MAT 685: C++ for Mathematicians

Prime pairs in Monte Carlo

John Perry

University of Southern Mississippi

Spring 2017

Outline

Outline

Classic problem

- Choose n
- Choose $a, b \in \{1, \ldots, n\}$
- Let p_n be probability that $\gcd(a, b) = 1$
- Does $\lim_{n \to \infty} p_n$ exist?
  - If so, what is its value?

Classic problem

- Choose n
- Choose $a, b \in \{1, \ldots, n\}$
- Let p_n be probability that $\gcd(a, b) = 1$
- Does $\lim_{n \to \infty} p_n$ exist?
  - If so, what is its value?

Example

Let $n = 8$.

- Possible outcomes: $\binom{8}{2} = \frac{8!}{2!6!} = 28$
- Relatively prime pairs: $(1,2)$, $(1,3)$, ..., $(1,8)$, $(2,3)$, $(2,5)$, $(2,7)$, $(3,4)$, $(3,5)$, $(3,7)$, $(3,8)$, $(4,5)$, $(4,7)$, $(6,7)$, $(7,8)$
- So $p_8 = {}^{18}/_{28} = 9/14 \approx 64.3\%$

A problem

There are too many numbers! In the words of the two wise sages,

A problem

There are too many numbers! In the words of the two wise sages,

I'm, like, angry at numbers. Yeah, there's, like, too many of them, and stuff.

— Beavis & Butthead

A problem

There are too many numbers! In the words of the two wise sages,

I'm, like, angry at numbers. Yeah, there's, like, too many of them, and stuff.

— *Beavis & Butthead*

$n \to \infty$: program takes longer, and longer, and looooooongerrrrrrr...

Probabilistic algorithms

Two major types

Las Vegas "always correct, probably fast"

Monte Carlo "always fast, probably correct"

Probabilistic algorithms

Two major types

Las Vegas "always correct, probably fast"

Monte Carlo "always fast, probably correct"

Our approach

For large n:

- choose large subset of pairs
- determine how many are rel. prime
- divide by size of subset

Which algorithm type is this?

Example

$n = 100$

- Choose 10: (rel prime blue)

$(53, 97)$	$(22, 99)$	$(30, 42)$
$(20, 69)$	$(3, 9)$	$(61, 80)$
$(3, 37)$	$(9, 74)$	$(22, 43)$
	$(67, 94)$	

Example

$n = 100$

- Choose 10: (rel prime blue)

$(53, 97)$	$(22, 99)$	$(30, 42)$
$(20, 69)$	$(3, 9)$	$(61, 80)$
$(3, 37)$	$(9, 74)$	$(22, 43)$
	$(67, 94)$	

- estimate: 70%
- actual: 69%

Not bad!

(and much less work than $\frac{100 \cdot 101}{2}$)

Outline

How should we choose pairs?

- no obvious pattern
- "uniform" distribution

$\therefore$ **Choose randomly**

(of course)

Question

But how do we generate random numbers?!?

We don't. We generate *pseudo*-random numbers.

linear congruential generator $x_{n+1} = (ax_n + b) \bmod c$

seed x_0 (first number in sequence)

Question

But how do we generate random numbers?!?

We don't. We generate *pseudo*-random numbers.

linear congruential generator $x_{n+1} = (ax_n + b) \bmod c$

seed x_0 (first number in sequence)

Example

Let $a = 3, b = 7, c = 10, x_0 = 2$

$$2, 3, 6, 5, 2, 3, 6, 5, 2, 3, 6$$

Question

But how do we generate random numbers?!?

We don't. We generate *pseudo*-random numbers.

linear congruential generator $x_{n+1} = (ax_n + b) \bmod c$

seed x_0 (first number in sequence)

Example

Let $a = 3, b = 7, c = 10, x_0 = 2$

$$2, 3, 6, 5, 2, 3, 6, 5, 2, 3, 6$$

...not a very good choice!

C++ library helps out

- Pseudo-random generator

```
#include <cstdlib>
void srand(unsigned seed)
int rand()
```

  - use `srand()` to seed, usually to system time
  - `rand()` result in [`0`, `RAND_MAX`]

C++ library helps out

- Pseudo-random generator

```
#include <cstdlib>
void srand(unsigned seed)
int rand()
```

  - use `srand()` to seed, usually to system time
  - `rand()` result in [`0`, `RAND_MAX`]

- System time?

```
#include <ctime>
time_t time(time_t * storage)
```

  - `storage` is "address" of optional variable to store time
  - use `nullptr` for now

`nullptr`?

- C++11 standard for "nowhere"
- previous conventions: 0, `NULL`
  - conventions, not standards
  - **avoid these**

Book offers adaptation

Why?

- specify different interval
  - not just `[0, RAND_MAX]`
- ways to avoid patterns in certain special sets
- different generator desired? easier to change

Interface (p. 1)

New folder, `rand_unif`

Listing 1: `uniform.hpp` (p. 1)

```
#ifndef UNIFORM_HPP
#define UNIFORM_HPP

/** Generate a random number between 0 and 1.*/
double unif();

/**
  Generate a random number in a real interval.
  @param a one endpoint of the interval
  @param b the other endpoint of the interval
  @return a uniform random number in [0,1]
*/
double unif(double a, double b);
```

Interface (p. 2)

Listing 2: `uniform.hpp` (p. 2)

```
/**
  Generate a random integer between 1 and a
  given value.
  @param n the largest possible value
  @return uniform random value in {1,...,n}
*/
long unif(long n);

/** Reset generator based on system clock */
void seed();

#endif
```

Implementation (p. 1)

Listing 3: `uniform.cpp` (p. 1)

```
#include <ctime>
using std::time;
#include <cstdlib>
using std::rand; using std::srand;

#include "uniform.hpp"

double unif() {
  return rand() / double(RAND_MAX);
}

double unif(double a, double b) {
  return (b - a)*unif() + a;
}
```

Implementation (p. 2)

Listing 4: `uniform.cpp` (p. 2)

```
long unif(long a) {
  if (a == 0) return 0;
  if (a < 0) a = -a;
  return long(unif()*a) + 1;
}

void seed() {
  srand(time(nullptr));
}
```

Outline

Organizing files

- It is common to organize source files in folders and subfolders.
- Currently, though, we have a mess:

```
c++_main
  gcd
    gcd.hpp
    gcd_recursive.cpp
    gcd_while.cpp
    prob_relprime_pair.cpp
    test_gcd_recursive.cpp
    test_gcd_while.cpp
    test_xgcd.cpp
  rand_unif
    uniform.cpp
    uniform.hpp
```

How to access both sets of files?

Monte Carlo approach needs both gcd and uniform files. They are in different folders.

How to access both sets of files?

Monte Carlo approach needs both gcd and uniform files. They are in different folders.

- Create new folder **`relprime_pairs`**
  - move `prob_relprime_pair.cpp` in
  - no longer compiles; we can fix it

How to access both sets of files?

Monte Carlo approach needs both gcd and uniform files. They are in different folders.

- Create new folder **`relprime_pairs`**
  - move `prob_relprime_pair.cpp` in
  - no longer compiles; we can fix it

`#include "/path/to/filename"` can use absolute or relative paths to an interface

- change `#include ``gcd.hpp''` to `#include ``../gcd/gcd.hpp''`
- compilation:

```
$ cd relprime_pairs
$ g++ -o prob_relprime_pair \
    ../gcd/gcd_while.cpp \
    prob_relprime_pair.cpp
$ ./prob_relprime_pair
```

Homework

pp. 63–65 #4.1, 4.4, 4.5

Outline

Monte Carlo method

Place in `relprime_pairs` folder

Listing 5: `monte_carlo_pairs.cpp` (p. 1)

```
#include <iostream>
using std::cin; using std::cout;
using std::endl;

#include "../gcd/gcd.hpp"
#include "../rand_unif/uniform.hpp"

int main() {
  long n;
  long reps;  // # times we perform experiment
  long a, b;  // pairs in {1,..,n}
  long count; // number of rel prime pairs

  count = 0;
  cout << "Enter n (max el of set) --> ";
  cin >> n;
```

Monte Carlo method

Place in `relprime_pairs` folder

Listing 6: `monte_carlo_pairs.cpp` (p. 2)

```
  cout << "Enter number of pairs to sample --> ";
  cin >> reps;

  seed();
  for (long k = 0; k < reps; ++k) {
    a = unif(n);
    b = unif(n);
    if (gcd(a,b) == 1) ++count;
  }

  cout << count / double(reps) << endl;

  return 0;
}
```

Compiling, executing

Compiling... pay attention to dots!

```
$ g++ -o monte_carlo_pairs \
    -std=c++11 \
    ../gcd/gcd_while.cpp \
    ../rand_unif/uniform.o \
    monte_carlo_pairs.cpp
```

(may need `-std=c++11` to avoid complaint about `nullptr`)

Compiling, executing

Compiling... pay attention to dots!

```
$ g++ -o monte_carlo_pairs \
    -std=c++11 \
    ../gcd/gcd_while.cpp \
    ../rand_unif/uniform.o \
    monte_carlo_pairs.cpp
```

(may need `-std=c++11` to avoid complaint about `nullptr`)

Executing

```
$ ./monte_carlo_pairs
Enter n (max el of set) --> 100000
Enter number of pairs to sample --> 1000
0.595
```

Comparison

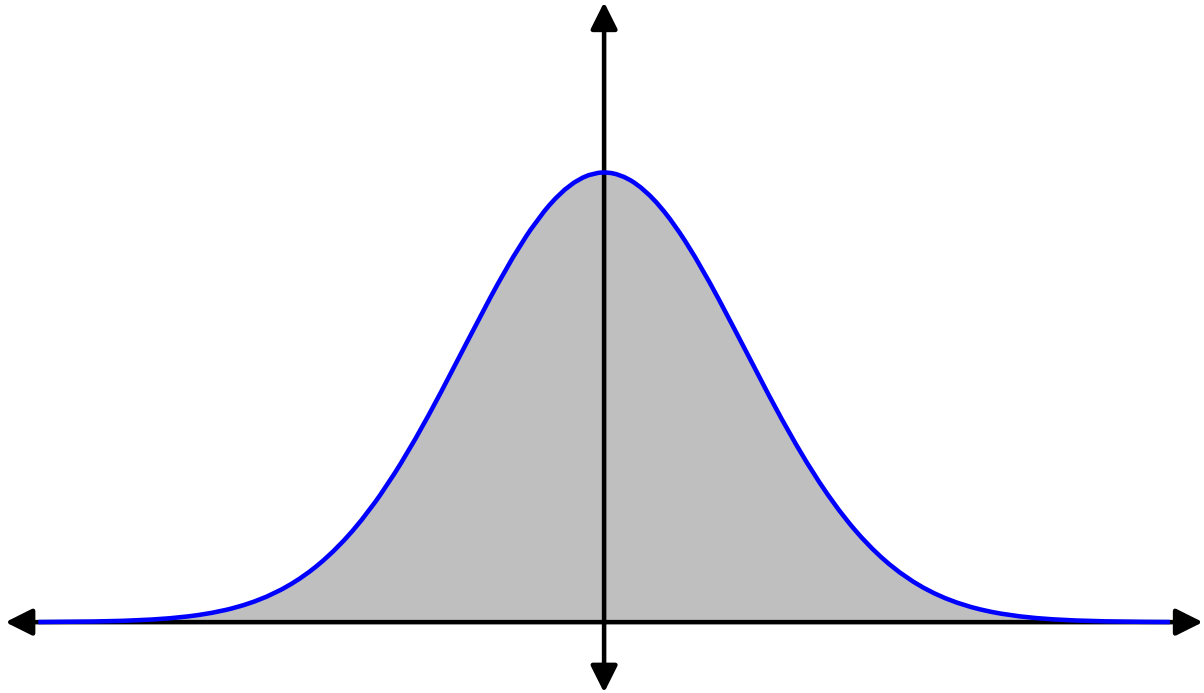
Estimate: 0.595
Actual: 0.614747

- `prob_relprime_pairs` takes *a long time* to find it
- can get different estimates, too:
  - different execution times
  - different values of `reps`
- can approximate correct value w/many estimates *faster* than finding exact

Outline

Not all distributions are uniform

"Normal" random variables are Gaussian:



$$f(t) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{t^2}{2}}$$

Probabilistic properties

$$\int_{\mathbb{R}} f \, dt = 1 \quad \text{area under curve}$$
$$\int_{\mathbb{R}} tf \, dt = 0 \quad \text{mean}$$
$$\int_{\mathbb{R}} t^2 f \, dt = 1 \quad \text{standard deviation}$$

Probabilistic properties

$$\begin{array}{ll} \int_{\mathbb{R}} f\ dt = 1 & \text{area under curve} \\ \int_{\mathbb{R}} tf\ dt = 0 & \text{mean} \\ \int_{\mathbb{R}} t^2 f\ dt = 1 & \text{standard deviation} \end{array}$$

$$\text{define } \Phi(x) = P|X \leq x| = \int_{-\infty}^{x} f(t)\ dt$$

Generating normal random values

Box-Muller method

generate (x, y) uniformly in unit disc

let

$$r = \sqrt{x^2 + y^2}, \quad \mu = \sqrt{\frac{-2 \ln r}{r}}, \quad z = \mu x$$

return z

Generating normal random values

Box-Muller method

generate (x, y) uniformly in unit disc
let

$$r = \sqrt{x^2 + y^2}, \quad \mu = \sqrt{\frac{-2 \ln r}{r}}, \quad z = \mu x$$

return z

Remark

$y = \mu y$ would also be random value

`static` "remembers" values

Listing 7: `memories.cpp`

```
#include <iostream>
using std::cin; using std::cout; using std::endl;

int memory(int val) {
  static int last_val; // static, so "remembers"
  int ret_val = last_val; // temporary
  last_val = val; // replace
  return ret_val; // return
}

int main() {
  for (int i = 0; i < 10; ++i) {
    int new_val;
    cout << "Enter #" << i << ": ";
    cin >> new_val;
    cout << "I remember " << memory(new_val) << endl;
  }
}
```

Compiling, executing

```
$ g++ -o memories memories.cpp
$ ./memories
Enter #0: 5
I remember 0
Enter #1: 3
I remember 5
Enter #2:
```

Compiling, executing

```
$ g++ -o memories memories.cpp
$ ./memories
Enter #0: 5
I remember 0
Enter #1: 3
I remember 5
Enter #2:
```

...now back to normal random values!

Interface

New folder, `rand_norm`

Listing 8: `randn.hpp`

```
#ifndef RANDN_HPP
#define RANDN_HPP

/**
  Generate a random number between -1 and 1
  according to a normal distribution.
*/
double randn();

#endif
```

Implementation

Listing 9: `randn.cpp` (p. 1)

```
#include <cmath>
using std::sqrt;

#include "../rand_unif/uniform.hpp"

double randn() {
  static bool has_saved = false;
  static double saved;

  // use saved value, if we have one
  if (has_saved) {
    has_saved = false;
    return saved;
  }

  // pick a point within unit disk
  double x, y;
  double r = 2.0;
```

Implementation

Listing 10: `randn.cpp` (p. 2)

```
  while (r > 1.0) {
    x = unif(-1.0, 1.0);
    y = unif(-1.0, 1.0);
    r = x*x + y*y;
  }

  // rescale by Box-Muller
  r = sqrt(r);
  double mu = sqrt(-2.0 * log(r) / r);

  // save y value, return x value
  saved = mu*y;
  has_saved = true;

  return mu*x;
}
```

Program to compare

Place in parent folder

Listing 11: `random_comparisons.cpp`

```
#include <iostream>
using std::cout; using std::endl;

#include "rand_unif/uniform.hpp"
#include "rand_norm/randn.hpp"

int main() {
  seed();

  cout << "Uniform:\n";
  for (int i = 0; i < 20; ++i)
    cout << unif(10) << ',';
  cout << endl;

  cout << "Normal:\n";
  for (int i = 0; i < 20; ++i)
    cout << long(randn()*10/4) + 5 << ',';
  cout << endl;
}
```

Compiling, executing

```
$ g++ -o random_comparisons rand_unif/uniform.o \
    rand_norm/randn.o random_comparisons.cpp
$ ./random_comparisons
Uniform:
9,4,8,8,10,2,4,8,3,6,5,7,4,6,10,10,7,8,2,7,
Normal:
5,5,4,5,5,7,6,3,8,5,5,5,4,6,5,7,5,6,3,4,
$
```

Compiling, executing

```
$ g++ -o random_comparisons rand_unif/uniform.o \
    rand_norm/randn.o random_comparisons.cpp
$ ./random_comparisons
Uniform:
9,4,8,8,10,2,4,8,3,6,5,7,4,6,10,10,7,8,2,7,
Normal:
5,5,4,5,5,7,6,3,8,5,5,5,4,6,5,7,5,6,3,4,
$
```

Notice bunching of normal

- 5,6,7 appear 14 times

Homework

p. 65 #4.6, 4.7

Outline

Summary

- Math stuff
  - Monte Carlo algorithms
  - Probability distributions
  - Pseudorandom generators
    - Linear congruential generator
    - Box-Muller method
- Programming stuff
  - C++ Standard Library: random
  - file organization
  - `static` variables