The Fibonacci sequence modulo p^2 – An investigation by computer for $p < 10^{14}$

Andreas-Stephan Elsenhans[1,2] and Jörg Jahnel[2]

Abstract

We show that for primes $p < 10^{14}$ the period length $\kappa(p^2)$ of the Fibonacci sequence modulo p^2 is never equal to its period length modulo p. The investigation involves an extensive search by computer. As an application, we establish the general formula $\kappa(p^n) = \kappa(p) \cdot p^{n-1}$ for all primes less than 10^{14}.

1 Introduction

1.1. —— The Fibonacci sequence $\{F_k\}_{k\geq 0}$ is defined recursively by $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$. Modulo some integer $l \geq 2$, it must ultimately become periodic as there are only l^2 different pairs of residues modulo l. Further, there will be no pre-period as the recursion may be reversed to $F_{k-2} = F_k - F_{k-1}$. The minimal period $\kappa(l)$ of the Fibonacci sequence modulo l is often called the *Wall number* as its main properties were discovered by D. D. Wall [Wa].

Wall's results may be summarized by the theorem below. It shows, in particular, that $\kappa(l)$ is in general a lot smaller than l^2. In fact, one always has $\kappa(l) \leq 6l$ whereas equality holds if and only if $l = 2 \cdot 5^n$ for some $n \geq 1$.

1.2. Theorem (Wall)**.** ——

a) *If* $\gcd(l_1, l_2) = 1$ *then* $\kappa(l_1 l_2) = \operatorname{lcm}(\kappa(l_1), \kappa(l_2))$.

In particular, if $l = \prod_{i=1}^{N} p_i^{n_i}$ *where the* p_i *are pairwise different prime numbers then* $\kappa(l) = \operatorname{lcm}(\kappa(p_1^{n_1}), \ldots, \kappa(p_N^{n_N}))$.

It is therefore sufficient to understand κ *on prime powers.*

---

[1]While this work was done the first author was supported in part by a Doctoral Fellowship of the Deutsche Forschungsgemeinschaft (DFG).

[2]The computer part of this work was executed on the Linux PCs of the Gauß Laboratory for Scientific Computing at the Göttingen Mathematical Institute. Both authors are grateful to Prof. Y. Tschinkel for the permission to use these machines as well as to the system administrators for their support.

b) $\kappa(2) = 3$ *and* $\kappa(5) = 20$. *Otherwise,*

- *if* p *is a prime such that* $p \equiv \pm 1 \pmod 5$ *then* $\kappa(p) | (p-1)$.
- *If* p *is a prime such that* $p \equiv \pm 2 \pmod 5$ *then* $\kappa(p) | (2p+2)$ *but* $\kappa(p) \nmid (p+1)$.

c) *If* $l \geq 3$ *then* $\kappa(l)$ *is even.*

d) *If* p *is prime,* $e \geq 1$, *and* $p^e | F_{\kappa(p)}$ *but* $p^{e+1} \nmid F_{\kappa(p)}$ *then*

$$\kappa(p^n) = \begin{cases} \kappa(p) & \text{for } n \leq e, \\ \kappa(p) \cdot p^{n-e} & \text{for } n > e. \end{cases} \tag{1}$$

2 Background

2.1. — Part a) is trivial.

For the **proof** of b), the formula

$$F_k = \frac{r^k - s^k}{\sqrt{5}}, \tag{2}$$

where $r = \frac{1+\sqrt{5}}{2}$ and $s = \frac{1-\sqrt{5}}{2}$, is of fundamental importance. It is easily established by induction. If $p \equiv \pm 1 \pmod 5$ then 5 is a quadratic residue modulo p and, therefore, $\frac{1\pm\sqrt{5}}{2} \in \mathbb{F}_p$. Fermat states their order is a divisor of $p-1$.

Otherwise, $\frac{1\pm\sqrt{5}}{2} \in \mathbb{F}_{p^2}$ are elements of norm (-1). As the norm map $N \colon \mathbb{F}_{p^2}^* \to \mathbb{F}_p^*$ is surjective, its kernel is a group of order $\frac{p^2-1}{p-1} = p+1$ and $\#N^{-1}(\{1,-1\}) = 2p+2$. As $\mathbb{F}_{p^2}^*$ is cyclic, we see that $N^{-1}(\{1,-1\})$ is even a cyclic group of order $2p+2$. $N(r) = N(s) = -1$ implies that both r and s are not contained in its subgroup of index two. Therefore,

$$r^{p+1} \equiv s^{p+1} \equiv -1 \pmod p. \tag{3}$$

From this, we find $F_{p+2} \equiv \frac{r^{p+2}-s^{p+2}}{\sqrt{5}} \equiv \frac{-r+s}{\sqrt{5}} \equiv -F_1 \equiv -1 \pmod p$ which shows $p+1$ is not a period of $\{F_k\}_{k\geq 0}$ modulo p.

c) In the case $p \equiv \pm 2 \pmod 5$ this follows from b). It is, however, true in general. Indeed, for every $k \in \mathbb{N}$, one has

$$\begin{aligned} F_{k+1}F_{k-1} - F_k^2 &= \tfrac{r^{2k}+s^{2k}-r^{k+1}s^{k-1}-r^{k-1}s^{k+1}}{5} - \tfrac{r^{2k}+s^{2k}-2r^ks^k}{5} \\ &= \frac{-(-1)^{k-1}(r^2+s^2)+2(-1)^k}{5} \\ &= (-1)^k \end{aligned} \tag{4}$$

as $rs = -1$ and $r^2 + s^2 = 3$. On the other hand,

$$F_{\kappa(l)+1}F_{\kappa(l)-1} - F_{\kappa(l)}^2 \equiv 1 \cdot 1 - 0^2 \equiv 1 \pmod l.$$

As $l \geq 3$ this implies $\kappa(l)$ is even.

For d), it is best to establish the following p-uplication formula first.

2.2. Lemma (Wall). — *One has*

$$F_{pk} = \frac{1}{2^{p-1}} \sum_{\substack{j=1 \\ j \text{ odd}}}^{p} \binom{p}{j} 5^{\frac{j-1}{2}} F_k^j V_k^{p-j}. \tag{5}$$

Here, $\{V_k\}_{k\geq 0}$ is the Lucas sequence *given by $V_0 = 2$, $V_1 = 1$, and $V_k = V_{k-1} + V_{k-2}$ for $k \geq 2$.*

Proof. Induction shows $V_k = r^k + s^k$. Having that in mind, it is easy to calculate as follows.

$$F_{pk} = \frac{(r^k)^p - (s^k)^p}{\sqrt{5}} = \frac{(\frac{V_k + \sqrt{5}F_k}{2})^p - (\frac{V_k - \sqrt{5}F_k}{2})^p}{\sqrt{5}}.$$

The assertion follows from the Binomial Theorem. □

2.3. Lemma. — *Assume $p \neq 2$ and l to be a multiple of $\kappa(p)$. Then, $p^e | F_l$ is sufficient for l being a period of $\{F_k \bmod p^e\}_{k\geq 0}$, i.e. for $\kappa(p^e)|l$.*

Proof. The claim is that, in our situation, $F_{l+1} \equiv 1 \pmod{p^e}$ is automatic.

For that, we note $F_{l+1}F_{l-1} - F_l^2 = 1$ where $F_l \equiv 0 \pmod{p^e}$ and, by virtue of the recursion, $F_{l-1} \equiv F_{l+1} \pmod{p^e}$. Therefore, $F_{l+1}^2 \equiv 1 \pmod{p^e}$. The assumption $\kappa(p)|l$ implies $F_{l+1} \equiv 1 \pmod{p}$.

As $p \neq 2$, Hensel's lemma says the lift is unique, i.e. $F_{l+1} \equiv 1 \pmod{p^e}$. □

2.4. — Lemma 2.3 allows us to prove d) for $p \neq 2$ in a somewhat simpler manner than D. D. Wall did it in [Wa].

First, we note that for $n \leq e$, Lemma 2.3 implies $\kappa(p^n)|\kappa(p)$. However, the divisibility the other way round is obvious.

For $n \geq e$, by Lemma 2.3, it is sufficient to prove $\nu_p(F_{\kappa(p)\cdot p^{n-e}}) = n$, i.e. that $p^n | F_{\kappa(p)\cdot p^{n-e}}$ but $p^{n+1} \nmid F_{\kappa(p)\cdot p^{n-e}}$. Indeed, the first divisibility implies $\kappa(p^n)|\kappa(p)\cdot p^{n-e}$ while the second, applied for $n-1$ instead of n, yields $\kappa(p^n) \nmid \kappa(p)\cdot p^{n-e-1}$. The result follows as $\kappa(p)|\kappa(p^n)$.

For $\nu_p(F_{\kappa(p)\cdot p^{n-e}}) = n$, we proceed by induction, the case $n = e$ being known by assumption. One has

$$F_{\kappa(p)\cdot p^{n-e+1}} = \tfrac{1}{2^{p-1}} p F_{\kappa(p)\cdot p^{n-e}} V_{\kappa(p)\cdot p^{n-e}}^{p-1} + \tfrac{1}{2^{p-1}} \sum_{\substack{j=3 \\ j \text{ odd}}}^{p} \tbinom{p}{j} 5^{\frac{j-1}{2}} F_{\kappa(p)\cdot p^{n-e}}^j V_{\kappa(p)\cdot p^{n-e}}^{p-j}.$$

In the second term, every summand is divisible by $F_{\kappa(p)\cdot p^{n-e}}^3$, i.e. by p^{3n}. The claim would follow if we knew $p \nmid V_{\kappa(p)\cdot p^{n-e}}$. This, however, is easy as there is the formula

$$V_l = F_{l-1} + F_{l+1} \tag{6}$$

which implies $V_l \equiv 2 \pmod{p}$ for l any multiple of $\kappa(p)$.

2.5. — For $p = 2$, as always, things are a bit more complicated. We still have $\kappa(2^n) = 3 \cdot 2^{n-1}$. However, for $n \geq 2$, one has $2^{n+1} | F_{3 \cdot 2^{n-1}}$ for which there is no analogue in the $p \neq 2$ case. On the other hand, $\nu_2(F_{3 \cdot 2^{n-1}+1} - 1) = n$ which is sufficient for our assertion.

The duplication formula provided by Lemma 2.2 is

$$F_{2k} = F_k V_k = F_k(F_{k-1} + F_{k+1}) = F_k^2 + 2F_k F_{k-1}. \tag{7}$$

As $F_6 = 8$, a repeated application of this formula shows $2^{n+1} | F_{3 \cdot 2^{n-1}}$ for every $n \geq 2$.

We further claim $F_{2k+1} = F_k^2 + F_{k+1}^2$. Indeed, this is true for $k = 0$ as $1 = 0^2 + 1^2$ and we proceed by induction as follows:

$$\begin{aligned} F_{2k+3} = F_{2k+1} + F_{2k+2} = F_k^2 + F_{k+1}^2 + F_{k+1}^2 + 2F_{k+1}F_k = \\ = F_{k+1}^2 + (F_k + F_{k+1})^2 = F_{k+1}^2 + F_{k+2}^2. \end{aligned} \tag{8}$$

The assertion $\nu_2(F_{3 \cdot 2^{n-1}+1} - 1) = n$ is now easily established by induction. We note that $F_7 = 13 \equiv 1 \pmod 4$ but the same is no more true modulo 8. Furthermore, $F_{3 \cdot 2^n+1} = F_{3 \cdot 2^{n-1}}^2 + F_{3 \cdot 2^{n-1}+1}^2$ where the first summand is even divisible by 2^{2n+2}. The second one is congruent to 1 modulo 2^{n+1}, but not modulo 2^{n+2}, by consequence of the induction hypothesis.

3 The Open Problems

3.1 The Period Length Modulo a Prime

3.1.1. — It is quite surprising that the Fibonacci sequence still keeps secrets. But there are at least two of them.

3.1.2. Problem. — The first open problem is "What the exact value of $\kappa(p)$?". Equivalently, one should understand what is the precise behaviour of the quotient Q given by $Q(p) := \frac{p-1}{\kappa(p)}$ for $p \equiv \pm 1 \pmod 5$ and $Q(p) := \frac{2(p+1)}{\kappa(p)}$ for $p \equiv \pm 2 \pmod 5$. One might hope for a formula expressing $Q(p)$ in terms of p but, may be, that is too optimistic.

3.1.3. — It is known that Q is unbounded. This is an elementary result due to D. Jarden [Ja, Theorem 3].

On the other hand, Q does not at all tend to infinity. If fact, in his unpublished Ph.D. thesis [Gö], G. Göttsch computes a certain average value of $\frac{1}{Q}$. To be more precise, under the assumption of the Generalized Riemann Hypothesis, he proves

$$\sum_{\substack{p \equiv \pm 1 \pmod 5 \\ p \leq x,\, p \text{ prime}}} \frac{1}{Q(p)} = C_1 \frac{x}{\log x} + O\Big(\frac{x \log\log x}{\log^2 x}\Big)$$

where $C_1 = \frac{342}{595}\prod_{p\text{ prime}}(1-\frac{p}{p^3-1}) \approx 0.331\,055\,98$. The proof shows as well that the density of $\{p \text{ prime} \mid Q(p) = 1, p \equiv \pm 1 \pmod 5\}$ within the set of all primes is equal to $C_2 = \frac{27}{38}\prod_{p\text{ prime}}(1-\frac{1}{p(p-1)}) \approx 0.265\,705\,4$.

Not assuming any hypothesis, it is still possible to verify that the right hand side constitutes an upper bound. For that, the error term needs to be weakened to $O(\frac{x \log\log\log x}{\log x \log\log x})$.

For the case $p \equiv \pm 2 \pmod 5$, G. Göttsch's results are less strong. Under the assumption of the Generalized Riemann Hypothesis, he establishes the estimate

$$\sum_{\substack{p\equiv\pm 2 \pmod 5\\ p\equiv 3 \pmod 4\\ p\leq x,\, p \text{ prime}}} \frac{1}{Q(p)} \leq C_2\frac{x}{\log x} + O\Big(\frac{x\log\log\log x}{\log x\cdot\log\log x}\Big)$$

where $C_3 = \frac{1}{4}\prod_{p\text{ prime}, p\neq 2,5}(1-\frac{p}{p^3-1}) \approx 0.210\,055\,99$. The density of the set $\{p \text{ prime} \mid Q(p) = 1, p \equiv \pm 2 \pmod 5, p \equiv 3 \pmod 4\}$ within the set of all primes is at most $C_4 = \frac{1}{4}\prod_{p\text{ prime}, p\neq 2,5}(1-\frac{1}{p(p-1)}) \approx 0.196\,818\,8$.

3.1.4. —— It seems, however, that the inequalities could well be equalities. In addition, the restriction to primes satisfying $p \equiv 3 \pmod 4$ might be irrelevant.

In fact, we performed a count for small primes $p < 2\cdot 10^7$ by computer. Up to that bound, there are 317 687 prime numbers such that $p \equiv \pm 2 \pmod 5$ and $p \equiv 3 \pmod 4$. At them, we find $Q(p) = 1$ exactly 250 246 times which is a relative frequency of $0.787\,712\,434\ \ldots = 4\cdot 0.196\,928\,108\ \ldots$.

On the other hand, there are 317 747 primes p satisfying $p \equiv \pm 2 \pmod 5$ and $p \equiv 1 \pmod 4$. Among them, $Q(p) = 1$ occurs 250 353 times which is basically the same frequency as in the case $p \equiv 3 \pmod 4$.

3.2 The Period Length Modulo a Prime Power

3.2.1. Problem. —— There is another open problem. In fact, one question was left open in the formulation of Theorem 1.2: What is the exact value of e in dependence of p? Experiments for small p show that $e = 1$. Is this always the case? In other words, does one always have

$$\kappa(p^n) = \kappa(p)\cdot p^{n-1} \tag{9}$$

similarly to the famous formula for Euler's φ function? This is the most perplexing point in D. D. Wall's whole study of the Fibonacci sequence modulo m. For $p < 10^4$, it was investigated by help of an electronic computer by Wall in 1960, already.

3.2.2. Definition. —— We call a prime number p *exceptional* if equation (9) is wrong for some $n \geq 2$.

3.2.3. Proposition. — *Let p be a prime number. Then, the following assertions are equivalent.*

i) *p is exceptional,*

ii) *$F_{\kappa(p)}$ is divisible by p^2.*

Proof. "i) $\Longrightarrow$ ii)" Assume, to the contrary, that $p^2 \nmid F_{\kappa(p)}$. By definition of $\kappa(p)$, we know for sure that nevertheless $p|F_{\kappa(p)}$. Together, these statements mean, Theorem 1.2.d) may be applied for $e = 1$ showing $\kappa(p^n) = \kappa(p) \cdot p^{n-1}$ for every $n \in \mathbb{N}$. This contradicts i).

"ii) $\Longrightarrow$ i)" We choose the maximal $e \in \mathbb{N}$ such that $p^e|F_{\kappa(p)}$. By assumption, $e \geq 2$. Then, Theorem 1.2.d) implies $\kappa(p^2) = \kappa(p)$ which shows equation (9) to be wrong for $n = 2$. p is exceptional. $\square$

3.2.4. Proposition. — *Let $p \neq 2, 5$ be a prime number.*

I. *If $p \equiv \pm 1 \pmod 5$ then the following assertions are equivalent.*

i) *p is exceptional,*

ii) *F_{p-1} is divisible by p^2,*

iii) *$r^{p-1} \equiv 1 \pmod{p^2}$.*

II. *If $p \equiv \pm 2 \pmod 5$ then the following assertions are equivalent.*

i) *p is exceptional,*

ii) *F_{2p+2} is divisible by p^2,*

iii) *F_{p+1} is divisible by p^2.*

Proof. I. "ii) $\Longrightarrow$ i)" As $(p - 1)$ is a multiple of $\kappa(p)$, Lemma 2.3 may be applied. It shows $\kappa(p^2)|(p - 1)$. This contradicts equation (9) for $n = 2$. p is exceptional.

"iii) $\Longrightarrow$ ii)" We have $r^{p-1}s^{p-1} = (-1)^{p-1} = 1$. Thus, $r^{p-1} \equiv 1 \pmod{p^2}$ implies $s^{p-1} \equiv 1 \pmod{p^2}$. Consequently, $F_{p-1} = \frac{r^{p-1}-s^{p-1}}{\sqrt{5}}$ is divisible by p^2.

"i) $\Longrightarrow$ iii)" By Proposition 3.2.3, $F_{\kappa(p)}$ is divisible by p^2. Therefore,

$$r^{\kappa(p)} = s^{\kappa(p)} = (r^{\kappa(p)})^{-1} \in (\mathbb{Z}/p^2\mathbb{Z})^*,$$

i.e. $(r^{\kappa(p)})^2 \equiv 1 \pmod{p^2}$. Since $\kappa(p)|(p-1)$, we may conclude $(r^{p-1})^2 \equiv 1 \pmod{p^2}$ from this. As we know $r^{p-1} \equiv 1 \pmod p$ by Fermat's Theorem uniqueness of Hensel's lift implies $r^{p-1} \equiv 1 \pmod{p^2}$.

II. "ii) $\Longrightarrow$ i)" As $(2p + 2)$ is a multiple of $\kappa(p)$, Lemma 2.3 may be applied. It shows $\kappa(p^2)|(2p + 2)$. This contradicts equation (9) for $n = 2$. p is exceptional.

"iii) $\Longrightarrow$ ii)" Note that $F_{2p+2} = F_{p+1}V_{p+1}$.

"i) $\Longrightarrow$ iii)" By Proposition 3.2.3, $F_{\kappa(p)}$ is divisible by p^2. In that situation, Lemma 2.3 implies that $\kappa(p)$ is actually a period of $\{F_k \bmod p^2\}_{k \geq 0}$. By consequence, $(2p + 2)$ is a period, too. This shows $p^2|F_{2p+2}$.

Since $F_{2p+2} = F_{p+1}V_{p+1}$, all we still need is $p \nmid V_{p+1}$. This, however, is clear as $V_{p+1} = r^{p+1} + s^{p+1} \equiv -2 \pmod{p}$. $\square$

4 A heuristic argument

4.1. —— By Proposition 3.2.4, the problem of finding exceptional primes is in perfect analogy to the problem of finding *Wieferich primes*.

In the Wieferich case, one knows $2^{p-1} \equiv 1 \pmod{p}$ and is interested to find the particular primes such that even $2^{p-1} \equiv 1 \pmod{p^2}$. Here, one knows $F_{\kappa(p)} \equiv 0 \pmod{p}$ and is interested in the particular primes that fulfill even $F_{\kappa(p)} \equiv 0 \pmod{p^2}$.

In the case $p \equiv \pm 1 \pmod{5}$, this is no more just an analogy. In fact, we deal with a special case of the generalized Wieferich problem with 2 being replaced by r.

4.2. —— We expect that there are infinitely many exceptional primes.

Our reasoning for this is as follows. $p|F_{\kappa(p)}$ is known by definition of $\kappa(p)$. Thus, for any individual prime p, $(F_{\kappa(p)} \bmod p^2)$ is one residue out of p possibilities. If it we were allowed to assume equidistribution, we could conclude that $p^2|F_{\kappa(p)}$ should occur with a "probability" of $\frac{1}{p}$. Further, by [RS, Theorem 5],

$$\log\log N + A - \frac{1}{2\log^2 N} \leq \sum_{\substack{p \text{ prime} \\ p \leq N}} \frac{1}{p} \leq \log\log N + A + \frac{1}{2\log^2 N},$$

at least for $N \geq 286$. Here, $A \in \mathbb{R}$ is Mertens' constant which is given by

$$A = \gamma + \sum_{p \text{ prime}} \left[\frac{1}{p} + \log\left(1 - \frac{1}{p}\right)\right] = 0.261\,497\,212\,847\,642\,783\,755\,\ldots$$

whereas γ denotes the Euler-Mascheroni constant.

This means that one should expect around $\log\log N + A$ exceptional primes less than N.

4.3. —— On the other hand, $p^3|F_{\kappa(p)}$ should occur only a few times or even not at all. Indeed, if we assume equidistribution again, then for any individual prime p, $p^3|F_{\kappa(p)}$ should happen with a "probability" of $\frac{1}{p^2}$. However,

$$\sum_{\substack{p=2 \\ p \text{ prime}}}^{\infty} \frac{1}{p^2} = 0.452\,247\,420\,041\,065\,498\,506\,\ldots\ .$$

is a convergent series.

4.4. Remark. — It is, may be, of interest that, for any exponent $n \geq 2$, one has the equality $\sum_{p \text{ prime}} \frac{1}{p^n} = \sum_{k=1}^{\infty} \frac{\mu(k)}{k} \log \zeta(nk)$ where the right hand converges a lot faster and may be used for evaluation. This equation results from the Moebius inversion formula and Euler's formula $\log \zeta(nk) = \sum\limits_{p \text{ prime}} -\log(1 - \frac{1}{p^{nk}}) = \sum_{j=1}^{\infty} \frac{1}{j} \sum\limits_{p \text{ prime}} \frac{1}{p^{jnk}}$.

4.5. — We carried out an extensive search for exceptional primes but, unfortunately, we had no success and our result is negative.

Theorem. *There are no exceptional primes $p < 10^{14}$.*
Down the earth, this means that one has $\kappa(p^n) = \kappa(p) \cdot p^{n-1}$ for every $n \in \mathbb{N}$ and all primes $p < 10^{14}$.

5 Algorithms

5.0.1. — We worked with two principally different types of algorithms. First, in the $p \equiv \pm 1 \pmod 5$ case, it is possible to compute $(r^{p-1} \bmod p^2)$. A second and more complete approach is to compute $(F_{p-1} \bmod p^2)$ in the $p \equiv \pm 1 \pmod 5$ case and $(F_{2p+2} \bmod p^2)$ or $(F_{p+1} \bmod p^2)$ in the case $p \equiv \pm 2 \pmod 5$.

5.0.2. Remark. — It is not difficult to prove that in the case $p \equiv \pm 2 \pmod 5$ exceptionality is equivalent to $r^{2p+2} \equiv 1 \pmod{p^2}$. Unfortunately, an approach based on that observation turned out to be impractical as it involves the calculation of a modular power in $R_p := \mathbb{Z}/p^2\mathbb{Z}[\sqrt{5}] = \mathbb{Z}[\sqrt{5}]/(p^2)$ in a situation where $\sqrt{5} \notin \mathbb{Z}/p^2\mathbb{Z}$. In comparison with $\mathbb{Z}/p^2\mathbb{Z}$, multiplication in R_p is a lot slower, at least in our (naive) implementations. This puts a modular powering operation in R_p out of competition with a direct approach to compute F_{2p+2} (or F_{p+1}) modulo p^2.

5.1 Algorithms based on the computation of $\sqrt{5}$

5.1.1. — If $p \equiv \pm 1 \pmod 5$ then one may routinely compute $(r^{p-1} \bmod p^2)$. The algorithm should consist of four steps.

i) Compute the square root of 5 in $\mathbb{Z}/p\mathbb{Z}$.

ii) Take the Hensel's lift of this root to $\mathbb{Z}/p^2\mathbb{Z}$.

iii) Calculate the golden ratio $r := \frac{1+\sqrt{5}}{2} \in \mathbb{Z}/p^2\mathbb{Z}$.

iv) Use a modular powering operation to find $(r^{p-1} \bmod p^2)$.

We call algorithms which follow this strategy *algorithms powering the golden ratio.*

Here, the final steps iii) and iv) are not critical at all. For iii), it is obvious that this is a simple calculation while for iv), carefully optimized modular powering operations are available. Further, ii) can be effectively done as $r^2 \equiv 5 \pmod p$ implies

$w := r - \frac{r^2-5}{p} \cdot (\frac{1}{2r} \bmod p) \cdot p$ is a square root of 5 modulo p^2. The most expensive operation here is a run of Euclid's extended algorithm in order to find $(\frac{1}{2r} \bmod p)$.

5.1.2. —— Thus, the most interesting point is i), the computation of $\sqrt{5} \in \mathbb{F}_p$. In general, there is a beautiful algorithm to find square roots modulo a prime number due to Shanks [Co, Algorithm 1.5.1]. We implemented this algorithm but let it finally run only in the $p \equiv 1 \pmod 8$ case. If $p \not\equiv 1 \pmod 8$ then there are direct formulae to compute the square root of 5 which turn out to work faster.

If $p \equiv 3 \pmod 4$ then one may simply put $w := (5^{\frac{p+1}{4}} \bmod p)$ to find a square root of 5 by one modular powering operation.

If $p \equiv 5 \pmod 8$ then one may put

$$w := (5^{\frac{p+3}{8}} \bmod p) \tag{10}$$

as long as $5^{\frac{p-1}{4}} \equiv 1 \pmod p$ and

$$w := (10 \cdot 20^{\frac{p-5}{8}} \bmod p) \tag{11}$$

if $5^{\frac{p-1}{4}} \equiv -1 \pmod p$. Note that 5 is a quadratic residue modulo p. Hence, we always have $5^{\frac{p-1}{4}} \equiv \pm 1 \pmod p$.

For sure, $(5^{\frac{p-1}{4}} \bmod p)$ can be computed using a modular powering operation. In fact, we implemented an algorithm doing that and let it run through the interval $[10^{12}, 5 \cdot 10^{12}]$.

However, $(5^{\frac{p-1}{4}} \bmod p)$ is nothing but a quartic residue symbol. For that reason, there is an actually faster algorithm which we obtained by an approach using the law of biquadratic reciprocity.

5.1.3. Theorem. —— *Let p be a prime number such that $p \equiv 5 \pmod 8$ and $p \equiv \pm 1 \pmod 5$ and let $p = a^2 + b^2$ be its (essentially unique) decomposition into a sum of two squares.*

a) *Then, a and b may be normalized such that $a \equiv 3 \pmod 4$ and b is even.*

b) *Assume a and b are normalized as described in a). Then, there are only the following eight possibilities.*

i) *$a \equiv 3, 7, 11,$ or $19 \pmod{20}$ and $b \equiv 10 \pmod{20}$.*
In this case, $5^{\frac{p-1}{4}} \equiv 1 \pmod p$, i.e. 5 is a quartic residue modulo p.

ii) *$a \equiv 15 \pmod{20}$ and $b \equiv 2, 6, 14,$ or $18 \pmod{20}$.*
Here, $5^{\frac{p-1}{4}} \equiv -1 \pmod p$, i.e. 5 is a quadratic but not a quartic residue modulo p.

Proof. a) As p is odd, among the integers a and b there must be an even and an odd one. We choose b to be even and force $a \equiv 3 \pmod 4$ by replacing a by $(-a)$, if necessary.

b) We first observe that $a^2 \equiv 1 \pmod 8$ forces $b^2 \equiv 4 \pmod 8$ and $b \equiv 2 \pmod 4$. Then, we realize that one of the two numbers a and b must be divisible by 5. Indeed, otherwise we had $a^2, b^2 \equiv \pm 1 \pmod 5$ which does not allow $a^2 + b^2 \equiv \pm 1 \pmod 5$. Clearly, a and b cannot be both divisible by 5.

If a is divisible by 5 then $a \equiv 3 \pmod 4$ implies $a \equiv 15 \pmod{20}$. $b \equiv 2 \pmod 4$ and b not divisible by 5 yield the four possibilities stated. On the other hand, if b is divisible by 5 then $b \equiv 2 \pmod 4$ implies $b \equiv 10 \pmod{20}$. $a \equiv 3 \pmod 4$ and a not divisible by 5 show there are precisely the four possibilities listed.

For the remaining assertions, we first note that $(5^{\frac{p-1}{4}} \bmod p)$ tests whether $x^4 \equiv 5 \pmod p$ has a solution $x \in \mathbb{Z}$, i.e. whether 5 is a quartic residue modulo p. By [IR, Lemma 9.10.1], we know

$$(5^{\frac{p-1}{4}} \bmod p) = \chi_{a+bi}(5)$$

where χ denotes the quartic residue symbol. The law of biquadratic reciprocity [IR, Theorem 9.2] asserts

$$\chi_{a+bi}(5) = \chi_5(a+bi).$$

For that, we note explicitly that $a + bi \equiv 3 + 2i \pmod 4$, $5 \equiv 1 \pmod 4$, and $\frac{N(5)-1}{4} = 6$ is even. Let us now compute $\chi_5(a+bi)$:

$$\begin{aligned}
\chi_5(a+bi) &= \chi_{-1+2i}(a+bi) \cdot \chi_{-1-2i}(a+bi) \\
&= \overline{\chi_{-1-2i}(a-bi)} \cdot \chi_{-1-2i}(a+bi) \\
&= \left(\frac{a+\frac{b}{2}}{5}\right) \cdot \chi_{-1-2i}(a-bi) \cdot \chi_{-1-2i}(a+bi) \\
&= \left(\frac{a+\frac{b}{2}}{5}\right) \cdot \chi_{-1-2i}(p).
\end{aligned}$$

Here, the first equation is the definition of the quartic residue symbol for composite elements while the second is [IR, Proposition 9.8.3.c)].

For the third equation, we observe that $\chi_{-1-2i}(a-bi)$ is either ± 1 or $\pm i$. By simply omitting the complex conjugation, we would make a sign error if and only if $\chi_{-1-2i}(a-bi) = \pm i$. By [IR, Lemma 9.10.1], this means exactly that $a - bi$ defines, under the identification $2i = -1$, not even a quadratic residue modulo 5. Therefore, the correction factor is $(\frac{a+\frac{b}{2}}{5})$. The final equation follows from [IR, Proposition 9.8.3.b)].

We note that, by virtue of [IR, Lemma 9.10.1], $\chi_{-1-2i}(p)$ tests whether p is a quartic residue modulo 5 or not. As p is for sure a quadratic residue, we may write

$$\chi_{-1-2i}(p) = \begin{cases} 1 & \text{if } p \equiv 1 \pmod 5, \\ -1 & \text{if } p \equiv -1 \pmod 5 \end{cases}$$

or, if we want, $\chi_{-1-2i}(p) = (p \bmod 5)$.

The eight possibilities could now be inspected one after the other. A more conceptual argument works as follows. In case i), we have

$$\left(\frac{a+\frac{b}{2}}{5}\right) = \left(\frac{a}{5}\right) = (a^2 \bmod 5) = (a^2+b^2 \bmod 5) = (p \bmod 5).$$

Therefore, $(5^{\frac{p-1}{4}} \bmod p) = 1$. On the other hand, in case ii),

$$\left(\frac{a+\frac{b}{2}}{5}\right) = \left(\frac{\frac{b}{2}}{5}\right) = \left(\frac{b^2}{4} \bmod 5\right) = (-b^2 \bmod 5) = (-a^2-b^2 \bmod 5) = \\ = -(p \bmod 5).$$

Hence, $(5^{\frac{p-1}{4}} \bmod p) = -1$. □

5.1.4. —— What we use in the actual application is merely the corollary below.

Corollary. *Let p be a prime number such that $p \equiv 5 \pmod 8$ and $p \equiv \pm 1 \pmod 5$ and let $p = a^2 + b^2$ be its decomposition into a sum of two squares.*

a) *Then, a and b may be normalized such that $a \equiv 3 \pmod 4$ and b is even.*

b) *In that situation, the following three statements are equivalent.*

i) 5 *is a quartic residue modulo p.*

ii) *b is divisible by* 5.

iii) *a is not divisible by* 5.

5.1.5. Algorithm. —— The *square sum sieve algorithm* for prime numbers p such that $p \equiv 21, 29 \pmod{40}$ runs as follows.

We investigate a rectangle $[N_1, N_2] \times [M_1, M_2]$ of numbers. We will go through the rectangle row-by-row in the same way as the electron beam goes through a screen.

a) We add 0, 1, 2, or 3 to M_1 to make sure $M_1 \equiv 2 \pmod 4$. Then, we let b go from M_1 to M_2 in steps of length four.

b) For a fixed b we sieve the odd numbers in the interval $[N_1, N_2]$.

Except for the odd case that $l|a, b$ which we decided to ignore as the density of these pairs is not too high, $l|a^2 + b^2$ implies that (-1) is a quadratic residue modulo l, i.e. we need to sieve only by the primes $l \equiv 1 \pmod 4$.

For each such l which is below a certain limit we cross out all those a such that $a \equiv \pm v_l b \pmod l$. Here, v_l is a square root of (-1) modulo l, i.e. $v_l^2 \equiv -1 \pmod l$. For practical application, this requires that the square roots of (-1) modulo the relevant primes have to be pre-computed and stored in an array once and for all.

c) For the remaining pairs (a, b), we compute $p = a^2 + b^2$ and do steps i) through iv) from 5.1.1. In step i), if b is divisible by 5 then we use formula (10) to compute the square root of 5 modulo p. Otherwise, we use formula (11).

5.1.6. — In practice, we ran the square sum sieve algorithm on the rectangles $[0, 4\,000\,000] \times [1\,580\,000, 4\,000\,000]$ and $[1\,580\,000, 4\,000\,000] \times [0, 1\,580\,000]$, thereby capturing every prime $p \in [5 \cdot 10^{12}, 1.6 \cdot 10^{13}]$ such that $p \equiv 21, 29 \pmod{40}$ plus several others.

In fact, on the second rectangle we ran a modified version, the *inverted square sum sieve,* where the two outer loops are reversed. That means, we let a go through the odd numbers in $[N_1, N_2]$ in the very outer loop. This has some advantage in speed as longer intervals are sieved at once. In other words, we go through the rectangle column-by-column.

We implemented the square sum sieve algorithms in C using the mpz functions of GNU's GMP package for arithmetic on long integers. On a single 1211 MHz Athlon processor, the computations for the first rectangle took around 22 days of CPU time. The computations for the smaller second rectangle were finished after nine days.

5.1.7. — For primes p such that $p \equiv 3 \pmod 4$ and $p \equiv \pm 1 \pmod 5$, the formula $w := (5^{\frac{p+1}{4}} \bmod p)$ for the square root of 5 makes things a lot easier. Instead of the square sum sieve we implemented the sieve of Eratosthenes. Caused by the limitations of main memory in today's PCs, we could actually sieve intervals of only about 250 000 000 numbers at once. For each such interval the remainders of its starting point have to be computed (painfully) by explicit divisions.

5.1.8. Algorithm. — More precisely, the *algorithm powering the golden ratio for primes* $p \equiv 11, 19 \pmod{20}$ runs as follows.

We investigate an interval $[N_1, N_2]$. We assume that $N_2 - N_1$ is divisible by $5 \cdot 10^9$ and that N_1 is divisible by 20.

a) We let an integer variable i count from 0 to $\frac{N_2 - N_1}{5 \cdot 10^9} - 1$.

b) For fixed i we work on the interval $I = [N_1 + 5 \cdot 10^9 \cdot i, N_1 + 5 \cdot 10^9 \cdot (i+1)]$. For each prime l which is below a certain limit, we compute $(N_1 + 5 \cdot 10^9 \cdot i \bmod l)$. Then, we cross out all $p \in I$, $p \equiv 11$ (or 19) mod 20 which are divisible by l.

c) For the remaining $p \in I$, $p \equiv 11$ (or 19) mod 20 we do steps i) through iv) from 5.1.1. In step i), we use the formula $w := (5^{\frac{p+1}{4}} \bmod p)$ to compute the square root of 5 modulo p.

5.1.9. — In practice, we ran this algorithm in order to test all prime numbers $p \in [10^{12}, 4 \cdot 10^{13}]$ such that $p \equiv 11 \pmod{20}$ or $p \equiv 19 \pmod{20}$. It was implemented in C using the mpz functions of the GMP package. Later, when testing primes above 10^{13}, we used the low level mpn functions for long natural numbers.

In particular, we implemented a modular powering function which is hand-tailored for numbers of the considered size. It uses the left-right base 2^3 powering algorithm [Co, Algorithm 1.2.3] and the sliding window improvement from mpz_powm.

Having done all these optimizations, work on the test interval $[4\cdot 10^{13}, 4\cdot 10^{13}+5\cdot 10^9]$ of 250 000 000 numbers p such that $p \equiv 11 \pmod{20}$, among them 19 955 067 primes, lasted 7:50 Minutes CPU time on a 1211 MHz Athlon processor. Sieving through the interval was done within the first 24 seconds.

5.1.10. —— Similarly, for prime numbers p satisfying the simultaneous congruences $p \equiv 1 \pmod 8$ and $p \equiv \pm 1 \pmod 5$, we implemented Shanks' algorithm [Co, Algorithm 1.5.1] to compute the square root of 5 modulo p.

5.1.11. Algorithm. —— More precisely, the *algorithm powering the golden ratio for primes* $p \equiv 1, 9 \pmod{40}$ runs as follows.

We investigate an interval $[N_1, N_2]$. We assume that $N_2 - N_1$ is divisible by 10^{10} and that N_1 is divisible by 40.

a) We let an integer variable i count from 0 to $\frac{N_2-N_1}{10^{10}} - 1$.

b) For fixed i we work on the interval $I = [N_1 + 10^{10}\cdot i, N_1 + 10^{10}\cdot(i+1)]$. For each prime l which is below a certain limit, we compute $((N_1 + 10^{10}\cdot i) \bmod l)$. Then, we cross out all $p \in I$, $p \equiv 1$ (or 9) $\pmod{40}$ which are divisible by l.

c) For the remaining $p \in I$, $p \equiv 1$ (or 9) $\pmod{40}$ we do steps i) through iv) from 5.1.1. In step i), we use Shanks' algorithm to compute the square root of 5 modulo p.

5.1.12. —— We ran this algorithm on the interval $[10^{12}, 4\cdot 10^{13}]$. After all optimizations, the test interval $[4\cdot 10^{13}, 4\cdot 10^{13} + 10^{10}]$ of 250 000 000 numbers p such that $p \equiv 1 \pmod{40}$, among them 19 954 152 primes, could be searched through on a 1211 MHz Athlon processor in 10:30 Minutes CPU time.

This is quite a lot more in comparison with the algorithm for $p \equiv 11 \pmod{20}$ or $p \equiv 19 \pmod{20}$. The difference comes entirely from the more complicated procedure to compute $\sqrt{5} \in \mathbb{F}_p$.

5.1.13. Remark. —— At a certain moment, such a running time was no longer found reasonable. A direct computation of the Fibonacci numbers could be done as well. After several optimizations of the code of the direct methods, it turned out that only the 3 mod 4 case could still compete with them. We discuss the direct methods in the subsection below.

5.2 Algorithms for a direct computation of Fibonacci numbers

5.2.1. Algorithm. —— A nice algorithm for the fast computation of a Fibonacci number is presented in O. Forster's book [Fo]. It is based on the formulae

$$\begin{aligned} F_{2k-1} &= F_k^2 + F_{k-1}^2, \\ F_{2k} &= F_k^2 + 2F_kF_{k-1}. \end{aligned} \tag{12}$$

and works in the spirit of the left-right binary powering algorithm using bits.

Our adaption uses modular operations modulo p^2 instead of integer operations. An implementation in O. Forster's Pascal-style multi precision interpreter language ARIBAS looks like this.

```
(*----------------------------------------------------------------*)
(*
** Schnelle Berechnung der Fibonacci-Zahlen mittels der Formeln
**      fib(2*k-1) = fib(k)**2 + fib(k-1)**2
**      fib(2*k)   = fib(k)**2 + 2*fib(k)*fib(k-1)
**
** Dabei werden alle Berechnungen mod m durchgeführt
*)
function fib(k,m : integer): integer;
var
    b, x, y, xx, temp: integer;
begin
    if k <= 1 then return k end;
    x := 1; y := 0;
    for b := bit_length(k)-2 to 0 by -1 do
        xx := x*x mod m;
        x := (xx + 2*x*y) mod m;
        y := (xx + y*y) mod m;
        if bit_test(k,b) then
            temp := x;
            x := (x + y) mod m;
            y := temp;
        end;
    end;
    return x;
end.
(*----------------------------------------------------------------*)


(** ein systematischer Versuch**)
function test() : integer
var
    p,r,r1 : integer;
    ptest  : boolean;
begin
    for p := 90000000001 to 95000000001 by 2 do
        if (p mod 10000) = 1 then
            writeln("getestete Zahl: ", p);
        end;
        ptest := rab_primetest(p);
        if (ptest = true) then
            if ((p mod 5 = 2) or (p mod 5 = 3)) then
                r := fib(2*p+2,p*p);
            else
                r := fib(p-1,p*p);
            end;
            if (r <= 30000000000000000) then
                r1 := r div p;
                writeln(p," ist eine interessante Primzahl. Quotient ", r1);
            end;
        end;
    end;
    return(0);
end;
```

A call to `fib(k,m)` computes ($F_k \bmod m$). test is the main function. `test()` executes an outer loop which contains a Rabin-Miller composedness test. For a pseudo prime p, it uses the function fib to compute $(F_{p-1} \bmod p^2)$ or $(F_{2p+2} \bmod p^2)$. As these are divisible by p we output the quotient instead. Note that in order to limit the output size we actually write an output only when the quotient is rather small.

5.2.2. —— ARIBAS is fast enough to ensure that this algorithm could be run from $p = 7$ up to 10^{11}. We worked on ten PCs in parallel for five days. That was our first bigger computing project concerning this problem. It showed that no exceptional primes $p < 10^{11}$ do exist, thereby a establishing a lightweight version of Theorem 4.5.

5.2.3. —— The running time made it clear that we had approached to the limits of an interpreter language. For a systematic test of larger prime numbers, the algorithm was ported to C. For the arithmetic on long integers we used the mpz functions of GMP. After only one further optimization, the integration of a version of the sieve of Eratosthenes, the interval $[10^{11}, 10^{12}]$ could be attacked. A test interval of 250 000 000 numbers was dealt with on a 1211 MHz Athlon processor in around 40 Minutes CPU time. Again, we did parallel computing on ten PCs. The search through $[10^{11}, 10^{12}]$ was finished in less than five days.

5.2.4. —— For the interval $[10^{12}, 10^{13}]$, the methods which compute $\sqrt{5} \in \mathbb{F}_p$ and square the golden ratio were introduced as they were faster than our implementation of O. Forster's algorithm at that time. For this reason, only the case $p \equiv \pm 2 \pmod 5$ was done by Forster's algorithm. It took us around 20 days on ten PCs.

6 Optimizations

6.1 Sieving

6.1.1. —— Near 10^{14}, one of about 32 numbers is prime. We work in a fixed prime residue class modulo 10, 20, or 40 but still, only one of about 13 numbers is prime. We feel that the computations of $(F_{p\pm1} \bmod p^2)$ should take the main part of the running time of our programs. Our goal is, therefore, to rapidly exclude (most of) the non-primes from the list and then to spend most of the time on the remaining numbers.

There are various methods to generate the list of all primes within an interval. Unfortunately, this section of our code is not as harmless as one could hope for. In fact, for an individual number p, one might have the idea to decide whether it is probably prime by computing $(F_{p\pm1} \bmod p)$. That is the Fibonacci composedness test. It would, unfortunately, not reduce our computational load a lot as it is almost as complex as the main computation. This clearly indicates the problem that the standard "pseudo primality tests" which are designed to test individual numbers are not well suited for our purposes. In this subsection, we will explain what we did instead in order to speed up this part of the program.

6.1.2. —— Our first programs in ARIBAS in fact used the internal primality test to check each number in the interval individually. At the ARIBAS level, this is optimal because it involves only one instruction for the interpreter.

When we migrated our programs to C, using the GMP library, we first tried the same. We used the function mpz_probab_prime with one repetition for every number to be tested. It turned out that this program was enormously inefficient. It took about 50 per cent of the running time for primality testing and 50 per cent for the computation of Fibonacci numbers. However, it could easily be tuned by a naive implementation of the sieve of Eratosthenes in intervals of length 1 000 000.

We first combined sieving by small primes and the mpz_probab_prime function because sieving by huge primes is slow. This made sure that the computation of Fibonacci numbers took the major part of the running time. However, mpz_probab_prime is not at all intended to be combined with a sieve. In fact, it checks divisibility by small primes once more. Thus, an optimization of the code for the Fibonacci numbers reversed the relation again. It became necessary to carry out a further optimization of the generation of the list of primes. We decided to abandon all pseudo primality tests. Further, we enlarged the length of the array of up to 250 000 000 numbers to minimize the number of initializations.

In principle, the sieve works as follows. Recall that we used different algorithms for the computation of the Fibonacci numbers, depending on the residue class of p modulo 10, 20, or 40. This leads to a sieve in which the number

$$S(i) := \text{starting point} + \text{residue} + \text{modulus} \cdot i$$

is represented by array position i. Since all our moduli are divisible by 2 and 5 we do no longer sieve by these two numbers.

Such a sieve is still easy to use. Given a prime $p \neq 2, 5$, one has to compute the array index i_0 of the first number which is divisible by p. Then, one can cross out the numbers at the indices $i_0, i_0+p, i_0+2p, \ldots$ until the end of the sieve is reached.

6.1.3. Optimization for the Cache Memory. —— An array of the size above fits into the memory of today's PCs but it does not fit into the cache. Thus, the speed-limiting part is the transfer between CPU and memory. Sieving by big primes is like a random access to single bytes. The memory manager has to transfer one block to the cache memory, change one byte, and then transfer the whole block back to the memory. This is the limiting bottleneck.

To avoid this problem as far as possible, we built a *two stage sieve.*

In the first stage, we sieve by the first 25 000, the “small”, primes. For that, we divide the sieve further into segments of length 30 000. These two constants were found to be optimal in practical tests. They are heavily machine dependent.

The first stage is now easily explained. In a first step, we sieve the first segment by all small primes. Then, we sieve the second segment by all small primes. We continue in that way until the end of the sieve is reached.

In the second stage, we work with all relevant “big” primes on the complete sieve, as usual.

The result of this strategy is a sieve whose segments fit into the machine’s cache. Thus, the speed of the first sieve stage is the speed of the cache, not the speed of the memory. The speed of the second stage is limited by the initialization.

On our machines the two stage sieve is twice as fast as the ordinary sieve.

6.1.4. —— The choice of the prime limit for sieving is a point of interest, too. As we search for one very particular example, it would do no harm if, from to time, we test a composite number p for $p^2|F_{p\pm1}$. When the computer would tell us p^2 divides $F_{p\pm1}$ which, in fact, it never did then it would be easy to do a reliable primality test.

As long as we sieve by small primes, it is clear that lots of numbers will be crossed out in a short time and this will reduce the running time as it reduces the number of times the actual computation of $(F_{p\pm1} \bmod p^2)$ is called. Afterwards, when we sieve by larger primes, the situation is no more that clear. We will often cross out a number repeatedly which was crossed out already before. This means, it can happen that further sieving costs actually more time than it saves.

Our tests show nevertheless that it is best to sieve *almost* till to the square root of the numbers to be tested. We introduced an automatic choice of the variable prime_limit as $\frac{\sqrt{p}}{\log\sqrt{p}}$ which means we sieve by the first $[\frac{\sqrt{p}}{\log\sqrt{p}}]$ primes. Here, p means the first prime of the interval we want to go through.

6.1.5. —— Another optimization was done by looking at the prime three. Every third number is crossed out when sieving by this prime and, when sieving by a bigger prime, every third step hits a number which is divisible by three and already crossed out.

Thus, we can work more efficiently as follows. Let p be a prime bigger than three and coprime to the modulus. We compute i_0, the first index of a number divisible by p. Then, we calculate the remainder of the corresponding number modulo three. If it is zero then we skip i_0 and continue with $i_0 := i_0+p$. Now, i_0 corresponds to the first number in the sieve which is divisible by p but not by three. Thus, we must cross out $i_0, i_0 + p, i_0 + 3p, i_0 + 4p, i_0 + 6p, \ \ldots$ or $i_0, i_0 + 2p, i_0 + 3p, i_0 + 5p, i_0 + 6p, \ \ldots$ depending on whether $i_0 + 2p$ corresponds to a number which is divisible by three or not.

6.2 The Montgomery Representation

6.2.1. —— The algorithms for the computation of Fibonacci numbers modulo m explained so far spend the lion's share of their running time on the divisions by m which occur as the final steps of modular operations such as `x := (xx + 2*x*y) mod m`. Unfortunately, on today's PC processors, divisions are by far slower than multiplications or even additions.

An ingenious method to avoid most of the divisions in a modular powering operation is due to P. L. Montgomery [Mo]. We use an adaption of Montgomery's method to O. Forster's algorithm which works as follows.

Let R be the smallest positive integer which does no more fit into one machine word. That will normally be a power of two. On our machines, $R = 2^{32}$. Recall that all operations on unsigned integers in C are automatically modular operations modulo R. We choose some exponent n such that the modulus $m = p^2$ fulfills $m \leq \frac{R^n}{5}$. In our situation $p < 10^{14}$, therefore $m = p^2 < 10^{28} < \frac{2^{96}}{5}$, such that $n = 3$ will be sufficient.

Instead of the variables $x, y, \ldots \in \mathbb{Z}/m\mathbb{Z}$, we work with their *Montgomery representations* $x_M, y_M, \ldots \in \mathbb{Z}$. These numbers are not entirely unique but bound to be integers from the interval $[0, \frac{R^n}{5})$ fulfilling $x_M \equiv R^n x \pmod{m}$. This means that modular divisions still have to be done in some initialization step, one for each variable that is initially there, but these turn out to be the only divisions we are going to execute!

A modular operation, for example $x := ((x^2 + 2xy) \bmod m)$, is translated into its *Montgomery counterpart.* In the example this is

$$x_M := \Big(\frac{x_M^2 + 2x_M y_M}{R^n} \bmod m\Big).$$

We see here that $x_M, y_M < \frac{R^n}{5}$ implies $x_M^2 + 2x_M y_M < 3 \cdot \frac{R^{2n}}{25}$. An inspection of O. Forster's algorithm shows that we always have to compute $(\frac{A}{R^n} \bmod m)$ for some $A < 5 \cdot \frac{R^{2n}}{25} = \frac{R^{2n}}{5}$.

$$A \mapsto \Big(\frac{A}{R^n} \bmod m\Big)$$

is Montgomery's REDC function. It occurs everywhere in the algorithm where normally a reduction modulo m, i.e. $A \mapsto (A \bmod m)$, would be done.

This looks as if we had not won anything. But, in fact, we won a lot as for computer hardware it is much easier to compute $(\frac{A}{R^n} \bmod m)$, which is a "reduction from below", than $(A \bmod m)$ which is a "reduction from above" and usually involves trial divisions.

Indeed, A fits into $2n$ machine words. It has $2n$ so-called *limbs.* The rightmost, i.e. the least significant, n of those have to be transformed into zero by adding some suitable multiple of m. Then, these n limbs may simply be omitted.

Which multiple of m is the suitable one that erases the rightmost limb A_0 of A? Well, $q \cdot m$ for $q := (-A_0 \cdot m^{-1} \bmod R)$ will do. This operation is in fact an ordinary multiplication of unsigned integers in C as $(-A_0)$ on unsigned integers means $(R - A_0)$ and multiplication is automatically modulo R. We add $q \cdot m$ to A and remove the last limb. This procedure of transforming the rightmost machine word of A into zero and removing it has to be repeated n times.

Still, m needs to be inverted modulo $R = 2^{32}$. The naive approach for this would be to use Euclid's extended algorithm which, unfortunately, involves quite a number of divisions. At least, we observe that it is necessary to do this only once, not n times although there are n iterations. However, for the purpose of inverting an odd number modulo 2^{32}, there exists a very elegant and highly efficient C macro in GMP, named modlimb_invert. It uses a table of the modular inverses of all odd integers modulo 2^8 and then executes two Hensel's lifts in a row. Note that, if $i \cdot n \equiv 1 \pmod{N}$ then $(2i - i^2 \cdot n) \cdot n \equiv 1 \pmod{N^2}$. We observe that, in this particular case, we need no division for the Hensel's lift.

What is the size of the representative of $(\frac{A}{R^n} \bmod m)$ found? We have $A < \frac{R^{2n}}{5}$. We add to that less than $R^n m$ and divide by R^n. Thus, the representative is less than

$$\frac{\frac{R^{2n}}{5} + R^n m}{R^n} = \frac{R^n}{5} + m.$$

We want $\text{REDC}(A) < \frac{R^n}{5}$, the same inequality we have for all variables in Montgomery representation. To reach that, we may now simply subtract m in the case we found an outcome $\geq \frac{R^n}{5}$. (This is the point where we use $m \leq \frac{R^n}{5}$.)

Our version of REDC looks as follows. In order to optimize for speed, we designed it as a C macro, not as a function.

```
#define REDC(mp, n, Nprim, tp)                                          \
do {                                                                    \
 mp_limb_t cy;                                                          \
 mp_limb_t qu;                                                          \
 mp_size_t j;                                                           \
                                                                        \
 for (j = 0; j < n; j++) {                                              \
  qu = tp[0] * Nprim;                                                   \
  /* q = tp[0]*invm mod 2^32. Reduktion mod 2^32 von selber! */         \
  cy = mpn_addmul_1 (tp, mp, n, qu);                                    \
  mpn_incr_u (tp + n, cy);                                              \
  tp++;                                                                 \
 }                                                                      \
                                                                        \
 if (tp[n - 1] >= 0x33333333)                 /* 2^32 / 5. */           \
  mpn_sub_n (tp, tp, mp, n);                                            \
} while(0);
```

It is typically called as `REDC (m, REDC_BREITE, invm, ?);`, with various variables in the place of the ?, after invm is set by `modlimb_invert (invm, m[0]);` and `invm = -invm;`. Up to now, we always had REDC_BREITE = 3.

At the very end of our algorithm we find $(F_k)_M$, the desired Fibonacci number in its Montgomery representation. To convert back, we just need one more call to REDC.

Indeed,

$$(F_k \bmod m) = \left(\frac{F_k R^n}{R^n} \bmod m\right) = \left(\frac{(F_k)_M}{R^n} \bmod m\right) = \mathrm{REDC}((F_k)_M).$$

Further, $(F_k)_M < \frac{R^n}{5}$ implies

$$\mathrm{REDC}((F_k)_M) < \frac{\frac{R^n}{5} + R^n \cdot m}{R^n} = \frac{1}{5} + m,$$

i.e. $\mathrm{REDC}((F_k)_M) \leq m$.

We note explicitly that there is quite a dangerous trap at this point. The residue 0, the one we are in fact looking for, will not be reported as 0 but as m. We work around this by outputting residues of small *absolute* value. If $(r \bmod m)$ is found and r is not below a certain output limit then $m - r$ is computed and compared with that limit.

6.2.2. Remark. — The integration of the Montgomery representation into our algorithm allowed us to avoid practically all the divisions. This caused a stunning reduction of the running time to about one third of its original value.

6.3 Other Optimizations

6.3.1. — We introduced several other optimizations. One, which is worth a mention, is the integration of a pre-computation for the first seven binary digits of p. Note, if we let p go linearly through a large interval then its first seven digits will change very slowly. This means, as a study of our algorithm for the computation of $(F_p \bmod p^2)$ shows, that the same first seven steps will be done again and again. We avoid this and do these steps once, as a pre-computation. As 10^{14} consists of 47 binary digits this saves about 14 per cent of the running time.

Of course, p is not a constant for the outer loop of our program and its first seven binary digits are only almost constant. One needs to watch out for the moment when the seventh digit of p changes.

6.3.2. — Another improvement by a few per cent was obtained through the switch to a different algorithm for the computation of the Fibonacci numbers. Our hand-tailored approach computes the k-th Fibonacci number F_k simultaneously with the k-th Lucas number V_k. It is based on the formulae

$$\begin{aligned} F_{2k} &= F_k V_k, \\ V_{2k} &= V_k^2 + 2(-1)^{k+1}, \\ F_{2k+1} &= \frac{F_k V_k + V_k^2}{2} + (-1)^{k+1}, \\ V_{2k+1} &= F_{2k+1} + 2F_k V_k. \end{aligned} \tag{13}$$

This is faster than the algorithm explained above as it involves only one multiplication and one squaring operation instead of one multiplication and two squaring operations. It seems here that the number of multiplications and the number of squaring operations determine the running time. Multiplications by two are not counted as multiplications as they are simple bit shifts. Bit shifts and additions are a lot faster than multiplications while a squaring operation costs about two thirds of what a multiplication costs.

From that point of view there should exist an even better algorithm. One can make use of the formulae

$$\begin{aligned} F_{2k+1} &= 4F_k^2 - F_{k-1}^2 + 2(-1)^k, \\ F_{2k-1} &= F_k^2 + F_{k-1}^2, \\ F_{2k} &= F_{2k+1} - F_{2k-1} \end{aligned} \tag{14}$$

which we found in the GMP source code. If we meet a bit which is set then we continue with F_{2k+1} and F_{2k}. Otherwise, with F_{2k} and F_{2k-1}.

Here, there are only two squaring operations involved and no multiplications, at all. This should be very hard to beat. Our tests, however, unearthed that the program made from (14) ran approximately ten per cent slower than the program made from (13). For that reason, we worked finally with (13). Nevertheless, we expect that for larger numbers p, in a situation where additions and bit shifts contribute even less proportion to the running time, an algorithm using (14) should actually run faster. It is possible that this is the case from the moment on that $p^2 > 2^{96}$ does no longer fit into three limbs but occupies four.

6.3.3. —— Some other optimizations are of a more practical nature. For example, instead of GMP's mpz functions we used the low level mpn functions for long natural numbers. Further, we employed some internal GMP low level functions although this is not recommended by the GMP documentation.

The point is that the size of the numbers appearing in our calculations is a-priori known to us and basically always the same. When, for example, we multiply two numbers, then it does not make sense always to check whether the base case multiplication, the Karatsuba scheme, or the FFT algorithm will be fastest. In our case, mpn_mul_basecase is always the fastest of the three, therefore we call it directly.

6.4 The Performance Finally Achieved

6.4.1. —— As a consequence of all the optimizations described, the CPU time it took our program to test the interval $[4 \cdot 10^{13}, 4 \cdot 10^{13} + 2.5 \cdot 10^9]$ of 250 000 000 numbers p such that $p \equiv 3 \pmod{10}$, among them 19 955 355 primes, was reduced to 8:08 Minutes. Sieving is done in the first 24 seconds.

The tests were made on a 1211 MHz Athlon processor. For comparison, on a 1673 MHz Athlon processor we test the same interval in around 6:30 Minutes and on a 3 GHz Pentium 4 processor in around 5:30 Minutes. (This relatively poor running time might partially be due to the fact that we carried out our trial runs on Athlon processors.)

6.4.2. The Main Computational Undertaking. — In a project of somewhat larger scale, we ran the optimized algorithm on all primes p in the interval $[10^{13}, 10^{14}]$ such that $p \equiv \pm 2 \pmod 5$. Further, as the methods which start with the computation of $\sqrt{5} \in \mathbb{F}_p$ are no longer faster, we ran it, too, on all prime numbers $p \in [4 \cdot 10^{13}, 10^{14}]$ such that $p \equiv \pm 1 \pmod 5$ and on all primes $p \in [1.6 \cdot 10^{13}, 4 \cdot 10^{13}]$ such that $p \equiv 5 \pmod 8$ and $p \equiv \pm 1 \pmod 5$.

Altogether, this means that we fully tested the whole interval $[10^{13}, 10^{14}]$. To do this took us around 820 days of CPU time. The computational work was done in parallel on up to 14 PCs from July till October 2004.

7 Output Data

7.1. A Computer Proof. — Neither our earlier computations for $p < 10^{13}$ nor the more recent ones for the interval $[10^{13}, 10^{14}]$ detected any exceptional primes. As we covered the intervals systematically and tested each individual prime, this establishes the fact that for all prime numbers $p < 10^{14}$ one has $p^2 \nmid F_{\kappa(p)}$. There are no exceptional primes below that limit. Theorem 4.5 is verified.

7.2. Statistical Observations. — We do never find $(F_{p\pm 1} \bmod p^2) = 0$. Does that mean, we have found some evidence that our assumption, the residues $(F_{p\pm 1} \bmod p^2)$ should be equidistributed in $\{ 0, p, 2p, \ \ldots \ , p^2 - p \}$, is wrong? Actually, it does not. Besides the fact that the value zero does not occur, all other reasonable statistical quantities seem to be well within the expected range.

Indeed, a typical piece of our output data looks as follows.

```
Durchsuche Fenster mit Nummer 34304.
Beginne sieben.
Restklassen berechnet.
Beginne sieben mit kleinen Primzahlen.
Sieben mit kleinen Primzahlen fertig.
Fertig mit sieben.
Initialisiere
x mit 110560307156090817237632754212345,
y mit 24722036241427551927727782157123 9
und vorz mit 1.
 10786          Quotient 1912354 mit p := 85760594147971.
 10787          Quotient 1072750 mit p := 85760627258851.
 10788          Quotient -1617348 mit p := 85760847493241.
 10789          Quotient -3142103 mit p := 85761104075891.
Initialisiere
x mit 17889033478518316825745528789179 2,
y mit 40001094909736480273272079631648 2
und vorz mit -1.
 10790          Quotient -9341211 mit p := 85761921174961.
Fertig mit Fenster mit Nummer 34304.
```

```
Durchsuche Fenster mit Nummer 34305.
Beginne sieben.
Restklassen berechnet.
Beginne sieben mit kleinen Primzahlen.
Sieben mit kleinen Primzahlen fertig.
Fertig mit sieben.
Initialisiere
x mit 178890334785183168257455287891792,
y mit 40001094909736480273272079 6316482
und vorz mit -1.
 10791          Quotient 3971074 mit p := 85763512710481.
 10792          Quotient 2441663 mit p := 85764391244491.
Fertig mit Fenster mit Nummer 34305.
```

To make the output easier to understand we do not print $(F_{p\pm1} \bmod p^2)$ which is automatically divisible by p but $R(p) := (F_{p\pm1} \bmod p^2)/p \in \mathbb{Z}/p\mathbb{Z}$. Such a quotient may be as large as p. We output only those which fall into $(-10^7, 10^7)$ which is very small in comparison to p.

The data above were generated by a process which had started at $8 \cdot 10^{13}$ and worked on the primes $p \equiv 1 \pmod{10}$. Till $8.5765 \cdot 10^{13}$ it found 10 792 primes p such that $R(p) = (F_{p-1} \bmod p^2)/p \in (-10^7, 10^7)$.

On the other hand, assuming equidistribution we would have predicted to find such a particularly small quotient for around

$$\begin{aligned}\sum_{\substack{p=8\cdot10^{13} \\ p\equiv1 \pmod{10} \\ p \text{ prime}}}^{8.5765\cdot10^{13}} \frac{2\cdot10^7-1}{p} &\approx (2\cdot10^7-1)\cdot\frac{1}{\varphi(10)}\cdot(\log(\log(8.5765\cdot10^{13}))-\log(\log(8\cdot10^{13}))) \\ &= \frac{2\cdot10^7-1}{4}\cdot(\log(\log(8.5765\cdot10^{13}))-\log(\log(8\cdot10^{13}))) \\ &\approx 10\,856.330\end{aligned}$$

primes which is astonishingly close to the reality.

Among the 10 792 small quotients found within this interval, the absolutely smallest one is $R(82\,789\,107\,950\,701) = -42$. We find 1 074 quotients of absolute value less than 1 000 000, 98 quotients of absolute value less than 100 000, and 10 of absolute value less than 10 000. These are, besides the one above,

$$\begin{aligned}R(80\,114\,543\,961\,461) &= -2437, \\ R(80\,607\,583\,847\,341) &= -6949, \\ R(80\,870\,523\,194\,401) &= -5751, \\ R(81\,232\,564\,906\,631) &= 3579, \\ R(81\,916\,669\,933\,751) &= -2397, \\ R(83\,575\,544\,636\,251) &= -1884, \\ R(84\,688\,857\,018\,011) &= -1183, \\ R(84\,771\,692\,838\,421) &= 2281, \\ R(85\,325\,902\,236\,661) &= -4473.\end{aligned}$$

There have been 5 235 positive and 5 557 negative quotients detected.

7.3. Remarks. —— a) We note explicitly that this is not at all a constructed example. One may basically consider every interval which is not too small and will observe the same phenomena.

b) Being very sceptical one might raise the objection that the computations done in our program do not really prove that the 10 792 numbers p which appear in the data are indeed prime.

It is, however, very unlikely that one of them is composite as they all passed two tests. First, they passed the sieve which in this case makes sure they have no prime divisor $\leq 8\,302\,871$. This means, if one is composite then it decomposes into the product of two almost equally large primes. Furthermore, they were all found probably prime by the Fibonacci composedness test $p|F_{p-1}$.

It is easy to check primality for all of them by a separate program.

7.4. Statistical Observations. —— A more spectacular interval is $[0, 10^{12}]$. One may expect a lot more small quotients as all small prime numbers are taken into consideration.

Here, we may do some statistical analysis on the small positive values of the quotient R' which is given by $R'(p) := (F_{p-1} \bmod p^2)/p$ for $p \equiv \pm 1 \pmod 5$ and by $R'(p) := (F_{2p+2} \bmod p^2)/p$ for $p \equiv \pm 2 \pmod 5$.

Our computations show that there exist 96 909 quotients less than 100 000, 12 162 quotients less than 10 000, 1 580 quotients less than 1 000, 216 quotients less than 100, and 30 quotients less than 10. The latter ones are

```
3 ist eine interessante Primzahl. Quotient 1
7 ist eine interessante Primzahl. Quotient 1
11 ist eine interessante Primzahl. Quotient 5
13 ist eine interessante Primzahl. Quotient 7
17 ist eine interessante Primzahl. Quotient 2
19 ist eine interessante Primzahl. Quotient 3
43 ist eine interessante Primzahl. Quotient 8
89 ist eine interessante Primzahl. Quotient 5
163 ist eine interessante Primzahl. Quotient 6
199 ist eine interessante Primzahl. Quotient 5
239 ist eine interessante Primzahl. Quotient 5
701 ist eine interessante Primzahl. Quotient 5
941 ist eine interessante Primzahl. Quotient 6
997 ist eine interessante Primzahl. Quotient 3
1063 ist eine interessante Primzahl. Quotient 2
1621 ist eine interessante Primzahl. Quotient 2
2003 ist eine interessante Primzahl. Quotient 1
27191 ist eine interessante Primzahl. Quotient 8
86813 ist eine interessante Primzahl. Quotient 6
123863 ist eine interessante Primzahl. Quotient 2
199457 ist eine interessante Primzahl. Quotient 7
508771 ist eine interessante Primzahl. Quotient 2
956569 ist eine interessante Primzahl. Quotient 4
1395263 ist eine interessante Primzahl. Quotient 3
1677209 ist eine interessante Primzahl. Quotient 1
3194629 ist eine interessante Primzahl. Quotient 5
11634179 ist eine interessante Primzahl. Quotient 2
467335159 ist eine interessante Primzahl. Quotient 4
1041968177 ist eine interessante Primzahl. Quotient 6
6_71661_90593 ist eine interessante Primzahl. Quotient 1
```

Except for 0 and 9, all one-digit numbers do appear.

Further, the counts are again well within the expected range. For example, consider one-digit numbers. $R(3)$ and $R(7)$ are automatically one-digit. Therefore, the expected count is

$$2 + \sum_{\substack{p=10 \\ p \text{ prime}}}^{10^{12}} \frac{10}{p} \approx 2 + 10 \cdot (\log(\log 10^{12}) - \log(\log 10)) \approx 26.849\,066$$

which is surprisingly close the 30 one-digit quotients which were actually found.

We note that already for two-digit quotients, it is no longer true that they appear only within the subinterval $[0, 10^{11}]$. In fact, there are twelve prime numbers $p \in [10^{11}, 10^{12}]$ such that $R'(p) < 100$. These are the following.

```
101876918491 liefert 87
115301883659 liefert 60
129316722167 liefert 44
147486235177 liefert 59
170273590301 liefert 78
233642484991 liefert 89
261836442223 liefert 45
277764184829 liefert 64
283750593739 liefert 37
305128713503 liefert 93
334015396151 liefert 79
442650398821 liefert 74
```

Once again, we may compare this to the expected count which is here

$$\sum_{\substack{p=10^{11} \\ p \text{ prime}}}^{10^{12}} \frac{100}{p} \approx 100 \cdot (\log(\log 10^{12}) - \log(\log 10^{11})) \approx 8.701\,137\,73.$$

References

[Co] Cohen, H.: A course in computational algebraic number theory, *Springer,* Graduate Texts Math. 138, Berlin 1993

[Fo] Forster, O.: Algorithmische Zahlentheorie (Algorithmic number theory), *Vieweg*, Braunschweig 1996

[Gö] Göttsch, G.: Über die mittlere Periodenlänge der Fibonacci-Folgen modulo p (On the average period length of the Fibonacci sequences modulo p), *Dissertation,* Fakultät für Math. und Nat.-Wiss., Hannover 1982

[IR] Ireland, K., Rosen, M.: A classical introduction to modern number theory, Second edition, *Springer,* Graduate Texts Math. 84, New York 1990

[Ja] Jarden, D.: Two theorems on Fibonacci's sequence, *Amer. Math. Monthly* 53 (1946)425–427

[Mo] Montgomery, P. L.: Modular multiplication without trial division, *Math. Comp.* 44(1985)519–521

[RS] Rosser, J. B., Schoenfeld, L.: Approximate formulas for some functions of prime numbers, *Illinois J. Math.* 6(1962)64–94

[Wa] Wall, D. D.: Fibonacci series modulo m, *Amer. Math. Monthly* 67(1960)525–532