Matlab - Octave science and engineering benchmarking and comparison

A. P. LEROS[1,2], A. ANDREATOS[2] and A. ZAGORIANOS[2]
[1]Department of Automation
School of Technological Applications
Technological Educational Institute of Chalkis
34400 Psachna, Evia
GREECE
[2]Department of Aeronautical Engineering
Hellenic Air Force Academy
Dekeleia, Attica, TGA-1010
GREECE
lerosapostolos@gmail.com, aandreatos@gmail.com, azagorianos@gmail.com

Abstract: - The purpose of this paper is to compare from various aspects two widely used numerical computation applications, namely Matlab and Octave, based on execution time performance tests. These tests implement three science and engineering applications which are carefully written to run on both Matlab and Octave. Based on our benchmark results as well as other factors for assessing software, we conclude that Octave is a noteworthy software for scientific and engineering applications and a free alternative to Matlab, adequate for most cases.

Key-Words: - Matlab, Octave, benchmarking, performance, mass-spring dumper system, PID controller, kalman filter, hanowa test matrices.

1 Introduction

Numerical computation applications such as Matlab [1] are used in Science and Engineering education worldwide in courses such as Mathematics, Physics, Simulation, Electronics, Electro-Magnetism, Telecommunications, Radars, Antennas, etc. An alternative free and open-source software (FOSS) is GNU Octave [2].

Although a lot have been written about the Matlab-Octave comparison [3, 4, 5, 6] and benchmarking, there is no complete application-oriented comparison known to us. Therefore we decided to proceed to this endeavour.

The objective of this paper is threefold: first, to present a benchmarking comparison using three scientific and engineering applications; second, to compare Matlab and Octave using other criteria; and third, to examine the degree to which Octave can substitute Matlab in science and engineering applications.

2 Octave – Matlab Benchmarking

Although some comparative benchmark results have been reported in the bibliography [6, 7] we have decided to perform our own tests for the following reasons:

a) Our purpose was to perform benchmarking using three complete science and engineering applications.
b) The code of these applications should be fully compatible with both Matlab and Octave;
c) These algorithms do not use any toolboxes so that all Matlab and Octave users can test them;
d) The very high code compatibility indicates the possible substitution of Matlab with Octave in a wide range of science and engineering applications.

In this section we describe the mathematical models of three science and engineering applications; we present their Octave and Matlab software coded simulations for the purpose of comparing their execution times and show their specific graphics results. The chosen simulated applications are: a) a Proportional Integral Derivative (PID) control of a Mass Spring Damper (MSD) system, b) a Kalman filter for vehicle state estimation using noisy position measurements, and c) a computationally demanding hanowa test matrix inversion problem. The choice for these three examples is that their

simulation involves enough and different commands to draw representative conclusions for the comparison of the execution times by both software.

2.1 PID control of an MSD System

A Mass-Spring-Damper (MSD) system, shown in Figure 1 below, can be described by the following second order differential equation [8]:

$$M\ddot{x}(t)+b\dot{x}(t)+kx(t)=F(t) \qquad \textbf{(1)}$$

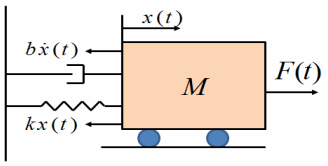

Figure 1: Mass-Spring-Damper System

The Laplace Transform of eq. (1) is:

$$Ms^2X(s)+bsX(s)+kX(s)=F(s) \qquad \textbf{(2)}$$

The system transfer function between the displacement $X(s)$ and the input $F(s)$ is:

$$G(s)=\frac{X(s)}{F(s)}=\frac{1}{Ms^2+bs+k} \qquad \textbf{(3)}$$

The standard PID control system configuration is shown in Figure 2 below [9, 10]. The control signal $u(t)$ is the sum of three terms. Each term is a function of the error signal $e(t)=[r(t)-y(t)]$, where $r(t)$ is the desired or reference input signal to be tracked and $y(t)$ is the output system response signal.

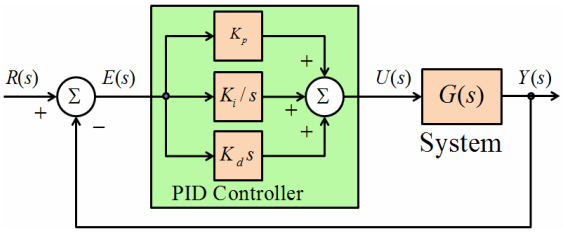

Figure 2: PID control system configuration

The K_p is a proportional term, the K_i/s is an integral term, and the $K_d s$ a derivative term. Although each control term appears to be independent of each other, in reality this is not exactly true since the whole thing works in the context of a closed-loop system.

The complete PID control signal $u(t)$ is:

$$u(t)=K_p e(t)+K_i\int e(t)dt+K_d\frac{de(t)}{dt} \qquad \textbf{(4)}$$

where K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain. The transfer function of the PID controller is:

$$\frac{U(s)}{E(s)}=K_p+\frac{K_i}{s}+K_d s=\frac{K_d s^2+K_p s+K_i}{s} \qquad \textbf{(5)}$$

Note 1: It is not uncommon for the integral gain to be related to the proportional gain by $K_i=K_p/\tau_i$, where τ_i is the integral time. Also since a differentiation is a "wide-band" process, meaning that its gain grows linearly with frequency, in general, the derivative term has an impractical nature. Moreover, when the error undergoes a sharp change, as when a step input is applied, the derivative term increases by a very large amount, and this requires an unreasonably large control effort. Thus in the implementation of the derivative term is very common to use the modified derivative, $u(t)=K_d\dot{x}(t)$, with $\dot{x}(t)=(e(t)-x(t))/\tau_1$, where the parameter τ_1 limits the bandwidth of the differentiator, and usually, $\tau_1=K_d/(NK_p)$, with $N\in[10,\ 20]$. The Laplace transform of $u(t)=K_d\dot{x}(t)$ is: $U(s)=K_d sX(s)=K_d sE(s)/(\tau_1 s+1)$.

Therefore, a modified practical PID controller is [9, 10, 11]:

$$\frac{U(s)}{E(s)}=K_p+\frac{K_i}{s}+\frac{K_d s}{\tau_1 s+1}=\frac{(K_p\tau_1+K_d)s^2+(K_p+K_i)s+K_i}{s(\tau_1 s+1)} \qquad \textbf{(6)}$$

The Octave – Matlab code to simulate the above modified PID control system configuration (eq. (6) and eq. (3)) is presented in Appendix A.

The execution of the code generates two plots, shown in Figure 3 below, for the comparison of the open-loop response with that of the PID controlled response. For the open-loop response, we scaled the step input by 20 so that the overall system output settled out to 1. From the open-loop response we observe that the DC gain of the plant transfer function is 1/20, so 0.05 is the final value of the output to an unit step input. This corresponds to the steady-state error of 0.95, which is quite large. In addition, the rise time is about one second, and the settling time is about 1.5 seconds. From the closed-loop response we observe that the designed PID controller performs much better giving an overall system with fast rise time, no overshoot, reduced settling time, and with no steady-state error.

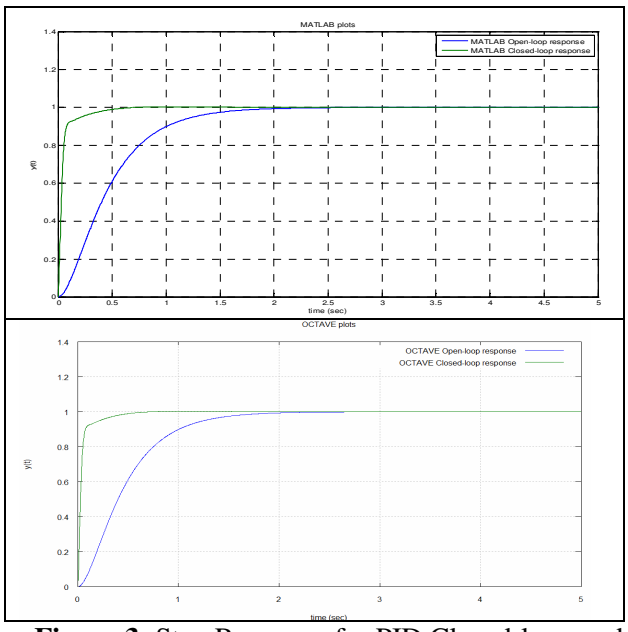

Figure 3: Step Response for PID Closed-loop and Open-Loop System by Matlab (top) and Octave (bottom)

2.2 Kalman filter for vehicle state estimation using noisy position measurements

For a vehicle moving in a straight line on the road if x_1 is its position and x_2 its velocity then from elementary physics laws (Newton's law) and under ideal conditions it is known that at every time instance Δt sec its position is given by $x_{1k+1} = x_{1k} + \Delta t x_{2k} + \left(\Delta t^2 / 2\right) u_k$ and its velocity by $x_{2k+1} = x_{2k} + \Delta t u_k$, where u_k is the acceleration. In vector form the linear model of the vehicle system in the state space $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ and in discrete form with input the acceleration u_k and output its position $y_k = x_{1k}$ is:

$$x_{k+1} = Fx_k + Bu_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \Delta t^2 / 2 \\ \Delta t \end{bmatrix} u_k \quad \textbf{(7a)}$$

$$y_k = Cx_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k \quad \textbf{(7b)}$$

However, for non ideal conditions as air resistance, potholes in the road, etc. the previous model should include different uncertainties which are described by the noise w_k at the acceleration with zero mean and standard deviation $\sigma_w (m / \sec^2)$. Also assuming that the system output is measured at every time instance Δt sec with an instrument which has zero mean error and standard deviation $\sigma_v (m)$, then the dynamic system model of the vehicle and the model of the measurements z_k are modified as follows:

$$x_{k+1} = Fx_k + Bu_k + w_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \Delta t^2 / 2 \\ \Delta t \end{bmatrix} u_k + w_k \quad \textbf{(8a)}$$

$$z_k = Hx_k + v_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v_k \quad \textbf{(8b)}$$

For the above stochastic system model of the vehicle (8α) and for the noisy measurements of its position (8β) every time instance Δt sec the algorithmic Kalman filter [3, 12, 13, 14] equations which give an estimate $\hat{x}_k$ of the state x_k, are:

$$K_k = P_k H^T (H P_k H^T + R)^{-1} \quad \textbf{(9a)}$$

$$\hat{x}_{k+1} = (F\hat{x}_k + Bu_k) + K_k (z_k - H\hat{x}_k) \quad \textbf{(9b)}$$

$$P_{k+1} = F(I - K_k H) P_k F^T + Q \quad \textbf{(9c)}$$

where K_k is the Kalman gain, P_k is the covariance of the error $e_k = \left(x_k - \hat{x}_k\right)$, $E\left\{w_k w_k^T\right\} = Q$ is the covariance of the noise w_k in the acceleration u_k which models the uncertainties in the process x_k and $E\left\{v_k v_k^T\right\} = R$ is the covariance of the noise v_k which models the instrument error in the measurements z_k.

For the determination of the covariance Q, it follows from the model (8a) that the position x_1 is $\Delta t^2 / 2$ times the acceleration u_k and the velocity x_2 is Δt times. Since the standard deviation of the acceleration is σ_w, then the standard deviation of the position $E\left\{x_1\right\}$ due to the acceleration noise standard deviation would be $E\left\{x_1\right\} = \left(\Delta t^2 / 2\right)\sigma_w$ and its covariance due to the acceleration noise covariance would be $E\left\{x_1^2\right\} = \left[\left(\Delta t^2 / 2\right)\sigma_w\right]^2 = \sigma_w^2 \Delta t^4 / 4$. Similarly, from the model (8b) the covariance of the velocity due to the acceleration noise covariance would be $E\left\{x_2^2\right\} = \left(\Delta t\right)^2 \sigma_w^2$. Finally, the covariance of the position with the velocity due to the acceleration noise covariance would be:

$$E\left\{x_1 x_2\right\} = \left[\left(\Delta t^2 / 2\right)\sigma_w\right]\left(\Delta t \sigma_w\right) = \sigma_w^2 \Delta t^3 / 2$$

Thus,

$$E\left\{w_k w_k^{\,T}\right\} = Q = E\begin{bmatrix} x_1^2 & x_1 x_2 \\ x_1 x_2 & x_1^2 \end{bmatrix} = \sigma_w^2 \begin{bmatrix} \Delta t^4/4 & \Delta t^3/2 \\ \Delta t^3/2 & \Delta t^2 \end{bmatrix} \quad \textbf{(10)}$$

For the measurements, similarly, since the measurement is one times the position x_1 and the standard deviation of the noise in the measurement position is σ_v, then the position standard deviation $E\{x_1\}$ due to the standard deviation of the noise in the position measurements would be $E\{x_1\} = 1 \cdot \sigma_v$ with covariance:

$$E\left\{v_k^2\right\} = R = E\left\{\left(1x_1\right)^2\right\} = E\left\{\left(1\sigma_v\right)^2\right\} = \sigma_v^2 \quad \textbf{(11)}$$

For the initialization of the Kalman filter algorithm we choose an estimate $\hat{x}_0$ for the state x_0 at the initial instance. Also we choose and some estimate for the covariance P_0 which represents the uncertainties for the estimate $\hat{x}_0$. If there is enough certainty for the initial estimate $\hat{x}_0$ then the value of P_0 is small, otherwise it is large, but after a few iterations of the algorithm these initial values for $\hat{x}_0$ and P_0 do not play important role in the performance of the filter. Thus for all simulations of the Kalman filter, in this example, these have been chosen to be always $x_0 = \hat{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ and $P_0 = Q$.

The rest parameters of the example they are left free to be determined from the user at his will as input for different simulation tests. These are the duration T in sec of the vehicle movement (simulation duration), the step $\Delta t = dt$ in sec, the standard deviation $\sigma_w = sigma\_w$ in $m/\sec^2$ of the acceleration noise w_k of the system, the standard deviation $\sigma_v = sigma\_v$ in m of the measurements noise v_k and the acceleration $u_k = u$ as input to the system.

The algorithmic function coded in Octave – Matlab of the Kalman filter of the system for the vehicle movement using noisy measurements of its position is presented in Appendix B.

Calling from Matlab and Octave command windows the Kalman_filter function with the specific values for its arguments as follows kalman_filter(100, 0.1, 0.03, 3, 0.3) gives the following plots:

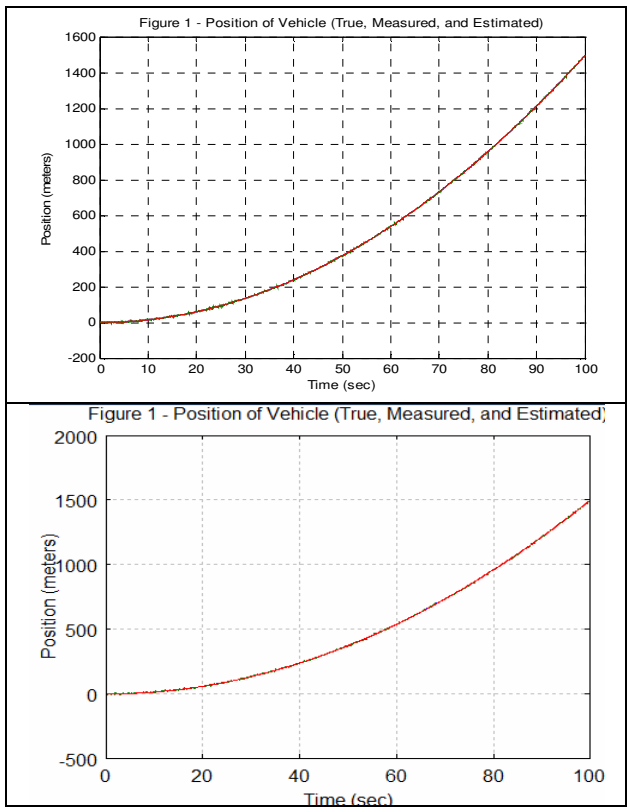

Figure 4a: Matlab (top) and Octave (bottom) Position of vehicle (True, Measured, and Estimated)

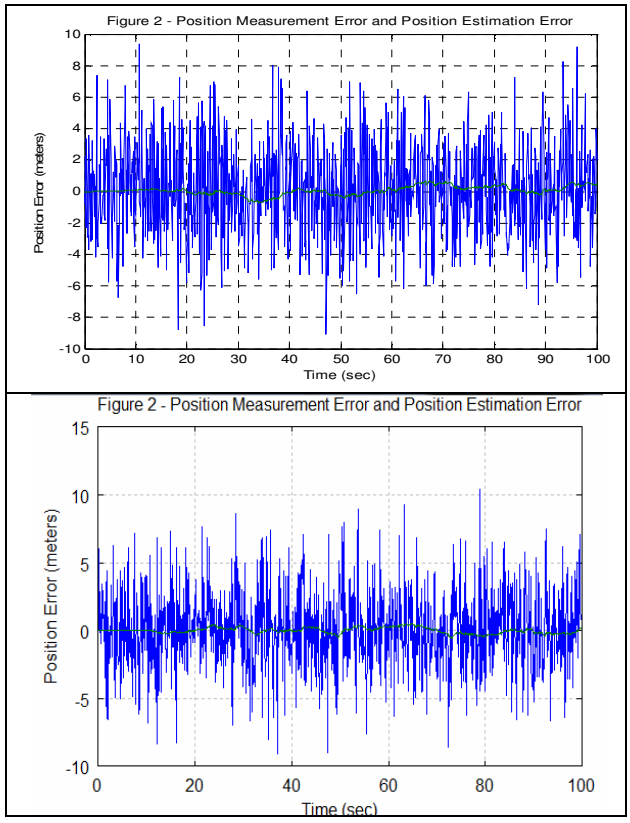

Figure 4b: Matlab (top) and Octave (bottom) Position Measurement Error and Position Estimated Error

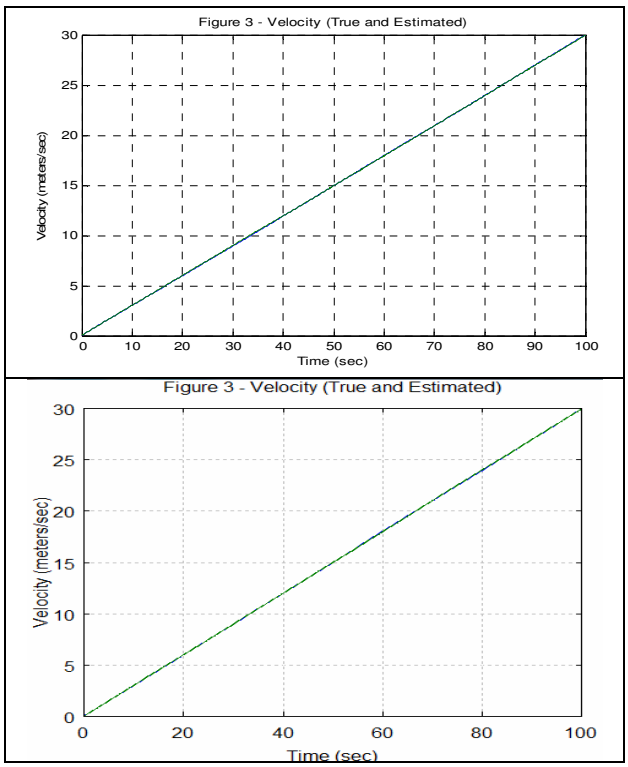

Figure 4c: Matlab (top) and Octave (bottom) Velocity (True and Estimated)

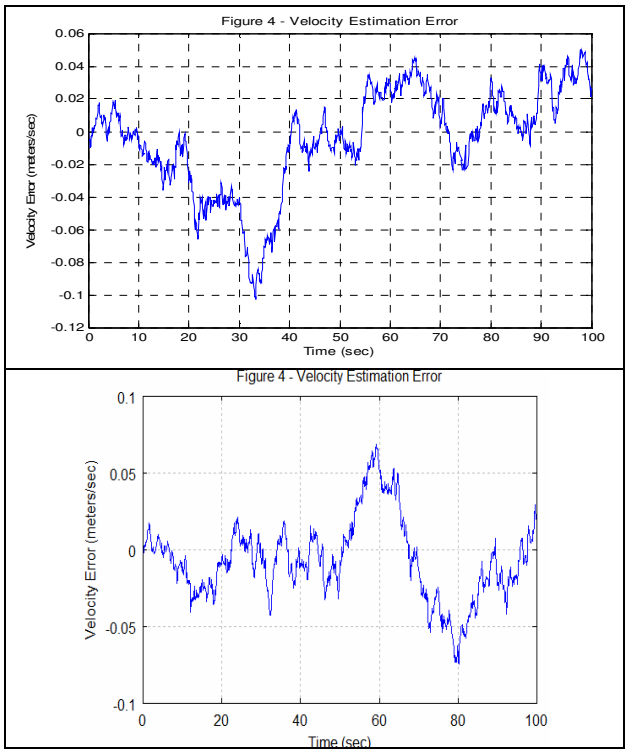

Figure 4d: Matlab (top) and Octave (bottom) Velocity Estimation Error

2.3 Hanowa test matrix Inversion

The hanowa matrix [15] is an n-by-n block 2-by-2 matrix of the form:

$$A = \begin{bmatrix} d*eye(m) & -diag(1:m) \\ diag(1:m) & d*eye(m) \end{bmatrix} \qquad \textbf{(12)}$$

where the dimension n is an even integer n=2*m. Its eigenvalues lie on the vertical line in the complex plane and have the form $d \pm jk$, for $1 \le k \le m$. The default value of $d = -1$.

The complete Octave – Matlab code with the command inv(A) to find the inverse of the above hanowa matrix A (eq. 12) with $d = -2$ and from $m = 1$ to $m = 300$, each time three repetitions, keeping tally of the execution times for every repetition, is presented in Appendix C.

The following plots present the computation times of the matrix inversion for dimensions up to 300. We must note here that we have avoided using Matlab command *gallery* because it does not exit in Octave. Instead, we have created hanowa matrices using simpler but common commands.

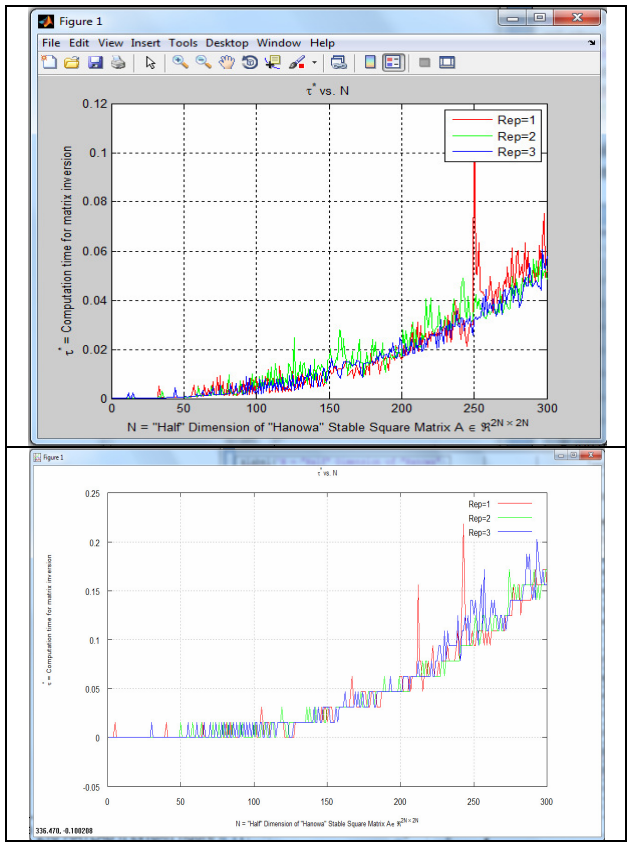

Figure 5: Matlab (top) and Octave (bottom) Computation Times for Hanowa Matrix inversion with dimensions up to 300.

From the above Figure 5 it can be seen that the Matlab execution times for the command inverse of the computationally demanding hanowa test matrix is almost two times faster than the corresponding Octave execution times.

2.4 Octave – Matlab benchmarking results

A summary of the different Octave – Matlab commands used in the codes of the previously

presented three complete benchmark applications tests is listed in Table 1 below.

Table 1: Commands used in each test

Test1 (PID MSD problem)	Test2 (Kalman filter)	Test3 (Hanowa matrix)
basic arithmetic operations	basic arithmetic operations	basic arithmetic operations
tic	function	tic
conv	vector and array arithmetic operations	nested for loops
tf	tic	array arithmetic operations
step	randn	inv array
length	inv array	plot
figure	figure	grid
plot	plot	title
grid	grid	xlabel
xlabel	xlabel	ylabel
ylabel	ylabel	legend
title	title	toc
legend	toc	
toc		

The codes of the three benchmark applications tests were executed on a Windows 7 PC with the following configuration: CPU Intel Core Duo CPU E8500, 32-bit processor running at 3.16GHz, 2GB RAM DDR2, Matlab 7.9.0 (R2009b), Octave3.2.2_gcc-4.3.0 (terminal version for Windows without QtOctave). No other applications were running at test time.

Table 2 below summarizes the results of the execution times for the complete Octave – Matlab codes of the three benchmark applications tests. Time is in seconds.

Table 2: Benchmark results in sec

S/W package	Test1 (PID MSD)	Test2 (Kalman filter)	Test3 (Hanowa matrix)
Matlab	0.042312	0.344092	19.0235
Octave	0.156	0.936001	12.464

In graphical format relative performance times are displayed in Figure 6.

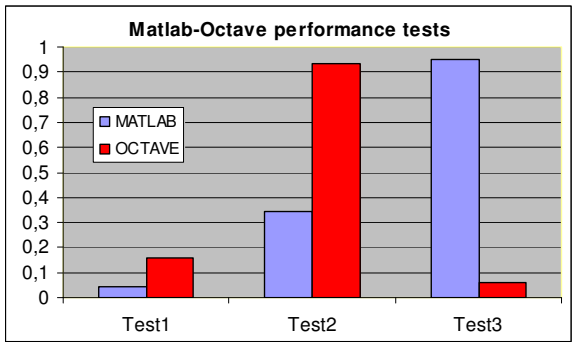

Figure 6: Matlab (left) and Octave (right) Relative normalized computation times for the three tests.

It is noticed that for the first two benchmark applications tests (see also Figure 6) the Matlab execution times are much faster than the corresponding Octave execution times. However, this is not true for the hanowa matrix inverse benchmark test. This is due to the gnuplot graphics package for the Octave which is much simpler than the corresponding Matlab GUI graphics. The Matlab GUI graphics demand more computational resources since they are more user-friendly and also provide many different processing capabilities not present in the Octave gnuplot graphics.

In general, our results are in accordance with the results reported [6, 7], where, most of the times Matlab performs faster than Octave. The code of the three applications presented here is freely available in electronic format from t-h. wikispaces.com/octave.

3 More on the Octave – Matlab comparison

Of course performance is not the only issue regarding software, especially when used in education. Other important issues also exist, such as usability, additional libraries, technical support and price. We shall only touch these issues here.

3.1 Toolbox comparison

Matlab toolboxes and Octave packages are libraries of functions specialized in specific scientific or engineering areas such as Control Theory, Image processing, signal processing, genetic algorithms, bioinformatics, networking etc [16]. Currently Matlab counts 49 toolboxes (Simulink not included) whereas Octave counts 80 packages; however, many of the latter are minimal. The important thing is that both Matlab

and Octave cover an impressive range of scientific and engineering applications.

3.2 GUI comparison

Matlab uses its own GUI which in general is user-friendly; Octave operates from a plain terminal window, but it can be enhanced by a GUI comparable to that of Matlab, namely QtOctave [16]. There is also the possibility to use the Eclipse GUI (Octclipse, see [17]).

3.3 Graphics comparison

Matlab provides better graphics than Octave as one can see from Fig. 5. It also provides a rich set of tools for image manipulation.

3.4 Development philosophy

Matlab is produced by a successful commercial company and is therefore properly supported by their web site, online documentation, international representatives and help desks. Since it has been around for many years, it has been widely accepted by academia, industry and research centers. It has therefore a wide established user community.

On the other hand, Octave is developed by a principal developer and supported (mainly in packages) by a small international community. Other people can also contribute and have developed packages. Technical support is available through forums and web sites of Octave users. Only a few books about Octave have been published; fortunately, the high compatibility to the Matlab language allows Octave users to use Matlab books which are numerous.

Another issue is the increasing penetration of FOSS products in large companies [18], as well as universities [3, 19]; this is due not only to cost reasons but also to the maturity of FOSS projects.

3.5 Availability and cost

The main but unbeatable advantage of Octave is its free availability. Although basic Matlab is sold at a reasonable cost, it does not include any toolboxes; if we add the full toolbox, cost we come up with a rather excessive figure.

4 Conclusion

In this paper we have presented a benchmarking comparison between Matlab and Octave. We have developed our own code especially for this purpose, for three compete science and engineering applications: a) a mass-spring dumper system controlled by a PID controller; b) a Kalman filter estimating the position and velocity of a vehicle in a noisy environment; c) matrix manipulation and calculation time plotting of hanowa test matrices. The tests showed that Matlab in general is faster than Octave, which is in accordance to published results; however, in the hanowa inversion benchmark, Octave performs faster; we attribute this result to the light plotting routine of Octave (gnuplot).

In general Matlab is faster, has more capabilities in graphics, is more user-friendly and more "polished".

We attribute this to the wider user-based since Matlab has been available for a longer time than Octave, is developed and supported by a large professional programming team and is widely used and supported by academia, industry and research centers. On the other hand Octave is maintained by one person supported by a small developers community contributing mainly in package functionality; GNU license gives Octave unpredictable development from worldwide developers with a potential to close the gap.

The three applications were carefully selected to cover three different but representative areas; we have used commands common in Matlab and Octave. This showed the high Matlab-Octave code compatibility on the one hand, and the sufficiency of Octave in engineering and scientific applications on the other.

Our feeling is that Octave can easily substitute Matlab in many science and engineering applications, especially in academia, as our examples have demonstrated. The complete code of the three applications is freely available to download from t-h.wikispaces.com.

References:

[1] *MATLAB - The Language of Technical Computing*. Accessed June 2, 2010, from http://www.mathworks.com/products/matlab

[2] *Octave official page*. Accessed June 2, 2010, from http://www.octave.org.

[3] A. Andreatos and A. Leros, Use of GNU Octave in the Simulation course of the Hellenic Air Force Academy, in *Proc. of the 1st International Conference on the use of FOSS in Education*. Chania, Crete, Apr. 16-18, 2010 (in Greek). Available online from: http://www.foss4edu.gr/praktika/fpapers/paper05-teliko.pdf.

[4] B. Pouli and G. Terzi, *Assessment of Open Source solutions such as SCILAB, GNU Octave, SciPy as Matlab alternatives for teaching automatic control applications*. Diploma Thesis, Technological Institute of Serres, School of Technologic Applications, Div. of Information Science and Tele-communications, Serres, Greece, 2007.

[5] T. T. Zea, *Technical Report on Literature Review on a Matlab Alternative – Octave*. Advanced Computing Group, 27 Feb 2008.
[6] Greek Linux Format, Comparative test: Number crunchers, *Greek Linux Format magazine*, no. 23, Sept-Oct. 2008, pp 42-47.
[7] www.sciviews.org/benchmark/
[8] http://www.engin.umich.edu/group/ctm/PID/PID.html
[9] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, 2nd Edition, Instrument Society of America, 1995.
[10] C.R. Knospe, PID Control: Tuning and Anti-Windup Techniques, *Practical Control Techniques for Control Engineering* workshop held at the 2000 American Control Conference.
[11] R. A. Paz, *The Design of the PID Controller*, Klipsch School of Electrical and Computer Engineering, June 12, (2001). Available from: http://ece.ut.ac.ir/classpages/S84/Mechatronics/PID%20Controller/pid.pdf.
[12] A. Gelb, *Applied Optimal Estimation*, Cambridge, MA: MIT Press, 1974.
[13] B. Anderson and J. Moore. *Optimal Filtering*, Englewood Cliffs, NJ: Prentice-Hall, 1979.
[14] M. Grewal and A. Andrews. *Kalman Filtering Theory and Practice*, Englewood Cliffs, NJ: Prentice-Hall, 1993.
[15] http://www.mathworks.com/access/helpdesk/help/techdoc/ref/gallery.html. Accessed June 21, 2010.
[16] A. Andreatos and A. Leros, Simulation course redesign and educational software selection, in *Proc. of the 9th European Conference on e-Learning* (ECEL 2010), 4-5 November 2010, Porto, Portugal.
[17] http://sourceforge.net/projects/octclipse/ Accessed June 21, 2010.
[18] D. Spinellis and V. Giannikas, Open Source adoption in large US companies. In *Proc. of the 4th Mediterranean Conference on Information Systems* (MCIS 2009). Athens, Greece, Sept. 25-27, 2009.
[19] H. Coll, D. Bri, M. Garcia and J. Lloret, Free software and open source applications in higher education. In *Proceedings of the 5th WSEAS / IASME International Conference on Engineering Education,* Heraklion, Greece, July 22-24, 2008.

Appendix

In this section we provide the code of our three applications; the commands used are both Octave and Matlab compatible.

Appendix A

The Octave-Matlab code to simulate the modified PID control system configuration (eq. (6) and eq. (3)) is as follows:

```
tic; % Measures performance using
stopwatch time
% System parameters M = 1 kg, b = 10
N.s/m, k = 20 N/m, F(s) = 1
M = 1; b = 10; k = 20;
% System in Transfer function form:
num = 1; den = [M b k];
% Modified PID controller
Kp = 200; ti = 0.5; t1 = 1/100; N=15; Ki
= Kp/ti; Kd = t1*N*Kp;
% Numerator controller
numc = [Kp*t1+Kd Kp+Ki*t1 Ki]/t1;
% Denominator controller
denc = [t1 1 0]/t1;
% Numerator Feedback system
num_feedback = conv(num, numc);
% Denominator Feedback system
den_feedback = conv(den, denc);
% Feedback system
den_closed_loop = den_feedback+[0 0
num_feedback];
% Output of Feedback system
num_y = num_feedback;
% Output of Open-loop system (scaled by
20 for it to be settled to 1)
sys_open_loop = tf(20*num, den);
% Output of Closed-loop system
sys_y = tf(num_y, den_closed_loop);
t = (0:.002:5)';
y_open_loop = step(sys_open_loop, t);
% y_open_loop = step(sys_open_loop, 1,
5, length(t));
y_closed_loop = step(sys_y, t);
% y_closed_loop = step(sys_y, 1, 5,
length(t));
figure(1);
plot(t, y_open_loop, t, y_closed_loop);
grid on;
xlabel('time (sec)'); ylabel('y(t)');
title('Matlab plots');
% title('Octave plots');
legend('Matlab Open-loop response',
'Matlab Closed-loop response');
% legend('Octave Open-loop response',
'Octave Closed-loop response');
toc; % Measures performance using
stopwatch time
```

Note 2: In the above code the four lines which are in bold, italic and underlined differ among the Octave and the Matlab cases. The rest of the lines are the same.

Appendix B

The algorithmic function coded in Octave – Matlab of the Kalman filter of the system for the vehicle movement using noisy measurements of its position is as follows:

```
tic; % Measures performance using
stopwatch time
function kalman_filter(duration, dt,
sigma_w, sigma_v, u)
F = [1 dt; 0 1]; % transition matrix
B = [dt^2/2; dt]; % input matrix
H = [1 0]; % measurement matrix
x = [0; 0]; % initial state vector
x_est = x; % initial state estimate
Q=sigma_w^2*[dt^4/4 dt^3/2; dt^3/2
dt^2]; % process noise covariance
R = sigma_v^2; % measurement error
covariance
P = Q; % initial estimation covariance
% Initialization of arrays for plotting
x1 = []; % true position array
x1_est=[]; % estimated position array
x2=[]; % true velocity array
x2_est=[]; % estimated velocity array
% measured position array
z_measured = [];
% Simulation of the discrete linear
dynamical system x=Fx+Bu+w
for t = 0 : dt: duration
% acceleration noise w
w=sigma_w*[(dt^2/2)*randn; dt*randn];
x = F * x + B * u + w;
% Simulation noisy measurements z=Hx+v
v=sigma_v*randn; % measurement noise v
z = H * x + v; % noisy measurements
% Propagate the state estimate
x_est = F * x_est + B * u;
% Form the residual vector
residual = z - H * x_est;
% Form the Kalman Gain
K = P * H' * inv(H * P * H' + R);
% Update the state estimate
x_est = x_est + K * residual;
% Update estimation error covariance
P = F*P*F'-F*K*H*P*F'+Q;
% Parameters for plotting
x1 = [x1; x(1)];  % true position
% estimated position
x1_est = [x1_est; x_est(1)];
% measured position
z_measured = [z_measured; z];
x2 = [x2; x(2)]; % true velocity
% estimated velocity
x2_est = [x2_est; x_est(2)];
end
% Plotting of the results
close all; t = 0:dt:duration;
figure (1);
plot(t,x1,t,z_measured,t,x1_est);
grid; xlabel('Time (sec)');
ylabel('Position (meters)');
title('Figure 1 - Position of Vehicle
(True, Measured, and Estimated)')
figure (2);
plot(t,x1- z_measured,t,x1-x1_est);
grid; xlabel('Time (sec)');
ylabel('Position Error (meters)');
title('Figure 2 - Position Measurement
Error and Position Estimation Error');
figure(3); plot(t,x2,t,x2_est); grid;
xlabel('Time (sec)');
ylabel('Velocity (meters/sec)');
title('Figure 3 - Velocity (True and
Estimated)');
figure(4); plot(t,x2-x2_est); grid;
xlabel('Time (sec)');
ylabel('Velocity Error (meters/sec)');
title('Figure 4 - Velocity Estimation
Error');
toc; % Measures performance using
stopwatch time
```

Appendix C

The complete Octave – Matlab code with the command inv(A) to find the inverse of the hanowa matrix A (eq. 12) with $d = -2$ and from $m = 1$ to $m = 300$, each time three repetitions, keeping tally of the execution times for every repetition, is as follows:

```
clear all; close all; clc
tStart = tic; % Measures performance
using stopwatch time
Npoints = 300; d = -2; RepeaterMax = 3;
for Repeater = 1:1:RepeaterMax
  for  m = 1:1:Npoints
   A=[d*eye(m) -diag(1:m);
      diag(1:m) d*eye(m)];
  tic; %--- tic = START COUNTING ---
  inv(A);
  toc; %--- toc = STOP   COUNTING ---
  save_for_plot(m,1) = m;
  save_for_plot(m,2) = toc;
  end
  if Repeater == 1
plot(save_for_plot(:,1),save_for_plot(:,
2),'r')
  elseif Repeater == 2
plot(save_for_plot(:,1),save_for_plot(:,
2),'g')
  else
plot(save_for_plot(:,1),save_for_plot(:,
2),'b')
  end
  hold on
end
grid; hold on
title('\tau^{*} vs. N')
xlabel('N = "Half" Dimension of "Hanowa"
Stable Square Matrix A \in \Re^{2N
\times 2N}')
ylabel('\tau^{*} = Computation time for
matrix inversion')
legend('Rep=1','Rep=2','Rep=3')
tElapsed = toc(tStart) % Measures
performance using stopwatch time
```