This problem is meant to illustrate how to do a number of basic matrix operations in the language of your choice. Also to consider whether the $2n^3$ operation counts for inverse and matmul, and the $(2/3)n^3$ for solve and lu are enough to predict what a computer's speed is. You might wish to mention in your homework, if you know your machine's processor and operating system. I just got a new machine running 32 bit Windows Vista on an Intel quadcore Q6700 processor running at 2.66GHz.

This means that every second, 2.66 billion instructions can be sent to each of four processors . Each processor can do both a multiply and an add in each cycle for a total of 2.66 * 8 or about 21 billion multiplies or adds per second.

Mathematica (and I did try MATLAB) took advantage of the parallelism somewhat, with Mathematica doing a bit of a better job than MATLAB in performance.

The Python numbers were run on a different machine as I couldn't load it onto Vista. If anyone succeeds please tell me.

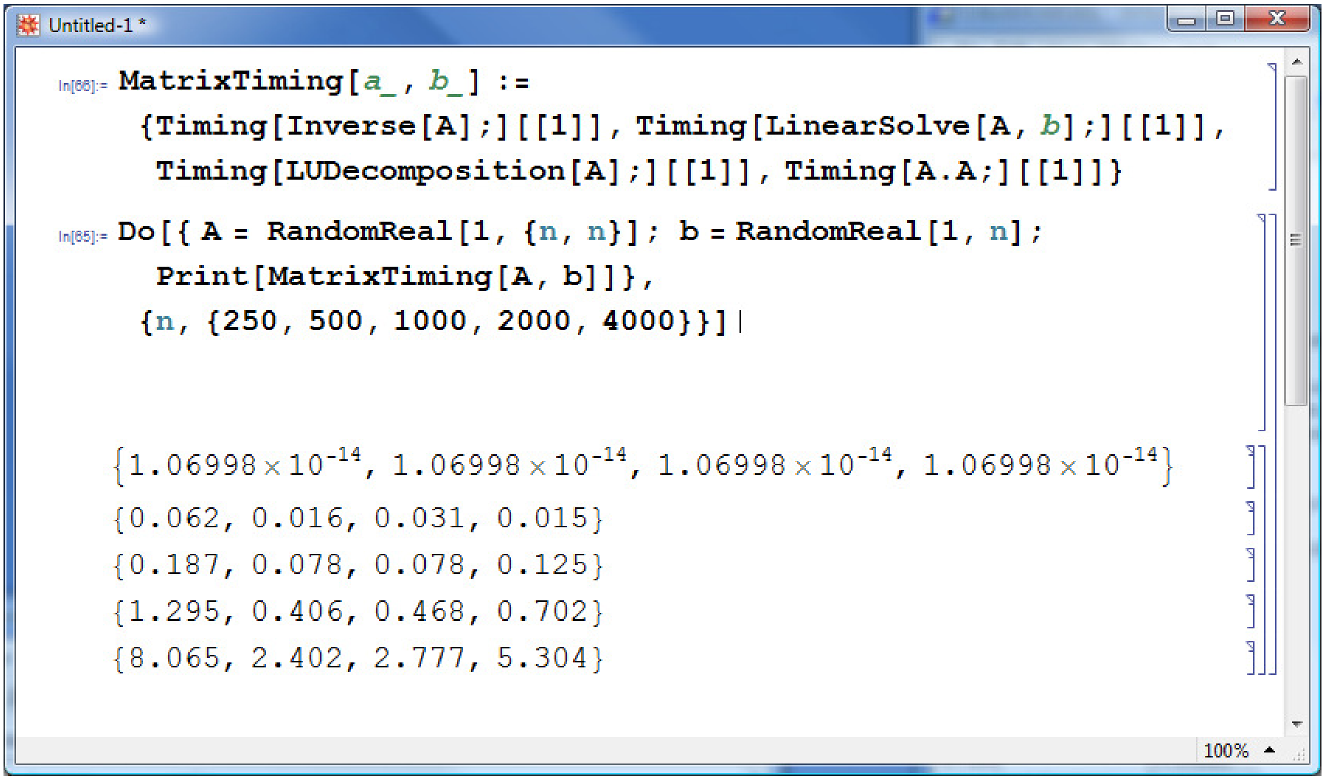
.

```
-->n=250; A=rand(n,n); b=rand(n,1);
 -->tic, inv(A); t(1)=toc(); tic, A\b; t(2)=toc(); tic, [L,U]=lu(A); t(3)=toc(); tic, A*A; t(4)=toc();t'
 ans  =
    0.011    0.006    0.007    0.006
 -->n=500; A=rand(n,n); b=rand(n,1);
 -->tic, inv(A); t(1)=toc(); tic, A\b; t(2)=toc(); tic, [L,U]=lu(A); t(3)=toc(); tic, A*A; t(4)=toc();t'
 ans  =
    0.081    0.026    0.043    0.043
 -->n=1000; A=rand(n,n); b=rand(n,1);
 -->tic, inv(A); t(1)=toc(); tic, A\b; t(2)=toc(); tic, [L,U]=lu(A); t(3)=toc(); tic, A*A; t(4)=toc();t'
                                           !--error 17
lu: stack size exceeded (Use stacksize function to increase it).
 -->stacksize
 ans  =
    5000000.    1535300.
  -->stacksize(1e8)
 -->tic, inv(A); t(1)=toc(); tic, A\b; t(2)=toc(); tic, [L,U]=lu(A); t(3)=toc(); tic, A*A; t(4)=toc();t'
 ans  =
    0.422    0.163    0.197    0.309
 -->n=2000; A=rand(n,n); b=rand(n,1);
 -->tic, inv(A); t(1)=toc(); tic, A\b; t(2)=toc(); tic, [L,U]=lu(A); t(3)=toc(); tic, A*A; t(4)=toc();t'
 ans  =
    3.141    1.156    1.516    2.33
```

TIME IN mili-seconds

n	inv	solve	lu	matmul
250	11	6	7	6
500	81	26	43	43
1000	422	163	197	309
2000	3141	1156	1516	2330

Comments: SciLab didn't take advantage of the quadcore chip. It also has a default memory stacksize which has to be adjusted to get to n=1000. LU is a bit slower than solve because it forms the permuted matrix moving memory around. Even though inv and matmul use $2n^3$ operations, matmul takes better advantage of modern computers than inv does.



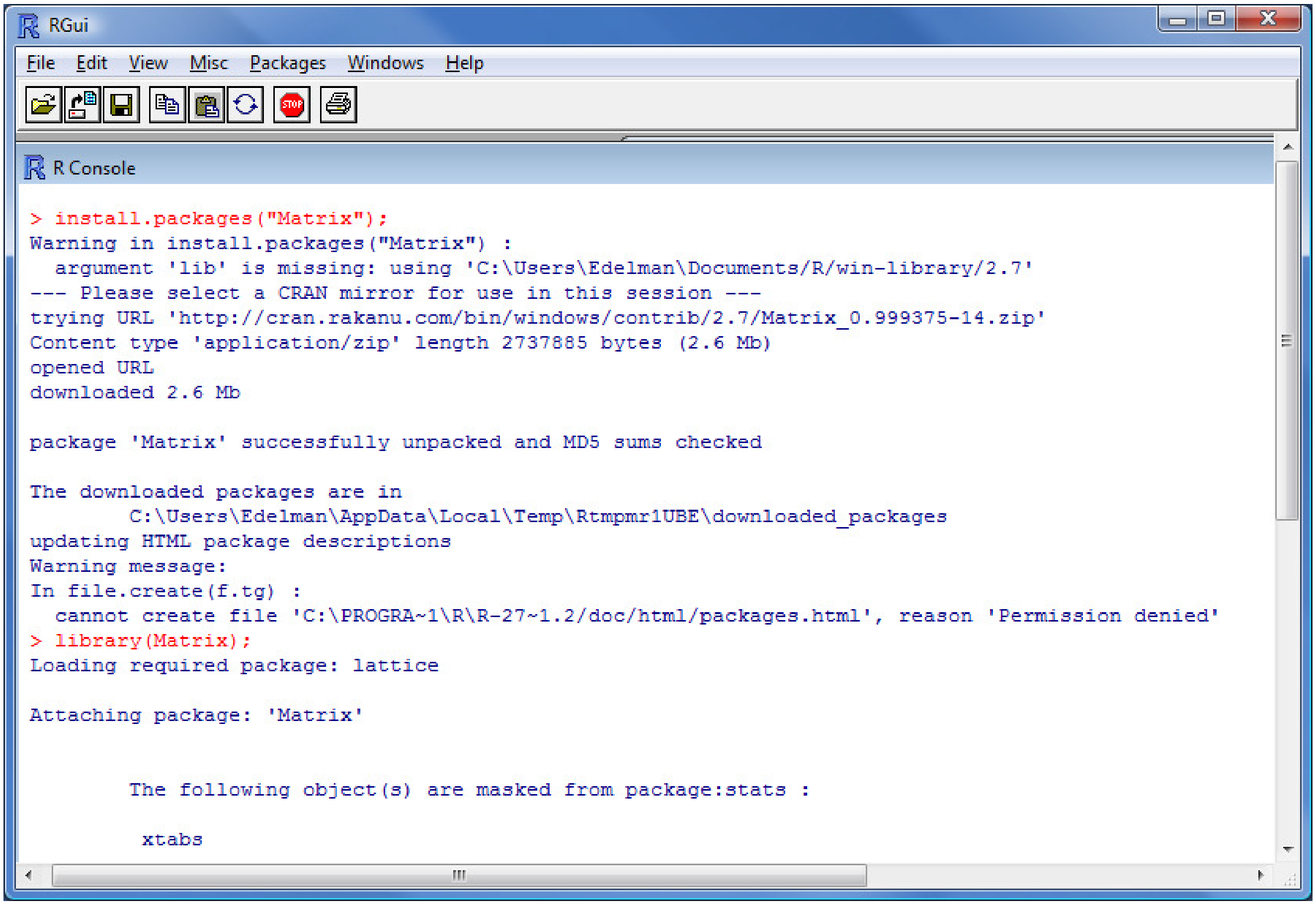
```
In[66]:= MatrixTiming[a_, b_] :=
          {Timing[Inverse[A];][[1]], Timing[LinearSolve[A, b];][[1]],
           Timing[LUDecomposition[A];][[1]], Timing[A.A;][[1]]}

In[65]:= Do[{ A = RandomReal[1, {n, n}]; b = RandomReal[1, n];
           Print[MatrixTiming[A, b]]},
          {n, {250, 500, 1000, 2000, 4000}}]


         {1.06998×10^-14, 1.06998×10^-14, 1.06998×10^-14, 1.06998×10^-14}
         {0.062, 0.016, 0.031, 0.015}
         {0.187, 0.078, 0.078, 0.125}
         {1.295, 0.406, 0.468, 0.702}
         {8.065, 2.402, 2.777, 5.304}
```

TIME IN mili-seconds

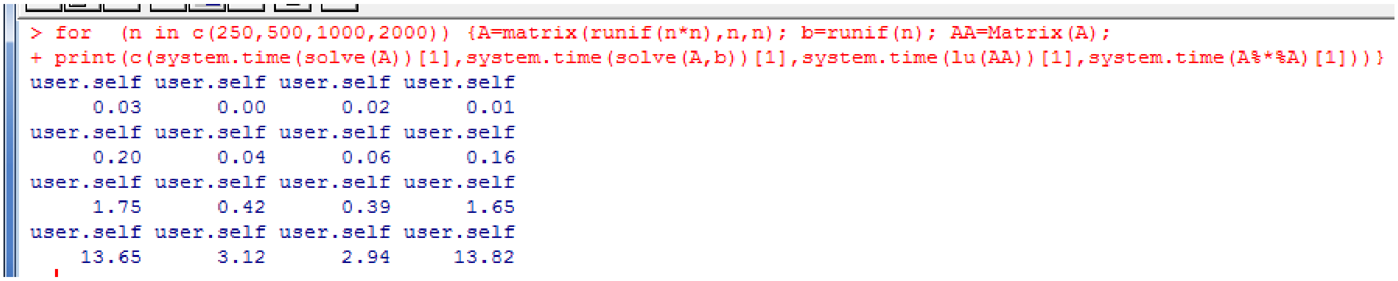
n	inv	solve	lu	matmul
250	≈0	≈0	≈0	≈0
500	62	16	31	15
1000	187	78	78	125
2000	1295	406	468	702
4000	8065	2402	2777	5304

Comments: This implementation sets up a function “MatrixTiming” to do the timing and a “Do” loop to run through matrices of size 250,500,etc. The command Timing[expr;][[1]] returns only the time and not the computed result. Mathematica takes excellent use of the quadcore that I happened to run on.

LU is a bit slower than solve because it forms the permuted matrix moving memory around. Even though inv and matmul use $2n^3$ operations, matmul takes better advantage of modern computers than inv does.



```
> install.packages("Matrix");
Warning in install.packages("Matrix") :
  argument 'lib' is missing: using 'C:\Users\Edelman\Documents/R/win-library/2.7'
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://cran.rakanu.com/bin/windows/contrib/2.7/Matrix_0.999375-14.zip'
Content type 'application/zip' length 2737885 bytes (2.6 Mb)
opened URL
downloaded 2.6 Mb

package 'Matrix' successfully unpacked and MD5 sums checked

The downloaded packages are in
        C:\Users\Edelman\AppData\Local\Temp\Rtmpmr1UBE\downloaded_packages
updating HTML package descriptions
Warning message:
In file.create(f.tg) :
  cannot create file 'C:\PROGRA~1\R\R-27~1.2/doc/html/packages.html', reason 'Permission denied'
> library(Matrix);
Loading required package: lattice

Attaching package: 'Matrix'


        The following object(s) are masked from package:stats :

         xtabs
```

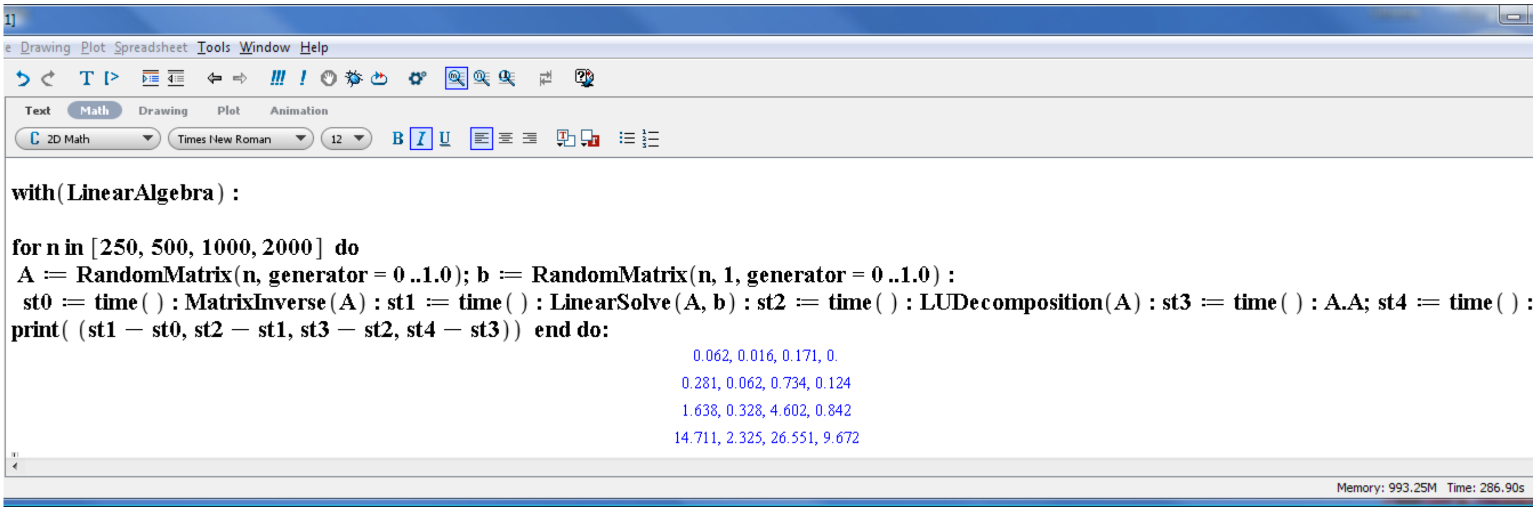


```
> for  (n in c(250,500,1000,2000)) {A=matrix(runif(n*n),n,n); b=runif(n); AA=Matrix(A);
+ print(c(system.time(solve(A))[1],system.time(solve(A,b))[1],system.time(lu(AA))[1],system.time(A%*%A)[1]))}
user.self user.self user.self user.self
     0.03      0.00      0.02      0.01
user.self user.self user.self user.self
     0.20      0.04      0.06      0.16
user.self user.self user.self user.self
     1.75      0.42      0.39      1.65
user.self user.self user.self user.self
    13.65      3.12      2.94     13.82
```

TIME IN mili-seconds

n	inv	solve	lu	matmul
250	30	16	171	0
500	200	62	734	124
1000	1750	328	4602	842
2000	13,650	2325	26551	9672

Comments: My version of R needed the installation of a Matrix Package for the LU. Relatively easy over the internet to do this and install. Seems to use one core,and inefficiently . I decided to loop over the four matrix sizes and “combine” (the “c” command) the output times. The system.time(expression)[1] construct gives the user time.

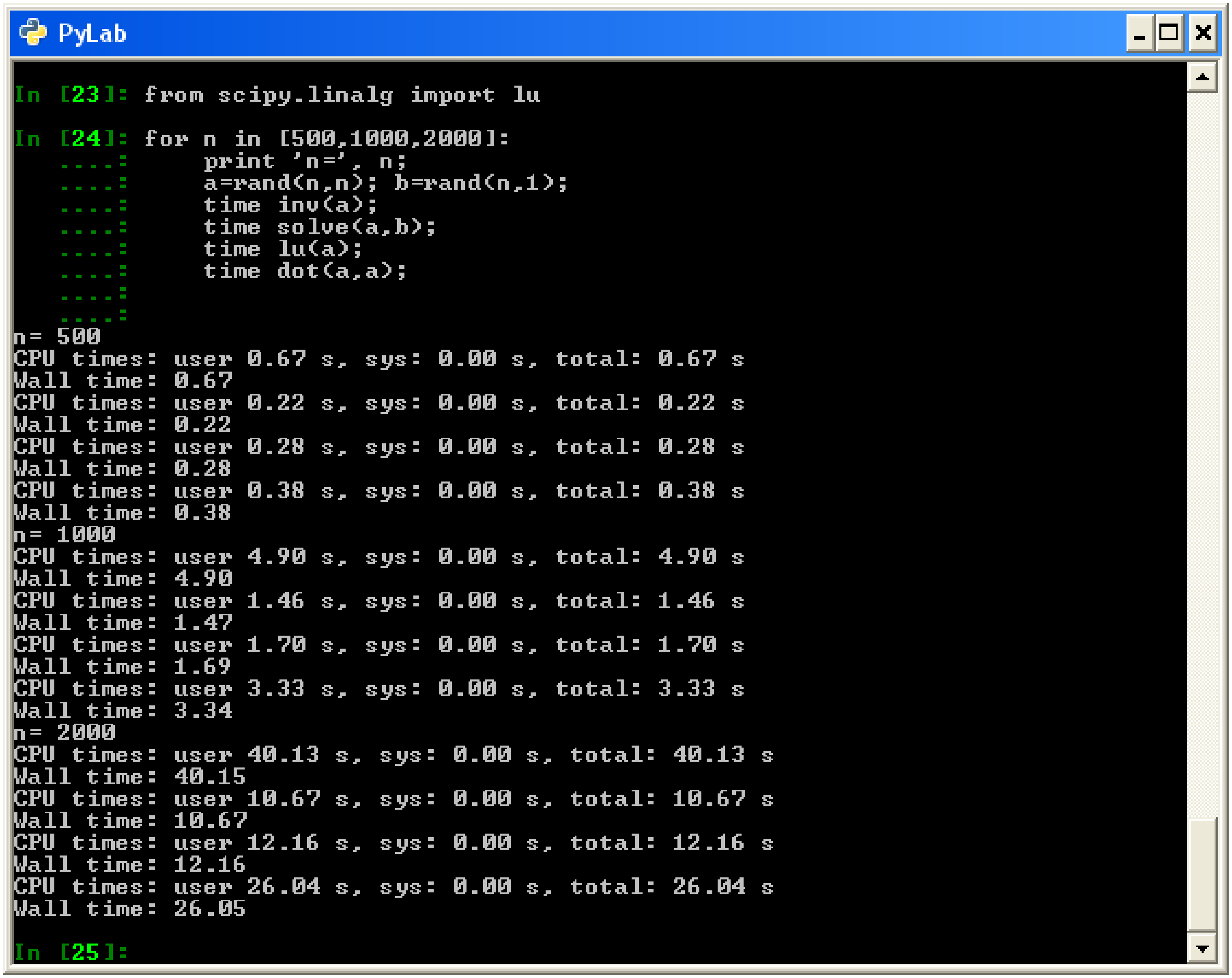

1]
e Drawing Plot Spreadsheet Tools Window Help
Text Math Drawing Plot Animation
2D Math
Times New Roman
12
B I U
Memory: 993.25M Time: 286.90s

```
with(LinearAlgebra) :

for n in [250, 500, 1000, 2000]  do
 A ≔ RandomMatrix(n, generator = 0 ..1.0); b ≔ RandomMatrix(n, 1, generator = 0 ..1.0) :
  st0 ≔ time( ) : MatrixInverse(A) : st1 ≔ time( ) : LinearSolve(A, b) : st2 ≔ time( ) : LUDecomposition(A) : st3 ≔ time( ) : A.A; st4 ≔ time( ) :
print( (st1 − st0, st2 − st1, st3 − st2, st4 − st3))  end do:
                                    0.062, 0.016, 0.171, 0.
                                   0.281, 0.062, 0.734, 0.124
                                   1.638, 0.328, 4.602, 0.842
                                  14.711, 2.325, 26.551, 9.672
```

TIME IN mili-seconds

n	inv	solve	lu	matmul
250	62	6	7	6
500	281	26	43	43
1000	1638	163	197	309
2000	14,711	1156	1516	2330

Comments: Maple seems inefficient even for numerical matrix operations. The numbers indicate software overheads and perhaps disk overheads.



```
In [23]: from scipy.linalg import lu

In [24]: for n in [500,1000,2000]:
   ....:     print 'n=', n;
   ....:     a=rand(n,n); b=rand(n,1);
   ....:     time inv(a);
   ....:     time solve(a,b);
   ....:     time lu(a);
   ....:     time dot(a,a);
   ....:
   ....:
n= 500
CPU times: user 0.67 s, sys: 0.00 s, total: 0.67 s
Wall time: 0.67
CPU times: user 0.22 s, sys: 0.00 s, total: 0.22 s
Wall time: 0.22
CPU times: user 0.28 s, sys: 0.00 s, total: 0.28 s
Wall time: 0.28
CPU times: user 0.38 s, sys: 0.00 s, total: 0.38 s
Wall time: 0.38
n= 1000
CPU times: user 4.90 s, sys: 0.00 s, total: 4.90 s
Wall time: 4.90
CPU times: user 1.46 s, sys: 0.00 s, total: 1.46 s
Wall time: 1.47
CPU times: user 1.70 s, sys: 0.00 s, total: 1.70 s
Wall time: 1.69
CPU times: user 3.33 s, sys: 0.00 s, total: 3.33 s
Wall time: 3.34
n= 2000
CPU times: user 40.13 s, sys: 0.00 s, total: 40.13 s
Wall time: 40.15
CPU times: user 10.67 s, sys: 0.00 s, total: 10.67 s
Wall time: 10.67
CPU times: user 12.16 s, sys: 0.00 s, total: 12.16 s
Wall time: 12.16
CPU times: user 26.04 s, sys: 0.00 s, total: 26.04 s
Wall time: 26.05

In [25]:
```

Comments: Like R Enthought Python didn't have an LU available but it was buried in the scipy.linalg package. How anyone is supposed to find these things is beyond me, but that's another story for another day. WARNING: These timings were on a much older computer so not to be compared with other timings.