

Grok 3 Think - Reasoning Engine termination analysis of Matiyasevich python program for Riemann Hypothesis





From

To

Date

<ksrinivasan@krishna-iresearch.org>
ka.shreenivaasan <ka.shreenivaasan@gmail.com>
2025-03-20 15:27

To determine whether the provided Python code terminates, we need to analyze its behavior, particularly the `while` loop that controls its execution. The loop runs as long as the condition `p**2 * (m - f0) < f3` is true, and we must assess whether this condition will eventually become false, causing the loop to exit, or if it remains true indefinitely.

Code Overview

The code initializes several variables:

- `h = m = p = 0`
- `d = f0 = f3 = n = q = 1`
- Empty lists: `lhs`, `rhs`, `lhsrhsratio`
- `sys.set_int_max_str_digits(100000)` to handle large integers
- `maxiterations = 3000`, but the iteration limit is commented out
- `integerdivision = True`

The loop performs updates to these variables and appends values to the lists, with the goal of evaluating the condition `p**2 * (m - f0) < f3` each iteration.

Loop Analysis

Let's break down the loop step by step to understand how the variables evolve.

Loop Condition

The loop continues while:

```
...
p**2 * (m - f0) < f3
...
```

- **Left-Hand Side (LHS)**:** `p**2 * (m - f0)`
 - `p` starts at 0 and increments only when `gcd(n, q) == 1`.
 - `m` is computed based on the number of times `q` can be divided by 2, multiplied by `d`.
 - `f0` is updated as `2 * n * h`, where `h` is the previous `f0`.
- **Right-Hand Side (RHS)**:** `f3`, which is updated as `(2 * n + 3) * f3`.

Variable Updates Inside the Loop

1. **Increment `n`**:** `n = n + 1` (starts at 1, so becomes 2 in the first iteration).
2. **Update `d`**:** `d = 2 * n * d - 4 * (-1)**n * h`.
3. **Compute GCD**:** `g = gcd(n, q)`.
4. **Update `q`**:**
 - Since `integerdivision` is `True`: `q = n * (q // g)`.
5. **Increment `p`**:** If `g == 1`, then `p = p + 1`.
6. **Compute `m`**:**
 - Start with `m = 0` and `g = q`.
 - While `g > 1`, divide `g` by 2 (`g = g // 2`) and add `d` to `m` (`m = m + d`).
7. **Update `h`**:** `h = f0`.
8. **Update `f0`**:** `f0 = 2 * n * h`.
9. **Update `f3`**:** `f3 = (2 * n + 3) * f3`.

Evolution of Key Variables

- **Initial Values**:**
 - `p = 0`, `m = 0`, `f0 = 1`, `f3 = 1`.
 - LHS = `p**2 * (m - f0) = 0 * (0 - 1) = 0`.
 - RHS = `f3 = 1`.
 - Condition: `0 < 1` is true, so the loop begins.
- **First Iteration**:**
 - `n = 2`, `h = 1`, `d = 2 * 1 * 1 - 4 * (-1)**1 * 0 = 2 - 0 = 2`.
 - `g = gcd(2, 1) = 1`.
 - `q = 2 * (1 // 1) = 2`.
 - `g == 1`, so `p = 1`.
 - `m`: `g = 2`, `g > 1`, so `g = 1`, `m = 0 + 2 = 2`.
 - `f0 = 2 * 2 * 1 = 4`.
 - `f3 = (2 * 2 + 3) * 1 = 7`.
 - LHS = `1**2 * (2 - 4) = 1 * (-2) = -2`.
 - RHS = `7`.
 - Condition: `-2 < 7` is true.
- **Second Iteration**:**
 - `n = 3`, `h = 4`, `d = 2 * 2 * 2 - 4 * (-1)**2 * 1 = 8 - 4 = 4`.
 - `g = gcd(3, 2) = 1`.
 - `q = 3 * (2 // 1) = 6`.
 - `p = 2`.
 - `m`: `g = 6`, `g = 3` (`m = 4`), `g = 1` (`m = 8`).
 - `f0 = 2 * 3 * 4 = 24`.
 - `f3 = (2 * 3 + 3) * 7 = 9 * 7 = 63`.

- LHS = `2**2 * (8 - 24) = 4 * (-16) = -64`.
- RHS = `63`.
- Condition: `-64 < 63` is true.

Long-Term Behavior

- **`f0` Growth**: $f_0 = 2 * n * h$, where h is the previous f_0 . This resembles factorial growth (e.g., $f_{0_n} \approx 2^n * n!$), making it increase extremely rapidly.
- **`m` Growth**: m depends on the number of times q can be halved, which is logarithmic in q . Since q grows as a product of n adjusted by GCD, m grows slower than f_0 .
- **$m - f_0$** : As f_0 outpaces m , this term becomes increasingly negative.
- **`p` Growth**: p increments when $\text{gcd}(n, q) == 1$, which happens frequently (e.g., when n is prime relative to q). However, $p**2$ is multiplied by a negative $m - f_0$, so LHS becomes more negative.
- **`f3` Growth**: $f_3 = (2 * n + 3) * f_3$, a product of terms like 5, 7, 9, etc., growing faster than exponentially.

Termination Assessment

- **LHS**: $p**2 * (m - f_0)$ is negative because $m - f_0$ is negative, and its magnitude increases as f_0 dominates m .
- **RHS**: f_3 is positive and grows rapidly.
- **Condition**: A negative LHS is always less than a positive RHS, and as iterations continue, LHS becomes more negative while RHS increases, reinforcing the inequality.

There's no mechanism to make $p**2 * (m - f_0) >= f_3$:

- f_0 's factorial-like growth ensures $m - f_0$ stays negative.
- p 's occasional increments cannot overcome the negativity.
- f_3 's rapid positive growth keeps RHS ahead.

Commented-Out Limit

The code includes a commented-out section:

```
```python
if n > maxiterations:
plt.plot(lhsrhsratio)
plt.show()
print("Stopping after ",maxiterations," iterations")
exit(1)
```
```

If active, this would stop the loop after 3000 iterations. Since it's commented out, the loop relies solely on the condition $p**2 * (m - f_0) < f_3$.

Conclusion

Without the iteration limit, the loop condition remains true indefinitely due to LHS becoming increasingly negative and RHS growing positively. Theoretically, the code does not terminate. In practice, it might halt due to memory limits or integer overflow (despite `sys.set_int_max_str_digits`), but based on its logic, it runs forever.

Thus, the Python code does not terminate.