




Octave package, User's Guide *

François Cuvelier[†]

April 26, 2018

Abstract

The  Octave package allows to benchmark functions and much more

*Compiled with Octave 4.2.1, with packages `fc-bench`[0.0.3] and `fc-tools`[0.0.22]

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

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

0 Contents

1	Introduction	3
2	Installation	6
2.1	Automatic installation, all in one (recommended)	7
3	fc_bench.bench function	7
3.1	Matricial product examples	9
3.2	LU factorization examples	18

1 Introduction

The `fc_bench` Octave package aims to perform simultaneous benchmarks of several functions performing the same tasks but implemented in different ways.

We are going on an example to illustrate its possibilities. This one will focus on different ways of coding the Lagrange interpolation polynomial. We first recall some generalities about this polynomial.

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

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

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

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

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 optimized writing ;
- `fc_bench.demos.polyLagrange`, using `polyfit` and `polyval` Octave functions.

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

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

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

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

Listing 1: `fc_bench.demos.setLagrange00` function

Listing 2 : fc_bench.demos.bench_Lagrange00 script

```

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

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

```

Output

```

#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
# Benchmarking functions:
# fun[0], Lagrange: @(X, Y, x) fc_bench.demos.Lagrange (X, Y, x)
# fun[1], lagint: @(X, Y, x) fc_bench.demos.lagint (X, Y, x)
# fun[2], polyLagrange: @(X, Y, x) fc_bench.demos.polyLagrange (X, Y, x)
#-----
#date:2018/04/26 11:08:36
#nbruns:5
#numpy: i4 i4 f4 f4 f4
#format: %d %d %.3f %.3f %.3f
#labels: m n Lagrange(s) lagint(s) polyLagrange(s)
100 5 0.025 0.017 0.025
500 5 0.126 0.089 0.125
1000 5 0.253 0.177 0.251
100 9 0.066 0.029 0.066
500 9 0.329 0.146 0.331
1000 9 0.656 0.296 0.659
100 15 0.163 0.047 0.163
500 15 0.815 0.235 0.821
1000 15 1.654 0.470 1.692

```

We now propose a slightly more elaborate version of the initialization function that allows to display some informations and to choose certain parameters when generating inputs datas. This new version named `fc_bench.demos.setLagrange` is given in Listing 3. A complete script is given in Listing 4 with its displayed output. In this script some options of the `fc_bench.bench` function are used `'error'`, `'info'`, `'labelsinfo'`, jointly with those of the `fc_bench.demos.setLagrange`: `'a'`, `'b'` and `'fun'`. One must be careful not to take as an option name for the initialization function one of those used in `fc_bench.bench` function. More details are given in section 3.

```

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

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

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

```

Listing 3: fc_bench.demos.setLagrange function

Listing 4: : fc_bench.demos.bench_Lagrange script

```

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

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

```

Output

```

#-----
# Benchmarking functions:
# fun[0], Lagrange: @(X, Y, x) fc_bench.demos.Lagrange (X, Y, x)
# fun[1], lagint: @(X, Y, x) fc_bench.demos.lagint (X, Y, x)
# fun[2], polyLagrange: @(X, Y, x) fc_bench.demos.polyLagrange (X, Y, x)
# cmpErr[1], error between fun[0] and fun[i] outputs computed with function
#   @(o1, o2) norm (o1 - o2, Inf)
# where
#   - 1st input parameter is the output of fun[0]
#   - 2nd input parameter is the output of fun[i]
#-----
# Setting inputs of Lagrange polynomial functions: y=LAGRANGE(X,Y,x)
# where X is a:(b-a)/n:b, Y=fun(X) and x is random values on [a,b]
# n is the order of the Lagrange polynomial
# fun function is: sin
# [a,b]=[-3.14159,3.14159]
# X: 1-by-(n+1) array
# Y: 1-by-(n+1) array
# x: 1-by-m array
#-----
#date:2018/04/26 11:09:45
#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]

```

m	n	Lagrange(s)	lagint(s)	cmpErr[1]	polyLagrange(s)	cmpErr[2]
100	5	0.026	0.018	7.772e-16	0.026	0.000e+00
500	5	0.129	0.091	7.772e-16	0.130	0.000e+00
1000	5	0.258	0.179	7.772e-16	0.261	0.000e+00
100	9	0.068	0.030	1.887e-15	0.065	0.000e+00
500	9	0.334	0.149	2.803e-15	0.336	0.000e+00
1000	9	0.664	0.288	2.665e-15	0.661	0.000e+00
100	15	0.163	0.046	4.552e-14	0.165	0.000e+00
500	15	0.816	0.231	5.995e-14	0.820	0.000e+00
1000	15	1.635	0.465	1.077e-13	1.641	0.000e+00

2 Installation

This package was tested on

Windows 10.0.16299: with Octave 4.2.0, 4.2.1 and 4.2.2

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

Ubuntu 16.04.3 LTS: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

Ubuntu 17.10: with Octave 4.2.0 and 4.2.1 and 4.2.2 (all compiled from source)

centOS 7.4: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

Fedora 27: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

OpenSUSE Leap 42.3: with Octave 4.2.0, 4.2.1 and 4.2.2 (all compiled from source)

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

2.1 Automatic installation, all in one (recommended)

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

```
ofc_bench_install.m
```

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

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

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

This is the output of the `ofc_bench_install` command on a Linux computer:

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

1- Downloading and extracting the packages
2- Setting the <fc-bench> package
Write in ...
  ~/Octave/packages/fc-bench-full/fc_bench-0.0.3/configure_loc.m ...
3- Using toolboxes :
  ->          fc-tools : 0.0.22
  ->          fc-bench : 0.0.3
*** Using instructions
To use the <fc-bench> toolbox:
  addpath('~/Octave/packages/fc-bench-full/fc_bench-0.0.3')
  fc_bench.init()

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

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

```
>> addpath('~/Octave/packages/fc-bench-full/fc_bench-0.0.3')
>> fc_bench.init()
Using fc_bench[0.0.3] with fc_tools[0.0.22].
```

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

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

3 `fc_bench.bench` function

The `fc_bench.bench` function run benchmark

Syntaxe

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

Description

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

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

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

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

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

where `Inputs` is given by

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

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

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

- `'LN'`, to set values of the first input of the `setfun` function of the `n` benchmark to be run. For `i`-th benchmark, the `setfun` function is used with the `i`-th `value` as follows
 - if `value` is an `n`-by-1 or 1-by-`n` array, `value(i)` is used,
 - if `value` is an `n`-by-`m` array, `value(i,:)` is used,
 - if `value` is an `n`-by-`m` cell array, `value{i,:}` is used,By default, `value` is `5:5:20`.
- `'names'`, set the names that will be displayed during the benchmarks to name each of the functions of `Lfun`. By default `value` is the empty cell and all the names are guessed from the handle functions of `Lfun`. Otherwise, `value` is a cell array with same length as `Lfun` such that `value{i}` is the string name associated with `Lfun{i}` function. If `value{i}` is the empty string, then the name is guessed from the handle function `Lfun{i}`.
- `'nbruns'`, to set number of benchmark runs for each case and the mean of computational times is taken. Default `value` is 5. In fact, `value+2` benchmarks are executed and the two worst are forgotten (see `fc_bench.mean_run` function)

- `'comment'`, string or cell of strings displayed before running the benchmarks. If `value` is a cell of strings, after printing the `value`, 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 use `'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 comparative errors between the various functions are desired when displaying. In this case an handle function must be given which returns error (as scalar) between the output of the first function `Lfun{1}` and one of the others.

3.1 Matricial product examples

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

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

Let $m = n = p$.

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

Listing 5: `fc_bench.demos.setMatProd00` function

The `fc_bench.demos.setMatProd00` function given in Listing 5 is used in `fc_bench.demos.bench_MatProd00` script (file `+fc_bench/+demos/bench_MatProd00.m` of the package directory)

Listing 6: : fc_bench.demos.bench_MatProd00 script

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

Output

```
#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
#date:2018/04/26 11:10:53
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
500 0.003
1000 0.008
1500 0.019
2000 0.039
2500 0.081
3000 0.198
3500 0.239
4000 0.336
```

3.1.2 Square matrices: fc_bench.demos.bench_MatProd01 script

Let $m = n = p$.

```
function [Inputs, bDs]=setMatProd01(m,verbose,varargin)
X=randn(m,m); Y=randn(m,m);
if verbose
fprintf('#_1st_input_parameter: %m-by-%m matrix\n')
fprintf('#_2nd_input_parameter: %m-by-%m matrix\n')
end
bDs{1}=fc_bench.bdata('m',m,'%d',5); % first column in bench output
Inputs={X,Y}; % is the inputs of the matricial product functions
end
```

Listing 7: fc_bench.demos.setMatProd01 function

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

Listing 8: : fc_bench.demos.bench_MatProd01 script

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

Output

```
#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#benchfile: benches/MadProd01.out
#date:2018/04/26 11:11:02
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
500 0.002
1000 0.010
1500 0.031
2000 0.046
2500 0.094
3000 0.191
3500 0.260
4000 0.344
```

```
#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
#
#benchfile: benches/MadProd01.out
#date:2018/04/26 11:11:02
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: m mtimes(s)
500 0.002
1000 0.010
1500 0.031
2000 0.046
2500 0.094
3000 0.191
3500 0.260
4000 0.344
```

Listing 9: Output file benches/MadProd01.out

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

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` package directory)

Listing 11: `fc_bench.demos.bench_MatProd02` script

```

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

```

Output

```

#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#benchfile: benches/MadProd02.out
#date:2018/04/26 11:11:12
#nbruns:5
#numpy: i4      f4
#format: %d      %.3f
#labels: m      mtimes(s)
      500      0.002
      1000     0.008
      1500     0.025
      2000     0.051
      2500     0.105
      3000     0.130
      3500     0.243
      4000     0.331

```

```

-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
-----
# benchmarking function @(X,Y) mtimes(X,Y)
# where X and Y are m-by-m matrices
-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
-----
#benchfile: benches/MatProd02.out
#date:2018/04/26 11:11:12
#nbruns:5
#numpy: 14          f4
#format: %d         % .3f
#labels: m          mtimes(s)
500      0.002
1000     0.008
1500     0.025
2000     0.051
2500     0.105
3000     0.130
3500     0.243
4000     0.331
-----

```

Listing 12: Output file benches/MatProd02.out

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

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

```

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

```

Listing 13: fc_bench.demos.matprod01 function

The `fc_bench.demos.setMatProd02` function given in Listing 10 is used in `fc_bench.demos.bench_MatProd03` script (file `bench_MatProd03.m` of the `+fc_bench/+demos` package directory)

Listing 14: : fc_bench.demos.bench_MatProd03 script

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

Output

```
#-----
# computer: cosmos
# system: Ubuntu 17.10 (x86_64)
# processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
# (1 procs/14 cores by proc/2 threads by core)
# RAM: 62.6 Go
# software: Octave
# release: 4.2.1
#-----
# benchmarking matricial product functions
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#date:2018/04/26 11:11:21
#nbruns:5
#numpy: i4          f4    f4          f4
#format: %d          %.3f  %.3f          %.3f
#labels: m  mtimes(s) @ (s)  matprod02(s)
100        0.000  0.000        0.117
200        0.001  0.006        0.535
300        0.001  0.003        1.256
400        0.002  0.001        2.374
```

As the second handle function in `Lfun` has no name, the guess name is `@`. One can set a more convenient name by using the `'names'` option: this is the object of Listing 15. When empty value is set in `'names'` cell then a guessed name is used.

Listing 15: : fc_bench.demos.bench_MatProd04 script

```
Lfun={@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y) };
names={'mtimes(X,Y)', 'X*Y', ''};
Comment={'#_benchmarking_functions_@(X,Y)_mtimes(X,Y)_and_(X,Y)_...
    X*Y', ...
    '#_where_X_and_Y_are_m-by-m_matrices'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'names',names, 'info',false);
```

Output

```
#-----
# benchmarking functions @(X,Y) mtimes(X,Y) and @(X,Y) X*Y
# where X and Y are m-by-m matrices
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#date:2018/04/26 11:11:51
#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        0.117
200        0.000  0.000        0.550
300        0.001  0.001        1.276
400        0.002  0.001        2.319
```

3.1.5 Square matrices: fc_bench.demos.bench_MatProd05 script

As previous section, we want to compare computational times between the `mtimes(X,Y)` function, the `X*Y` command and the `fc_bench.demos.matprod01` function given in Listing 13. In addition, we also want to display errors between the outputs of the functions. The first function is the reference one and errors are always computed by using output of this reference function and output of the functions.

Two examples, using the `fc_bench.bench` function with `'error'` option to display comparative errors, are proposed. They both use the `fc_bench.demos.setMatProd02` function given in Listing 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 automatically print some informations on labels columns.

```

Listing 16 : fc_bench.demos.bench_MatProd05 script
-----
Lfun=@(X,Y) mtimes(X,Y), @(X,Y) X*Y, @(X,Y) ...
    fc_bench.demos.matprod02(X,Y) );
names={'mtimes(X,Y)', 'X*Y', ''};
Comment={'# Benchmarking functions: ' ...
    '# A1=mtimes(X,Y) (reference)', ...
    '# A2=X*Y', ...
    '# A3=fc_bench.demos.matprod02(X,Y)', ...
    '# where X and Y are m-by-m matrices', ...
    '# cmpErr[1] is the norm(A1-A2,Inf)', ...
    '# cmpErr[2] is the norm(A1-A3,Inf)'};
error=@(o1,o2) norm(o1-o2,Inf);
setfun=@(varargin) fc_bench.demos.setMatProd02(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
    'names',names, 'error',error, 'info',false);
-----

Output

#-----
# Benchmarking functions:
#   A1=mtimes(X,Y) (reference)
#   A2= X*Y
#   A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are m-by-m matrices
# cmpErr[1] is the norm(A1-A2,Inf)
# cmpErr[2] is the norm(A1-A3,Inf)
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#date:2018/04/26 11:12:22
#nbruns:5
#numpy:  i4          f4      f4          f4          f4          f4
#format: %d          %.3f   %.3f          %.3e          %.3f          %.3e
#labels: m  mtimes(X,Y) (s) X*Y(s)  cmpErr[1]  matprod02(s)  cmpErr[2]
   100      0.000   0.000   0.000e+00    0.118   3.235e-13
   200      0.001   0.000   0.000e+00    0.554   1.235e-12
   300      0.001   0.001   0.000e+00    1.299   2.174e-12
   400      0.002   0.001   0.000e+00    2.366   3.475e-12

```

Listing 17: : fc_bench.demos.bench_MatProd05bis script

```

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

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

```

Output

```

#-----
# Benchmarking functions:
# fun[0], mtimes(X,Y): @(X, Y) mtimes (X, Y)
# fun[1],    X*Y: @(X, Y) X * Y
# fun[2], matprod02: @(X, Y) fc_bench.demos.matprod02 (X, Y)
# cmpErr[1], error between fun[0] and fun[i] outputs computed with function
#    @(o1, o2) norm (o1 - o2, Inf)
# where
# - 1st input parameter is the output of fun[0]
# - 2nd input parameter is the output of fun[i]
#-----
# 1st input parameter: m-by-m matrix
# 2nd input parameter: m-by-m matrix
#-----
#date:2018/04/26 11:12:53
#nbruns:5
#numpy:  i4          f4      f4          f4          f4          f4
#format: %d          %.3f   %.3f          %.3e          %.3f          %.3e
#labels: m  mtimes(X,Y) (s) X*Y(s)  cmpErr[1]  matprod02(s)  cmpErr[2]
100      0.000  0.000  0.000e+00  0.115  3.252e-13
200      0.000  0.000  0.000e+00  0.554  1.289e-12
300      0.001  0.001  0.000e+00  1.255  2.094e-12
400      0.002  0.001  0.000e+00  2.351  3.512e-12

```

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

As previous section, we want to compare computational times between the `mtimes(X,Y)` function, the `X*Y` command and the `fc_bench.demos.matprod01` function given in Listing 13 but this time with non-square matrices. In addition, we also want to display errors between the outputs of the functions. The first function is the reference one and errors are always computed by using output of this reference function and output of the functions.

```

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

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

    if verbose
        if isreal(X), name=class(X); else, name=['complex_',class(X)]; end
        Fprintf('#_1st_input_parameter:_m-by-n_matrix_[%s]\n',name)
        if isreal(Y), name=class(Y); else, name=['complex_',class(Y)]; end
        Fprintf('#_2nd_input_parameter:_n-by-p_matrix_[%s]\n',name)
    end

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

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

```

Listing 18: fc_bench.demos.setMatProd03 function

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

Listing 19: : fc_bench.demos.bench_MatProd06 script

```

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

```

Output

```

#-----
# benchmarking functions:
#   A1=mtimes(X,Y) (reference)
#   A2= X*Y
#   A3= fc_bench.demos.matprod02(X,Y)
# where X and Y are m-by-m matrices
# Error[1] is the norm(A1-A2,Inf)
# Error[2] is the norm(A1-A3,Inf)
#-----
# 1st input parameter: m-by-n matrix [complex double]
# 2nd input parameter: n-by-p matrix [single]
#-----
#date:2018/04/26 11:13:23
#nbruns:5
#numpy:   i4   i4   i4           f4   f4           f4           f4           f4
#format:  %d   %d   %d           %.3f  %.3f           %.3e           %.3f           %.3e
#labels:  m    n    p   mtimes(X,Y) (s) X*Y (s)   cmpErr[1]   matprod01 (s)   cmpErr[2]
100    50   100           0.000  0.000   0.000e+00   2.895   1.047e-04
150    50   100           0.000  0.000   0.000e+00   4.332   9.835e-05
200    50   100           0.000  0.000   0.000e+00   5.794   1.029e-04
150    100  300           0.001  0.001   0.000e+00  25.966   4.968e-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 `bench` package, of the `fc_bench.demos.permLU` function. The `fc_bench.demos.setLU00` function given in Listing 21 is used in the script `fc_bench.demos.bench_LU00` (file `bench_LU00.m` of the `+fc_bench/+demos` package directory). The source code and the printed output are given in Listing 22.

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

Listing 21: `fc_bench.demos.setLU00` function

Listing 22 : `fc_bench.demos.bench_LU00` script

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

Output

```
#-----
# benchmarking fc_bench.demos.permLU function (LU factorization)
#-----
# input parameter: m-by-n matrix [double]
#-----
#date:2018/04/26 11:17:57
#nbruns:5
#numpy:   i4      f4
#format:  %d      %.3f
#labels:  m      permLU(s)
          100    0.137
          200    0.559
          300    1.294
          400    2.426
```

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:

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

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

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

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

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

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;
    Ffprintf=R.ffprintf;
    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
        Ffprintf('#_Prototype_ functions _without_ wrapper:_...
                [L,U,P]=fun(A)\n',class(A))
        Ffprintf('#_Input_parameter_A:_m-by-n_matrix_[%s]\n',class(A))
        Ffprintf('#_Outputs_are_[L,U,P]_such_that_P*A=L*U\n')
        Ffprintf('#_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: it converts the three outputs `[L,U,P]` of the `fc_bench.demos.permLU` in a 1-by-3 cell array `{L,U,P}`. We give in Listing 24 an example of a such function for a generic LU factorization function given by a function handle named `fun`.

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

Listing 24: `fc_bench.demos.wrapperLU` function

Listing 25 : fc_bench.demos.bench_LU01 script

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

Output

```
#-----
# benchmarking LU factorization functions
#-----
# Prototype functions without wrapper: [L,U,P]=fun(A)
# Input parameter A: m-by-n matrix [double]
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
# @ (L, U, P) norm (L + U - P * A, Inf)
#-----
#date:2018/04/26 11:18:28
#nbruns:5
#numpy:   i4          f4          f4
#format:  %d          %.3f         %.3e
#labels:  m    permLU(s)    Error[0]
          100      0.137    9.425e-14
          200      0.575    3.425e-13
          300      1.314    7.441e-13
          400      2.403    1.251e-12
```

3.2.3 fc_bench.demos.bench_LU02

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

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

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

Listing 26 : fc_bench.demos.bench_LU02 script

```

Lfun=@(A) fc_bench.demos.wrapperLU(@lu,A), ...
      @(A) fc_bench.demos.wrapperLU(@(X) fc_bench.demos.permLU(X),A) };
names={'lu','permLU'};
Comment='#_benchmarking_LU_factorization_functions';
error=@(o1,o2) ...
      norm(o1{o1}-o2{o1},Inf)+norm(o1{o2}-o2{o2},Inf)+norm(o1{o3}-o2{o3},Inf);
setfun=@(varargin) fc_bench.demos.setLU01(varargin{:});
fc_bench.bench(Lfun, setfun, 'LN',100:100:400, 'comment',Comment, ...
              'names',names, 'error',error, 'info',false, 'labelsinfo',true);

```

Output

```

#-----
# benchmarking LU factorization functions
#-----
# Benchmarking functions:
# fun[0],      lu: @(A) fc_bench.demos.wrapperLU (@lu, A)
# fun[1],      permLU: @(A) fc_bench.demos.wrapperLU (@(X) fc_bench.demos.permLU (X), A)
# cmpErr[i],   error between fun[0] and fun[i] outputs computed with function
# @ (o1, o2) norm (o1 {i} - o2 {i}, Inf) + norm (o1 {2} - o2 {2}, Inf) + norm (o1 {3} - o2 ...
#             {3}, Inf)
# where
# - 1st input parameter is the output of fun[0]
# - 2nd input parameter is the output of fun[i]
#-----
# Prototype functions without wrapper: [L,U,P]=fun(A)
# Input parameter A: m-by-n matrix [double]
# Outputs are [L,U,P] such that P*A=L*U
# Error[i] computed with fun[i] outputs :
# @ (L, U, P) norm (L * U - P * A, Inf)
#-----
#date:2018/04/26 11:18:59
#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 0.012 7.050e-14 0.141 9.805e-14 8.282e-13
200 0.001 2.715e-13 0.572 3.511e-13 6.670e-12
300 0.002 5.993e-13 1.321 7.436e-13 1.479e-11
400 0.003 9.756e-13 2.366 1.283e-12 2.307e-11

```

Informations for git maintainers of the bench Octave package

git informations on the packages used to build this manual

```
-----  
name : fc-bench  
tag : 0.0.3  
commit : ebbb87485449672a9c0287c1482af3e78bba4cb3  
date : 2018-04-26  
time : 06:29:25  
status : -1  
-----
```

```
-----  
name : fc-tools  
tag : 0.0.22  
commit : ea3d8d2104c720330ce262923d2b942197f8e00d  
date : 2018-04-26  
time : 08:27:32  
status : -1  
-----
```

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

```
-----  
name : fctools  
tag :  
commit : f75855ddf94c6a449248f570db5b0a8186bb7e5f  
date : 2018-04-17  
time : 09:53:32  
status : 1  
-----
```

Using the remote configuration repository:

```
url      ssh://lagagit/MCS/Cuvelier/Matlab/fc-config  
commit   55114cd8d81f150bdc3e2fbc1627405721528e5a
```