15.10

Continued Fraction Factorisation

Contents

Introduction

This project is programmed in **Python 3.5**. Consult section A for program documentation, listings and information on the structure of the programming for the project, as appropriate. This report is written in $\LaTeX\,2_\varepsilon$.

1 Factor Bases

Q1. The function fb(B, n) tests if $n \in \mathbb{N}_0$ is smooth over $B \in \mathbb{P}^*$ by trial division, returning a prime factorisation if so and 0 if not. Some sample output:

```
00, 0                        17, 0                        34, 0
01, []                       18, [2, 3, 3]                35, 0
02, [2]                      19, 0                        36, [2, 2, 3, 3]
03, [3]                      20, [2, 2, 5]                37, 0
04, [2, 2]                   21, 0                        38, 0
05, [5]                      22, 0                        39, 0
06, [2, 3]                   23, 0                        40, [2, 2, 2, 5]
07, 0                        24, [2, 2, 2, 3]             41, 0
08, [2, 2, 2]                25, [5, 5]                   42, 0
09, [3, 3]                   26, 0                        43, 0
10, [2, 5]                   27, [3, 3, 3]                44, 0
11, 0                        28, 0                        45, [3, 3, 5]
12, [2, 2, 3]                29, 0                        46, 0
13, 0                        30, [2, 3, 5]                47, 0
14, 0                        31, 0                        48, [2, 2, 2, 2, 3]
15, [3, 5]                   32, [2, 2, 2, 2, 2]          49, 0
16, [2, 2, 2, 2]             33, 0                        50, [2, 5, 5]
```

Q1a

By generating a large sample of random d-digit numbers, we can approximate the probability that a d-digit number is B-smooth, as below:

```
01, 1.0                              09, 0.0005295
02, 0.8891655                        10, 0.00011775
03, 0.4878505                        11, 2.95e-05
04, 0.21479025                       12, 6.5e-06
05, 0.07941875                       13, 1e-06
06, 0.02574875                       14, 2.5e-07
07, 0.0074845                        15, 0.0
08, 0.00202425                       16, 0.0
```

Q1b

By running this random script a few times, I concluded that these numbers are acceptable, though imperfect, estimates to 3 decimal places (and so not reliable beyond $d = 9$).

2 Continued Fractions

Q2: Algorithm. Let $N \in \mathbb{N}_0$, $d = \lfloor \sqrt{N} \rfloor$. If N is square, $d = \sqrt{N}$, so the continued fraction of $\sqrt{N}$ is $(a_0) = (\sqrt{N})$. For the rest of this chapter, assume it's not; then $\sqrt{N}$ is irrational, so its continued fraction is infinite. Denote its partial quotients, remainders and convergents by (a_n), (x_n), (p_n), (q_n) $(n \in \mathbb{N}_0)$ respectively. To make its computation amenable to integer arithmetic, we introduce two new sequences of integers. Let $n \in \mathbb{N}_0$. Then

$$\sqrt{N} = \frac{x_n p_{n-1} + p_{n-2}}{x_n q_{n-1} + q_{n-2}}$$

Rearranging and rationalising the denominator,[1]

$$x_n = -\frac{(q_{n-2}\sqrt{N} - p_{n-2})(q_{n-1}\sqrt{N} + p_{n-1})}{N q_{n-1}^2 - p_{n-1}^2}$$

Expanding with the identity $p_k q_{k-1} - p_{k-1} q_k = (-1)^{k+1}$ $(\forall k \geq -1)$ gives $x_n = \frac{\sqrt{N}+r_n}{s_n}$, where

$$\begin{aligned} r_n &= (-1)^n (N q_{n-1} q_{n-2} - p_{n-1} p_{n-2}) \\ s_n &= (-1)^n (p_{n-1}^2 - N q_{n-1}^2) \end{aligned}$$

so that r_n, $s_n \in \mathbb{Z}$. Moreover, since $\frac{p_{2k}}{q_{2k}} \uparrow \sqrt{N}$ and $\frac{p_{2k+1}}{q_{2k+1}} \downarrow \sqrt{N}$ as $k \to \infty$ in $\mathbb{N}_0$ (both strictly), and since p_n, $q_n \geq 1$ (as $a_0 = d \geq 1$, by induction), it follows that, if $n \geq 1$,

$$\begin{aligned} n-1 \text{ is even} &\implies \frac{p_{n-1}}{q_{n-1}} < \sqrt{N} \implies p_{n-1}^2 - N q_{n-1}^2 < 0 \implies s_n > 0 \\ n-1 \text{ is odd} &\implies \frac{p_{n-1}}{q_{n-1}} > \sqrt{N} \implies p_{n-1}^2 - N q_{n-1}^2 > 0 \implies s_n > 0 \end{aligned}$$

and if $n = 0$, $s_n = 1 > 0$. Therefore, in all cases, $s_n \in \mathbb{N}$.

These sequences are not useful in their current form, since they require convergents p_n, q_n to be computed, which itself requires the continued fraction to have been computed some other way. Fortunately, the sequences have an integer-only recursive characterisation.

Lemma. $\forall n \in \mathbb{N}_0$

$$\begin{aligned} r_0 &= 0 & s_0 &= 1 & a_0 &= d \\ r_{n+1} &= a_n s_n - r_n & s_{n+1} &= \frac{N - r_{n+1}^2}{s_n} & a_{n+1} &= \left\lfloor \frac{d + r_{n+1}}{s_{n+1}} \right\rfloor \end{aligned}$$

Proof. By induction on $n \in \mathbb{N}_0$. $n = 0$ is immediate; suppose truth for $n-1$ $(n \in \mathbb{N})$. Then

$$\begin{aligned} a_n s_n - r_n &= (-1)^n \big(a_n p_{n-1}^2 - a_n N q_{n-1}^2 - (N q_{n-1} q_{n-2} - p_{n-1} p_{n-2})\big) \\ &= (-1)^n \big(p_{n-1}(a_n p_{n-1} + p_{n-2}) - N q_{n-1}(a_n q_{n-1} + q_{n-2})\big) \\ &= (-1)^{n+1} (N q_n q_{n-1} - p_n p_{n-1}) = r_{n+1} \\ r_{n+1}^2 + s_n s_{n+1} &= (N q_n q_{n-1} - p_n p_{n-1})(N q_n q_{n-1} - p_n p_{n-1}) - (p_{n-1}^2 - N q_{n-1}^2)(p_n^2 - N q_n^2) \\ &= -2N p_n p_{n-1} q_n q_{n-1} + N(p_n^2 q_{n-1}^2 + q_n^2 p_{n-1}^2) \\ &= N(p_n q_{n-1} - q_n p_{n-1})^2 = N(-1)^{2(n+1)} = N \\ a_{n+1} = \lfloor x_{n+1} \rfloor &= \lfloor (\sqrt{N} + r_{n+1})/s_{n+1} \rfloor = \lfloor (\lfloor \sqrt{N} \rfloor + r_{n+1})/s_{n+1} \rfloor \end{aligned}$$

where the last step uses easy facts about the floor function,[2] namely $\forall m \in \mathbb{Z}$ $\forall n \in \mathbb{N}$ $\forall x \in \mathbb{R}$

$$\lfloor x + m \rfloor = \lfloor x \rfloor + m \qquad \lfloor x/n \rfloor = \lfloor \lfloor x \rfloor / n \rfloor$$

□

[1]Noting that $q_{n-1}\sqrt{N} \pm p_{n-1} \neq 0$ since $\sqrt{N}$ is irrational and $q_{n-1} = 0 \implies n-1 = -1 \implies p_{n-1} = 1$.

[2]This is why we require that $\forall n \in \mathbb{N}_0$ $s_n > 0$.

The result of all that is an algorithm for computing partial continued fractions of $\sqrt{N}$, $n \in \mathbb{N}_0$, using only integer arithmetic, implemented as the function `cf` (which computes $(a_i)_{i=0}^k$, $(r_i)_{i=0}^k$, $(s_i)_{i=0}^k$ for $\sqrt{N}$).[3] Both of the specified division operations can be implemented as *floor division*,[4] and computing $d = \lfloor\sqrt{N}\rfloor$ can be done by testing squares (see the function `rf`).

Q2: Empirics and Theory. The output of `cf` for some small N with $k = 17$ is listed below.

```
 2; a: 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
    r: 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    s: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

 3; a: 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1
    r: 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    s: 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

 5; a: 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
    r: 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
    s: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

 6; a: 2, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2
    r: 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
    s: 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

 7; a: 2, 1, 1, 1, 4, 1, 1, 1, 4, 1, 1, 1, 4, 1, 1, 1, 4, 1
    r: 0, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2
    s: 1, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3, 1, 3

 8; a: 2, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1
    r: 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
    s: 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4

13; a: 3, 1, 1, 1, 1, 6, 1, 1, 1, 1, 6, 1, 1, 1, 1, 6, 1, 1
    r: 0, 3, 1, 2, 1, 3, 3, 1, 2, 1, 3, 3, 1, 2, 1, 3, 3, 1
    s: 1, 4, 3, 3, 4, 1, 4, 3, 3, 4, 1, 4, 3, 3, 4, 1, 4, 3

14; a: 3, 1, 2, 1, 6, 1, 2, 1, 6, 1, 2, 1, 6, 1, 2, 1, 6, 1
    r: 0, 3, 2, 2, 3, 3, 2, 2, 3, 3, 2, 2, 3, 3, 2, 2, 3, 3
    s: 1, 5, 2, 5, 1, 5, 2, 5, 1, 5, 2, 5, 1, 5, 2, 5, 1, 5

15; a: 3, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1
    r: 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
    s: 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6

19; a: 4, 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2
    r: 0, 4, 2, 3, 3, 2, 4, 4, 2, 3, 3, 2, 4, 4, 2, 3, 3, 2
    s: 1, 3, 5, 2, 5, 3, 1, 3, 5, 2, 5, 3, 1, 3, 5, 2, 5, 3
```

Q2a

[3]If n is not square; else, `cf` computes a_0, r_0, s_0.

[4]This is real division followed by floor, but is implementable in integers (e.g. by testing products).

There is a plethora of patterns visible here. The most striking is the periodicity – eventually, the (a_n, r_n, s_n) triple starts to repeat. By the design of the recursive algorithm, the triple $(a_{n+1}, r_{n+1}, s_{n+1})$ depends exclusively on (a_n, r_n, s_n). Hence, if, for $m, n \in \mathbb{N}_0$ s.t. $m < n$, $(a_n, r_n, s_n) = (a_m, r_m, s_m)$, then the sequence of triples is periodic from m onwards with period at most $n - m$. Thus, to prove eventual periodicity, it suffices to find a single repetition, and this seems likely to occur because each sequence seems to be bounded by some small number depending on $d = a_0$. Indeed, these are the maxima of the sequences for $N \in [10, 24]$ (computing 101 terms).

```
N:  10,  11,  12,  13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24
a:   6,   6,   6,   6,   6,   6,   4,   8,   8,   8,   8,   8,   8,   8,   8
r:   3,   3,   3,   3,   3,   3,   0,   4,   4,   4,   4,   4,   4,   4,   4
s:   1,   2,   3,   4,   5,   6,   1,   1,   2,   5,   4,   5,   6,   7,   8
```

Q2b

The obvious conjecture is this:

Theorem. $\forall n \in \mathbb{N} \quad a_n \in [1, 2d] \;\wedge\; r_n \in [1, d] \;\wedge\; s_n \in [1, 2d]$

Proof. $a_n \geq 1$. It was shown earlier that $s_n \geq 1$. The other bounds on r_n and s_n can be established by checking that $\frac{\sqrt{N} - r_n}{s_n} \in (0, 1)$ by induction, noting that $\frac{\sqrt{N} + r_n}{s_n} = x_n \in (1, \infty)$ and manipulating the resulting inequalities *(see the proof of theorem 2.4, Beceanu)*. Finally, $x_n = \frac{d + r_n}{s_n} \leq 2d$, so $a_n = \lfloor x_n \rfloor \leq 2d$. □

Hence, as $\forall n \in \mathbb{N}\ (r_n, s_n) \in [1, d] \times [1, 2d]$, a repetition in the sequence (r_n, s_n) must occur within $\big|[1, d] \times [1, 2d]\big| + 1 = 2d^2 + 1 \leq 2N + 1$ terms (excluding the 0th term). $a_n = \left\lfloor \frac{d + r_n}{s_n} \right\rfloor$, so this is a repetition of the entire triple; hence, (a_n, r_n, s_n) is periodic with period at most $2N$. This is evidently a very loose bound.[5]

As an aside, it is easy to check that our upper bounds on the sequences a, r, s are tight. In the case of r_n, this is always the case since $r_1 = a_0 s_0 - r_0 = a_0 = d$. For the others, we exhibit an example. In general, for the case $N = \alpha^2 - 1$ $(\alpha \geq 2)$[6], whence $d = \alpha - 1$, direct computation shows that

$$a = (d, \overline{1, 2d}) \qquad r = (0, \overline{d, d}) \qquad s = (1, \overline{2d, 1})$$

Thus, for any $d \in \mathbb{N}$, the bounds are attained with $N = (d + 1)^2 - 1$.

There are many more striking patterns visible in the continued fraction a. For instance, the termwise maximum of a is empirically attained, and there is a symmetry among the terms. These patterns are all summarised by the following theorem.

The *period* of periodic sequence $(t_n)_{n=0}^{\infty}$ is the minimal $\pi \in \mathbb{N} : \exists m \in \mathbb{N}_0 : \forall n \geq m \;\; t_{n+\pi} = t_n$. Let $\pi = \pi_N \in \mathbb{N}$ be the *period of the continued fraction of* $\sqrt{N}$ – i.e. the period of (a_n, r_n, s_n).

Theorem. *$a = (d, \overline{a_1, \ldots, a_\pi})$ and $(a_1, \ldots, a_{\pi-1})$ is palindromic (reversion-invariant). Also, π is the minimum $n \in \mathbb{N} : a_n = 2d$. In particular, π is the period of a.*

Proof. (a_n) is π-periodic, so by *theorem 2.5, Beceanu*, $\forall n \in \mathbb{N}\ a_{n+\pi} = a_n$ (so $a = (a_0, \overline{a_1, \ldots, a_\pi})$), $a_\pi = 2d$ and $(a_1, \ldots, a_{\pi-1})$ is palindromic. Let $\rho = \min\{n \in \mathbb{N} : a_n = 2d\}$. S.t.p. $\pi = \rho$.

$a_\pi = 2d$, so $\rho \leq \pi$. Since $a_\rho = 2d$, computation and bounding with the inequalities for a, r, s show that $r_\rho = d$ and $s_\rho = 1$, whence $(a_{\rho+1}, r_{\rho+1}, s_{\rho+1}) = (a_1, r_1, s_1)$. Thus, (a_n, r_n, s_n) is eventually periodic with period $\leq \rho$, so $\pi \leq \rho$. □

[5] N is in fact also a (loose) upper bound *(see corollary 2.1, Beceanu)*. The Beceanu paper is an investigation into more precise asymptotics for this quantity.

[6] Hence, N is not square.

Remark. As $(a_{\pi+1}, r_{\pi+1}, s_{\pi+1}) = (a_1, r_1, s_1)$, we also have $r = (0, \overline{r_1, \ldots, r_\pi})$ and $s = (1, \overline{s_1, \ldots, s_\pi})$.

This lends itself to a simple method of computing a closed-form continued fraction expansion of a square root of non-square $N \in \mathbb{N}_0$ – stop when the partial quotient hits double its initial value.[7] This is implemented as `cfc`(N). Thus, the expansions of $N \in [0, 50]$ are presented below.

```
 0: [0]                                16: [4]
 1: [1]                                17: [4, 8]
 2: [1, 2]                             18: [4, 4, 8]
 3: [1, 1, 2]                          19: [4, 2, 1, 3, 1, 2, 8]
 4: [2]                                20: [4, 2, 8]
 5: [2, 4]                             21: [4, 1, 1, 2, 1, 1, 8]
 6: [2, 2, 4]                          22: [4, 1, 2, 4, 2, 1, 8]
 7: [2, 1, 1, 1, 4]                    23: [4, 1, 3, 1, 8]
 8: [2, 1, 4]                          24: [4, 1, 8]
 9: [3]                                25: [5]
10: [3, 6]                             26: [5, 10]
11: [3, 3, 6]                          27: [5, 5, 10]
12: [3, 2, 6]                          28: [5, 3, 2, 3, 10]
13: [3, 1, 1, 1, 1, 6]                 29: [5, 2, 1, 1, 2, 10]
14: [3, 1, 2, 1, 6]                    30: [5, 2, 10]
15: [3, 1, 6]

31: [5, 1, 1, 3, 5, 3, 1, 1, 10]
32: [5, 1, 1, 1, 10]
33: [5, 1, 2, 1, 10]
34: [5, 1, 4, 1, 10]
35: [5, 1, 10]
36: [6]
37: [6, 12]
38: [6, 6, 12]
39: [6, 4, 12]
40: [6, 3, 12]
41: [6, 2, 2, 12]
42: [6, 2, 12]
43: [6, 1, 1, 3, 1, 5, 1, 3, 1, 1, 12]
44: [6, 1, 1, 1, 2, 1, 1, 1, 12]
45: [6, 1, 2, 2, 2, 1, 12]
46: [6, 1, 3, 1, 1, 2, 6, 2, 1, 1, 3, 1, 12]
47: [6, 1, 5, 1, 12]
48: [6, 1, 12]
49: [7]
50: [7, 14]
```

`Q2c`

[7]Though, on a computer, it is just as efficient to stop when $(r_n, s_n) = (r_1, s_1)$.

3 Pell's Equation

Q3: Observations. As in chapter 2, Pell's equations are trivial if $N \in \mathbb{N}$ is a square number. Let $x, y \in \mathbb{N}_0 : x^2 - Ny^2 = \pm 1$.[8] Then $(x + \sqrt{N}y)(x - \sqrt{N}y) = \pm 1$. But because $x \pm \sqrt{N}y$ are integers, we have $x + \sqrt{N}y$, $x - \sqrt{N}y = \pm 1$. In the case of Pell+, $x + \sqrt{N}y = x - \sqrt{N}y$, so $y = 0$ and $x = 1$, this being the only solution. Regarding Pell−, we get $x + \sqrt{N}y = -(x - \sqrt{N}y)$, so $x = 0$ and $y^2 = \frac{1}{N}$, so $N = 1$ and $y = 1$ – i.e. the only solution is $(0, 1)$ if $N = 1$ and there is no solution otherwise. These solutions are *trivial* in the sense that a component is always 0.

Moving swiftly on, let $N \in \mathbb{N}$ be non-square. We'll try to find solutions among pairs of convergents. Here are the values of $p_n^2 - Nq_n^2$, with rows corresponding to values of N (labelled on the left) and columns to values of n (counting from 0 to $k = 14$), as computed by `PellPQ`(N, k).

```
 2: -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1
 3: -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2
 5: -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1,  1, -1
 6: -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2,  1, -2
 7: -3,  2, -3,  1, -3,  2, -3,  1, -3,  2, -3,  1, -3,  2, -3
 8: -4,  1, -4,  1, -4,  1, -4,  1, -4,  1, -4,  1, -4,  1, -4
13: -4,  3, -3,  4, -1,  4, -3,  3, -4,  1, -4,  3, -3,  4, -1
14: -5,  2, -5,  1, -5,  2, -5,  1, -5,  2, -5,  1, -5,  2, -5
15: -6,  1, -6,  1, -6,  1, -6,  1, -6,  1, -6,  1, -6,  1, -6
19: -3,  5, -2,  5, -3,  1, -3,  5, -2,  5, -3,  1, -3,  5, -2
```

`Q3a`

To avoid rounding error when working with p_n^2 and other large numbers in real arithmetic, we must enforce an upper limit on it, like 10^{15}. This seemingly restricts p_n to $10^{7.5}$ or so. However, we can recover the original limit by splitting p_n into halves w.r.t. its digits (to get two numbers $\leq 10^{7.5}$ (left), 10^7 (right)) and cross-multiplying them to obtain 3 numbers $\leq 10^{15}$. To check that $p_n^2 - Nq_n^2 = \pm 1$, having computed 3 numbers apiece for p_n, q_n, we subtract them pointwise and check they're all 0 except at the products of right halves, the only influence on the units digit, which must be ± 1. Alternatively, one could just use integer arithmetic.[9]

Q3: Theory. This looks promising; there seems to always be a non-trivial solution to Pell+, and often also to Pell−. The signs of these numbers appear to oppose the parity of n, and there is periodicity but patterns are hard to spot beyond that. Happily, however, we have seen the expression $p_n^2 - Nq_n^2$ appear before, in the definition of s_n. Hence, we have the formula $\forall n \in \mathbb{N}_0$

$$p_n^2 - Nq_n^2 = (-1)^{n+1} s_{n+1}$$

To pinpoint the indices of solutions, we thus need to know when $s_n = 1$. Let π be the period of the continued fraction of $\sqrt{N}$, as previously defined. Recall[10] that $(r_\pi, s_\pi) = (d, 1)$ and that $r = (0, \overline{r_1, \ldots, r_\pi})$ and $s = (1, \overline{s_1, \ldots, s_\pi})$.

Lemma. $\forall n \in \mathbb{N}_0 \quad s_n = 1 \iff \pi | n$

Proof. ($\Leftarrow$) $s_0 = 1$. and $n \in \mathbb{N} \implies s_n = s_\pi = 1$.

($\Rightarrow$) Suppose not, so that $n \neq 0$. By the proof of [], $\frac{\sqrt{N} - r_n}{s_n} < 1$, so $r_n \leq d < \sqrt{N} < r_n + 1$, so $r_n = d$, so $(r_n, s_n) = (r_\pi, s_\pi)$. Let $\nu \in \mathbb{N} : \pi | \nu \wedge |\nu - n| \in [1, \pi - 1]$ so that $(r_\nu, s_\nu) = (r_\pi, s_\pi)$. Then $(r_n, s_n) = (r_\nu, s_\nu)$, so (a_n, r_n, s_n) is periodic with period $(= \pi) \leq |\nu - n| \leq \pi - 1$, so $\pi \leq \pi - 1$ – contradiction. $\square$

[8] Note that $(x, y) \in \mathbb{Z}^2$ is a solution of a given Pell equation iff $(|x|, |y|)$ is a solution of that equation, so we may assume w.l.o.g. that $(x, y) \in \mathbb{N}_0^2$.

[9] But they don't call it *Computer-Aided Teaching of Applied Mathematics* for nothing.

[10] From the proof of/remark following theorem [].

Now, we can characterise how convergents yield solutions to Pell's equations. $\forall n \in \mathbb{N}_0$

$$p_n^2 - Nq_n^2 = 1 \iff s_{n+1} = (-1)^{n+1} \iff n \text{ is odd } \wedge\ s_{n+1} = 1 \iff n \text{ is odd } \wedge\ \pi | n+1$$
$$p_n^2 - Nq_n^2 = -1 \iff s_{n+1} = (-1)^{n} \iff n \text{ is even } \wedge\ s_{n+1} = 1 \iff n \text{ is even } \wedge\ \pi | n+1$$

Hence, denoting $P^+ = \{n \in \mathbb{N}_0 : p_n^2 - Nq_n^2 = 1\}$ and $P^- = \{n \in \mathbb{N}_0 : p_n^2 - Nq_n^2 = -1\}$,

$$\begin{array}{lll} \pi \text{ is odd}: & P^+ = \{k\pi - 1 : k \in 2\mathbb{N}\} & P^- = \{k\pi - 1 : k \in 2\mathbb{N} - 1\} \\ \pi \text{ is even}: & P^+ = \{k\pi - 1 : k \in \mathbb{N}\} & P^- = \emptyset \end{array}$$

Thus, in most cases, some solutions to Pell's equations are generated by a subsequence of (p_n, q_n), which is distinct (as it is strictly increasing under the pointwise ordering on $\mathbb{N}_0^2$). In particular, there are always infinitely-many solutions to Pell+. The next theorem shows that these are the only solutions in $\mathbb{N}_0^2$.

Theorem. *Let $(x, y) \in \mathbb{N}_0^2 : x^2 - Ny^2 = \pm 1$. Then $\exists n \in [-1, \infty) : (x, y) = (p_n, q_n)$.*

Proof. $y = 0 \implies x = 1 \implies (x, y) = (p_{-1}, q_{-1})$. Suppose $y \in \mathbb{N}$, so that $x \neq 0$ (since $N \neq 1$). S.t.p. $\left|\sqrt{N} - \frac{x}{y}\right| < \frac{1}{2y^2}$, whence, by a property of continued fractions, $\exists n \in \mathbb{N}_0 : \frac{x}{y} = \frac{p_n}{q_n}$, so, as x, y are clearly coprime, as are p_n, q_n, $(x, y) = (p_n, q_n)$. Indeed,

$$\left|\sqrt{N} - \frac{x}{y}\right| < \frac{1}{2y^2} \impliedby \left|\sqrt{N}y - x\right| = \left|\frac{Ny^2 - x^2}{\sqrt{N}y + x}\right| = \frac{1}{\sqrt{N}y + x} < \frac{1}{2y} \impliedby 0 < (\sqrt{N} - 2)y + x$$

Also, $x^2 = Ny^2 \pm 1 \geq Ny^2 - 1 = N(y^2 - \frac{1}{N}) \geq y^2 - \frac{1}{N} > y^2 - 1$ (as $\frac{1}{N} < 1 \leq y^2$), so $x^2 \geq y^2$, so $x \geq y$, so $(\sqrt{N} - 2)y + x \geq (\sqrt{N} - 1)y > 0$. □

The two important consequences of this are that:

1. Every solution to Pell's equations turns up eventually when enumerating convergents.
2. Pell− has a solution iff π is even.

The function `PellMinus`(N) uses this latter condition to test if Pell− is solvable for $N \in \mathbb{N}$. This approach is better than manual computation because it doesn't require the computation of convergents (which get much larger than the integers used to compute the continued fraction). The script `Q3b` verifies the condition for successive integers until interrupted by the user.

```
Stopped at N = 432205.
```

Q3b

Q3: Solvability of Pell−. Here are the integers $N \in [1, 100]$ for which Pell− is solvable.

```
1, 2, 5, 10, 13, 17, 26, 29, 37, 41, 50, 53, 58, 61, 65, 73, 74,
    82, 85, 89, 97
```

Q3c

Though (ii) above seems to elucidate the solvability of Pell− at first, it turns out to be hard to classify when either of the equivalent conditions actually occurs. We can get reasonably far with a simple criterion for non-solvability.

Suppose $\exists n \in \{p \in \mathbb{P} : p \cong_4 3\} \cup \{4\} : n | N$. Then Pell− has no solution, as $\big(\exists (x, y) \in \mathbb{N}_0^2 : x^2 - Ny^2 = -1\big) \implies x^2 \cong_n -1$ – contradiction, since -1 is not a quadratic residue modulo n.

Hence, a necessary condition for the solvability of Pell− is that neither 4 nor any prime $\cong_4 3$ divide N. But is it sufficient? Let's try listing non-square numbers that don't satisfy the condition but for which Pell− is not solvable.

```
34, 146, 178, 194, 205, 221, 305, 377, 386, 410, 466
```

Q3d

So, no. I'd always felt like there was something shady about the number 34.

Q3: Computing Solutions. Finally, we turn to computing solutions to Pell's equation, $x^2 - Ny^2 = s$ ($(x, y) \in \mathbb{N}_0$), where $N \in \mathbb{N}$ is non-square and $s \in \{\pm 1\}$. The aim is to compute $n \in \mathbb{N}$ non-trivial solutions. One method follows on from previous work; we can enumerate convergents, knowing exactly how many/which convergents to keep as solutions. We also know that any solution can eventually be reached in principle. This method is implemented by the function `PellA`(N, s, n).

Another approach is sketched out for interest's sake[11] (and implemented as `PellB`(N, s, n)). $\mathbb{Z}[\sqrt{N}] = \{a + b\sqrt{N} : (a, b) \in \mathbb{Z}^2\}$ is a subring of $\mathbb{R}$ and there is a bijection $\phi : \mathbb{Z}^2 \leftrightarrow \mathbb{Z}[\sqrt{N}]$, $(a, b) \mapsto a + b\sqrt{N}$. We have two useful algebraic relations: $\forall (a, b), (x, y) \in \mathbb{Z}^2$

$$(x + y\sqrt{N})(a + b\sqrt{N}) = xa + Nyb + (xb + ya)\sqrt{N}$$

$$(a, b) \neq 0 \implies (a + b\sqrt{N})^{-1} = \frac{a - b\sqrt{N}}{a^2 - Nb^2} = s(a - b\sqrt{N})$$

Define $(\alpha, \beta) := (p_{\min(P^s)}, q_{\min(P^s)})$ to be the *fundamental solution* of Pell$s$. It is the first non-trivial solution enumerated by convergents. Firstly, consider $s = +1$:

Theorem. *The solution set of Pell+ on* $\mathbb{N}_0^2$ *is* $S := \left\{\phi^{-1}\left((\alpha + \beta\sqrt{N})^n\right)\right\}_{n \in \mathbb{N}_0}$.

Proof. By *theorem 2, Pang*, the solution set of Pell+ on $\mathbb{Z}^2$ is $X := \left\{\phi^{-1}\left(\pm(\alpha + \beta\sqrt{N})^n\right)\right\}_{n \in \mathbb{Z}}$. For $n \in \mathbb{N}_0$, let $(x_n, y_n) = \phi^{-1}\left((\alpha + \beta\sqrt{N})^n\right) \in \mathbb{Z}^2$.

$(1, 0)$, $(\alpha, \beta) \in \mathbb{N}_0^2$, so by induction, $\forall n \in \mathbb{N}_0$ $(x_n, y_n) \in \mathbb{N}_0^2$, so $S \subseteq X \cap \mathbb{N}_0^2$.

Also by induction, $\forall n \in \mathbb{Z}$ $\phi^{-1}\left(\pm(\alpha + \beta\sqrt{N})^n\right) = (\pm x_{|n|}, \pm\text{sgn}(n) y_{|n|})$, so $X \cap \mathbb{N}_0^2 \subseteq S$. $\square$

Hence, we can recursively generate solutions via (for $n \in \mathbb{N}$):

$$(x_0, y_0) = (1, 0) \qquad (x_n, y_n) = (x_{n-1}\alpha + Ny_{n-1}\beta, x_{n-1}\beta + y_{n-1}\alpha)$$

The kth solution so-generated is the kth solution given by computing convergents (including the first convergent-derived solution, $(p_{-1}, q_{-1}) = (1, 0)$), since both procedures enumerate all solutions as a pointwise–strictly increasing sequence. Similarly, the solution set on $\mathbb{N}_0^2$ of Pell− is $\left\{\phi^{-1}\left((\alpha + \beta\sqrt{N})^n\right)\right\}_{n \in 2\mathbb{N}-1}$, which can be enumerated analogously.

Q3: Output. Here are the first 4 non-trivial solutions to Pell's equations for $N \in [1, 8]$ non-square, where existent.

```
2+: [3, 2], [17, 12], [99, 70], [577, 408]
2-: [1, 1], [7, 5], [41, 29], [239, 169]
3+: [2, 1], [7, 4], [26, 15], [97, 56]
3-:
5+: [9, 4], [161, 72], [2889, 1292], [51841, 23184]
5-: [2, 1], [38, 17], [682, 305], [12238, 5473]
6+: [5, 2], [49, 20], [485, 198], [4801, 1960]
6-:
7+: [8, 3], [127, 48], [2024, 765], [32257, 12192]
7-:
8+: [3, 1], [17, 6], [99, 35], [577, 204]
8-:
```

Q3e

[11]This approach (B) is more efficient, both spatially and temporally, than the original approach (A), but differences are only significant when A would be required to compute 10^5 or so convergents, by my estimate. As an example, I tested both with input $(61, 1, 4000)$. A caused about 4s of CPU load and required about 1GB of extra RAM, whereas B produced no noticeable effect. Python could verify the equality of the output instantaneously, and that each of the solutions was valid (with no integer overflow optimisations) with 12s of CPU load. However, printing the 4000th solution in the console caused major CPU load and instability!

Here are the fundamental solutions to Pell+, for $N \in [1, 100] \cup [500, 550]$ non-square.

```
 2: [3, 2]
 3: [2, 1]
 5: [9, 4]
 6: [5, 2]
 7: [8, 3]
 8: [3, 1]
10: [19, 6]
11: [10, 3]
12: [7, 2]
13: [649, 180]
14: [15, 4]
15: [4, 1]
17: [33, 8]
18: [17, 4]
19: [170, 39]
20: [9, 2]
21: [55, 12]
22: [197, 42]
23: [24, 5]
24: [5, 1]
26: [51, 10]
27: [26, 5]
28: [127, 24]
29: [9801, 1820]
30: [11, 2]
31: [1520, 273]
32: [17, 3]
33: [23, 4]
34: [35, 6]
35: [6, 1]
37: [73, 12]
38: [37, 6]
39: [25, 4]
40: [19, 3]
41: [2049, 320]
42: [13, 2]
43: [3482, 531]
44: [199, 30]
45: [161, 24]
46: [24335, 3588]
47: [48, 7]
48: [7, 1]
50: [99, 14]
51: [50, 7]
52: [649, 90]
```

```
53: [66249, 9100]
54: [485, 66]
55: [89, 12]
56: [15, 2]
57: [151, 20]
58: [19603, 2574]
59: [530, 69]
60: [31, 4]
61: [1766319049, 226153980]
62: [63, 8]
63: [8, 1]
65: [129, 16]
66: [65, 8]
67: [48842, 5967]
68: [33, 4]
69: [7775, 936]
70: [251, 30]
71: [3480, 413]
72: [17, 2]
73: [2281249, 267000]
74: [3699, 430]
75: [26, 3]
76: [57799, 6630]
77: [351, 40]
78: [53, 6]
79: [80, 9]
80: [9, 1]
82: [163, 18]
83: [82, 9]
84: [55, 6]
85: [285769, 30996]
86: [10405, 1122]
87: [28, 3]
88: [197, 21]
89: [500001, 53000]
90: [19, 2]
91: [1574, 165]
92: [1151, 120]
93: [12151, 1260]
94: [2143295, 221064]
95: [39, 4]
96: [49, 5]
97: [62809633, 6377352]
98: [99, 10]
99: [10, 1]
```

```
500: [930249, 41602]
501: [1124731902975, 502288218432]
```

```
502: [3832352837, 171046278]
503: [24648, 1099]
504: [449, 20]
505: [809, 36]
506: [45, 2]
507: [1351, 60]
508: [44757606858751, 1985797689600]
509: [313201220822405001, 13882400040814700]
510: [271, 12]
511: [4188548960, 185290497]
512: [665857, 29427]
513: [13771351, 608020]
514: [4625, 204]
515: [17406, 767]
516: [16855, 742]
517: [590968985399, 25990786260]
518: [2367, 104]
519: [14851876, 651925]
520: [6499, 285]
521: [32961431500035201, 1444066532654320]
522: [19603, 858]
523: [81810300626, 3577314675]
524: [225144199, 9835470]
525: [6049, 264]
526: [84056091546952933775, 3665019757324295532]
527: [528, 23]
528: [23, 1]
530: [1059, 46]
531: [530, 23]
532: [2588599, 112230]
533: [74859849, 3242540]
534: [3678725, 159194]
535: [1618804, 69987]
536: [145925, 6303]
537: [192349463, 8300492]
538: [9536081203, 411129654]
539: [3970, 171]
540: [119071, 5124]
541: [3707453360023867028800645599667005001,
   159395869721270110077187138775196900]
542: [4293183, 184408]
543: [669337, 28724]
544: [2449, 105]
545: [1961, 84]
546: [701, 30]
547: [160177601264642, 6848699678673]
548: [6083073, 259856]
549: [1766319049, 75384660]
550: [30580901, 1303974]
```

Q3f

Finally, here are the fundamental solutions to Pell−, when existent, for $N \in [1, 100] \cup [500, 550]$ non-square.

```
  2: [1, 1]                          53: [182, 25]
  5: [2, 1]                          58: [99, 13]
 10: [3, 1]                          61: [29718, 3805]
 13: [18, 5]                         65: [8, 1]
 17: [4, 1]                          73: [1068, 125]
 26: [5, 1]                          74: [43, 5]
 29: [70, 13]                        82: [9, 1]
 37: [6, 1]                          85: [378, 41]
 41: [32, 5]                         89: [500, 53]
 50: [7, 1]                          97: [5604, 569]

509: [395727950, 17540333]
521: [128377240, 5624309]
530: [23, 1]
533: [6118, 265]
538: [69051, 2977]
541: [1361516316469227450, 58536158470221581]
```

Q3g

4 Continued Fraction Factorisation

Q4: Introduction. Continued fraction factorisation (CFF) aims to factorise $N \in \mathbb{N}$ by finding $x, y \in [0, N-1] : x^2 \equiv_N y^2$. Given such numbers, we have $(x-y)(x+y) \equiv_N 0$, i.e. $N|(x-y)(x+y)$. Supposing that $N \not| x \pm y$, any proper prime factor p of N (existent as $N \neq 1$ is not prime) must divide either of $x-y$ or $x+y$, so the highest common factor of N with one of $x-y$ and $x+y$, lying in $[1, N]$, is guaranteed to be at least p but is not N itself – i.e. is a non-trivial factor of N.

Is it even possible to find such x, y? If $N \neq 1$ is odd and composite, it turns out that it always is.[12] Let $\{p_i\}_{i=1}^n \subset \mathbb{P}\backslash\{2\}$ be distinct, $\{a_i\}_{i=1}^n \subset \mathbb{N} : N = \prod_i p_i^{a_i}$. If $\exists j \in [1, n] : a_j \geq 2$, then $x = 0$ and $y = \prod_i p_i^{a_i - \delta_{ij}}$ clearly work. Else, $N = \prod_{i=1}^n p_i$. Let $\alpha \in \{\pm 1\}^n$. Then by Chinese Remainder Theorem (CRT) existence (as $(p_i)_i$ is pairwise-coprime), $\exists x_\alpha \in [0, N-1] : \forall i \in [1, n]\ \ x_\alpha \equiv_{p_i} \alpha_i$, so $\forall i \in [1, n]\ \ x_\alpha^2 \equiv_{p_i} 1$, so by CRT uniqueness, $x_\alpha^2 \equiv_N 1$. Now, $\forall i \in [1, n]$ $-1 \not\equiv_{p_i} 1$, so $\{x_\alpha\}_{\alpha \in \{\pm 1\}^n} \subseteq [0, N-1]$ is distinct. As $n \geq 2$, there are thus $2^n \geq 4$ square roots of 1 modulo N. Let x be one of them, and y another s.t. $y \neq x, N-x$. Then x, y work.

Since the construction of x, y requires explicit knowledge of the factorisation of N, this is useless as a way of finding a factorisation; CFF aims to find x, y by educated guesswork.

Q4: Computation of HCFs. The determination of $\text{hcf}(N, x \pm y)$ is done by Euclid's algorithm, implemented as `Euclid`(a, b) $(a, b \in \mathbb{N}_0)$. This simple algorithm executes one check and one integer division (both of complexity $O(1)$) in each recursive step except the last, when the division is forgone. Hence, its complexity is the number of steps. Denote the residue of a mod b (if $b \neq 0$) by $a \,\%\, b$.[13] If $a < b$, the first step just calls the algorithm on (b, a), where $b \geq a$, so suppose $a \geq b \geq 1$. Then at every step,[14] $a = qb + a \,\%\, b$ $(q \geq 1)$, so $a \geq b + a \,\%\, b > 2(a \,\%\, b)$ – i.e. $a \,\%\, b < \frac{a}{2}$ (so every two non-halting steps, a more than halves, and clearly $b \leq a$ always). The algorithm halts when $b = 0$, which must happen at most one step after $a \leq 1$. By the above, $a \leq a_0 2^{-\lfloor \frac{n}{2} \rfloor}$ where n is the number of steps (if not yet halted), so $a \leq 1 \impliedby a_0 2^{-\lfloor \frac{n}{2} \rfloor} \leq 1 \impliedby a_0 \leq 2^{\frac{n-1}{2}} \impliedby 2\log_2(a_0) + 1 \leq n$. Therefore, there are at most $2\log_2(a) + 3$ steps, and Euclid's algorithm has complexity $O(\log(\max(a, b)))$. Hence, given $x, y \in [0, N-1] : x^2 \equiv_N y^2$, the determination of a proper factor of N (if existent) has complexity $O(\log(N))$.

Q5. `cfpm`(N, k) computes $(p_n \,\%\, N)_{n=0}^k$ by alternately generating p_n and reducing them modulo N. Here are initial values of $(p_n \,\%\, N)$ (`p`) and $(p_n^2 \,\%\, N)$ (`pp`) for some N, thus computed.

```
  N: 2012449237
  p: 0, 1,      44860, 134581,     4889776, 5024357,    9914133
 pp: 0, 1, 2012419600,   2428, 2012394616,   34521, 2012406609

  N: 2575992413
  p: 0, 1,      50754, 203017,      862822, 23499211,   47861244
 pp: 0, 1, 2575968516,  23681, 2575988740,     40124, 2575952112

  N: 3548710699
  p: 0, 1,      59571, 1012708,     1072279, 9590940,   20254159
 pp: 0, 1, 3548704041,  101253, 3548698064,   54821, 3548703580
```

Q5

[12]If it is even and composite, well, it is divisible by 2.

[13]Mimicking Python's notation.

[14]Except the last, when $b = 0$ becomes true.

Denote the residue of $a \in \mathbb{Z}$ mod N by $\overline{a} \in [0, N-1]$. To avoid “integer overflow” in the computation of $\overline{p_n^2}$, we conjure another pointless technique. Given $p_n \in [0, N-1]$ ($N \leq 10^{10}$), we decompose its decimal expansion as $p_n = 10^5 a + b$, $a, b \in [0, 10^5 - 1]$. Then,

$$\overline{p_n^2} = \overline{10^{10}a^2 + 2 \cdot 10^5 ab + b^2} = \overline{\overline{\overline{10^{10}a}a} + 2 \cdot \overline{10^5 ab} + b^2}$$

demonstrating a way to evaluate $\overline{p_n^2}$ while keeping all intermediate values below 10^{15}.

Q6. The generator function `ker(`A`)` takes as input $A \in \mathrm{Mat}_{mn}(\mathbb{Z}_2)$ ($m, n \in \mathbb{N}$) and yields, upon request,[15] first the dimension of its kernel, and then a listing of $\ker(A) \backslash \{0\}$. Sample output follows:

```
_  _  _  _
                                ker\{0}: (dim = 2)
1  1  0  0                      0, 0, 0, 1
1  0  1  0                      1, 1, 1, 0
1  1  0  0                      1, 1, 1, 1
_  _  _  _
```

`Q6`

Q7: Introduction. Our aim, in factorising $N \in \mathbb{N}$, is essentially to find $x, y \in [0, N-1]$: $x^2 \equiv_N y^2$ and $x \not\equiv_N \pm y$ (conditions that guarantee a proper non-trivial factor of N as per Q4). We make the following guesses:

- Fix $B \subset \mathbb{P}$ finite, a factor base.
- Find $\{b_i\}_{i \in A} \subseteq [2, N-1]$, a finite set of numbers that are B-smooth.

Now, we analyse them as follows. We set $x = \prod_{i \in I} b_i$, for some $I \subseteq A$ to be determined later, whence x^2 is square. To find another square that is congruent to it mod N, note that $x^2 \equiv_N \prod_{i \in I} \langle b_i^2 \rangle$, where $\langle x \rangle$ is the residue of $x \in \mathbb{Z}$ mod N contained in $(-\frac{N}{2}, \frac{N}{2}]$.[16] $\prod_{i \in I} \langle b_i^2 \rangle$ is not necessarily square; our hope to find $I \subseteq A$ ($I \neq \emptyset$)[17] such that it is. If we can do this, we let $y = \sqrt{\prod_{i \in I} \langle b_i^2 \rangle}$. Now, $\bar{x}, \bar{y}$ satisfy all the targeted conditions except perhaps $\bar{x} \not\equiv_N \pm \bar{y}$. Hopefully, computing $\mathrm{hcf}(N, \bar{x} \pm \bar{y})$ will still yield proper non-trivial factors.

We want $\{b_i\}_{i \in A}$ to be smooth over a factor base because this yields an efficient way of checking that $\prod_{i \in I} \langle b_i^2 \rangle$ is square for given $I \subseteq A$. Indeed, one can construct the matrix $(x_{st})_{(s,t)} \in \mathrm{Mat}_{B \cup \{-1\}, A}(\mathbb{Z}_2)$, where x_{st} is 1/0 s appears to an odd/even power in the prime factorisation of b_t (if $s \in B$), or if b_t is itself odd/even (if $s = -1$). Then I works iff $(1[a \in I])_{a \in A}$ is in $\ker((x_{st})) \backslash \{0\}$, as computable by `ker`.

Suppose that N is not square.[18] Then the convergents $(p_n)_{n=0}^{\infty}$ have the useful property that

$$p_n^2 \equiv_N p_n^2 - N q_n^2 = (-1)^{n+1} s_{n+1} \in (-2\sqrt{N}, 2\sqrt{N})$$

so if $N \geq 16$, whence $2\sqrt{N} \leq \frac{N}{2}$, then

$$\langle p_n^2 \rangle \in (-2\sqrt{N}, 2\sqrt{N})$$

Thus, as $\frac{2\sqrt{N}}{N/2} = \frac{4}{\sqrt{N}} \to 0$ as $n \to \infty$, the numbers $\langle p_n^2 \rangle$ are asymptotically small.[19] This makes them a good source of B-smooth numbers, because their factorisations are likely to

[15] I.e. being called by the `next` or `for` statements.

[16] Though any residue will do.

[17] If $I = \emptyset$, $x = y = 1$, so $\mathrm{hcf}(N, \bar{x} \pm \bar{y})$ (as below) yields N or 2.

[18] We can check this by checking if $\lfloor \sqrt{N} \rfloor^2 = N$, and have found a factor if so.

[19] Note that this accords with the data given by `Q5`, where all digits of $\overline{p_n^2}$ except the last 5 agreed with those of N, a 10-digit number (because $2\sqrt{N}$ is a 5-digit number in each case).

consist of small primes, making it more likely that there is repetition of factors between their factorisations, and so that we can find the I we hoped for (making $\prod_{i\in I}\langle b_i^2\rangle$ square).

Hence, we compute a fixed number of convergents, say $\{p_i\}_{i=0}^k$, $k \in \mathbb{N}$. To construct B, we factorise the $\langle p_i^2\rangle$ using trial division[20] and keep any primes that appear at least twice among the (unified) factorisation of $\prod_{i=0}^k\langle p_i^2\rangle$, since if a $\langle p_i^2\rangle$ is to be included in square $\prod_{i\in I}\langle p_i^2\rangle$, it cannot contain a single instance of a prime that appears in no other $\langle p_j^2\rangle$. Finally, we keep the convergents that are B-smooth as the set A.

Q7: Implementation. The version of CFF detailed above is implemented as `cff`(N, k), which attempts to factorise $N \in \mathbb{N}$ using the convergents $\{p_i\}_{i=0}^k$, $k \in \mathbb{N}_0$. The algorithm returns proper non-trivial factors if it finds them, and nothing otherwise; it also prints `p` $= \{p_i\}_{i=0}^k$, `q` $= \{\langle p_i^2\rangle\}_{i=0}^k$, a list of full factorisations `f` of $\{\langle p_i^2\rangle\}_{i=0}^k$, the factor basis `B`, the indices of B-smooth convergents `A`, and then iteratively until a factor is found, the following: subsets `I` $\subseteq$ `A` : $\prod_{i\in\mathtt{I}}\langle p_i^2\rangle$ is square, and $\bar{x}$, $\bar{y}$. `p`, `q`, `f` are disabled by default. Sample output:

```
B = [3, 37, 89, 2, 7, 17, 19, 379, 53, 31, 107]
A = [0, 2, 7, 9, 11, 19, 26, 31, 35, 37, 40, 41, 42, 43, 50]

I = [26, 50] ; x = 163394263 ; y = 17442
[45751, 43987]
```

`cff(2012449237,50)`

```
B = [23, 7, 17, 2, 211, 41, 11, 241, 149, 71, 31, 709, 19, 59, 107]
A = [5, 8, 13, 17, 21, 23, 24, 31, 34, 37, 43, 45, 46, 47]

I = [5, 13, 17, 23, 45] ; x = 476063174 ; y = 387922435
[36467, 70639]
```

`cff(2575992413,50)`

```
B = [2, 3, 5, 7, 19, 13, 41, 37, 61, 29, 163]
A = [2, 10, 11, 17, 18, 24, 26, 35, 41, 44, 48]

I = [2, 41, 44] ; x = 3393643449 ; y = 7521150
[31267, 113497]
```

`cff(3548710699,50)`

```
B = [2, 3, 5, 19, 7, 31, 17, 59, 103, 281, 71, 37, 13, 127, 467,
     73, 23]
A = [3, 7, 8, 13, 16, 21, 24, 28, 33, 34, 35, 37, 40, 46, 49]

I = [8, 16, 33, 40, 46] ; x = 1349853548 ; y = 1188247663
I = [8, 16, 35] ; x = 2633813 ; y = 2633813
I = [33, 35, 40, 46] ; x = 2317095244 ; y = 221005967
I = [3, 24, 34] ; x = 2537666901 ; y = 434310
[3]
```

`cff(2538101211,50)`

[20]Trial division is likely much faster on the smaller $\langle p_i^2\rangle$ than on the original N.

Q7: Number of Convergents Required. If a factorisation succeeds for some $k \in \mathbb{N}$ number of convergents, it will succeed for any higher number (assuming unlimited computation time), since the factor basis and set of smooth numbers both expand if k increases, and so the same selection of convergents will be enumerated eventually. Thus, the minimum number of convergents is given by any k for which factorisation succeeds but fails for $k-1$.

Thus, the example polynomials require, respectively, 9/45/44/34 convergents. Examples of similar order of magnitude can be constructed that require much greater numbers of convergents, like 1380947153, which requires 462. This number is a product of two primes and so is also relatively hard to factorise by trial division.

By generating random numbers and attempting to factorise them, I estimate that around 20 convergents are typically required for 5-digit numbers, though there was bias introduced by rejecting numbers that required over 100 convergents (typically 1 in a sample of 20).[21]

Q7: Efficiency. For very large N, CFF becomes bottlenecked by the time taken to factorise the $\langle p_i^2 \rangle$ by trial division. This can be avoided by applying CFF recursively (since each value of $\langle p_i^2 \rangle$ is much smaller than N, as above), but this would be difficult to implement without further adjustment because CFF is not guaranteed to factorise its (composite) input. Another remedy could be keeping the factor base fixed, to an initial segment of prime numbers, for example. Testing smoothness over a relatively-small factor base is much faster generally than fully factorising a number, even of half the order of magnitude of N, as in the case of $\langle p_i^2 \rangle$. However, this would necessitate more convergents to be verified as smooth, and so to be computed.

[21]The gigantic kernels that arise in certain attempts to factorise composite numbers made it infeasible to test this fairly for 10-digit numbers. `ker` is designed to halt after yielding 200 vectors.

A Programs

The programs take the form of five modules: `B.py`, `CF.py`, `Pell.py`, `Ker.py`, `CFF.py`. Program output printed in the report is generated by functions in `Output.py`, as annotated throughout the report. `InitB.py` is a script that, when run in a Python shell, loads all functions into the global memory.

A.1 Documentation

This section outlines the purpose of the project's functions.

`rf(n):` $n \in \mathbb{N}_0$. Returns $\lfloor\sqrt{n}\rfloor$ using integer arithmetic only.

`sm(a,b):` $a \in \mathbb{Z}$, $b \in \mathbb{N}_0$. Returns the representative of $\lfloor\sqrt{n}\rfloor$ modulo N in $(\frac{n}{2}, \frac{n}{2}]$.

`euclid(a,b):` $a, b \in \mathbb{N}_0$. Returns $\mathrm{hcf}(a, b)$,[22] from the remainder list of Euclid's algorithm.

`factor(n):` $n \in \mathbb{N}_0$. Returns a trial-division prime factorisation of n as a list with repeats.[23]

`fb(B,n):` $B \in \mathbb{P}^*$, $n \in \mathbb{N}_0$. Returns a trial-division prime factorisation of n if n is B-smooth; else, returns 0.

`cf(N,k):` $N, k \in \mathbb{N}_0$. Returns `[a,r,s]`, the sequences $(a_n)_{n=0}^k$, $(r_n)_{n=0}^k$, $(s_n)_{n=0}^k$ derived from the continued fraction expansion of $\sqrt{N}$.

`cfc(N):` $N, k \in \mathbb{N}_0$. Returns the closed-form variant of `cf(N,k)`.

`cfp(X); cfq(X):` Returns the p-convergents (`cfp`)/the q-convergents (`cfq`) of the continued fraction expansion X. the p-convergents modulo N (`cfpm`) of $\sqrt{N}$ truncated to $k+1$ terms.

`cfpm(N,k):` $N, k \in \mathbb{N}_0$. Returns the p-convergents modulo N from index 0 to k of the continued fraction expansion of $\sqrt{N}$.

`PellPQ(N,X):` $N \in \mathbb{N}_0$ with continued fraction expansion X of its square root. Returns the sequence $p_i^2 - Nq_i^2$.

`PellMinus(N):` $N \in \mathbb{N}$. Returns the truth value of the solvability of Pell$-$ for N.

`PellA(N,s,n):` $N \in \mathbb{N}$ non-square, $s \in \{\pm 1\}$, $n \in \mathbb{N}_0$. Returns n non-trivial solutions to Pell$s$ for N via convergent computation.

`PellB(N,s,n):` $N \in \mathbb{N}$ non-square, $s \in \{\pm 1\}$, $n \in \mathbb{N}_0$. Returns n non-trivial solutions to Pell$s$ for N via recursive unit generation.

`ker(A):` $A \in \mathrm{Mat}_{m,n}(\mathbb{F}_2)$. Yields first the dimension of $\ker(A)$, and then successive vectors in $\ker(A)$ each time it's called (usually via a for loop).

`cff(N,k):` $N, k \in \mathbb{N}_0$. Runs continued fraction factorisation with $k+1$ partial quotients and returns either some factors of N or an error message.

[22]Convenient convention: $\mathrm{hcf}(0, 0) = 0$; $0^0 = 1$

[23]In the case $n = 0$, `factor(n)` returns [0].

A.2 B

```
def rf(n):              # returns floor(sqrt(n))
    i = 1; x = 0
    while True:
        if (x + i) ** 2 <= n:
            i <<= 1
        elif i != 1:
            x += i >> 1; i = 1
        else:
            return x

def sm(a,b):            # symmetric a % b
    x = a % b
    if x > b // 2:
        x -= b
    return x

def euclid(a,b):        # Euclid's algorithm
    return a if b == 0 else euclid(b,a%b)

def factor(n):          # trial division factoriser
    x = []; r = 2; m = rf(n)
    while r <= m:
        if n % r == 0:
            n //= r; m = rf(n)
            x.append(r)
        else:
            r += 1
    if n != 1:
        x.append(n)
    return x

def fb(B,n):            # factor base algorithm
    if n == 0:
        return 0
    y = []
    while True:
        if n == 1:
            return y
        for r in B:
            if n % r == 0:
                n //= r
                y.append(r)
                break
        else:
            return 0
```

A.3 CF

```
from B import *

def cf(N,k):          # cf of sqrt(N) (repeating)
    d = rf(N)
    a = [d]; r = [0]; s = [1]
    if d ** 2 != N:
        for i in range(k):
            r.append(a[i] * s[i] - r[i])
            s.append((N - r[i+1] ** 2) // s[i])
            a.append((d + r[i+1]) // s[i+1])
    return [a,r,s]

def cfc(N):           # cf of sqrt(N) (non-repeating)
    d = rf(N)
    a = [d]; r = [0]; s = [1]
    if d ** 2 != N:
        i = 0
        while True:
            r.append(a[i] * s[i] - r[i])
            s.append((N - r[i+1] ** 2) // s[i])
            a.append((d + r[i+1]) // s[i+1])
            i += 1
            if a[i] == d << 1:
                break
    return [a,r,s]

def cfp(X):           # p convergents of sqrt(N) up to index k
    a = X[0]; p = [0,1]
    for i in range(len(a)):
        p.append(a[i]*p[i+1]+p[i])
    return p

def cfq(X):           # q convergents of sqrt(N) up to index k
    a = X[0]; q = [1,0]
    for i in range(len(a)):
        q.append(a[i]*q[i+1]+q[i])
    return q

def cfpm(N,k):        # p conv. of sqrt(N) mod N up to index k
    a = cf(N,k)[0]; p = [0,1]
    for i in range(len(a)):
        p.append((a[i]*p[i+1]+p[i])%N)
    return p
```

A.4 Pell

```
from CF import *

def PellPQ(N,X):      # p^2 - Nq^2
    p = cfp(X); q = cfq(X)
    return [p[i+2]**2-N*q[i+2]**2 for i in range(len(p)-2)]

def PellMinus(N):     # p^2 - Nq^2
    return len(cfc(N)[0]) % 2 == 0 if N != 1 else True

def PellA(N,s,n):
    l = len(cfc(N)[0]) - 1; b = l % 2
    if l == 0 or [b,s] == [0,-1] or n == 0:
        r = []
    else:
        if [b,s] == [0,1]:
            r = [i*l - 1 for i in range(1,n+1)]
        elif [b,s] == [1,1]:
            r = [2*i*l - 1 for i in range(1,n+1)]
        elif [b,s] == [1,-1]:
            r = [(2*i-1)*l - 1 for i in range(1,n+1)]
        X = cf(N,r[-1]); p = cfp(X); q = cfq(X)
    return [[p[t+2],q[t+2]] for t in r]

def PellB(N,s,n):
    X = PellA(N,s,min(1,n))
    if len(X) == 0:
        return X
    f = [X[0][0],X[0][1]]; x = f; y = [f]
    M = lambda a,b,N: [a[0]*b[0]+N*a[1]*b[1], a[0]*b[1]+a[1]*b
       [0]]
    if s == -1:
        f = M(f,f,N)
    for i in range(n-1):
        x = M(x,f,N)
        y.append(x)
    return y
```

A.5 Ker

```python
from B import *

def ker(A):
    m = len(A); n = len(A[0])
    B = [int(''.join([str(i) for i in A[i]]),2) for i in range(m)
       ]
    i = 0; j = 0; p = []
  # Convert matrix to row-echelon and find pivot variables p.
    while i < m and j < n:
        for k in range(i,m):
            if (B[k] >> (n-1-j)) & 1 == 1:
                B[i],B[k] = B[k],B[i]
                for l in range(i+1,m):
                    if (B[l] >> (n-1-j)) & 1 == 1:
                        B[l] ^= B[i]
                p.append(j)
                i += 1; j += 1
                break
        else:
            j+=1
  # Find free variables f and output how many there are.
    f = []
    for s in range(n):
        if s not in p:
            f.append(s)
    yield len(f)
  # Generate non-trivial kernel vectors.
    y = [0 for t in range(n)]
    for e in range(1,min(1 << len(f),201)):
      # Assign 0/1 to free variables of y as per e in binary.
        for i in range(len(f)-1,-1,-1):
            y[f[i]] = e & 1
            e >>= 1
      # Determine pivot variables of y by back-substituting.
        for k in range(len(p)-1,-1,-1):
            l = p[k]
            x = 0; b = B[k]
            for s in range(n-1,l,-1):
                x ^= y[s] & b
                b >>= 1
            y[l] = x
        yield y
    return
```

A.6 CFF

```
from B import *; from CF import *; from Ker import *

def cff(N,k):       # cf factoriser
  # Find p convergents, q = <p^2> and its TD factorisations.
    p = cfpm(N,k)[2:];                #print('p =',p,'\n')
    q = [sm(x**2,N) for x in p];      #print('<p^2> =',q,'\n')
    f = [factor(abs(x)) for x in q];  #print('f =',f,'\n')
  # Find factor basis B.
    g = [z for z in [y for x in f for y in x] if z != 0]
    B = []
    for i in g:
        if g.count(i) > 1 and i not in B:
            B.append(i)
    if len(B) == 0:
        print('No␣basis.'); return
    print('B␣=',B)
  # Find indices of B-numbers A.
    f = [fb(B,abs(x)) for x in q]
    A = [i for i in range(len(q)) if f[i] != 0 and q[i] != 1]
    if len(A) == 0:
        print('No␣non-trivial␣B-numbers.'); return
    print('A␣=',A,'\n')
  # Prepare selection matrix X.
    X = [[f[i].count(j) & 1 for i in A] for j in B]
    X += [[int(q[i] < 0) for i in A]]
    Y = ker(X)
    if next(Y) == 0:
        print('Trivial␣kernel.'); return
  # Test successive kernel vectors until success.
    for K in Y:
      # Determine selection I from kernel vector K
        I = [A[i] for i in range(len(A)) if K[i] == 1]
      # Test selection I
        x = 1; t = 1
        for i in I:
            x *= p[i]; t *= q[i]
        y = rf(t)
        if t != y**2: print('ERROR:␣t␣is␣not␣square.')
        x %= N; y %= N
        if (x**2 - y**2) % N != 0: print('ERROR:␣not␣x^2~y^2.')
        print('I␣=',I,';␣x␣=',x,';␣y␣=',y)
        a = [euclid(N,abs(x-y)),euclid(N,x+y)]
        a = [t for t in a if t not in [1,N]]
        if len(a) != 0:
           return a
  # If kernel is exhausted:
    print('No␣factors␣found.','\n')
```