Tiny programs for constants computation

Xavier Gourdon and Pascal Sebah
May 28, 2003[1]

(The smallest C codes to compute classical mathematical constants)

This page contains some very tiny programs in relation with the computation of classical mathematical constants and prime numbers.

Wanted : any shorter C codes for any constants, or tiny C codes for $\zeta(3)$, or any other classical constant. Send any new tiny code to : *xgourdon@yahoo.fr* or to *psebah@yahoo.fr*.

All the following programs (except the wonderful tiny program that computes Euler constant γ) use the so called *Spigot-Algorithm* [1], that is, algorithms for which the digits are calculated and printed one at the time.

1 Computation of π

1.1 C programs

The following tiny C code computes digits of π. Boris Gourevitch kindly sent me the information that this program is from Dik T. Winter (cwi institute, Holland).

```
int a=10000,b,c=8400,d,e,f[8401],g;main(){
for(;b-c;)f[b++]=a/5;
for(;d=0,g=c*2;c-=14,printf("%.4d",e+d/a),e=d%a)
for(b=c;d+=f[b]*a,f[b]=d%--g,d/=g--,--b;d*=b);}
```

The program has 158 characters long. Finding the algorithm used is not so easy. Sebastian Wedeniws worked hard to find a shorter program (142 characters) :

```
main(){int a=1e4,c=3e3,b=c,d=0,e=0,f[3000],g=1,h=0;
for(;b;!--b?printf("%04d",e+d/a),e=d%a,h=b=c-=15:f[b]=
(d=d/g*b+a*(h?f[b]:2e3))%(g=b*2-1));}
```

Jean-Charles Meyrignac pointed me out a tiny code from the page http://www.isr.umd.edu/~jasonp/pigjerr. The program is by Gjerrit Meinsma, of 141 characters long, and computes 1000 digits of pi.

```
long k=4e3,p,a[337],q,t=1e3;
  main(j){for(;a[j=q=0]+=2,--k;)
  for(p=1+2*k;j<337;q=a[j]*k+q%p*t,a[j++]=q/p)
  k!=j>2?:printf("%.3d",a[j-2]%t+q/p/t);}
```

[1] This pages are from *//numbers.computation.free.fr/Constants/constants.html*

1.2 Assembly programs

Michal Majer improved his assembly program and afforded an tiny executable (MSDOS i386) of just 121 bytes that computes 9280 digits of pi. The program can be downloaded here : *pi.com*. Assembly *source code* is also available.

2 Computation of e

This is a tiny C program from Xavier Gourdon (Jul 1999) to compute 9000 decimal digits of e on your computer.

```
main(){int N=9009,n=N,a[9009],x;while(--n)a[n]=1+1/n;
for(;N>9;printf("%d",x))
for(n=N--;--n;a[n]=x%n,x=10*a[n-1]+x/n);}
```

This program has 117 characters (try to do better !). It can be changed to compute more digits (change the value 9009 to more) and to be faster (change the constant `10` to another power of 10 and the `printf` command). A not so obvious question is to find the algorithm used.

3 Computation of $\log(2)$

The two following programs are short C program from Pascal Sebah (Jan 2000) to compute 2400 decimal places of $\log(2)$. In the two algorithms, the number of digits is given by $\log_{10}(a).n$ where a is the working base (here $a = 10^3$ or less if you want to compute more digits) and c is the numbers of iteration required, the size of the array f[] is $c + 1$.

The first program is based on the *hypergeometric representation* of $x^{-1}\log(1-x)$ with $x = 1/2$ (see [1]) :

```
main(){int a=1000,b=0,c=7973,d,f[7974],n=800,k;
for(;b<c;f[b++]=5);
for(;n--;d+=*f*a,printf("%.3d",d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=k)f[k]=(d+=f[k]*a)%(b=2*k+2);}
```

The second program is almost the same but deduced from an alternating series of $\log(2)$ given in the essay on this number, it's about three time faster than the previous one :

```
main(){int a=1000,b=0,c=2658,d=75,f[2659],n=800,k;
for(;b<c;f[b++]=d,d=-d);
for(;n--;d+=*f*a,printf("%.3d",d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=k)f[k]=(d+=f[k]*a)%(b=8*k+4);}
```

More digits may be given by those programs, just change the value of n and proportionally the value of c. For example, the last program, with $n = 4000$ and $c = 13288$, will give about 12000 digits.

4 Computation of square root of 2 and φ

With exactly the same pattern, P. Sebah also gave tiny codes for the computations of $\sqrt{2}$ and the *Golden Ratio* $\varphi = (1+\sqrt{5})/2$. The meaning of the parameters is the same as for $\log(2)$, the number of calculated digits is 2400.

The algorithm to compute $\sqrt{2}$ is based on the relation

$$\sqrt{2} = \frac{7}{5}\left(1 - \frac{1}{50}\right)^{-1/2},$$

and on the expansion

$$(1-x)^{-m} = 1 + mx + \frac{m(m+1)}{2!}x^2 + ...$$

it produces the following program :

```
main(){int a=1000,b=0,c=1413,d,f[1414],n=800,k;
for(;b<c;f[b++]=14);
for(;n--;d+=*f*a,printf("%.3d",d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=2*k-1)f[k]=(d+=f[k]*a)%(b=100*k);}
```

And the algorithm to find the *Golden Ratio* is based on

$$\varphi = \frac{1+\sqrt{5}}{2} = \frac{1}{2} + \left(1 - \frac{1}{5}\right)^{-1/2},$$

giving

```
main(){int a=1000,b=0,c=3434,d,f[3435],n=800,k;
for(;b<c;f[b++]=1);f[1]+=5;
for(;n--;d+=*f*a,printf("%.3d",d/a),*f=d%a)
for(d=0,k=c;--k;d/=b,d*=2*k-1)f[k]=(d+=f[k]*a)%(b=10*k);}
```

5 Generalization

In fact the previous pattern is more general and may be easily adapted to many constants C defined by a Gaussian hypergeometric function [1] :

$$\begin{aligned} C &= \alpha F(a,b;c;z) = \alpha \sum_{k=0}^{\infty} d_k = \alpha\left(1 + \frac{a.b}{c}\frac{z}{1!} + \frac{a(a+1).b(b+1)}{c(c+1)}\frac{z^2}{2!} + ...\right) \\ d_k &= d_{k-1}\frac{(a+k-1)(b+k-1)}{(c+k-1)k}z, \end{aligned}$$

with (a, b, c, z, α) being real numbers.

For example

$$\begin{aligned}(1-x)^{-m} &= F(m,b,b;x)\\ d_k &= d_{k-1}\frac{(m+k-1)}{k}x,\end{aligned}$$

and with $m=1/2$, $x=1/50$ and $\alpha=14$ it becomes

$$\begin{aligned}10\sqrt{2} &= 14(1-\frac{1}{50})^{-1/2} = 14F(\frac{1}{2},b,b;\frac{1}{50})\\ d_k &= d_{k-1}\frac{(1/2+k-1)}{k}\frac{1}{50} = d_{k-1}\frac{2k-1}{100k} < \frac{d_{k-1}}{50}.\end{aligned}$$

This is the justification of the following line used in the program to compute $\sqrt{2}$:

```
for(d=0,k=c;--k;d/=b,d*=2*k-1)f[k]=(d+=f[k]*a)%(b=100*k);
```

and the number of iteration is proportional to $\log(10)/\log(50) \approx 0.589$.

6 Computation of Euler constant γ

On april 1st of 2003, we receiced from Norihito Sasaki a wonderful tiny `C` program that computes digits of Euler constant. The program is 307 characters long and computes 1688 decimals of Euler constant :

```
int*d,*e,w[2850],p=475,b=10000,r=3,i,j,x;s(int k){d=w+k/14%4*p;
e=w+k/56*p;for(x=k&1,j=k/2%7*p;j--;k&1?*d=(x+=k-49?*e++-*d+b-1:
(r?r-1?4096:3888:4374)**d)%b,d++,x/=b:(x=x%i*b+e[j],d[j]=x/i));
}main(){for(;r<3&i++<33*p+!r?s(216),s(49),s(242),s(59),1:r?i=0,
w[5*p]=--r?r-1?23:84:60:printf("%.4d",w[--p])&&p>53;);}
```

The approach does not make use of the spigot-algorithm and thus, is of a different family compared to tiny codes presented in the previous sections. The formula used is

$$\gamma = a_1S(x_1) + a_2S(x_2) + a_3S(x_3) + O(e^{-\min\{x_1,x_2,x_3\}}) \tag{1}$$

where $S(x)$ denotes the series

$$S(x) = \sum_{n>0}(-1)^{n-1}\frac{x^n}{n!n}.$$

In the `C` code presented here, the parameters are defined by

$$x_1 = 4096 = 2^{12}, \quad x_2 = 3888 = 2^4\times 3^5, \quad x_3 = 4374 = 2\times 3^7$$

and

$$a_1 = -23, \quad a_2 = 84, \quad a_3 = -60.$$

The formula comes from the fact that for any $x > 0$,

$$\gamma = S(x) - \log(x) + O(e^{-x})$$

(formula used by Sweeney in 1963, see *The Euler constant* γ for more details), and (1) is obtained by a linear combination of this formula. The parameters a_i and x_i are rational numbers (or even integers) choosen so that

$$\sum_i a_i = 1 \quad \text{and} \quad \prod_i x_i^{a_i} = 1.$$

This property permits to have the logarithmic term vanishing, making (1) true.

To obtain more digits of γ with this program, modifications of the parameters (a_i) and (x_i) are possible. For example, to obtain 22797 digits of γ, one can choose

$$(x_1, x_2, x_3) = (52488, 55296, 59049) \quad \text{and} \quad (a_1, a_2, a_3) = (110, -30, -79).$$

A list of other possible choices for these parameters, together with a more precise description of the approach, are available in very nice *notes* by Norihito Sasaki.

7 Prime numbers

In may 2003, Pascal Sebah and Sébastien Dauby wrote two very short programs to generate consecutive prime numbers 2, 3, 5, 7, ... The first program is a very tiny `C` code of just 68 characters but is very slow (it just relies on the trivial definition of prime numbers : p is prime if it has no divisors i such $1 < i < p$)

```
main(){for(int p=1,i;++p<1e5;i>p?printf("%d ",p):0)for(i=2;p%i++;);}
```

A more decent version (in terms of speed) consists in a tiny implementation of the Erathostene sieve of just 119 characters that permits to list a significant number of primes :

```
main(){int a[100000],i=1e5,n=i,p=1;while(--i)a[i]=1;
while(++p<n)for(a[p]?printf("%d ",i=p):0;p<4e4&&i*p<n;a[i++*p]=0);}
```

8 Interesting tiny programs in other mathematical areas

The following interesting tiny C program (477 bytes) is from Fabrice Bellard and prints the biggest prime number that was known in the year 2000 : $2^{6972593} - 1$ (note : this is no longer the biggest known prime number). Fabrice Bellard won the *International Obfuscated C Code Contest* of Year 2000 with this code.

```
int m=754974721,N,t[1<<22],a,*p,i,e=4625195,j,s,b,c,U;f(d){for(s
=1<<23;s;s/=2,d=d*1LL*d%m)if(s<N)for(p=t;p<t+N;p+=s)for(i=s,c=1;
i;i--)b=*p+p[s],p[s]=(m+*p-p[s])*1LL*c%m,*p++=b%m,c=c*1LL*d%m;
for(j=0;i<N-1;){for(s=N/2;!((j^=s)&s);s/=2);if(++i<j)a=t[i],t[i]
=t[j],t[j]=a;}}main(){*t=2;U=N=1;while(e/=2){N*=2;U=U*1LL*(m+1)/
2%m;f(362);for(p=t;p<t+N;)*p++=*p*1LL**p%m*U%m;f(415027540);for(
a=0,p=t;p<t+N;)a+=*p<<(e&1),*p++=a%10,a/=10;}while(!*--p);t[0]--
;while(p>=t)printf("%d",*p--);}
```

It should be noted that this program uses integer FFT so that it can run in a reasonable time.

The following informations are quoted from Fabrice Bellard page :

I compiled it successfully with gcc with i86 Linux and Sparc Solaris. It takes about 9 minutes on a Pentium 120 and 2 minutes on a Pentium II/450. In order to compile it, your C compiler must support the 64 bit long long type.

This program basically converts from base 2 to base 10. It is a non trivial task because it deals with numbers of millions of digits. The usual method (with repeated divisions by 10^N*) would be far too slow. So I decided to use a Integer Fast Fourier Transform. I believe it is one of the smallest implementation of such an algorithm.*

More can be found on Fabrice Bellard *page*.

References

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, (1964)

[2] S. Rabinowitz, *A Spigot-Algorithm for* π, Abstract of the American Mathematical Society, (1991), vol. 12, p. 30