

CALCULATING TO HUNDREDS OF DIGITS OF ACCURACY

BY MICHAEL E. TAYLOR

Contents

1. Arithmetic to hundreds of digits.
2. The file **arithmetic.pas**.
3. Approximation of e .
4. Square roots.
5. 200 digits of pi.
6. Long multiplication and fast multiplication.
7. Division and square roots via Newton's method.

1. Doing arithmetic with hundreds of digits

One can store a representation of a real number to a large number of decimal places, as an array of integers, which we will identify as a variable of type ‘digits.’ The following program asks one for an integer n and prints out $1/n$, to 198 places after the decimal point. It is reliable for $1 \leq n \leq 327$.

```

program oneovern;
(* reliable for n<328 *)
uses
  crt;
type
  digits=array[1..100] of integer;
var
  n,j:integer;
  a,q:digits;
  ans:char;
{$I arithmetic.pas}

begin
  a[1]:=1;
  for j:=2 to 100 do a[j]:=0;
  repeat
    writeln('Specify an integer. ');
    readln(n);
    writeln('One over ', n, ' is: ');
    divide(a,n,q);
    write(q[1], '. ');
    for j:=2 to 100 do
      begin
        if q[j]<10 then write(0);
        write(q[j]);
      end;
    writeln;
    writeln('Do it again? (Y/N) ');
    readln(ans);
  until upcase(ans)='N'
end.

```

The statements specifying $a[j]$ represent $1.000000\dots$ as an array

01 00 00 00...

of 100 two-digit integers. It is up to the programmer to keep track of where the decimal point should be. In another context, this array might represent the huge integer 10^{198} . The division $1/n$ is performed by

`divide(a,n,q);`

The procedure **divide** in the file **arithmetic.pas** is the following.

```
(* arithmetic.pas *)

procedure divide(var numer:digits;denom:integer;var quot:digits);
var
  jjj,rrr,mmm:integer;
begin
  rrr:=0;
  for jjj:=1 to 100 do
  begin
    mmm:=rrr*100+numer[jjj];
    quot[jjj]:=mmm div denom;
    rrr:=mmm-denom*quot[jjj];
  end;
end;
```

The division is carried out much as one learns to do it in the third grade, except that we work with 2 digits at a time instead of just one. Thus, if $n = 11$, the procedure **divide** performs the following calculation:

$$\begin{array}{r} 0.09\ 09\ \dots \\ 11 \overline{)01.00\ 00\ 00\ 00} \\ \underline{0} \\ 1\ 00 \\ \underline{99} \\ 1\ 00 \\ \underline{99} \end{array}$$

Exercise. Make a collection of procedures, to be included in the file **arithmetic.pas**, performing the following basic operations, in addition to the procedure **divide** above.

$$\begin{array}{ll} b = na & \text{multiply}(a,n,b) \\ c = a + b & \text{sum}(a,b,c) \\ c = a - b & \text{difference}(a,b,c) \end{array}$$

Here, n is an integer, $1 \leq n \leq 327$, and a, b, c are variables of the type ‘digits’. An answer is given in the next section, so don’t peek.

The next program computes $n!$, given an integer n , $1 \leq n \leq 120$. In this case, an array of digits is regarded as an integer, less than or equal to $10^{200} - 1 = 99 \dots 9999$.

```
program factorial;
(* reliable for n<121 *)
uses
  crt,printer;
type
```

```
    digits=array[1..100] of integer;
var
    n,j:integer;
    a:digits;
    ans:char;
    {$I arithmetic.pas}

begin
    repeat
        writeln('Specify an integer. ');
        readln(n);
        writeln(n, ' factorial is: ');
        a[100]:=1;
        for j:=1 to 99 do a[j]:=0;
        for j:=1 to n do multiply(a,j,a);
        for j:=1 to 100 do
            begin
                if a[j]<10 then write(0);
                write(a[j]);
            end;
        writeln;
        writeln('Do it again? (Y/N)');
        readln(ans);
    until upcase(ans)='N'
end.
```

Note that the program uses the procedure **multiply**, which was requested in the last exercise.

2. The file arithmetic.pas

The following file consists of procedures which allow one to perform a number of basic operations on 200 digit numbers.

```
(* arithmetic.pas *)

procedure sum(terma:digits;termb:digits;var total:digits);
var
  mmm,ccc,kkk,jjj:integer;
begin
  ccc:=0;
  for jjj:=0 to 99 do
  begin
    kkk:=100-jjj;
    mmm:=terma[kkk]+termb[kkk]+ccc;
    ccc:=mmm div 100;
    total[kkk]:=mmm-ccc*100;
  end;
end;

procedure difference(terma:digits;termb:digits;var diff:digits);
var
  jjj:integer;
  uuu:digits;
begin
  for jjj:=1 to 100 do uuu[jjj]:=99-termb[jjj];
  sum(terma,uuu,diff);
  for jjj:=1 to 99 do uuu[jjj]:=0;
  uuu[100]:=1;
  sum(diff,uuu,diff);
end;

procedure multiply(factor:digits;nnn:integer;var prod:digits);
var
  mmm,ccc,kkk,jjj:integer;
begin
  ccc:=0;
  for jjj:=0 to 99 do
  begin
    kkk:=100-jjj;
    mmm:=nnn*factor[kkk]+ccc;
    ccc:=mmm div 100;
    prod[kkk]:=mmm-ccc*100;
  end;
end;
```

```
procedure divide(var numer:digits;denom:integer;var quot:digits);
var
    jjj,rrr,mmm:integer;
begin
    rrr:=0;
    for jjj:=1 to 100 do
        begin
            mmm:=rrr*100+numer[jjj];
            quot[jjj]:=mmm div denom;
            rrr:=mmm-denom*quot[jjj];
        end;
    end;

procedure tail(inn:integer;var out:digits);
{must have inn<10000}
var
    jjj,kkk:integer;
begin
    for jjj:=1 to 100 do out[jjj]:=0;
    kkk:=inn div 100;
    out[99]:=kkk;
    kkk:=inn-100*kkk;
    out[100]:=kkk;
end;

procedure head(inn:integer;var out:digits);
{must have inn<10000}
var
    jjj,kkk:integer;
begin
    for jjj:=1 to 100 do out[jjj]:=0;
    kkk:=inn div 100;
    out[1]:=kkk;
    kkk:=inn-100*kkk;
    out[2]:=kkk;
end;

procedure same(var aaa:digits;var bbb:digits);
var
    jjj:integer;
begin
    for jjj:=1 to 100 do bbb[jjj]:=aaa[jjj];
end;

procedure show(var a:digits);
var
    jjj:integer;
begin
    for jjj:=1 to 100 do
```

```

begin
  if(a[jjj]<=9) then write(0);
  write(a[jjj]);
end;
end;

procedure prtshow(aaa:digits;n:integer);
var
  jjj:integer;
begin
  for jjj:=1 to 100 do
  begin
    if (aaa[jjj]<=9) then write(1st, 0);
    write(1st,aaa[jjj]);
    if ((jjj>n) and ((jjj+n) mod 40=0)) then writeln(1st, '');
  end;
end;

procedure bigdiv(var numer:digits;denom:integer;var quot:digits);
var
  jjj:integer;
  rrr,mmm,nnn,qqq:longint;
begin
  rrr:=0;
  for jjj:=1 to 100 do
  begin
    nnn:=numer[jjj];
    mmm:=rrr*100+nnn;
    qqq:=mmm div denom;
    quot[jjj]:=qqq;
    rrr:=mmm-denom*qqq;
  end;
end;

procedure bigmult(factor:digits;nnn:integer;var prod:digits);
var
  jjj,kkk:integer;
  mmm,ccc,ppp,aaa:longint;
begin
  ccc:=0;
  for jjj:=0 to 99 do
  begin
    kkk:=100-jjj;
    aaa:=factor[kkk];
    mmm:=nnn*aaa+ccc;
    ccc:=mmm div 100;
    ppp:=mmm-ccc*100;
    prod[kkk]:=ppp;
  end;
end;

```

end;

We will discuss each of the procedures above, in order. The procedure **sum** performs addition just as one is taught in grade school, from right to left, with carrying, except it does it two digits at a time. We illustrate this with the following example.

$$\begin{array}{r} 1 \quad 1 \\ \dots 22 \quad 37 \quad 46 \\ \underline{\dots 11 \quad 92 \quad 65} \\ \dots 34 \quad 30 \quad 11 \end{array}$$

For $jjj = 0$ in the **for...do** loop, we compute $terma[100] + termb[100] = 46 + 65 = 111$. Then we set $ccc = 111 \text{ div } 100 = 1$ and $total[100] = 11$, and proceed to $jjj = 1$.

One could write a **difference** procedure that makes use of borrowing. Instead, we first compute $99 \dots 99 - termb$, which involves no borrowing, then add $terma$, and then add $00 \dots 01$. Note that the **sum** procedure gives

$$99 \dots 99 + 00 \dots 01 = 00 \dots 00.$$

The procedure **multiply** also does multiplication the grade school way, from right to left, with carrying, but it computes $factor \times nnn = prod$ by working on $factor$ two digits at a time. An example, with $nnn = 124$, looks like this:

$$\begin{array}{r} 46 \quad 57 \\ \dots 22 \quad 37 \quad 46 \\ \underline{\hspace{10em} 124} \\ \dots 74 \quad 45 \quad 04 \end{array}$$

For $jjj = 0$ in the **for...do** loop, we compute $factor[100] \times nnn = 46 \times 124 = 5704$. Thus we obtain $ccc = 57$ and $prod[100] = 04$, and proceed to $jjj = 1$.

The procedure **divide** was described in §1. As remarked there, it is reliable only for denominators ≤ 327 . If the denominator exceeds 327, then an intermediate step can yield an integer $> 2^{15} - 1 = 32,767$. There is a procedure we call **bigdiv**, described a little further down, which allows for a larger denominator.

The procedure **tail** takes a 4-digit integer inn , and produces a number, out , of type digits, such that $out[jjj] = 0$ for $1 \leq jjj \leq 98$, $out[99]$ consists of the first two digits of inn , and $out[100]$ consists of the last two digits of inn . For example,

$$\begin{aligned} inn = 4532 &\implies out = 00 \ 00 \ \dots \ 00 \ 45 \ 32 \\ inn = 17 &\implies out = 00 \ 00 \ \dots \ 00 \ 00 \ 17 \end{aligned}$$

The procedure **head** takes a 4-digit integer inn , and produces a number, out , of type digits, such that $out[1]$ consists of the first two digits of inn , $out[2]$ consists of the last two digits of inn , and $out[jjj] = 0$ for $3 \leq jjj \leq 100$. For example,

$$\begin{aligned} inn = 4500 &\implies out = 45 \ 00 \ 00 \ \dots \ 00 \ 00 \\ inn = 45 &\implies out = 00 \ 45 \ 00 \ \dots \ 00 \ 00 \end{aligned}$$

The procedure **same** takes a number aaa , of type `digits`, and produces another number bbb , of type `digits`, such that $bbb = aaa$.

The procedure **show** prints a number a of type `digits` on the screen. The procedure **prtshow** prints such a number a on paper, using a printer. More precisely, the command `prtshow(a,n)` prints a , with a carriage return after $40k - n$ two-digit entries of the array a have been printed, for $k = 1, 2, \dots$. Here, n must be < 40 .

The procedure **bigdiv** has a structure similar to that of **divide**, discussed in §1, but intermediate calculations are performed with integers of type `longint`, so that the denominator can be any integer between 1 and 32,767.

Likewise, the procedure **bigmult** has a structure similar to that of **multiply**, discussed above, but using integers of the type `longint` for intermediate calculations, so you can multiply a number $factor$, of type `digits`, by an integer nnn between 1 and 32,767.

Exercises.

1. Write a program to compute the binomial coefficient $\binom{n}{m}$, when $0 \leq m \leq n \leq 327$. Recall that

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \frac{n(n-1)\cdots(n-m+1)}{m(m-1)\cdots(1)}.$$

2. Compute the number of possible bridge hands, i.e., the number of partitions of 52 cards into 4 ordered piles (labeled north, east, south, and west) of 13 cards each. Hint. Write this number as a product of binomial coefficients.

You should get a 29-digit number.

3. Write a program to compute k^n , for $1 \leq k \leq 327$, provided n is in the range so that $k^n < 10^{200}$.

3. Approximation of e

The exponential function is given by

$$(1) \quad e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$

In particular, setting $x = 1$ gives

$$(2) \quad e = \sum_{n=0}^{\infty} \frac{1}{n!}.$$

If we sum over $0 \leq n \leq 150$, the error will be less than 10^{-200} ; that is what is done in the following program.

```

program napier;
uses
  crt,printer;
type
  digits=array[1..100] of integer;
var
  a,b :digits;
  j,n :integer;
{$I arithmetic.pas}

begin
  a[1]:=1;
  for j:=2 to 100 do a[j]:=0;
  same(a,b);
  sum(a,b,a);
  for n:=2 to 150 do
  begin
    divide(b,n,b);
    sum(a,b,a);
  end;
  writeln;
  write(a[1],'.');
  for j:=2 to 100 do
  begin
    if(a[j]<10) then write(0);
    write(a[j]);
  end;
  readln;
end.

```

It is useful to know for sure just how close an approximation to e is a given partial sum of the infinite series (2). The following simple lemma gives an adequate estimate.

Lemma. *For any $k > 0$,*

$$(3) \quad \sum_{n>k} \frac{1}{n!} < \frac{2}{(k+1)!}.$$

Proof. Note that, if $j > 0$, then $(k+j)! = (k+1)! \{(k+2) \cdots (k+j)\} \geq (k+1)! \cdot 2^{j-1}$, with strict inequality if $j > 1$. Hence the left side of (3) is less than

$$(4) \quad \frac{1}{(k+1)!} \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \right),$$

which sums to the right side of (3).

A use of the program **factorial** shows that

$$120! > 6 \times 10^{198}.$$

Hence $121! > 6 \times 10^{200}$. Therefore we can conclude that

$$0 < e - \sum_{n=0}^{120} \frac{1}{n!} < 10^{-200}.$$

Hence, in the program above, it would suffice to sum over $0 \leq n \leq 120$.

We now give another application of the estimate (3), to the following qualitative fact about the number e .

Proposition. *The number e is irrational.*

Proof. The content of the proposition is that it is not possible to write

$$(5) \quad e = \frac{m}{k},$$

for any two integers m and k . Indeed, if an identity of the form (5) were valid, we would have

$$0 < \frac{m}{k} - \sum_{n=0}^k \frac{1}{n!} < \frac{2}{(k+1)!}$$

and if we multiply by $k!$ we get

$$0 < m(k-1)! - \sum_{n=0}^k \frac{k!}{n!} < \frac{2}{k+1}.$$

This is a set of two inequalities among three numbers, the middle of which is an integer. Since the last number $2/(k+1)$ is less than one, this is not possible; we have contradicted the hypothesis that (5) holds.

4. Square roots

The program **sqrt** listed below gives a 200-digit approximation to \sqrt{k} , when k is an integer between 2 and 99. In order to approximate \sqrt{k} to high precision, we first use the 'sqrt' function to find an approximation y to \sqrt{k} . Provided $2 \leq k < 100$, we have $1 < y < 10$. We then set

$$(1) \quad j = [10y] + 1, \quad n = j^2, \quad m = n - 100k.$$

Here, $[x]$ is the greatest integer $\leq x$. Note that $10y < j \leq 10y + 1$, so (if we neglect the small difference between y^2 and k)

$$(2) \quad 100k < n \leq 100k + 20\sqrt{k} + 1, \quad 0 < m \leq 20\sqrt{k} + 1.$$

Now, we write \sqrt{k} in the form

$$(3) \quad \sqrt{k} = \sqrt{\frac{j^2}{100} + k - \frac{j^2}{100}} = \frac{j}{10} \sqrt{1 - \frac{m}{n}}.$$

Note that, by (2),

$$(4) \quad 0 < \frac{m}{n} < \frac{20\sqrt{k} + 1}{100k} = \frac{1}{5\sqrt{k}} + \frac{1}{100k} < \frac{1}{7}.$$

This makes it tempting to evaluate $\sqrt{1-x}$ at $x = \frac{m}{n}$ by a power series expansion about $x = 0$.

This is what is done in the following program. After the line `m:=n-100*k`; is executed, the integers j, m , and n defined above have been computed. We will describe the rest of the program after listing it.

```

program sqrt;
uses
  crt,printer;
type
  digits=array[1..100] of integer;
var
  s,a,b,c:digits;
  i,j,k,m,n:integer;
  x,y,z:double;
  ch:char;
  {$I arithmetic.pas}

begin
  writeln;
  writeln('We will compute the square root of k. ');
  writeln('Specify an integer k, between 1 and 100. ');
  read(k);
  x:=k;

```

```

y:=sqrt(x);
writeln;
writeln(y);
z:=10*y;
j:=trunc(z)+1;
n:=j*j;
m:=n-100*k;

head(100,c);
same(c,s);
bigdiv(c,n,c);
divide(c,2,c);
multiply(c,m,c);
difference(s,c,s);
for i:=2 to 250 do
begin
  bigdiv(c,n,c);
  multiply(c,m,c);
  divide(c,i,a);
  divide(a,2,a);
  multiply(a,3,a);
  difference(c,a,c);
  difference(s,c,s);
end;
divide(s,10,s);
multiply(s,j,s);
writeln;
write(s[1], '.');
for i:=2 to 100 do
begin
  if (s[i]<10) then write(0);
  write(s[i]);
end;
ch:=readkey;
end.

```

The line `head(100,c)`; begins the part of the program making use of numbers of type digits. Here, a number a of type digits represents a real number $r \in [0, 100)$ as illustrated:

$$a = 01\ 26\ 35 \ \cdots\ 22\ 94 \Leftrightarrow r = 1.2635 \cdots 2294$$

$$a = 73\ 01\ 35 \ \cdots\ 22\ 94 \Leftrightarrow r = 73.0135 \cdots 2294$$

Thus `head(100,a)` has a represent the number 1.0 . As mentioned above, we calculate $\sqrt{1-x}$, with $x = \frac{m}{n}$, via the power series

$$(5) \quad \sqrt{1-x} = 1 - \sum_{i=1}^{\infty} b_i x^i.$$

Here, with $f(x) = (1+x)^{\frac{1}{2}}$, we have $b_j = (-1)^{j-1} f^{(j)}(0)/j!$, for $j \geq 1$. Consequently,

$$(6) \quad b_1 = \frac{1}{2}, \quad b_2 = \frac{1}{8}, \quad b_3 = \frac{1}{16}, \dots$$

There is the inductive formula

$$(7) \quad b_{j+1} = b_j - \frac{3}{2} \frac{1}{j+1} b_j, \quad j \geq 1.$$

Equivalently, we have

$$(8) \quad \sqrt{1 - \frac{m}{n}} = 1 - \sum_{i=1}^{\infty} c_i$$

where

$$(9) \quad c_1 = \frac{1}{2} \frac{m}{n}, \quad c_{j+1} = \frac{m}{n} c_j - \frac{3}{2} \frac{1}{j+1} \frac{m}{n} c_j.$$

This last induction is implemented within the **for...do** loop, in which $2 \leq i \leq 250$. Note that, according to (2), n can exceed 327, so we use **bigdiv** to divide by n .

We now consider how many terms in the sum (5) are needed to guarantee 200 digit accuracy of the resulting partial sum. It is clear from (7) that b_j is monotonically decreasing; note also that $b_7 = \frac{11}{14} \cdot \frac{105}{5120} < \frac{1}{50}$. Since, by (4), we have $x = \frac{m}{n} \in (0, \frac{1}{7})$, we deduce that

$$(10) \quad 0 < b_j < \frac{1}{50} \frac{1}{7^j} \quad \text{for } j \geq 7.$$

It follows that

$$(11) \quad 0 < \sum_{j>k} b_j x^j < \frac{1}{300} \frac{1}{7^k},$$

or equivalently

$$(12) \quad 0 < \sqrt{1-x} - \left(1 - \sum_{i=1}^k b_i x^i\right) < \frac{1}{300} \cdot \frac{1}{7^k},$$

provided $k \geq 6$. Thus, to insure that the error is $< 10^{-200}$, it suffices to pick k large enough that $10^{198} < 3 \cdot 7^k$. Now, if exercise 3 of §2 has been done, you will find that

$$7^{234} > 5 \cdot 10^{197},$$

so $3 \cdot 7^{234} > 10^{198}$. Thus it suffices to sum (4) over $1 \leq i \leq 234$. Just to pick a round number, we chose $i = 250$ for the upper limit of the sum. Note that, when dividing c by such i , in **divide(c,i,a)**, we do not need **bigdiv**.

Exercises.

1. Modify the program **sqroot**, replacing (1) by

$$j = [100y] + 1, \quad n = j^2, \quad m = n - 10000k,$$

so (3) is replaced by

$$\sqrt{k} = \frac{j}{100} \sqrt{1 - \frac{m}{n}}.$$

This time you'll need **bigmult** as well as **bigdiv**. How many terms in the power series expansion does it take in this case to achieve 200 digit accuracy? Is the modified program faster than the original?

5. 200 digits of pi

One approach to the accurate approximation of π , which has been popular since the time of Newton, is to express π in terms of an inverse trigonometric function, which is evaluated by summing a power series. For example, as can be seen by bisecting one angle of an equilateral triangle, we have

$$(1) \quad \frac{\pi}{6} = \sin^{-1} \frac{1}{2}.$$

We can find the power series of $\sin^{-1} x$ via the formula

$$(2) \quad \sin^{-1} x = \int_0^x \frac{ds}{\sqrt{1-s^2}}.$$

We write $(1-s^2)^{-\frac{1}{2}}$ as a power series

$$(3) \quad (1-s^2)^{-\frac{1}{2}} = \sum_{n=0}^{\infty} a_n s^{2n}$$

and integrate term by term, to obtain

$$(4) \quad \sin^{-1} x = \sum_{n=0}^{\infty} \frac{a_n}{2n+1} x^{2n+1}.$$

In these formulas, $a_n = f^{(n)}(0)/n!$, with $f(y) = (1-y)^{-\frac{1}{2}}$. We have the recursive formula

$$(5) \quad a_0 = 1, \quad a_{n+1} = \frac{1}{n+1} \frac{2n+1}{2} a_n.$$

Thus, with $b_n = a_n/(2n+1)$, we have

$$(6) \quad b_0 = 1, \quad b_{n+1} = \frac{(2n+1)^2}{(2n+2)(2n+3)} b_n,$$

and

$$(7) \quad \frac{\pi}{6} = \sum_{n=0}^{\infty} b_n \left(\frac{1}{2}\right)^{2n+1}.$$

Alternatively,

$$(8) \quad \frac{\pi}{6} = \sum_{n=0}^{\infty} c_n, \quad c_0 = \frac{1}{2}, \quad c_n = \frac{(2n-1)^2}{8n(2n+1)} c_{n-1}.$$

To see how far we should take the sum (8), we find k such that

$$(9) \quad 6 \cdot \sum_{n>k} c_n < 10^{-200}.$$

Since b_n is monotonically decreasing and $b_0 = 1$, we have

$$(10) \quad \sum_{n>k} c_n < \sum_{n>k} \left(\frac{1}{2}\right)^{2n+1} = \frac{1}{3} \cdot 4^{-k}.$$

Thus it suffices to take k so that

$$4^k > 2 \cdot 10^{200}.$$

One can verify that any $k \geq 334$ will work.

```

program pidigits;
uses
  crt,printer;
type
  digits=array[1..100] of integer;
var
  pie,a,c:digits;
  j,n:integer;
{$I arithmetic.pas}

begin
  head(50,a);
  same(a,c);
  for n:=1 to 350 do
  begin
    bigdiv(c,8*n,c);
    bigmult(c,2*n-1,c);
    bigdiv(c,2*n+1,c);
    bigmult(c,2*n-1,c);
    sum(a,c,a);
  end;
  multiply(a,6,pie);
  writeln;
  write(pie[1], '.');
  for j:=2 to 100 do
  begin
    if (pie[j]<10) then write(0);
    write(pie[j]);
  end;
  readln;
end.

```

There are infinite series for π that converge a bit more rapidly, obtained by expressing π in terms of the arctangent, which is given by

$$(11) \quad \tan^{-1}x = \int_0^x \frac{ds}{1+s^2} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$$

They are more subtle than the formula (1); some are obtained using the identities

$$(12) \quad \tan 2z = \frac{2 \tan z}{1 - \tan^2 z}, \quad \tan 4z = \frac{2 \tan 2z}{1 - \tan^2 2z}.$$

In particular, if $\tan z = \frac{1}{5}$, then $\tan 2z = \frac{5}{12}$ and $\tan 4z = \frac{120}{119}$, which happens to be very close to 1. It follows that $\tan(4z - \frac{\pi}{4}) = \frac{1}{239}$, and this is equivalent to the formula

$$(13) \quad \pi = 16 \tan^{-1} \frac{1}{5} - 4 \tan^{-1} \frac{1}{239}.$$

Now the series (11) converges somewhat faster at $x = 1/5$ than the series (4) does at $x = 1/2$, and the series (11) at $x = 1/239$ converges quite fast indeed, by comparison.

Euler discovered an even faster series for the arctangent, namely

$$(14) \quad \tan^{-1} x = \frac{y}{x} \left(1 + \frac{2}{3}y + \frac{2 \cdot 4}{3 \cdot 5}y^2 + \frac{2 \cdot 4 \cdot 6}{3 \cdot 5 \cdot 7}y^3 + \cdots \right), \quad y = \frac{x^2}{1 + x^2}.$$

Euler also discovered the representation

$$(15) \quad \pi = 20 \tan^{-1} \frac{1}{7} + 8 \tan^{-1} \frac{3}{79}.$$

A more recent representation, discovered this century, is

$$(16) \quad \pi = 24 \tan^{-1} \frac{1}{8} + 8 \tan^{-1} \frac{1}{57} + 4 \tan^{-1} \frac{1}{239}.$$

More on this can be found in [Bk]. More exotic, and more powerful, methods of approximating π are discussed in [AB] and [BB].

Exercise. Write a program to compute π to 200 digits, using (13). How many terms in each series for $\tan^{-1}x$, $x = \frac{1}{5}$ or $\frac{1}{239}$, must you take? Is the new program faster than the one described above?

6. Long multiplication and fast multiplication

The procedure **multiply** discussed in §2 allows one to multiply a 200 digit number a by an integer j between 1 and 327, and **bigmult** allows one to take j between 0 and 32,767. The procedure **longmult**, given below, allows one to multiply two 100 digit integers, fact1 and fact2, obtaining a 200 digit integer prod.

```

procedure longmult(fact1:digits;fact2:digits;var prod:digits);
var
  nnn,jjj,kkk:integer;
  ccc:digits;
begin
  multiply (fact1,fact2[100],prod);
  for nnn:=1 to 49 do
  begin
    kkk:=100-nnn;
    multiply(fact1,fact2[kkk],ccc);
    for jjj:=50-nnn to kkk do ccc[jjj] := ccc[jjj+nnn];
    for jjj:=kkk+1 to 100 do ccc[jjj] := 0;
    sum(prod,ccc,prod);
  end;
end;

```

Note that this procedure calls **multiply** 50 times, and hence uses 10,000 integer multiplications, as well as 5000 integer divisions and a comparable number of additions and subtractions. It takes 0.19 seconds for this procedure to be executed on the 16 Mhz IBM 386SX machines in Hanes 02. Generally, to multiply two n -digit integers using such a procedure would require $\sim C \cdot n^2$ arithmetical operations. One can cut down this number substantially, when n is very large, using a technique which exploits the Fast Fourier Transform. The FFT is discussed in some detail in the following chapter.

Suppose n is a power of 2; $n = 2^k$. We want to multiply two n digit numbers a and b , and obtain a $2n$ -digit integer c . Write

$$(1) \quad \begin{aligned} a &= a_0 + a_1 \cdot 10 + \cdots + a_{n-1} \cdot 10^{n-1}, \\ b &= b_0 + b_1 \cdot 10 + \cdots + b_{n-1} \cdot 10^{n-1}, \end{aligned}$$

where a_j and b_j are between 0 and 9. Now consider the expressions

$$(2) \quad \begin{aligned} A(\omega) &= a_0 + a_1\omega + \cdots + a_{n-1}\omega^{n-1}, \\ B(\omega) &= b_0 + b_1\omega + \cdots + b_{n-1}\omega^{n-1}. \end{aligned}$$

We can regard these as polynomials in ω ; alternatively, taking $\omega = e^{2\pi i/2n}$, we can regard them as Fourier series, with Fourier coefficients given by a_j and b_j , respectively, for $0 \leq j \leq n-1$, and vanishing Fourier coefficients for $n \leq j \leq 2n-1$.

Using the FFT, we can evaluate $A(\omega)$ and $B(\omega)$, as functions on the cyclic group Γ_{2n} generated by ω , using $\sim C(\log_2 n)n$ additions and multiplications. Then we can evaluate

$$(3) \quad C(\omega) = A(\omega)B(\omega),$$

as a function on Γ_{2n} , using $2n$ further multiplications of complex numbers, i.e., $4n$ multiplications and $4n$ additions of real numbers. Then, again via the FFT, using $\sim C(\log_2 n)n$ such operations, we can evaluate the Fourier coefficients c'_j in

$$(4) \quad C(\omega) = c'_0 + c'_1\omega + \cdots + c'_{2n-1}\omega^{2n-1}.$$

This is almost the decimal expansion

$$(5) \quad ab = c = c_0 + c_1 \cdot 10 + \cdots + c_{2n-1} \cdot 10^{2n-1},$$

where $0 \leq c_j \leq 9$. In fact, the c'_j are integers ≥ 0 , but they may exceed 9. To obtain the c_j , you let

$$(6) \quad c_0 = c'_0 \bmod 10,$$

replace c'_1 by $c'_1 + (c'_0 - c_0)/10$, let c_1 be the residue mod 10 of this quantity, and continue. This takes $\sim Cn$ further arithmetical operations.

There is one further wrinkle. To implement the FFT, with $\omega = e^{\pi i/n} = \cos \pi/n + i \sin \pi/n$, one typically uses floating point operations. Since the coefficients c'_j in (4) are known a priori to be integers, satisfying $0 \leq c'_j < 100n$, it suffices to do such calculations to a precision of $\sim C \log n$ digits. If one does this 'naively' (in $\sim C(\log n)^2$ steps), the total number of basic arithmetical operations required to compute ab is $\sim Cn(\log_2 n)^3$. If n is so large that $\log n$ is also 'large,' one might use an inductive procedure, reducing $Cn(\log_2 n)^3$ a bit.

Exercises.

1. Write a program to implement the fast multiplication described above, when $n = 128$, making use of a 256-point FFT.

7. Division and square roots via Newton's method

Generally, Newton's method for finding x such that $f(x) = y$ proceeds from an approximation x_j to x_{j+1} , given by

$$(1) \quad x_{j+1} = x_j + \frac{y - f(x_j)}{f'(x_j)}.$$

When $f(x) = 1/x$, a 'miracle' occurs; the sequence of approximations x_j to $1/y$ is given inductively by

$$(2) \quad x_{j+1} = x_j(2 - x_j y),$$

which involves one subtraction and two multiplications! It is not hard to show that, if $0 < x_1 < 1/y$, then $x_j \nearrow 1/y$.

Note that, if one needs $1/y$ to n digits accuracy, it suffices (roughly) to have x_j to $n/2$ digits and then use (2) once, performing the multiplications and subtraction to n digits. Looking backwards, one can obtain x_j from x_{j-1} , performing such operations to only $n/2$ digits of accuracy, and so on. Thus, for n large, accurate division takes not more than 6 times as much time as accurate multiplication.

Now we look at square roots. When $f(x) = x^2$, we approximate \sqrt{y} by taking

$$(3) \quad x_{j+1} = \frac{1}{2} \left(x_j + \frac{y}{x_j} \right).$$

One can show that, if $0 < \sqrt{y} < x_1$, then $x_j \searrow \sqrt{y}$. We can combine this with the method described above for division, to produce a quadratically convergent approximation to the pair $(\sqrt{y}, 1/\sqrt{y})$, using only addition, subtraction, and multiplication, plus a division by 2, which is very easy for a computer. Namely, given (x_j, w_j) , we set

$$(4) \quad \begin{aligned} w_{j1} &= w_j(2 - w_j x_j) \\ x_{j+1} &= \frac{1}{2}(x_j + w_{j1} y) \\ w_{j+1} &= w_{j1}(2 - w_{j1} x_{j+1}). \end{aligned}$$

To see how (4) works, suppose that x_j approximates \sqrt{y} to k digits and w_j approximates $1/x_j$ to k digits. Then w_{j1} approximates $1/x_j$ to (roughly) $2k$ digits, so x_{j+1} approximates \sqrt{y} to $2k$ digits. Consequently, w_{j1} approximates $1/x_{j+1}$ to k digits, and then w_{j+1} approximates $1/x_{j+1}$ to $2k$ digits.

For more material on these operations, and on fast computation of other functions, see [BBB].

Exercises.

1. Write computer programs to implement the algorithms described above.

2. Suppose the iteration (4) is simplified to

$$(5) \quad w_{j+1} = w_j(2 - w_j x_j), \quad x_{j+1} = \frac{1}{2}(x_j + w_{j+1} y).$$

How fast does this sequence converge to \sqrt{y} ? Why is it so much slower than (4)?

3. How many terms in the series

$$(6) \quad e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n$$

are required to compute e^x to N digits of accuracy? How many arithmetical operations does that involve, if you use **sum**, **divide** (by n), or rather the analogues of these procedures when ‘digits’ are arrays of the appropriate length, and an FFT-based multiplication (by x)? Of course, the answer depends on the size of x ; consider in particular the cases $x = 1$ and $x = \frac{1}{2}$.

4. Given $x > 0$, let us compute e^x by writing

$$(7) \quad e^x = (e^{x/\ell})^\ell$$

and computing $e^{x/\ell}$ using (6), with x replaced by x/ℓ . Given N , estimate how large k must be to guarantee that replacing $e^{x/\ell}$ by

$$(8) \quad S_k(x/\ell) = \sum_{n < k} \frac{1}{n!} \left(\frac{x}{\ell}\right)^n$$

yields $e^x - S_k(x/\ell)^\ell < 10^{-N}$.

Along the lines of problem 3, how many operations are required to evaluate $S_k(x/\ell)$ to appropriate accuracy? As in problem 3, consider in particular the cases $x = 1$ and $x = \frac{1}{2}$.

5. If now $\ell = 2^m$, compute S^ℓ by the iterative process

$$(9) \quad S_0 = S, S_1 = S^2, S_2 = S_1^2 = S^4, \dots, S_{j+1} = S_j^2, \dots, S_m = S_{m-1}^2 = S^{2^m}.$$

Thus m multiplications are involved. If $S = S_k(x/\ell)$ is the quantity described in problem 4, how many operations are then involved in approximating e^x to N digits? As before, consider particularly the cases $x = 1$ and $x = \frac{1}{2}$. Investigate the consequences of taking

$$k \approx C_1\sqrt{N}, \quad \log_2 \ell \approx C_2\sqrt{N}.$$

A faster method for computing e^x is discussed in [BB].

6. Using Euler's formula, $e^{ix} = \cos x + i \sin x$, give a discussion of approximating $\sin x$ and $\cos x$, analogous to that of problems 3-5. Consider particularly the case $x \approx \pi/6 \approx 0.524$.

7. In light of problem 6, if you use Newton's method to approximate $\pi/6$, as the solution to

$$(10) \quad \sin x = \frac{1}{2},$$

how many operations will it take to approximate π to n digits? Is this faster than the method used in §5?

While working on this problem, keep in mind that computations giving an $\frac{n}{2}$ digit approximation x_j to the solution to (10) need only be done to $\frac{n}{2}$ digits of accuracy; likewise the $\frac{n}{4}$ digit approximation x_{j-1} can arise from computations to $\frac{n}{4}$ digits of accuracy, etc.

Methods of approximating π described in [AB] and [BBB] are faster than this.

References

- [AB] G.Almkvist and B.Berndt, ‘Gauss, Landen, Ramanujan, the Arithmetic-Geometric Mean, Ellipses, π , and the *Ladies Diary*’ Amer. Math. Monthly 95(1988), 585-608.
- [Bk] Petr Beckman, A History of Pi, St. Martin’s Press, New York, 1971.
- [BB] J.Borwein and P.Borwein, ‘The arithmetic-geometric mean and fast computation of elementary functions,’ SIAM Review 26(1984), 351-366.
- [BB2] J.Borwein and P.Borwein, Pi and the AGM – A Study in Analytic Number Theory and Computational Complexity, Wiley, New York, 1987.
- [BBB] J.Borwein, P.Borwein, and D.Bailey, ‘Ramanujan, modular equations, and approximations to pi, or how to compute one billion digits of pi,’ Amer. Math. Monthly 96(1989), 201-219.