CÁLCULO DE PI

MÉTODO DE MONTE CARLO - GRUPO 5

Brena Freitas
RA - 2465710

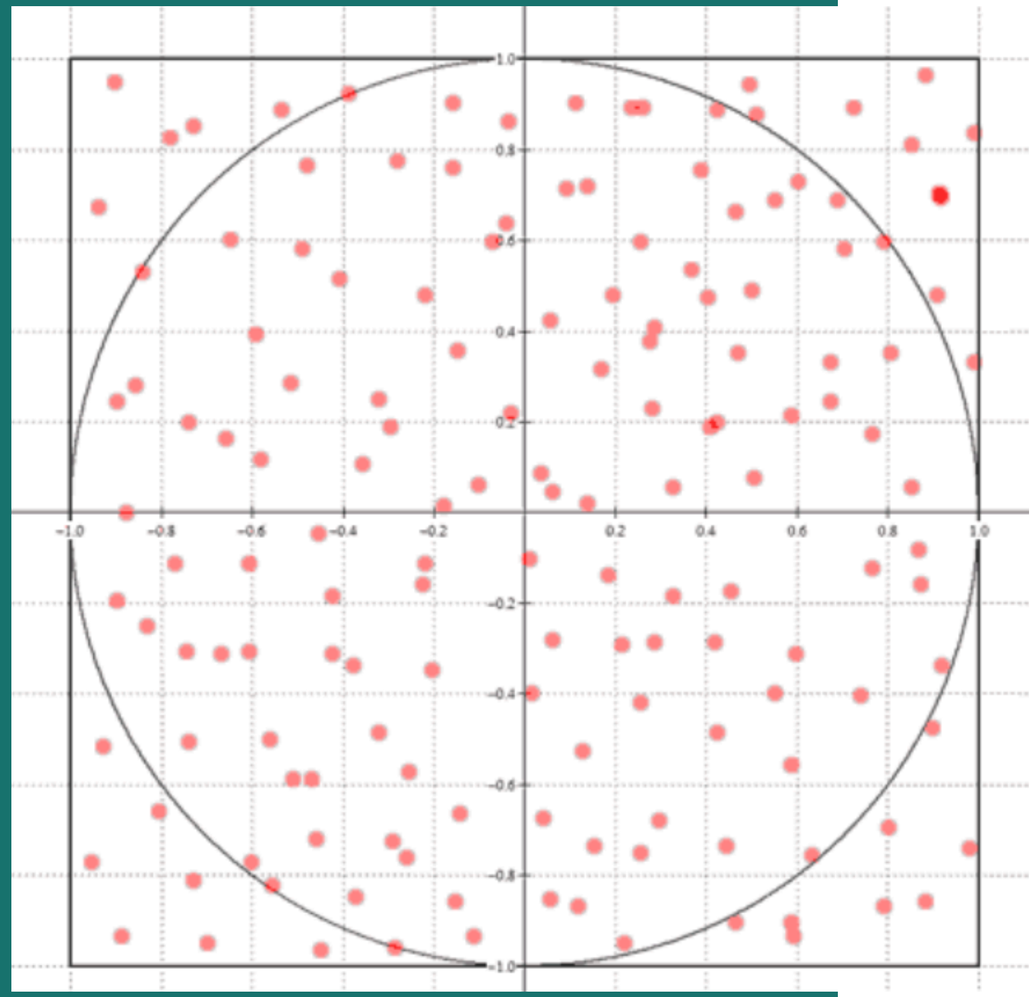
Lucas Carvalho Silva
RA - 1997068

Rafael Gonçalves
RA 2053845

Vitória Cardoso
RA - 2312174

Vitória Millnitz
RA - 2210070

SUMÁRIO



MÉTODO MONTE CARLO

O cálculo de π com o método de Monte Carlo baseia-se na ideia de que a proporção de pontos gerados aleatoriamente dentro de um círculo inscrito em um quadrado pode ser usada para calcular π.

EXEMPLOS DE APLICAÇÕES

A técnica de Monte Carlo é amplamente utilizada em várias áreas para resolver problemas complexos. Segue alguns exemplos reais:

- Física médica:
  - A física médica vem utilizado dessa abordagem para fazer diagnósticos.

Yoriyaz, H.m , (2015). Método de Monte Carlo: princípios e aplicações em Física Médica.

- Finanças e Estatísticas:
  - Quando se utiliza essa abordagem, há uma redução significativa na probabilidade de erros

Carvalho, A. R. (2017)
Método Monte Carlo e suas aplicações.

SEQUENCIAL

Gerador de pontos randomicos

```python
import random

def generateRandomPoints(numberOfPoints):

    for _ in range (numberOfPoints):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        yield x, y
```

SEQUENCIAL

Calculadora de PI

```python
from src.utils.generator import generateRandomPoints

def calculatePi(numberOfPoints):

  pointsInsideTheCircle = 0

  for x, y in generateRandomPoints(numberOfPoints):
    if x**2 + y**2 <= 1:
      pointsInsideTheCircle += 1


  estimatedPi = 4 * (pointsInsideTheCircle/ float(numberOfPoints))
  return estimatedPi
```

SEQUENCIAL

Execução

```
import time

def main():
  print("=> Cálculo de Pi pelo Método de Monte Carlo <=")

  try:
    startTime = time.time()
    numberOfPoints = int(input("Digite a quantidade de pontos que devem ser simulados: "))
    if numberOfPoints <= 0:
      raise ValueError("Digite um valor valído, precisa ser um número maior do que 0")
  except ValueError as e:
    print(f"Entrada inválida: {e}")
    return

  estimatedPi = calculatePi(numberOfPoints)
  print(f"Com {numberOfPoints} pontos, o valor estimado de Pi é {estimatedPi:.6f}")
  endTime = time.time()

  executionTime = (endTime - startTime) * 1000
  print("Tempo de execução =>", executionTime)

if __name__ == "__main__":
  main()
```

DISTRIBUÍDO

O cálculo de π é dividido em várias máquinas ou instâncias computacionais (nós), onde cada nó realiza uma parte do trabalho, e os resultados parciais são agregados ao final.

CLIENTE

DISTRIBUIDA

```python
def worker(host, port, points_to_calculate, result_queue):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((host, port))

        batch_size = 1000000
        remaining = points_to_calculate

        while remaining > 0:
            current_batch = min(batch_size, remaining)
            client_socket.send(str(current_batch).encode())
            response = client_socket.recv(1024).decode()

            if response.startswith("ERRO"):
                print(f"Erro: {response[6:]}")
                break
            else:
                dentro, total = map(int, response.split(','))
                result_queue.put((dentro, total))
                remaining -= current_batch

    except Exception as e:
        print(f"Erro na thread: {str(e)}")
    finally:
        client_socket.close()
```

CLIENTE

DISTRIBUIDA

```python
def connect_to_server():
    host = 'localhost'
    port = 5000

    try:
        print("=> Cálculo de Pi pelo Método de Monte Carlo <=")
        total_points = int(input("Digite a quantidade de pontos que devem ser simulados: ")
        startTime = time.time()

        num_threads = 5
        points_per_thread = total_points // num_threads
        remaining_points = total_points % num_threads

        result_queue = Queue()
        threads = []

        for i in range(num_threads):
            points = points_per_thread + (remaining_points if i == num_threads-1 else 0)
            thread = threading.Thread(
                target=worker,
                args=(host, port, points, result_queue)
            )
            threads.append(thread)
            thread.start()
            print(f"Thread {i+1} iniciada com {points} pontos")

        pontos_dentro_total = 0
        pontos_totais = 0
        resultados_processados = 0
```

```python
        while resultados_processados < total_points:
            try:
                dentro, total = result_queue.get(timeout=1)
                pontos_dentro_total += dentro
                pontos_totais += total
                resultados_processados += total

                pi_parcial = 4 * pontos_dentro_total / pontos_totais
                progresso = (pontos_totais * 100) // total_points
                print(f"Progresso: {progresso}% - Pi estimado atual: {pi_parcial:.6f}")

            except Queue.Empty:
                continue

        for thread in threads:
            thread.join()

        if pontos_totais > 0:
            pi_final = 4 * pontos_dentro_total / pontos_totais
            endTime = time.time()
            print(f"\nResultado final com {pontos_totais} pontos:")
            print(f"Valor estimado de Pi: {pi_final:.6f}")
            executionTime = (endTime - startTime) * 1000
            print("Tempo de execução =>", executionTime)

    except ConnectionRefusedError:
        print("Não foi possível conectar ao servidor. Verifique se ele está em execução.")
    except Exception as e:
        print(f"Erro: {str(e)}")
```

SERVIDOR

```python
def calculate_pi_points(num_points):
    points_inside = 0
    total_points = num_points

    batch_size = 1000000
    remaining_points = num_points

    while remaining_points > 0:
        current_batch = min(batch_size, remaining_points)

        for x, y in generateRandomPoints(current_batch):
            if x**2 + y**2 <= 1:
                points_inside += 1

        remaining_points -= current_batch

    return points_inside, total_points
```

DISTRIBUIDA

```python
def start_server():
    host = 'localhost'
    port = 5000

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)

    print("Servidor iniciado. Aguardando conexões...")

    try:
        while True:
            client_socket, address = server_socket.accept()
            print(f"Cliente conectado: {address}")

            try:
                while True:
                    data = client_socket.recv(1024).decode()
                    if not data:
                        break

                    try:
                        num_points = int(data)
                        points_inside, total_points = calculate_pi_points(num_points)
                        response = f"{points_inside},{total_points}"
                        client_socket.send(response.encode())
```

PARALELO

- O código utiliza programação paralela para dividir a carga de trabalho entre várias threads, acelerando o processo para um grande número de pontos.

- numPontos: O número total de pontos a serem gerados para o cálculo.
- numThreads: O número de threads que serão usadas para executar o cálculo em paralelo

Vantagens

- escalabilidade
- redução do tempo

PARALELO

```java
public class GerenciadorParalelo {
    private final long numPontos; // Número total de pontos
    private final int numThreads; // Número de threads

    public GerenciadorParalelo(long totalPontos, int numThreads) {
        this.numPontos = totalPontos;
        this.numThreads = numThreads > 0 ? numThreads : Runtime.getRuntime().availableProcessors();
    }

    public List<PointData> iniciarCalculo() {
        System.out.println("Iniciando cálculo com " + numThreads + " threads e " + numPontos + " pontos...");

        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        long pontosPorThread = numPontos / numThreads;

        // Lista para armazenar os resultados das threads
        ArrayList<Future<List<PointData>>> resultados = new ArrayList<>();

        // Submete as tarefas para as threads
        for (int i = 0; i < numThreads; i++) {
            Worker worker = new Worker(pontosPorThread);
            resultados.add(executor.submit(worker));
        }
```

PARALELO

```
executor.shutdown();

// Combina os resultados
List<PointData> todosPontos = new ArrayList<>();
try {
    for (Future<List<PointData>> resultado : resultados) {
        todosPontos.addAll(resultado.get()); // Aguarda o resultado de cada thread
    }
} catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
}

// Calcula a estimativa final de Pi
long totalPontosDentroDoCirculo = todosPontos.stream().filter(PointData::isDentroDoCirculo).count
double estimativaPi = 4.0 * totalPontosDentroDoCirculo / numPontos;
System.out.println("Estimativa final de Pi: " + estimativaPi);

return todosPontos;
}
```

COMPARATIVO

Quantidade de pontos	Sequencial	Paralela	Distribuida
1000 pontos	1 ms	28 ms	9 ms
100 000 pontos	78 ms	43 ms	95 ms
10 000 000 pontos	7.72 s	0.6 s	7.78 s
100 000 000 pontos	77.7 s	PC trava	77.5 s
1 000 000 000 pontos	13 min	PC trava	12 min
10 000 000 000 pontos	+1 dia	PC trava	128 min

DESAFIOS E SOLUÇÕES

DESAFIOS

Definir a quantidade de threads ideais

Compreender o cálculo de monte carlo

SOLUÇÕES

Estudamos as principais aplicações e fizemo
reuniões para que todos compreendessem.

A definição de Threads foi através de testes
realizados.

PARTICIPANTES

COM O QUE CADA UM CONTRIBUIU

01 **Implementação de código e testes.**

Rafael Gonçalves e Lucas Carvalho Silva

02 **Pesquisa e Apresentação visual.**

Brena Freitas, Vitória Cardoso e Vitória Millnitz.

REFERÊNCIAS

1. **Carvalho, A. R. (2017)** Método Monte Carlo e suas aplicações. Mestrado Profissional em Matemática em Rede Nacional, Universidade Federal de Roraima, Boa Vista, RR.
2. **Paula, R. R. (2014)** Método de Monte Carlo e Aplicações. Volta Redonda, RJ.
3. **Paixão, J. L.,** Lima, D. A. C., Fabrin, F. G., Santana, G. C., Baldissera, L. B., & Silva, R. N. (s.d.) Métodos matemáticos de modelagem e otimização: teoria e aplicações do método de Monte Carlo. Apresentado na XXVI Jornada de Pesquisa.
4. **Yoriyaz, H. (2015**) Método de Monte Carlo: princípios e aplicações em Física Médica. Revista Brasileira de Física Médica, 3(1), 141–149. https://doi.org/10.29384/rbfm.2009.v3.n1.p141-149

OBRIGADO