Last three digits before decimal point

Uwe Hoffmann

April 2013

1 Problem

Find the last three digits before the decimal point for the number $(3+\sqrt{5})^n$. For example, when $n = 5$, $(3+\sqrt{5})^5 = 3935.73982\ldots$, the answer is 935. For $n = 2$, $(3+\sqrt{5})^2 = 27.4164079\ldots$, the answer is 027. The value of n is in the range $2 \leq n \leq 2000000000$. [1]

2 Solution

Looking at the numbers $(3+\sqrt{5})^n$, we can see that in general they are not integers. Ideally we would like to deal with integers. This sparks the idea of introducing the complement of $(3+\sqrt{5})$ into the mix, namely $(3-\sqrt{5})$. Let's look at the binomial expansion of $(3+\sqrt{5})^n$:

$$(3+\sqrt{5})^n = \sum_{i=0}^{n} \binom{n}{i} 3^i (\sqrt{5})^{n-i}$$

Compare this to the binomial expansion of $(3-\sqrt{5})^n$:

$$(3-\sqrt{5})^n = \sum_{i=0}^{n} \binom{n}{i} 3^i (-\sqrt{5})^{n-i}$$

When $n-i$ is even, then $(\sqrt{5})^{n-i}$ and $(-\sqrt{5})^{n-i}$ are integers. When $n-i$ is odd, then the binomial terms for $(\sqrt{5})^{n-i}$ and $(-\sqrt{5})^{n-i}$ in the binomial expansions cancel each other out. So it follows that

$$\forall n \in \mathbb{N} : (3+\sqrt{5})^n + (3-\sqrt{5})^n \in \mathbb{N}$$

This is encouraging, so we define for all n:

$$\begin{aligned} a_n &= (3+\sqrt{5})^n \\ b_n &= (3-\sqrt{5})^n \\ c_n &= a_n + b_n \end{aligned}$$

We see that $\forall n \in \mathbb{N} : 0 < b_n < 1$, so $c_n = \lceil a_n \rceil$.

Concentrating on c_n, lets try to find the hundreds digit, the tens digit and the units digit of c_n.

Consider the polynomial:

$$(x-(3+\sqrt{5}))(x-(3-\sqrt{5})) = x^2 - 6x + 4$$

It leads to the recurrence relation: $f_n = 6f_{n-1} - 4f_{n-2}$, for which any linear combination of a_n and b_n is a solution. We set the initial values of f_n such that the linear combination $c_n = a_n + b_n$ is the solution: $f_0 = 2, f_1 = 6$. Therefore c_n satisfies the recurrence:

$$\begin{aligned} c_n &= 6c_{n-1} - 4c_{n-2} \\ c_0 &= 2 \\ c_1 &= 6 \end{aligned}$$

In theory we could just use this recurrence to compute c_n and then extract the hundreds digit, the tens digit and the units digit. Unfortunately this is not feasible for large n, since c_n grows quickly to very large values. But since we only need the last three digits of the values, we don't need to compute the values completely, computing them modulo 1000 will suffice.

Fortunately according to modulo arithmetic, the recurrence relation for c_n is still valid when doing modulo 1000. Let:

$$d_n \equiv c_n \mod 1000$$

and so

$$d_n \equiv 6d_{n-1} - 4d_{n-2} \mod 1000$$

Now consider the ordered pairs $(d_n, d_{n+1}), n \in \mathbb{N}$. Because $d_n \in \{0, 1, 2, \ldots, 999\}$, there are only 10^6 distinct pairs of (d_n, d_{n+1}) possible. So it must be that there exist two indices $i, j \in \mathbb{N}^+$ such that:

$$(d_i, d_{i+1}) = (d_j, d_{j+1})$$

From the recurrence it follows that:

$$\forall k \in \mathbb{N} : (d_{i+k}, d_{i+k+1}) = (d_{j+k}, d_{j+k+1})$$

The sequence of ordered pairs (d_n, d_{n+1}) is periodic with a period p of at most 10^6. We can construct a lookup table holding values of d_n from one period p and then compute d_n for large n by going into the lookup table at $n \mod p$.

The periodic part of the sequence doesn't necessarily start with the first pair in the sequence or with the second or the third etc... There might be a sequence prefix of ordered pairs that don't repeat before it goes into the sequence loop of repeating pairs. We want to write a function that computes the prefix and the period of the sequence.

We will use the trick of two cursors traversing the sequence at different speeds. The slower cursor visits one pair in one iteration. The faster cursor visits two pairs in one iteration. Eventually they will both end up in the repeating loop of the sequence. Here the faster cursor will eventually run into the slower cursor because it visits two pairs in the loop whenever the slower visits one in loop. When both the slow and the fast cursor are at the same pair we know that pair is a member of the repeating loop.

We don't know the period yet, but that's easy. We continue with the slow one until we hit that particular pair again, counting the pairs and also building a set of pairs that are members of the loop.

The pairs in the prefix must necessarily not be in the set of pairs from the loop. So to compute the prefix we start at the beginning of the sequence and we march on with a cursor until we find a pair that is part of the set of pairs from the loop (this set was built in the step before).

Listing 1 has Go code that generates the lookup table. Here the sequence of pairs has been flattened out into the sequence of values, which is what we actually need for the lookup table. Luckily it turns out that in this case the

prefix is $2, 6, 28$ and the periodic sequence has period 100 and is:

$$144, 752, 936, 608, 904, 992, 336, 48, 944, 472, 56, 448, 464, 992, 96,$$
$$608, 264, 152, 856, 528, 744, 352, 136, 408, 904, 792, 136, 648, 344,$$
$$472, 456, 848, 264, 192, 96, 808, 464, 552, 456, 528, 344, 952, 336,$$
$$208, 904, 592, 936, 248, 744, 472, 856, 248, 64, 392, 96, 8, 664, 952,$$
$$56, 528, 944, 552, 536, 8, 904, 392, 736, 848, 144, 472, 256, 648, 864,$$
$$592, 96, 208, 864, 352, 656, 528, 544, 152, 736, 808, 904, 192, 536, 448,$$
$$544, 472, 656, 48, 664, 792, 96, 408, 64, 752, 256, 528$$

With this lookup table we can compute d_n for any large n in constant time. d_n gives us the last three digits of c_n. Our goal though was to compute the last three digits before the decimal point of a_n.

We know $c_n = \lceil a_n \rceil$, so $c_n - 1 = \lfloor a_n \rfloor$. This means that to get the digits for a_n, we need to extract them from $d_n - 1$.

This concludes the solution. A complete implementation can be found at the author's github account.

Listing 1: Go code

```
type seqPair struct {
        a   int
        b   int
        mod int
}

func (s *seqPair) advance() {
        c := (6*s.b - 4*s.a) % s.mod
        if c < 0 {
                c = c + s.mod
        }

        s.a, s.b = s.b, c
}

func (s *seqPair) same(ss *seqPair) bool {
        return s.a == ss.a && s.b == ss.b
}

func findPeriod(m int) (int, int) {
        s1 := &seqPair{
                a:   2,
                b:   6,
                mod: m,
        }
        s2 := &seqPair{
                a:   2,
                b:   6,
                mod: m,
        }

        for {
                s1.advance()
                s2.advance()
                s2.advance()

                if s1.same(s2) {
                        break
                }
        }

        fa, fb := s1.a, s1.b

        p := 1
        loop := make(map[int]bool)
        loop[1000*s1.a+s1.b] = true

        for {
                s1.advance()

                if s1.a == fa && s1.b == fb {
                        break
                } else {
                        p++
                        loop[1000*s1.a+s1.b] = true
                }
        }

        s3 := &seqPair{
                a:   2,
                b:   6,
                mod: m,
        }

        q := 0

        for {
                if loop[1000*s3.a+s3.b] {
                        break
                }
                s3.advance()
                q++
        }

        return p, q
}
```

References

[1] Cosmin Negruseri. *Codejam 2008 Round 1A: Problem C: Numbers*. 2008. URL: https://code.google.com/codejam/contest/32016/dashboard#s=p2&a=2.