tp stat

February 23, 2017

```
In [1]: import math
        import time
        import random
        import pandas as pd

In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline

In [251]:                                     '''
                    Fonction qui permet de diviser un echantillon de taille n ent
                                              '''
          def xfrange(start, stop, step):
              index = 0
              while start + index * step < stop:
                  yield start + index * step
                  index += 1

In [3]: tab_n = list(range(10000 , 110000,10000)) #liste du nombres a considerer pa
```

1 1.1) Simulation par Monte Carlo

```
In [5]:                                     '''
                               Fonction a approcher
                                            '''
        def f(x):
            return math.sqrt(1-(x*x))

In [253]: data = []
          interval_inf = 0
          interval_sup = 1

          for n in tab_n:

              start = time.time()
              surface     = 0
```

```
    dx          = (interval_sup - interval_inf) / n
    xi_simule   = []
    for x in xfrange(0,1,dx):
        xi_simule.append(f(x)*dx)
        surface = surface + f(x)*dx

    end           = time.time()
    moyenne_emp   = surface
    surface       = surface*4

    erreur_abs    = abs((math.pi- surface)/ math.pi )

    variance      = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**
    ecart_type    = math.sqrt(variance)
    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [interval_inf,interval_sup]

    data.append([
          "Monte Carlo",n,abs(start-end),surface,math.pi,erreur_abs,ecart
     ])

df_monte_carlo = pd.DataFrame(data,columns=["methode","n","temps ecoulé",
                                            "val. réelle","erreur relati
```

In [254]:
```
df_monte_carlo
```

Out[254]:

	methode	n	temps ecoulé	val. approchée	val. réelle
0	Monte Carlo	10000	0.015010	3.141791	3.141593
1	Monte Carlo	20000	0.027501	3.141692	3.141593
2	Monte Carlo	30000	0.040032	3.141659	3.141593
3	Monte Carlo	40000	0.052530	3.141643	3.141593
4	Monte Carlo	50000	0.064049	3.141633	3.141593
5	Monte Carlo	60000	0.078043	3.141626	3.141593
6	Monte Carlo	70000	0.090564	3.141621	3.141593
7	Monte Carlo	80000	0.102572	3.141618	3.141593
8	Monte Carlo	90000	0.113580	3.141615	3.141593
9	Monte Carlo	100000	0.126589	3.141613	3.141593

	erreur relative	ecart-type	interval de conf
0	0.000063	78.536932	[0, 1]
1	0.000032	111.070041	[0, 1]
2	0.000021	136.033295	[0, 1]
3	0.000016	157.078198	[0, 1]
4	0.000013	175.619086	[0, 1]
5	0.000011	192.381304	[0, 1]
6	0.000009	207.795739	[0, 1]
7	0.000008	222.143134	[0, 1]

```
8          0.000007  235.618494           [0, 1]
9          0.000006  248.363801           [0, 1]
```

2 1.2) Simulation par jet aléatoire

In [255]:
```
data = []
start = time.time()
for n in tab_n:
    cpt = 0.0
    xi_simule = []
    for i in range(0,n):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        if( x*x+y*y <= 1):
            cpt = cpt+1
            xi_simule.append(1)

    end         = time.time()

    surface     = cpt*4/n
    erreur_abs = abs((math.pi- surface)/ math.pi)

    moyenne_emp = 1/n*cpt
    variance      = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**
    ecart_type    = math.sqrt(variance)

    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [interval_inf,interval_sup]

    data.append([
          "Jet aléatoire",n,abs(start-end),surface,math.pi,erreur_abs,eca
     ])

df_jet_aleatoire = pd.DataFrame(data,columns=["methode","n","temps ecoulé
                                            "val. réelle","erreur relative","
```

In [256]:
```
df_jet_aleatoire
```

Out[256]:
```
          methode      n  temps ecoulé  val. approchée  val. réelle  \
0  Jet aléatoire  10000      0.013012        3.113600     3.141593
1  Jet aléatoire  20000      0.041534        3.121400     3.141593
2  Jet aléatoire  30000      0.081552        3.136000     3.141593
3  Jet aléatoire  40000      0.132593        3.137500     3.141593
4  Jet aléatoire  50000      0.198137        3.134240     3.141593
5  Jet aléatoire  60000      0.276181        3.145200     3.141593
6  Jet aléatoire  70000      0.366253        3.130000     3.141593
7  Jet aléatoire  80000      0.470827        3.132950     3.141593
```

```
8  Jet aléatoire   90000      0.588910        3.133422     3.141593
9  Jet aléatoire  100000      0.720519        3.137800     3.141593

   erreur relative  ecart-type interval de conf
0         0.008910   17.249344           [0, 1]
1         0.006428   24.240169           [0, 1]
2         0.001780   29.331241           [0, 1]
3         0.001303   33.826172           [0, 1]
4         0.002340   37.922310           [0, 1]
5         0.001148   41.159342           [0, 1]
6         0.003690   45.029034           [0, 1]
7         0.002751   48.020051           [0, 1]
8         0.002601   50.912889           [0, 1]
9         0.001207   53.470383           [0, 1]
```

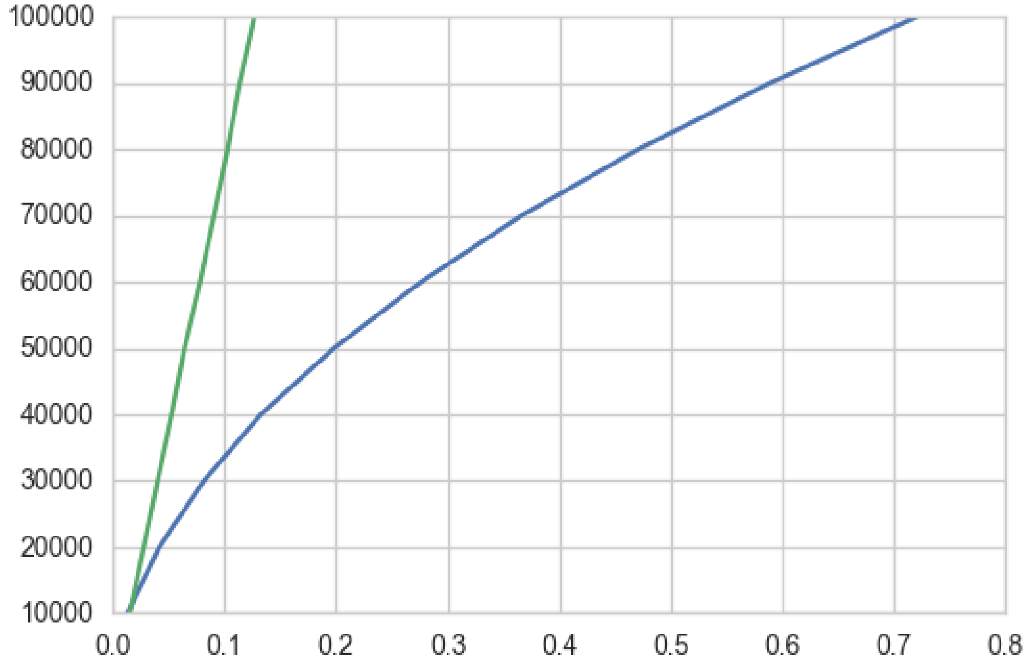
3 1.3) Comparaison des methodes

In [257]:

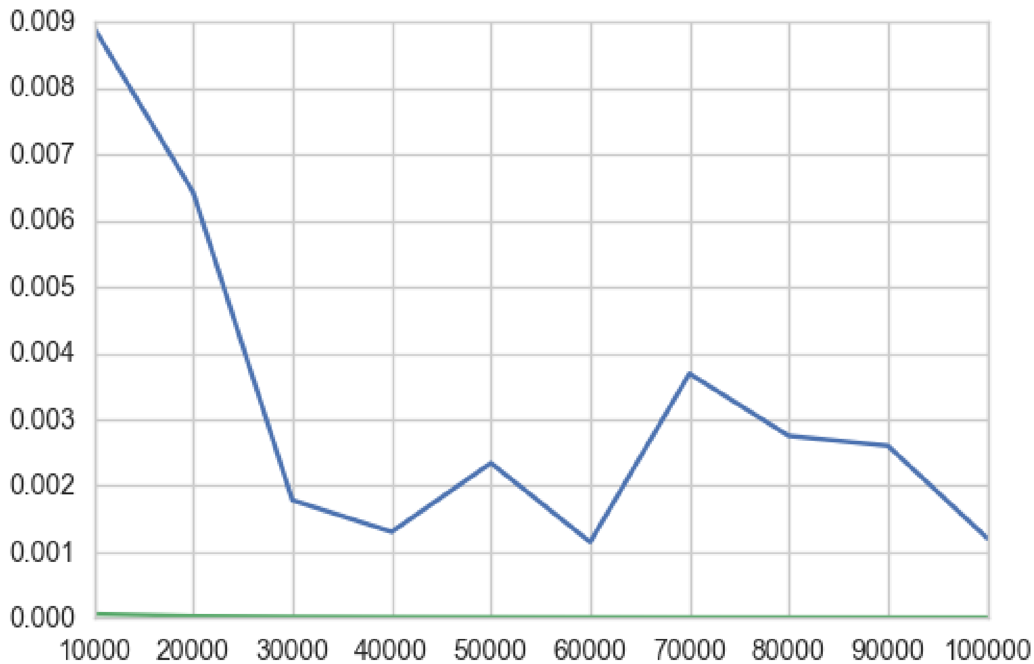
```
'''

VITESSE D'APPROXIMATION

'''
sns.plt.plot(df_jet_aleatoire["temps ecoulé"],df_jet_aleatoire["n"]) # EN
sns.plt.plot(df_monte_carlo["temps ecoulé"],df_jet_aleatoire["n"]) # EN V
'''
    On remarque dans ce graphe que la methode de monte carlo est beaucoup

    EN X: le temps ecoulé
    EN Y: le nombre d'individus pour chaque echantillon

'''
```

Out[257]:

```
"\n    On remarque dans ce graphe que la methode de monte carlo est beauc
```




```
In [270]: '''

          PRECISION DE CONVERGENCE

          '''
          sns.plt.plot(tab_n,df_jet_aleatoire["erreur relative"]) # EN BLEU L"ALEA
          sns.plt.plot(tab_n,df_monte_carlo["erreur relative"]) # EN VERT POUR MON
          '''
              On remarque dans ce graphe que la methode de monte carlo est beaucou

              EN X: le nombre d'individus pour chaque echantillon
              EN Y: le ratio d'erreur relative

          '''

Out[270]: "\n    On remarque dans ce graphe que la methode de monte carlo est beau
```



4 2.1) Simulation de loi normale

```
In [260]: val   = list(range(1,5))
          probs = [0.10,0.20,0.25,0.45]
```

```
In [261]: esperance_par_xi_reel = list(map(lambda x,y:x*y,val,probs))
          esperance_reel        = sum(esperance_par_xi_reel)
```

```
In [262]: def returnX(prob,probs,val):
              stack = 0
              for i in range(len(probs)):
                  stack += probs[i]
                  if prob < stack:
                      return val[i]
              return val[len(probs)-1]
```

```
In [263]: def returnX_reverse(prob,probs,val):
              stack = 1
              for i in reversed(range(len(probs))):
                  stack -= probs[i]
                  if prob > stack:
                      return val[i]
              return val[0]
```

4.0.1 2.1.1) loi normale en considerant les proba croissantes

In [264]:
```
esperance_reel = sum(esperance_par_xi_reel)
data           = []

for n in tab_n:
    xi_simule = []
    start = time.time()

    for i in range(n):
        u = random.uniform(0,1)
        x = returnX(u,probs,val)
        xi_simule.append(x)

    end          = time.time()

    moyenne_emp  = 1/n * (sum(xi_simule))
    variance     = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**
    ecart_type   = math.sqrt(variance)
    erreur_abs   = abs( (esperance_reel - moyenne_emp) / esperance_reel)

    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [str(cinterval_inf),str(cinterval_sup)]

    data.append([
          "random de v.a iid",n,abs(start-end),moyenne_emp,esperance_reel
    ])

df_loi_normale_continue = pd.DataFrame(data,columns=["methode","n","temps
                                        "val. réelle","erreur relative","

    #results.append(["Monte-Carlo",n,end-start,erreur_abs,ecart_type,""])
```

In [265]:
```
df_loi_normale_continue
```

Out[265]:

	methode	n	temps ecoulé	val. approchée	val. réelle
0	random de v.a iid	10000	0.013510	3.053400	3.05
1	random de v.a iid	20000	0.028023	3.044500	3.05
2	random de v.a iid	30000	0.041541	3.051267	3.05
3	random de v.a iid	40000	0.055040	3.053650	3.05
4	random de v.a iid	50000	0.070553	3.056180	3.05
5	random de v.a iid	60000	0.083549	3.053767	3.05
6	random de v.a iid	70000	0.095567	3.049214	3.05
7	random de v.a iid	80000	0.108577	3.045288	3.05
8	random de v.a iid	90000	0.123620	3.050411	3.05
9	random de v.a iid	100000	0.137607	3.049490	3.05

```
   erreur relative    ecart-type                           interval de co
```

```
0         0.001115  6.382450e-14   [3.053399999999999, 3.053400000000017
1         0.001803  9.117841e-14   [3.044499999999999, 3.044500000000015
2         0.000415  2.392678e-13   [3.051266666666664, 3.051266666666669
3         0.001197  6.428902e-13   [3.053649999999994, 3.053650000000064
4         0.002026  1.686256e-13   [3.056179999999999, 3.056180000000017
5         0.001235  1.051480e-12   [3.0537666666666583, 3.05376666666667
6         0.000258  1.914693e-12     [3.0492142857142714, 3.0492142857143
7         0.001545  8.194313e-14  [3.0452874999999997, 3.045287500000006
8         0.000135  3.217486e-13   [3.050411111111109, 3.050411111111113
9         0.000167  1.268047e-12    [3.049489999999992, 3.04949000000000
```

4.0.2 2.1.2) loi normale en mesurant les proba décroissantes

In [266]:
```python
esperance_reel = sum(esperance_par_xi_reel)
data = []

for n in tab_n:
    xi_simule = []
    start = time.time()

    for i in range(n):
        u = random.uniform(0,1)
        x = returnX_reverse(u,probs,val)
        xi_simule.append(x)

    end          = time.time()

    moyenne_emp  = 1/n * (sum(xi_simule))
    variance     = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**
    ecart_type   = math.sqrt(variance)
    erreur_abs   = abs( (esperance_reel - moyenne_emp) / esperance_reel)

    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [str(cinterval_inf),str(cinterval_sup)]

    data.append([
          "random de v.a iid",n,abs(start-end),moyenne_emp,esperance_reel
    ])

df_loi_normale_continue_reverse = pd.DataFrame(data,columns=["methode","n
                                        "val. réelle","erreur relative","
```

In [267]:
```python
df_loi_normale_continue_reverse
```

Out[267]:

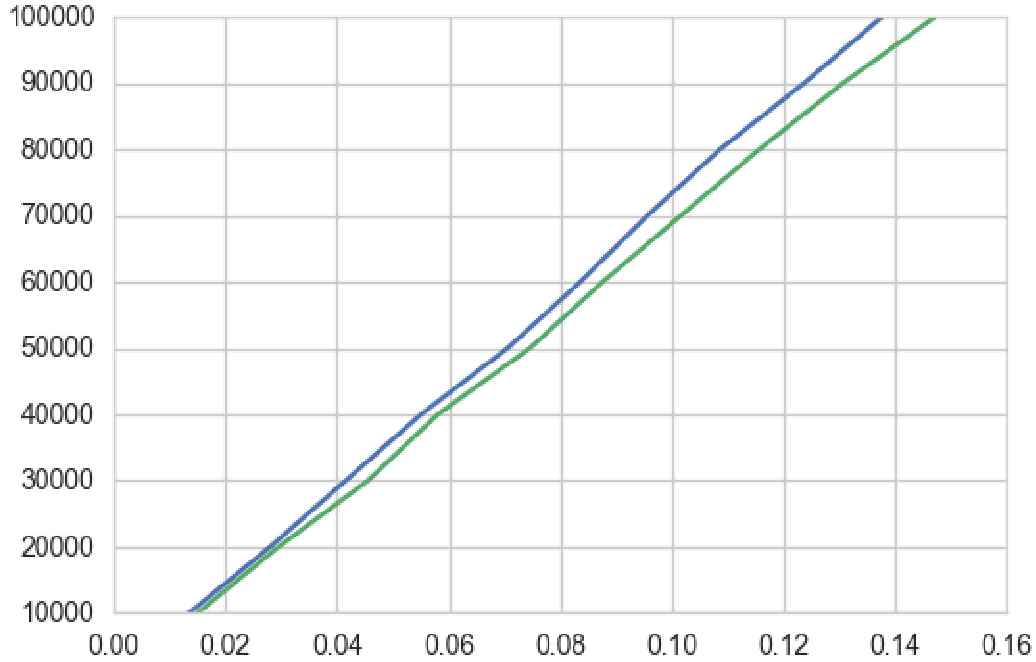
	methode	n	temps ecoulé	val. approchée	val. réelle
0	random de v.a iid	10000	0.015010	3.067400	3.05
1	random de v.a iid	20000	0.029521	3.052350	3.05

```
2  random de v.a iid   30000      0.045532        3.048967         3.05
3  random de v.a iid   40000      0.058054        3.045550         3.05
4  random de v.a iid   50000      0.074545        3.054800         3.05
5  random de v.a iid   60000      0.087566        3.048900         3.05
6  random de v.a iid   70000      0.101567        3.048257         3.05
7  random de v.a iid   80000      0.115585        3.050938         3.05
8  random de v.a iid   90000      0.130592        3.048867         3.05
9  random de v.a iid  100000      0.147102        3.052910         3.05

   erreur relative    ecart-type                          interval de cor
0         0.005705  8.665957e-14   [3.0673999999999984, 3.067400000000002
1         0.000770  4.760834e-14  [3.0523499999999997, 3.052350000000006
2         0.000339  1.783791e-13   [3.0489666666666646, 3.048966666666669
3         0.001459  1.866840e-13   [3.045549999999998, 3.045550000000001
4         0.001574  4.067384e-14  [3.0547999999999997, 3.054800000000006
5         0.000361  6.363112e-13   [3.0488999999999953, 3.048900000000005
6         0.000571  2.668076e-13   [3.048257142857141, 3.048257142857144
7         0.000307  5.621685e-13  [3.0509374999999963, 3.050937500000004
8         0.000372  1.779252e-12    [3.048866666666655, 3.048866666666667
9         0.000954  7.108665e-13   [3.052909999999996, 3.052910000000004
```

5 2.3) Comparaison des methodes

In [268]:
```
'''

VITESSE D'APPROXIMATION

'''
sns.plt.plot(df_loi_normale_continue["temps ecoulé"],tab_n) # EN BLEU
sns.plt.plot(df_loi_normale_continue_reverse["temps ecoulé"],tab_n) # EN
'''
    On remarque dans ce graphe que la methode reverse (prob décroissante)
    vitesse(bleu) > vitesse(vert)

    EN X: le temps ecoulé
    EN Y: le nombre d'individus pour chaque echantillon

'''
```

Out[268]:
```
"\n    On remarque dans ce graphe que la methode reverse (prob décroissan
```




```
In [19]: '''

         PRECISION DE CONVERGENCE

         '''
         sns.plt.plot(tab_n,df_loi_normale_continue["erreur relative"]) # EN BLEU I
         sns.plt.plot(tab_n,df_loi_normale_continue_reverse["erreur relative"]) #VE
         '''
             On remarque dans ce graphe que la methode de monte carlo est beaucoup

             EN X: le nombre d'individus pour chaque echantillon
             EN Y: le ratio d'erreur relative

         '''


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-19-41e44da47a16> in <module>()
          4
          5 '''
    ----> 6 sns.plt.plot(tab_n,df_loi_normale_continue["erreur relative"]) # EN BLE
          7 sns.plt.plot(tab_n,df_loi_normale_continue_reverse["erreur relative"])
```

```
8 '''
```

```
NameError: name 'df_loi_normale_continue' is not defined
```

6 3.1) Simulation de loi géometrique (pseudo inverse de la f.d.r)

In [14]:
```
data = []

q = 0.7
esperance_reel = q/(1-q)

for n in tab_n:
    xi_simule = []
    stack = 0
    start = time.time()
    for index in range(n):
        u = random.uniform(0,1)
        i = 1
        while True:
            if 1-(q**(i-1)) <= u < 1-(q**(i)):
                break
            else:
                i+=1
        xi_simule.append(i-1)
    end = time.time()
    moyenne_emp = 1/n * sum(xi_simule)
    variance     = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**2
    ecart_type   = math.sqrt(variance)
    erreur_abs   = (esperance_reel - moyenne_emp) / esperance_reel

    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [str(cinterval_inf),str(cinterval_sup)]

    data.append([
           "invers fdr geo",n,abs(start-end),moyenne_emp,esperance_reel,abs
    ])

df_loi_geom_inv_fdr = pd.DataFrame(data,columns=["methode","n","temps ecou
                                        "val. réelle","erreur relative","e
```

In [15]:
```
df_loi_geom_inv_fdr
```

Out[15]:
```
          methode      n  temps ecoulé  val. approchée  val. réelle  \
0  invers fdr geo  10000      0.030033        2.332300     2.333333
1  invers fdr geo  20000      0.060563        2.322300     2.333333
```

```
2  invers fdr geo   30000      0.083060        2.335067     2.333333
3  invers fdr geo   40000      0.112580        2.344475     2.333333
4  invers fdr geo   50000      0.139601        2.333440     2.333333
5  invers fdr geo   60000      0.169620        2.360433     2.333333
6  invers fdr geo   70000      0.204654        2.353686     2.333333
7  invers fdr geo   80000      0.231168        2.332613     2.333333
8  invers fdr geo   90000      0.255170        2.342978     2.333333
9  invers fdr geo  100000      0.278206        2.317990     2.333333

   erreur relative    ecart-type                          interval de conf
0         0.000443  2.735945e-13  [2.3322999999999947, 2.3323000000000054]
1         0.004729  5.073911e-13   [2.322299999999993, 2.3223000000000074]
2         0.000743  3.579351e-13  [2.3350666666666626, 2.3350666666666706]
3         0.004775  1.595484e-12  [2.3444749999999845, 2.3444750000000156]
4         0.000046  6.985573e-13   [2.333439999999994, 2.3334400000000066]
5         0.011614  9.269075e-14  [2.3604333333333325, 2.3604333333333343]
6         0.008722  1.555734e-13   [2.353685714285713, 2.3536857142857155]
7         0.000309  1.831215e-12   [2.3326124999999873, 2.332612500000013]
8         0.004133  1.809187e-12     [2.342977777777766, 2.34297777777779]
9         0.006576  1.639497e-12      [2.31798999999999, 2.31799000000001]
```

7 3.2) Simulation de loi géometrique par la formule LN

In [24]:
```
esperance_reel = q/(1-q)
data = []
for n in tab_n:
    xi_simule = []

    start = time.time()

    for index in range(n):

        xi = int(math.log( (1-random.uniform(0,1) ) ) / math.log(q))
        xi_simule.append(xi)

    end = time.time()

    moyenne_emp = 1/n * sum(xi_simule)
    variance     = 1/n*(sum(map(lambda x : x - moyenne_emp, xi_simule))**2
    ecart_type   = math.sqrt(variance)
    erreur_abs   = (esperance_reel - moyenne_emp) / esperance_reel

    cinterval_inf = moyenne_emp-(1.96*ecart_type/math.sqrt(n))
    cinterval_sup = moyenne_emp+(1.96*ecart_type/math.sqrt(n))
    confidence = [str(cinterval_inf),str(cinterval_sup)]

    data.append([
```

```
                "Loi geo app. par LN",n,abs(start-end),moyenne_emp,esperance_ree
        ])

    df_loi_geom_LN = pd.DataFrame(data,columns=["methode","n","temps ecoulé","
                                                "val. réelle","erreur relative","e
```

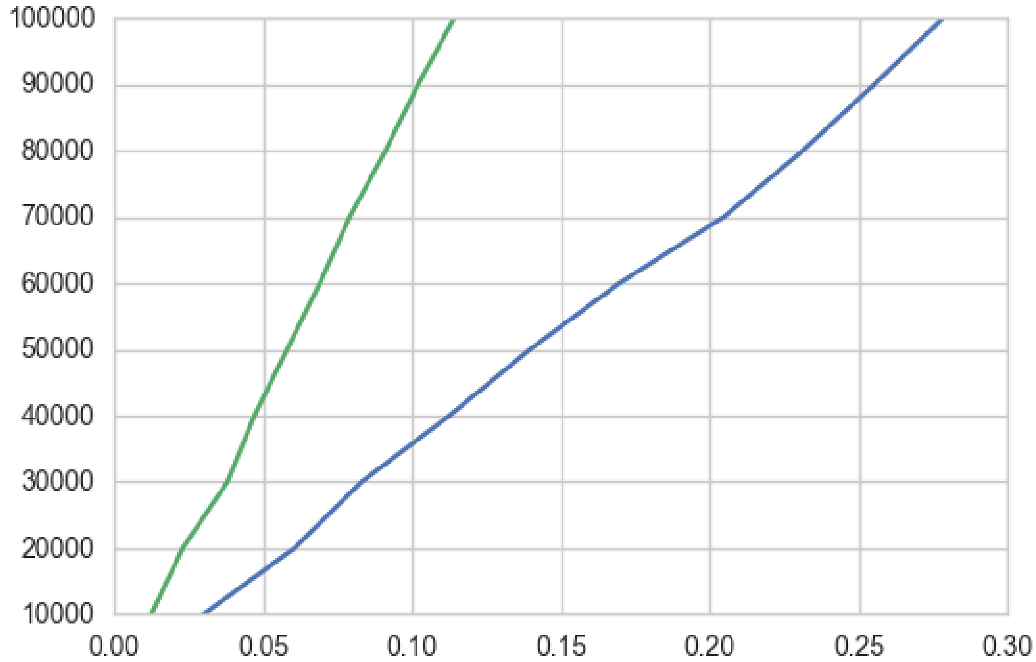
```
In [25]: df_loi_geom_LN
```

```
Out[25]:               methode       n  temps ecoulé  val. approchée  val. réelle
         0  Loi geo app. par LN   10000      0.012511        2.382500     2.333333
         1  Loi geo app. par LN   20000      0.023019        2.311450     2.333333
         2  Loi geo app. par LN   30000      0.038031        2.347667     2.333333
         3  Loi geo app. par LN   40000      0.047080        2.337300     2.333333
         4  Loi geo app. par LN   50000      0.058053        2.330640     2.333333
         5  Loi geo app. par LN   60000      0.069044        2.353200     2.333333
         6  Loi geo app. par LN   70000      0.079063        2.335557     2.333333
         7  Loi geo app. par LN   80000      0.091060        2.350975     2.333333
         8  Loi geo app. par LN   90000      0.102070        2.325656     2.333333
         9  Loi geo app. par LN  100000      0.114081        2.321460     2.333333

            erreur relative    ecart-type                          interval de conf
         0         0.021071  3.439737e-13   [2.3824999999999936, 2.382500000000007]
         1         0.009379  3.609957e-14              [2.31145, 2.3114500000000007]
         2         0.006143  2.735530e-13  [2.3476666666666635, 2.3476666666666697]
         3         0.001700  1.971316e-12  [2.3372999999999804, 2.3373000000000195]
         4         0.001154  5.819735e-13   [2.3306399999999954, 2.330640000000005]
         5         0.008514  2.117742e-12   [2.3531999999999833, 2.353200000000017]
         6         0.000953  6.230657e-13   [2.3355571428571382, 2.335557142857147]
         7         0.007561  6.450756e-13  [2.3509749999999956, 2.3509750000000045]
         8         0.003290  2.272641e-13   [2.3256555555555543, 2.325655555555557]
         9         0.005089  5.337759e-13    [2.321459999999997, 2.321460000000003]
```

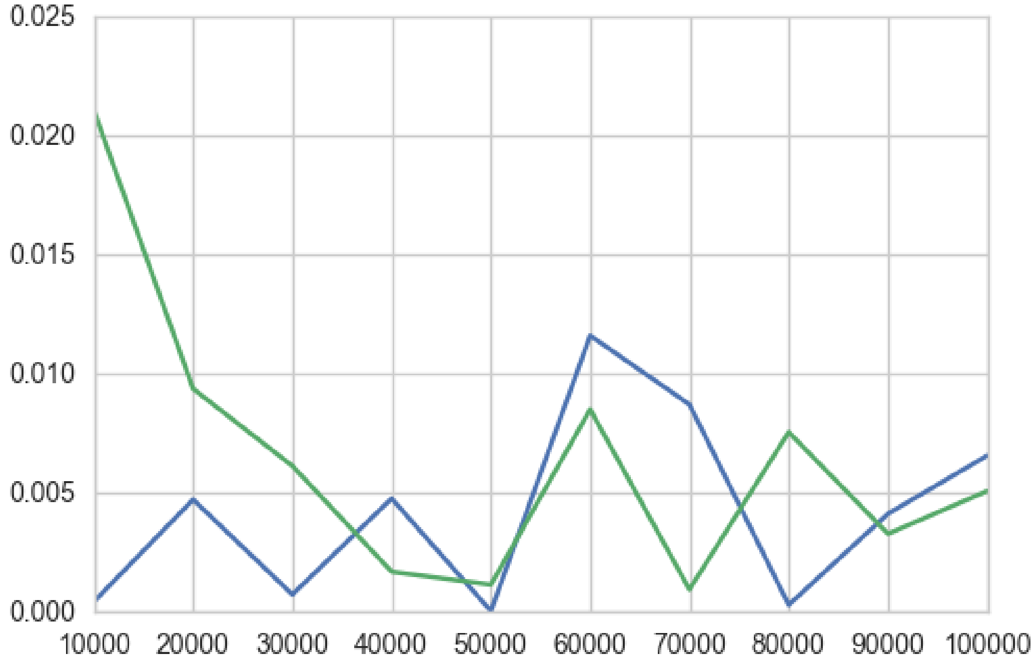
8 3.3) Comparaison des methodes

```
In [27]: '''

         VITESSE D'APPROXIMATION

         '''
         sns.plt.plot(df_loi_geom_inv_fdr["temps ecoulé"],tab_n) # EN BLEU, inverse
         sns.plt.plot(df_loi_geom_LN["temps ecoulé"],tab_n) # EN VERT, ln(1-q)/ln(
         '''

             EN X: le temps ecoulé
             EN Y: le nombre d'individus pour chaque echantillon

         '''
```

```
Out[27]: "\n    \n    EN X: le temps ecoulé \n    EN Y: le nombre d'individus pour
```




```
In [26]: '''

         PRECISION DE CONVERGENCE

         '''
         sns.plt.plot(tab_n,df_loi_geom_inv_fdr["erreur relative"])  # EN BLEU, inv
         sns.plt.plot(tab_n,df_loi_geom_LN["erreur relative"])  # EN VERT, ln(1-q)/
         '''

             EN X: le nombre d'individus pour chaque echantillon
             EN Y: le ratio d'erreur relative

         '''

Out[26]: "\n        \n    EN X: le nombre d'individus pour chaque echantillon\n
```


0.025
0.020
0.015
0.010
0.005
0.000
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000