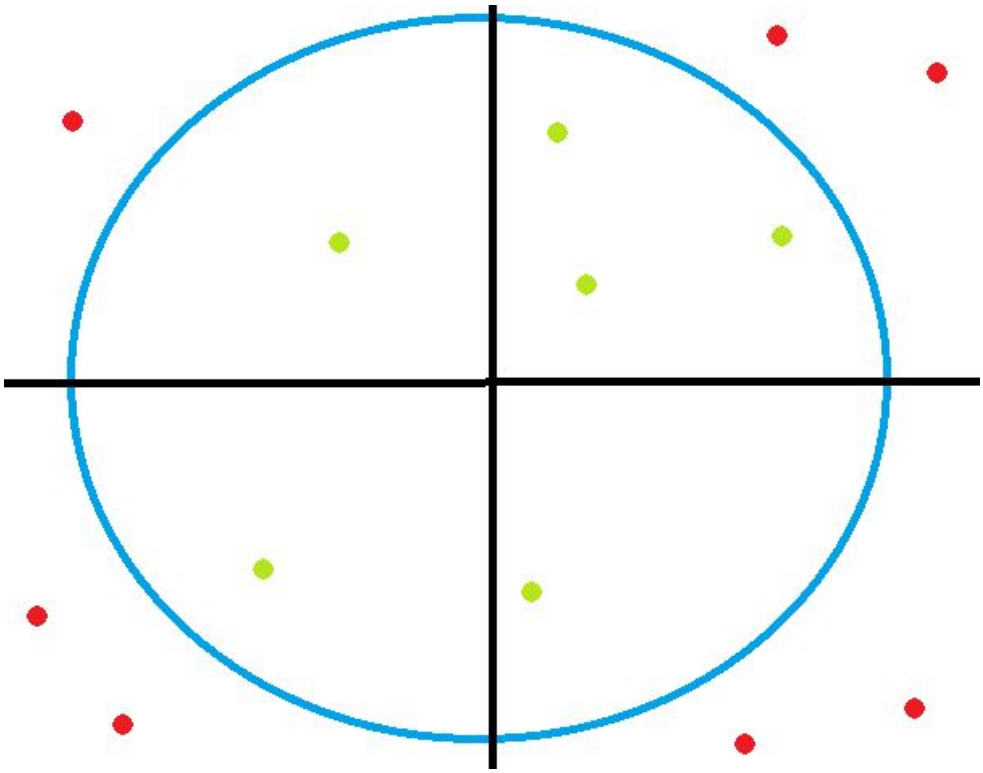
Aluno: Igor Cervelin

Cálculo do Pi pelo método de Monte Carlo utilizando Python Parallel

O método de Monte Carlo consiste em utilizar elementos de aleatoriedade para a resolução de problemas complexos, sendo um deles o cálculo de Pi. Nesse caso em específico, o algoritmo jogaria pontos aleatórios em um plano cartesiano com um círculo desenhado. Caso o ponto esteja entre o período [0, 1], significa que o mesmo está dentro do raio do círculo, ou seja, está no interior da circunferência. Caso o valor esteja fora do intervalo, significa que o ponto foi colocado fora da circunferência.



Após colocar todos os pontos aleatoriamente, é possível estimar o valor de Pi através da fórmula:

Pi = 4*(pontos dentro do circulo/número de pontos)

Quanto maior a quantidade de pontos alocados, maior a precisão dos números decimais de pi.

A aplicação do método monte carlo utilizando Python Parallel baseia-se na implementação de um serviço Mestre-Escravo, onde o mestre solicita a task enquanto os escravos (threads) executam de forma paralela. O escravo recebe como parâmetro o número do processo, a função que retorna o resultado e a quantidade total de pontos a serem calculados.

```
def threads(nprocesso, return_dict, qtdPontos):
    """worker function"""
    py = []
    pi = montepi(random.random(),qtdPontos);
    return_dict[nprocesso] = pi
```

A criação dos escravos ocorre na função main, e através de um laço for é definida a quantidade de trabalhadores criadas (8 no caso da imagem), passando como parâmetro de quantidade de pontos o valor de 1 milhão.

```
if __name__ == "__main__":
    manager = multiprocessing.Manager()
    return_dict = manager.dict() #resultado do worker
    jobs = [] #array dos resultados

    for i in range(8):
        p = multiprocessing.Process(target=threads, args=(i, return_dict, 1000000)) #parametros worker
        jobs.append(p)
        p.start()
```

```
#Gera um ponto X e Y entre(0,1)
def pontoxy():
    if (math.pow(random.random(),2) + math.pow(random.random(),2) ) < 1: return 1 # x*x + y*y.
    else: return 0

# Cada vez que estiver entre 0 e 1, os acertos sobem
def montepi(rand_seed, num_tests):
    random.seed(rand_seed)
    circlepts = 0

    for i in range(int(num_tests)):
        circlepts += pontoxy()

    return [circlepts , num_tests]
```

Conforme a imagem acima, a função pontoxy é responsável por definir de forma aleatória onde o ponto será alocado. Caso esteja dentro do círculo, será incrementado +1 na variável circlepts da função montepi, responsável por contabilizar os pontos dentro do círculo.

Por fim, ocorre a soma dos pontos obtidos pelos escravos e é aplicada a fórmula do Pi para realizar o cálculo aproximado.

```
for points in return_dict.values():
    circlepts += points[0]
    num_tests += points[1]

valor_pi = 4.0 * float(circlepts )/num_tests

print(("Valor estimado de Pi:", valor_pi))
```

```
rowzer@IGORFC:/mnt/c/Users/igorc/Desktop/Igor_ProgParalela$ python Pi_Monte_Carlo.py
[[785763, 1000000], [785411, 1000000], [785882, 1000000], [784788, 1000000], [785277, 1000000], [785503, 1000000], [7852
35, 1000000], [785947, 1000000]]
('Valor estimado de Pi:', 3.141903)
rowzer@IGORFC:/mnt/c/Users/igorc/Desktop/Igor_ProgParalela$ python Pi_Monte_Carlo.py
[[786137, 1000000], [784979, 1000000], [786168, 1000000], [785213, 1000000], [785780, 1000000], [785787, 1000000], [7856
17, 1000000], [784982, 1000000]]
('Valor estimado de Pi:', 3.1423315)
rowzer@IGORFC:/mnt/c/Users/igorc/Desktop/Igor_ProgParalela$ python Pi_Monte_Carlo.py
[[785528, 1000000], [784860, 1000000], [785470, 1000000], [785353, 1000000], [786019, 1000000], [785314, 1000000], [7858
97, 1000000], [784942, 1000000]]
('Valor estimado de Pi:', 3.1416915)
```