Spigot algorithms

Jeremy Gibbons
University of Oxford
PSC, November 2003

1. A short program for π

```
a[52514],b,c=52514,d,e,f=1e4,g,h;
main(){for(;b=c-=14;h=printf("%04d",e+d/f))
for(e=d%=f;g=--b*2;d/=g)
d=d*b+f*(h?a[b]:f/5),a[b]=d%--g;}
```

(Dik Winter and Achim Flammenkamp)

2. The same program, beautified

```
#define NDIGITS 15000          /* max. digits to compute  */
#define LEN (NDIGITS/4+1)*14 /* nec. array length   */

int a[LEN];                    /* array of 4 digit-decimals*/
int b;                         /* nominator prev. base    */
int c = LEN;                   /* index                   */
int d = 0;                     /* accumulator and carry   */
int e = 0;                     /* save prev. 4 digits     */
int f = 10000;                 /* new base, 4 dec. digits */
int g;                         /* denom prev. base        */
int h = 0;                     /* init switch             */
```

```
int main() {
   for ( ;(b=c-=14) > 0; )          /* outer loop:4 digits/loop*/
   {
     for (; --b > 0;    )          /* inner loop: radix conv  */
     {
       d *= b;                     /* acc *= nom. prev base   */
       if (h == 0)
          d += 2000 * f;           /* first outer loop        */
       else
          d += a[b] * f;           /* non-first outer loop    */
       g=b+b-1;                    /* denom prev. base        */
       a[b] = d % g;
       d /= g;                     /* save carry              */
     }
     h = printf("%04d", e + d/f);  /* print prev 4 digits     */
     d = e = d % f;                /* save current 4 digits   */
   }
   return 0;
}
```

3. Series for π

From Wallis' formula (1655):

$$\frac{2}{\pi} = \frac{1\times3}{2\times2} \times \frac{3\times5}{4\times4} \times \frac{5\times7}{6\times6} \times \frac{7\times9}{8\times8} \times \cdots$$

one can (apparently) derive:

$$\begin{aligned} \pi &= 2 + \tfrac{1}{3}(2 + \tfrac{2}{5}(2 + \tfrac{3}{7}(2 + \tfrac{4}{9}(2 + \cdots)))) \\ &= 3 + \tfrac{1}{10}(1 + \tfrac{1}{10}(4 + \tfrac{1}{10}(1 + \tfrac{1}{10}(5 + \cdots)))) \end{aligned}$$

In other words, π has a simple representation

$$2;2,2,2,2,\ldots$$

in a mixed-radix base $\mathcal{B}$, where the digits after the point have weights $\frac{1}{3}, \frac{2}{5}, \ldots$ (whereas the digits in decimal all have weight $\frac{1}{10}$).

The *spigot algorithm* converts this representation to decimal.

But it isn't *streaming*.

4. Spigot algorithm (Rabinowitz & Wagon, 1995)

To compute decimal digits of π:

- $\lfloor 10n/3 \rfloor + 1$ terms of series needed for n decimal digits
- start with $2; 2, 2, 2, 2, \ldots 2_{\mathcal{B}}$
- repeatedly: output integer part; scale fractional part by 10
- scale each term by 10
- normalize, so term in position $\frac{i}{2i+1}$ is at most $2i$: in position $\frac{p}{q}$, term is reduced modulo q, and quotient times p is carried to term to the left

Seems inherently bounded: must commit to number of terms, and hence to number of digits.

4.1. Working of spigot algorithm

	π	1	$\frac{1}{3}$	$\frac{2}{5}$	$\frac{3}{7}$	$\frac{4}{9}$	$\frac{5}{11}$	$\frac{6}{13}$	$\frac{7}{15}$	$\frac{8}{17}$	$\frac{9}{19}$	$\frac{10}{21}$	$\frac{11}{23}$	$\frac{12}{25}$
Init		2	2	2	2	2	2	2	2	2	2	2	2	2
Scale		20	20	20	20	20	20	20	20	20	20	20	20	20
Carry	3	10	12	12	12	10	12	7	8	9	0	0	0	–
Sum		30	32	32	32	30	32	27	28	29	20	20	20	20
Rem		0	2	2	4	3	10	1	13	12	1	20	20	20
Scale		0	20	20	40	30	100	10	130	120	10	200	200	200
Carry	1	13	20	33	40	65	48	98	88	72	150	132	96	–
Sum		13	40	53	80	95	148	108	218	192	160	332	296	200
Rem		3	1	3	3	5	5	4	8	5	8	17	20	0
Scale		30	10	30	30	50	50	40	80	50	80	170	200	0
Carry	4	11	24	30	40	40	42	63	64	90	120	88	0	–
Sum		41	34	60	70	90	92	103	144	140	200	258	200	0
Rem		1	1	0	0	0	4	12	9	4	10	6	16	0
Scale		10	10	0	0	0	40	120	90	40	100	60	160	0
Carry	1	4	2	9	24	55	84	63	48	72	60	66	0	–
Sum		14	12	9	24	55	124	183	138	112	160	126	160	0

4.2. Minor snag

Number $0;2,4,6,8,\ldots_{\mathcal{B}}$ represents 2.

Therefore, integer part of normalized $a_0;a_1,a_2,a_3,\ldots_{\mathcal{B}}$ is either a_0 or a_0+1, depending on whether $0;a_1,a_2,a_3,\ldots_{\mathcal{B}}$ is in $[0,1)$ or $[1,2)$.

So must buffer output digits, and occasionally increment them.
(First shows up at 32nd digit of π.)

4.3. Haskell implementation

```
> spigot :: Int -> [Int]
> spigot n = concat (loop n next ([],table n))
>   where
>     table n = [ (i,2) | i <- [1..(10*n) `div` 3 + 1] ]
>     next (predigits, table) = case cq of
>       10 -> (map ((`mod`10).(+1)) predigits, ([0],table'))
>       9  -> ([], (predigits++[9], table'))
>       otherwise -> (predigits, ([cq], table'))
>       where
>         table1 = [ (i,10*d) | (i,d) <- table ]
>         (c,(1,0):table2) = scan red (table1,0)
>         (cq,cr) = c `divMod` 10
>         table' = (1,cr) : table2
>     red ((i,d),c) = (q,(i,r))
>       where (q,r) = (d+c*i) `divMod` (2*i-1)
```

4.4. Auxilliaries

```
> loop :: Int -> (a->(b,a)) -> (a->[b])
> loop 0 f x = []
> loop (n+1) f x
>   = let (a,y) = f x
>     in a : loop n f y

> scan :: ((a,b)->(b,c)) -> ([a],b) -> (b,[c])
> scan op ([],b) = (b,[])
> scan op (a:as,b)
>   = let (b',cs) = scan op (as,b)
>         (b'',c) = op (a,b')
>     in (b'',c:cs)
```

4.5. The obfuscated C program

```
a[52514],b,c=52514,d,e,f=1e4,g,h;
main(){for(;b=c-=14;h=printf("%04d",e+d/f))
for(e=d%=f;g=--b*2;d/=g)
d=d*b+f*(h?a[b]:f/5),a[b]=d%--g;}
```

- conversion to base 10000, not base 10
- buffer of exactly one superdigit suffices up to 54,936 digits of π
- $\lfloor 4 \times 10/3 + 1 \rfloor = 14$ terms may be discarded for each superdigit
- $52514 = (15000/4 + 1) \times 14$

5. A streaming program

$$
\begin{aligned}
3 &= 2 + \tfrac{1}{3}(2 + \tfrac{1}{3}(2 + \tfrac{1}{3}(2 + \tfrac{1}{3}(2 + \cdots)))) \\
&< 2 + \tfrac{1}{3}(2 + \tfrac{2}{5}(2 + \tfrac{3}{7}(2 + \tfrac{4}{9}(2 + \cdots)))) &&= \pi \\
&= \begin{pmatrix} 1 & 6 \\ 0 & 3 \end{pmatrix}\begin{pmatrix} 2 & 10 \\ 0 & 5 \end{pmatrix}\begin{pmatrix} 3 & 14 \\ 0 & 7 \end{pmatrix}\cdots\begin{pmatrix} k & 4k+2 \\ 0 & 2k+1 \end{pmatrix}\cdots \\
&< 2 + \tfrac{1}{2}(2 + \tfrac{1}{2}(2 + \tfrac{1}{2}(2 + \tfrac{1}{2}(2 + \cdots)))) &&= 4
\end{aligned}
$$

```
> spigot = f (1,0,0,1) [(k, 4*k+2, 0, 2*k+1) | k<-[1..]] where
>   f (q,r,s,t) xs@((q',r',s',t'):ys)
>     | safe (q,r,s,t) n = n : f (10*(q-n*s),10*(r-n*t),s,t) xs
>     | otherwise = f (q*q'+r*s',q*r'+r*t',s*q'+t*s',s*r'+t*t') ys
>     where n = (2*q+r) `div` (2*s+t)
>   safe (q,r,s,t) n = (4*s+t>0) && (4*q+r) `div` (4*s+t) == n
```

But it isn't as fast, or as short!

6. A shapely program (Roemer Lievaart)

```
                                                                    char
                                                                _3141592654[3141
        ],__3141[3141];_314159[31415],_3141[31415];main(){register char*
     _3_141,*_3_1415, *_3__1415; register int _314,_31415,__31415,*_31,
   _3_14159,__3_1415;*_3141592654=__31415=2,_3141592654[0][_3141592654
  -1]=1[__3141]=5;__3_1415=1;do{_3_14159=_314=0,__31415++;for( _31415
 =0;_31415<(3,14-4)*__31415;_31415++)_31415[_3141]=_314159[_31415]= -
1;_3141[*_314159=_3_14159]=_314;_3_141=_3141592654+__3_1415;_3_1415=
__3_1415    +__3141;for               (_31415 = 3141-
          __3_1415  ;                 _31415;_31415--
          ,_3_141 ++,                 _3_1415++){_314
          +=_314<<2 ;                 _314<<=1;_314+=
         *_3_1415;_31                 =_314159+_314;
         if(!(*_31+1)                 )* _31 =_314 /
         __31415,_314                 [_3141]=_314 %
         __31415 ;* (                 _3__1415=_3_141
        )+= *_3_1415                   = *_31;while(*
        _3__1415 >=                    31415/3141 ) *
        _3__1415+= -                   10,(*--_3__1415
       )++;_314=_314                   [_3141]; if ( !
       _3_14159 && *                   _3_1415)_3_14159
       =1,__3_1415 =                   3141-_31415;}if(
       _314+(__31415                    >>1)>=__31415 )
       while ( ++ *                     _3_141==3141/314
      )*_3_141--=0                      ;}while(_3_14159
      ) ; { char *                      __3_14= "3.1415";
      write((3,1),                      (--*__3_14,__3_14
      ),(_3_14159                        ++,++_3_14159))+
     3.1415926; }                        for ( _31415 = 1;
    _31415<3141-                         1;_31415++)write(
   31415% 314-(                          3,14),_3141592654[
  _31415    ] +                         "0123456789","314"
  [ 3]+1)-_314;                         puts((*_3141592654=0
,_3141592654))                           ;_314= *"3.141592";}
```