Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN FÍSICA

TRABAJO DE FIN DE GRADO

OPTIMIZACION DEL PROBLEMA DE THOMSON

SERGIO GONZÁLEZ GONZÁLEZ
Dirigido por
Dr. MANUEL MARTÍN BRAVO
y
Dra. MARÍA FUENCISLA GILSANZ MUÑOZ

CURSO 2022-2023

TÍTULO: OPTIMIZACIÓN DEL PROBLEMA DE THOMSON

AUTOR: SERGIO GONZÁLEZ GONZÁLEZ

TITULACIÓN: GRADO EN FÍSICA

DIRECTORES DEL PROYECTO: : Dr. MANUEL MARTÍN BRAVO y Dra. MARÍA FUENCISLA GILSANZ MUÑOZ

FECHA: MAYO de 2023

1. ABSTRACT

Keywords: Thomson Problem, Basin-Hopping, Bacteriophage T4.

The Thomson problem was proposed in 1904 by the physicist Joseph John Thomson when he developed the thomson's model of the atom, which said that the electrons were evenly distributed inside the atom. This problem consists on finding the structure of a determined number of particles inside a sphere so that the potential energy between them is as minimum as possible; therefore, in equilibrium.

There is not a single function capable of resolving the distribution that the particles must have, so in order to obtain it, we are going to design a Python script in which we will use the Basin-Hopping method.

We will apply the Thomson problem not only for resolving spherical structures but also for resolving the structure of the bacteriophage T4 virus, which has an elliptical head. To tackle this, we develop the functions that will calculate the potential energy and the gradient of the system. Then we will insert them in the algorithm to obtain the results in function of the number of particles and the ellipse parameters. For the positions of the particles we use spherical coordinates, which means that each particle will have two distinct variables: θ, which goes from 0 to π, and ϕ, which goes from 0 to 2π.

2. RESUMEN

Palabras Clave: Problema de Thomson, Basin-Hopping, Bacteriófago T4.

El problema de Thomson fue planteado en 1904 por el físico Joseph John Thomson cuando desarrolló el modelo atómico de Thomson, que decía que los electrones estaban distribuidos uniformemente en el interior del átomo. Este problema consiste en encontrar la distribución de un número determinado de partículas en una esfera de tal manera que la energía potencial entre ellas sea la mínima posible; es decir, que se encuentren en equilibrio.

Para este problema no existe una función capaz de resolver la distribución que tienen que tener las partículas, por lo que para poder llegar a ella vamos a diseñar un programa en Python en el que utilizaremos el método de optimización Basin-Hopping (salto de cuenca).

Aplicaremos el problema de Thomson no solo para resolver estructuras esféricas sino también para resolver la estructura del virus bacteriófago T4, cuya cabeza es una elipse. Para ello desarrollamos unas funciones que calculen la energía potencial y el gradiente del sistema que luego introduciremos en el algoritmo para poder obtener los resultados en función del número de partículas y de los parámetros de la elipse. Para las posiciones de las partículas utilizaremos las coordenadas esféricas, es decir, cada partícula tendrá dos variables distintas: θ, que irá de 0 a π, y ϕ, que irá de 0 a 2π.

3. AGRADECIMIENTOS

A ellos, que me han llenado de buenos momentos durante estos años. A Gonzalo y Alberto por esos fines de semana sin parar. A Félix, Dani y Ainhoa por los momentos en la universidad. A Diego por nuestras conversaciones profundas y sobretodo por apoyarme y ayudarme siempre que lo he necesitado. Sin ellos nada hubiera sido posible, he podido disfrutar por vosotros.

También agradecer a mis profesores por todo lo que me han enseñado, por escuchar nuestras dudas y siempre estar dispuestos a ayudarnos, pero sobretodo por la relación que tenemos para poder tratarnos no solo como estudiantes, sino como amigos.

"Lo que ahoga a alguien no es caerse al río,
sino mantenerse sumergido en él."

Paulo Coelho.

4. TABLA RESUMEN

	Datos
Nombre y Apellidos	Sergio González González
Título del proyecto	Optimización del Problema de Thomson
Directores del proyecto	Dr. Manuel Martín Bravo y Dra. María Fuencisla Gilsanz Muñoz
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa	NO
El proyecto ha implementado un producto	NO
El proyecto ha consistido en el desarrollo de una investigación o innovación	NO
Objetivo general del proyecto	Desarrollar un programa que encuentre soluciones para el Problema de Thomson

Índice

Índice de figuras

Índice de tablas

5. INTRODUCCIÓN Y ESTADO DEL ARTE

El problema de Thomson surgió en 1904 cuando el físico Joseph John Thomson propuso el modelo atómico de Thomson, en el que el átomo estaba compuesto por electrones distribuidos uniformemente en su interior [1]. Es uno de los dieciocho problemas matemáticos que no pudo resolver el matemático Steve Smale y que propuso en su libro "Mathematical Problems for the Next Century"[2]. Tiene como objetivo determinar la distribución de un número determinado de partículas restringidas en una esfera para que la energía potencial del sistema sea mínima [3],[4],[5],[6]. A día de hoy, se aplica este problema o modelo en distintos sistemas con geometrías variadas [7],[8] con el objetivo de acercarse a las configuraciones de mínima energía.

El bacteriófago T4 es uno de los virus más estudiados. Sin embargo, aún no ha sido posible caracterizar estructuralmente el bacteriófago completo en detalle atómico (aunque algunos han comenzado a acercarse) [9] con los múltiples modelos esquemáticos generales publicados [10]. Conocer su estructura y comportamiento resulta clave a la hora de diseñar tratamientos más efectivos, ya que muchos virus presentan dicha forma. En la actualidad se sigue estudiando su funcionamiento y mecanismos celulares [11] con el objetivo de usarlo en aplicaciones médicas.

El estudio de nuestros casos va a ser realizado con Python, un lenguaje de programación que ha ido ganando popularidad durante estos últimos años gracias a su gran versatilidad. También tiene incorporado en la librería 'scipy' el método de Basin-Hopping (salto de cuenca) [12], que a día de hoy es usado para optimizar superficies [13] y estructuras cristalinas [14] entre otras cosas.

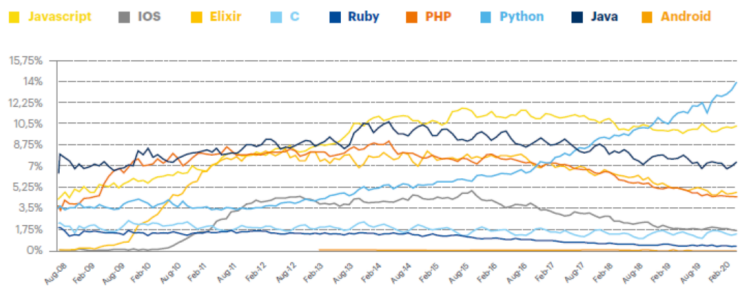


Figura 1: Evolución Lenguajes de Programación. Tomada de Epitech-it.

El método Basin-Hopping es un algoritmo de optimización que va distribuyendo las partículas teniendo en cuenta las distribuciones que han resultado en menor energía [15]. Para ello va iterando y realiza perturbaciones a las coordenadas, donde después las acepta o desecha dependiendo de la función minimizada [16].

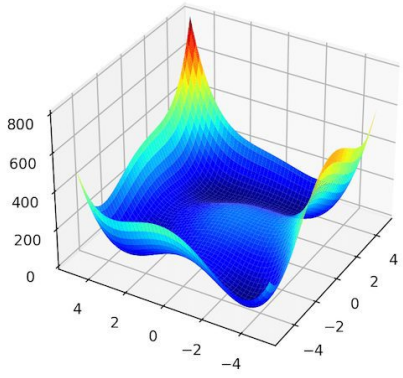


Figura 2: Ejemplo Basin-Hopping. Tomada de TopBigData.

En esta imagen podemos observar un ejemplo de como buscaríamos la menor energía. Gracias al método Basin-Hopping podemos centrarnos en esa zona donde se puede apreciar una caída, que es lo que buscamos ya que esa sería la zona de energía mínima. Así podemos centrarnos en buscar las distribuciones que están en esa zona y poder ajustar nuestras distribuciones de tal manera que consigamos la mínima energía posible.

Este algoritmo se suele utilizar para poder encontrar estructuras de mínima energía en moléculas, que es justo lo que queremos hacer en este proyecto. También está demostrado que el modelo de Basin-Hopping es el que da mejores resultados para el Problema de Thomson cuando $N < 1000$ [17], y por ello, es el modelo que hemos seleccionado.

Para apoyar nuestros resultados utilizaremos la base de datos de Wales [17], donde están recogidos los resultados para distintos números de partículas en una esfera de radio 1.

6. OBJETIVOS DEL PROYECTO

El objetivo del proyecto es desarrollar un programa eficiente que consiga resultados aproximados de distribuciones para N partículas y poder visualizarlas, de tal manera que los resultados obtenidos se acerquen lo máximo posible a los resultados que ya existen. Es importante tener en cuenta que no existe ninguna manera analítica de poder obtener los resultados de estas distribuciones [3], por lo que los resultados que se obtienen mediante estos algoritmos de optimización son aproximaciones y para algunos números concretos de partículas puede llegar a ser muy difícil encontrar una solución.

Lo primero que haremos será diseñar un script que consiga encontrar las estructuras de mínima energía, después mejoraremos y optimizaremos las funciones para que el programa compile lo más rápido posible, y finalmente sacaremos resultados para distintos números de partículas y el bacteriófago T4 y los compararemos con los resultados ya conocidos.

7. DESARROLLO

7.1. Energía Potencial

Lo primero que necesitamos es la energía potencial resultante entre las partículas contenidas en el elipsoide. Para ello utilizaremos la siguiente fórmula:

$$V = \sum_{i<j}^{j=N} \frac{1}{r_{ij}} \tag{1}$$

donde r_{ij} es la distancia entre dos partículas. Poniendo el sumatorio de esta manera nos aseguramos que nunca contaremos la energía dos veces entre dos mismas partículas, ya que por ejemplo la energía entre la partícula 1 y 2 solo tenemos que tenerla en cuenta una vez [5],[6],[15],[17],[18].

Para calcular esta distancia, tenemos que transformar las posiciones de esféricas a cartesianas:

$$\begin{aligned} x_n &= a \cdot \sin(\theta_n) \cdot \cos(\phi_n) \\ y_n &= b \cdot \sin(\theta_n) \cdot \sin(\phi_n) \\ z_n &= c \cdot \cos(\theta_n) \end{aligned} \tag{2}$$

donde a, b, y c son los radios que definen la elipse, y θ_n y ϕ_n las variables que definen la posición de cada partícula.

Finalmente, para calcular r_{ij} simplemente habrá que hacer el módulo del vector resultante al restar las posiciones de dos partículas:

$$r_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{3}$$

7.2. Gradiente Numérico

El gradiente numérico no es necesario para el Basin-Hopping pero nos sirve para corroborar que los resultados obtenidos son correctos. Se calcula con la definición de derivada; y al tener cada partícula dos variables (θ y ϕ), tenemos que calcular dos límites distintos para cada partícula:

$$\begin{aligned} &\lim_{h \to 0} \frac{V(\theta + h, \phi) - V(\theta, \phi)}{h} \\ &\lim_{h \to 0} \frac{V(\theta, \phi + h) - V(\theta, \phi)}{h} \end{aligned} \tag{4}$$

Estos resultados los guardaremos en un vector cuya dimensión será de $N \cdot D$, donde N es el número de partículas y D=2 ya que nuestras partículas están definidas en coordenadas esféricas (θ y ϕ) y solo tienen estas dos variables. Finalmente haremos el módulo de este vector para hallar el valor del gradiente de las partículas en esa posición.

7.3. Gradiente Exacto

Primero calculamos las distintas derivadas parciales respecto $\theta_i, \phi_i, \theta_j$ y ϕ_j de la energía potencial:

$$A = \left(\left(b^2 \sin^2(\varphi_1) + a^2 \cos^2(\varphi_1) - c^2\right)\cos(\theta_1) + c^2\cos(\theta_2)\right)\sin(\theta_1) + \\ + \left(b^2 \sin(\varphi_1)\sin(\varphi_2) - a^2\cos(\varphi_1)\cos(\varphi_2)\right)\sin(\theta_2)\cos(\theta_1)$$

$$B = \left(\left(b^2 \sin^2(\varphi_2) + a^2 \cos^2(\varphi_2) - c^2\right)\cos(\theta_2) + c^2\cos(\theta_1)\right)\sin(\theta_2) + \\ + \left(-b^2 \sin(\varphi_1)\sin(\varphi_2) - a^2\cos(\varphi_1)\cos(\varphi_2)\right)\sin(\theta_1)\cos(\theta_2)$$

$$C = \sin(\theta_1)\left(\left(\left(b^2 - a^2\right)\sin(\theta_1)\cos(\varphi_1) + a^2\cos(\varphi_2)\sin(\theta_2)\right)\sin(\varphi_1) + \\ -b^2\sin(\varphi_2)\sin(\theta_2)\cos(\varphi_1)\right)$$

$$D = \sin(\theta_2)\left(\left(\left(b^2 - a^2\right)\sin(\theta_2)\cos(\varphi_2) + a^2\cos(\varphi_1)\sin(\theta_1)\right)\sin(\varphi_2) + \\ -b^2\sin(\varphi_1)\sin(\theta_1)\cos(\varphi_2)\right)$$

$$E = \left((b\sin(\varphi_2)\sin(\theta_2) - b\sin(\varphi_1)\sin(\theta_1))^2 + (a\cos(\varphi_2)\sin(\theta_2) - a\cos(\varphi_1)\sin(\theta_1))^2 + \right. \\ \left. + (c\cos(\theta_2) - c\cos(\theta_1))^2\right)^{\frac{3}{2}}$$

Derivada respecto θ_1:

$$-\frac{A}{E} \tag{5}$$

Derivada respecto θ_2:

$$-\frac{B}{E} \tag{6}$$

Derivada respecto ϕ_1:

$$-\frac{C}{E} \tag{7}$$

Derivada respecto ϕ_2:

$$-\frac{D}{E} \tag{8}$$

Al conocer las distintas derivadas parciales, tendremos que calcular un vector de dimensiones $N \cdot D$, donde cada posición corresponde a la suma total de su correspondiente variable con las demás partículas.

7.4. Sumatorio Radios

Esta función, al igual que el gradiente numérico, no es necesaria para el Basin-Hopping pero es interesante para calcular la distancia final entre todas las partículas. Simplemente sumamos los vectores de posición de cada partícula y hacemos la norma del vector resultante:

$$\left\| \sum_{i=1}^{N} (x_i, y_i, z_i) \right\| \tag{9}$$

Cuanto más próximo a 0, más simétrica es la estructura, y a pesar de que la mayoría de simetrías requieren que el sumatorio vectorial de las posiciones sea cero, dependiendo del número de partículas hay estructuras que no son simétricas.

7.5. Implementación

Tendremos que implementar estas funciones en nuestro programa para poder realizar el análisis con el Basin-Hopping. Es importante optimizar las funciones porque a medida que aumente el número de partículas e iteraciones, el tiempo de compilación aumenta exponencialmente [3].

Primero simularemos aleatoriamente N partículas en la elipse que hayamos definido, y finalmente agregaremos las posiciones de estas a la función de Basin-Hopping junto a la función de Energía Potencial y Gradiente Exacto. Finalmente obtendremos un resultado con las nuevas posiciones, con las que calcularemos la energía potencial, sumatorio de radios, gradiente exacto y numérico resultantes para corroborar que el resultado es correcto.

8. RESULTADOS

Los resultados obtenidos los podemos contrastar con la base de datos de Wales [17].

8.1. Pocas Partículas

Para 12 y 34 partículas en una esfera de radio 1, los resultados obtenidos de energía potencial son exactamente iguales a los teóricos [17]. También podemos comprobar que los gradientes y el sumatorio de radios son prácticamente 0 ya que la diferencia entre cifras significativas es de orden 7 y 8, por lo que podemos confirmar que estos resultados nos dan la energía mínima del sistema.

	N=12	N=34
Energía Potencial	49.1652531	468.9048533
Gradiente Numérico	0.0000028	0.0000829
Gradiente Exacto	0.0000017	0.0000209
Sumatorio Radios	0.0000002	0.0000029

Tabla 1: Resultados para pocas partículas

Figura 3: Representaciones para 12 y 34 partículas respectivamente. [17]

8.2. 192 Partículas

Para este caso cada iteración requiere ya de unos 9 minutos, pero vamos a comparar los resultados obtenidos con 50 iteraciones y con 100.

	50 iteraciones	100 iteraciones
Energía Potencial	16963.338386	16963.338386
Gradiente Numérico	0.011972	0.009542
Gradiente Exacto	0.000217	0.000057
Sumatorio Radios	0.000006	0.000002

Tabla 2: Resultados para 192 partículas

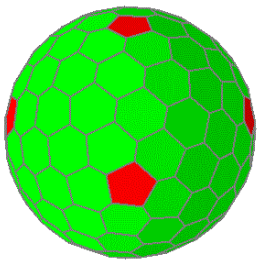


Figura 4: Representación de 192 partículas. [17]

Podemos observar que la energía potencial entre ambos resultados no difieren a pesar de duplicar el número de iteraciones, y que son iguales a los teóricos [17]. Lo único que conseguimos es reducir el gradiente y la función del sumatorio de radios (que por la diferencia de cifras significativas también podemos decir que es 0), por lo que podemos deducir que al aumentar el número de iteraciones, las posiciones resultantes se van acercando más a las posiciones de equilibrio. Aun así, si solo queremos conocer la energía potencial mínima de un sistema y no las posiciones con exactitud, no es necesario iterar demasiadas veces.

8.3. Bacteriófago T4

El bacteriófago T4 es un virus que tiene un tamaño aproximadamente de unos 200 nanómetros (1280 armstrongs de largo y 860 armstrongs de ancho) [19]. Tiene una estructura compleja que consta de una cabeza (donde lleva el ADN), un cuello y una cola. Lo que vamos a hacer nosotros es ver la estructura de la cabeza.

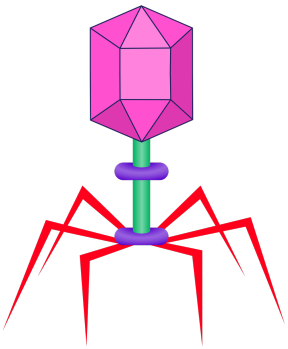


Figura 5: Estructura Bacteriófago T4. Tomada de Pixabay

Para ello debemos saber el tamaño de esta cabeza, que es aproximadamente de 950 armstrongs de largo y 700 armstrongs de ancho [19]. Por último debemos saber el número de partículas que tiene, y para ello conocemos que una estructura tipo T tiene 12 pentámeros, 10·(T-1) hexámeros y otros 10·T+2 capsómeros, por lo que al ser T=4, resulta en 84 partículas [20],[21].

Una vez sabiendo esto, introducimos los datos en el programa y los resultados obtenidos son los siguientes:

Bacteriófago T4	
Energía Potencial	2778.17478
Gradiente Numérico	0.00108
Gradiente Exacto	0.00003
Sumatorio Radios	0.00225

Tabla 3: Resultados del Bacteriófago T4

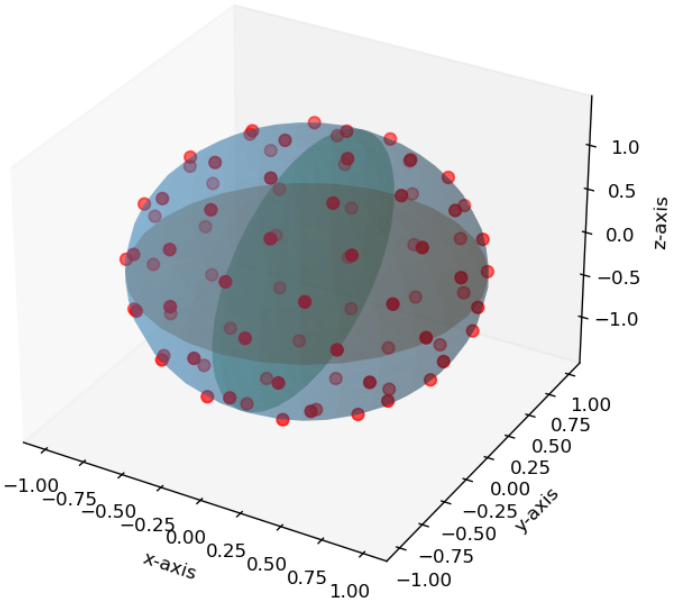


Figura 6: Representación Cabeza Bacteriófago T4

Podemos observar que la energía resultante es menor que la teórica [17] para una esfera de radio 1 con el mismo número de partículas (3103.46512) debido a que al ser mayor el volumen, tenemos más espacio para distribuir estas partículas y que la energía potencial entre ellas sea menor.

9. DISCUSIÓN Y CONCLUSIONES

A pesar de que los resultados obtenidos son correctos, es importante destacar que conseguir resultados para un mayor número de partículas es muy complicado debido al tiempo que tarda en compilar el programa [3]. Esto se puede disminuir con un ordenador mejor u optimizando las funciones.

Al principio las funciones estaban desarrolladas mediante bucles por lo que el programa no era muy eficiente y tardaba demasiado, pero al modificar el código para que los cálculos los hiciera mediante operaciones con vectores e implementar en las funciones más argumentos para que no tuviera que acceder a variables globales, conseguimos reducir el tiempo de compilación a un cuarto. Aun así, para 192 partículas, 100 iteraciones tardaron unas 15

horas en ejecutarse, por lo que sacar resultados para un número mayor era muy complicado. No obstante, recalco que podemos confirmar que el código es correcto y que los resultados obtenidos, a pesar del tiempo de ejecución, son iguales a los teóricos [17].

10. FUTURAS LÍNEAS DE TRABAJO

Como futuras líneas de trabajo, lo primero sería intentar seguir optimizando el programa lo máximo posible, ya que el gradiente exacto se tuvo que hacer mediante bucles y eso ralentiza el tiempo de ejecución. Después podríamos estudiar diferentes estructuras para N más grandes y poder contrastarlas con otros resultados.

En otras ramas de investigación, podríamos incorporar este método en la teoría de la repulsión para estudiar las estructuras de las moléculas [22]. También se puede utilizar para observar los distintos patrones de fullerenos que tienen los átomos de carbono [23], o si aparecen nuevos virus con estructura elíptica, conocer su distribución y estructura al igual que hemos hecho con el bacteriófago T4.

11. ANEXOS

Adjunto un ejemplo del código que utilizamos para poder calcular la energía mínima para 10 partículas. Si queremos calcular otro tipo de estructura o de número de partículas solo tendríamos que cambiar las variables 'a', 'b', 'c' y 'n' del programa, que corresponden a los radios de la elipse y al número de partículas.

El código tiene escrito comentarios sobre lo que es cada parte y por último una explicación de las distintas variables que nos pide el algoritmo Basin-Hopping.

En el anexo se puede observar como la función 'distancia' al final no fue utilizada ya que el cálculo de la energía potencial lo hice con operaciones con vectores para que el programa no tuviera que estar accediendo a la función constantemente, realizar menos bucles y poder así mejorar el rendimiento del programa.

```
import numpy as np          #para senos, cosenos, pi, norma de un
vector
from numpy import linalg as LA
import random as rand          #para poder generar posiciones
aleatorias
import matplotlib.pyplot as plt      #para poder graficar los
resultados
from matplotlib import style
from mpl_toolkits.mplot3d import Axes3D
import scipy            #para poder utilizar el algoritmo basin-
hopping
from scipy import optimize
from scipy.optimize import basinhopping
import time                          #para calcular tiempos de
ejecución
```

FUNCIONES

```
def distancia(q1, q2, a, b, c): #calcula la distancia entre dos
partículas, q1 corresponde al vector de ángulos de
                                #posición de la primera partícula y q2
de la segunda, a b y c radios de la elipse

    sinth1, costh1 = np.sin(q1[0]), np.cos(q1[0])
    sinph1, cosph1 = np.sin(q1[1]), np.cos(q1[1])
    sinth2, costh2 = np.sin(q2[0]), np.cos(q2[0])
    sinph2, cosph2 = np.sin(q2[1]), np.cos(q2[1])

    x1, y1, z1 = a*sinth1*cosph1, b*sinth1*sinph1, c*costh1
    x2, y2, z2 = a*sinth2*cosph2, b*sinth2*sinph2, c*costh2

    dx, dy, dz = x2-x1, y2-y1, z2-z1

    r = np.sqrt(dx*dx + dy*dy + dz*dz)

    return r

def sumar(ang, a, b, c): #función sumatorio de radios para calcular la
distancia entre todas las partículas
                         #ang es el vector de posiciones de todas las
partículas
    sinth = np.sin(ang[::2]) #calcula un vector con todos los senos de
theta
    costh = np.cos(ang[::2]) #calcula un vector con todos los cosenos
de theta
    sinph = np.sin(ang[1::2]) #calcula un vector con todos los senos
de phi
    cosph = np.cos(ang[1::2]) #calcula un vector con todos los cosenos
de phi
```

```
    x = a*sinth*cosph   #vector con posiciones x
    y = b*sinth*sinph   #vector con posiciones y
    z = c*costh #vector con posiciones z

    rf = np.sum(np.column_stack((x, y, z)), axis=0)
    return np.linalg.norm(rf)

def Potencial(ang, a, b, c): #función para calcular la energía
potencial del sistema
    sinth = np.sin(ang[::2])
    costh = np.cos(ang[::2])
    sinph = np.sin(ang[1::2])
    cosph = np.cos(ang[1::2])

    x = a*sinth*cosph
    y = b*sinth*sinph
    z = c*costh

    n_part = int(len(ang)/2) #número de partículas
    V = 0
    for i in range(n_part): #cada partícula se resta con las
siguientes, se calcula un vector de energía potencial y se suma
        dx, dy, dz = x[i+1:]-x[i], y[i+1:]-y[i],z[i+1:]-z[i]
        d = np.sqrt(dx*dx + dy*dy + dz*dz)
        V += np.sum(1 / d)
    return V

def GradNumVec(ang, a, b, c, h=1e-8): #función para calcular el vector
del gradiente numérico
    angh = ang.copy()
    n_part = int(len(ang)/2)
    V0 = Potencial(ang,a,b,c)
    g = np.zeros(shape=len(ang))
    for i in range(n_part):
        angh[2*i] += h    #sumamos h a la posición correspondiente
        VH = Potencial(angh,a,b,c)
        g[2*i] = (VH-V0)/h   #calculamos la derivada en ese punto y
guardamos en vector g en la posición correspondiente
        angh[2*i] -= h   #dejamos el vector como antes restando h a
ese punto

        angh[2*i+1] += h
        VH = Potencial(angh,a,b,c)
        g[2*i+1] = (VH-V0)/h
        angh[2*i+1] -= h

    return g

def GradienteExacto(ang, a, b, c): #función para calcular el vector
del gradiente exacto
```

```
    g=np.zeros(shape=len(ang))
    for i in range (int(len(ang)/2)):
        th1 = ang[i*2:i*2+2][0] #theta_i
        ph1 = ang[i*2:i*2+2][1] #phi_i

        sinth1, costh1 = np.sin(th1), np.cos(th1)
        sinph1, cosph1 = np.sin(ph1), np.cos(ph1)

        for j in range(i+1,int(len(ang)/2)):
            th2 = ang[j*2:j*2+2][0] #thetaj
            ph2 = ang[j*2:j*2+2][1] #phij

            sinth2, costh2 = np.sin(th2), np.cos(th2)
            sinph2, cosph2 = np.sin(ph2), np.cos(ph2)

            #Denominador
            D = ((b*sinth2*sinph2-b*sinph1*sinth1)**2+
(a*cosph2*sinth2-a*cosph1*sinth1)**2+(c*costh2-c*costh1)**2)**(3/2)
            #Derivada respecto theta_i
            T1 = -(((b**2*sinph1**2+a**2*cosph1**2-
c**2)*costh1+c**2*costh2)*sinth1+(-b**2*sinth2*sinph1*sinph2-
a**2*sinth2*cosph1*cosph2)*costh1)/D
            #Derivada respecto phi_i
            P1 = -(sinth1*(((b**2-
a**2)*sinth1*cosph1+a**2*sinth2*cosph2)*sinph1-
b**2*sinth2*sinph2*cosph1))/D
            #Derivada respecto theta_j
            T2 = -(((b**2*sinph2**2+a**2*cosph2**2-
c**2)*costh2+c**2*costh1)*sinth2+(-b**2*sinth1*sinph1*sinph2-
a**2*sinth1*cosph1*cosph2)*costh2)/D
            #Derivada respecto phi_j
            P2 = -(sinth2*(((b**2-
a**2)*sinth2*cosph2+a**2*sinth1*cosph1)*sinph2-
b**2*sinth1*sinph1*cosph2))/D

            g[2*i] += T1
            g[2*i+1] += P1
            g[2*j] += T2
            g[2*j+1] += P2
    return g
```

Ejemplo para 3 Particulas en equilibrio (Triángulo Equilátero) para ver que las funciones están bien

```
a = b = c = 1   #esfera de radio 1
ang0 = np.array([np.pi/2, 0,            #Posiciones de equilibrio de 3
partículas
                 np.pi/2, 2*np.pi/3,
                 np.pi/2, -2*np.pi/3])
```

```
print("Potencial Triángulo Equilátero:", Potencial(ang0,a,b,c))
print("Gradiente Numérico Vector:", LA.norm(GradNumVec(ang0,a,b,c)))
print("Gradiente Exacto:", LA.norm(GradienteExacto(ang0,a,b,c)))
print("Sumatorio radios:", sumar(ang0,a,b,c))
```

```
Potencial Triángulo Equilátero: 1.7320508075688774
Gradiente Numérico Vector: 3.845925372767128e-08
Gradiente Exacto: 2.0559182460496663e-16
Sumatorio radios: 4.805827935071591e-16
```

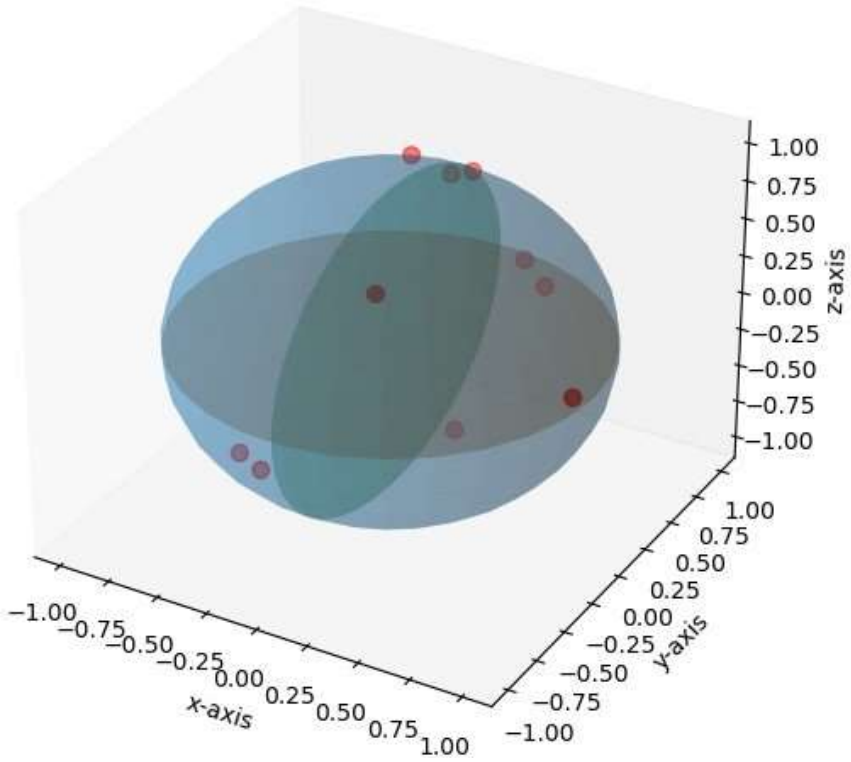
EJEMPLO PARA 10 PARTÍCULAS ESFERA RADIO 1

```
#DATOS INICIALES PARA 10 PARTÍCULAS
a = 1
b = 1
c = 1
n = 10 #número de partículas
angv = [] #lista donde guardaremos las posiciones aleatorias de cada
partícula

for i in range(n):
    angv.append(rand.uniform(0,np.pi)) #guardamos posicion de theta_i
en angv
    angv.append(rand.uniform(0,np.pi)*2) #guardamos posicion de phi_i
en angv

angv = np.asarray(angv) #vector de posiciones de las partículas

print("\033[1m DATOS INICIALES N =", n, "PARTÍCULAS\033[0m")
print("Energía Potencial", Potencial(angv,a,b,c))
print("Gradiente Numérico:", LA.norm(GradNumVec(angv,a,b,c)))
print("Gradiente Exacto:", LA.norm(GradienteExacto(angv,a,b,c)))
print("Sumatorio radios:", sumar(angv,a,b,c))
```

```
 DATOS INICIALES N = 10 PARTÍCULAS
Energía Potencial 54.92059215377027
Gradiente Numérico: 99.59401432619248
Gradiente Exacto: 99.59401428234007
Sumatorio radios: 3.1519391848632905
```

```
#Creamos figura con datos iniciales
%matplotlib widget
fig = plt.figure()
ax = Axes3D(fig)
ax.grid(False) #quita grid

#listas donde guardaremos coordenadas
xi = []
yi = []
```

```
zi = []
for i in range(n):
    xi.append(a*np.sin(angv[i*2])*np.cos(angv[i*2+1]))
    yi.append(b*np.sin(angv[i*2])*np.sin(angv[i*2+1]))
    zi.append(c*np.cos(angv[i*2]))

#ploteando partículas
plot_geeks = ax.scatter(xi, yi, zi, s=50, color='red')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')

#ploteando esfera y elipses interiores para poder apreciar mejor la
posición de las partículas
u, v = np.mgrid[0:2 * np.pi:30j, 0:np.pi:20j]
xe = a*np.cos(u) * np.sin(v)
ye = b*np.sin(u) * np.sin(v)
ze = c*np.cos(v)
ax.plot_surface(xe, ye, ze,alpha=0.3)
ax.plot_surface(xe, ye, ze*0,alpha=0.2)
ax.plot_surface(xe*0, ye, ze,alpha=0.2)

plt.show()
```

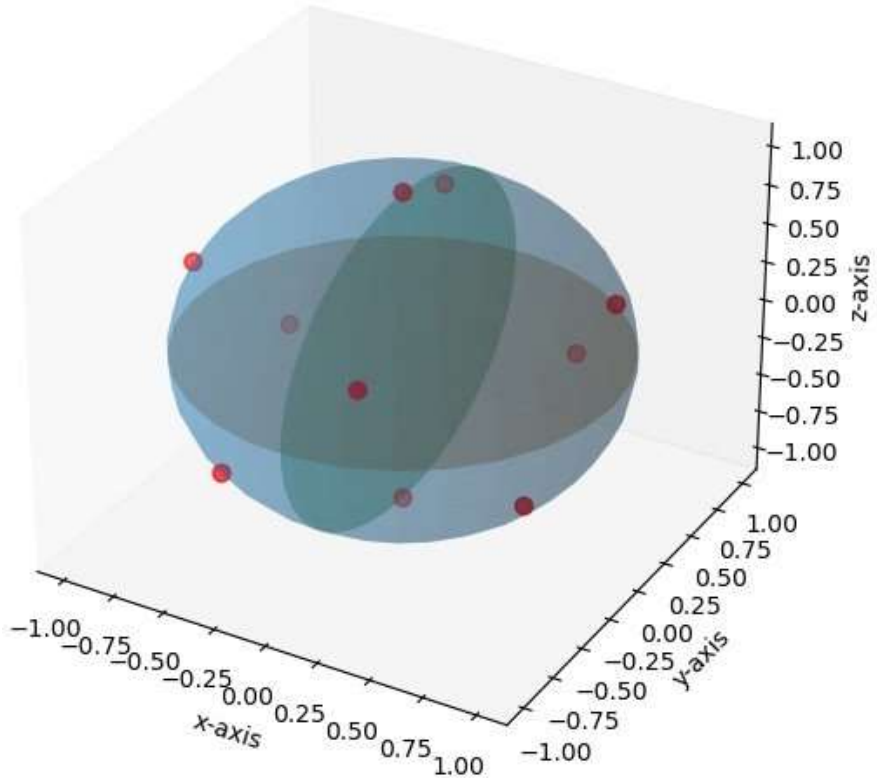


```
#APLICAMOS BASIN-HOPPING
res = scipy.optimize.basinhopping( func=Potencial, x0=angv, T=1.0,
stepsize=0.5,
                                  minimizer_kwargs= {'method':'CG',
'jac':GradienteExacto, "args":(a,b,c)}, niter=200)

print("\033[1m RESULTADOS N =", n, "PARTÍCULAS\033[0m")
print("Energía Potencial", Potencial(res.x,a,b,c))
print("Gradiente Numérico:", LA.norm(GradNumVec(res.x,a,b,c)))
print("Gradiente Exacto:", LA.norm(GradienteExacto(res.x,a,b,c)))
print("Sumatorio radios:", sumar(res.x,a,b,c))
```

```
 RESULTADOS N = 10 PARTÍCULAS
Energía Potencial 32.71694946016465
Gradiente Numérico: 6.275343527539581e-06
Gradiente Exacto: 6.38494568893869e-06
Sumatorio radios: 1.4871113742805048e-06
```

```
#Creamos figura con el resultado
%matplotlib widget
fig = plt.figure()
ax = Axes3D(fig)
ax.grid(False) #quita grid

#listas donde guardaremos coordenadas
xf = []
yf = []
zf = []
for i in range(n):
    xf.append(a*np.sin(res.x[i*2])*np.cos(res.x[i*2+1]))
    yf.append(b*np.sin(res.x[i*2])*np.sin(res.x[i*2+1]))
    zf.append(c*np.cos(res.x[i*2]))

#ploteando partículas
plot_geeks = ax.scatter(xf, yf, zf, s=50, color='red')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
```

```
#ploteando esfera y elipses interiores para poder apreciar mejor la
posición de las partículas
u, v = np.mgrid[0:2 * np.pi:30j, 0:np.pi:20j]
xe = a*np.cos(u) * np.sin(v)
ye = b*np.sin(u) * np.sin(v)
ze = c*np.cos(v)
ax.plot_surface(xe, ye, ze,alpha=0.3)
ax.plot_surface(xe, ye, ze*0,alpha=0.2)
ax.plot_surface(xe*0, ye, ze,alpha=0.2)

plt.show()
```



Explicación de datos que nos pide el algoritmo basin-hopping:

-func: la función que queremos minimizar, en este caso la energía potencial.

-x0: el primer argumento de la función que queremos optimizar y el que va a ser modificado, que es el vector de posiciones iniciales.

-T: la temperatura, refiriéndose al criterio de aceptación del algoritmo. Cuanto mayor sea T, más posibilidades de tomar resultados malos tiene, pero a su vez esto puede llevar a una zona de menor energía. El valor predeterminado es el que utilizamos.

-stepsize: es el valor máximo que puede variar cada variable de x0 en cada iteración. A menor stepsize, menos variación tendrá por lo que a lo mejor nunca llegamos a la zona de mínima energía, pero también puede ser que al ser un stepsize muy grande siempre estemos pasando esa zona. El valor predeterminado es el que utilizamos.

-minimizer_kwargs: es un diccionario que tiene como claves "method", "jac" y "args". Method es el tipo de derivada que utilizas para minimizar, en este caso CG ("conjugate gradient"). Jac es la función de gradiente que va a utilizar el algoritmo, que en este caso es la función "GradienteExacto" que hemos creado, y args son los argumentos extras que utilizan la función "Potencial" y "GradienteExacto". Para ello la función "GradienteExacto" y "Potencial" deben tener los mismos argumentos, ya que ambos utilizan x0 como primer argumento y es el que varía el algoritmo, y los demás argumentos está bien meterlos en las funciones ya que ayudan a optimizar el programa y el rendimiento debido a que no tienen que estar en cada ejecución accediendo a variables globales del sistema.

-niter: el número de iteraciones del algoritmo.

Referencias

[1] J. J. Thomson, "XXIV. On the structure of the atom: an investigation of the stability and periods of oscillation of a number of corpuscles arranged at equal intervals around the circumference of a circle; with application of the results to the theory of atomic structure," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 7, n.º 39, pp. 237-265, 1904.

[2] S. Smale, "Mathematical problems for the next century," *Mathematics: frontiers and perspectives*, pp. 271-294, 2000.

[3] C. Luque, P. Isasi y J. C. Hernández, "Distribución de cargas en una esfera mediante estrategias evolutivas," 2004.

[4] N. A. Garcıa, L. R. Gómez, E. M. Vallés, D. A. Vega et al., "El problema Thomson en copolımeros bloque confinados a cascarones esféricos," *Mecánica Computacional*, vol. 28, n.º 29, pp. 2485-2493, 2009.

[5] A. BĂUTU y E. BĂUTU, "Energy minimization of point charges on a sphere with particle swarms," en *Paper presented at the 8th International Balkan Workshop on Applied Physics*, vol. 5, 2007, p. 7.

[6] H. LAKHBAB, S. El Bernoussi y A. El Harif, "Energy minimization of point charges on a sphere with a hybrid approach," *Applied Mathematical Sciences*, vol. 6, n.º 30, pp. 1487-1495, 2012.

[7] L. Kurakin e I. Ostrovskaya, "Resonances in the stability problem of a Thomson vortex N-gon inside/outside circular domain and a point vortex quadrupole on a plane," *Advances in Nonlinear Science*, p. 47, 2021.

[8] E. S. Chen, "Simulations on Adapted Thomson Problem with Different Frames," 2021.

[9] J. F. Hunt, S. M. Van der Vies, L. Henry y J. Deisenhofer, "Structural adaptations in the specialized bacteriophage T4 co-chaperonin Gp31 expand the size of the Anfinsen cage," *Cell*, vol. 90, n.º 2, pp. 361-371, 1997.

[10] P. Leiman, S. Kanamaru, V. Mesyanzhinov, F. Arisaka y M. Rossmann, "Structure and morphogenesis of bacteriophage T4," *Cellular and Molecular Life Sciences CMLS*, vol. 60, pp. 2356-2370, 2003.

[11] M. L. Yap y M. G. Rossmann, "Structure and function of bacteriophage T4," *Future microbiology*, vol. 9, n.º 12, pp. 1319-1327, 2014.

[12] A. Verma, A. Schug, K. Lee y W. Wenzel, "Basin hopping simulations for all-atom protein folding," *The Journal of chemical physics*, vol. 124, n.º 4, p. 044 515, 2006.

[13] M. N. Bauer, M. I. Probert y C. Panosetti, "Systematic Comparison of Genetic Algorithm and Basin Hopping Approaches to the Global Optimization of Si (111) Surface Reconstructions," *The Journal of Physical Chemistry A*, vol. 126, n.º 19, pp. 3043-3056, 2022.

[14] S. Yang y G. M. Day, "Exploration and optimization in crystal structure prediction: Combining basin hopping with quasi-random sampling," *Journal of Chemical Theory and Computation*, vol. 17, n.º 3, pp. 1988-1999, 2021.

[15] D. J. Wales y J. P. Doye, "Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms," *The Journal of Physical Chemistry A*, vol. 101, n.º 28, pp. 5111-5116, 1997.

[16] M. M. Bravo, "Simulación numérica de cápsides vıricas con modelos de grano grueso," Tesis doct., Universidad de La Laguna, 2022.

[17] D. J. Wales y S. Ulker, "Structure and dynamics of spherical crystals characterized for the Thomson problem," *Physical Review B*, vol. 74, n.º 21, p. 212 101, 2006.

[18] V. A. Yudin, "Minimum potential energy of a point system of charges," *Diskret. Mat*, vol. 4, n.º 2, pp. 115-121, 1992.

[19] Q. Fang et al., "Structures of a large prolate virus capsid in unexpanded and expanded states generate insights into the icosahedral virus assembly," *Proceedings of the National Academy of Sciences*, vol. 119, n.º 40, e2203272119, 2022.

[20] A. Luque, R. Zandi y D. Reguera, "Optimal architectures of elongated viruses," *Proceedings of the National Academy of Sciences*, vol. 107, n.º 12, pp. 5323-5328, 2010.

[21] D. L. Caspar y A. Klug, "Physical principles in the construction of regular viruses," en *Cold Spring Harbor symposia on quantitative biology*, Cold Spring Harbor Laboratory Press, vol. 27, 1962, pp. 1-24.

[22] C. E. BUELNA GARCIA, "Estudio de estructuras de cúmulos bimetálicos de Cu-Ni mediante el método Basin-Hopping," 2017.

[23] Á. Morales Garcıa, "Qumica cuántica de polimorfos inducidos por presión: óxidos binarios y ternarios de silicio y carbono," 2014.