version 0.1.4

François Cuvelier[†]

January 25, 2025

Abstract

The  Matlab toolbox allows to benchmark functions and much more

* $\text{\LaTeX}$ manual, revision 0.1.4.a, compiled with Matlab 2022a, and toolboxes `fc-bench`[0.1.4], `fc-tools`[0.0.36].

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

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

1	Introduction	3
2	Installation	6
2.1	Automatic installation, all in one (recommended)	6
2.2	Manual installation	6
3	Description	7
3.1	fc_bench.bench function	7
3.2	fc_bench.bdata object	9
4	Examples	9
4.1	Matricial product examples	9
4.2	LU factorization examples	16

1 Introduction

The **fc_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 $\leq n$ that passes through the (**n** + 1) points (**X(j)**, **Y(j)**) and is given by

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

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

$$y = \text{fun}(X, Y, x)$$

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** Matlab functions.

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

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** toolbox.

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: gloglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
# 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:2025/01/25 06:06:14
#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 0.001 0.001 0.001
100 5 0.001 0.001 0.001
100 7 0.001 0.002 0.001
100 11 0.001 0.003 0.001
500 3 0.002 0.005 0.002
500 5 0.003 0.007 0.003
500 7 0.004 0.009 0.004
500 11 0.007 0.014 0.007
```

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 $k \in \llbracket 1, \text{size}(\text{In},1) \rrbracket$. 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.fprintf;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('#_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
    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: gloglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
# 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:2025/01/25 06:06:23
#nbruns:5
#numpy: i4 i4 f4 f4 f4 f4 f4 f4 f4 f4
#format: %d %d %3f %3e %3f %3e %3e %3e %3f %3e %3e
#labels: m n Lagrange(s) Error[0] lagint(s) Error[1] comp[0] polyLagrange(s) Error[2] comp[1]
100 3 0.001 1.521e+00 0.001 1.521e+00 4.441e-16 0.001 1.521e+00 0.000e+00
100 5 0.001 1.081e-01 0.001 1.081e-01 4.441e-16 0.001 1.081e-01 0.000e+00
100 7 0.001 6.098e-03 0.002 6.098e-03 6.661e-16 0.001 6.098e-03 0.000e+00
100 11 0.001 4.560e-06 0.003 4.560e-06 4.469e-15 0.001 4.560e-06 0.000e+00
500 3 0.002 3.223e+00 0.005 3.223e+00 6.661e-16 0.002 3.223e+00 0.000e+00
500 5 0.003 2.553e-01 0.007 2.553e-01 9.992e-16 0.003 2.553e-01 0.000e+00
500 7 0.004 1.336e-02 0.009 1.336e-02 9.992e-16 0.004 1.336e-02 0.000e+00
500 11 0.007 1.339e-05 0.014 1.339e-05 8.327e-15 0.007 1.339e-05 0.000e+00

```

2 Installation

This toolbox was only tested on Matlab 2022a under Ubuntu 24.04 LTS.

2.1 Automatic installation, all in one (recommended)

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

`mfc_bench_install.m`

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:

`mfc_bench_install('dir',dirname)`

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) 2018-2025 F. Cuvelier

1- Downloading and extracting the toolboxes
2- Setting the <fc-bench> toolbox
Write in ~/Matlab/toolboxes/fc-bench-full/fc_bench-0.1.4/configure_loc.m ...
3- Using toolboxes :
->          fc-tools : 0.0.36
with       fc-bench : 0.1.4
*** Using instructions
To use the <fc-bench> toolbox:
addpath('~/Matlab/toolboxes/fc-bench-full/fc_bench-0.1.4')
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-0.1.4')
>> fc_bench.init()
Try to use default parameters!
Use fc_tools.configure to configure.
Write in ~/Matlab/toolboxes/fc-bench-full/fc_tools-0.0.36/configure_loc.m ...
Using fc_bench[0.1.4] with fc_tools[0.0.36].
```

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

`~/Matlab/toolboxes/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 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-0.1.4-full.tar.gz` and extract it in the `~/Matlab/toolboxes` directory:

```
wget http://www.math.univ-paris13.fr/~cuvelier/software/codes/Matlab/fc-bench/0.1.4/fc-bench-0.1.4-full
tar xzf fc-bench-0.1.4-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-0.1.4/fc_bench-0.1.4')
>> addpath('~/Matlab/toolboxes/fc-bench-0.1.4/fc_tools-0.0.36')
```

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 **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.
- **'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 optional. 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`² 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

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: gloglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
#date:2025/01/25 06:06:32
#nbruns:5
#numpy: i4      f4
#format: %d      %.3f
#labels: m      mtimes(s)
      500      0.002
      1000     0.010
      1500     0.025
      2000     0.062
      2500     0.113
      3000     0.190
      3500     0.299
      4000     0.429
```

²file `+fc_bench/+demos/bench_MatProd00.m` of the toolbox directory

4.1.2 Square matrices: `fc_bench.demos.bench_MatProd01`⁴ 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:

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: glogglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
# 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:2025/01/25 06:06:49
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
      500 0.002
     1000 0.010
     1500 0.025
     2000 0.062
     2500 0.114
     3000 0.191
     3500 0.299
     4000 0.427
```

```
#-----
# computer: glogglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
#date:2025/01/25 06:06:49
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
      500 0.002
     1000 0.010
     1500 0.025
     2000 0.062
     2500 0.114
     3000 0.191
     3500 0.299
     4000 0.427
```

Listing 10: Output file `benchs/MadProd01.out`

⁴file `+fc_bench/+demos/bench_MatProd01.m` of the toolbox directory

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

4.1.3 Square matrices: `fc_bench.demos.bench_MatProd02`⁶ 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: gloglop
# system: Ubuntu 24.04.1 LTS (x86_64)
# processor: AMD Ryzen 9 6900HX with Radeon Graphics
# (1 procs/8 cores by proc/2 threads by core)
# RAM: 11.4 Go
# software: Matlab
# release: 2022a
# MaxThreads: 8
#-----
# 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:2025/01/25 06:07:06
#nbruns:5
#numpy: i4          f4
#format: %d         %.3f
#labels: m          mtimes(s)
      500          0.001
      1000         0.010
      1500         0.025
      2000         0.061
      2500         0.115
      3000         0.191
      3500         0.302
      4000         0.428
```

4.1.4 Square matrices: `fc_bench.demos.bench_MatProd03`⁹ and `04`¹⁰ 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.

⁶file `+fc_bench/+demos/bench_MatProd02.m` of the toolbox directory

¹⁰file `+fc_bench/+demos/bench_MatProd04.m` of the toolbox directory

```

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: gloglop
#   system: Ubuntu 24.04.1 LTS (x86_64)
#   processor: AMD Ryzen 9 6900HX with Radeon Graphics
#             (1 procs/8 cores by proc/2 threads by core)
#   RAM: 11.4 Go
#   software: Matlab
#   release: 2022a
#   MaxThreads: 8
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
# benchmarking matricial product functions
#-----
#date:2025/01/25 06:07:22
#nbruns:5
#numpy: i4      f4      f4      f4
#format: %d      %.3f   %.3f   %.3f
#labels: m      mtimes(s) @ (s) matprod02(s)
      200      0.000 0.000      0.017
      400      0.000 0.001      0.083
      600      0.002 0.002      0.260
      800      0.005 0.005      0.611
     1000      0.010 0.009      1.177

```

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'};
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:2025/01/25 06:07:46
#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      0.003
      200      0.000 0.000      0.016
      300      0.000 0.000      0.037
      400      0.001 0.001      0.082

```

4.1.5 Square matrices: `fc_bench.demos.bench_MatProd05`¹² 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 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.

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:2025/01/25 06:07:56
#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          0.003    3.526e-13
200          0.000    0.000    0.000e+00          0.016    1.346e-12
300          0.000    0.000    0.000e+00          0.037    2.966e-12
400          0.001    0.001    0.000e+00          0.081    4.956e-12
```

¹²file `+fc_bench/+demos/bench_MatProd05.m` of the toolbox directory

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:2025/01/25 06:08:04
#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          0.003    3.526e-13
      200          0.000    0.000    0.000e+00          0.016    1.346e-12
      300          0.000    0.000    0.000e+00          0.037    2.966e-12
      400          0.001    0.000    0.000e+00          0.081    4.956e-12

```

4.1.6 Non-square matrices: `fc_bench.demos.bench_MatProd06`¹⁴ 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.

¹⁴file `+fc_bench/+demos/bench_MatProd06.m` of the toolbox 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);
    Fprintf=R.fprintf;
    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
        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
    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` 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', ...
        '#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:2025/01/25 06:08:13
#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    0.016  1.097e-04
150    50    100      0.000    0.000  0.000e+00    0.023  1.065e-04
200    50    100      0.000    0.000  0.000e+00    0.031  1.045e-04
150    100   300      0.000    0.000  0.000e+00    0.118  5.038e-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 `fcbench` 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 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;
Fprintf=R.fprintf;
m=in;
A=randn(m,m);
if verbose
    Fprintf('#_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:2025/01/25 06:08:23
#nbruns:5
#numpy:   i4      f4
#format:  %d      %.3f
#labels:  m      permLU(s)
          100     0.008
          200     0.029
          300     0.071
          400     0.141
```

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]=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)

```
@(L,U,P) norm(L*U-P*A,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: 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, 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:2025/01/25 06:08:32
#nbruns:5
#numpy:      i4          f4          f4
#format:    %d          %.3f          %.3e
#labels:    m    permLU(s)    Error[0]
100         0.009    9.253e-14
200         0.031    3.105e-13
300         0.072    6.239e-13
400         0.142    1.144e-12
```

4.2.3 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

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

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, by using the `compfun` option, 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 first function `Lfun{1}` and L_i, U_i, P_i are the three matrices returned by the function `Lfun{i}`.

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{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, 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{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]
#-----
#date:2025/01/25 06:08:42
#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      0.000    8.008e-14  0.008    9.253e-14  8.835e-13
200      0.000    3.235e-13  0.030    3.105e-13  3.168e-12
300      0.000    5.234e-13  0.072    6.239e-13  1.526e-11
400      0.001    9.541e-13  0.143    1.144e-12  2.882e-11
```

4.2.4 fc_bench.demos.bench_LU03

We now want to change, to previous example, the error computation. We want to display the L^∞ , L^1 and L^2 norms of $L*U-P*A$. So in a new initialization function, `fc_bench.demos.setLU03`, we set the third output variable `Errors` as given in lines 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:[L,U,P]=fun(A)\n',class(A))
15        Fprintf('#_Input_parameter_A: m-by-n matrix [%s]\n',class(A))
16        Fprintf('#_Outputs_are [L,U,P] such that P*A=L*U\n')
17        Fprintf(ltd);
18        for j=1:3
19            Fprintf('#_Error [i,%d] computed with fun [i] outputs: \n#_ %s\n',j-1,func2str(Errors{j}))
20        end
21    end
22    Bdatas{1}=fc_bench.bdata('m',m,'%d',7);
23    Inputs={A};
24 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:2025/01/25 06:08:51										
#nbruns:5										
#numpy:	i4	f4	f4	f4	f4	f4	f4	f4	f4	f4
#format:	%d	%.3f	%.3e	%.3e	%.3e	%.3f	%.3e	%.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	8.008e-14	9.290e-14	1.813e-14	0.008	9.253e-14	1.137e-13	1.935e-14	
	200	0.000	3.235e-13	3.028e-13	4.293e-14	0.030	3.105e-13	3.260e-13	4.670e-14	
	300	0.000	5.234e-13	6.745e-13	6.899e-14	0.072	6.239e-13	7.491e-13	8.577e-14	
	400	0.001	9.541e-13	1.054e-12	1.007e-13	0.142	1.144e-12	1.183e-12	1.304e-13	

4.2.5 `fc_bench.demos.bench_LU04`

As an other example, we want to compare the three matrices **L**, **U** and **P** given by the two functions. So we use the `fc_bench.demos.setLU01` initialization function and the `'compfun'` option of the `fc_bench.bench` function. The complete code is given by `fc_bench.demos.bench_LU04` script in Listing 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:2025/01/25 06:09:01
#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 0.000 8.008e-14 0.008 9.253e-14 8.355e-14 7.999e-13 0.000e+00
200 0.000 3.235e-13 0.030 3.105e-13 4.027e-13 2.765e-12 0.000e+00
300 0.000 5.234e-13 0.071 6.239e-13 7.816e-13 1.448e-11 0.000e+00
400 0.002 9.541e-13 0.141 1.144e-12 1.384e-12 2.743e-11 0.000e+00

```

Informations for git maintainers of the Matlab toolbox

git informations on the toolboxes used to build this manual

```
-----  
name : fc-bench  
tag : 0.1.4  
commit : 25b29c9bb56222847b57a13780267aa17639fca4  
date : 2025-01-24  
time : 17-11-30  
status : 0  
-----
```

```
-----  
name : fc-tools  
tag : 0.0.36  
commit : e235d2dd55b3157c5194b07cfc62ec6a261c6b5b  
date : 2025-01-24  
time : 17-18-50  
status : 0  
-----
```

git informations on the L^AT_EX package used to build this manual

```
-----  
name : fctools  
tag :  
commit : 03d38737a795cdbf4e1a8754470e963cdfe83316  
date : 2025-01-24  
time : 09:58:52  
status : 1  
-----
```

Using the remote configuration repository:

```
url      ssh://lagagit/MCS/Cuvelier/Matlab/fc-config  
commit   8d132912de386979f7e6adeb8b78ccb190b35006
```