Emerveillement (Denise Vella-Chemla, 8.7.2017)

Après avoir étudié la note de Bernhard Riemann *"Sur le nombre des nombres premiers inférieurs à une grandeur donnée"* et en avoir programmé quelques formules, on aboutit au programme suivant, en C++, qui résume toute la puissance de la fonction Logarithme intégral $Li(x)$, qui permet l'obtention d'une excellente approximation du nombre de nombres premiers inférieurs à un nombre donné à partir de son logarithme intégral auquel on soustrait la moitié du nombre de nombres premiers inférieurs à sa racine carrée.

Présentons les résultats que l'on peut obtenir par programme (la colonne dans laquelle on soustrait directement le nombre de nombres premiers inférieurs à la racine du nombre considéré plutôt que la moitié d'un tel nombre est dû au fait d'une coquille de programmation initiale qui montre que l'une ou l'autre des possibilités donnent des résultats semblables (du moins jusqu'à 10^9).

x	$Li(x)$	$\sqrt{x}$	$\pi(\sqrt{x})$	$\pi(x)$	$Li(x) - \pi(\sqrt{x})$	$Li(x) - (1/2)\pi(\sqrt{x})$
10^2	29	10	4	25	25	27
10^3	177	31	11	168	166	172
10^4	1 245	100	25	1 229	1 220	1 233
10^5	9 629	316	65	9 592	9 564	9 597
10^6	78 627	1 000	168	78 498	78 459	78 543
10^7	664 917	3 162	446	664 579	664 471	664 694
10^8	5 762 208	10 000	1 229	5 761 455	5 760 979	5 761 594
10^9	50 849 233	31 622	3 401	50 847 534	50 845 832	50 847 533

Voici le tableau des écarts relatifs à $\pi(x)$.

x	$\frac{\pi(x)-(Li(x)-\pi(\sqrt{x})))}{\pi x}$	$\frac{\pi(x)-(Li(x)-\frac{1}{2}\pi(\sqrt{x})))}{\pi(x)}$
10^2	0	-0.08 ($= -2/25$)
10^3	0.0119 ($=$ 2/168)	0.0238 ($=$ 4/168)
10^4	0.0073 ($=$ 9/1 229)	0.003254 ($=$ 4/1 229)
10^5	0.0029 ($=$ 28/9 592)	0.00052126 ($=$ 5/9 592)
10^6	0.00049 ($=$ 39/78 498)	0.00057326 ($=$ 45/78 498)
10^7	0.000162 ($=$ 108/664 579)	0.000173041 ($=$ 115/664 579)
10^8	0.00008261 ($=$ 476/5 761 455)	0.000024125 ($=$ 139/5 761 455)
10^9	0.00000586065 ($=$ 298/50 847 534)	$1,96.10^{-8}$ ($=$ 1/50 847 533)

```cpp
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <fstream>

int prime(int atester) {
  bool pastrouve=true;
  unsigned long k = 2;

  if (atester == 1) return 0;
  if (atester == 2) return 1;
  if (atester == 3) return 1;
  if (atester == 5) return 1;
  if (atester == 7) return 1;
  while (pastrouve) {
    if ((k * k) > atester) return 1;
    else
      if ((atester % k) == 0) return 0 ;
      else k++;
    }
}

int main (int argc, char* argv[]) {
  int n, i, nbpremiers ;
  float res, rac, nbpremrac ;
  float logainte[10010] ;

  std::ifstream fichier("petitlog", std::ios::in);
  if (fichier) {
    float flotte ;

    i = 2 ;
    while (not fichier.eof()) {
      fichier >> flotte ;
      logainte[i] = flotte-1.04516378011749 ;
      //std::cout << "li(" << i << ") = " << logainte[i] << "\n" ;
      i = i+1 ;
    }
    fichier.close();
  }
  else std::cerr << "Impossible d'ouvrir le fichier !" << std::endl ;

  for (n = 1 ; n <= 10000 ; ++n) {
    rac = sqrt(n) ;
    nbpremrac = 0 ;
    nbpremiers = 0 ;
    for (i = 2 ; i <= rac ; ++i) if (prime(i)) nbpremrac = nbpremrac+1 ;
    for (i = 2 ; i <= n ; ++i) if (prime(i)) nbpremiers = nbpremiers+1 ;
    res = logainte[n]-0.5*nbpremrac ;
    std::cout << n << " -> " << nbpremiers << " approx " << res ;
    std::cout << `` erreur = `` << (nbpremiers-res)/nbpremiers << ``\n'' ;
  }
}
```

Les premières et dernières lignes de l'exécution de ce programme sont :

```
3 -> 2 approx 1.11842 erreur 0.440788
4 -> 2 approx 1.42242 erreur 0.288789
5 -> 3 approx 2.08942 erreur 0.303525
6 -> 3 approx 2.67706 erreur 0.107647
7 -> 4 approx 3.21189 erreur 0.197028
8 -> 4 approx 3.70855 erreur 0.0728613
9 -> 4 approx 3.67607 erreur 0.0809815
10 -> 4 approx 4.12044 erreur -0.0301089
11 -> 5 approx 4.54585 erreur 0.0908309
12 -> 5 approx 4.95538 erreur 0.00892324
13 -> 6 approx 5.35138 erreur 0.108103
14 -> 6 approx 5.73566 erreur 0.0440563
15 -> 6 approx 6.10966 erreur -0.0182769
16 -> 6 approx 6.47455 erreur -0.0790921
17 -> 7 approx 6.8313 erreur 0.0240998
18 -> 7 approx 7.18071 erreur -0.0258157
19 -> 8 approx 7.52346 erreur 0.0595673
20 -> 8 approx 7.86014 erreur 0.017483
21 -> 8 approx 8.19123 erreur -0.0239042
22 -> 8 approx 8.51719 erreur -0.0646486
23 -> 9 approx 8.83838 erreur 0.0179575
24 -> 9 approx 9.15515 erreur -0.017239
25 -> 9 approx 8.96779 erreur 0.00357861
26 -> 9 approx 9.27657 erreur -0.03073
27 -> 9 approx 9.58172 erreur -0.0646358
28 -> 9 approx 9.88346 erreur -0.0981627
29 -> 10 approx 10.182 erreur -0.0181988
30 -> 10 approx 10.4775 erreur -0.0477468
31 -> 11 approx 10.7701 erreur 0.0209031
32 -> 11 approx 11.0599 erreur -0.00544799
33 -> 11 approx 11.3472 erreur -0.0315625


9971 -> 1228 approx 1229.44 erreur -0.00117498
9972 -> 1228 approx 1229.55 erreur -0.00126345
9973 -> 1229 approx 1229.66 erreur -0.000537149
9974 -> 1229 approx 1229.77 erreur -0.000625449
9975 -> 1229 approx 1229.88 erreur -0.000713848
9976 -> 1229 approx 1229.99 erreur -0.000802247
9977 -> 1229 approx 1230.09 erreur -0.000890646
9978 -> 1229 approx 1230.2 erreur -0.000978946
9979 -> 1229 approx 1230.31 erreur -0.00106735
9980 -> 1229 approx 1230.42 erreur -0.00115565
9981 -> 1229 approx 1230.53 erreur -0.00124404
9982 -> 1229 approx 1230.64 erreur -0.00133244
9983 -> 1229 approx 1230.75 erreur -0.00142074
9984 -> 1229 approx 1230.85 erreur -0.00150914
9985 -> 1229 approx 1230.96 erreur -0.00159744
9986 -> 1229 approx 1231.07 erreur -0.00168584
9987 -> 1229 approx 1231.18 erreur -0.00177424
9988 -> 1229 approx 1231.29 erreur -0.00186254
9989 -> 1229 approx 1231.4 erreur -0.00195094
9990 -> 1229 approx 1231.51 erreur -0.00203924
9991 -> 1229 approx 1231.61 erreur -0.00212764
9992 -> 1229 approx 1231.72 erreur -0.00221594
9993 -> 1229 approx 1231.83 erreur -0.00230434
9994 -> 1229 approx 1231.94 erreur -0.00239264
9995 -> 1229 approx 1232.05 erreur -0.00248104
9996 -> 1229 approx 1232.16 erreur -0.00256934
9997 -> 1229 approx 1232.27 erreur -0.00265774
9998 -> 1229 approx 1232.37 erreur -0.00274604
9999 -> 1229 approx 1232.48 erreur -0.00283443
10000 -> 1229 approx 1232.59 erreur -0.00292273
```

En python, le programme est plus court, et plus rapide à l'exécution :

```
import mpmath
from mpmath import *
from math import *

def prime(atester):
  pastrouve = True
  k = 2
  if (atester == 1): return False
  if (atester == 2): return True
  if (atester == 3): return True
  if (atester == 5): return True
  if (atester == 7): return True
  while (pastrouve):
    if ((k * k) > atester):
        return True
    else:
        if ((atester % k) == 0):
            return False
        else: k=k+1

def pi(x):
    nbpremiers = 0
    for y in range(1,x):
        if prime(y):
            nbpremiers=nbpremiers+1
    return nbpremiers

x=int(input())
res = li(x)-li(2)
a = pi(x)
print "pi(x) = %d" %a
b = int(sqrt(x))
c = pi(b)
res=res-(1.0/2.0)*c
print "res = %10.6f" %res
erreur = (res-a)/a
print "erreur = %10.6f" %erreur
```

Par exécution des programmes en python, on obtient les résultats suivants, légèrement différents de ceux obtenus en c++ : x = 100 pi(x) = 25
res = 27.080978
erreur = 0.083239

x = 1000 pi(x) = 168
res = 171.564494
erreur = 0.021217

x = 10000 pi(x) = 1229
res = 1232.592052
erreur = 0.002923

x = 100000 pi(x) = 9592
res = 9596.263837
erreur = 0.000445

x = 1000000 pi(x) = 78498
res = 78542.503996
erreur = 0.000567

x = 10000000 pi(x) = 664579
res = 664694.359885
erreur = 0.000174

x = 100000000 pi(x) = 5761455
res = 5761593.830284
erreur = 0.000024

Référence

Traduction en français par L.Laugel in *Riemann, œuvres mathématiques*, Gauthier-Villars, 1898., de *Über die Anzahl der Primzahlen unter einer gegebenen Grösse, Monat. der Königl. Preuss. Akad. der Wissen. zu Berlin aus der Jahre 1859 (1860) 671-680.*