```python
"""
Biblioteca de métodos Montecarlo
(c) Carlos M. Martinez, marzo-abril 2022
carlos@cagnazzo.uy
"""

import random
import math
import tabulate
import time
from scipy.stats import norm
import functools
from pathos.multiprocessing import ProcessPool as Pool

random.seed()

_VERSION = "Volúmenes en R^N MMC v0.1.1 - Carlos Martinez marzo 2022"

def version():
    return _VERSION
# end def

def sortearPuntoRN(dim, randfun):
    """
    Seortea un punto en R^N dentro del hiper-cubo [0,1]^N
    randfun es una funcion con la misma API que random.uniform
    """
    punto = []
    for n in range(0, dim):
        # punto.append(random.uniform(0.0, 1.0))
        punto.append(randfun(0.0, 1.0))
    # end for

    return punto
# end fun sortearPuntoRN

# Implemento pseudocodigo Montecarlo

#@functools.lru_cache(maxsize=128)
def MetodoMonteCarlo(N, FVolumen, randfun = random.uniform):
    """
    Implementa el pseudocodigo de MC
    N: cantidad de muestras
    FVolumen: funcion que define el volumen, devuelve 0 si el punto esta fuera, 1 si esta dentro
    """
    random.seed()
    t0 = time.perf_counter()
    S = 0
    for j in range(0, N):
        punto = sortearPuntoRN(6, randfun)
        if FVolumen(punto):
            phi = 1
        else:
            phi = 0
        S = S + phi
    # end for
```

```
57      VolR = S / N
58      VarVorR = (S/N)*(1-S/N)/(N-1)
59      return (VolR, VarVorR, S,  time.perf_counter()-t0)
60  # end def
61
62  # Version paralelizada de Montecarlo
63  def MetodoMonteCarloParalelo(N, hilos, FVolumen, randfun=random.uniform):
64      """
65          version paralelizada del montecarlo
66          N: numero de muestras
67          FVolumen: funcion que implementa el volumen
68          randfun: funcion para generar numeros aleatorios con la misma firma que
    random.uniform()
69          hilos: cantidad de hilos en el pool de tareas
70      """
71      t0 = time.perf_counter()
72
73      args1 = []
74      args2 = []
75      args3 = []
76      for x in range(0,hilos):
77          args1.append( math.ceil(N/hilos) )
78          args2.append(FVolumen)
79          args3.append(randfun)
80
81      p = Pool(hilos)
82      resultados = p.map(MetodoMonteCarlo, args1, args2, args3 )
83      #print(resultados)
84
85      # unir los resultados para producir el resultado final
86      Stotal = 0
87      Ntotal = 0
88      for i in range(0, hilos):
89          Stotal = Stotal + resultados[i][2]
90          Ntotal = Ntotal + math.ceil(N/hilos)
91      #
92      VolR = Stotal / Ntotal
93      VarVorR = (Stotal/Ntotal)*(1-Stotal/Ntotal)/(Ntotal-1)
94
95      return (VolR, VarVorR, Stotal,  time.perf_counter()-t0)
96
97  # end def
98
99  # Formula de Chebyshev
100 def tamMuestraChebyshev(epsilon, delta):
101     """
102     Calculo del tamaño de muestra de acuerdo al criterio de Chebyshev.
103     epsilon: error deseado
104     delta: intervalo de confianda (1-delta)
105     """
106     nc =  1.0 / (4.0 * delta * epsilon**2)
107     return math.ceil(nc)
108 #
109
110 # Formula Teo Central Limite
111 def tamMuestraTeoCentralLimite(epsilon, delta):
112     """
```

```
113     Cálculo del tamaño de muestra de acuerdo al Teorema Central del Límite
114     epsilon: error deseado
115     delta: intervalo de confianda (1-delta)
116     """
117     x = norm.ppf(1.0 - delta/2.0)
118     # nn = norm.ppf(x)**2
119     return math.ceil( ( x/ (2.0*epsilon) ) **2 )
120     # return x
121 #
122
123 # Formula de Hoeffding
124 def tamMuestraHoeffding(epsilon, delta):
125     """
126     Estimacion del tamano de muestra segun Hoeffding.
127     epsilon: error deseado
128     delta: intervalo de confianza
129     """
130     num = 2 * math.log(2/delta)
131     den = 4 * epsilon**2
132     return math.ceil(num/den)
133 # end def
134
135 ## Calculo de int de confianza por Chebyshev
136
137 def intConfianzaChebyshev(S, n, delta):
138     """
139     Intervalo de confianza segun Chebyshev.
140     Parámetros:
141       - S: estimador, cantidad de puntos que caen dentro del volumen
142       - n: cantidad de replicas (puntos sorteados)
143       - delta: margen
144     """
145     def w1(z, n, beta):
146         num = z + beta**2 - beta*math.sqrt( beta**2/4 + z*(n-z)/n )
147         den = n + beta**2
148         return num / den
149     # end def w1
150
151     def w2(z, n, beta):
152         num = z + beta**2 + beta*math.sqrt( beta**2/4 + z*(n-z)/n )
153         den = n + beta**2
154         return num / den
155     # end def w2
156
157     return ( w1(S, n, delta), w2(S, n, delta) )
158 ## end intConfianzaChebyshev
159
160 def intConfianzaAC(S, n, delta):
161     """
162     Intervalo de confianza segun Agresti Coull.
163     Parámetros:
164       - S: estimador, cantidad de puntos que caen dentro del volumen
165       - n: cantidad de replicas (puntos sorteados)
166       - delta: margen, si el intervalo de conf es 95%, entonces delta = 0.05
167     """
168     kappa = norm.ppf(1-delta/2)
169
```

```
170     Xg = S + kappa**2/2
171     ng = n + kappa**2
172 
173     pg = Xg / ng
174     qg = 1 - pg
175 
176     disc = kappa * math.sqrt(pg*qg)*( 1/math.sqrt(ng))
177 
178     return (pg-disc, pg+disc)
179 ## end intConfianzaAC
```