```
"""
Montecarlo para integrales.
(c) Carlos M. martinez, marzo-abril 2022
"""

import random
import math
import tabulate
import time
from scipy.stats import norm
import functools
from cm2c.fing.mmc.utils import sortearPuntoRN
from pathos.multiprocessing import ProcessPool as Pool

_VERSION = "Integracion MMC v0.1.3 - Carlos Martinez abril-mayo 2022"

def version():
    return _VERSION
# end def

def integracionMonteCarlo(Phi, n, sortearPunto):
    """
    Integracion por Montecarlo.
    Phi: funcion a integrar
    n: tamaño de la muestra (cantidad de iteraciones)
    sortearPunto: funcion que sortea un punto en un espacio dim-dimensional
    delta: intervalo de confianza

    Resultado: (estimacion valor integral, estimacion varianza, cant. puntos dentro, T)
    """
    S = 0
    T = 0
    for j in range(1, n+1):
        # sortear X({j} con distribución uniforme en R(n)
        Xj = sortearPunto(0)
        # print(Xj, Phi(Xj))
        if j>1:
            T = T + (1-1/j)*(Phi(Xj)-S/(j-1))**2
```

```
        S = S + Phi(Xj)
    # end for
    estimZ = S / n
    estimSigma2 = T / (n-1)
    estimVar = estimSigma2 / n

    return (estimZ, estimVar, S, T)
## end def

def integracionMonteCarloStieltjes(Kappa, n, sortearPunto):
    """
    Integracion por Montecarlo.
    Phi: funcion a integrar
    n: tamaño de la muestra (cantidad de iteraciones)
    dim: dimensionalidad del problema
    sortearPunto: funcion que sortea un punto en un espacio dim-dimensional con una
    cierta distribucion F
    delta: intervalo de confianza

    Resultado: (estimacion valor integral, estimacion varianza)
    """
    S = 0
    T = 0
    for j in range(1, n+1):
        # sortear Z({j} con distribución dF en R(n)
        Zj = sortearPunto('dummy')
        # print(Xj, Phi(Xj))
        if j>1:
            T = T + (1-1/j)*(Kappa(Zj)-S/(j-1))**2
        S = S + Kappa(Zj)
    # end for
    estimZ = S / n
    estimSigma2 = T / (n-1)
    estimVar = estimSigma2 / n

    return (estimZ, estimVar, S, T)
## end def
```

```
## intervalo de confianza aproximación normal
def intConfianzaAproxNormal(estimZ, estimV, n, delta):
    """
    Intervalo de confianza para la integración de Monte Carlo, según el criterio
    de la aproximación normal.

    estimZ : valor estimado de la integraal
    estimV : valor estimado de la varianza
    n : cantidad de iteraciones
    delta : amplitud del intervalo de confianza
    """

    D = norm.ppf(1-delta/2)*math.sqrt(estimV)

    I0 = estimZ - D

    I1 = estimZ + D

    return (I0, I1)
# end def



# Version paralelizada de Montecarlo
def integracionMonteCarloParalelo(Phi, dim, n, hilos):
    """
        version paralelizada del montecarlo
        N: numero de muestras
        Phi: funcion que implementa el volumen
        hilos: cantidad de hilos en el pool de tareas
    """

    args1 = []
    args2 = []
    args3 = []
    for x in range(0,hilos):
        args3.append( math.ceil(n/hilos) )
        args2.append(dim)
```

```
        args1.append(Phi)

    p = Pool(hilos)
    resultados = p.map(integracionMonteCarlo, args1, args2, args3 )
    #print(resultados)

    # unir los resultados para producir el resultado final
    Stotal = 0
    Ntotal = 0
    Ttotal = 0
    for i in range(0, hilos):
        Stotal = Stotal + resultados[i][2]
        Ttotal = Ttotal + resultados[i][3]
        Ntotal = Ntotal + math.ceil(n/hilos)
    #
    VolR = Stotal / Ntotal
    VarVorR = (Stotal/Ntotal)*(1-Stotal/Ntotal)/(Ntotal-1)

    estimZ = Stotal / Ntotal
    estimSigma2 = Ttotal / (Ntotal-1)
    estimVar = estimSigma2 / Ntotal

    return (estimZ, estimVar, Stotal, Ttotal)
# end def integral montecarlo paralelo

if __name__ == "__main__":
    print("Es una biblioteca, no es para correr directamente")
```