Computational Methods in Physics (PHY 365)

FA23

Dr. Muhammad Kamran

Department of Physics

COMSATS University Islamabad

Lab 24

Monte Carlo methods

- Monte Carlo methods (MCM) rely on repeated random sampling to obtain numerical results.
- The idea is to use randomness to solve problems that might be deterministic in principle.

Monte Carlo methods

- Monte Carlo methods (MCM) rely on repeated random sampling to obtain numerical results.
- The idea is to use randomness to solve problems that might be deterministic in principle.
- These problems could arise "naturally" as part of the modeling of a real-life system.
  - A complex road network.
  - Neutrons transport in a reactor.
  - Evolution of the stock market.

Monte Carlo methods

- Monte Carlo methods (MCM) rely on repeated random sampling to obtain numerical results.
- The idea is to use randomness to solve problems that might be deterministic in principle.
- These problems could arise "naturally" as part of the modeling of a real-life system.
  - A complex road network.
  - Neutrons transport in a reactor.
  - Evolution of the stock market.
- In many cases, however, the random objects in MCM are introduced "artificially" in order to solve purely deterministic problems.
- In this case the MCM simply involves random sampling from certain probability distributions.

Monte Carlo methods

- MCM are mainly used in three distinct problem classes.

Monte Carlo methods

- MCM are mainly used in three distinct problem classes.
  - Optimization.

Monte Carlo methods

- MCM are mainly used in three distinct problem classes.
  - Optimization.
  - Numerical integration.

Monte Carlo methods

- MCM are mainly used in three distinct problem classes.
  - Optimization.
  - Numerical integration.
  - Generating draws from a probability distribution (sampling).

Monte Carlo methods

- MCM are mainly used in three distinct problem classes.
  - Optimization.
  - Numerical integration.
  - Generating draws from a probability distribution (sampling).
- MCM vary, but tend to follow a particular pattern:
  - Define a domain of possible inputs.
  - Generate inputs randomly from a probability distribution over the domain.
  - Perform a deterministic computation on the inputs.
  - Aggregate the results.

Monte Carlo methods

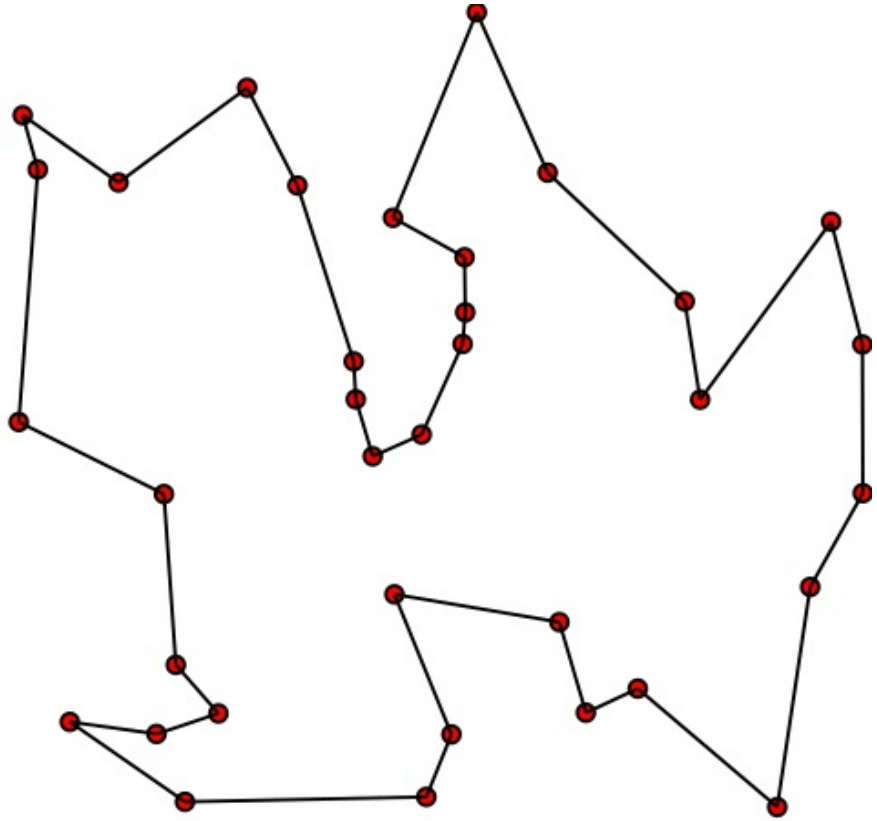


Figure: The traveling salesman problem

Monte Carlo methods

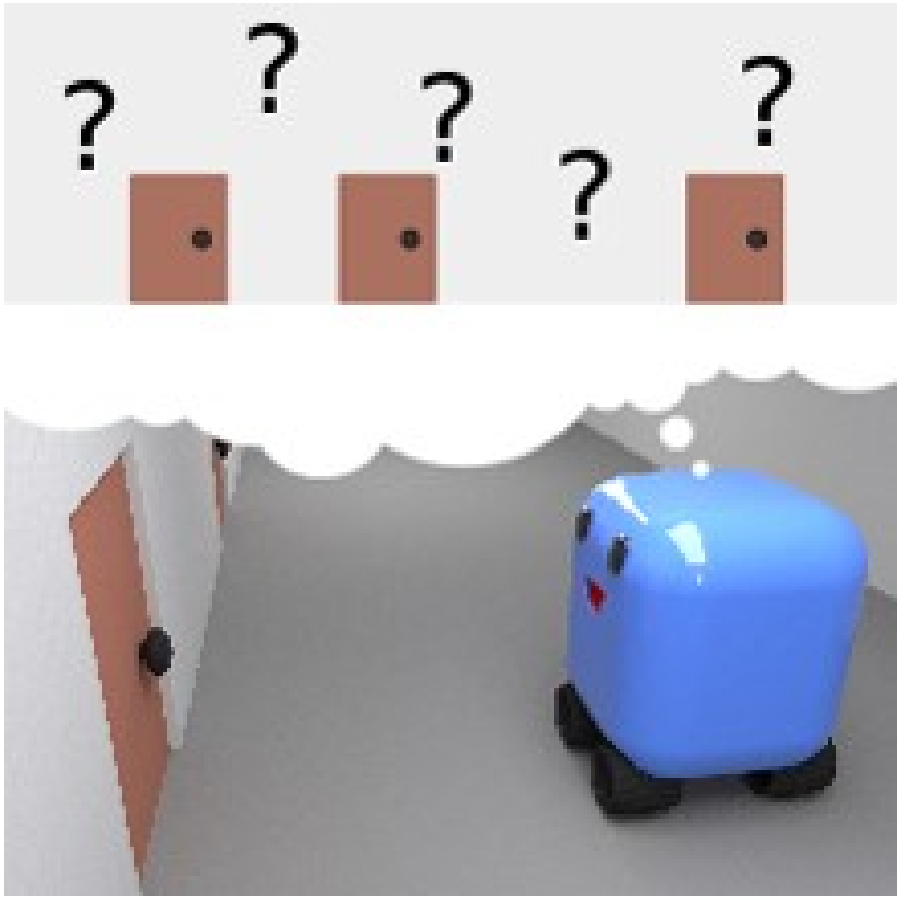


Figure: Monte Carlo localization is an algorithm to let a robot determine its position based on its sensor observations

Monte Carlo methods

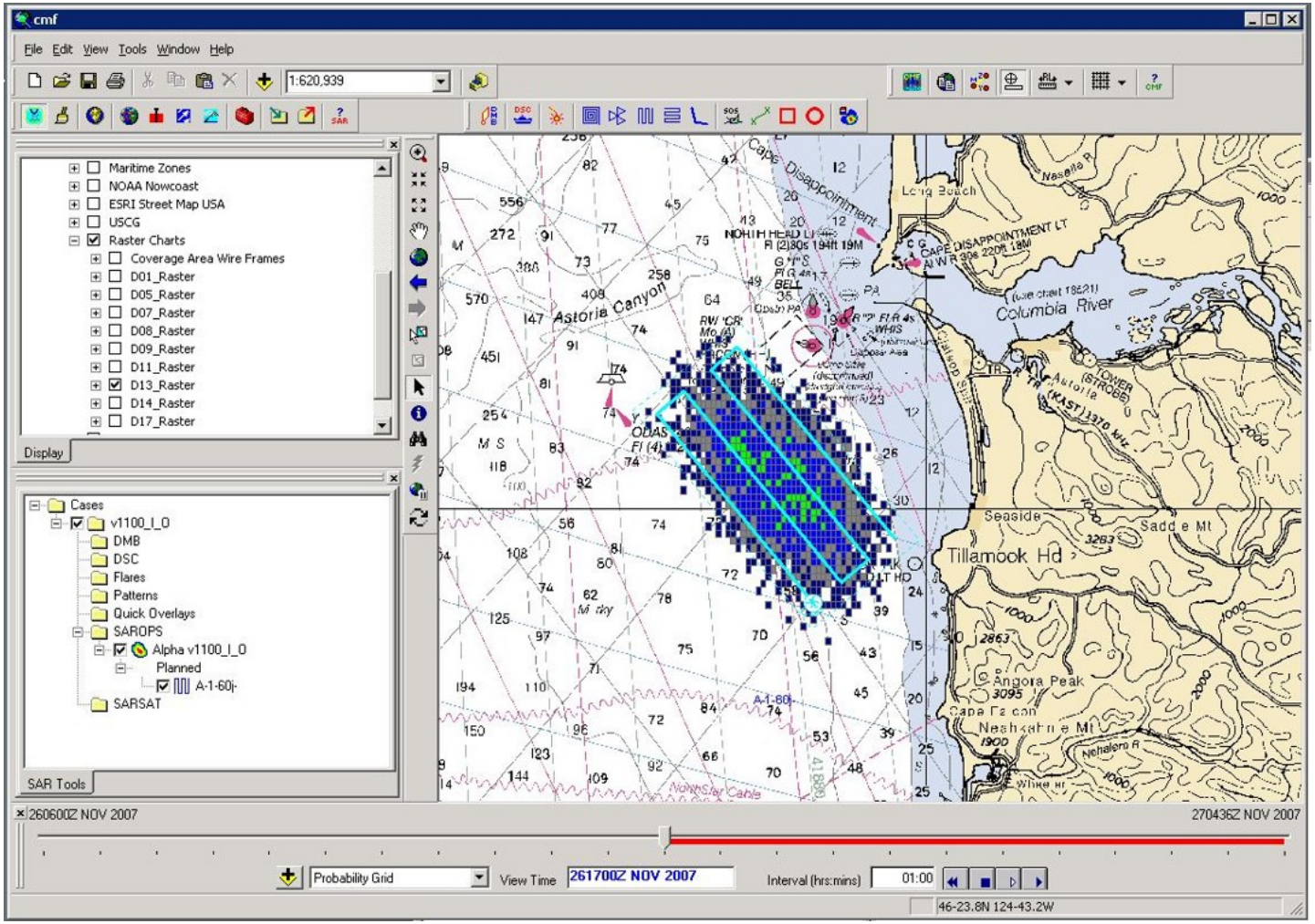


Figure: SAROPS was utilized in the response to the Deepwater Horizon explosion

Monte Carlo methods

Figure: Go is a complex board game that requires intuition, creative and strategic thinking

Monte Carlo methods

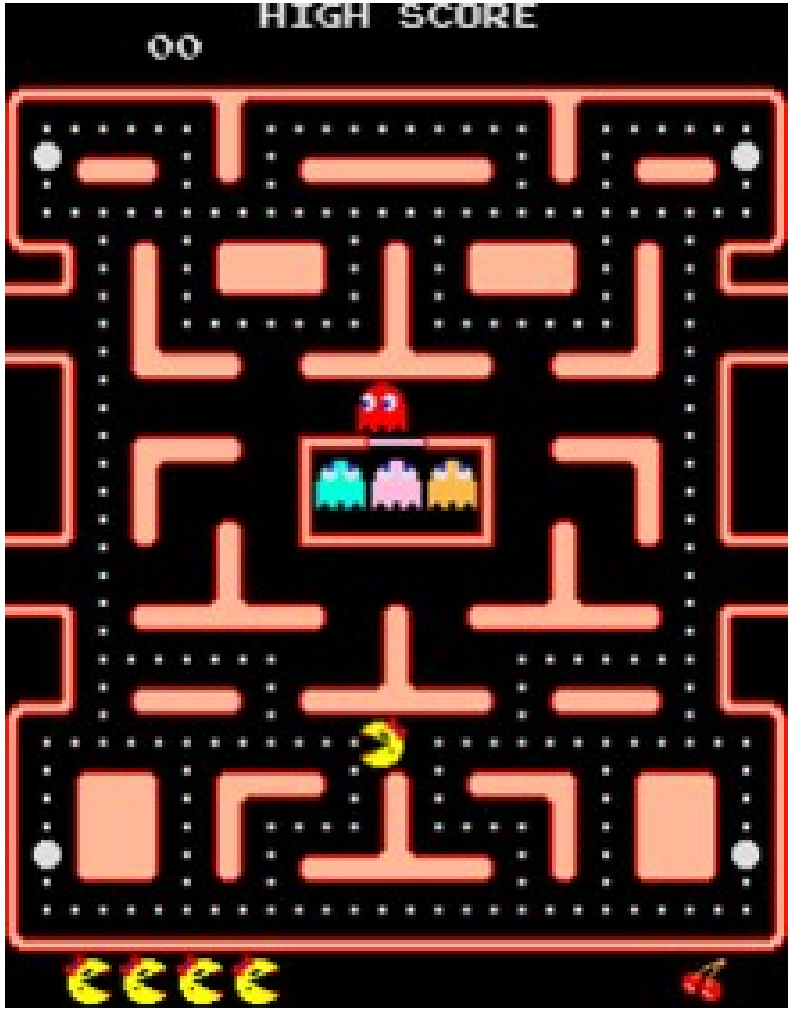


Figure: Ms. Pac-Man is a 1982 maze arcade game developed by General Computer Corporation and published by Midway

Monte Carlo methods

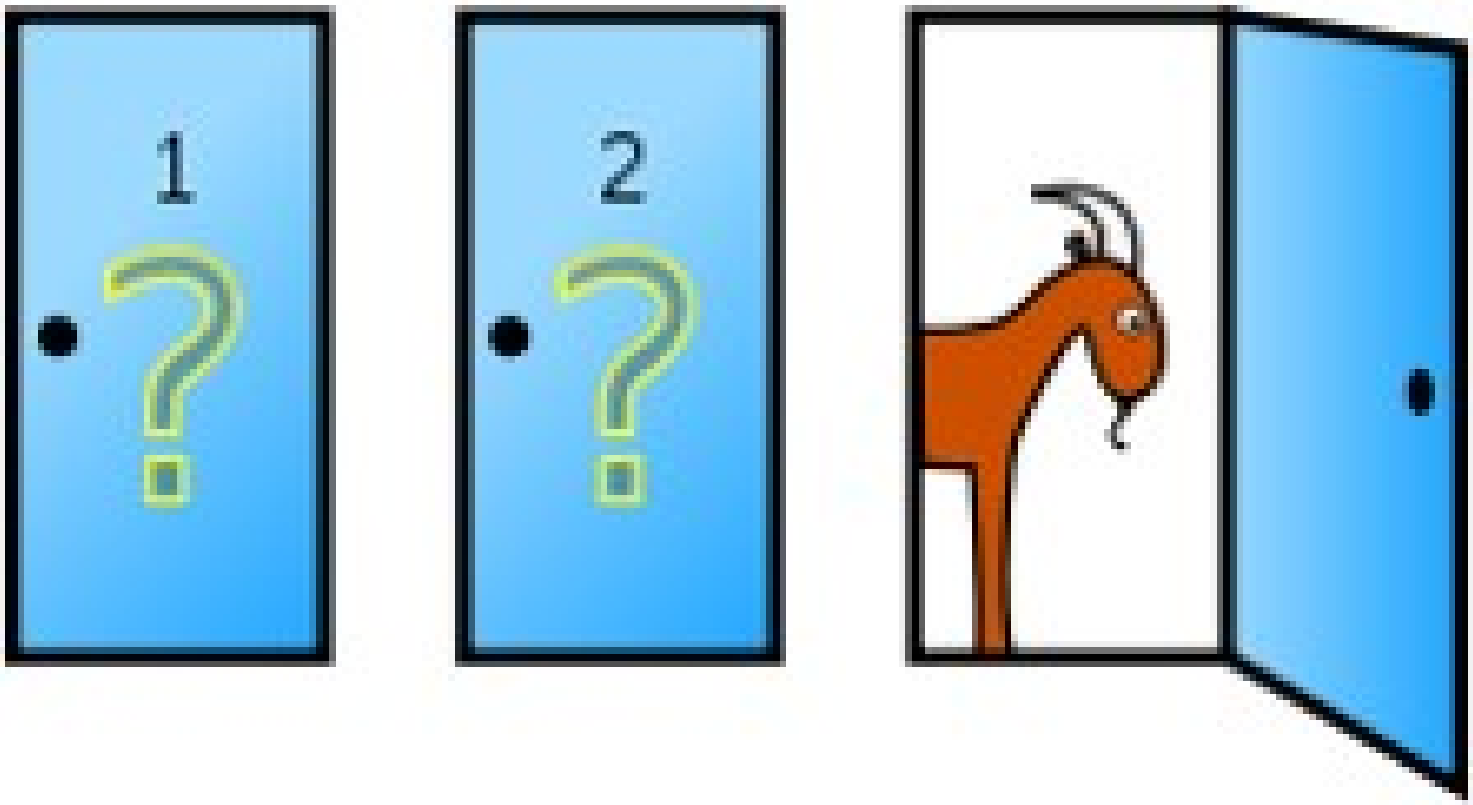


Figure: In search of a new car, the player picks a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player switch from door 1 to door 2

MATLAB's rng function

- rng(seed) seeds the random number generator using the nonnegative integer seed.
  - Every time a predictable sequence of numbers is generated.

MATLAB's rng function

- rng(seed) seeds the random number generator using the nonnegative integer seed.
  - Every time a predictable sequence of numbers is generated.
- rng('shuffle') seeds the random number generator based on the current time.
  - Every time a different sequence of numbers is generated.

MATLAB's rng function

- rng(seed) seeds the random number generator using the nonnegative integer seed.
  - Every time a predictable sequence of numbers is generated.
- rng('shuffle') seeds the random number generator based on the current time.
  - Every time a different sequence of numbers is generated.
- rng(___, generator) additionally specifies the type of the random number generator.

MATLAB's rng function

- rng(seed) seeds the random number generator using the nonnegative integer seed.
  - Every time a predictable sequence of numbers is generated.
- rng('shuffle') seeds the random number generator based on the current time.
  - Every time a different sequence of numbers is generated.
- rng(___, generator) additionally specifies the type of the random number generator.
- rng('default') puts the settings of the random number generator to the default ones.
  - This way, the same random numbers are produced as if you restarted MATLAB.
  - The default settings are the Mersenne Twister with seed 0.

MATLAB's rng function

- s = rng returns the current settings of the RNG.
- rng(s) restores the settings of the RNG back to the values captured previously.

MATLAB's rng function

- s = rng returns the current settings of the RNG.
- rng(s) restores the settings of the RNG back to the values captured previously.

RNG	Based on
'twister'	Mersenne Twister
'simdTwister'	SIMD-oriented Fast Mersenne Twister
'combRecursive'	Combined Multiple Recursive
'multFibonacci'	Multiplicative Lagged Fibonacci
'v5uniform'	Legacy MATLAB 5.0 uniform generator
'v5normal'	Legacy MATLAB 5.0 normal generator
'v4'	Legacy MATLAB 4.0 generator

Table: MATLAB's RNG types

MATLAB's rand function

- X = rand returns a single uniformly distributed random number in the interval (0,1).
- X = rand(n) returns an n-by-n matrix of random numbers.

MATLAB's rand function

- X = rand returns a single uniformly distributed random number in the interval (0,1).
- X = rand(n) returns an n-by-n matrix of random numbers.
- X = rand(sz1,...,szN) returns an sz1-by-...-by-szN array of random numbers where sz1, ..., szN indicate the size of each dimension.

MATLAB's rand function

- X = rand returns a single uniformly distributed random number in the interval (0,1).
- X = rand(n) returns an n-by-n matrix of random numbers.
- X = rand(sz1,...,szN) returns an sz1-by-...-by-szN array of random numbers where sz1, ..., szN indicate the size of each dimension.
- X = a + (b – a) * rand(n) returns an n-by-n matrix of random numbers in the interval [a, b].

MATLAB's rand function

- X = rand returns a single uniformly distributed random number in the interval (0,1).
- X = rand(n) returns an n-by-n matrix of random numbers.
- X = rand(sz1,...,szN) returns an sz1-by-...-by-szN array of random numbers where sz1, ..., szN indicate the size of each dimension.
- X = a + (b – a) * rand(n) returns an n-by-n matrix of random numbers in the interval [a, b].
- X = rand(___, typename) returns an array of random numbers of data type typename.
  - The typename input can be either 'single' or 'double'.

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval $[-3\ ,\ 5]$.

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval $[-3\ ,\ 5]$.

```
x = -3 + 8 * rand(3,4)
```

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval [−3 , 5].

  x = −3 + 8 * rand(3,4)

- Problem: Generate a 3×4×5 array of random numbers between 0 and 1.

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval [−3 , 5].

  x = −3 + 8 * rand(3,4)

- Problem: Generate a 3×4×5 array of random numbers between 0 and 1.

  x = rand(3 , 4 , 5)

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval [−3 , 5].

  x = −3 + 8 * rand(3,4)

- Problem: Generate a 3×4×5 array of random numbers between 0 and 1.

  x = rand(3 , 4 , 5)

- Problem: Generate twice a specific sequence of random numbers.

MATLAB's rand function

- Problem: Generate a 3×4 matrix of random numbers in the interval [−3 , 5].

```
x = −3 + 8 * rand(3,4)
```

- Problem: Generate a 3×4×5 array of random numbers between 0 and 1.

```
x = rand(3 , 4 , 5)
```

- Problem: Generate twice a specific sequence of random numbers.

```
s = rng;
rand(3)
rng(s)
rand(3)
```

MATLAB's randi function

- X = randi(imax) returns a pseudorandom scalar integer between 1 and imax.
- X = randi(imax,n) returns an n-by-n matrix of pseudorandom integers drawn from the discrete uniform distribution on the interval [1,imax]

MATLAB's randi function

- X = randi(imax) returns a pseudorandom scalar integer between 1 and imax.
- X = randi(imax,n) returns an n-by-n matrix of pseudorandom integers drawn from the discrete uniform distribution on the interval [1,imax]
- X = randi(imax,sz1,...,szN) returns an sz1-by-...-by-szN array.
  - → sz1,...,szN indicates the size of each dimension.
  - → randi(10,3,4) returns a 3-by-4 array of pseudorandom integers between 1 and 10.

MATLAB's randi function

- X = randi(imax) returns a pseudorandom scalar integer between 1 and imax.
- X = randi(imax,n) returns an n-by-n matrix of pseudorandom integers drawn from the discrete uniform distribution on the interval [1,imax]
- X = randi(imax,sz1,...,szN) returns an sz1-by-...-by-szN array.
  - → sz1,...,szN indicates the size of each dimension.
  - → randi(10,3,4) returns a 3-by-4 array of pseudorandom integers between 1 and 10.
- X = randi([imin,imax],___) returns an array containing integers drawn from the discrete uniform distribution on the interval [imin,imax], using any of the above syntaxes.

Value of π

- There are many methods to calculate the value of π using MCM.

Value of π

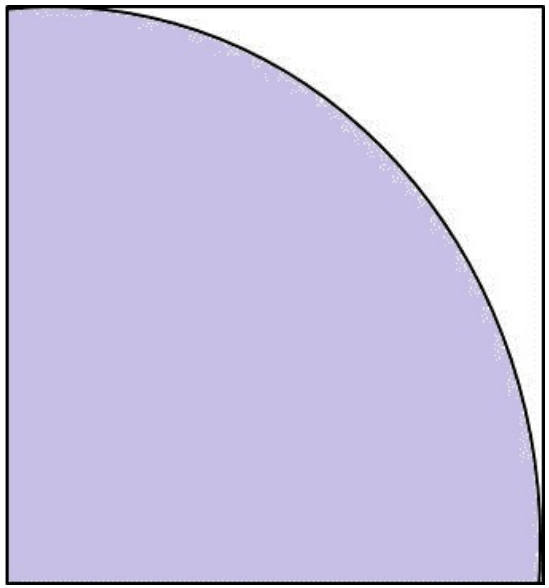
- There are many methods to calculate the value of π using MCM.
- In the method we use here
  - A square is drawn, whose side length is 'r'.

Value of π

- There are many methods to calculate the value of π using MCM.
- In the method we use here
  - A square is drawn, whose side length is 'r'.
  - A (quarter) circle of radius 'r' in inscribed in the square.

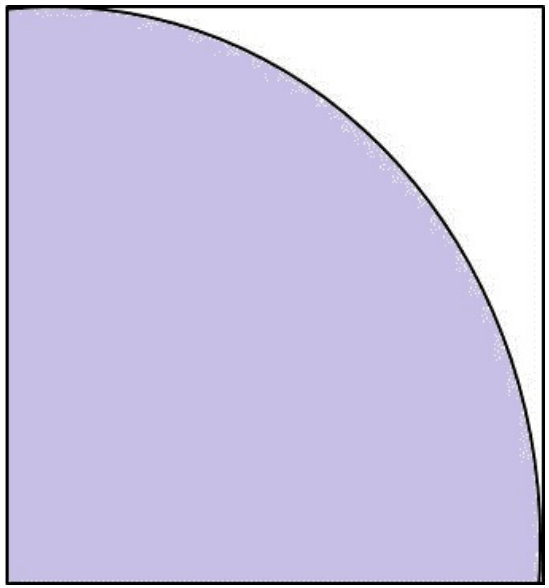
Value of π

- There are many methods to calculate the value of π using MCM.
- In the method we use here
  - A square is drawn, whose side length is 'r'.
  - A (quarter) circle of radius 'r' in inscribed in the square.



Value of π

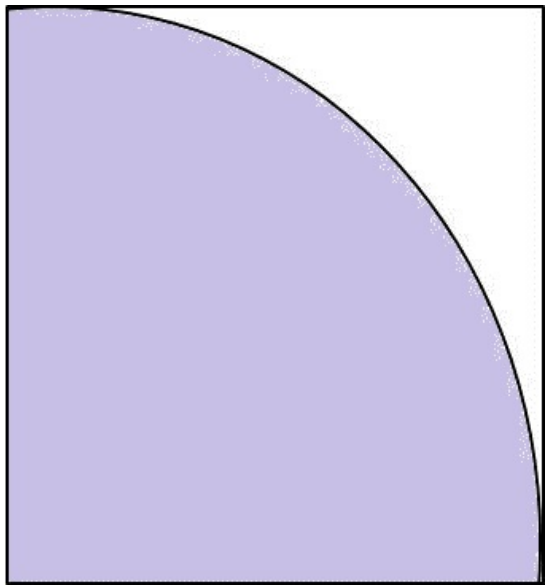
- There are many methods to calculate the value of π using MCM.
- In the method we use here
  - A square is drawn, whose side length is 'r'.
  - A (quarter) circle of radius 'r' in inscribed in the square.



- The ratio of circle's area to the square's area is equal to

Value of π

- There are many methods to calculate the value of π using MCM.
- In the method we use here
  - A square is drawn, whose side length is 'r'.
  - A (quarter) circle of radius 'r' in inscribed in the square.



- The ratio of circle's area to the square's area is equal to $\pi/4$.

Value of π

- Problem: Calculate the value of π using MCM.

Value of π

- Problem: Calculate the value of π using MCM.

- Generating random numbers

```
Total_no_pnts = input('Please give the number of random points: ');
disp(' ')
x = rand (Total_no_pnts,1);
y = rand (Total_no_pnts,1);
```

Value of π

- Problem: Calculate the value of π using MCM.

- Generating random numbers

```
Total_no_pnts = input('Please give the number of random points: ');
disp(' ')
x = rand (Total_no_pnts,1);
y = rand (Total_no_pnts,1);
```

- Radius of the circle

```
r = sqrt(x .^2 + y .^2);
```

Value of π

- Separating the points inside the circle of unit radius from the outside ones

```
points_inside_circle = find (r <= 1);
points_outside_circle = find (r > 1);
inside_points_x = x (points_inside_circle);
inside_points_y = y (points_inside_circle);
outside_points_x = x (points_outside_circle);
outside_points_y = y(points_outside_circle);
```

Value of π

- Separating the points inside the circle of unit radius from the outside ones

```
points_inside_circle = find (r <= 1);
points_outside_circle = find (r > 1);
inside_points_x = x (points_inside_circle);
inside_points_y = y (points_inside_circle);
outside_points_x = x (points_outside_circle);
outside_points_y = y(points_outside_circle);
```

- No. of points inside the circle

```
no_inside = length(points_inside_circle);
```

Value of π

- The value of pi

```
value_of_pi = 4 * no_inside / Total_no_pnts;
disp([ 'The calculated value of pi = ', num2str
(value_of_pi) ])
```

Value of π

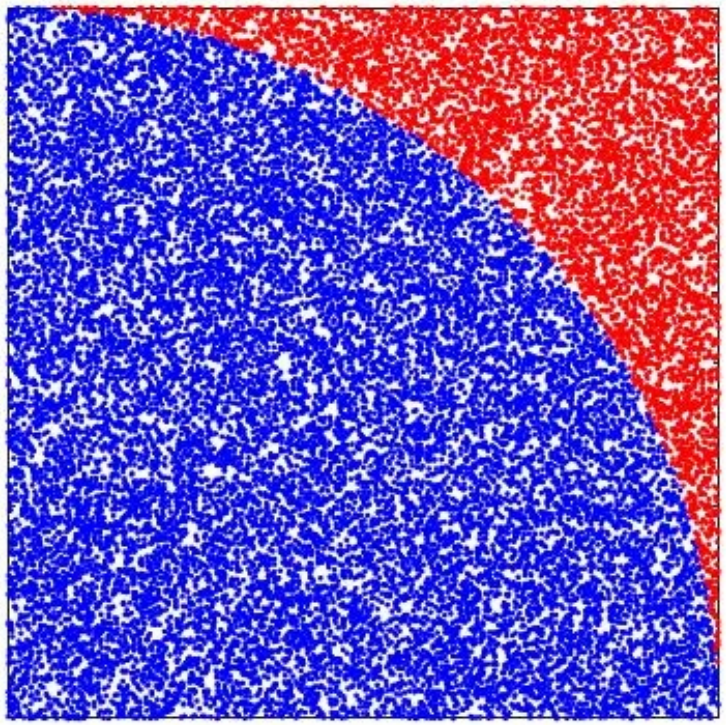
- The value of pi

```
value_of_pi = 4 * no_inside / Total_no_pnts;
disp([ 'The calculated value of pi = ', num2str
(value_of_pi) ])
```

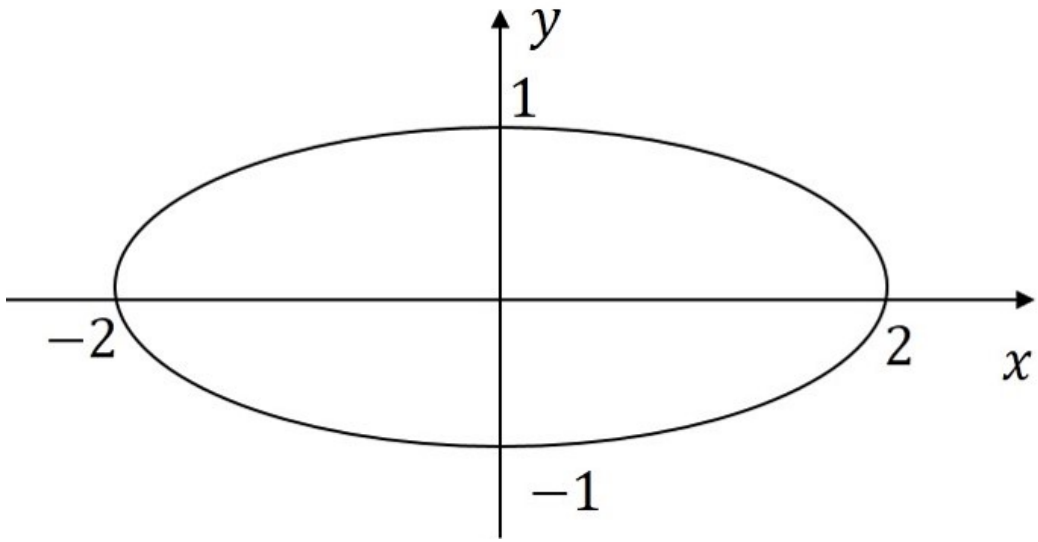
- Plotting

```
plot(inside_points_x, inside_points_y, 'b.')
hold on
plot(outside_points_x, outside_points_y, 'r.')
hold off
axis square
```

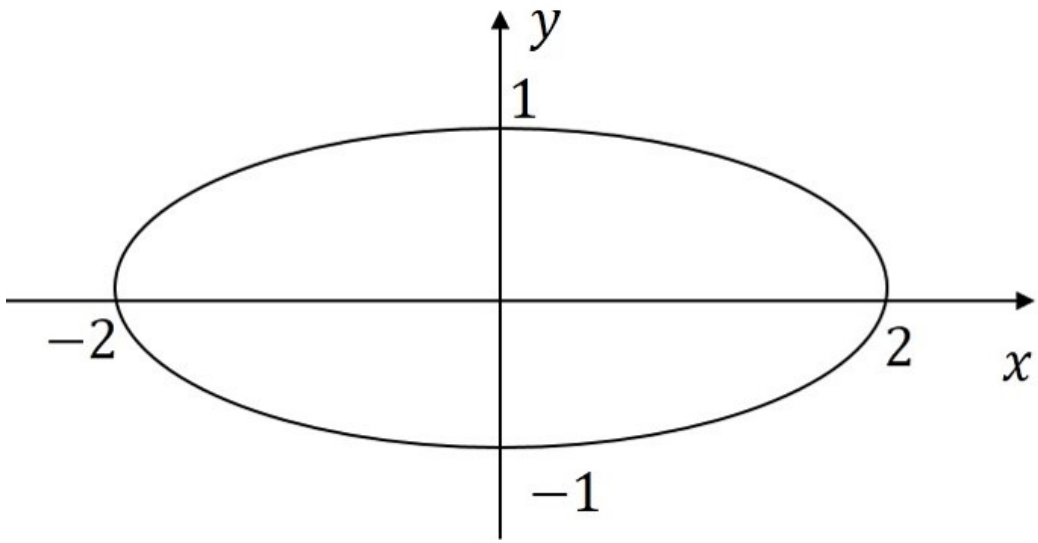
Value of π



- Problem: Generate random points uniformly distributed inside the ellipse $x^2 + 4y^2 = 4$.



- Problem: Generate random points uniformly distributed inside the ellipse $x^2 + 4y^2 = 4$.



- Hints
  - Generate random numbers in the ranges $-2 \le x \le 2, \quad -1 \le y \le 1$.
  - Use the rejection technique.

References

- https://en.wikipedia.org/wiki/Monte_Carlo_method
- https://en.wikipedia.org/wiki/Travelling_salesman_problem
- https://en.wikipedia.org/wiki/Monte_Carlo_localization
- https://en.wikipedia.org/wiki/Search_and_Rescue_Optimal_Planning_System
- https://en.wikipedia.org/wiki/Go_(game)
- https://en.wikipedia.org/wiki/Pac-Man
- https://en.wikipedia.org/wiki/Monty_Hall_problem
- https://www.mathworks.com/help/matlab/ref/rng.html
- https://www.mathworks.com/help/matlab/random-number-generation.html

References

- https://en.wikipedia.org/wiki/Mersenne_Twister
- https://en.wikipedia.org/wiki/Lagged_Fibonacci_generator
- https://www.mathworks.com/help/matlab/ref/rand.html
- https://www.mathworks.com/help/matlab/ref/randi.html
- https://www.youtube.com/watch?v=ADA82D0j9HY