

```

1  """
2  Montecarlo para integrales.
3  (c) Carlos M. martinez, marzo-abril 2022
4  """
5
6  import random
7  import math
8  import tabulate
9  import time
10 from scipy.stats import norm
11 import functools
12 from cm2c.fing.mmc.utils import sortearPuntoRN
13 from pathos.multiprocessing import ProcessPool as Pool
14
15 _VERSION = "Integracion MMC v0.1.3 - Carlos Martinez abril-mayo 2022"
16
17 def version():
18     return _VERSION
19 # end def
20
21 def integracionMonteCarlo(Phi, dim, n, sortearPunto):
22     """
23     Integracion por Montecarlo.
24     Phi: funcion a integrar
25     n: tamaño de la muestra (cantidad de iteraciones)
26     dim: dimensionalidad del problema
27     sortearPunto: funcion que sortea un punto en un espacio dim-dimensional
28     delta: intervalo de confianza
29
30     Resultado: (estimacion valor integral, estimacion varianza)
31     """
32     S = 0
33     T = 0
34     for j in range(1, n+1):
35         # sortear X({j} con distribución uniforme en R(n)
36         Xj = sortearPuntoRN()
37         # print(Xj, Phi(Xj))
38         if j>1:
39             T = T + (1-1/j)*(Phi(Xj)-S/(j-1))**2
40             S = S + Phi(Xj)
41     # end for
42     estimZ = S / n
43     estimSigma2 = T / (n-1)
44     estimVar = estimSigma2 / n
45
46     return (estimZ, estimVar, S, T)
47 ## end def
48
49 def integracionMonteCarloStieltjes(Kappa, dim, n, sortearPunto):
50     """
51     Integracion por Montecarlo.
52     Phi: funcion a integrar
53     n: tamaño de la muestra (cantidad de iteraciones)
54     dim: dimensionalidad del problema
55     sortearPunto: funcion que sortea un punto en un espacio dim-dimensional con una
56     cierta distribucion F
57     delta: intervalo de confianza
58
59     Resultado: (estimacion valor integral, estimacion varianza)
60     """
61     S = 0
62     T = 0
63     for j in range(1, n+1):
64         # sortear Z({j} con distribución dF en R(n)
65         Zj = sortearPunto('dummy')
66         # print(Xj, Phi(Xj))
67         if j>1:
68             T = T + (1-1/j)*(Kappa(Zj)-S/(j-1))**2
69             S = S + Kappa(Zj)
70     # end for
71     estimZ = S / n
72     estimSigma2 = T / (n-1)

```

```

72     estimVar = estimSigma2 / n
73
74     return (estimZ, estimVar, S, T)
75 ## end def
76
77
78 ## intervalo de confianza aproximación normal
79 def intConfianzaAproxNormal(estimZ, estimV, n, delta):
80     """
81     Intervalo de confianza para la integración de Monte Carlo, según el criterio
82     de la aproximación normal.
83
84     estimZ : valor estimado de la integraal
85     estimV : valor estimado de la varianza
86     n : cantidad de iteraciones
87     delta : amplitud del intervalo de confianza
88     """
89
90     D = norm.ppf(1-delta/2)*math.sqrt(estimV)
91
92     I0 = estimZ - D
93
94     I1 = estimZ + D
95
96     return (I0, I1)
97 # end def
98
99
100 # Version paralelizada de Montecarlo
101 def integracionMonteCarloParalelo(Phi, dim, n, hilos):
102     """
103     version paralelizada del montecarlo
104     N: numero de muestras
105     Phi: funcion que implementa el volumen
106     hilos: cantidad de hilos en el pool de tareas
107     """
108
109     args1 = []
110     args2 = []
111     args3 = []
112     for x in range(0, hilos):
113         args3.append( math.ceil(n/hilos) )
114         args2.append(dim)
115         args1.append(Phi)
116
117     p = Pool(hilos)
118     resultados = p.map(integracionMonteCarlo, args1, args2, args3 )
119     #print(resultados)
120
121     # unir los resultados para producir el resultado final
122     Stotal = 0
123     Ntotal = 0
124     Ttotal = 0
125     for i in range(0, hilos):
126         Stotal = Stotal + resultados[i][2]
127         Ttotal = Ttotal + resultados[i][3]
128         Ntotal = Ntotal + math.ceil(n/hilos)
129     #
130     VolR = Stotal / Ntotal
131     VarVorR = (Stotal/Ntotal)*(1-Stotal/Ntotal)/(Ntotal-1)
132
133     estimZ = Stotal / Ntotal
134     estimSigma2 = Ttotal / (Ntotal-1)
135     estimVar = estimSigma2 / Ntotal
136
137     return (estimZ, estimVar, Stotal, Ttotal)
138 # end def integral montecarlo paralelo
139
140 if __name__ == "__main__":
141     print("Es una biblioteca, no es para correr directamente")

```