Análisis de algoritmos
Tarea 7
Calculo de π usando el método de Montecarlo

1) Generar los puntos aleatorios usando congruencia lineal.
2) Generar los puntos usando secuencia de Halton.
3) Generar los puntos usando el generador de puntos de su lenguaje de programación.

   Para cada caso graficar las curvas de convergencia.

   Analizar y determinar que método fue mas preciso.

Código implementado en python:

```
import random
import matplotlib.pyplot as plt
import numpy as np
import math

k=100000

#random congruencia lineal
def random1( x0, a, b, m):
        aux=(a*x0 + b) % m;
        return aux

#random Secuencia de Halton
def next_prime():
        def is_prime(num):
                "Checks if num is a prime value"
                for i in range(2,int(num**0.5)+1):
                        if(num % i)==0: return False
                return True

        prime = 3
        while(1):
                if is_prime(prime):
                        yield prime
                prime += 2
```

```
def vdc(n, base=2):
        vdc, denom = 0, 1
        while n:
                denom *= base
                n, remainder = divmod(n, base)
                vdc += remainder/float(denom)
        return vdc

def halton_sequence(size, dim):
        seq = []
        primeGen = next_prime()
        next(primeGen)
        for d in range(dim):
                base = next(primeGen)
                seq.append([vdc(i, base) for i in range(size)])
        return seq

#random libreria
def random3():
        x=np.random.uniform(-1,1)
        y=np.random.uniform(-1,1)
        rad=math.sqrt((x**2)+(y**2))

        return x,y,rad

#Congruencia lineal
cont=0
xs=np.array([])
ys=np.array([])
xn=np.array([])
yn=np.array([])
vpi=np.array([])
pi=0.1

for i in range(1,k):
        #x0>=0, a>=0, b>=0, m>=x0 and m>=a
        a=214013
        b=2531011
        m=2**32
        x=(random1(x0,a,b,m))*random.choice((-1,1))
        x0=x
        y=(random1(x0,a,b,m))*random.choice((-1,1))
        rad=math.sqrt(((x/(m-1))**2)+((y/(m-1))**2))
```

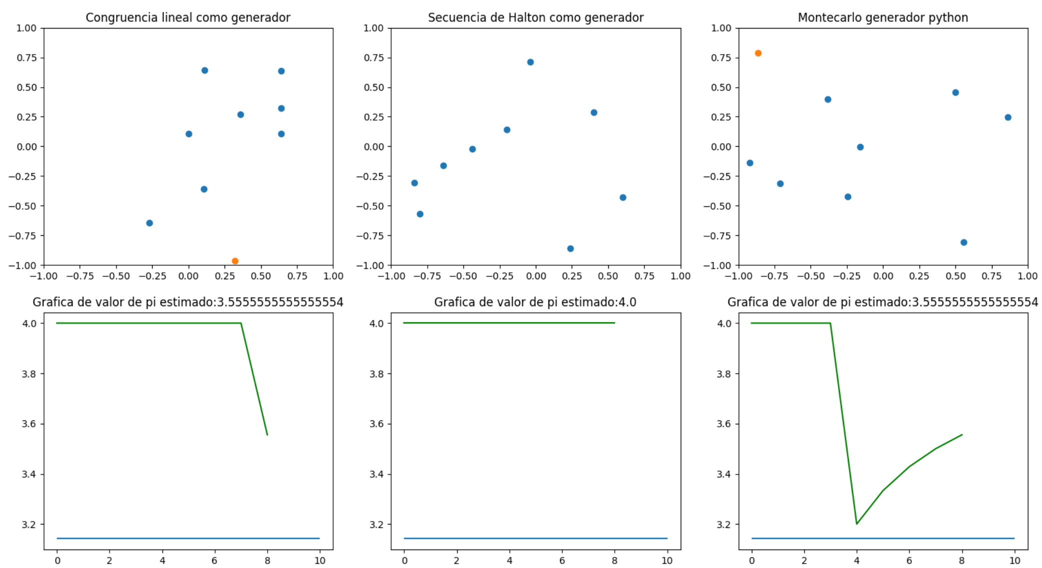
```
        if rad<=1:
                cont+=1
                xs=np.append(xs,(x/(m-1)))
                ys=np.append(ys,(y/(m-1)))
        else:
                xn=np.append(xn,(x/(m-1)))
                yn=np.append(yn,(y/(m-1)))
        pi=4*cont/i
        vpi=np.append(vpi,pi)
        x0=x

plt.subplot(2,3,1)
plt.title("Secuencia de Halton como generador")
plt.scatter(xs,ys)
plt.scatter(xn,yn)
plt.xlim(-1,1)
plt.ylim(-1,1)

plt.subplot(2,3,4)
tit="Grafica de valor de pi estimado:"+str(pi)
plt.title(tit)
plt.plot(vpi,'g')
plt.hlines(y=math.pi,xmin=0,xmax=k)

#Halton sequence
cont=0
xs=np.array([])
ys=np.array([])
xn=np.array([])
yn=np.array([])
vpi=np.array([])
pi=0.1
seq=np.array(halton_sequence(k,2))
for a in range(2):
        for j in range(k):
                seq[a][j]=seq[a][j]*random.choice((-1,1))
for i in range(1,k):
        x=seq[0][i]
        y=seq[1][i]
        rad=math.sqrt((x**2)+(y**2))
        if rad<=1:
                cont+=1
                xs=np.append(xs,x)
```

```
			ys=np.append(ys,y)
		else:
			xn=np.append(xn,x)
			yn=np.append(yn,y)
		pi=4*cont/i
		vpi=np.append(vpi,pi)

plt.subplot(2,3,2)
plt.title("Secuencia de Halton como generador")
plt.scatter(xs,ys)
plt.scatter(xn,yn)
plt.xlim(-1,1)
plt.ylim(-1,1)

plt.subplot(2,3,5)
tit="Grafica de valor de pi estimado:"+str(pi)
plt.title(tit)
plt.plot(vpi,'g')
plt.hlines(y=math.pi,xmin=0,xmax=k)

#libreria python
cont=0
xs=np.array([])
ys=np.array([])
xn=np.array([])
yn=np.array([])
vpi=np.array([])
pi=0.1

for i in range(1,k):
		x,y,rad=random3()
		if rad<=1:
			cont+=1
			xs=np.append(xs,x)
			ys=np.append(ys,y)
		else:
			xn=np.append(xn,x)
			yn=np.append(yn,y)
		pi=4*cont/i
		vpi=np.append(vpi,pi)

plt.subplot(2,3,3)
plt.title("Montecarlo generador python")
```

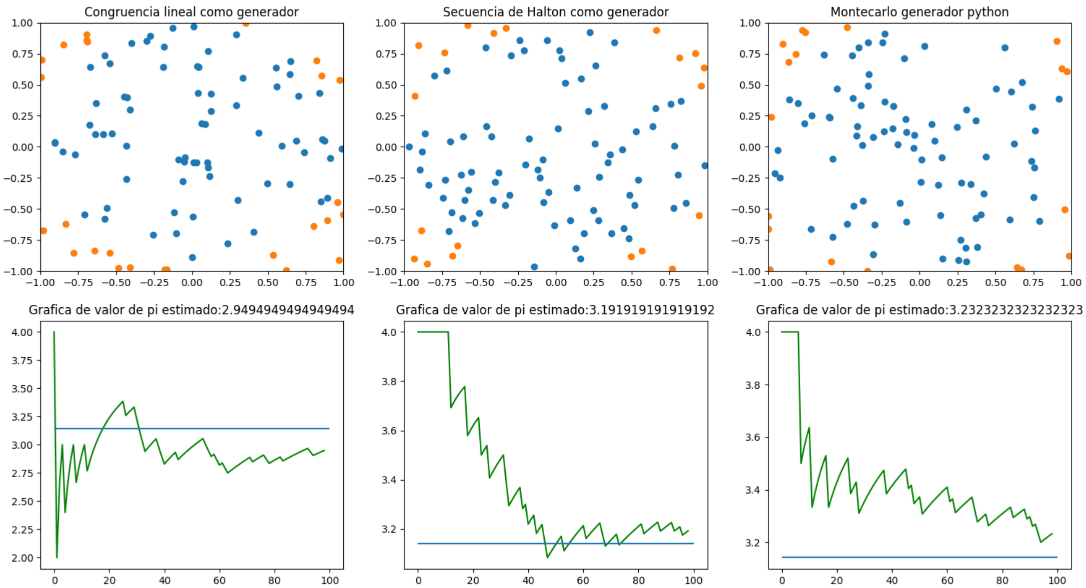
```
plt.scatter(xs,ys)
plt.scatter(xn,yn)
plt.xlim(-1,1)
plt.ylim(-1,1)

plt.subplot(2,3,6)
tit="Grafica de valor de pi estimado:"+str(pi)
plt.title(tit)
plt.plot(vpi,'g')
plt.hlines(y=math.pi,xmin=0,xmax=k)

plt.show()
```

Realizando pruebas, se modifico el k con diferentes valores para ver que resultados se obtenían en cada caso, las graficas obtenidas son las siguientes:

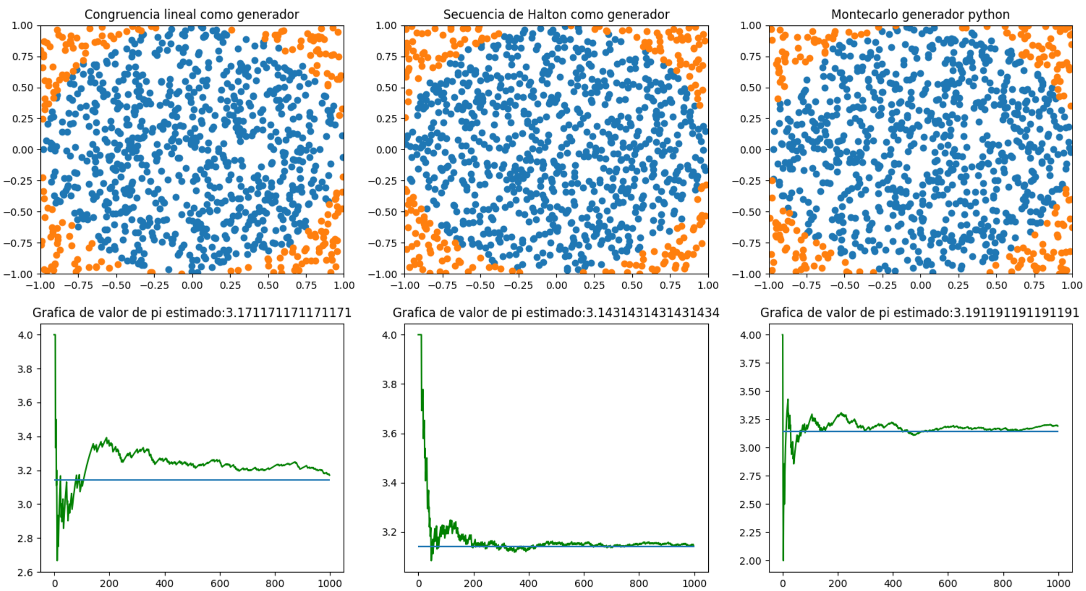
K=10



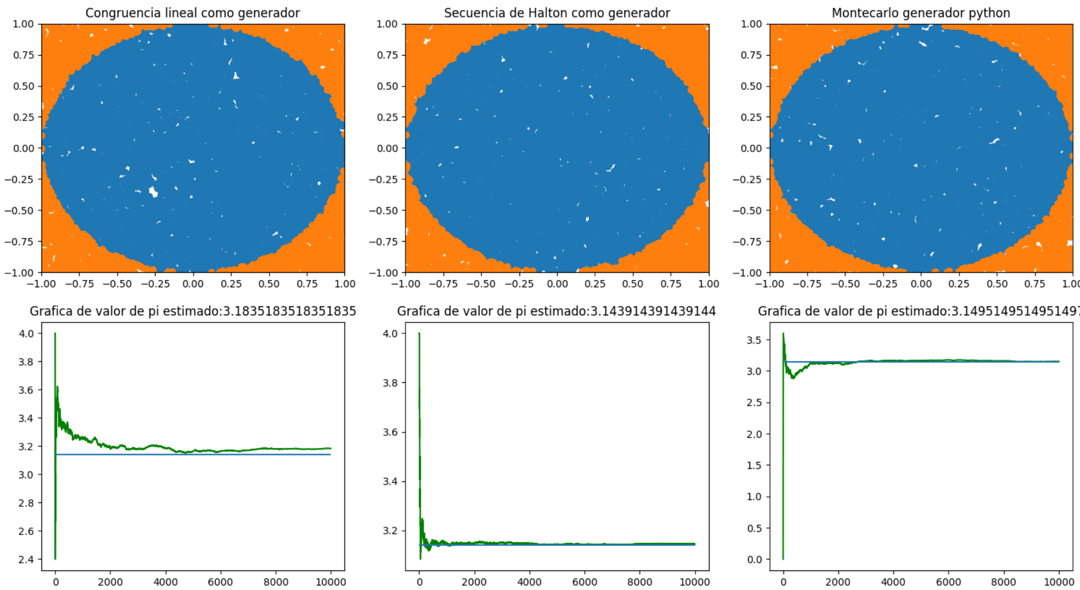
K=100


Congruencia lineal como generador
Secuencia de Halton como generador
Montecarlo generador python
Grafica de valor de pi estimado:2.9494949494949494
Grafica de valor de pi estimado:3.191919191919192
Grafica de valor de pi estimado:3.2323232323232323

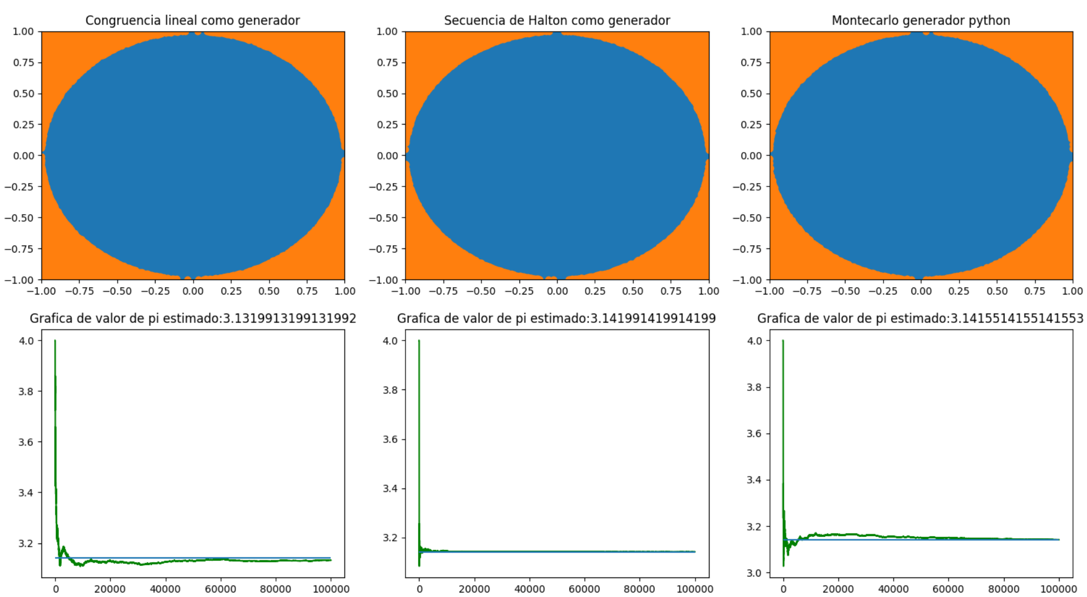
K=1000


Congruencia lineal como generador
Secuencia de Halton como generador
Montecarlo generador python
Grafica de valor de pi estimado:3.171171171171171
Grafica de valor de pi estimado:3.1431431431431434
Grafica de valor de pi estimado:3.191191191191191

K=10000



K=100000



Como se puede ver con k=10, el valor obtenido con los tres métodos es el mas alejado de π, mientras que cada que va aumentando el valor de K va aumentando la precisión de los métodos, sin embargo, el de la congruencia lineal, aunque se utilizaron los valores dados en wikipedia para el generador de números aleatorios de C++. En el caso del generador de la secuencia de Halton se utilizo una base de dos para generar los puntos en ambas dimensiones, y se puede apreciar que el valor de π calculado va convergiendo alrededor del valor de π dado por la librería math de python, que es justo lo esperado, siendo el mejor método ya que su valor es mas cercano a π y se puede apreciar mejor en el caso de K=100000 ya que llega un momento donde la linea ya es muy semejante a la linea horizontal del valor de π, mientras que el generador de python también se estabiliza alrededor del valor de π, sin embargo comparado con la secuencia de Halton tarda mas iteraciones en estabilizarse. En general los tres métodos logran acercarse al valor real de π con buena precisión, solamente cambiando el numero de iteraciones necesarias para cada uno de los métodos para poder reducir su error.