CS3430 S26: Scientific Computing
My Observations on HW 5, Problem 4 – Pushing Digits of Pi

Vladimir Kulyukin

HW 5, Problem 4: Deep Convergence, Precision Budgeting, and Verification

In this problem, I used the binary-splitting Chudnovsky algorithm together with arbitrary-precision arithmetic (`mpmath`) to compute decimal digits of π. The goal was not only to approximate π, but to determine when those approximations can be trusted.

Correctness was verified against a reference file containing 99,999 digits of the mantissa of π. Instead of comparing long digit strings directly, I used SHA-256 hashes to detect mismatches efficiently and unambiguously.

Methodology

I generated successive approximations of π using the generator `pi_chudnovsky_bs_mp`, consuming one value per Chudnovsky term. For each experiment, I fixed three parameters:

- the number of Chudnovsky terms consumed,
- the working precision `mp.dps`,
- the number of requested decimal digits of the mantissa.

After computing π at the chosen precision, I extracted the first n digits of its decimal mantissa and compared the hash of those digits to the hash of the corresponding prefix from the reference file.

Experimental Results

With 15 Chudnovsky terms and `mp.dps = 250`, the computed mantissa matched the reference for the first 200 digits. However, when I increased the request to

500 digits using 40 terms and `mp.dps = 550`, the hash no longer matched the reference on my hardware.

The numerical value of π continued to converge, but the least significant decimal digits became incorrect. This mismatch was detected immediately by the hash comparison.

Numerical Epistemology: When Is a Number Correct?

This experiment highlights a central issue in numerical epistemology: when do we know that a computed number is correct?

In mathematics, π is defined independently of any computation. In scientific computing, however, we only ever interact with finite approximations. A numerical algorithm may converge in a mathematical sense while still producing incorrect decimal digits due to insufficient precision.

In this case, the Chudnovsky series converged extremely fast, but correctness was limited by the numerical representation and the chosen working precision. As a result, correctness became a property of the entire computational pipeline, not just of the algorithm itself.

Precision Budgeting

This problem demonstrates that precision is not something to maximize blindly. It is something to budget deliberately.

Although increasing `mp.dps` extends the range of correctness, it also increases computational cost and memory usage. More importantly, correctness still depends on matching the working precision to the number of requested decimal digits and to the behavior of intermediate computations, including binary-to-decimal conversion.

A principled numerical workflow therefore:

- estimates how many correct digits an algorithm can produce per iteration,
- chooses a working precision with sufficient guard digits,
- verifies correctness empirically,
- and increases precision only when needed.

Simply setting `mp.dps` to a very large value may work for small examples, but it is inefficient and unprincipled for real scientific problems.

Anticipating Questions

Q: Why not just set `mp.dps` to a very large value and keep going? Answer: Increasing `mp.dps` does allow more digits to be computed correctly, but it does not remove the need for reasoning about precision. Larger precision increases computational cost and still requires justification. In scientific computing, correctness comes from deliberate precision choices, not from unbounded precision.

Q: If the Chudnovsky series is still converging, why do the digits become wrong? Answer: Convergence refers to the real-number value, not to the correctness of its decimal expansion. If working precision is insufficient, rounding and conversion errors can corrupt the least significant digits even though the value appears numerically stable.

Q: Why use hashes instead of comparing digit strings directly? Answer: Hashes provide a scalable and reliable way to verify correctness for very long digit strings. A single incorrect digit produces a completely different hash, making errors easy to detect without expensive string comparisons.

Conclusion

This problem shows that fast convergence alone does not guarantee correctness. Numerical algorithms must be paired with appropriate precision budgeting and explicit verification. Hash-based comparison makes it possible to test correctness at a scale far beyond what can be inspected directly, reinforcing the idea that scientific computing is as much about epistemic discipline as it is about algorithms.