Theoretical Computer Science

Probabilistic Algorithms

Technische Hochschule Rosenheim
Sommer 2022
Prof. Dr. Jochen Schmidt

Overview

- Pseudo random numbers
- Monte-Carlo Methods
  - example: probabilistic primality tests

Probabilistic Algorithm Result in…

- either an approximation of the actual result
  - typically by random sampling of the value range, using many sampling points
  - e.g.: (numerical) calculation of integrals, computer graphics (Photon Mapping)

- or a result that is only correct with a certain probability
  - e.g., primality tests:
    - Result: Number is not prime ⟶ always correct
    - Result: Number is prime ⟶ only correct with a very high probability

- in many cases, this is the only way to make the calculation practicable
- random numbers are required

Numerical Integration Using Monte-Carlo Algorithm

Technische Hochschule Rosenheim

- Calculate approximation of the value of
  $F = \int_a^b f(x)dx$

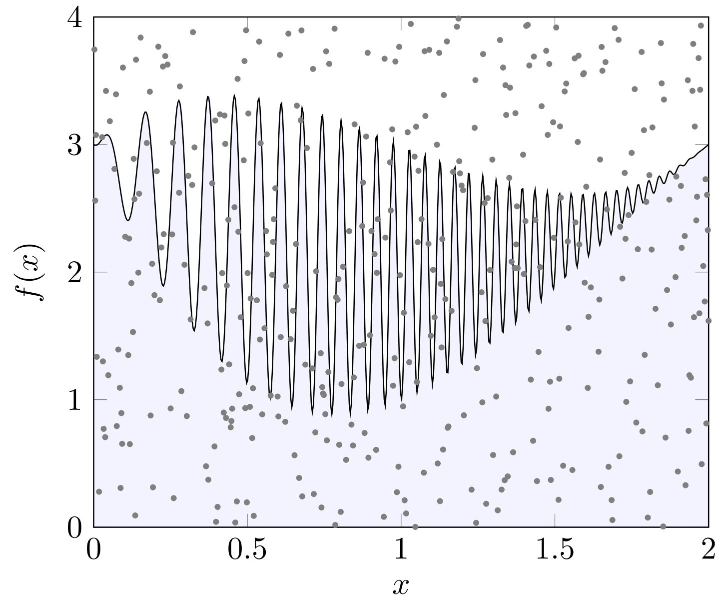
- Idea:
  - determine bounding box (area R) of $f(x)$ – given by maxima/minima in the interval [a; b]
  - generate N pairs of random numbers that define coordinates (x, y) within the rectangle
  - Count how many points N_f are below the function $f(x)$
  - An approximate value of F is given by

$$F = R\frac{N_\mathrm{f}}{N}$$

- Works the same way for multi-dimensional functions

- Example: Calculate integral

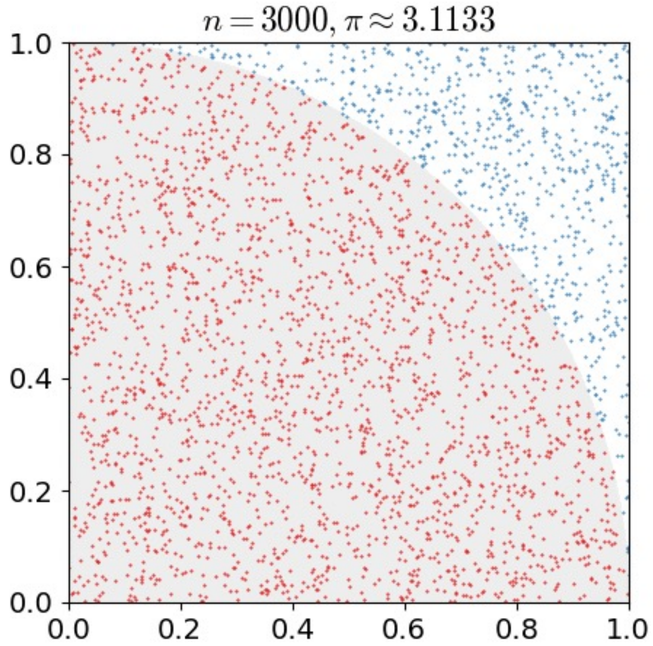
$$F = \int_0^2 \left(2 + (x-1)^2 + \sin\left[40 \cdot (x + x^2)\right] \cdot x \cdot (x-2)^2\right) dx$$



- Monte-Carlo with 10,000 random points:
  F = 4.671
- Exact value (to three decimal places):
  F = 4.667

Approximation of π

- Circle area: A = $r^2\pi \quad \rightarrow \qquad \pi = A / r^2$
- Use quarter circle as a function
- Test whether $\sqrt{x^2 + y^2} \leq r$ holds for random points (x, y)
  - then the point is in the circle
  - Calculate area as before in the integral example
- We will obtain $\pi/4$ from the equation above



Random Numbers

- True random numbers
  - use natural random processes
  - radioactive decay, thermal noise of electronic components, quantum physical processes
  - cannot be generated by standard computers

- Pseudo random numbers (*Pseudozufallszahlen*)
  - algorithmic calculation of "random numbers"
  - typically:
    - iterative calculation
    - sequence is repeated after a certain amount of numbers
  - deterministic: using the same initial value, the sequence is exactly reproducible
  - Initialization, e.g., using
    - Current system time
    - Current state of memory, registers, location of r/w head of hard drives, …

Distributions & Test for Randomness

- Important probability distributions:
  - uniform distribution (*Gleichverteilung*)
  - Normal distribution (Gaussian distribution, *Normalverteilung/Gaußverteilung*)

- Basic (pseudo) random number generators generate uniformly distributed numbers
  - which can be used to derive normally distributed ones

- Are the numbers “good”? Do they correspond to the desired distribution?
  $\longrightarrow$ use statistical tests, e.g.,
  - χ^2-Test (Pearson, 1900),
  - Kolmogorov-Smirnov-Test (1933/39).

Generation of Uniformly Distributed Random Numbers

- widespread: Linear Congruential Generator (LCG, *linearer Kongruenzgenerator*)
  - Lehmer 1949
- Calculate (integer) random numbers using the recurrence $x_{n+1} = (a\, x_n + c) \bmod m$
- where
  - m modulus $0 < m$
  - a multiplier $0 \leq a < m$
  - c increment $0 \leq c < m$
  - x_0 initial value (seed) $0 \leq x_0 < m$
- generates integers in the interval $[0; m - 1]$
- How do we choose the parameters?

LCG – Parameters / Examples

Technische Hochschule **Rosenheim**

Example: binary random sequence with m = 2?

$$x_{n+1} = (a\, x_n + c) \bmod m$$

- a = 0 and c = 0/1:
  $x_{n+1} = 0 \bmod 2$ or $x_{n+1} = 1 \bmod 2$
  - 0, 0, 0, 0, 0, … $\longrightarrow$ not very random
  - 1, 1, 1, 1, 1, … $\longrightarrow$ not very random
- a = 1 and c = 0: $x_{n+1} = x_n \bmod 2$
  - always returns the initial value x_0
- a = 1, c = 1, x_0 = 0: $x_{n+1} = (x_n + 1) \bmod 2$
  - 0, 1, 0, 1, 0, 1, 0, 1, 0, … $\longrightarrow$ not very random
- a = 1, c = 1, x_0 = 1: $x_{n+1} = (x_n + 1) \bmod 2$
  - 1, 0, 1, 0, 1, 0, 1, 0, 1, … $\longrightarrow$ not very random

Two other examples

- m = arbitrary, c = 1, a = 1, x_0 = 0:

  $x_{n+1} = (x_n + 1) \bmod m$
  - 0, 1, 2, 3, …, m – 1, 0, 1, 2, 3, …
    $\longrightarrow$ not very random
- m = 10, c = 7, a = 7, x_0 = 7:

  $x_{n+1} = (7x_n + 7) \bmod 10$
  - 7, 6, 9, 0, 7, 6, 9, 0, …
    $\longrightarrow$ not very random
- Conclusion: Choice of parameters is extremely important

LCG – Choice of Parameters

$x_{n+1} = (a\,x_n + c) \bmod m$

- $a \geq 2$
  $a = 0$ and $a = 1$ do not generate a random sequence
  (this obviously implies $m \geq 2$)

- $c = 0$ has the effect of
  - faster calculation
  - shorter period length

- m: as desired
  - in practice: based on word length of CPU, e.g., 2^{32}

- We get maximum period length m if and exactly if
  - c and m have no common prime factors: gcd(c, m) = 1
  - for each prime factor p of m: a – 1 is a multiple of p
  - if m is a multiple of 4: a – 1 is also a multiple of 4

Examples from actual implementations

- stdlib in gcc
  - $m = 2^{32}$
  - a = 1103515245
  - c = 12345

- Numerical Recipes
  - $m = 2^{32}$
  - a = 1664525
  - c = 1013904223

- Java Random Class
  - $m = 2^{48}$
  - a = 25214903917
  - c = 11
  - only (upper) 32 bits of the result are used
  - upper bits produce longer periods

Conversions

Given: uniformly distributed integer random number r in the interval [0; m – 1]

- Conversion to another integer interval [A, B]: $x = A + (r \bmod (B - A + 1))$
- Conversion to another real-valued interval [A, B]: $x = A + r\,(B - A) / (m - 1)$
- Conversion to normal distribution, in two steps:
  - Conversion to standard normal distribution $\mu = 0$, $\sigma = 1$
    - Polar method by Box, Muller, Marsaglia (1958/1962)
  - Conversion to any other normal distribution
    - let x be normally distributed with $\mu = 0$, $\sigma = 1$
    - then $ax + b$ is normally distributed with $\mu = b$, $\sigma = a$

Conversion to Normal Distribution – Polar Method

- Generate two uniformly distributed random numbers v_1 and v_2 in interval [–1; +1]
- Calculate $S = v_1^2 + v_2^2$
  - repeat these steps until S < 1
  - this is necessary on average 1.27 times, with standard deviation 0.587

- This results in two standard normally distributed random numbers x_1 and x_2 as follows:

$$x_1 = v_1\sqrt{\frac{-2\ln S}{S}} \qquad x_2 = v_2\sqrt{\frac{-2\ln S}{S}}$$

Primality Tests

- Given: Natural number n
- Question: Is n a prime number?

- Application: public key cryptography
  - e.g., RSA (1978)
  - public key: product of two very large prime numbers
  - 1024 – 4096 binary digits for key (= approx. 308 to 1233 decimal digits)

- Methods for primality tests
  - Sieve of Eratosthenes – exponential runtime
  - AKS-Test (2002) – polynomial runtime $\longrightarrow$ PRIMES $\in$ P
    - too slow for practical purposes
  - Instead: probabilistic primality tests

Fermats Little Theorem

- Pierre de Fermat (ca. 1607 – 1665)
- If p is prime, then for any natural number a that is not a multiple of p (just use a < p)

$$a^{p-1} \bmod p = 1$$

- The reverse is not true
  - there are numbers p that satisfy the equation even though they are not prime
  - e.g., $p = 11 \cdot 31 = 341$ $\longrightarrow$ $2^{340} \bmod 341 = 1$

- Fermat primality test
  - check for many (randomly generated) a whether Fermat's little theorem is satisfied
    - if there is an a for which it fails: p is definitely not prime
    - otherwise: inconclusive (interpreted as: p is probably prime)
  - Problem: there are numbers p
    - that are not prime,
    - but for which Fermat's little theorem is satisfied for **any** a $\longrightarrow$ Carmichael numbers

Miller-Rabin Primality Test

- published in 1976
- any odd number n can be represented as $n = 1 + q2^k$
- if n is prime then according to Fermat: $a^{n-1} \bmod n = 1 \longrightarrow a^{q2^k} \bmod n = 1$
- in addition, the following applies:
  $a^q \bmod n = 1$ or $a^{q2^r} \bmod n = n - 1 \equiv -1$ for some r with $0 \leq r \leq k - 1$

- **Idea**: Calculate the sequence $(a^q, a^{2q}, a^{4q}, \ldots, a^{q2^{k-1}}, a^{q2^k})$
  - use as many random "a" from the interval [2; n – 1] as required
- If n is a prime number, the sequence must have one of the following forms:
  - (1, 1, 1, …, 1)
  - $(x_1, x_2, x_3, \ldots, x_m, -1, 1, 1, \ldots, 1)$ x_i arbitrary numbers, m may be zero

Miller-Rabin Test – Examples

- Test n = 11 with a = 2

  $11 = 1 + 5 \cdot 2 \longrightarrow q = 5, k = 1$

  $2^5 \bmod 11 = 10 \equiv -1$
  $2^{10} \bmod 11 = 1$
  $\longrightarrow$ probably prime,
  maybe test some other a

- Test n = 65 with a = 2

  $65 = 1 + 1 \cdot 2^6 \longrightarrow q = 1, k = 6$

  $2^1 \bmod 65 = 2$
  $2^2 \bmod 65 = 4$
  $2^4 \bmod 65 = 16$
  $2^8 \bmod 65 = 61$
  $2^{16} \bmod 65 = 16$
  $2^{32} \bmod 65 = 61$
  $2^{64} \bmod 65 = 16$
  $\longrightarrow$ definitely not prime

- Test n = 561 with a = 2

  $561 = 1 + 35 \cdot 2^4 \longrightarrow q = 35, k = 4$

  $2^{35} \bmod 561 = 263$
  $2^{70} \bmod 561 = 166$
  $2^{140} \bmod 561 = 67$
  $2^{280} \bmod 561 = 1$
  $2^{560} \bmod 561 = 1$
  $\longrightarrow$ definitely not prime
  561 is the smallest Carmichael number

- Note: The calculation is greatly simplified if you use the following relationship:

$$(x \cdot y) \bmod n = \big((x \bmod n) \cdot (y \bmod n)\big) \bmod n$$

Miller-Rabin Test – Notes

- The most widely used primality test
- Error probability
  - the result “not prime” is always 100% correct
  - the result “prime” is for a single randomly chosen a is incorrect with a probability of ¼ = 25%
  - by repeated testing with different values for “a” we can make it arbitrarily small
    - e.g.: test 12 times ⟶ error probability $(¼)^{12}$ = 0.00000596%

- For compound numbers, the test does not provide any information about the prime factors!
  - Prime factorization is much more difficult
  - how difficult exactly, is currently unknown
    - probably not in P
    - but highly likely not NP-complete
      (otherwise, we would get P = NP, since factorization has been proven to be both, in NP **and** co-NP)

Primality Tests – Application Notes

How do we find a prime number?

- Use random number generator to get an odd number n from desired range
  - for cryptographic applications there are special cryptographically-secure pseudorandom number generators
- Use, e.g., Miller-Rabin test to check whether n is prime
  - this again requires generating a certain number of random integers to test (the “a” parameter)

- For 2048-Bit RSA two primes with approx. 308 decimal digits each are required
- How likely is it to find a prime in this range if we just pick some random number?

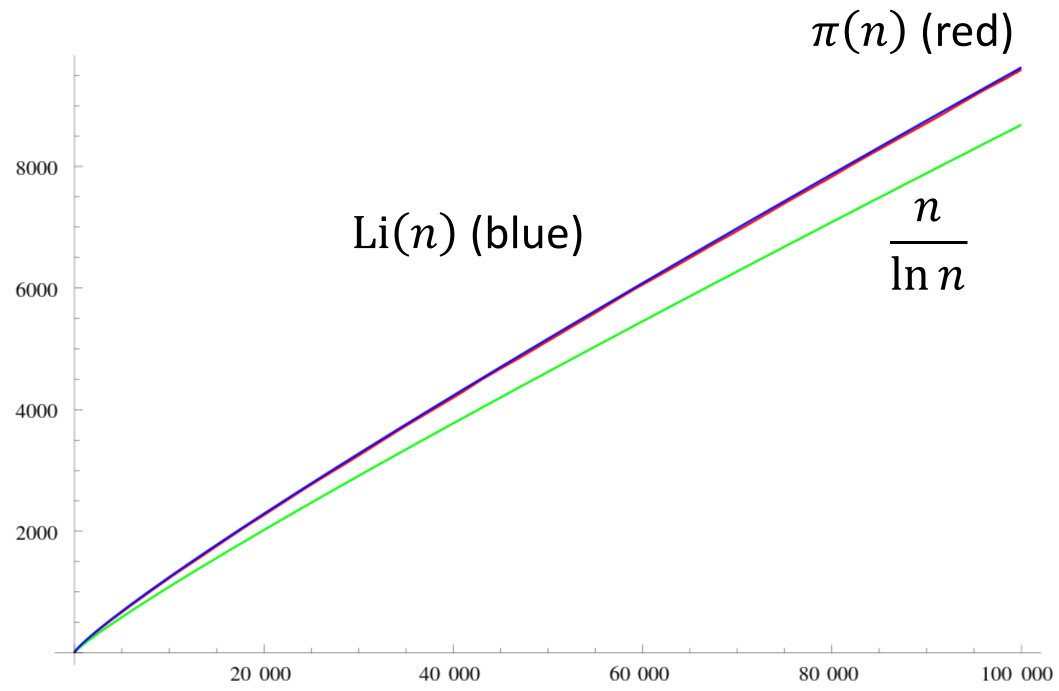
How Many Prime Numbers < n Are There?

- Let $\pi(n)$ the number of prime numbers less than or equal to n
- Prime number theorem (*Primzahlsatz*): $\pi(n) \sim \dfrac{n}{\ln n}$
  - the actual number is even slightly larger
  - conjectured by Gauss (1792/93), Legendre (1797/98)

- better approximation: $\pi(n) \sim \mathrm{Li}(n)$

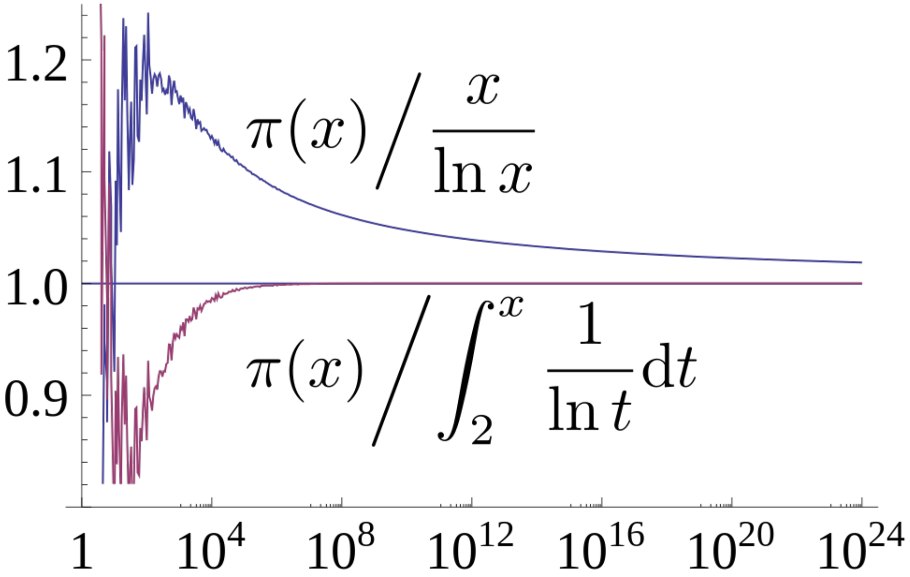
  where $\mathrm{Li}(n) = \int_2^n \frac{1}{\ln x} dx$
  - conjectured by Dirichlet (1838)

- Proofs by: Hadamard and Vallée-Poussin (1896)

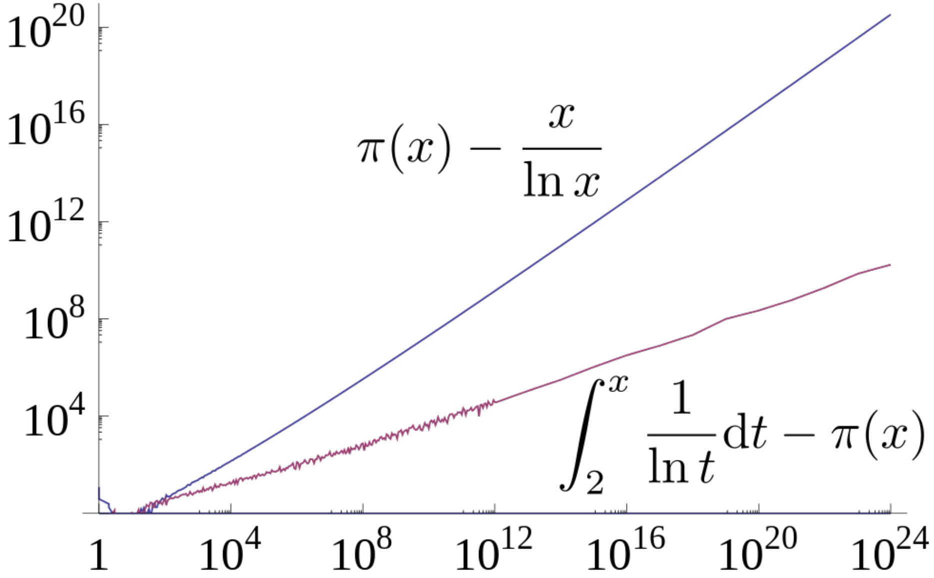


How Many Prime Numbers < n Are There?

convergence



absolute error



Probability for Primes

- Probability p that a randomly drawn number $x < n$ is prime (using relative frequency):

$$p(x \text{ prime}) = \frac{\frac{n}{\ln n}}{n} = \frac{1}{\ln n}$$

- Probability p that a randomly drawn odd number $x < n$ is prime :

$$p(x \text{ prime}) = \frac{\frac{n}{\ln n}}{n/2} = \frac{2}{\ln n}$$

- Probability p, that an odd random number x from the interval $[10^a;\ 10^{a+1}]$ is prime:

$$p(x \text{ prime}) = \frac{\frac{10^{a+1}}{\ln 10^{a+1}} - \frac{10^a}{\ln 10^a}}{1/2\,(10^{a+1} - 10^a)}$$

- first-order approximation:

$$p(x \text{ prime}) = \frac{\frac{10^{a+1}}{\ln 10^{a+1}}}{1/2\; 10^{a+1}} = \frac{2}{\ln 10^{a+1}}$$

$$= \frac{2}{(\mathrm{a}+1)\ln 10}$$

… which is the same as on the left

Example

Probability p, that an odd random number x from the interval $[10^{300};\ 10^{301}]$ is prime:

$$p(x \text{ prime}) = \frac{2}{301 \ln 10} \approx \quad 0.00288567 \quad \text{approx. } 0.29\%$$

Exact value for interval without first-order approximation: 0.00288461 approx. 0.29%

Better approximation by Li-integral:

$$p(x \text{ prime}) = \frac{\mathrm{Li}(10^{a+1}) - \mathrm{Li}(10^{a})}{{}^{1}\!/_{2}\,(10^{a+1} - 10^{a})}$$

Result for the example: 0.00321095 approx. 0.32%

Probability that out of 100 numbers drawn, at least one is prime:

$$1 - (1 - 0.0032)^{100} \approx 27.4\%$$

Summary

- probabilistic algorithms
  - deliver solutions that are likely to be correct (e.g., primality tests)
  - or approximations of solutions (e.g., numerical integration)
- there are many other applications of the Monte-Carlo method
  - physics simulations
  - micro-electronics
  - finance
  - …
- in all cases good pseudo-random numbers are needed

Sources

- H. Ernst, J. Schmidt und G. Beneken: *Grundkurs Informatik*. Springer Vieweg, 7. Aufl., 2020.
- D.E. Knuth: *The Art of Computer Programming*. Vol. 2, *Seminumerical Algorithms*. 3rd edition, Addison-Wesley, 1998.
- H. Scheid: Zahlentheorie. 2nd edition, BI Wissenschaftsverlag, 1994.