Problema 1. Sobre el algoritmo Yates-Fisher.

2.(a) Explique qué hace el algoritmo de Yates-Fisher. Implemente el algoritmo para ocho elementos e imprima el ordenamiento de los elementos a cada paso del algoritmo.

Solución.

El algoritmo de Yates-Fisher es un algoritmo que se encarga de realizar permutaciones de en un arreglo finito con el propósito de 'desordenarlo de manera aleatoria'. El funcionamiento del algoritmo en su versión original es el siguiente:

1. Recibimos un arreglo A, digamos, con 8 entradas. $A = [A_1, A_2, A_3, ..., A_8]$.
2. Generamos un arreglo vacío D que contendrá los elementos 'desordenados'.
3. Generamos un número entero aleatorio s entre 1 y 8. Tomamos el elemento en la posición s en A, en este caso es A_s, lo removemos de A y lo insertamos en D. Así, en este punto, si por ejemplo $s = 3$, tenemos que: $A = [A_1, A_2, A_4, ..., A_8]$ y $D = [A_3]$.
4. Repetimos el procedimiento del paso 3, sólo que ahora s estará entre 1 y 7, pues tenemos un elemento menos en A. Y si por ejemplo $s = 4$, entonces ahora, $A = [A_1, A_2, A_4, A_6, A_7, A_8]$ y $D = [A_3, A_5]$.

Paso	Rango de elección	Posición aleatoria	Conjunto	Desorden
			Input	
1	1-8	3	$A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8$	A_3
2	1-7	6	$A_1, A_2, A_4, A_5, A_6, A_7, A_8$	A_3, A_7
3	1-6	5	$A_1, A_2, A_4, A_5, A_6, A_8$	A_3, A_7, A_6
4	1-5	3	A_1, A_2, A_4, A_5, A_8	A_3, A_7, A_6, A_4
5	1-4	4	A_1, A_2, A_5, A_8	A_3, A_7, A_6, A_4, A_8
6	1-3	1	A_1, A_2, A_5	$A_3, A_7, A_6, A_4, A_8, A_1$
7	1-2	2	A_2, A_5	$A_3, A_7, A_6, A_4, A_8, A_1, A_2$
				Output
8	1	1	A_5	$A_3, A_7, A_6, A_4, A_8, A_1, A_2, A_5$

Figure 1: Algoritmo Yates-Fisher graficamente

5. Repetimos el procedimiento, reduciendo cada vez el rango de elección del número aleatorio, hasta que todos los elementos de A estén ahora en D, i.e. , hasta que A se encuentre vacía.

Listing 1: Código Simulación Variables Aleatorias

```
import random

def randent(k):
        #Genera un número aleatorio en el rango [0,k-1]
        return int(random.random()*k)

def YaFi(A):
        print("Arreglo inicial: "+str(A)+"\n")
        a=len(A) #Lonitud inicial del arreglo
        D=[] #Aquí iremos construyendo nuestro arreglo desordenado
        for j in range(a):
                indice=randent(len(A))
                #Seleccionamos posición aleatoria
                D.append(A[indice])
                #Agregamos el elemento en esa posición a D
                A.pop(indice)
                #Lo quitamos de A
                print("Arreglo original al paso "+str(j+1)+": "
                +str(A))
                print("Arreglo desordenado al paso "+str(j+1)+":
                "+str(D)+"\n")
        print("Arreglo desordenado: "+str(D))

YaFi([2,1,4,102,32,-1,5,10])
```

```
Arreglo inicial: [2, 1, 4, 102, 32, -1, 5, 10]

Arreglo original al paso 1: [2, 1, 4, 32, -1, 5, 10]
Arreglo desordenado al paso 1: [102]

Arreglo original al paso 2: [2, 4, 32, -1, 5, 10]
Arreglo desordenado al paso 2: [102, 1]

Arreglo original al paso 3: [2, 4, 32, -1, 10]
Arreglo desordenado al paso 3: [102, 1, 5]

Arreglo original al paso 4: [2, 4, -1, 10]
Arreglo desordenado al paso 4: [102, 1, 5, 32]

Arreglo original al paso 5: [2, 4, 10]
Arreglo desordenado al paso 5: [102, 1, 5, 32, -1]

Arreglo original al paso 6: [2, 10]
Arreglo desordenado al paso 6: [102, 1, 5, 32, -1, 4]

Arreglo original al paso 7: [10]
Arreglo desordenado al paso 7: [102, 1, 5, 32, -1, 4, 2]

Arreglo original al paso 8: []
Arreglo desordenado al paso 8: [102, 1, 5, 32, -1, 4, 2, 10]

Arreglo desordenado: [102, 1, 5, 32, -1, 4, 2, 10]
```

Figure 2: Ejecución del algoritmo

```
Arreglo inicial: [2, 1, 4, 102, 32, -1, 5, 10]

Arreglo original al paso 1: [2, 1, 102, 32, -1, 5, 10]
Arreglo desordenado al paso 1: [4]

Arreglo original al paso 2: [1, 102, 32, -1, 5, 10]
Arreglo desordenado al paso 2: [4, 2]

Arreglo original al paso 3: [1, 102, -1, 5, 10]
Arreglo desordenado al paso 3: [4, 2, 32]

Arreglo original al paso 4: [1, 102, -1, 5]
Arreglo desordenado al paso 4: [4, 2, 32, 10]

Arreglo original al paso 5: [1, -1, 5]
Arreglo desordenado al paso 5: [4, 2, 32, 10, 102]

Arreglo original al paso 6: [-1, 5]
Arreglo desordenado al paso 6: [4, 2, 32, 10, 102, 1]

Arreglo original al paso 7: [5]
Arreglo desordenado al paso 7: [4, 2, 32, 10, 102, 1, -1]

Arreglo original al paso 8: []
Arreglo desordenado al paso 8: [4, 2, 32, 10, 102, 1, -1, 5]

Arreglo desordenado: [4, 2, 32, 10, 102, 1, -1, 5]
```

Figure 3: Ejecución del algoritmo

Existe una versión optimizada del algoritmo que consiste en ir cambiando las posiciones del arreglo recibido, en lugar de ir borrando los elementos del arreglo e introducirlos en uno nuevo. La dinámica de esta variante es la siguiente:

1. Recibimos un arreglo A, digamos, con 8 entradas. $A = [A_1, A_2, A_3, ..., A_8]$.
2. Generamos un número entero aleatorio s entre 1 y 8. Tomamos los elementos en la posiciones s en la última de A, y los intercambiamos de lugar. Así, en este punto, si por ejemplo $s = 3$, tenemos que: $A = [A_1, A_2, A_8, ..., A_3]$. De esta manera ya habremos 'desarreglado la última posición.
3. Ahora, generamos un número aleatorio s entre 1 y 7, pues la última posición ya está desordenada y permanecerá de esa manera hasta el final del proceso. Tomamos el elemento en la posición 7 y lo intercambiamos con el elemento en la posición s. Si $s = 5$, tendríamos en este paso entonces que: $A = [A_1, A_2, A_8, A_4, A_7, A_6, A_5, A_3]$.
4. Repetimos el proceso hasta que hayamos desordenado a todos los elementos, es decir, hasta que llegemos al primer elemento del arreglo.

Paso	Rango de elección	Posición aleatoria	Conjunto
			Input
0	-	-	$A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8$
1	1-8	3	$A_1, A_2, A_8, A_4, A_5, A_6, A_7, A_3$
2	1-7	6	$A_1, A_2, A_8, A_4, A_5, A_7, A_6, A_3$
3	1-6	5	$A_1, A_2, A_8, A_4, A_7, A_5, A_6, A_3$
4	1-5	3	$A_1, A_2, A_7, A_4, A_8, A_5, A_6, A_3$
5	1-4	4	$A_1, A_2, A_7, A_4, A_8, A_5, A_6, A_3$
6	1-3	1	$A_7, A_2, A_1, A_4, A_8, A_5, A_6, A_3$
7	1-2	2	$A_7, A_2, A_1, A_4, A_8, A_5, A_6, A_3$
			Output
8	1	1	$A_7, A_2, A_1, A_4, A_8, A_5, A_6, A_3$

Figure 4: Algoritmo Yates-Fisher optimizado, graficamente

Los que se quedan fijos porque ya fueron desordenados
Los que estaban en la última posición y fueron intercambiados
Los que estaban en la posición aleatoria y fueron intercambiados

Listing 2: Código Simulación Variables Aleatorias

```
def YaFiO(A):
        print("Arreglo inicial: "+str(A)+"\n")
        a=len(A)
        for j in range(a-1,0,-1):
                indice=randent(j)
                aux=A[j]
                A[j]=A[indice]
                A[indice]=aux
                print("Arreglo al paso "+str(a-j)+": "+str(A)+"\n")
        print("Arreglo desordenado: "+str(A))

YaFiO([2,1,4,102,32,-1,5,10])
```

```
Arreglo inicial: [2, 1, 4, 102, 32, -1, 5, 10]
Arreglo al paso 1: [2, 1, 4, 10, 32, -1, 5, 102]
Arreglo al paso 2: [2, 1, 4, 10, 32, 5, -1, 102]
Arreglo al paso 3: [2, 1, 5, 10, 32, 4, -1, 102]
Arreglo al paso 4: [2, 1, 32, 10, 5, 4, -1, 102]
Arreglo al paso 5: [2, 1, 10, 32, 5, 4, -1, 102]
Arreglo al paso 6: [2, 10, 1, 32, 5, 4, -1, 102]
Arreglo al paso 7: [10, 2, 1, 32, 5, 4, -1, 102]
Arreglo desordenado: [10, 2, 1, 32, 5, 4, -1, 102]
```

Figure 5: Ejecución del algoritmo

```
Arreglo inicial: [2, 1, 4, 102, 32, -1, 5, 10]
Arreglo al paso 1: [2, 1, 10, 102, 32, -1, 5, 4]
Arreglo al paso 2: [5, 1, 10, 102, 32, -1, 2, 4]
Arreglo al paso 3: [-1, 1, 10, 102, 32, 5, 2, 4]
Arreglo al paso 4: [-1, 32, 10, 102, 1, 5, 2, 4]
Arreglo al paso 5: [102, 32, 10, -1, 1, 5, 2, 4]
Arreglo al paso 6: [10, 32, 102, -1, 1, 5, 2, 4]
Arreglo al paso 7: [32, 10, 102, -1, 1, 5, 2, 4]
Arreglo desordenado: [32, 10, 102, -1, 1, 5, 2, 4]
```

Figure 6: Ejecución del algoritmo

Problema 2. Variables aleatorias.

Realizar programas para simular las siguientes variables aleatorias:

2.(a) Simular un experimento con k resultados posibles, de modo que el resulta i ocurra con probabilidad pi. Asuma que el usuario le da estas probabilidades en el vector $P = (p1, p2, ..., pk)$.

Solución.

Listing 3: Código Simulación Variables Aleatorias

```
1  import random
2  import numpy as np
3
4  def experimento(P):
5          #P es el vector que contiene las probabilidades asociadas
6          a cada resultado.
7          resultado=random.random()
8
9          s=0
10         for i in range(len(P)):
11                 s+=P[i]
12                 if(resultado<=s):
13                         print("Salió el resultado "+str(i+1))
14                         break
15
16         return i
```

Consideraremos que los k resultados son ajenos por pares, y que son todos los posibles resultados del experimento aleatorio.

Para simular el problema, definimos la función experimento. Esta función, recibe como parámetro a un vector P, cuyas entradas son las probabilidades asociadas a cada uno de los k posibles resultados del experimento, i.e., $P = [p1, p2, ..., pk]$, donde p_i es la probabilidad asociada al resultado i.

Consideremos el intervalo $[0, 1]$. La idea es dividir el intervalo $[0, 1]$ en tantos subintervalos como resultados haya, la división no será uniforme, si no que la longitud de cada subintervalo i va a ser igual a p_i y se acomodarán uno tras otro, comenzando por el subintervalo asociado a p_1, luego p_2 y así sucesivamente, hasta finalizar con p_k. Esto estará bien definido, pues, dado que estamos considerando todos los posibles resultados y estamos suponiendo como eventos ajenos a los resultados, $\sum_{i=1}^{k} p_i = 1 = long([0, 1])$.

Generaremos entonces una variable aleatoria uniforme, (con ayuda de la función $random.random()$) en el intervalo [0,1]. Este proceso simulará la realización del experimento y el numero arrojado será el resultado del mismo. A continuación, se evaluará a qué subintervalo pertenece el número, y diremos que el resultado del experimento fue el resultado asociado al subintervalo en que cayó el número.

Ejemplificaremos la resolución del problema con un esquema cuando $P = [.15, .50, .30, .05]$.

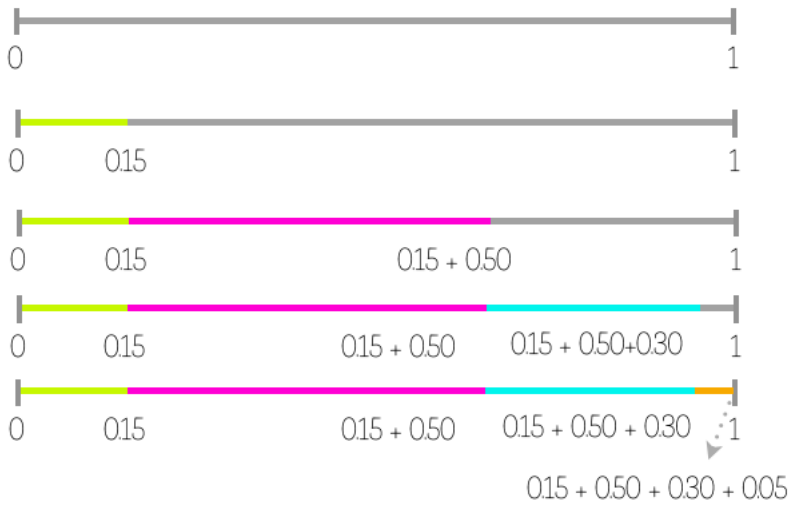


Figure 7: División del intervalo $[0, 1]$

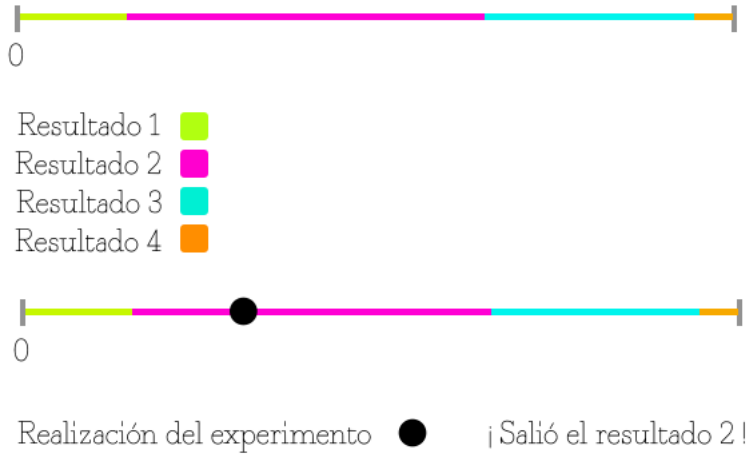


Figure 8: Simulación experimento

Listing 4: Código para verificar la efectividad de la simulación

```
def conteo(P,n):
        conteo=np.zeros(len(P))
        for i in range(n):
                a=experimento(P)
                for j in range(len(P)):
                        if a==j:
                                conteo[j]+=1
        print(conteo)
```

Para verificar la simulación, definí la función conteo, que realiza la simulación del experimento n veces y cuenta cuántas veces ocurrió cada uno de los resultados. Se puede observar con esta función que entre más grande es el valor de n, el número de veces que ocurrió el resultado i puede ser aproximado por el valor $p_i \cdot n$.

Listing 5: Ejemplificación del programa

```
Input
P=[.15,.50,.30,.05]
conteo(P,100000)

Output
[15114. 50014. 30006.  4866.]
#Así, salieron 15114 veces el resultado 1.
```

Problema 3. Sobre el método Monte Carlo.

Estimar el área de un círculo de diámetro unitario utilizando el método Monte Carlo.

2.(a) Genere n puntos aleatorios en el cuadrado unitario $(x_i, y_i) \in [0,1]^2$, con $i \leq n$

Solución.

Listing 6: Generación de puntos aleatorios en el cuadrado unitario.

```
import random
def generapunto():
        xi=random.random()
        yi=random.random()
        return xi,yi
```

La función *generapunto()* regresa una pareja de números aleatorios en el cuadrado unitario, ya que ambas coordenadas se encuentran en el intervalo [0,1]. Los números aleatorios son generados con la función *random.random()*. No genero n puntos en la misma función, porque me parece más fácil generarlos uno por uno y no trabajar después con arreglos de pares ordenados.

2.(b) Haga una gráfica con la estimación dada por los promedios parciales.

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_c(x_i, y_i)$$

Solución.
Primero, definamos la función indicadora de la circunferencia dada. Primero caractericemos la circunferencia. Notemos que se trata de una circunferencia de diámetro uno en el cuadrado unitario, así se puede fácilmente deducir que se trata de la circunferencia con centro en el $C = (0.5, 0.5)$ y radio $r = 0.5$. Así la ecuación de la circunferencia puede escribirse como:

$$\mathscr{C} := (x - 0.5)^2 + (y - 0.5)^2 = (0.5)^2 = 0.25$$

Así, sólo nos interesan aquellos puntos x_i, y_i tales que se encuentren en la zona verde brillante de la figura 9, es decir, aquellos que satisfagan la siguiente condición:

$$\mathscr{C} := (x - 0.5)^2 + (y - 0.5)^2 = (0.5)^2 < 0.25 \tag{1}$$

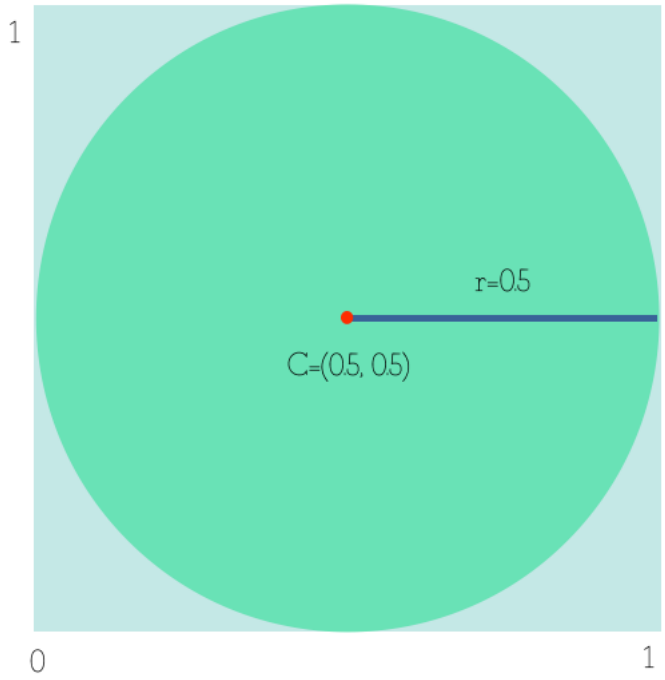


Figure 9: Circunferencia de diámetro 1 en el cuadrado unitario.

Siendo así, definiremos la función indicadora de la siguiente manera:

Listing 7: Función indicadora.

```
1 def Ind(P):
2         i=0
3         if(((P[0]-.5)**2+(P[1]-.5)**2)<=.25):
4                 i=1
5         return i
```

Donde P es un punto, de la forma $P = [x, y]$. Y debemos verificar que P cumpla la ecuación (1). En caso de cumplirla, la función indicadora regresará 1, si no, 0. Para calcular los promedios parciales con n puntos, realizamos n iteraciones, en cada una de las cuales generaremos un punto aleatorio en el cuadrado unitario, le aplicamos la función indicadora y sumamos el resultado de la misma a la variable s, que será inicializada en cero. Regresaremos $\frac{s}{n}$

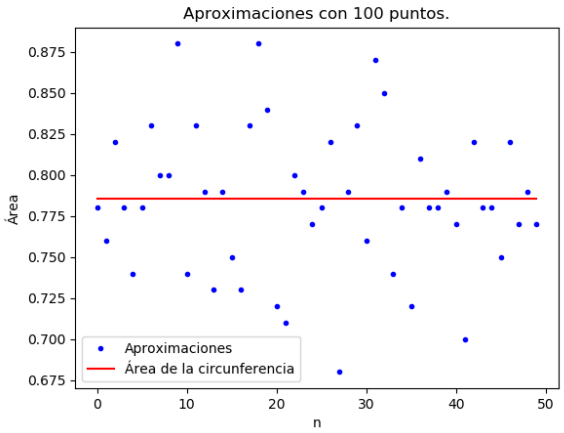
Listing 8: Sumas parciales.

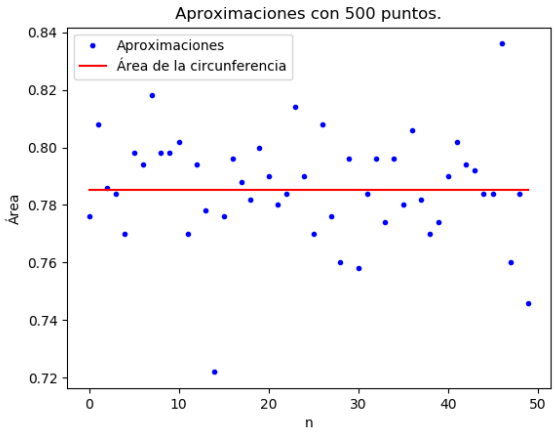
```
def promedioparcial(n):
        s=0
        for i in range(n):
                P=generapunto()
                s+=Ind(P)
        r=(s/n)
        return r
```

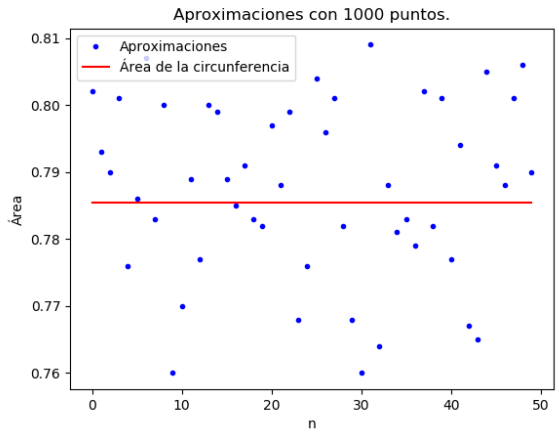
A continuación, se muestra una serie de gráficas estructuradas de la siguiente manera. La altura a la que se encuentra la recta horizontal roja simboliza el área real del círculo con el que hemos estado trabajando:

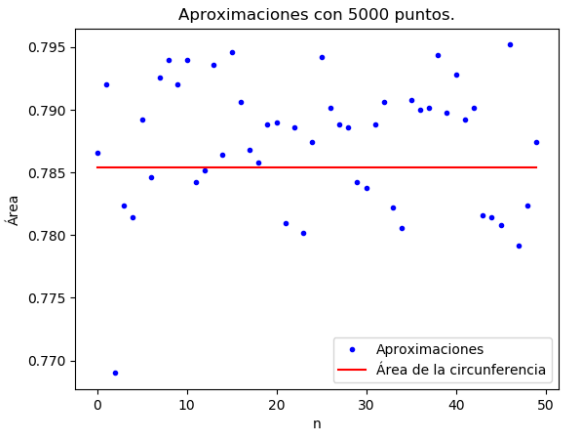
$$\acute{A}rea(\mathscr{C}) = \pi(0.5)^2 \approx 0.78539816339$$

Se realizó en cada gráfica 50 veces el mismo experimento, en cada punto azul (n, y), la coordenada y (altura de cada punto) representa el resultado de la aproximación del área del círculo obtenida mediante el método Monte Carlo en el intento y. Cada gráfica expone el resultado del mismo experimento, es decir, con la misma cantidad de puntos aleatorios generados.

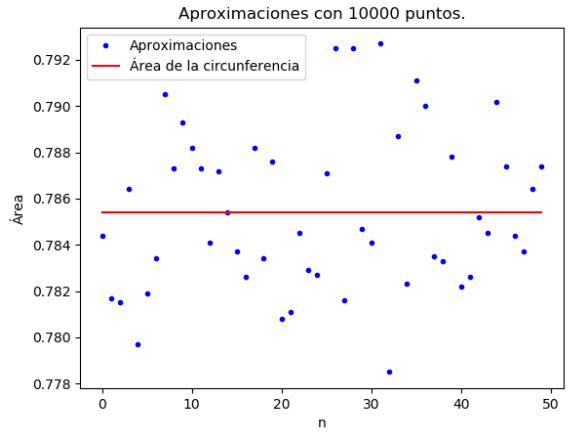


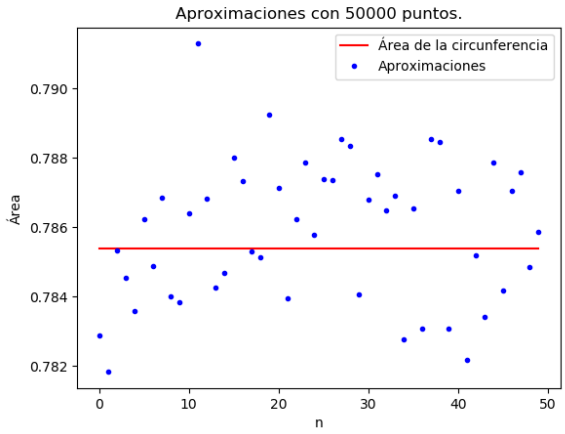


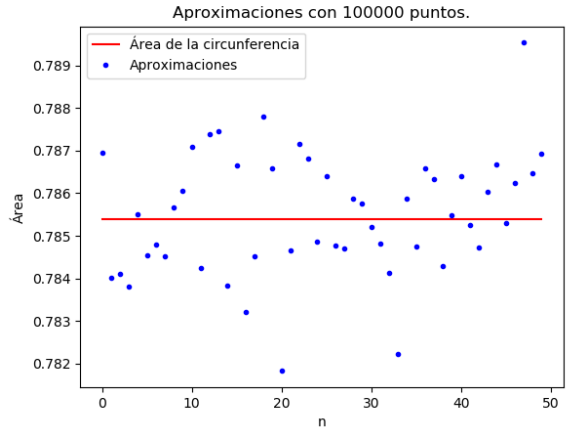


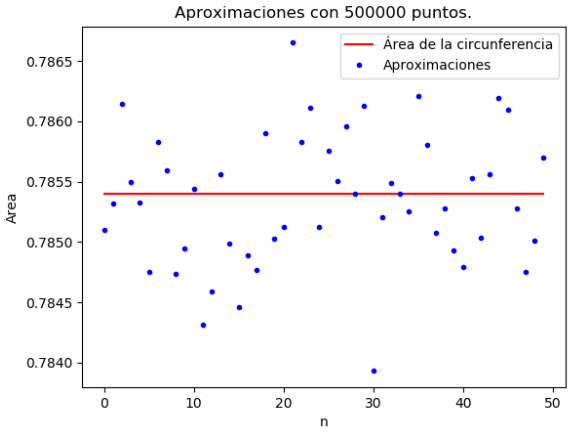


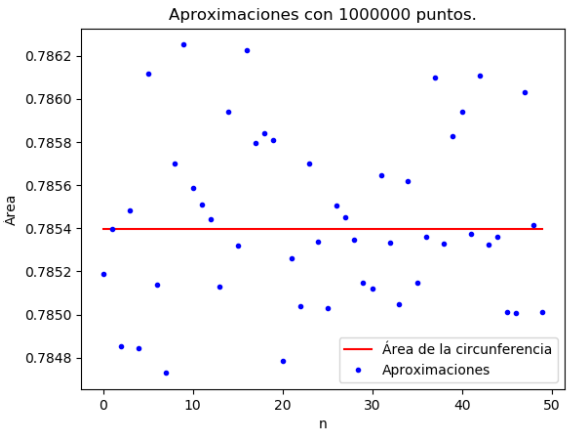
De las gráficas anteriores se puede concluir que a medida que mayor es el número de puntos aleatorios generados, la aproximación al área del círculo es, en promedio, cada vez mejor.





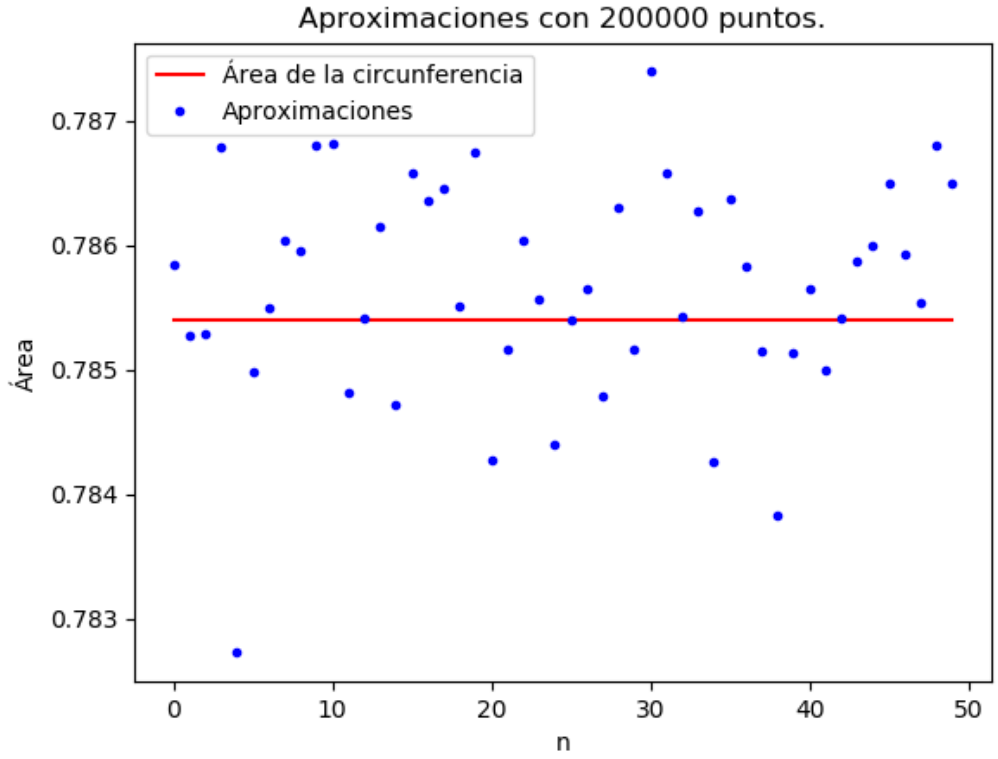


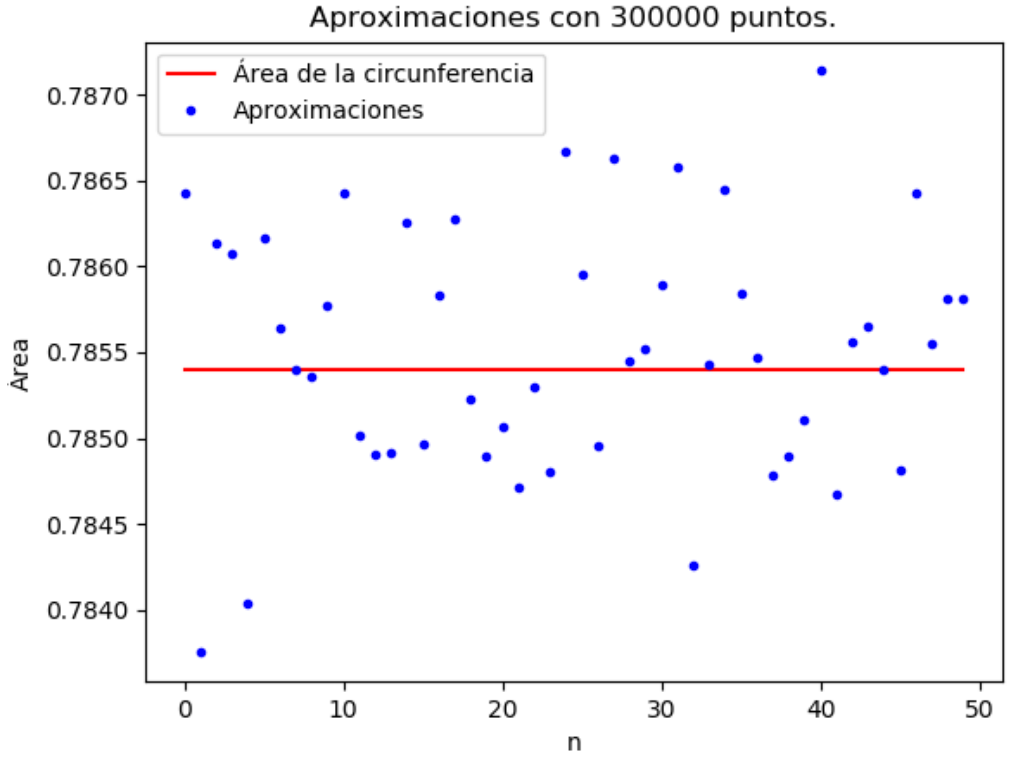




2.(c) ¿Cuántos puntos se necesitaron para tener una aproximación a 2 decimales?

De las gráficas anteriores podemos intuir que la aproximación a dos decimales se logra al rededor de los **100,000** puntos. Para tener mayor certeza probamos con 200,000 y 300,000 puntos. En ambos casos la aproximación fue a dos decimales.


Aproximaciones con 200000 puntos.
Área de la circunferencia
Aproximaciones
0.787
0.786
0.785
0.784
0.783
Área
0
10
20
30
40
50
n


Aproximaciones con 300000 puntos.
Área de la circunferencia
Aproximaciones
0.7870
0.7865
0.7860
0.7855
0.7850
0.7845
0.7840
Área
0
10
20
30
40
50
n