Using Mathematica in a Course on Number Theory

Kenneth Kramer
Queens College (CUNY)
Flushing, NY 11367
kramer@qcvaxa.qc.edu

Mathematica was introduced in the undergraduate number theory course at Queens College (City University of NY) during the Spring 1997 semester, to encourage the discovery of mathematical ideas through guided experiments. Mathematica offered the capacity for rapid calculation, infinite precision arithmetic and symbolic algebra. Our goal was to present Mathematica to students with limited or no computer experience. The first laboratory projects acquainted students with the basic syntax of Mathematica and illustrated the creation of Mathematica procedures, known as modules. By modifying these simple modules, students were able to conduct experiments and test conjectures. As students progressed, they also designed their own modules. Not only did students learn the rudiments of computer programming, but they also gained more realistic computational experience with some of the algorithms of number theory. Because the machine does not award partial credit when syntactical or logical errors are present, students became more sensitive to the need for precision in formulating their algorithms.

Students who were not familiar with computers were asked to spend an hour in the lab, going through the standard guided tour to the Macintosh provided by Apple. It was expected that they would be familiar with the keyboard, the mouse and the basic ideas of point and click technology before the class officially met in the laboratory. One or two hours of class time were

alloted for each project, and students were expected to spend two to four hours outside of class to complete their projects. Descriptions of some of these projects follow.

Introduction to Mathematica. Examples were given to illustrate arbitrary precision arithmetic and exact computation with rational numbers. Students learned how to define functions and substitutie for variables. Some data structures, such as lists and tables, were described. Students were asked to find the errors in some incorrect commands involving, for example, faulty order of operations or the use of lower case letters where Mathematica expects upper case. Students experimented with the idea of caching to accelerate the computation of recursive functions.

The Sieve of Eratosthenes. Procedures were given for producing a list of integers from 1 to 50 in a convenient rectangular array, and for successively replacing the higher multiples of a fixed integer in the list by zero. Students modified these procedures to treat lists of arbitrary length n. Using these procedures, they performed the usual sieve for various values of n, and observed the number of passes through the list that were needed before all the composites were replaced by zeros. They were asked to prove their observations.

Primes in Arithmetic Progression. A procedure was given for counting the number of primes in each of the congruence classes modulo 3 among the integers less than 500. Students were asked to modify the procedure to count the number of primes in each of the congruence classes modulo r among the integers less than n. After experimenting with various suggested values of r, they

formulated general conjectures. They were asked to prove simple observations, such as the paucity of primes when gcd(n,r) > 1.

A Linear Diophantine Equation Solver. Making use of the ExtendedGCD routine in Mathematica, students designed modules to find the solution (if any) to ax+by=c with minimal positive x. They also wrote modules to solve (if possible) a system of linear congruences involving an arbitrary number of moduli. It was particularly instructive to have them observe and explain why their algorithms were able to solve certain systems, in which the moduli were not pairwise relatively prime, when Mathematica's ChineseRemainderTheorem routine failed to do so.

Primality, Factoring and Codes. Students wrote modules to perform Miller's test for primality of an integer n, as follows.

1. Express n in the form $2^s t$, for example by dividing repeatedly by 2.
2. Choose a suitable base b and compute y = PowerMod[b,t,n] = b^t (modulo n). If y=1, then state that n is probably prime and end.
3. Successively check whether or not $y^{2j} = -1$ (modulo n), for j=0,...,s-1. If at any point this congruence is true, state that n is probably prime and end.
4. Report that n is composite.

Students checked examples, such as n=15579919981, for which Miller's test falsely asserts primality when the base b is 2 or 3. Once they became acquainted with algorithms for testing primality without factoring, they were asked to compared the running time for Mathematica's PrimeQ routine with that of FactorInteger.

Then students were asked to manufacture primes in an interval around a large integer N. Building on their experience with primes in arithmetic progression, they were led to the idea of searching for primes of the form dx+r, with d a product of many small primes, to improve the yield.

Students wrote modules to encode numerical data x via the function E(x) = PowerMod[x,e,n], with n a product of two of the primes found above. They constructed decoders D(x) = PowerMod[x,d,n], such that x^{de} = x (modulo n). To test their understanding of the relationship between factoring n and finding d, they were asked to factor integers n that were the product of two large primes, given e and d.

Finally, students were given a procedure to convert between characters and their ASCII code numbers. They then attempted to decode messages which they sent to each other.