Computer Engineering & Informatics Department
University of Patras

Scientific Calculation

Academic Year 2015-2016
Laboratory Exercise
E-mail: marinou@ceid.upatras.gr

I n d e x

Question 1

(ι) System Characterization

Processor: *2*nd generation Intel® Core™ i3-2348M Processor

OS: Windows 7 (6.1) Professional 64-bit

Hard drive: *500* GB (5,400 rpm) σειριακό ATA

System memory: 4,096 (1x) MB, μέγιστη επεκτασιμότητα: 8,192 MB, τεχνολογία: DDR3 RAM (1.333 MHz)

3 levels of cache:

Level 1 cache size:

- 2 x 32 Kbytes , 8way set associative, 64- byte line size (L1 D-cache)
- 2 x 32 Kbytes , 8way set associative, 64- byte line size (L1 I- cache)

Level 2 cache size

- 2 x 256 Kbytes, 8way set associative, 64- byte line size

Level 3 cache size

- 3 Mbytes, 12 way set associative, 64- byte line size

(II) OS & MATLAB:

•Windows 7 (6.1) Professional 64-bit

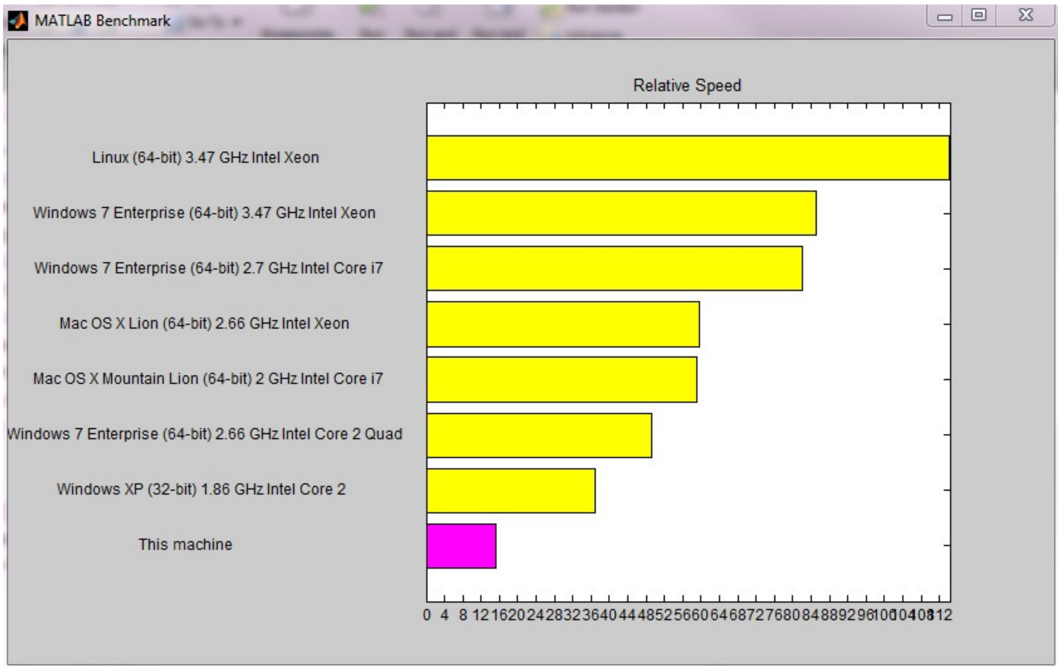
•*MATLAB_R2013a*

(III) Metroprogram and command result bench

```
   Matlab Benchmark
   ================
Number of times each test is run__________________________: 3

   I. Matrix calculation
   ---------------------
Creation, transp., deformation of a 1200x1200 matrix (sec): 0.084263
1250x1250 normal distributed random matrix ^1000____ (sec): 0.15168
Sorting of 1,100,000 random values__________________ (sec): 0.098116
550x550 cross-product matrix (b = a' * a)___________ (sec): 0.015296
Linear regression over a 700x700 matrix (c = a \ b') (sec): 0.032516
                      --------------------------------------------
                      Trimmed mean (2 extremes eliminated): 0.071632

   II. Matrix functions
   --------------------
FFT over 900,000 random values______________________ (sec): 0.031067
Eigenvalues of a 220x220 random matrix______________ (sec): 0.090773
Determinant of a 750x750 random matrix______________ (sec): 0.061002
Cholesky decomposition of a 1000x1000 matrix________ (sec): 0.087038
Inverse of a 500x500 random matrix__________________ (sec): 0.13686
                      --------------------------------------------
                      Trimmed mean (2 extremes eliminated): 0.079604

   III. Programmation
   ------------------
225,000 Fibonacci numbers calculation (vector calc)_ (sec): 0.18627
Creation of a 1500x1500 Hilbert matrix (matrix calc) (sec): 0.15504
Grand common divisors of 35,000 pairs (recursion)___ (sec): 0.10264
Creation of a 220x220 Toeplitz matrix (loops)_______ (sec): 0.0019962
Escoufier's method on a 22x22 matrix (mixed)________ (sec): 0.54551
                      --------------------------------------------
                      Trimmed mean (2 extremes eliminated): 0.14798


Total time for all 15 tests_________________________ (sec): 1.7801
Overall mean (sum of I, II and III trimmed means/3)_ (sec): 0.099739
                      --- End of test ---
>>
```

Using the official matlab platform bench, we have the following results:

```
>> bench

ans =

    0.6160    0.3289    0.6830    1.2787    1.8909    0.9542

>>
```



MATLAB Benchmark (times in seconds)

Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Linux (64-bit) 3.47 GHz Intel Xeon	0.0626	0.0661	0.1617	0.1161	0.2055	0.0911
Windows 7 Enterprise (64-bit) 3.47 GHz Intel Xeon	0.0701	0.0719	0.1094	0.1297	0.2784	0.7044
Windows 7 Enterprise (64-bit) 2.7 GHz Intel Core i7	0.0857	0.0801	0.0958	0.1319	0.2975	0.7003
Mac OS X Lion (64-bit) 2.66 GHz Intel Xeon	0.0547	0.1278	0.2008	0.1877	0.6670	0.6299
Mac OS X Mountain Lion (64-bit) 2 GHz Intel Core i7	0.0725	0.1292	0.1881	0.1587	0.7203	0.6476
Windows 7 Enterprise (64-bit) 2.66 GHz Intel Core 2 Quad	0.1239	0.2333	0.1561	0.2822	0.4819	0.7390
Windows XP (32-bit) 1.86 GHz Intel Core 2	0.3406	0.3178	0.1883	0.3542	0.5775	0.3601
This machine	0.6160	0.3289	0.6830	1.2787	1.8909	0.9542

Place the cursor near a computer name for system and version details. Before using this data to compare different versions of MATLAB, or to download an updated timing data file, see the help for the bench function by typing "help bench" at the MATLAB prompt.

Question -Timing of Functions

(I) Basic Operations & Functions of MATLAB

• **[L,U]=lu(A) :** Table L returns a lower triangular array and o U an upper upper triangle so LU = PA, where P is a matrix matrix.

• **[L,U,P]=lu(A) :** Table P returns the matrix P

• **qr:** It represents a register as the product of a real or complex ortho-regular register (Q) and an upper triangular (R)

• **svd:** Analysis of M at particular prices. Generating a table in the form M = USV * where U is a ortho-mononial matrix. A rectangular array with non-negative values in the diagonal (its diagonal elements are M's) and V * are real ortho-mononial matrix.

• **det:** Calculates the table definition

• **rank:** Calculates an estimate of the number of linearly independent lines or columns of a complete registry

• **polyval (p,x)** The x statement can be either an element, or a vector, or a register. The result y is the same size as x and contains the polynomial values at x. The method you use to calculate the value of the polynomial is the Horner method

Intrinsic functions are the functions we do not have access to in their code, while External, the ones we have access to. In addition, svd, qr, lu and det are built in functions, while rank and polyval are exogenous.

We run the following code:

```
clear all;
clc;
%Erwthma 2-(ii)- (a)

k=1;
   for i=7:12
        n=2^i;
        A=rand(n);
        b=rand(n,1);

       %xronometrisi gia thn lu
       tic;
       [L, U, P]=lu(A);
       time_lu(k,1)=toc;
       gflopi(k,1)=2*n^2/time_lu(k,1)/10^9;

       %xronometrisi gia thn A*b
```

```
      tic;
      c=A*b;
      time_c(k,1)=toc;
      gflopii(k,1)=2*n^2/time_c(k,1)/10^9;

     k=k+1;
  end

subplot(1,2,1)
i=2.^[7:12];
plot(i,time_lu(:,1), 'g.-', i, time_c(:,1), 'b.--');
legend('lu(A)', 'c');
xlabel('n');
ylabel('t sec');
title('Χρηση tic-toc');

subplot(1,2,2)
i=2.^[7:12];
plot(i,gflopi(:,1), 'g.-', i, gflopii(:,1), 'b.--');
legend('lu(A)', 'c');
xlabel('n');
ylabel('gflop');
title('Επίδοση  gflop');
```

Execution times

time_lu <6x1 double>

	1	2
1	0.0057	
2	0.0062	
3	0.0070	
4	0.0416	
5	0.2786	
6	2.1254	

gflopi <6x1 double>

	1	2
1	0.0057	
2	0.0212	
3	0.0749	
4	0.0504	
5	0.0301	
6	0.0158	

time_c <6x1 double>

	1	2
1	4.8192e-05	
2	1.1825e-04	
3	4.6095e-04	
4	0.0016	
5	0.0054	
6	0.0151	

gflopii <6x1 double>

	1	2
1	0.6799	
2	1.1084	
3	1.1374	
4	1.2954	
5	1.5393	
6	2.2239	

(ιι)- (β) Execution times for **tic, toc** *στα [L, U, P]=lu(A) και c=A*b for 20 times and gflop performance. We run the following code:*

```
clear all;
clc;

%Erwthma 2-(ii)- (b)

k=1;
   for i=7:12
        n=2^i;
        A=rand(n);
        b=rand(n,1);

       %xronometrisi gia thn lu

       tic;
       for l=1:20
           [L, U, P]=lu(A);
       end
       time_lu(k,1)=toc/20;
       gflopi(k,1)=2*n^2/time_lu(k,1)/10^9;

       %xronometrisi gia thn A*b
       tic;
       for l=1:20
           c=A*b;
       end
       time_c(k,1)=toc/20;
       gflopii(k,1)=2*n^2/time_c(k,1)/10^9;

      k=k+1;
   end

 subplot(1,2,1)
 i=2.^[7:12];
 plot(i,time_lu(:,1), 'g.-', i, time_c(:,1), 'b.--');
 legend('lu(A)', 'c');
 xlabel('n');
 ylabel('t sec');
 title('Χρηση tic-toc');

 subplot(1,2,2)
 i=2.^[7:12];
 plot(i,gflopi(:,1), 'g.-', i,gflopii(:,1), 'b.--');
 legend('lu(A)', 'c');
 xlabel('n');
 ylabel(' gflop');
 title('Επίδοση  gflop');
```

Execution times

time_lu <6x1 double>

	1	2
1	4.8487e-04	
2	0.0016	
3	0.0104	
4	0.0507	
5	0.3476	
6	2.7958	

gflopi <6x1 double>

	1	2
1	0.0676	
2	0.0810	
3	0.0504	
4	0.0414	
5	0.0241	
6	0.0120	

time_c <6x1 double>

	1	2
1	1.1200e-05	
2	1.9790e-05	
3	1.0123e-04	
4	9.8827e-04	
5	0.0040	
6	0.0151	

gflopii <6x1 double>

	1	2
1	2.9257	
2	6.6232	
3	5.1794	
4	2.1220	
5	2.0902	
6	2.2225	

(ιι) Using the timeit function in [L, U, P] = lu (A) and c = A * b and performance gflop:

We are running the following code

```
clear all
clc
%Erwthma 2-(g)

k=1;
   for i=7:12
       n=2^i;
       A=rand(n);
       b=rand(n,1);

       f = @()lu(A);
       time_lu(k,1)=timeit(f,3);
       gflopi(k,1)=2*n^2/time_lu(k,1)/10^9;

      f = @()A*b;
      time_c(k,1)=timeit(f,1);
      gflopii(k,1)=2*n^2/time_c(k,1)/10^9;

    k=k+1;

   end

 subplot(1,2,1)
 i=2.^[7:12];
 plot(i,time_lu(:,1), 'g.-', i, time_c(:,1), 'b.--');
 legend('lu(A)', 'c');
 xlabel('n');
 ylabel('t sec');
```

```
title('Χρηση timeit')

subplot(1,2,2)
i=2.^[7:12];
plot(i, gflopi(:,1), 'g.-', i, gflopii(:,1), 'b.--');
legend('lu(A)', 'c');
xlabel('n');
ylabel(' gflop');
title('Επίδοση  gflop');
```

Execution times

time_lu <6x1 double>

	1	2
1	2.9004e-04	
2	0.0017	
3	0.0080	
4	0.0444	
5	0.3302	
6	2.8016	

gflopi <6x1 double>

	1	2
1	0.1130	
2	0.0763	
3	0.0659	
4	0.0472	
5	0.0254	
6	0.0120	

time_c <6x1 double>

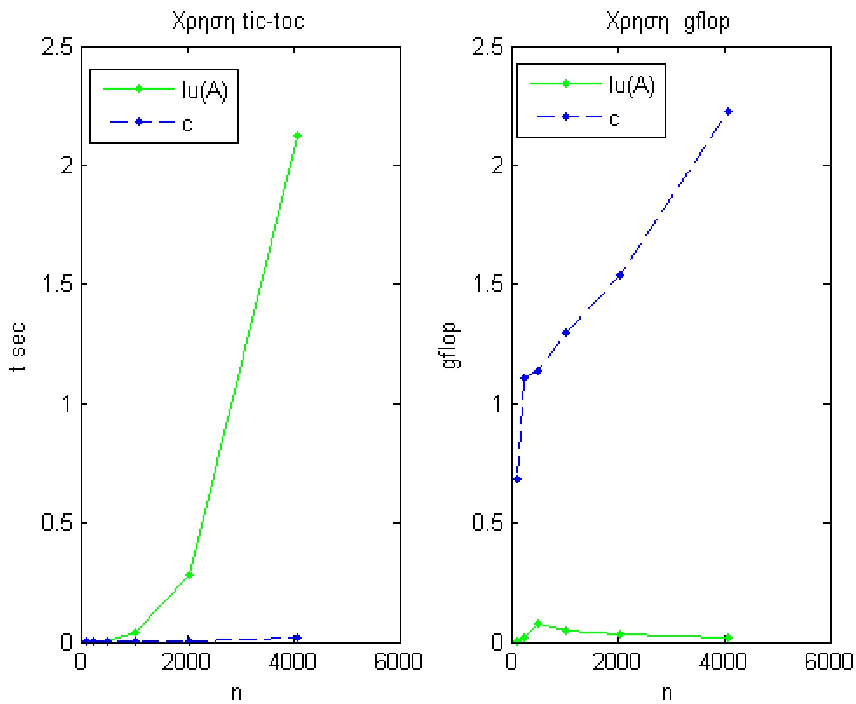
	1	2
1	1.0833e-05	
2	2.2852e-05	
3	7.6324e-05	
4	9.0531e-04	
5	0.0035	
6	0.0172	

gflopii <6x1 double>

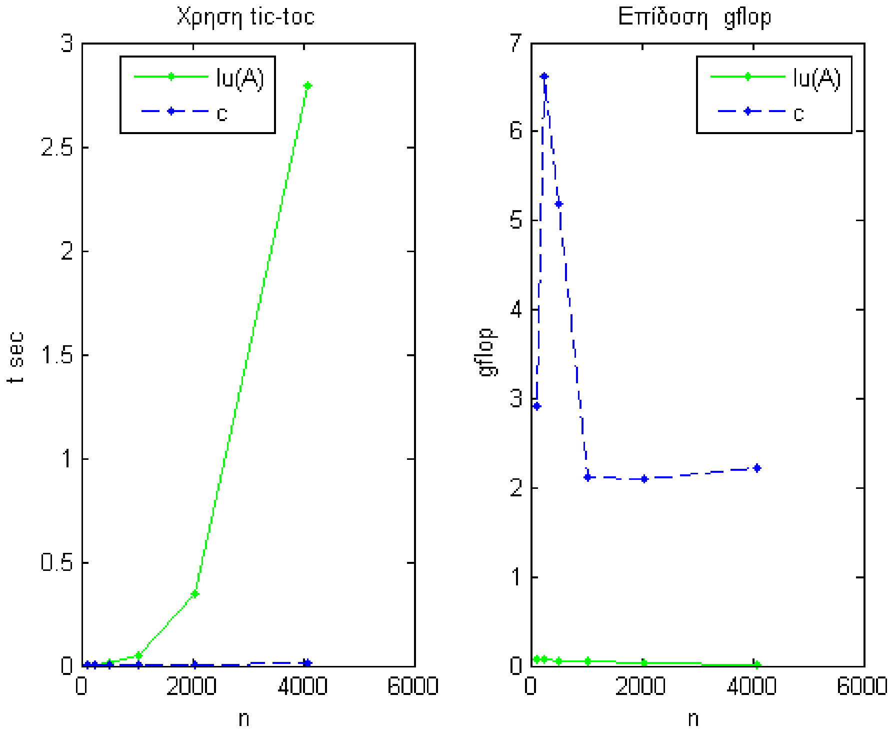
	1	2
1	3.0247	
2	5.7358	
3	6.8693	
4	2.3165	
5	2.3809	
6	1.9455	

(iii) Visualization for each of the above numerical results:

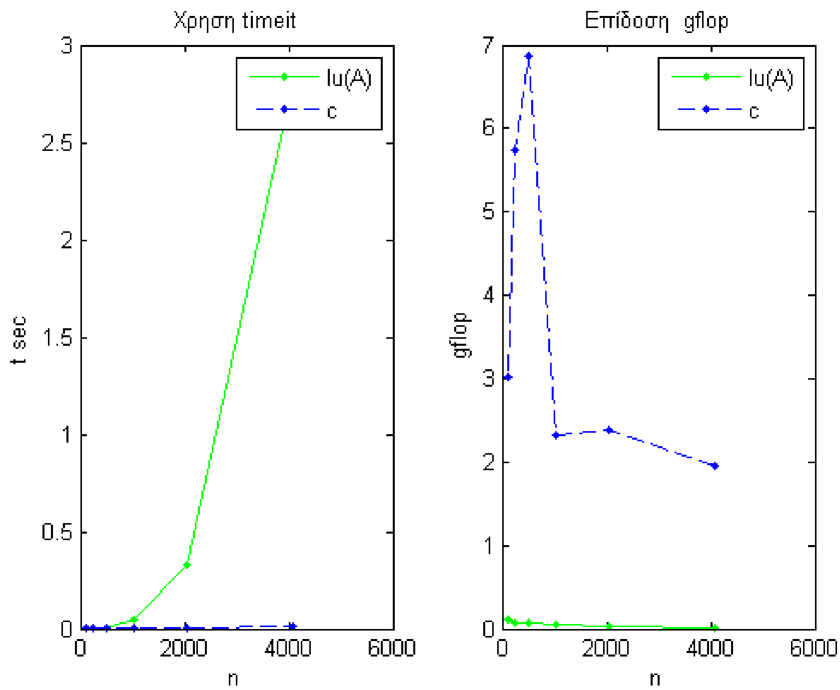
(a) Use tic, toc functions to perform each operation once and gflop performance



(b) Using tic, toc averaged functions for greater reliability and performance gflop



(c) Use timeit function and gflop performance



Commenting on Results

*(1) Act lu requires considerably more operations to find the L and U registers from A * b.*

*(2) The lu act exponentially expands as the number of register elements increases, as opposed to A * b which increases almost linearly at a low rate of change - regardless of the value of n.*

(3) The timeit function takes so many measurements to be considered more reliable. In the above examples, the times from the two previous methods (tic, toc and averages) and timeit have a slight deviation.

Question 3- Evaluating endogenous functions and m-functions

(a) Using the timeit function, we make timing of mtimes for the above types of registers, size n = 2 ^ [7:12]

(i) Random registers
(ii) Registers having random data and being triangular
(iii) Registers having random data and being upper triangular
(iv) Registers with random data and is over Hessenberg

We run the following code:

```
clear all;
clc;

%Erwthma 3-(a)- (i)

 k=1;
   for i=7:12
       n=2^i;


       %(i)random matrices
       Ai=rand(n);
       Bi=rand(n);
       f = @() mtimes(Ai, Bi);
       ti(k,1)=timeit(f,1);
       gflopi(k,1)=2*n^2/ti(k,1)/10^9;

       %(ii) tridiagwnio matrices

       upo_diag=rand(n-1,1);
       kuria_diag=rand(n,1);
       uper_diag=rand(n-1,1);
       Aii=[diag(upo_diag,-1)+diag(kuria_diag)+diag(uper_diag,1)];

       upo_diag=rand(n-1,1);
       kuria_diag=rand(n,1);
       uper_diag=rand(n-1,1);
```

```
    Bii=[diag(upo_diag,-1)+diag(kuria_diag)+diag(uper_diag,1)];

    f = @() mtimes(Aii, Bii);
    tii(k,1)=timeit(f,1);
    gflopii(k,1)=2*n^2/tii(k,1)/10^9;


    %(iii) upper triangular matrices

    Aiii=triu(rand(n));
    Biii=triu(rand(n));

    f = @() mtimes(Aiii, Biii);
    tiii(k,1)=timeit(f,1);
    gflopiii(k,1)=2*n^2/tiii(k,1)/10^9;



    %(iv) matrices

    Aiv= hess(rand(n));
    Biv= hess(rand(n));

    f = @() mtimes(Aiv, Biv);
    tiv(k,1)=timeit(f,1);
    gflopiv(k,1)=2*n^2/tiv(k,1)/10^9;
   k=k+1;

end;

n=2.^[7:12];

subplot(1,2,1)
plot(n, ti(:,1), 'y.-', n,tii(:,1),'r.--', n, tiii(:,1),'b.-', n, tiv(:,1), 'g.-');
legend('i', 'ii','iii', 'iv');
xlabel('n');
ylabel('t sec');
title('Question 3 (a)- mtimes');


subplot(1,2,2)
plot(n, gflopi(:,1), 'y.-', n,gflopii(:,1),'r.--', n, gflopiii(:,1),'b.-', n, gflopiv(:,1), 'g.-');
legend('i', 'ii','iii', 'iv');
xlabel('n');
ylabel('Gflop/s');
title('Question 3 (a)- mtimes');
```

<u>Times for each of the registers (ti, tii, tiii, tiv)</u>

ti <6x1 double>

	1
1	3.4351e-04
2	0.0025
3	0.0218
4	0.3898
5	1.1398
6	14.8938

	1
1	3.2512e-04
2	0.0026
3	0.0553
4	0.4039
5	1.0837
6	13.7299

tiii <6x1 double>

	1
1	3.3893e-04
2	0.0026
3	0.0603
4	0.4087
5	1.6275
6	15.8906

tiv ×

tiv <6x1 double>

	1
1	3.4116e-04
2	0.0028
3	0.0529
4	0.3907
5	1.0435
6	14.4361

(b) I implement a myTridMult function to perform the multiplication between triangular registers which only takes into account the non-zero elements while ignoring the operations with zeros. By using timeit again, we make the timing of the function for the above triangular registers:

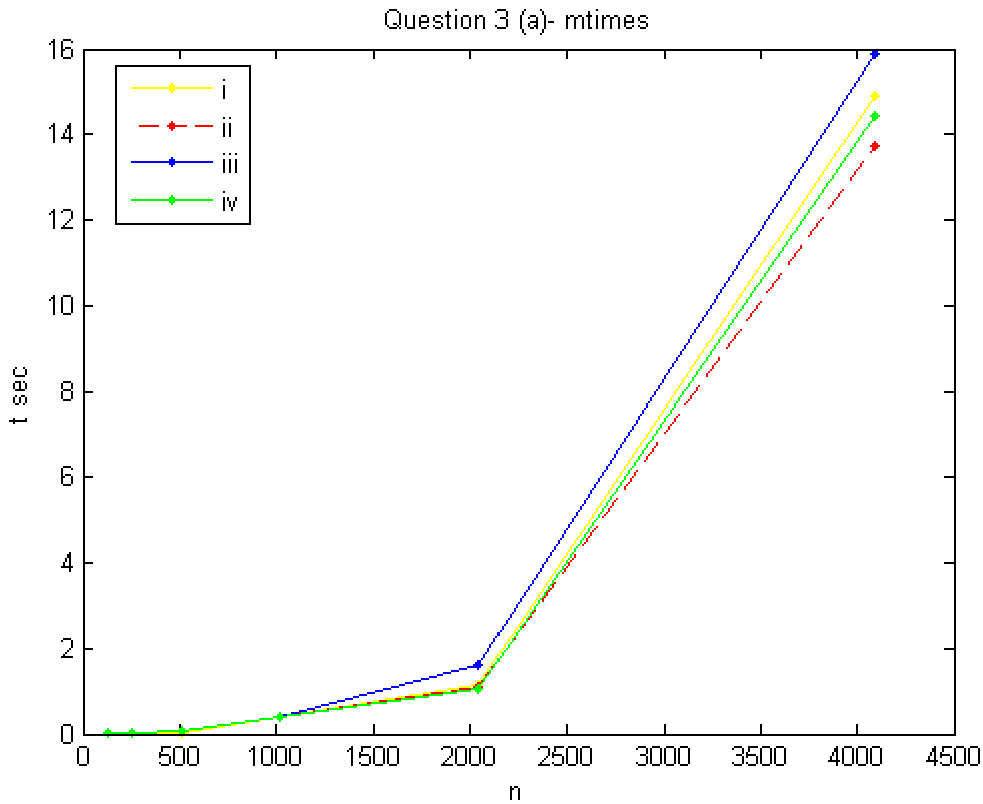
We run the following code:

```
function [c] = myTridMult(A,B,n)


 c=zeros(n,n);
for i=1:n
   for j=1:n
      for k=1:n
         if A(i,k)~=0 || B(k,j)~=0
            c(i,j)=c(i,j)+A(i,k)* B(k,j);
         end
      end
   end
end
```

```
clear all;
clc;

%Erwthma 3-(b)

 k=1;
   for i=7:12
       n=2^i;

       %(ii) tridiagwnio matrices

       upo_diag=rand(n-1,1);
       kuria_diag=rand(n,1);
       uper_diag=rand(n-1,1);
       Aii=[diag(upo_diag,-1)+diag(kuria_diag)+diag(uper_diag,1)];

       upo_diag=rand(n-1,1);
       kuria_diag=rand(n,1);
       uper_diag=rand(n-1,1);
       Bii=[diag(upo_diag,-1)+diag(kuria_diag)+diag(uper_diag,1)];

       f = @() myTridMult(Aii, Bii,n);
       t_myTridMult(k,1)=timeit(f,1);


       f = @() mtimes(Aii, Bii);
       t_mtimes(k,1)=timeit(f,1);
             gflop(k,1)=2*n^2/t(k,1)/10^9;

      k=k+1;
  end;

   n=2.^[7:12];

   plot(n,t_myTridMult(:,1),'r.--',n,t_mtimes(:,1),'y.-');
   legend('t_myTridMult','t_mtimes');
   xlabel('n');
   ylabel('t sec');
   title('Question 3 (b)');
```

The Code while it ran did not end. Possible cause of this is to call the myTridMult function in which many forces are executed and for big n the algorithm drops to a large Loop.

(c) Visualization of results:



(d) Commentary on Results

From the times that emerged and from the graphical representation of these according to the size of the registers, we conclude that most of the time for the execution of the operation requires the triangular registers, then the Hessenberg registers, followed by the random registers and finally the upper triangles. The more peculiar features they have, the longer they require to carry out their actions.

We note more specifically that the Hessenberg registers are close to those of the random ones and the upper triangles with the triangles show large deviations as n grows. The preferred embodiment for the different sizes of the problem is that which makes the shortest time, that is, the fastest, even for a large number of n.

Query 4- Comparison of Embodiments

In this question we will evaluate both the time, pace and actions of different ways of implementing the same computational problem.

(1) Theoretical Calculation of Operations

We have the act x = (u * u '+ v * v') ^ P * b
For the multiplication between u and u 'we will need (2n-1) operations, as well as for the product v * v', so I have 4n-2. In the end I will have to add the data in parentheses +1. So far I have (4n + 1). After I raise to force p I will have (4n + 1) * p and finally + n by multiplication with vector b. I have therefore (4n + 1) * p + n = 4np + p + n = n (4p + 1) + p operations = = 0 (n).

(2) We create the following function rank2_power returning the result of operation x ,. U, v and b are random vectors

```
function x= rank2_power(u,v,b)

            x=(u*u'+v*v')^10*b;
end
```

We then run the following code:

```
clear all;
clc;

%Question 4


k=1;

for i=7:12
   n=2^i;

   u=rand(n,1);
```

```
    v=rand(n,1);
    b=rand(n,1);


   f = @() rank2_power(u,v,b);
   ta(k,1)=timeit(f,1);
   gflopa(k,1)=2*n^2/ta(k,1)/10^9;

   f = @() my_rank2_power(u,v,b);
   tb(k,1)=timeit(f,1);
   gflopb(k,1)=2*n^2/tb(k,1)/10^9;

   k=k+1;
end


i=2.^[7:12];
subplot(2,2,1)
plot(i, ta(:,1), 'y.-');
legend('rank2_power');
xlabel('n');
ylabel('t sec');
title('Σύγκριση χρόνου');


i=2.^[7:12];
subplot(2,2,2)
plot(i, gflopa(:,1), 'g.-');
legend('rank2_power');
xlabel('n');
ylabel('gflopi');
title('Σύγκριση gflopi');

i=2.^[7:12];
subplot(2,2,3)
plot(i, tb(:,1), 'y.-');
legend('my_rank2_power');
xlabel('n');
ylabel('t sec');
title('Σύγκριση χρόνου');

i=2.^[7:12];
subplot(2,2,4)
plot(i, gflopb(:,1), 'g.-');
legend('my_rank2_power');
xlabel('n');
ylabel('gflopii');
title('Σύγκριση gflopii');
```

(3) Transformation into the given expression will be implemented as follows:
We will make paralleling from right to left with b after first breaking the force that is raised in the

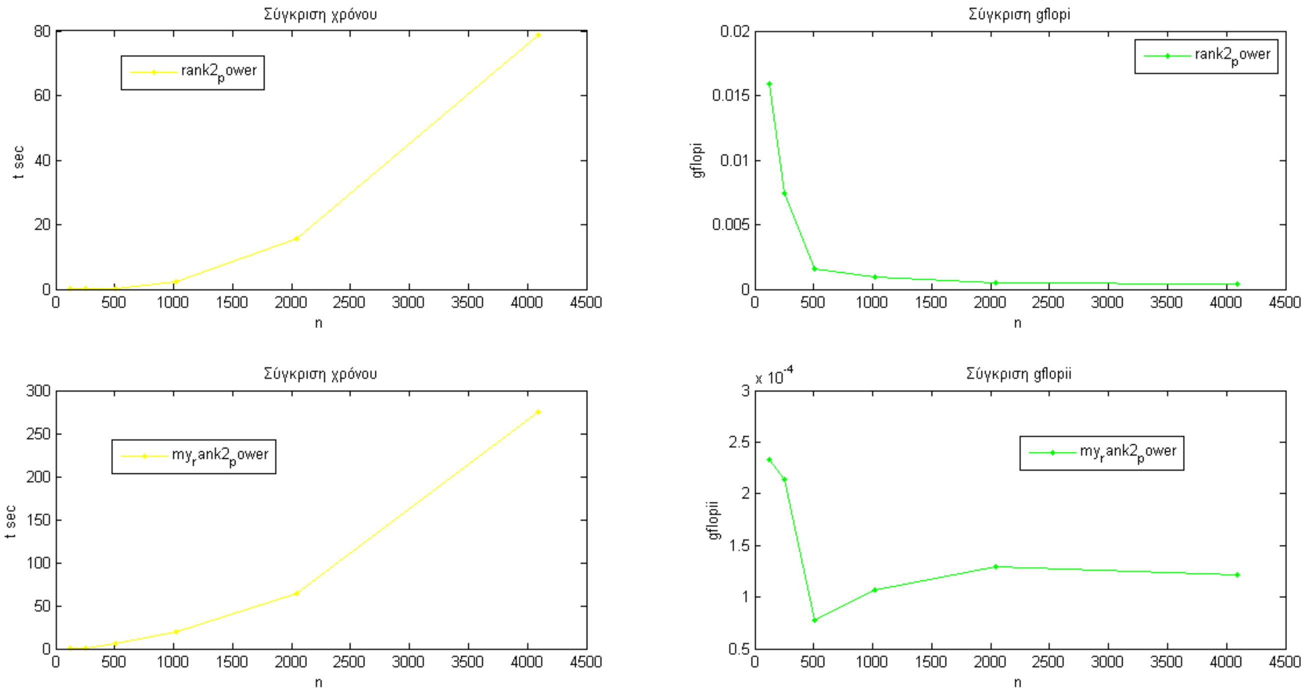
brackets.

```
function x= my_rank2_power(u,v,b)

x=(u*u'+v*v');
x= x*b ;

i=1:10
    x=(u*u'+v*v')*x;
end
```

(4) The performance of the two functions for n = 2 ^ [7:12] in

(a) a common chart of execution times

(b) execution rates (in Gflop / s) in a common graph



Commenting on Results

We notice that the gflop performance over time is countermetric as the table size increases.

Bibliography

[1] CLEVE B. MOLER. NUMERICAL METHODS WITH MATLAB. PUBLICATIONS KLIDARITHMOS LTD. 2010.

[2] GILBERT STRANG. INTRODUCTION TO LINE ALGER. PUBLICATIONS OF PATRAS UNIVERSITY, 2006.

[3] Efstratios Galopoulos. SCIENTIFIC CALCULATION I. University of Patras, Autumn 2013.