Table of Contents

行列の計算速度競争

cc by Shigeto R. Nishitani, 2017-10-30

- file: ~/python/jupyter_num_calc/notebooks_python/LA_speed_race.ipynb

Mapleと，C, pythonで競争させてみました．Cはそのままです．Mapleは

```
with(LinearAlgebra):
data:=[1000,2000,4000];
for n in data do
  A:=RandomMatrix(n,n,generator = 0..1.0):
  b:=RandomVector(n,generator = 0..1.0):
  st:=time();
  LUDecomposition(<A|b>):
  print(time()-st);
end:
```

です．pythonのcodeは，lapackにあるdgesvを指定して 呼び出すようにしています．scipy.laのsolveでは何を使っているかはよくわからないので．

```
import numpy as np
import scipy.linalg as la
import time

sizes = [1000, 2000, 4000]
for n in sizes:
    A = np.random.random((n, n))
    b = np.random.random((n))
    start = time.time()
    la.lapack.dgesv(A, b)
    print('%s [dim], %10.5f [sec] # python' % (n,time.time()-start))
```

結果は，次の通りです．

size	Maple[sec]	C[sec]	python[sec]
1000	0.339	0.0589	0.04101
2000	1.384	0.3826	0.12682
4000	45.482	3.0111	0.81373

pythonの圧勝ですね．絶対値はあまり意味がありません．MacBook Air(13-inch, Early 2015/2.2GHz Core i7, ElCapitan 10.11.6)ですが，softのversionにもよります．

ただし，どんなrandomを生成しているかで結果は大きく変わるので，ちょっと怪しいです． たとえば，MapleのRandomMatrixでgeneratorを指定しないと size=1000でも9.773[sec]ととんでもないぐらい時間がかかります．

MapleではさらにRandomMatrixの生成にも時間がかかります． MapleではNAGのライブラリを使っているので．．．

ちなみにrubyではSciRuby に従ってnmatrixをinstallして，

```
require 'nmatrix'
require 'time'

#n = NMatrix.new(3, [4,9,2,3,5,7,8,1,6], dtype: :float64)
[1000,2000,4000].each do |size|
  n = size
  aa = NMatrix.random([n,n])
  start = Time.now
  lu = aa.factorize_lu
  p Time.now - start
end
```

素直に走らせると，

size	nmatrix[sec]	nmaxtrix/lapacke
1000	0.2118	0.028607
2000	1.9287	0.191315
4000	16.9059	1.313174

のnmatrixです．2017/10/28現在nmatrix-0.2.4です． やっぱ，Rubyは数値計算苦手なんかと諦めかけたんですが，

```
gem install nmatrix nmatrix-lapacke
```

して

```
Require 'nmatrix/lapacke'
```

すると前表のnmatrix/lapacke欄です．これだと，C版に遜色ないですね． nmatrix-lapacke-0.2.3がinstallされています．このあたりの解説がsciruby以外見当たらなくて．．．