Comparative Analysis of Primality Testing Algorithms for Enhanced Pseudoprimes Detection

Background

Modern cryptography algorithms like RSA or ElGamal rely on primality tests to generate large random primes.

Types of Primality Tests

Deterministic Primality Test

- Provide definite answers about the primality of a number
- Require more computational time for large numbers
- AKS primality test (2002) is polynomial time but not efficient enough in practice

Probabilistic Primality Test

- Do not provide definite answers about the primality of a number: there are certain numbers classified as primes but they are no, such numbers are called pseudoprimes
- Require less computational time for large numbers, often relies on sequences

Current Predicament

A deterministic primality test that is able to handle the cases needed for modern cryptography does not exist currently. With Quantum Computing changing modern cryptography, fundamental questions, such as finding efficient deterministic primality tests, are becoming more significant.

Procedure

1. Study the whole class of Lucas-Type Sequence
2. Change the parameters and the initial values for the special cases of Lucas-Type sequence
3. For each sequence, compute the first several pseudoprimes
4. Generate graphs that visualize the size and amount of initial pseudoprimes of Lucas-Type sequence
5. Identify and analyze special cases that stand out from these visualizations

Hypothesis

- Sizes of the smallest pseudoprimes is indicative of the quality of the primality tests (for instance the recent paper by someone)
- Many pseudoprimes are expected to be divisible by 2 and 3 In general, very little structure is expected.
- The quality of primality tests should be examined by more factors other than the number of pseudoprimes
- Not expecting to be able to predict the pseudoprimes by simple formulas

Lucas-Type Sequence

$V_{n+2} = c_1 V_{n+1} + c_0 V_n,\ V_0 = 2,\ V_1 = c_1$

If p is prime, then we have the congruence: $V_p \equiv V_1\ (mod\ p)$

```
def lucas_type_primality_test(n, c0, c1):
    a = 2
    b = c1
    for _ in range(n - 1):
        c = (c1 * b + c0 * a) % n
        a = b
        b = c
    return (b - c1) % n == 0
```

For instance, when $c_0 = 1$ and $c_1 = 2$, the sequence is:

2	2	6	14	34	82	198	478
V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7

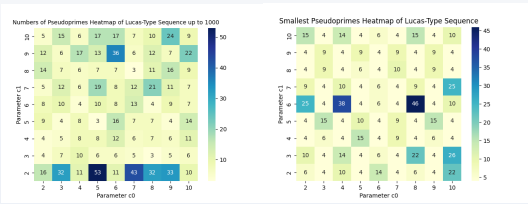
$v_3 = 14 \equiv 2 (\mathrm{mod}\ 3)$ ⟶ 3 is a prime

$v_6 = 198 \equiv 0 (\mathrm{mod}\ 6)$ ⟶ 6 is not a prime

$v_4 = 34 \equiv 2 (\mathrm{mod}\ 4)$ ⟶ 4 is a pseudoprime

The case where $c_0 = 5, c_1 = 2$ has a relatively high number of pseudoprimes.
The case where $c_0 = 8, c_1 = 6$ has a relatively big smallest pseudoprime.

*See if restricting the pseudoprimes by the multiples of 2 and 3 can significantly improve the quality of primality tests



The least promising primality test: Lucas-Type Sequence(5, 2)

Pseudoprimes of Lucas-Type Sequence(5, 2) up to 1,000:
[4, 6, 8, 9, 12, 16, 18, 24,..... 782, 864, 972] ⟶ 53 pseudoprimes

Pseudoprimes after filtered out with 2_rough_number:
[9, 27, 45, 81, 243, 405, 729, 759] ⟶ 8 pseudoprimes

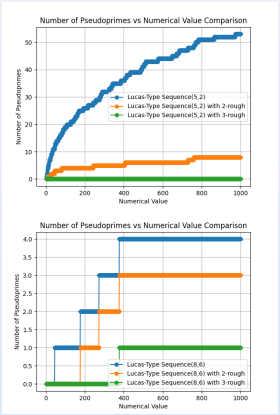
Pseudoprimes after filtered out with 3_rough_number:
[] ⟶ 0 pseudoprimes

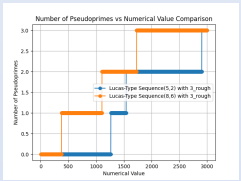
A promising primality test: Lucas-Type Sequence(8, 6)

Pseudoprimes of Lucas-Type Sequence(8, 6) up to 1000:
[46, 177, 273, 377] ⟶ 4 pseudoprimes

Pseudoprimes after filtered out with 2_rough_number:
[177, 273, 377] ⟶ 3 pseudoprimes

Pseudoprimes after filtered out with 3_rough_number:
[377] ⟶ 1 pseudoprimes

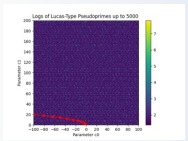


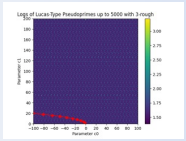


***A lot of pseudoprimes among the Luca—Type Sequence primality tests are the multiples of 2 and 3.**

***After pseudoprimes being restricted to 3-rough numbers, the Lucas-Type(5, 2) is actually a better primality test than Lucas-Type(8, 6)**

Lucas-Type Sequence with more parameters





Red curve highlights the special case of Lucas-Type Sequence, Fermat's special case, where $c_0 = -\alpha^2,\ c_1 = 2\alpha,\ \alpha \in \mathbb{Z}$.

Through the visualization with larger scales and parameters, it shows that restricting pseudoprimes with 3-rough numbers can effectively improve the quality of Lucas-Type Sequence primality tests.

Should I dig more into fermat here and explain its relationship With lucas-type?

Real-World Applications

Mention public key n is the product of two primes, and these primes have to be random, but we don't have a way to directly generate primes; we always have to generate number and test them with primality tests.

Conclusion

1. To determine the quality of these primality test is hard

2. One winner from the largest smallest pseudoprims
3. And one from the number of pseudoprimes
4. Reducing the multiples of two and three to improve the future primality test
5. There'll be a breakthrough if we can make the deterministic working with big numbers

For certain sequences, there are some patterns among these pseudoprimes

Future Direction

1. Longer recursion
2. Studying stronger congruences similar to how Miller Rabin is stronger than Fermat's little theorem
3. Implement a version of binary exponenatiation

Reference