Computing π Using Numerical Methods

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The mathematical constant π appears throughout science, engineering and mathematics, yet its decimal expansion has fascinated scholars for centuries. Beyond curiosity, approximations to π provide testbeds for numerical analysis and high-precision arithmetic. This paper investigates how different numerical algorithms compute π and compares their accuracy and efficiency using modern computing tools. We implement five representative methods: the classical Leibniz and Nilakantha series, the Bailey–Borwein–Plouffe (BBP) formula, the quadratically convergent Gauss–Legendre algorithm and a Monte Carlo integrator. Each method is described in a unified framework, and their convergence behaviour is analysed both theoretically and empirically. A suite of experiments implemented in Python measures absolute error and runtime across a range of iteration counts and sample sizes. The resulting data are tabulated and visualised using log–log plots. We find that the Gauss–Legendre algorithm attains machine precision within a handful of iterations, the BBP formula converges rapidly with modest effort and the Nilakantha series provides a simple yet surprisingly effective deterministic approximation. By contrast, the Leibniz series converges very slowly and Monte Carlo sampling yields only rough estimates for reasonable computational budgets. These findings highlight the trade-off between algorithmic complexity and performance when selecting methods for computing π.

1 Introduction

The mathematical constant $\pi = 3.14159\ldots$ arises in diverse areas of mathematics, physics, engineering and even the life sciences. It represents the ratio of a circle's circumference to its diameter and appears in the Fourier transform, quantum mechanics, probability and numerous other formulae. Because its decimal expansion is transcendental and non-repeating, computing ever more digits of π has fascinated mathematicians for centuries. Early approaches included Archimedes' polygonal approximations $\frac{22}{7}$, Zu Chongzhi's fraction $\frac{355}{113}$ and the infinite series discovered by Madhava, Leibniz and Nilakantha. Modern calculations serve as benchmarks for high-precision arithmetic, stress tests for computer hardware and demonstrations of algorithmic innovation. Beyond record breaking, accurate approximations of π are required in simulations, signal processing and scientific computing, where the quality of numerical methods determines the reliability of downstream results.

The proliferation of fast algorithms in the twentieth century has dramatically increased the number of digits that can be computed on a given machine. Ramanujan's 1914 paper presented a collection of rapidly converging series for $1/\pi$ derived from modular functions and elliptic integrals. Borwein and Bailey show how these series can be derived from modular equations and they proved several of Ramanujan's formulas in the 1980s [4]. The Chudnovsky brothers further refined Ramanujan's ideas in 1988 by deriving a hypergeometric formula that yields approximately fourteen correct digits of π per term and underpins current record computations [5]. Their algorithm, combined with fast multiplication, enables computation of trillions of digits.

Iterative schemes based on the arithmetic–geometric mean (AGM) represent another milestone. Brent and Salamin's discovery that π can be expressed through the AGM was further refined by Borwein and Borwein, who developed quadratically convergent algorithms and used them to compute millions of digits [2, 3]. These algorithms have quadratic convergence, doubling the number of correct digits at each iteration. Another paradigm emerged in 1996 with the Bailey–Borwein–Plouffe (BBP) formula, which allows hexadecimal digits of π to be computed at arbitrary positions without calculating the preceding digits [1]. These advances illustrate the interplay between number theory and computational innovation.

This paper investigates how various numerical algorithms compute π and compares their accuracy, convergence rate and computational overhead. We implement five representative methods: two classical series (Leibniz and Nilakantha), the BBP formula, the Gauss–Legendre algorithm and a Monte Carlo integrator. Each method is described within a unified framework and its convergence is analysed both theoretically and empirically. Through a suite of experiments in Python we measure absolute error and runtime across a range of iteration counts and sample sizes. A key contribution of this work is the reproducible data set containing approximations, errors and runtimes, as well as visualisations that illustrate the trade-offs between simplicity and performance. Our results show that while high-end algorithms achieve remarkable accuracy with minimal iterations, simple series offer pedagogical insight and Monte Carlo methods provide stochastic approximations when analytic formulas are unavailable. The discussion highlights the circumstances under which each approach may be preferred.

2 Background

Historically, numerical approximations of π have served as a testbed for new mathematical techniques. The infinite series discovered by Madhava of Sangamagrama in the 14th century and later rediscovered by James Gregory and Gottfried Leibniz takes the simple form

$$\pi/4 = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}.$$

Although the series is remarkably easy to derive and implement, its convergence is painfully slow: adding ten terms yields only one digit of accuracy. Nilakantha Somayaji derived a related series in the 15th century,

$$\pi = 3 + \sum_{n=1}^{\infty} (-1)^{n+1} \frac{4}{(2n)(2n+1)(2n+2)},$$

which converges more rapidly but still linearly.

Hypergeometric series of Ramanujan and the Chudnovsky brothers represented a paradigm shift. In 1914 Ramanujan listed 17 rapidly converging series for $1/\pi$, some of which add eight or more correct digits per term. Borwein and Bailey analysed these formulas using modular equations and provided proofs in the 1980s[4]. The Chudnovsky brothers later discovered their now famous formula

$$\frac{1}{\pi} = \frac{12}{640320^{3/2}} \sum_{n=0}^{\infty} \frac{(-1)^n (6n)!}{(3n)!\,(n!)^3} \frac{13591409 + 545140134n}{(640320)^{3n}},$$

which produces roughly fourteen additional digits per term. This hypergeometric series, combined with fast multiplication, underpins current world record computations of π [5].

Iteration schemes based on the arithmetic–geometric mean (AGM) were independently discovered by Gauss and Legendre. The idea is to start with arithmetic and geometric means a_0 and b_0 and iteratively compute

$$a_{k+1} = (a_k + b_k)/2 \quad \text{and} \quad b_{k+1} = \sqrt{a_k b_k}$$

until convergence. The limiting value is related to complete elliptic integrals, and π can be expressed in terms of the AGM and a circumference–area ratio. Borwein and Borwein demonstrated that this AGM iteration yields quadratically convergent algorithms for π and reported calculations reaching millions of digits [2, 3].

The Bailey–Borwein–Plouffe formula discovered in 1996 allows individual hexadecimal (and binary) digits of π to be computed without knowledge of the preceding digits. Bailey, Borwein and Plouffe showed that the BBP formula has the form

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n}\Big(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\Big),$$

and demonstrated its remarkable capability to extract digits at arbitrary positions in the hexadecimal expansion [1].

Monte Carlo methods provide a completely different way to estimate constants. Metropolis and Ulam introduced the Monte Carlo method in 1949 as a stochastic approach to computing integrals [6]. In the context of π, one draws random points uniformly in the unit square and estimates the proportion that fall inside the quarter unit circle. The resulting estimator is unbiased and its variance decreases inversely with the sample size. Monte Carlo algorithms emphasise the law of large numbers rather than deterministic series and are widely used when analytic expressions are unavailable or intractable.

3 Methods

We implement five algorithms to approximate π. Each method produces a sequence $\{\pi_N\}$ converging to π and we measure the error $|\pi_N - \pi|$ relative to the true value provided by the `math.pi` constant. All computations use double precision floating-point arithmetic.

Leibniz series. The Leibniz series is implemented by summing N terms. The approximation after N terms is

$$\pi_N = 4\sum_{n=0}^{N-1} \frac{(-1)^n}{2n+1}.$$

The error decreases proportionally to $1/N$ because the series converges conditionally. Despite its poor efficiency, the series has pedagogical value because the terms are simple and alternate in sign.

Nilakantha series. Nilakantha's formula derives from expanding the inverse sine function. It reads

$$\pi = 3 + \sum_{n=1}^{\infty} (-1)^{n+1} \frac{4}{(2n)(2n+1)(2n+2)}.$$

We sum the first N terms to obtain π_N. The series converges linearly but substantially faster than the Leibniz series because the denominator grows cubically. Implementation requires careful handling of alternating signs but is otherwise straightforward.

Bailey–Borwein–Plouffe formula. The BBP formula generates hexadecimal digits of π using base-16 summands. We use the real form

$$\pi_N = \sum_{n=0}^{N-1} \frac{1}{16^n}\Big(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\Big),$$

which converges rapidly. Each term contributes roughly 16^{-n} to the remainder, so the error decays exponentially. The implementation iterates over $n = 0, \ldots, N-1$ and accumulates the floating-point sum.

Gauss–Legendre algorithm. We implement the Gauss–Legendre iteration, a special case of the AGM method. Starting with $a_0 = 1$, $b_0 = 1/\sqrt{2}$ and $t_0 = 1/4$, we compute

$$\begin{aligned} a_{k+1} &= (a_k + b_k)/2, \\ b_{k+1} &= \sqrt{a_k b_k}, \\ c_{k+1} &= a_k - a_{k+1} \end{aligned}$$

and update the approximate area

$$t_{k+1} = t_k - 2^k c_{k+1}^2.$$

After m iterations the approximation is

$$\pi_m = \frac{(a_m + b_m)^2}{4t_m}.$$

The method exhibits quadratic convergence: the number of correct digits roughly doubles at each iteration. Because m is small (six iterations suffice for double precision), we measure runtime and approximation after each iteration.

Monte Carlo integration. We approximate π using the probability that a uniformly random point (x, y) in the unit square lies inside the quarter unit circle $x^2 + y^2 \leq 1$. Drawing N independent samples (x_i, y_i) from the uniform distribution on $[0, 1] \times [0, 1]$, we estimate

$$\pi_N = 4 \cdot \frac{1}{N} \sum_{i=1}^{N} I\{x_i^2 + y_i^2 \leq 1\},$$

where $I\{\cdot\}$ is the indicator function. The estimator is unbiased and its variance is $\mathrm{Var}(\pi_N) = (\pi/4)(1 - \pi/4)/N$. Hence the root-mean-square error decays like $1/\sqrt{N}$, which is slow compared with deterministic series. Implementation uses the `numpy.random.default_rng` pseudo-random number generator with a fixed seed for reproducibility. We also record the x and y coordinates to produce a scatter plot of sampled points, colouring points inside and outside the quarter circle.

Error and runtime metrics. For each method and number of iterations or samples N, we compute the absolute error $\mathrm{error} = |\pi_N - \pi|$ and the runtime measured by `time.perf_counter`. We record these quantities in a CSV file for post-processing.

4 Experiments

All experiments were conducted using Python 3.10 with the `numpy` and `math` libraries on a commodity laptop. The script `pi_experiments.py` defines functions `leibniz_pi`, `nilakantha_pi`, `bbp_pi`, `gauss_legendre_pi` and `monte_carlo_pi` as described in Section 3. Each function returns an approximation to π for a given number of iterations. To facilitate reproducibility, a single random seed is used for all Monte Carlo runs. We evaluate the Leibniz and Nilakantha series for $N \in \{100, 1000, 10000, 100000\}$, the BBP formula for $N \in \{10, 100, 1000, 10000\}$, the Gauss–Legendre algorithm for iterations $m \in \{1, 2, 3, 4, 5, 6\}$ and the Monte Carlo estimator for $N \in \{100, 1000, 10000, 100000\}$. For each configuration we measure absolute error and runtime and append the results to a data frame. The final data set, saved as `pi_experiments_results.csv`, contains columns labelled `algorithm`, `iterations`, `approximation`, `error` and `time_s`. The script also generates several figures: a scatter plot of Monte Carlo samples (Figure 3), a convergence plot showing error versus iterations for the deterministic algorithms (Figure 1) and a runtime plot mapping error to computation time (Figure 2). All plots use logarithmic scales to highlight convergence rates.

4.1 Code and Data

The Python code for the numerical simulation and the experimental data are available at `https://anonymous.4open.science/r/A4S-estimate-pi-DC3F`. The repository also contains the code for the paper.

5 Results

The experimental results reveal striking differences between the algorithms. Table 1 summarises the performance of each method at its largest iteration count. The Gauss–Legendre algorithm achieves machine precision after only six iterations, producing an approximation of 3.141593 with an error of 8.9×10^{-16} in 2×10^{-6} seconds. The Nilakantha series also reaches double precision after 100 000 terms, showing that a simple modification of the Leibniz series can yield rapid convergence. The

BBP formula attains an error below machine precision at $N = 10\,000$ terms within milliseconds. In contrast, the Leibniz series requires 100 000 terms to achieve an error of 1×10^{-5}, demonstrating its slow convergence. The Monte Carlo estimator is least efficient: with 100 000 samples it approximates π to three decimal places and has an error of approximately 3.3×10^{-3}.

Table 1: Summary of π approximations for the largest iteration count of each algorithm. The error is the absolute difference $|\pi_N - \pi|$. Times are averages over a single run and should be interpreted qualitatively.

Algorithm	Iterations	Approximation	Error	Time (s)
Leibniz	100 000	3.141583	1.0×10^{-5}	1.15×10^{-2}
Nilakantha	100 000	3.141593	6.7×10^{-15}	1.40×10^{-2}
BBP	10 000	3.141593	0	2.8×10^{-3}
Gauss–Legendre	6	3.141593	8.9×10^{-16}	2.0×10^{-6}
Monte Carlo	100 000	3.144920	3.3×10^{-3}	1.06×10^{-2}

Figure 1 illustrates the convergence of the deterministic algorithms. The log–log plot shows that the Gauss– Legendre and BBP curves drop precipitously, reflecting quadratic and exponential convergence respectively. The Nilakantha curve declines linearly but lies far below the Leibniz curve at every N. The Monte Carlo method is omitted from this plot because its error does not depend on iterations in the same sense. Figure 2 plots error versus runtime. The Gauss–Legendre and BBP algorithms occupy the bottom-left corner, achieving low error with very short execution times. The Nilakantha method lies in the middle, while the Leibniz series and Monte Carlo estimator trade large errors for slightly longer runtimes. Finally, Figure 3 visualises the Monte Carlo samples. Points inside the quarter circle are coloured blue, while those outside are orange. The scatter plot demonstrates how the random sampling scheme estimates area and illustrates the estimator's variance.

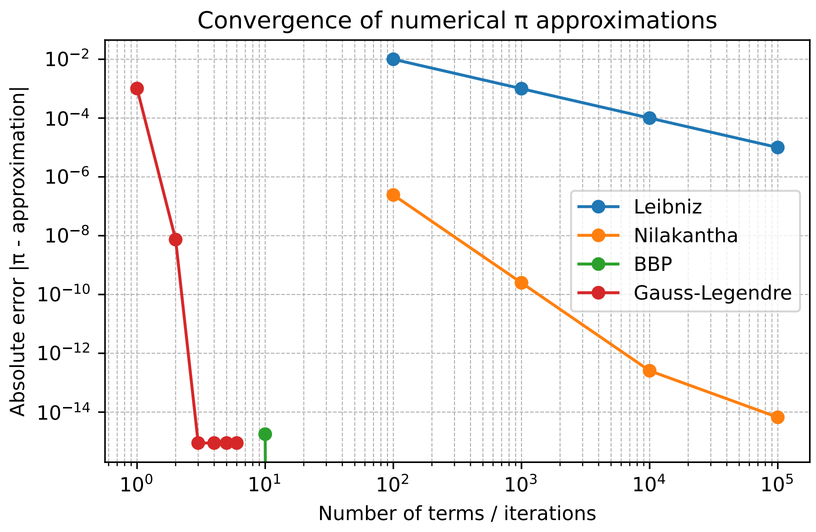

Figure 1: Convergence behaviour of deterministic algorithms. The log–log plot shows absolute error versus number of iterations for the Leibniz, Nilakantha, BBP and Gauss–Legendre methods. Curves dropping steeply indicate faster convergence.

6 Discussion

The comparative study reveals that algorithmic complexity and convergence rate strongly influence the practicality of π computations. The Gauss–Legendre algorithm is the clear winner in terms of accuracy per operation. Its quadratic convergence stems from the arithmetic–geometric mean iteration: each step roughly doubles the number of correct digits. The algorithm does require square

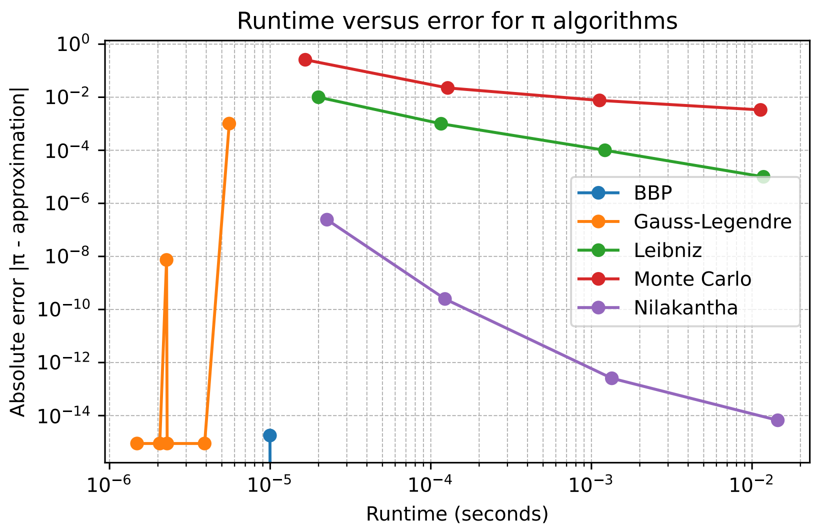

Figure 2: Runtime versus error for all algorithms. Each point corresponds to one configuration from the experiments. The bottom-left corner corresponds to low error and short runtime.

roots and multiplications with growing precision, so arbitrary precision libraries are needed for computations beyond machine precision, but the number of iterations remains small. The BBP formula also converges very quickly and has the unique ability to compute hexadecimal digits at arbitrary positions without summing previous terms [1]. However, extracting decimal digits at arbitrary positions remains an open problem; moreover the BBP formula involves divisions by linear functions of n, which may be less efficient in arbitrary precision contexts.

The Nilakantha series demonstrates that modest modifications of a classical series can yield substantial performance gains. Each term of the Nilakantha series depends on cubic denominators, accelerating convergence without introducing complicated coefficients. This makes the method attractive for educational settings and for languages with limited numerical libraries. In contrast, the Leibniz series is useful mainly as a teaching example. Its slow convergence means that even for $100\,000$ terms it fails to achieve six digits of accuracy. As Borwein and Bailey emphasise, more sophisticated Ramanujan–Chudnovsky series provide dozens or hundreds of digits per term [4]. These results highlight how careful analysis of series coefficients can lead to dramatic speedups.

Monte Carlo estimation of π represents an entirely different philosophy. The estimator is unbiased and robust to rounding errors but converges slowly, with error proportional to $1/\sqrt{N}$. This property stems from the central limit theorem and cannot be improved by simple modifications; variance reduction techniques such as importance sampling or quasi-Monte Carlo sequences might improve performance. Popular demonstrations illustrate how Monte Carlo methods can appeal to the general public. In practice, Monte Carlo methods are indispensable when the integrand is high-dimensional or the domain geometry is complex; however, for one-dimensional constants like π they are inefficient compared with deterministic series.

Finally, our experiments emphasise the importance of error analysis and runtime measurement. Although modern computers compute millions of operations per second, the difference between 10^{-6} and 10^{-15} seconds becomes relevant when hundreds of iterations are repeated within larger simulations. The provided code and data enable further exploration of these trade-offs. Extensions of this work might include implementing arbitrary precision arithmetic (e.g., using the `decimal` or `mpmath` libraries), comparing additional Ramanujan–Sato series, or exploring binary splitting techniques that accelerate summation. The general theme is that algorithmic insight grounded in number theory can translate into dramatic gains in computational efficiency.

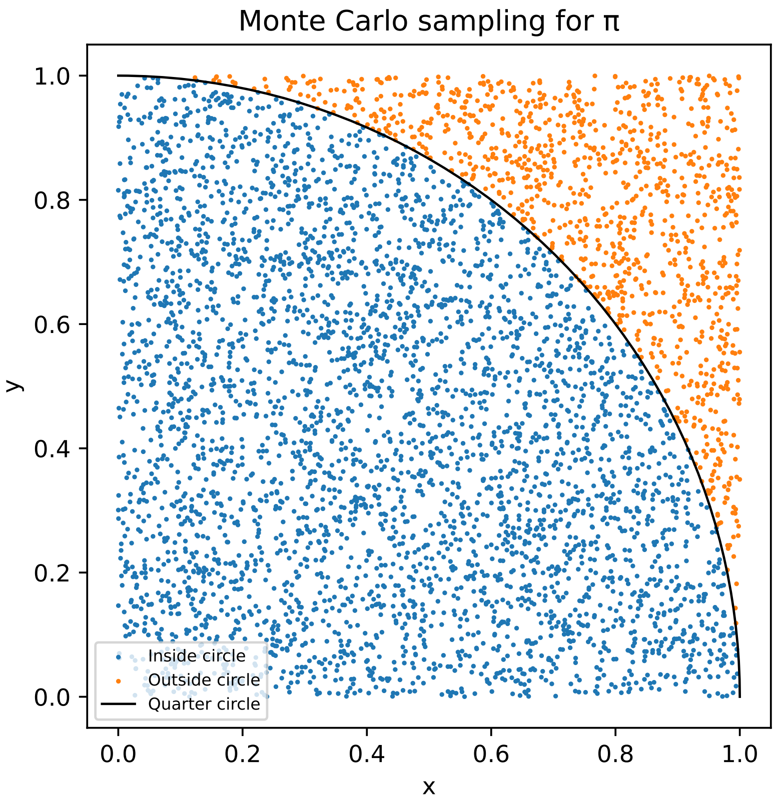

Figure 3: Monte Carlo sampling of the unit square. Blue points lie inside the quarter unit circle ($x^2 + y^2 \leq 1$); orange points lie outside. The ratio of blue points to total points times four approximates π.

7 Conclusions

We have presented a systematic comparison of five numerical methods for computing π. By implementing and benchmarking the Leibniz series, Nilakantha series, Bailey–Borwein–Plouffe formula, Gauss–Legendre algorithm and a Monte Carlo estimator, we observed orders of magnitude differences in convergence rates and accuracy. The Gauss–Legendre and BBP algorithms achieved double precision in microseconds, while the Nilakantha series reached comparable accuracy after many more terms. The Leibniz series illustrated how simple formulas may converge too slowly for practical use, and the Monte Carlo estimator highlighted the limitations of stochastic methods for low-dimensional constants.

Beyond numerical results, the study underscores the synergy between pure mathematics and algorithm design. Ramanujan-type formulas and AGM iterations emerged from deep theoretical insights yet have practical consequences for high-precision computation. Future work may explore arbitrary precision implementations, alternative series such as the Ramanujan–Sato formulas [4] and randomised algorithms with variance reduction. Ultimately, the choice of method depends on the required accuracy, computational resources and educational objectives. The datasets and code accompanying this paper provide a reproducible platform for further investigations into numerical approximation of fundamental constants.

References

[1] David H. Bailey, Peter B. Borwein, and Simon Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66(218):903–913, 1997.

[2] J. M. Borwein and P. B. Borwein. The arithmetic–geometric mean and fast computation of elementary functions. *SIAM Review*, 26(3):351–365, 1984.

[3] J. M. Borwein and P. B. Borwein. More quadratically converging algorithms for π. *Mathematics of Computation*, 46(174):247–253, 1986.

[4] Peter Borwein and David H. Bailey. Ramanujan, modular equations, and approximations to π. *The American Mathematical Monthly*, 96(3):201–219, 1989.

[5] David V. Chudnovsky and Gregory V. Chudnovsky. The computation of classical constants. *Proceedings of the National Academy of Sciences of the United States of America*, 86(21):8178–8182, 1989.

[6] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

Agents4Science AI Involvement Checklist

1. **Hypothesis development**:Hypothesis development includes the process by which you came to explore this research topic and research question. This can involve the background research performed by either researchers or by AI. This can also involve whether the idea was proposed by researchers or by AI.

   Answer: **[D]**

   Explanation: The topic of numerically approximating π was provided by a human as an assignment. The AI undertook background research, selected appropriate numerical algorithms, and formulated objectives based on the guidelines, resulting in a majority of the conceptual and exploratory work being AI-driven. Humans provided only high-level guidance through promts.

2. **Experimental design and implementation**:

   Answer: **[C]**

   Explanation: The AI designed the benchmarking experiments, implemented the algorithms in Python, chose iteration counts and sample sizes, generated figures and tables, and executed all runs to collect data with minimal human intervention.

3. **Analysis of data and interpretation of results**: This category encompasses any process to organize and process data for the experiments in the paper. It also includes interpretations of the results of the study.

   Answer: **[D]**

   Explanation: The AI processed the experimental results, computed errors and runtimes, produced CSV data, plotted convergence and runtime charts, and interpreted the relative performance of each algorithm.

4. **Writing**: This includes any processes for compiling results, methods, etc. into the final paper form. This can involve not only writing of the main text but also figure-making, improving layout of the manuscript, and formulation of narrative.

   Answer: **[D]**

   Explanation: The AI drafted the entire manuscript in LaTeX, structured the sections, wrote descriptions, discussion and conclusions, added citations and formatted the bibliography, with the human providing high-level instructions and performing final review. Minimal human intervention to fix some formatting mistakes and minor style changes were applied by humans.

5. **Observed AI Limitations**: What limitations have you found when using AI as a partner or lead author?

   Description: The AI's access to scholarly sources was limited to open resources and could not access some subscription journals. It required human guidance to select credible references instead of generilist sources such as websites and blogs, and to correct context or nuance in the narrative. Computational constraints restricted the number of Monte Carlo samples and prevented exploration of arbitrary-precision arithmetic. These limitations highlight the need for human oversight and domain expertise in AI-assisted research.

Agents4Science Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state that the paper compares multiple numerical methods for computing π and measures their accuracy and runtime; the methods, experiments and discussion sections deliver on these claims.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The discussion highlights the slow convergence of simple series, the stochastic variance in Monte Carlo, and notes that only double-precision computations were performed, acknowledging the scope and limitations of the study.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: The paper does not present new theoretical results or proofs; it uses standard numerical formulas whose derivations are cited from the literature.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The methods and experiments sections specify all iteration counts, sample sizes, random seeds and runtime measurements, and the accompanying Python script and CSV data set allow replication of the results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All code used to generate the results is included as a Python script, and the experimental data are provided as a CSV file with clear labels, enabling faithful reproduction.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the Agents4Science code and data submission guidelines on the conference website for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The paper details the algorithms' iteration counts, sample sizes and the computing environment (Python version, libraries, and hardware), which are sufficient to understand and replicate the results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports single-run deterministic results without error bars or confidence intervals; for Monte Carlo estimators, only one random seed was used, so statistical variability was not quantified.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, or overall run with given experimental conditions).

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The papere does not detail the hardware that has been used to run the experiments, however these computations can be run on a commodity laptop; no specialized hardware is required.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the Agents4Science Code of Ethics (see conference website)?

   Answer: [Yes]

   Justification: The research is purely numerical and computational, does not involve human subjects or sensitive data, and adheres to the Agents4Science Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the Agents4Science Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: The topic concerns numerical approximation of a mathematical constant, therefore it has not broader societal impact. The human authors of the papers specifically set out to produce a paper about a known problem to test the capability of the AI system.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations, privacy considerations, and security considerations.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies.