BENCHMARKING **OCTAVE**, **R** AND **PYTHON** PLATFORMS FOR CODE PROTOTYPING IN DATA ANALYTICS AND MACHINE LEARNING

Harris Georgiou (MSc,PhD)
hgeorgiou@unipi.gr

FossComm 2018 @ 13-14 October, Heraklion, Greece

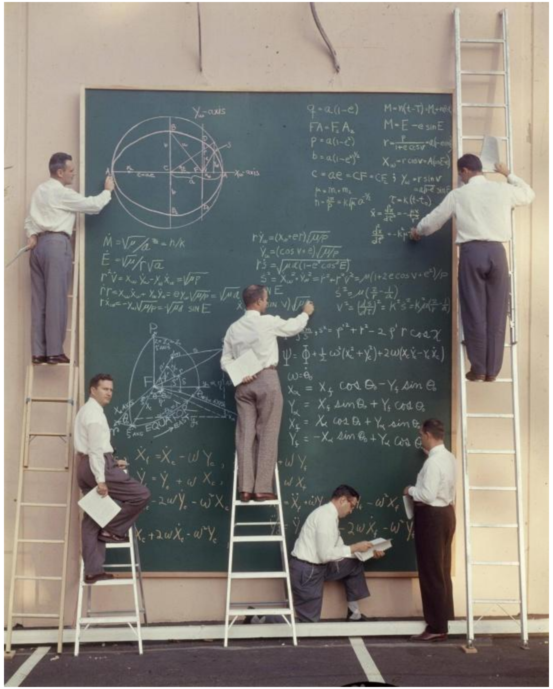
Challenge

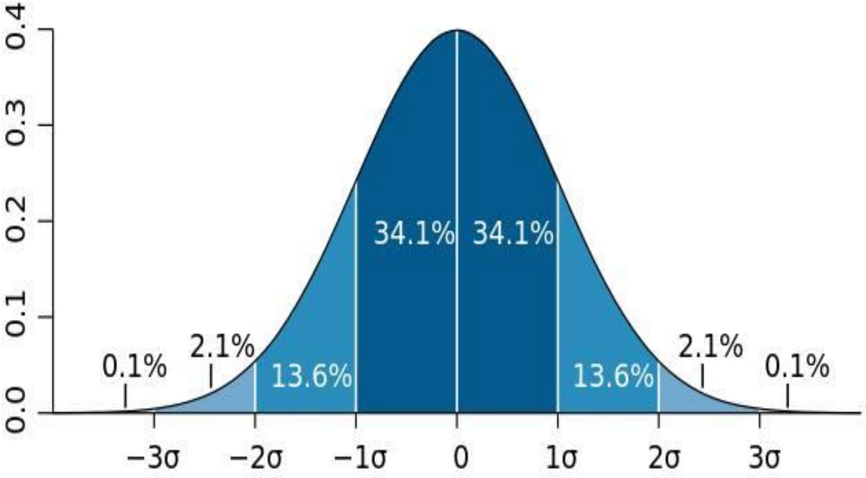
- Implementation of models & simulations are **essential** in almost all science topics.
- Code **prototyping** is an essential development stage in R&D.
- …But it is a special type of software development process, highly **iterative** (exploratory).
- Thus, special tools & platforms are needed.

17 Equations That Changed the World
by Ian Stewart

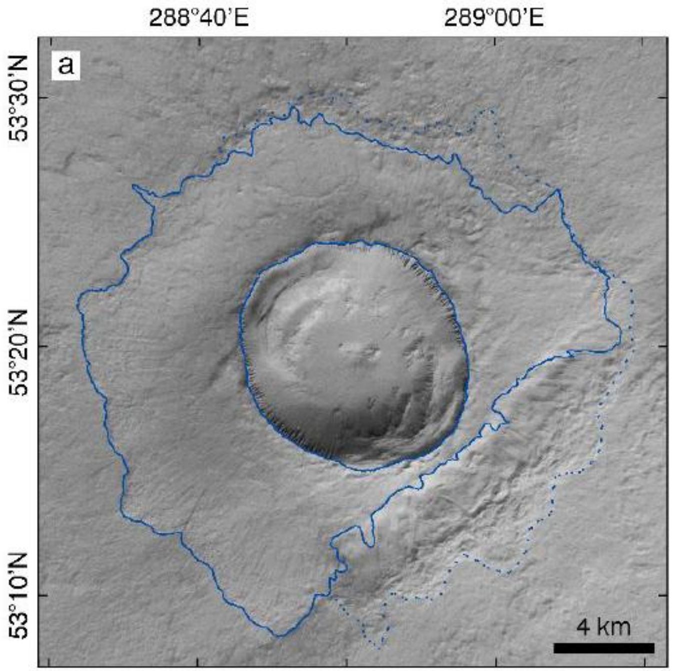
1.	**Pythagoras's Theorem**	$a^2 + b^2 = c^2$	Pythagoras,530 BC
2.	**Logarithms**	$\log xy = \log x + \log y$	John Napier, 1610
3.	**Calculus**	$\frac{\mathrm{d}f}{\mathrm{d}t} = \lim_{h \to 0} = \frac{f(t+h) - f(t)}{h}$	Newton, 1668
4.	**Law of Gravity**	$F = G\frac{m_1 m_2}{r^2}$	Newton, 1687
5.	**The Square Root of Minus One**	$i^2 = -1$	Euler, 1750
6.	**Euler's Formula for Polyhedra**	$V - E + F = 2$	Euler, 1751
7.	**Normal Distribution**	$\Phi(x) = \frac{1}{\sqrt{2\pi\rho}} e^{\frac{(x-\mu)^2}{2\rho^2}}$	C.F. Gauss, 1810
8.	**Wave Equation**	$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$	J. d'Almbert, 1746
9.	**Fourier Transform**	$f(\omega) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \omega} \mathrm{d}x$	J. Fourier, 1822
10.	**Navier-Stokes Equation**	$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$	C. Navier, G. Stokes, 1845
11.	**Maxwell's Equations**	$\nabla \cdot \mathbf{E} = 0$ $\quad \nabla \cdot \mathbf{H} = 0$ $\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{H}}{\partial t}$ $\quad \nabla \times \mathbf{H} = \frac{1}{c}\frac{\partial E}{\partial t}$	J.C. Maxwell, 1865
12.	**Second Law of Thermodynamics**	$\mathrm{d}S \geq 0$	L. Boltzmann, 1874
13.	**Relativity**	$E = mc^2$	Einstein, 1905
14.	**Schrodinger's Equation**	$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi$	E. Schrodinger, 1927
15.	**Information Theory**	$H = -\sum p(x) \log p(x)$	C. Shannon, 1949
16.	**Chaos Theory**	$x_{t+1} = kx_t(1 - x_t)$	Robert May, 1975
17.	**Black-Scholes Equation**	$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} - rV = 0$	F. Black, M. Scholes, 1990

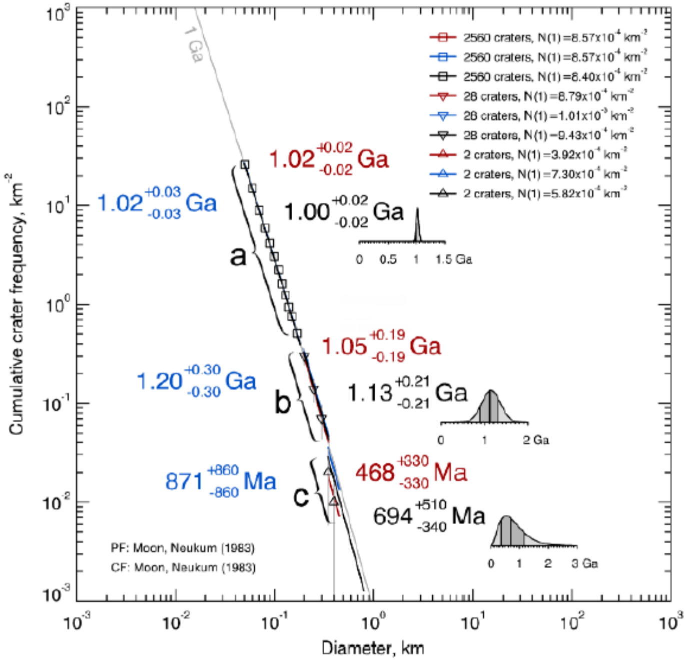
From ideas to working prototypes





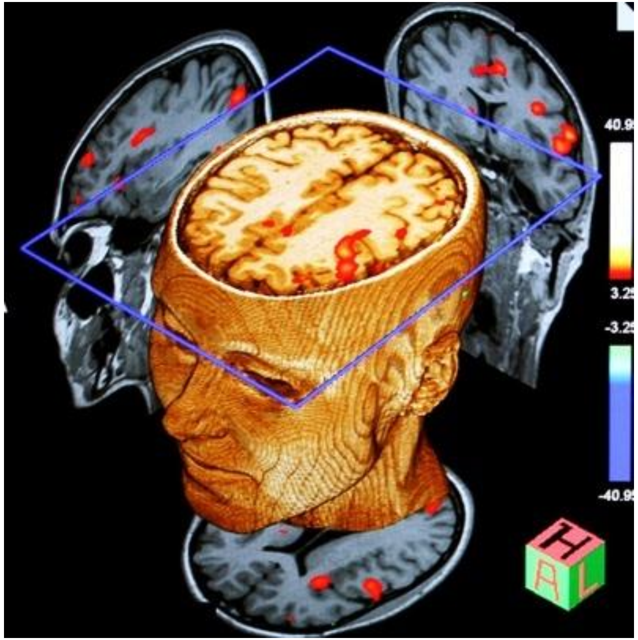
Example: Moon age estimation



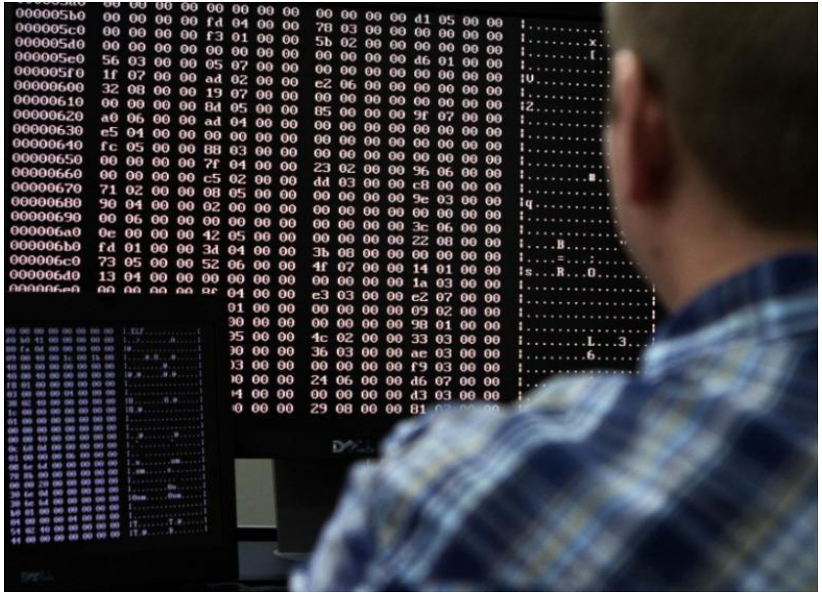


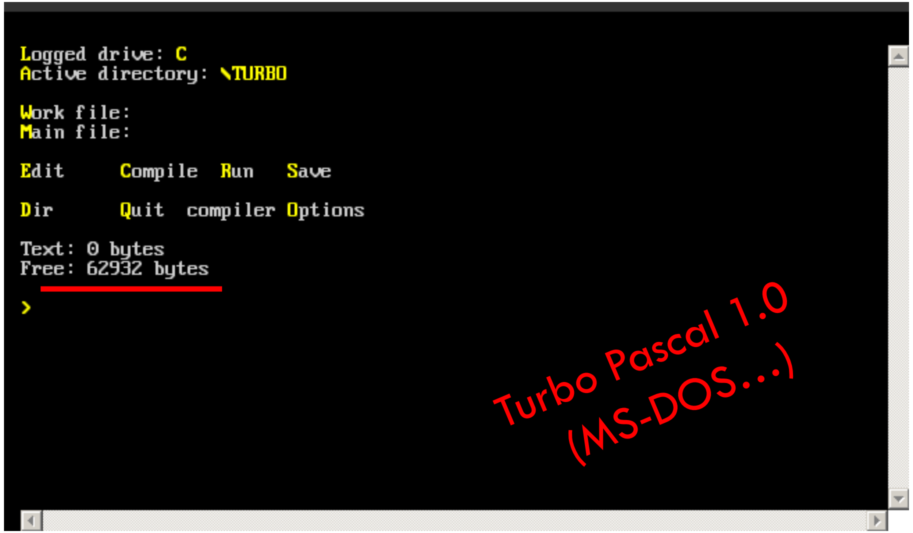
Example: Brain activity imaging

→

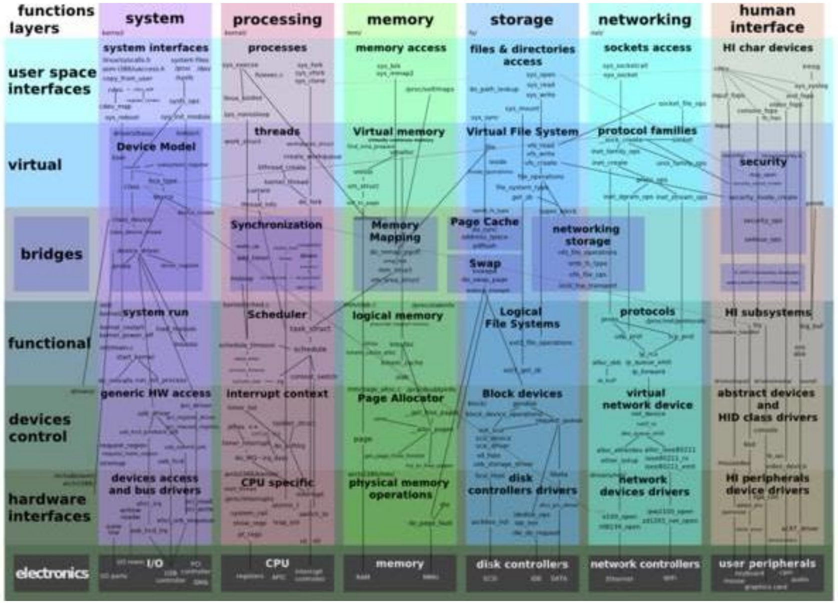


…But real-world coding is different

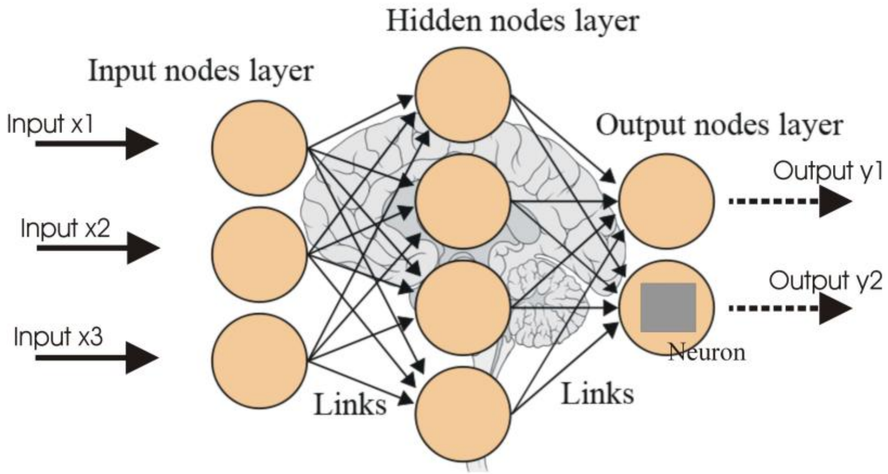




System vs. Model



Linux kernel map (2018)

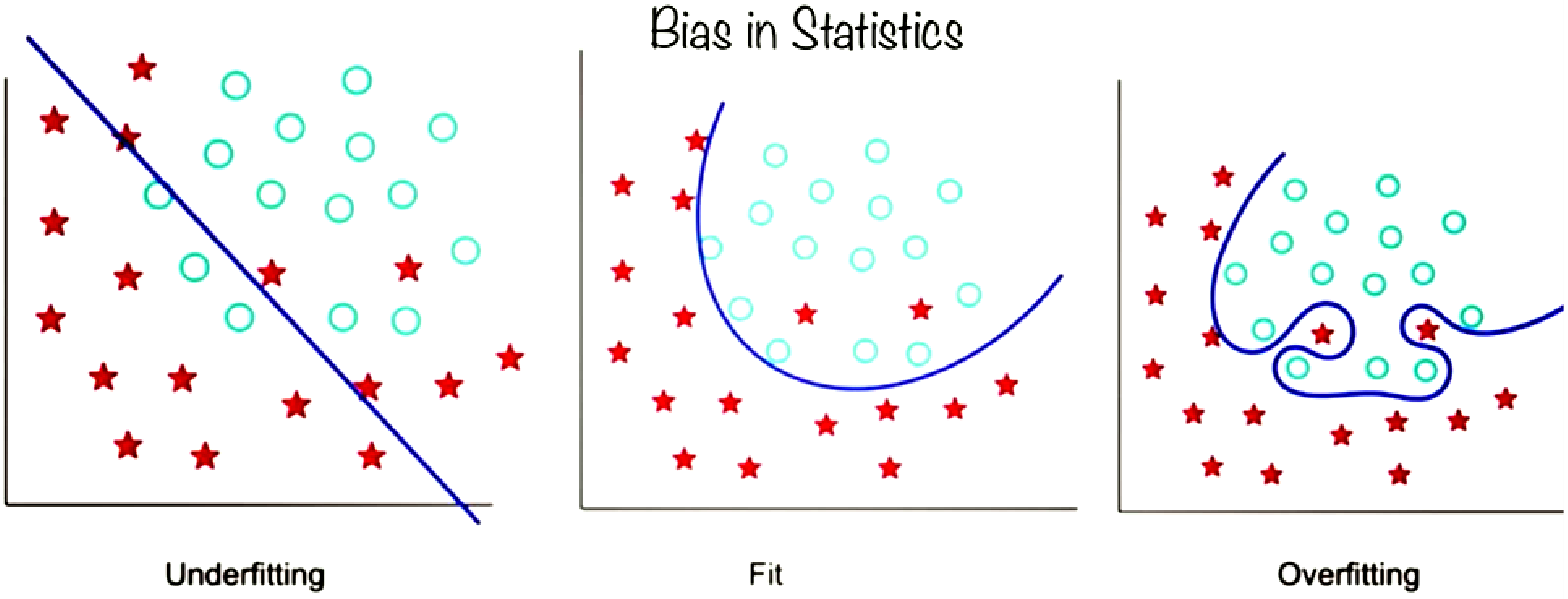


Typical NN model (BP-MLP)

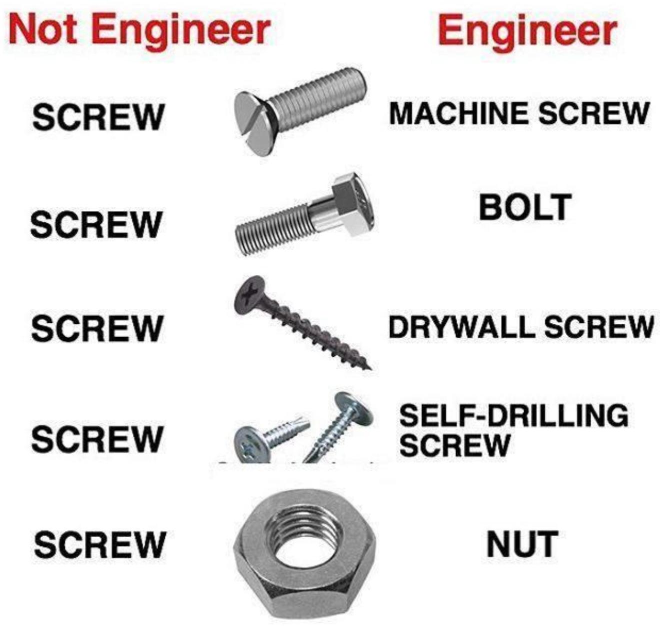
Requirements

- ✓ High-level programming.
- ✓ Good data abstractions.
- ✓ IDE, workspace (memory).
- ✓ Interpreted/script source code.
- ✓ Good data import/export.
- ✓ Extended supporting libs/APIs.
- ✓ Highly portable framework (OS).
- ✓ High-performance run-time core.
- ✓ Good community support (FOSS).

Exploratory Data Analytics



Use proper tools for each task



Experimental Setup

- Focus is on **Data Analytics** and **Machine Learning** code prototyping.
- Common algorithms and data operations were selected as benchmarks for run-time tests.
- **Octave**, **R** and **Python** were selected as the most appropriate, popular and API/package-rich platforms.

Implementations:
The exact same sequence of operations and loops has been coded as closely-matched as possible in the three coding platforms.

Platform versions:
- Octave : 4.4.1
- R : 3.5.1
- Python : 2.7.14

Benchmarks

Run-time tests included:
Pseudo-inverse matrix,
Linear equations system,
Linear Regression, SVD,
FFT, Bubblesort.

External APIs used:
None

(Python: numpy, sklearn)

- Main benchmark is execution time from the end-user perspective (elapsed).
- Timing mechanisms as provided in platform.
- Comparison on the same machine.
- Testing on multiple machines (low/high-end).
- Tests: Mostly matrix and vector operations, plus a reference Bubblesort implementation for testing branching/loop code segments.
- Multiple data matrix/vector sizes
(N = 100, 300, 500, 1000, 2000, 4000).

Machines used

Low-end, "embedded":

- OS: Ubuntu 16.04 LTS (kernel 4.4.0/i686)
- CPU: N270 Atom, 1x2L cores, 1.6 GHz
- Cache: 512 KB
- RAM: 2 GB

Mid-end, "office":

- OS: MS-Windows 8.1 (x64)
- CPU: i7-3537U, 2x2L cores, 2.0 GHz
- Cache: 4 MB
- RAM: 8 GB

High-end, "small server":

- OS: MS-Windows 10 (x64)
- CPU: i7-8550U, 2x2L cores, 1.8 GHz
- Cache: 8 KB
- RAM: 32 GB

Some additional comparative tests:

- 48x Xeon-X5675 (6 cores), 3.07 GHz, 12 MB cache, 48 GB RAM @ Ubuntu 16.04 LTS
- Mathworks Matlab 9.4.8 (R2018a/x64) @ Ubuntu 16.04 LTS, MS-Windows 8.1 & 10

Same model, multiple code variants

```python
Nsz=1000
Nlp=10

Tsum=[0]*6

print "Matrix size: %d , Iterations: %d\nStarting tests...\n" % (Nsz,Nlp)


print "[1]: Pseudo-inverse..."
i=1
while (i<=Nlp):
    A=numpy.random.randn(Nsz,Nsz)
    tlap=time.clock()

    B=numpy.linalg.inv(A.dot(A.transpose()))        # calculate pse

    Tsum[0]=Tsum[0]+(time.clock()-tlap)
    print "%g" % (B[1,1])
    i=i+1

Tsum[0]=Tsum[0]/Nlp


print "[2]: Linear system..."
i=1
while (i<=Nlp):
    A=numpy.random.randn(Nsz,Nsz)
    C=numpy.random.randn(Nsz,1)
    tlap=time.clock()

    B=numpy.linalg.solve(A,C)          # solve the linear system

    Tsum[1]=Tsum[1]+(time.clock()-tlap)
```

(source code: Python)

```r
Nsz <- 1000
Nlp <- 10

Tsum <- seq(0,0,length.out=6)

cat(sprintf('Matrix size: %d , Iterations: %d\nStarting tests...\n\n',


cat(sprintf('[1]: Pseudo-inverse...\n'))
for (i in 1:Nlp)
{
    A <- matrix(rnorm(Nsz*Nsz),Nsz,Nsz)
    I <- diag(nrow( A %*% t(A) ))
    tlap <- proc.time()

    B <- solve((A %*% t(A)),I)  # calculate pseudo-inverse

    Tsum[1] <- Tsum[1] + (proc.time() - tlap)[3]
    cat(sprintf('%g\n',B[1,1]))
}
Tsum[1] <- Tsum[1]/Nlp


cat(sprintf('[2]: Linear system...\n'))
for (i in 1:Nlp)
{
    A <- matrix(rnorm(Nsz*Nsz),Nsz,Nsz)
    C <- matrix(rnorm(Nsz),Nsz,1)
```

(source code: R)

```octave
Nsz=2000;
Nlp=30;

Tsum=zeros(6,1);

fprintf('Matrix size: %d , Iterations: %d\nStarting tests...\n\n',Nsz,Nlp);


fprintf('[1]: Pseudo-inverse...\n');
for i=1:Nlp
    A=randn(Nsz,Nsz);
    tlap=tic;

    B=(A*A')^-1;        % calculate pseudo-inverse

    Tsum(1)=Tsum(1)+toc(tlap);
    fprintf('%g\n',B(1,1));
end
Tsum(1)=Tsum(1)/Nlp;


fprintf('[2]: Linear system...\n');
for i=1:Nlp
    A=randn(Nsz,Nsz);
    C=randn(Nsz,1);
    tlap=tic;

    B=A\C;              % solve the linear system
```

(source code: Octave)

Set #1: Small-scale, low-end h/w

i=30 / N=100	octave	python	R	*matlab*
Pinv	0,009459	0,011474	0,016433	
Lin.sys	0,003832	0,002955	0,003867	
LSE.regr	0,012923	0,014731	0,013367	
SVD	0,015043	0,058328	0,063000	
FFT	0,007849	0,000507	0,005700	
Bsort	0,208750	0,011294	0,034733	

i=30 / N=100	octave	python	R	*matlab*
Pinv	1,000	1,213	1,737	
Lin.sys	1,297	1,000	1,308	
LSE.regr	1,000	1,140	1,034	
SVD	1,000	3,877	4,188	
FFT	15,493	1,000	11,251	
Bsort	18,484	1,000	3,076	

i=30 / N=300	octave	python	R	*matlab*
Pinv	0,169812	0,246665	0,318833	
Lin.sys	0,062735	0,051681	0,060633	
LSE.regr	0,131710	0,137957	0,084600	
SVD	0,367588	0,952583	0,968100	
FFT	0,061680	0,007901	0,081233	
Bsort	1,884780	0,108336	0,296267	

i=30 / N=300	octave	python	R	*matlab*
Pinv	1,000	1,453	1,878	
Lin.sys	1,214	1,000	1,173	
LSE.regr	1,557	1,631	1,000	
SVD	1,000	2,591	2,634	
FFT	7,807	1,000	10,282	
Bsort	17,398	1,000	2,735	

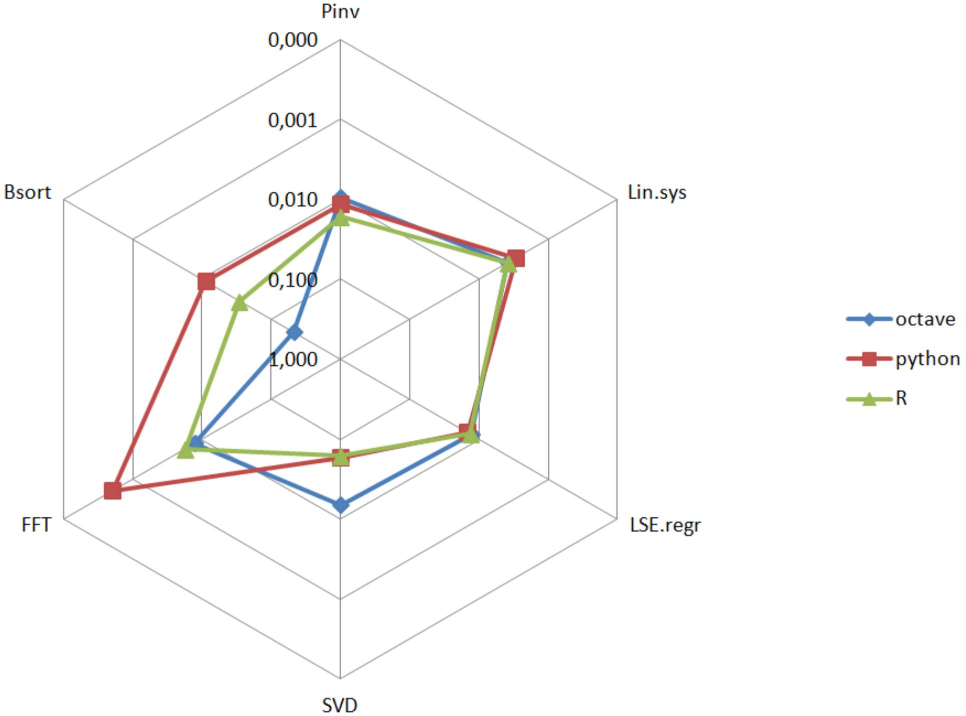
i=30 / N=500	octave	python	R	*matlab*
Pinv	0,668037	1,075870	1,343430	
Lin.sys	0,239858	0,214843	0,234567	
LSE.regr	0,477595	0,483087	0,310333	
SVD	1,304820	3,528020	3,595470	
FFT	0,024104	0,021135	0,276067	
Bsort	5,337370	0,306418	0,811767	

i=30 / N=500	octave	python	R	*matlab*
Pinv	1,000	1,610	2,011	
Lin.sys	1,116	1,000	1,092	
LSE.regr	1,539	1,557	1,000	
SVD	1,000	2,704	2,756	
FFT	1,140	1,000	13,062	
Bsort	17,419	1,000	2,649	

Results:
Rows are operations & algorithms, columns are platforms. Left matrix is execution times (sec), right matrix is ratios (1.0=fastest).

Set #1: Small-scale, low-end h/w

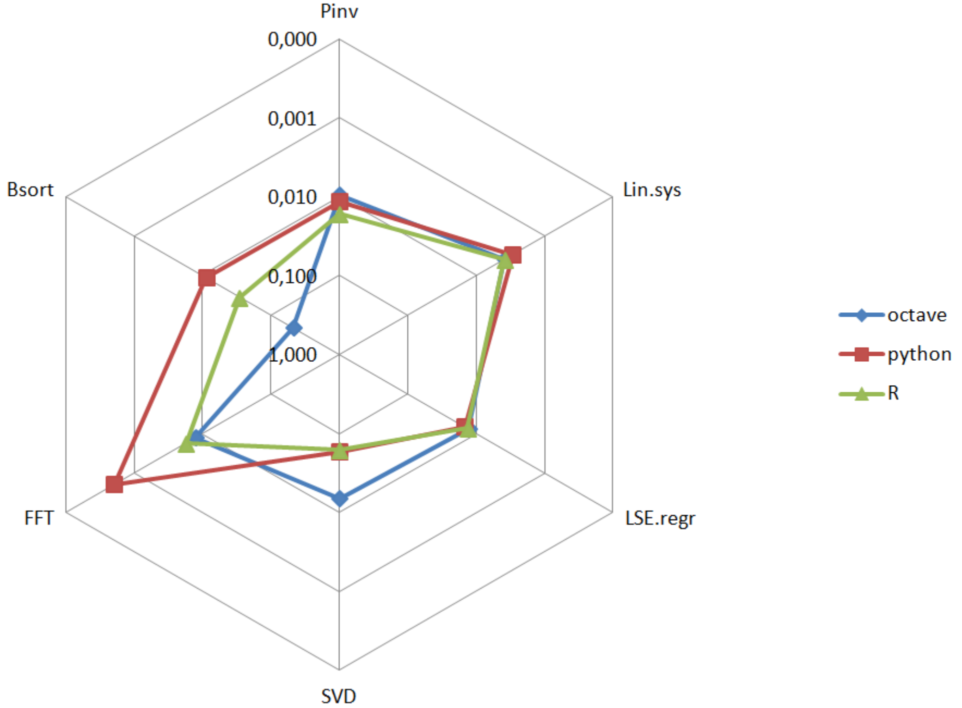
Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).



30 iterations
N=100

Set #1: Small-scale, low-end h/w

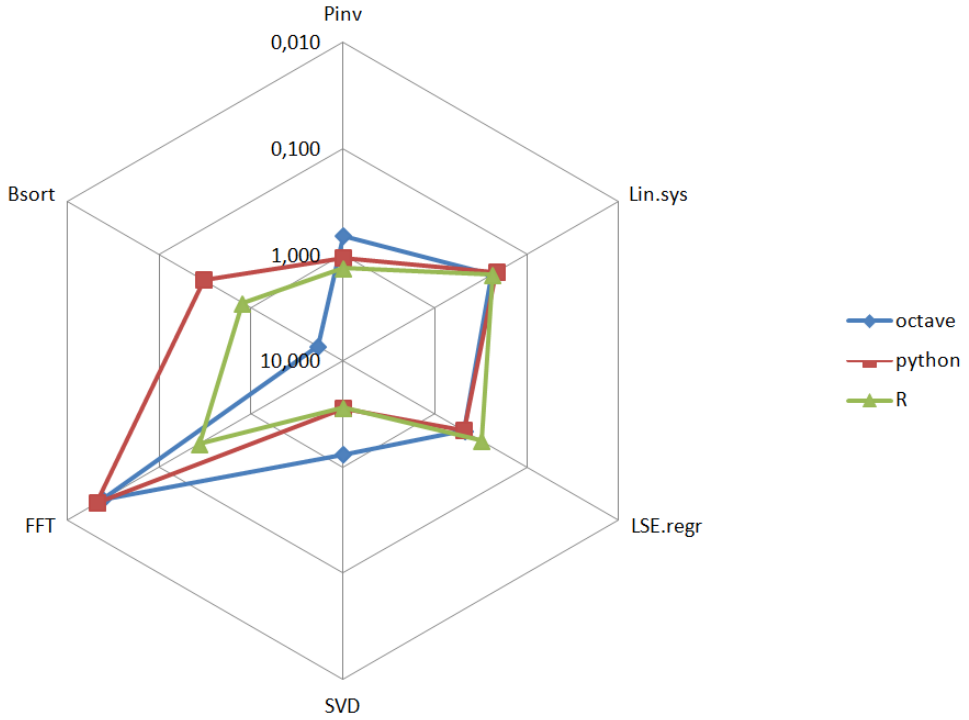
Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).



30 iterations
N=300

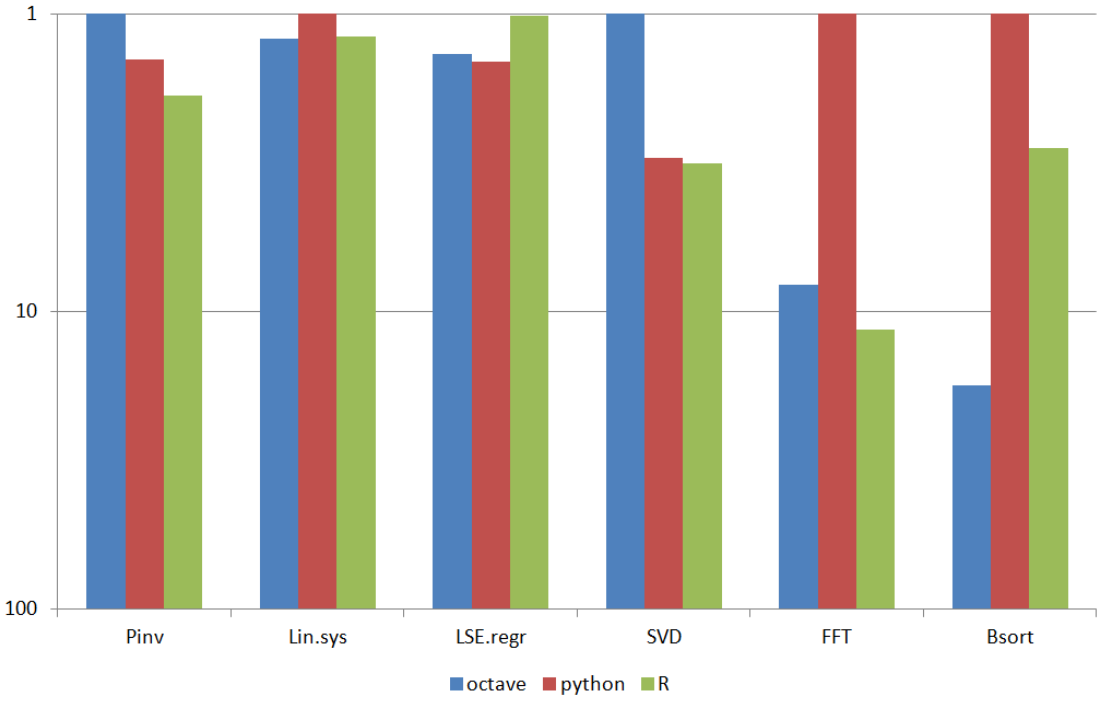
Set #1: Small-scale, low-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).



30 iterations
N=500

Set #1: Small-scale, low-end h/w



Bars plot:
Horizontal axis (groups) are operations & algorithms, colored bars are platforms, vertical axis value is performance ratio, mean over N sizes (1.0/top=fastest).

Set #2: Medium-scale, mid-end h/w

i=30 / **N=500**	octave	python	R	*matlab*
Pinv	0,023371	0,036059	0,188333	*0,058356*
Lin.sys	0,011733	0,011382	0,031333	*0,015995*
LSE.regr	0,037778	0,054364	0,028000	*0,013790*
SVD	0,067411	0,202288	0,413333	*0,045580*
FFT	0,006154	0,005057	0,016000	*0,035996*
Bsort	0,806911	0,043220	0,014667	*0,005150*

i=30 / **N=500**	octave	python	R	*matlab*
Pinv	1,000	1,543	8,059	2,497
Lin.sys	1,031	1,000	2,753	1,405
LSE.regr	2,740	3,942	2,030	1,000
SVD	1,479	4,438	9,068	1,000
FFT	1,217	1,000	3,164	7,118
Bsort	156,693	8,393	2,848	1,000

i=30 / **N=1000**	octave	python	R	*matlab*
Pinv	0,131107	0,234132	1,624330	*0,106040*
Lin.sys	0,050122	0,058332	0,206333	*0,038823*
LSE.regr	0,149848	0,229829	0,165667	*0,047169*
SVD	0,421308	1,230460	3,246670	*0,345464*
FFT	0,021400	0,017160	0,119000	*0,019820*
Bsort	3,291050	0,175126	0,055000	*0,003777*

i=30 / **N=1000**	octave	python	R	*matlab*
Pinv	1,236	2,208	15,318	1,000
Lin.sys	1,291	1,502	5,315	1,000
LSE.regr	3,177	4,872	3,512	1,000
SVD	1,220	3,562	9,398	1,000
FFT	1,247	1,000	6,935	1,155
Bsort	871,300	46,364	14,561	1,000

i=30 / **N=2000**	octave	python	R	*matlab*
Pinv	0,907910	1,598780	12,629000	*0,677876*
Lin.sys	0,259733	0,349798	1,516670	*0,233716*
LSE.regr	0,755758	1,167020	1,233330	*0,470988*
SVD	3,284050	7,810280	25,266700	*3,044780*
FFT	0,082671	0,067671	0,677333	*0,122214*
Bsort	13,225800	0,709954	0,224333	*0,009295*

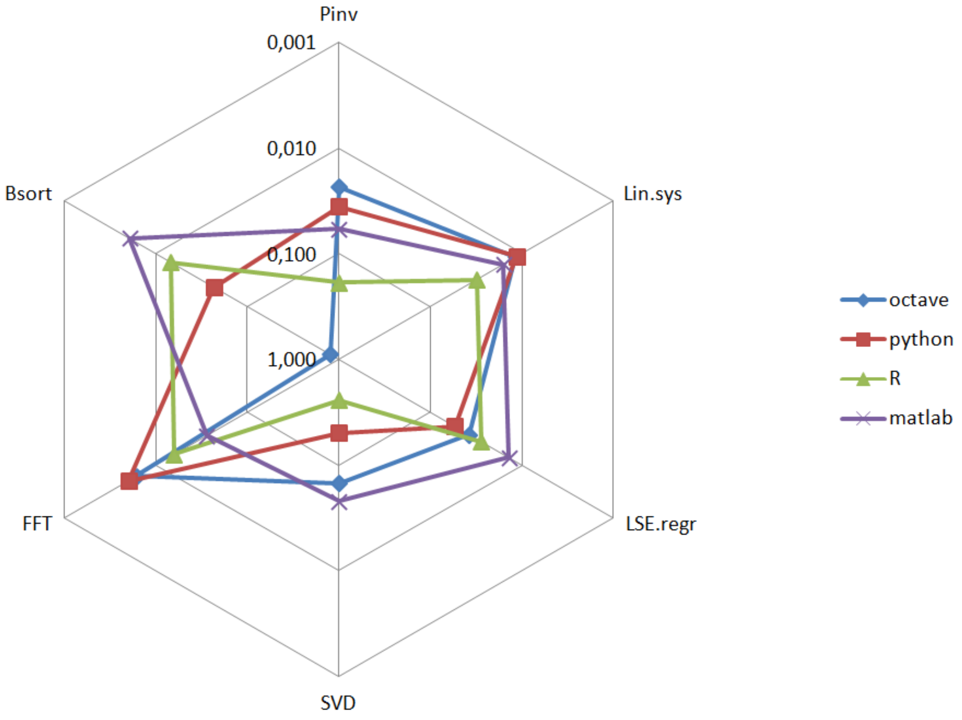
i=30 / **N=2000**	octave	python	R	*matlab*
Pinv	1,339	2,359	18,630	1,000
Lin.sys	1,111	1,497	6,489	1,000
LSE.regr	1,605	2,478	2,619	1,000
SVD	1,079	2,565	8,298	1,000
FFT	1,222	1,000	10,009	1,806
Bsort	1422,900	76,381	24,135	1,000

Results:
Rows are operations & algorithms, columns are platforms.
Left matrix is execution times (sec), right matrix is ratios (1.0=fastest).

Set #2: Medium-scale, mid-end h/w

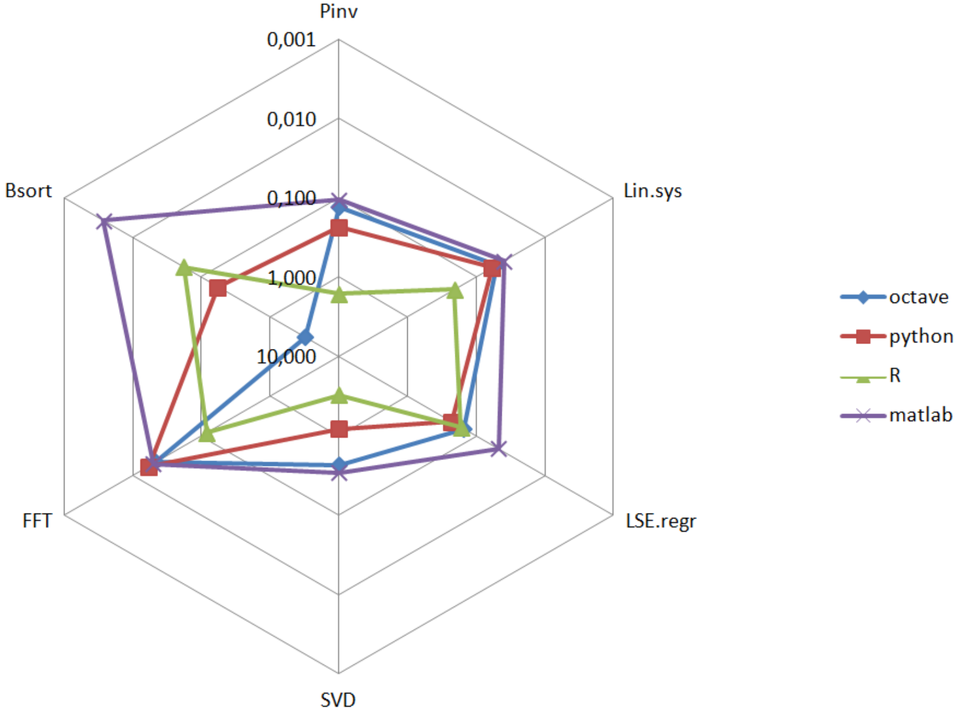
Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

30 iterations
N=500



Set #2: Medium-scale, mid-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

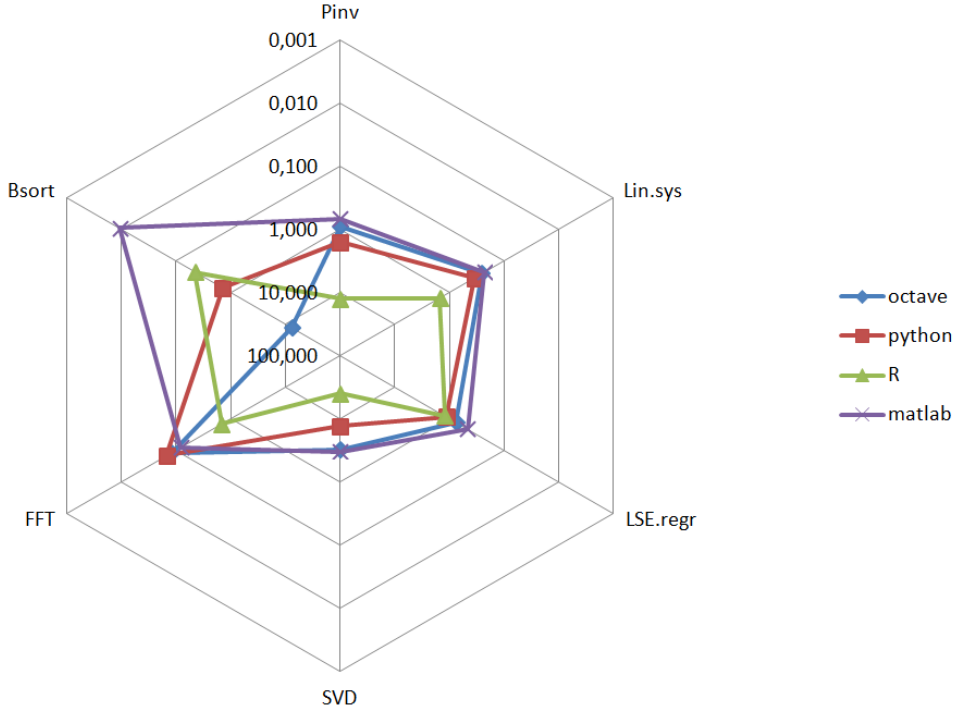


30 iterations
N=1000

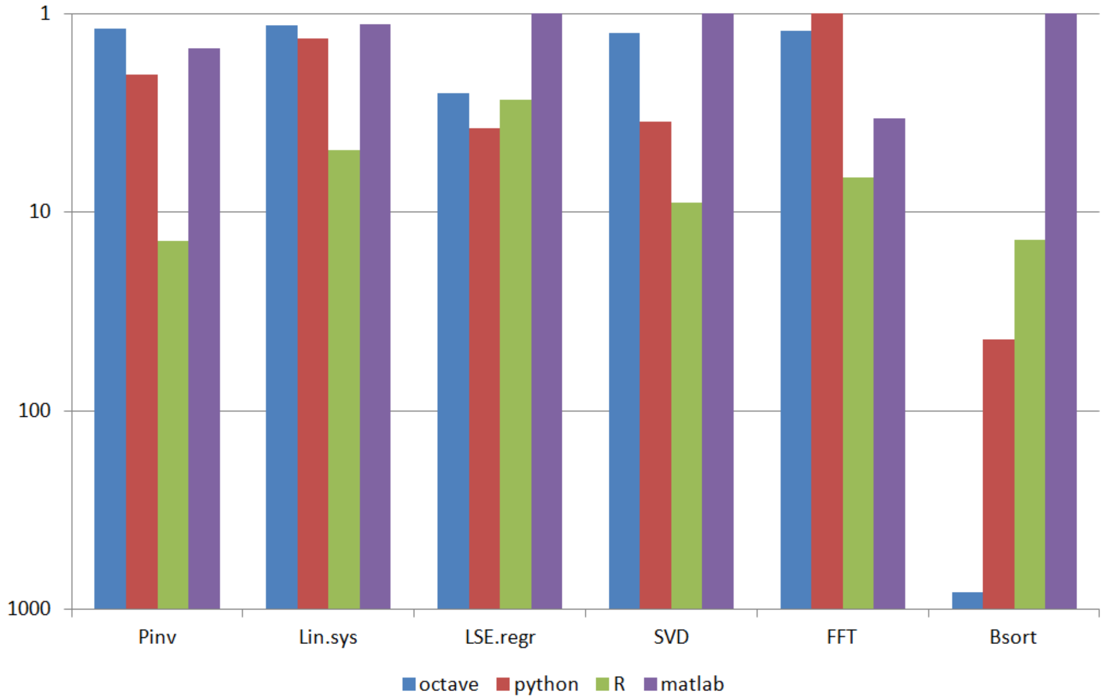
Set #2: Medium-scale, mid-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

30 iterations
N=2000



Set #2: Medium-scale, mid-end h/w



Bars plot:
Horizontal axis (groups) are operations & algorithms, colored bars are platforms, vertical axis value is performance ratio, mean over N sizes (1.0/top=fastest).

Set #3: Large-scale, high-end h/w

i=30 / **N=1000**	octave	python	R	*matlab*
Pinv	0,126029	0,113029	0,881000	*0,040521*
Lin.sys	0,054299	0,042061	0,125333	*0,016207*
LSE.regr	0,191812	0,212101	0,148667	*0,016863*
SVD	0,518621	0,966931	2,203330	*0,117619*
FFT	0,014278	0,016298	0,057667	*0,017238*
Bsort	2,258240	0,111375	0,036000	*0,004143*

i=30 / **N=1000**	octave	python	R	*matlab*
Pinv	3,110	2,789	21,742	1,000
Lin.sys	3,350	2,595	7,734	1,000
LSE.regr	11,375	12,578	8,816	1,000
SVD	4,409	8,221	18,733	1,000
FFT	1,000	1,141	4,039	1,207
Bsort	545,076	26,883	8,689	1,000

i=30 / **N=2000**	octave	python	R	*matlab*
Pinv	0,588381	0,829929	8,804670	*0,260603*
Lin.sys	0,242156	0,215862	0,967000	*0,086459*
LSE.regr	0,918587	1,094840	1,137670	*0,179034*
SVD	2,665340	5,552390	18,568700	*1,791460*
FFT	0,056744	0,058099	0,423333	*0,051546*
Bsort	12,334300	0,457412	0,135333	*0,005952*

i=30 / **N=2000**	octave	python	R	*matlab*
Pinv	2,258	3,185	33,786	1,000
Lin.sys	2,801	2,497	11,184	1,000
LSE.regr	5,131	6,115	6,354	1,000
SVD	1,488	3,099	10,365	1,000
FFT	1,101	1,127	8,213	1,000
Bsort	2072,299	76,850	22,737	1,000

i=30 / **N=4000**	octave	python	R	*matlab*
Pinv	3,853420	7,583870	72,096700	*2,148590*
Lin.sys	1,035350	1,427660	7,738000	*0,678165*
LSE.regr	4,006610	5,246910	9,510670	*2,136730*
SVD	20,194800	0,000000	155,019000	*16,066100*
FFT	0,213613	0,000000	2,483330	*0,294636*
Bsort	37,508100	2,028400	0,540333	*0,021809*

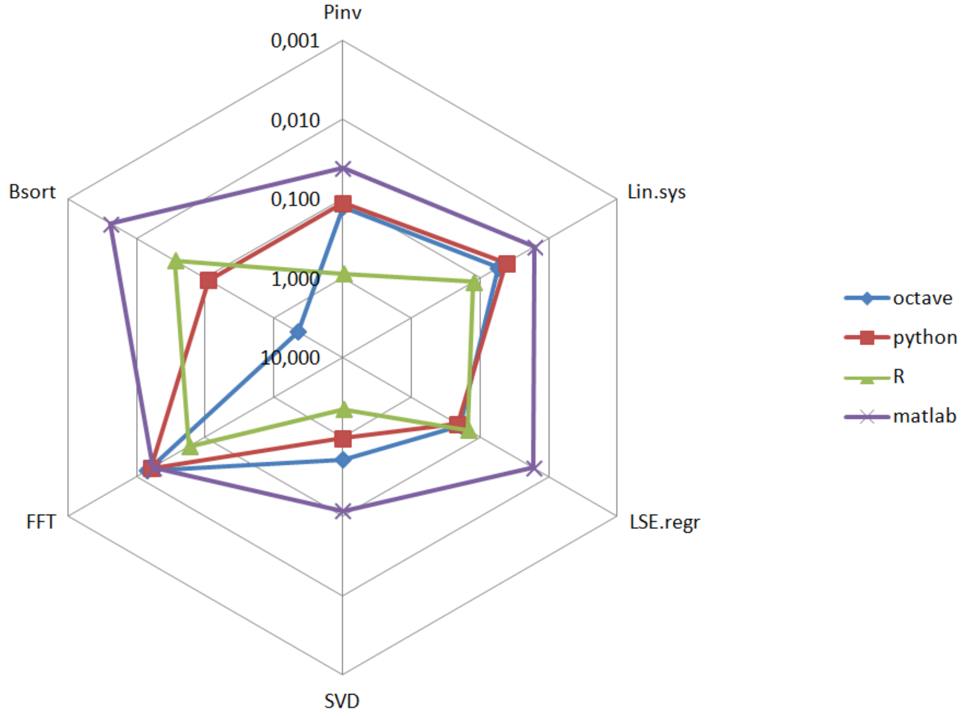
i=30 / **N=4000**	octave	python	R	*matlab*
Pinv	1,793	3,530	33,555	1,000
Lin.sys	1,527	2,105	11,410	1,000
LSE.regr	1,875	2,456	4,451	1,000
SVD	1,257	0,000	9,649	1,000
FFT	1,000	0,000	11,625	1,379
Bsort	1719,837	93,007	24,776	1,000

Results:
Rows are operations & algorithms, columns are platforms. Left matrix is execution times (sec), right matrix is ratios (1.0=fastest).

Set #3: Large-scale, high-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

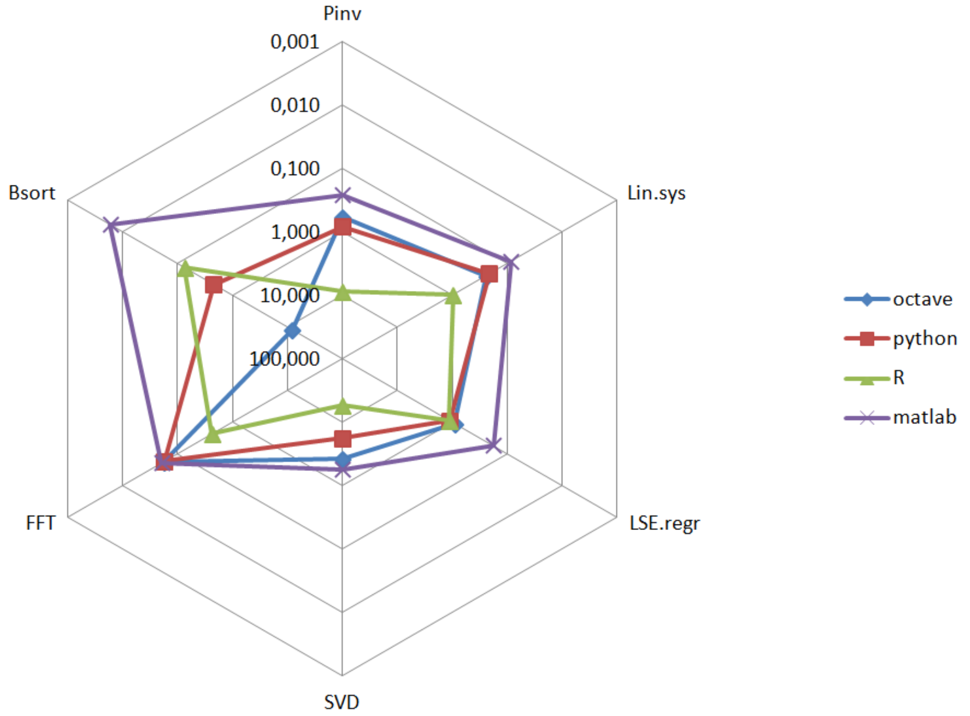
30 iterations
N=1000



Set #3: Large-scale, high-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

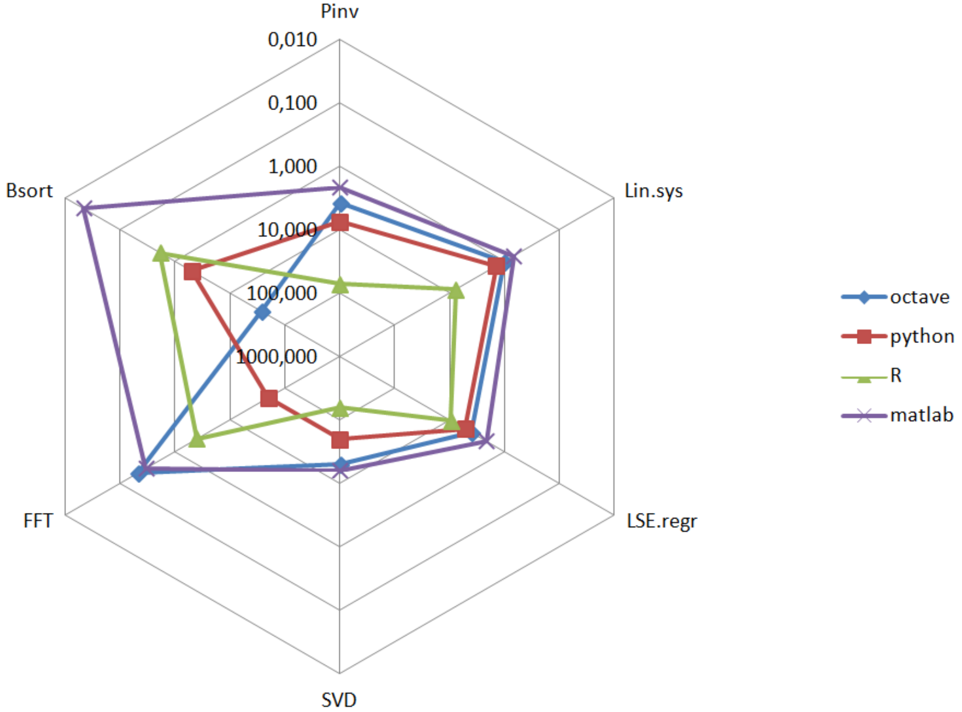
30 iterations
N=2000



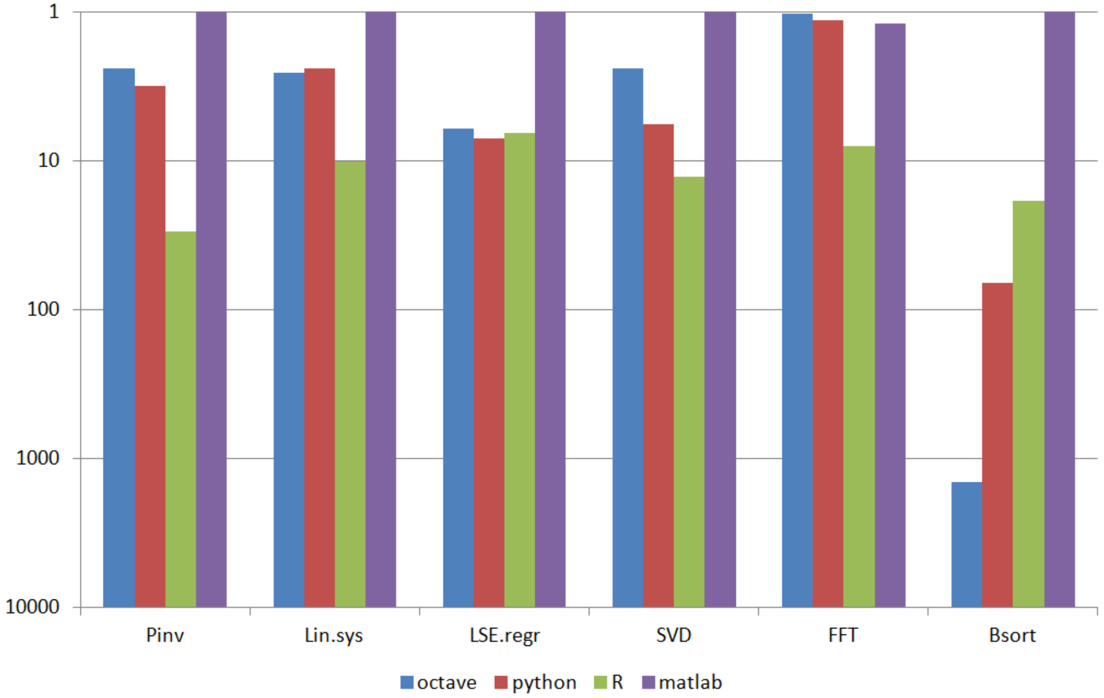
Set #3: Large-scale, high-end h/w

Profile Plot:
Colored lines are platforms
Axes are performance ratios in operations & algorithms (1.0/inner=fastest).

30 iterations
N=4000



Set #3: Large-scale, high-end h/w



Bars plot:
Horizontal axis (groups) are operations & algorithms, colored bars are platforms, vertical axis value is performance ratio, mean over N sizes (1.0/top=fastest).

Performance Assessment: **Octave**

- ☑ Superb performance on low-end, small-scale linear Algebra operations; Tops at 7 of 18 tests.
- ☑ Good FFT at medium/large-scale.

- ☒ Extremely bad performance in branching/loop code, at any scale.

Performance Assessment: **R**

- ☑ Good Linear Regression at small-scale.
- ☑ Fastest branching/loop execution at medium/large-scale.

- ☒ Performance of matrix operations (inverse, SVD) degrades quickly as scale increases.
- ☒ Overall stable, but slower in almost all tests.

Performance Assessment: **Python**

- ☑ Superb performance on low-end, small-scale linear operations, FFT and branching/loops; Tops at 9 of 18 tests.
- ☑ Good FFT at medium/large-scale.

- ☒ Unstable/crashes on some large-scale cases (FFT, SVD); out-of-memory errors.

Additional Comparative Tests

High-end server (48x Xeon 6-core CPUs):

- ☒ Interpreter core in all platforms does not take full advantage of the underlying hardware, even in fully parallelizable operations.
- ☒ All platforms seem to be oriented to single-thread code execution, except built-in native-code APIs/packages optimized at compile-time per-se.

Mathworks Matlab (non-FOSS, performance baseline):

- ☑ Superb performance on medium/large-scale operations in almost all cases, as expected; Tops at 26 of 32 tests.
- ☑ ...**But** Octave and Python are still competitive or even faster (e.g. FFT).

Conclusions

BAD practices:

☒ Using Octave for branching/loop operations, at **any** scale.

☒ Using R for speed-critical matrix operations.

☒ Using Python for large-scale matrix operations (N>2000).

GOOD practices:

☑ Using Octave for low-end, small-scale Algebra; FFT at medium/large-scale.

☑ Using R for linear regression; branching/loop at medium/large-scale.

☑ Using Python as default for mixed-type, all-scale projects (caution: FFT, SVD).

Questions…

FossComm 2018 @ 13-14 October, Heraklion, Greece

Harris Georgiou (MSc,PhD) – Email: hgeorgiou@unipi.gr

Δοκιμαστικό μοτίβο ευρείας οθόνης (16:9)

Δοκιμή αναλογιών εικόνας

(Πρέπει να εμφανίζεται κυκλικό)

4x3

16x9