PROGRAMACION PARALELA

**METEDO DE MONTECARLO**

**PARA APROXIMA EL VALOR DE Π**

## INDICE

[1. INDICE 2](#_Toc88748272)

[2. INTRODUCCION 3](#_Toc88748273)

[3. METODO DE MONTECARLO 4](#_Toc88748274)

[3.1. INTRODUCCION AL METODO 4](#_Toc88748275)

[3.2. METODO DE MONTECARLO PARA APROXIMAR П 4](#_Toc88748276)

[3.3. ¿COMO SE OBTENDRAN LOS VALORES DE П? 5](#_Toc88748277)

[3.4. IMPLEMENTACION EN C 5](#_Toc88748278)

[4. CONCLUSIONES 8](#_Toc88748279)

[5. REFERENCIAS BIBLIOGRAFICAS 9](#_Toc88748280)

## INTRODUCCION

A lo largo de la historia, muchas personas sobre todo científicos han tenido un cierto interés en tratar de aproximar el valor del número П. Se sabe que este número es irracional de infinitas cifras decimales que se prolongan tras la coma y sin jamás repetir el mismo patrón, aunque nosotros en nuestra práctica cotidiana lo aproximemos a 3.14 . Tal interés o ambición ha llevado al desarrollo de nuevos conceptos matemáticos como los algoritmos iterativos y los límites.

En este trabajo se van a explicar y diseñar con un método para estimar el valor del número. El método que se utilizara y que nos ayudara a describir el proceso es el **Método de Montecarlo** utilizando programación en paralelo programado en el lenguaje de programación C importando librerías MPI; se van a ejecutar las simulaciones considerando diferentes tamaños o valores de entrada. Una vez obtenidos estos resultados se determinará que tan buenas son estas aproximaciones en función del tamaño de la muestra generada.

## METODO DE MONTECARLO

* 1. INTRODUCCION AL METODO

El método Monte Carlo es un método en el que por medio de la estadística y la probabilidad podemos determinar valores o soluciones de ecuaciones que calculados con exactitud son muy complejas, pero que mediante este método resulta sencillo calcular una aproximación al resultado que buscamos. El método Monte Carlo fue desarrollado en 1944 en Laboratorio Nacional de Los Álamos, como parte de los estudios que condujeron al desarrollo de la bomba atómica. En un principio lo desarrollaron los matemáticos John Von Neumann y Stanislaw Ulam aunque fueron otros matemáticos quienes con su trabajo le dieron una solidez científica, Harris y Herman Kahn. La idea le surgió a Ulam, mientras jugaba a las cartas. Se le ocurrió un método en el que mediante la generación de números aleatorios, pudieran determinar soluciones a ecuaciones complejas que se aplican en el estudio de los neutrones. Era como generar los números con la ayuda de una ruleta, de ahí su nombre.

* 1. METODO DE MONTECARLO PARA APROXIMAR П

La idea con este método será generar de forma aleatoria una serie de puntos (x, y) en un plano 2-D cuadrado de lado 1. Dentro de ese plano se insertará un círculo con el mismo diámetro, luego se calculará la proporción de los puntos numéricos que se encuentren dentro del círculo y el número total de puntos generados para que, a partir de esos datos, calcular el posible valor de П.

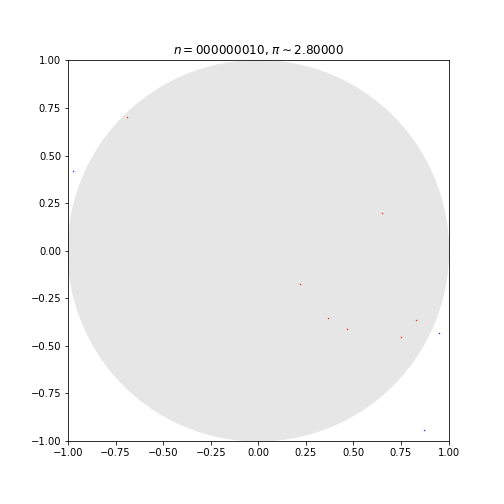


Ilustración.1

*En la animación se muestra la manera en la que cambia la estimación de pi a*

*medida que se varía el número de puntos generados.*

[Link de la pagina](https://programacionpython80889555.files.wordpress.com/2021/05/estimacion_de_pi_por_montercarlo.gif?w=621&zoom=2)

* 1. ¿COMO SE OBTENDRAN LOS VALORES DE П?

Para obtener el valor de П a partir de las estimaciones del área del círculo y del cuadrado, se comienza partiendo desde las superficies del área del círculo y del cuadrado:

Se tienes que:

A partir de esta ecuación se podrá estimar el valor de П, sabiendo que a partir de aquí la estimación del área del círculo vendrá dada por los puntos que se generen en su interior y el área del cuadrado por la totalidad de los puntos generados.

* **OBSERVACIONES:**
  + Si se inscribe un círculo en un cuadrado, y se tiran dardos aleatoriamente en el cuadrado, la probabilidad de que caigan dentro del círculo es П/4.
  + El método es muy paralelizable porque puede dividirse en pequeños procesos sin compartir datos, pero no es determinístico.
  + El Método de Montecarlo tiene una tasa de convergencia lenta ().

Con toda esta información recolectada se procederá a hacer la implementación.

* 1. IMPLEMENTACION EN C

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116 | #include <math.h>  #include <limits.h>  #include <mpi.h>  #include <stdio.h>  #include <stdlib.h>  /\* no. of random nos. to generate at one time \*/  #define CHUNKSIZE 1000  /\* message tags \*/  #define REQUEST 1  #define REPLY 2  **int** **main**(**int** argc, **char** \*\*argv) {  **int** iter;  **int** in, out, i, iters, max, ix, iy, ranks[**1**], done, temp;  **double** x, y, Pi, error, epsilon;  **int** numprocs, myid, server, totalin, totalout, workerid;  **int** rands[CHUNKSIZE], request;  MPI\_Comm world, workers;  MPI\_Group world\_group, worker\_group;  MPI\_Status stat;  MPI\_Init(&argc,&argv);  world = MPI\_COMM\_WORLD;  MPI\_Comm\_size(world, &numprocs);  MPI\_Comm\_rank(world, &myid);  server = numprocs-**1**;  **if** (numprocs==**1**)  printf("Error. At least 2 nodes are needed");  /\* process 0 reads epsilon from args and broadcasts it toeveryone \*/  **if** (myid == **0**) {  **if** (argc<**2**) {  epsilon = **1e-2**;  }  **else** {  sscanf ( argv[**1**], "%lf", &epsilon );  }  }  MPI\_Bcast ( &epsilon, **1**, MPI\_DOUBLE, **0**,MPI\_COMM\_WORLD );  /\* define the workers communicator by using groups and  excluding the server from the group of the whole world \*/  MPI\_Comm\_group ( world, &world\_group );  ranks[**0**] = server;  MPI\_Group\_excl ( world\_group, **1**, ranks, &worker\_group );  MPI\_Comm\_create ( world, worker\_group, &workers);  MPI\_Group\_free ( &worker\_group);  /\* the random number server code - receives a non-zero  request, generates random numbers into the array rands,  and passes them back to the process who sent the  request. \*/  **if** ( myid == server ) {  **do** {  MPI\_Recv(&request, **1**, MPI\_INT, MPI\_ANY\_SOURCE,  REQUEST, world, &stat);  **if** ( request ) {  **for** (i=**0**; i<CHUNKSIZE; i++) rands[i] = random();  MPI\_Send(rands, CHUNKSIZE, MPI\_INT,  stat.MPI\_SOURCE, REPLY, world);  }  } **while** ( request > **0** );  /\* the code for the worker processes - each one sends a  request for random numbers from the server, receives  and processes them, until done \*/  }  **else** {  request = **1**;  done = in = out = **0**;  max = INT\_MAX;  /\* send first request for random numbers \*/  MPI\_Send( &request, **1**, MPI\_INT, server, REQUEST, world );  /\* all workers get a rank within the worker group \*/  MPI\_Comm\_rank ( workers, &workerid );  iter = **0**;  **while** (!done) {  iter++;  request = **1**;  /\* receive the chunk of random numbers \*/  MPI\_Recv(rands, CHUNKSIZE, MPI\_INT, server,  REPLY, world, &stat );  **for** (i=**0**; i<CHUNKSIZE; ) {  x = (((**double**) rands[i++])/max) \* **2** - **1**;  y = (((**double**) rands[i++])/max) \* **2** - **1**;  **if** (x\*x + y\*y < **1.0**)  in++;  **else**  out++;  }  /\* the value of in is sent to the variable totalin in  all processes in the workers group \*/  MPI\_Allreduce(&in, &totalin, **1**,MPI\_INT, MPI\_SUM, workers);  MPI\_Allreduce(&out, &totalout, **1**,MPI\_INT, MPI\_SUM, workers);  Pi = (**4.0**\*totalin)/(totalin + totalout);  error = fabs ( Pi - M\_PI);  done = ((error < epsilon) || ((totalin+totalout)>**1000000**));  request = (done) ? **0** : **1**;  /\* if done, process 0 sends a request of 0 to stop the  rand server, otherwise, everyone requests more  random numbers. \*/  **if** (myid == **0**) {  printf("pi = %23.20lf**\n**", Pi );  MPI\_Send( &request, **1**, MPI\_INT, server, REQUEST, world);  }  **else** {  **if** (request)  MPI\_Send(&request, **1**, MPI\_INT, server, REQUEST, world);  }  }  }  **if** (myid == **0**)  printf("total %d, in %d, out %d**\n**",  totalin+totalout, totalin, totalout );  **if** (myid<server) MPI\_Comm\_free(&workers);  MPI\_Finalize();  } |

## CONCLUSIONES

* Mientras mayor sea el numero de procesos, mas cercano es el valor de П.
* Por lo anterior, también aumentan la cantidad de puntos dentro y fuera de la gráfica.
* Al aumentar el numero de procesos, el programa toma más tiempo en ejecución por lo cual lo hace menos eficiente. Esto teniendo en cuanta la arquitectura o entorno con la que se esta trabajando.

## REFERENCIAS BIBLIOGRAFICAS

* *GeoGebra:* <https://www.geogebra.org/m/cF7RwK3H>
* *El comercio:* <https://elcomercio.pe/respuestas/que/numero-pi-significado-origen-decimales-nnda-noticia-546986-noticia/?ref=ecr>
* <https://ucema.edu.ar/publicaciones/download/documentos/591.pdf>
* *Wikipedia:*
  + <https://es.wikipedia.org/wiki/M%C3%A9todo_de_Montecarlo>
  + [https://es.wikipedia.org/wiki/N%C3%BAmero\_%CF%80#Historia\_del\_c%C3%A1lculo\_del\_valor\_%CF%80](https://es.wikipedia.org/wiki/N%C3%BAmero_%CF%80%23Historia_del_c%C3%A1lculo_del_valor_%CF%80)
* *Programmer Clicki:* <https://programmerclick.com/article/56321270780/>