Primes Project

MATH 242 • Spring 2026

Due: Friday, April 17

(Following the due date and initial grading, there will be an opportunity to revise and resubmit for a higher grade.)

This project has two parts, both involving prime numbers.

Part 1: Implement the Sieve of Sundaram and compare it with the Sieve of Eratosthenes. The Sieve of Sundaram algorithm is found on pages 156–158 in the *Exploring Mathematics* text (see especially Algorithm 4.3). Demonstrate that your implementation works. Compare the runtime required for computing lists of primes with each sieve method. You should measure the runtime of each sieve for computing the primes up to n, for various n. Then make a plot that shows your results (n on the horizontal axis, runtimes on the vertical axis).

Part 2: Investigate at least two (three for a score of *Excellent*) of the following conjectures about primes. For each, either provide computational evidence in support of the conjecture, or find a counterexample showing that the conjecture is false.

Do not look up information about these conjectures. This project is about what *you* can discover from your own computational exploration, not about what information exists on the internet.

- **Conjecture A.** Every even integer greater than 2 is the sum of two primes.
- **Conjecture B.** For every integer $N > 2$, the number of positive integers less than N with an *even* number of prime factors is less than or equal to the number of positive integers less than N with an *odd* number of prime factors. For this, prime factors are counted *with multiplicity*; for example, $24 = 2^3 \cdot 3$ has 4 prime factors, while $588 = 2^2 \cdot 3 \cdot 7^2$ has 5 prime factors.
- **Conjecture C.** For every positive integer n, there exists at least one prime between n^2 and $(n+1)^2$.
- **Conjecture D.** All odd numbers greater than 1 are either prime, or can be expressed as the sum of a prime and twice a square.

For Part 2, you may use code from class, your own code, and number-theoretic functions supplied by Sage. Do not use artificial intelligence tools to write code.

As you investigate these conjectures, it's important to use print statements to confirm that your code is really checking the cases that you intend to check. Use printed output to show that your procedures works for small cases, and then hide the output when running your experiments on larger cases. Try to push the limits of the computational power available to you. For example, don't just test 100 cases and declare that the conjecture must be true. Can you test one million cases? One billion? Of course, you should stop if you find a counterexample, since a single counterexample shows that the conjecture is false.

For projects in MATH 242, *communication* is as important as *computation*. You should turn in a well-organized notebook that clearly explains, using sentences and paragraphs, what you computed and what conclusions you can draw.

This project will be graded on the EMRN scale, as described in the syllabus. To receive a grade of *Meets Expectations*, your notebook should exhibit the following characteristics:

- You implement the Sieve of Sundram and compare its efficiency with the Sieve of Eratosthenes.
- You investigate two of the conjectures above. For each, you either find evidence in support of the conjecture or a counterexample that disproves it.
- Your code is appropriate for the given tasks and produces reasonable output. You provide sample input and output which clearly shows that your code works as desired.
- Your reasoning is explained using sentences, and your notebook is well-formatted and easy to read.
- No significant gaps or errors are present.

To receive a grade of *Excellent*, your notebook should further exhibit the following:

- You investigate *three* of the above conjectures. For each, you either find substantial evidence in support of the conjecture or a counterexample that disproves it.
- Computational methodology demonstrates mastery of the computational techniques that we have studied in this course.
- Your code is of high quality, demonstrating skillful use of programming constructs (e.g., variables, lists, functions, modules).
- Exposition is clear and precise, thoroughly explaining your methodology and reasoning. Discussion should things like assumptions made designing your methods, limitations of your computational techniques, and possible extensions for future study.
- The work must extend beyond the project requirements in a creative or insightful direction.