November 2024

Introduction to C++: Workshop Six

Dr. Alexander Hill

a.d.hill@liverpool.ac.uk

LIV.INNO

Last Week

- Monte Carlo Basics
- Generating random numbers in C++

Aim of Workshop Six

- Homework recap
- Markov Chains
- Group Project

Resources

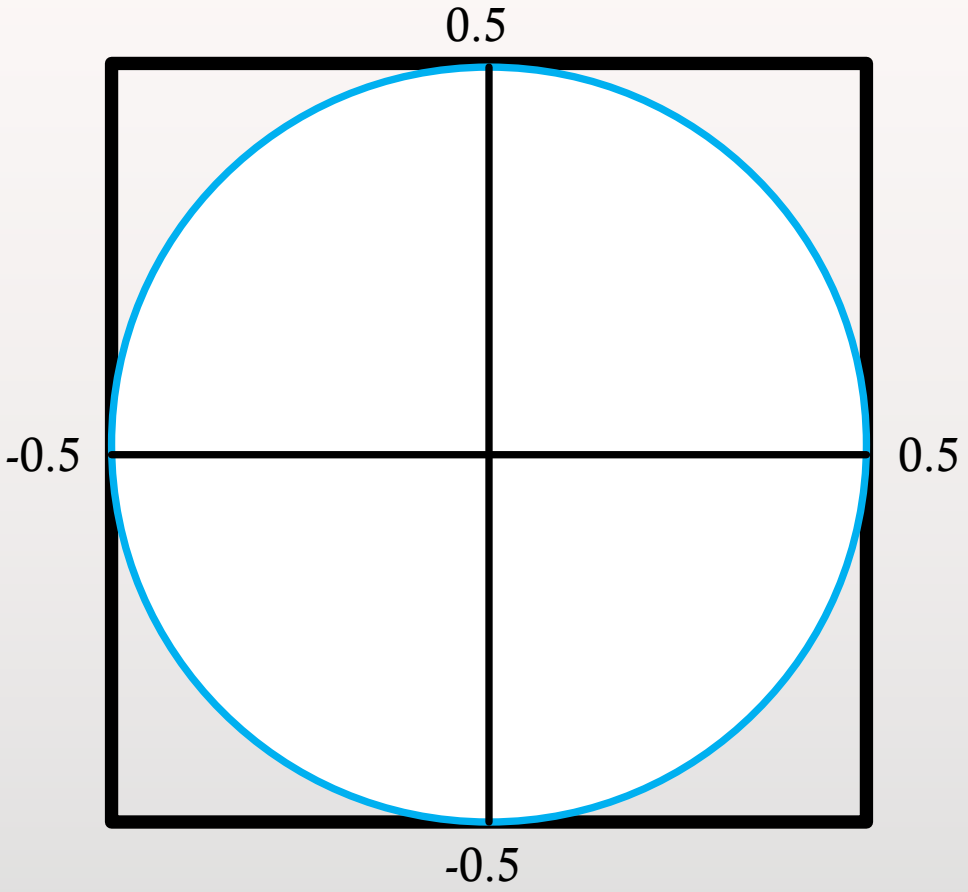
- Previous lecture slides for details on random numbers, classes, functions, etc.
- alex-hill94.github.io/#WS6 for slides and recordings of previous workshops
- https://cplusplus.com/reference/random

Challenge Ten (Homework)

- For the circle described by $x^2 + y^2 = 0.5^2$, compute its area using a Monte Carlo method

- i.e. draw random x and y between -0.5 and 0.5, and compute the fraction of draws that satisfy $x^2 + y^2 < 0.5^2$

- How many draws do you need to get ~1% error on πr^2 ?



Andrew

```cpp
#include <random>
#include <iostream>
#include <cmath>

using namespace std;

bool isInside(double x, double y, double radius)
{
return (x * x + y * y <= radius * radius);
}
```

```
$ ./output
Pi is approximately: 3.168
Error: 0.840572%
$ ./output
Pi is approximately: 3.184
Error: 1.34987%
$ ./output
Pi is approximately: 3.196
Error: 1.73184%
```

Nice and concise

Computes error on pi, not the area

```cpp
int main(int argc, char const *argv[])
{
int trials = 1e3; // number of trials
int inside = 0; // number of points inside the circle
double radius = .5;

random_device rd;
mt19937 gen(rd());
uniform_real_distribution<> distr(-radius, radius);
for (size_t i = 0; i < trials; i++)
{
double x = distr(gen);
double y = distr(gen);
inside += isInside(x, y, radius);
}
double res = 4.0 * inside / trials;
cout << "Pi is approximately: " << res << endl;
cout << "Error: " << 100 * abs(M_PI - res) / M_PI << "%" << endl;
return 0;
}
```

Marina

```cpp
int main(){
int ndraws;
cout << "Number of draws: ";
cin >> ndraws;
cout << endl;

double radius = 0.5;
const double range_from_x = -radius;
const double range_to_x = radius;
const double range_from_y = -radius;
const double range_to_y = radius;

double in = 0;
double out = 0;
for (int i=0; i<ndraws; i++){
auto x_draw = draw(range_from_x,range_to_x);
auto y_draw = draw(range_from_y,range_to_y);
```

```cpp
if (pow(x_draw,2)+pow(y_draw,2) <= pow(radius,2)){
in += 1;
}
else {out += 1;}

}
double ratio = in/ndraws;
double area = ratio*pow(2*radius,2);

cout << "Area: " << area << endl;

double error = abs((area -
M_PI*pow(radius,2))/(M_PI*pow(radius,2)))*100;

cout << "Error: " << error << endl;

return 0;
}}
```

Marina

```
$ ./output
Number of draws: 100000

Area: 0.78429
Error: 0.141096

$ ./output
Number of draws: 10000

Area: 0.7802
Error: 0.661851
$ ./output
Number of draws: 1000

Area: 0.806
Error: 2.62311
```

```cpp
// Function to estimate the area
double estimate_circle_area(int num_samples, double square_area) {

int count_inside_circle = 0;
double square_side = std::sqrt(square_area);
double radius = square_side/2;

// Generate random no's
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_real_distribution<double> dis(-square_side/2, square_side/2); // Uniform
distribution between -0.5 and 0.5



for (int i = 0; i < num_samples; ++i) {
double x = dis(gen);
double y = dis(gen);

if (x * x + y * y <= radius*radius) { // Check if the point is inside the circle
count_inside_circle++;
}
}

// Circle area calculation with respect to the square area
double circle_area_estimate = (static_cast<double>(count_inside_circle) / num_samples) *
square_area;

return circle_area_estimate;
}
```

```
$ ./output
Estimated area of the circle: 0.78339
Actual area of the circle: 0.785398
```

Converts integer to a float

Emily

```
$ ./output
Fraction inside the circle: 0.764
True area =0.785398
```

```cpp
#include <iostream>
#include <vector>
#include <random>
using namespace std;

bool area(double x, double y){
return (x * x + y * y) < 0.5 * 0.5;
}

int main() {
// Number of random values you want to generate
int numValuesToGenerate = 1000;
int numValuesInsideCircle = 0;

double range_from = -0.5;
double range_to = 0.5;

for (int i = 0; i < numValuesToGenerate; i++) {
random_device rand_dev;
mt19937 generator(rand_dev());
uniform_real_distribution<double> distr(-0.5, 0.5);
```

```cpp
double x = distr(generator);
double y = distr(generator);

if (area(x, y)) {
numValuesInsideCircle++;
}
}

// Compute the fraction of draws that satisfy x^2 + y^2 < 0.5^2
double fractionInsideCircle = static_cast<double>(numValuesInsideCircle) /
numValuesToGenerate;

cout << "Fraction inside the circle: " << fractionInsideCircle << endl;

double TrueArea = M_PI * 0.5 * 0.5;
cout << "True area =" << TrueArea << "\n";

return 0;
}
```

Nice and concise, good job!

Mehul

```
int main(){
vector <int>
n_values={100,1000,10000,100000,1000000,10000000};
vector <double> pi_values;
vector <double> error_pi_value;
for(int j: n_values)
{
double pi_value = pi(j);
pi_values.push_back(pi_value);
double error = 100*(1-(pi_value/M_PI));
error_pi_value.push_back(error);
cout << "The value for pi with n = " << j << " is " <<
pi_value << " The error on this is : " << error << " %
" << endl;
}
return 0;
}
```

```
$ ./output
The value for pi with n = 100 is 3.16 The error
on this is : -0.585924 %
The value for pi with n = 1000 is 3.228 The
error on this is : -2.75043 %
The value for pi with n = 10000 is 3.142 The
error on this is : -0.0129662 %
The value for pi with n = 100000 is 3.13148
The error on this is : 0.321896 %
The value for pi with n = 1000000 is 3.14173
The error on this is : -0.00430821 %
```

Looks good, error trend is strange though, is there a bug?

```cpp
using namespace std;

vector<double> x;
vector<double> y;
int i;
int j;
int k;
int draw = 0; // Initialize 'draw' to 0
float fraction;
double area;
```

Sinead

```
$ ./output
How many draws should be done? 100
Area of circle is: 0.1925
Actual area is: 0.19635
Error is: -1.99976
Draw count: 77
```

```cpp
float fraction = static_cast<float>(draw) / k; // Use k as the
total count, not a hardcoded number
//cout << fraction << endl;
double area = fraction * pow(0.5, 2); // Change 0.5 to 1.0 for
the full circle
double real_area = M_PI * pow(0.25, 2);
double error = (1-real_area/area) *100;

cout << "Area of circle is: " << area << endl;
cout << "Actual area is: " << real_area << endl;
cout << "Error is: " << error << endl;
// need approx 1000 draws to get error of ~ 1% on the area

cout << "Draw count: " << draw << endl;
```

Joe

Nice adaptive approach

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <cmath>
using namespace std;

int main(){
// Set up random number generator
const double range_from = -0.5;
const double range_to = 0.5;
random_device rand_dev;
mt19937 generator(rand_dev());
uniform_real_distribution<double> distr(range_from,
range_to);

// Define tolerance
double tolerance = 1e-2;
// Set up counts
double count = 0;
double totalCount = 0;
// Define each loop's estimate for pi and its error,
initialised as a nonzero value
double piGuess;
double absoluteError = 1;
// Initialise values for x and y of each loop
double thisX;
double thisY;
```

```cpp
while( absoluteError > tolerance){
// Generate random values for x and y
thisX = distr(generator);
thisY = distr(generator);
// If point is inside the circle, x^2 + y^2 <= r^2, and r
= 0.5
if (thisX*thisX + thisY*thisY <= 0.25){
// If point is in circle, tick the count up by 1
count++;
}
// Either way, tick up the total count by 1
totalCount++;
// Use count/totalCount = pi r^2 / Area of square
piGuess = 4*(count/totalCount);
// Calculate error based on cmath's value of pi
absoluteError = abs(piGuess - M_PI);
// Print results of each loop
cout << "Trial " << totalCount <<", count is " << count
<< ", piGuess = " << piGuess << endl;
}
return 0;
}
```

Joe

```
$ ./output
Trial 1, count is 1, piGuess = 4
Trial 2, count is 2, piGuess = 4
Trial 3, count is 3, piGuess = 4
Trial 4, count is 3, piGuess = 3
Trial 5, count is 4, piGuess = 3.2
Trial 6, count is 5, piGuess = 3.33333
Trial 7, count is 6, piGuess = 3.42857
```

```
ount is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62076e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62077e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62077e+06, piGuess = 3.14159
Trial 4.6101e+06, count is 3.62077e+06, piGuess = 3.14159
Trial 4.61010e+06, count is 3.62077e+06, piGuess = 3.14159
```

Sakircan

```
int in = 0;
int out = 0;

for(int i =0; i < n_data_points; ++i)
{
if (circle(x_values.at(i), y_values.at(i)) < pow(0.5, 2))
{ in++;
}
else{out++;
}
}

double area = in / n_data_points;
double pi = area / pow(0.5, 2);


cout << in << "\n";
cout << out << "\n";
cout << area << "\n";
cout << pi << "\n";
```

```
$ ./output
100
0
1
4
```

Rupesh

```cpp
#include <iostream>
#include <vector>
#include <random>
using namespace std;
int main() {
const float range_from = -0.5;
const float range_to = 0.5;
float x, y, fractionn = 1, area, errorr = 0.1;
area = 3.141592 * 0.25;
for (int kk = 10; kk < 100; kk = kk + 10) {
int i = 0, j = 0;
for (int k = 0; k < kk; k++) {
i = i + 1;
random_device rand_dev;
mt19937 generator(rand_dev());
uniform_real_distribution<float> distr(range_from, range_to);
x = distr(generator);
y = distr(generator);
double distance_sq = pow(x, 2) + pow(y, 2);
if (distance_sq < 0.25) {
j = j + 1;
}
};
fractionn = (float)j / (float)i; //fraction is area (mult factor is area of square, here
1)
errorr = (area - fractionn) / area;
cout << "The area of circle by Monte Carlo methode is " << fractionn << " with
fractional error " << errorr << " with sample size of " << i << '\n';
};
return 0;
}
```

```
$ ./output
The area of circle by Monte Carlo methode is 0.6 with fractional error 0.236056
with sample size of 10
The area of circle by Monte Carlo methode is 0.65 with fractional error 0.172394
with sample size of 20
The area of circle by Monte Carlo methode is 0.866667 with fractional error -
0.103475 with sample size of 30
The area of circle by Monte Carlo methode is 0.8 with fractional error -0.0185919
with sample size of 40
The area of circle by Monte Carlo methode is 0.8 with fractional error -0.0185919
with sample size of 50
The area of circle by Monte Carlo methode is 0.766667 with fractional error
0.0238495 with sample size of 60
The area of circle by Monte Carlo methode is 0.785714 with fractional error -
0.000402678 with sample size of 70
The area of circle by Monte Carlo methode is 0.7625 with fractional error
0.0291547 with sample size of 80
The area of circle by Monte Carlo methode is 0.766667 with fractional error
0.0238495 with sample size of 90
```

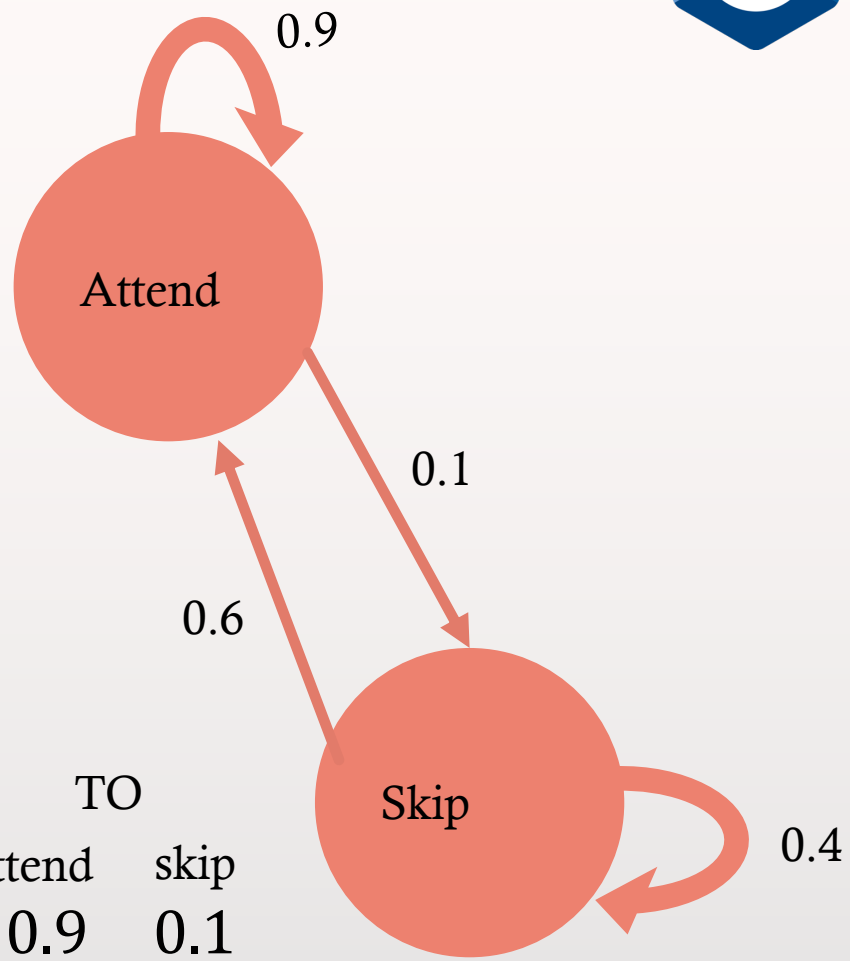
Markov Chain Monte Carlo (MCMC)

- Monte Carlo: estimate the expected value or probability density of some unknown space by drawing independent random values
- For high-dimension probabilistic models, Monte Carlo sampling may not be effective, as volume of sample space grows exponentially with additional parameters
- MCMCs try to sample more intelligently, the next random draw depends on the current one

Andrey Markov

Markov Chains

- A simple Markov Chain uses stochastic processes to determine the evolving state of a system
- Consider this system, it describes whether someone attends class given their previous attendance
- E.g. if you attend class one week, there's a 90% chance you will the next

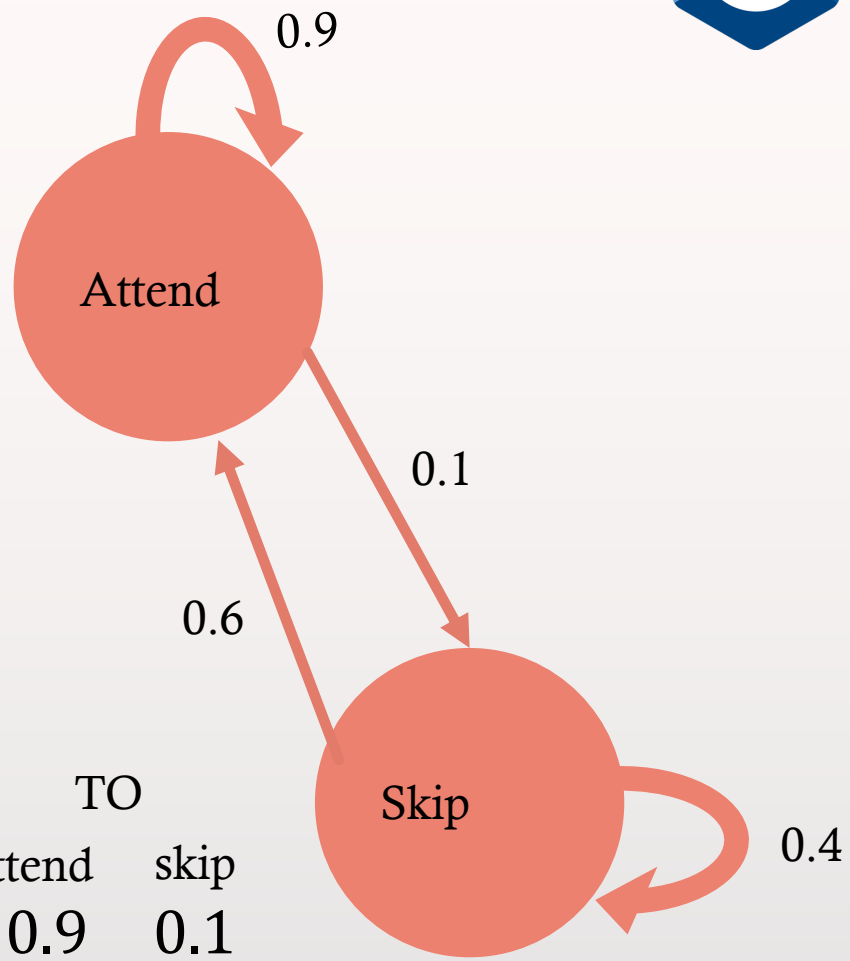


FROM \ TO	attend	skip
attend	0.9	0.1
skip	0.6	0.4

Transition matrix

Markov Chains

- Start with initial state of attendance: $X_0 = [1,0]$
- $X_1 = X_0T = [0.9, 0.1]$
- $X_2 = X_1T = [0.87, 0.13]$
- In the long-run, you approach a **steady state**, i.e. $X_{n+1} = X_n$

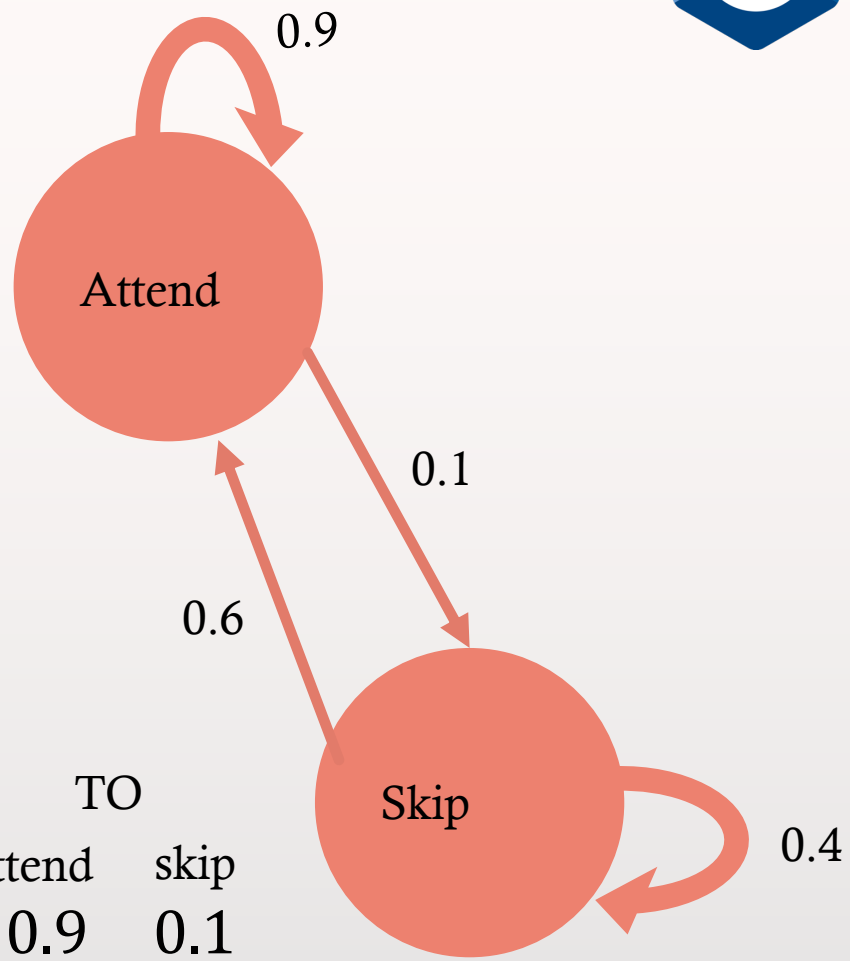


		TO	
		attend	skip
FROM	attend	0.9	0.1
	skip	0.6	0.4

Transition matrix

Markov Chains

- $X_s - X_s T = 0$
- $X_s(I - T) = 0$
- $[x, y]\begin{pmatrix} 0.1 & -0.1 \\ -0.6 & 0.6 \end{pmatrix} = 0$
- $0.1x - 0.6y = 0$ and $x + y = 1$
- $X_s = \left[\frac{6}{7}, \frac{1}{7}\right]$



		TO attend	skip
FROM	attend	0.9	0.1
	skip	0.6	0.4

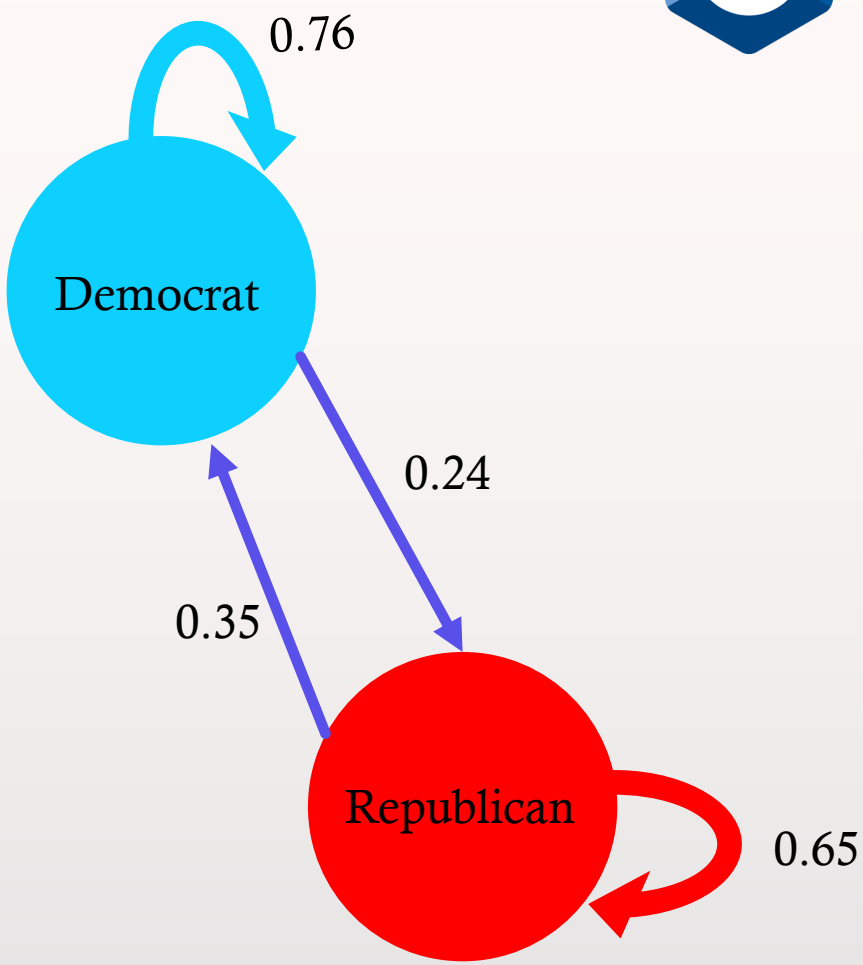
Transition matrix

Markov Chains

- Markov chains can also be used to generate a sequence of random variables where the current value is dependent on the value of the prior value
- An example of this is a number line, where possible moves are -1 and 1 (chosen with equal probability)
- MCMCs are Monte Carlo methods where a Markov chain is used to draw samples
- The idea is that the chain will settle (find equilibrium) on the desired quantity we are inferring

Markov Chains

- Create a class that generates random numbers from a uniform distribution
- Create a Markov Chain class that predicts which US party will win the next election (lookup matrices, matrix mutlipliaction, if statements etc.)
- Assume initially a Dem is in power X0 = [1,0], create a method in the MC class that calculates analytically the steady state vector, i.e. the probability that in a given year either party will be in power
- Create another method that uses random draws from the random number class to stochastically predict who will be in power for each of the next 20 cycles
- Create a figure showing how the holder of office changed over the 20 cycles



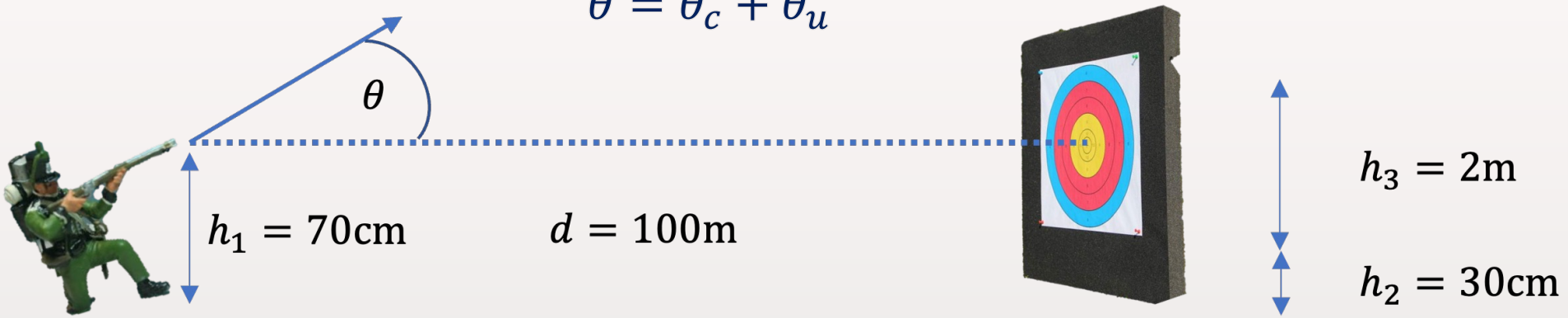
Group Project

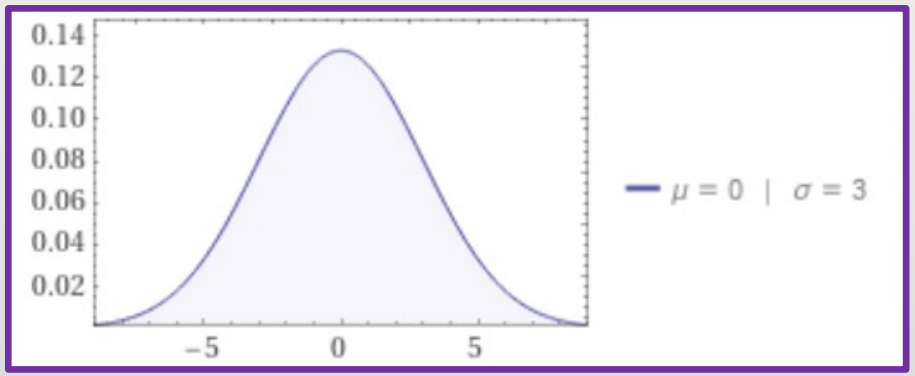
Group Project

- Context: ballistics in the 1800s
- Aim: determine the accuracy of various weapons at a given distance
- alex-hill94.github.io/#Proj

Project Description

$$\theta = \theta_c + \theta_u$$





θ_u is sampled from a normal distribution

Project Description

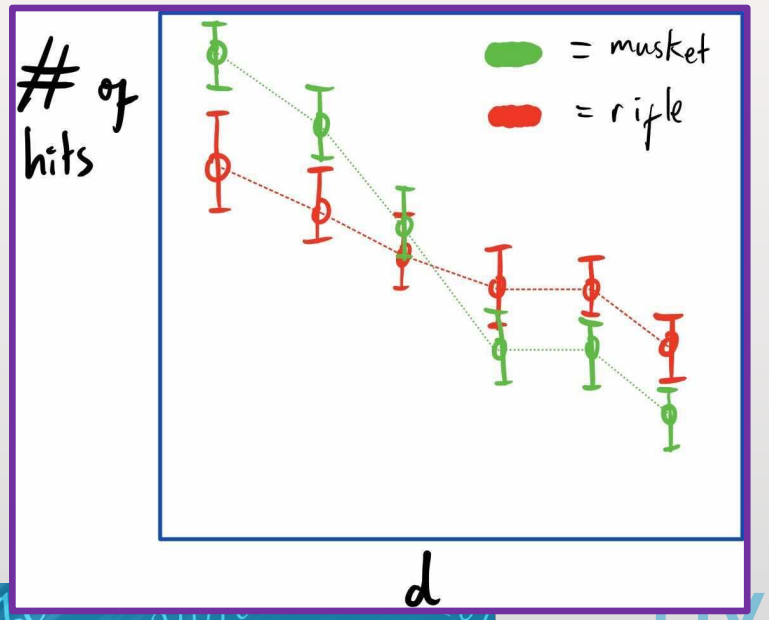
- **Step one**: find the optimal angle required to hit the bullseye.
- **Step two**: set this to be θ_c. Sample θ_u from a normal distribution and add to θ_c to find θ.
- **Step three**: the soldier fires three times per minute. Simulate one 'trial' as being five minutes of firing. How many times does he hit the target?
- **Step four**: run 1000, 10,000, 100,000 trials. What is the distribution of the number of hits?

$$\theta = \theta_c + \theta_u$$

Project Description

- **Step five**: compare the performance of a rifle vs a musket at 100m (details on the main document)

- **Step six**: run the experiment for muskets and rifles at various distances. At what distance does it become better to use one over the other?



Project Delivery

- Group presentation on your planned approach, your code, and your results (10-15 minutes plus time for questions)
- Live demonstration on my laptop, will it compile and run first time?
  - Produce data stream, save data, produce plots…

Tips

- Don’t repeat yourself!
- Plan your approach and allocate tasks before you start coding
- Generalise things as much as possible, and consider where it would be useful to use classes
- Communicate via Slack, or book study rooms in the teaching hub (502)
- Try using GitHub if the project becomes complex

- Group One
  - Sam
  - Emily
  - Marina
  - Rupesh
- Group Two
  - Khang
  - Andrew
  - Ana
  - Sakrican
- Group Three
  - Luke
  - Sinead
  - Mehul
  - Joe

Room: 502-TR4(cap.84)
Date(s): Friday, 24/11/2023
Time: 13:00-15:00


READY STEADY COOK