

Math 4997-3

Lecture 2: Monte Carlo, Vectors, and Algorithms

<https://www.cct.lsu.edu/~pdiehl/teaching/2019/4977/>

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.



Reminder

Monte Carlo methods

Containers & Algorithms

Functions

Summary

References

Reminder

Lecture 1

What you should know from last lecture

- ▶ Structure of a C++ program
- ▶ `std::string`
 - ▶ Reading strings `std::cin`
 - ▶ Writing strings `std::cout`
- ▶ Looping and counting
 - ▶ `while`
 - ▶ `for`
- ▶ Conditionals `if`
- ▶ Operators
- ▶ Built-in types

Monte Carlo methods

Monte Carlo method

Monte Carlo methods are computational algorithms which rely on repeated random sampling to obtain numerical results. More details [3].

Three problem classes [2]:

1. Optimization
2. Numerical integration
3. Probability distributions

General pattern:

1. Define the input parameters
2. Randomly chose input parameters
3. Do deterministic computations on the inputs
4. Aggregate the results

Example: Estimate the value of π

Ingredients



- ▶ Unit square 1×1
- ▶ circle with the radius of $r = 1/2$
- ▶ Area of the circle $A_c = \pi r^2 = \pi/4$
- ▶ Area of the square $A_s = 1 \cdot 1 = 1$
- ▶ Recall: $A_c = \pi/4 \rightarrow \pi = 4A_c$
- ▶ Hint: $A_c/A_s = A_c$ because $A_s = 1$

Now compute PI by using the two areas:

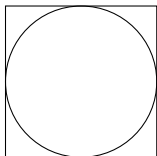
$$\pi \approx 4A_c/A_s$$

The areas can be approximated by using N random samples (x, y) and count the points inside the circle N_c

$$\pi \approx 4N_c/N$$

Example: Estimate the value of π

Ingredients



- ▶ Unit square 1×1
- ▶ circle with the radius of $r = 1/2$
- ▶ Area of the circle $A_c = \pi r^2 = \pi/4$
- ▶ Area of the square $A_s = 1 \cdot 1 = 1$
- ▶ Recall: $A_c = \pi/4 \rightarrow \pi = 4A_c$
- ▶ Hint: $A_c/A_s = A_c$ because $A_s = 1$

Now compute PI by using the two areas:

$$\pi \approx 4A_c/A_s$$

The areas can be approximated by using N random samples (x, y) and count the points inside the circle N_c

$$\pi \approx 4N_c/N$$

Algorithm

1. Generate random coordinates $(x, y) \in \mathbb{R}^2$
2. Check if $x^2 + y^2 \leq 1$
 - Update N_c if ≤ 1
3. Increment n the interval
4. If $n < n_{\text{total}}$ go to 1
5. Calculate $\pi \approx 4N_c/n_{\text{total}}$
6. Print result

Interactive question:

Which features of C++ do we need?

Algorithm

1. Generate random coordinates $(x, y) \in \mathbb{R}^2$
2. Check if $x^2 + y^2 \leq 1$
 - Update N_c if ≤ 1
3. Increment n the interval
4. If $n < n_{\text{total}}$ go to 1
5. Calculate $\pi \approx 4N_c/n_{\text{total}}$
6. Print result

Interactive question:

Which features of C++ do we need?

`for`, `if`, `std::cout`, and random numbers

Random numbers¹

```
// Include for using rand
#include <cstdlib>
#include <iostream>
//Include for getting the current time
#include <ctime>

int main()
{
    // Use the current time as random seed
    std::srand(std::time(0));
    // Get one random number
    int random_variable = std::rand();
    std::cout
        << "Random value on [0 " << RAND_MAX << "]: "
        << random_variable << '\n';
}
```

More details: Section 3.2 [1]

¹<https://en.cppreference.com/w/cpp/numeric/random/srand>

Uniform distributed random numbers^{2 3}

```
// Include for advanced random numbers
#include <random>
#include <iostream>

int main()
{
    //Generate a random number device
    std::random_device rd;
    //Set the standard mersenne_twister_engine
    std::mt19937 gen(rd());
    //Specify the interval [1,6]
    std::uniform_int_distribution<size_t> dis(1, 6);
    //Specifiy the interval [1.0,6.0]
    std::uniform_real_distribution<double> disd(1,6);
    std::cout << dis(gen) << " " << disd(gen) << '\n';
}
```

²https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution

³https://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution

Containers & Algorithms

Computation of the average⁴

```
#include <iostream>
#include <iomanip>

int main()
{
    double sum = 0;
    size_t count = 0;
    double x = 0;
    while (std::cin >> x)
    {
        sum += x;
        ++count;
    }
    std::cout << "Average: "
              << std::setprecision(3)
              << sum / count << std::endl;
}
```

⁴<https://en.cppreference.com/w/cpp/io/manip/setprecision/>

Reading multiple values

Recall: Reading one value

```
std::cin >> x
```

Reading multiple values

```
size_t count = 0;
double x = 0;
while (std::cin >> x)
{
    sum += x;
    ++count;
}
```

Note that the `while` loop stops by inserting `n`

What about computing the median?

We have to:

- ▶ We have to store all values without knowing the amount in advance
- ▶ We have to sort all the values
- ▶ We have to select the middle value(s) efficiently

What about computing the median?

We have to:

- ▶ We have to store all values without knowing the amount in advance
- ▶ We have to sort all the values
- ▶ We have to select the middle value(s) efficiently

What we will use:

- ▶ Containers: `std::vector` to store the elements
- ▶ Algorithm, like sorting or accumulate
- ▶ Iterators for accessing elements

Advanced computing of the average

```
#include <iostream>
#include <vector>
#include <numeric>

int main()
{
    std::vector<double> values;
    double x;
    while (std::cin >> x)
    {
        values.push_back(x);
    }
    double sum =
        std::accumulate(values.begin(), values.end(), 0.0f);
    std::cout << "Average: "
        << sum / values.size() << std::endl;
}
```

Initialization of Vectors⁵

Initialization of an empty vector

- ▶ `std::vector<double> values;`
- ▶ `values.empty()` will return `true`
- ▶ `values.size()` will return zero

Initialization

- ▶ `std::vector<double> v = {1, 2.5};`
- ▶ `values.empty()` will return `false`
- ▶ `values.size()` will return two

Note that C++ starts counting by 0 and the length is always number of elements minus one!

⁵<https://en.cppreference.com/w/cpp/container/vector>

Manipulating vectors

Inserting

- ▶ `values.push_back(3.5);` append 3.5 at the end
- ▶ `values[3] = 1.5;` replaces the third element by 1.5

Accessing

- ▶ `values[i]` returns element i
- ▶ `values.first()` returns the first element
- ▶ `values.last()` return the last element

Deleting

- ▶ `values.pop_back();` deletes the last element
- ▶ `values.erase(values.start()+i)` deletes the i -th element

Algorithms

Accumulate⁶

```
double sum = std::accumulate(v.begin(), v.end(), 0.0f);
```

- ▶ The first two arguments define the range of the vector
- ▶ The third is the initial value of the sum
- ▶ The header `#include <numerics>` provides this functionality

Sorting⁷

```
std::sort(s.begin(), s.end());
```

- ▶ The header `#include <algorithm>` provides this functionality

⁶<https://en.cppreference.com/w/cpp/algorithm/accumulate>

⁷<https://en.cppreference.com/w/cpp/algorithm/sort>

Looping over a vector

Long form

```
for( size_t i = 0; i < value.size() ; i++)  
{  
    v[i] *= 2  
}
```

Short form

```
for (auto& v : values)  
{  
    v *= 2  
}
```

► `auto` is equivalent to `double v : values`⁸

⁸<https://en.cppreference.com/w/cpp/language/auto>

Recall: Advanced computing of the average

```
#include <iostream>
#include <vector>
#include <numeric>

int main()
{
    std::vector<double> values;
    double x;
    while (std::cin >> x)
    {
        values.push_back(x);
    }
    double sum =
        std::accumulate(values.begin(), values.end(), 0.0f);
    std::cout << "Average: "
        << sum / values.size() << std::endl;
}
```

Computing the median

Code

```
typedef std::vector<double>::size_type vec_sz;  
vec_sz size = values.size();  
double median = 0;  
  
if( size % 2 == 0 )  
    median = 0.5*(values[mid] + values[mid-1]);  
else  
    median = values[mid];
```

Features

- ▶ `typedef`⁹ is useful to not write large expression again
- ▶ Conditional parameter¹⁰

```
median = size % 2 == 0 ? true : false;
```

⁹<https://en.cppreference.com/w/cpp/language/typedef>

¹⁰https://en.cppreference.com/w/cpp/language/operator_other

Example: Advanced computing of the average

```
#include <iostream>
#include <vector>
#include <algorithm>

int main(){
    typedef std::vector<double>::size_type vec_sz;
    std::vector<double> values;
    double x;
    while (std::cin >> x)
    {
        values.push_back(x);
    }
    std::sort(values.begin(), values.end());
    vec_sz mid = values.size() / 2;
    double median = values.size() % 2 == 0 ?
        0.5*(values[mid]+values[mid-1]) : values[mid];
    std::cout << "Median: "
        << median << std::endl;
}
```

Functions

Function definition¹¹

Example: maximum

```
int max(int a, int b)
{
    return a>b?a:b;
}
```

Each function has

- ▶ a name `max`
- ▶ a return type `int` `max()`
- ▶ a return value `return a>b?a:b;`
- ▶ some function parameters `max(int a, int b)` or none `max(void)`

Function are defined between `#include` and `int main (void)`

¹¹https://en.cppreference.com/w/c/language/function_definition

Function definition for the median

Function definition

```
double median(std::vector<double> values)
{
    std::sort(values.begin(), values.end());
    vec_sz mid = values.size() / 2;
    double median = values.size() % 2 == 0 ?
        0.5*(values[mid]+values[mid-1]) : values[mid];

    return median;
}
```

Example

```
double median = median(values)
```

Summary

Summary

After this lecture, you should know

- ▶ Monte Carlo Methods
- ▶ Random numbers
- ▶ Containers like `std::vector`
- ▶ Functions

References

References I



Donald Ervin Knuth.

The art of computer programming: Seminumerical Algorithms, volume 2.

Pearson Education, 1968.



Dirk P Kroese, Tim Brereton, Thomas Taimre, and Zdravko I Botev.

Why the monte carlo method is so important today.

Wiley Interdisciplinary Reviews: Computational Statistics,
6(6):386–392, 2014.



Ronald W Shonkwiler and Franklin Mendivil.

Explorations in Monte Carlo Methods.

Springer Science & Business Media, 2009.