hNombre: Juan Diego Figueroa Hernández **Código:** 2200815
Juan Andrés Guarín Rojas 2201870
Gabriela Sánchez Ariza 2200816
Nicolas Toledo Parra 2200017

Problema de N-Cuerpos

En física, la cuestión del problema de los n-cuerpos trata de determinar los movimientos individuales de un grupo de partículas materiales las cuales interactúan constantemente con el conjunto entero de partículas, por lo tanto, la descripción del movimiento de cada cuerpo altera la descripción del movimiento del resto de cuerpos y por ende la del cuerpo en si misma de nuevo y así sucesivamente. En sus orígenes el problema se planteó para un conjunto de objetos astronómicos que interactúan mutuamente según las leyes de la gravitación universal de Newton.

Utilidad del problema desde el punto de vista físico

De resolver el problema se podría predecir los movimientos del Sol, la Luna, los planetas y las estrellas visibles, o en general cualquier tipo de sistema de n-cuerpos que afecten su movimiento entre sí mismos.

Descripción matemática del método seleccionado Método de Runge-Kutta:

Los métodos de Runge-Kutta (RK) son un conjunto de métodos iterativos (implícitos y explícitos) para la aproximación de soluciones de ecuaciones diferenciales ordinarias, concretamente, del problema de valor inicial.
Sean:

$$y'(t) = f(t, y(t))$$

una ecuación diferencial ordinaria, con $f\colon \Omega \subset R \times R^n \to R^n$ donde Ω es un conjunto abierto, junto con la condición de que el valor inicial de f sea

$$(t_0, y_0) \ pertenecientes\ a\ \Omega$$

Entonces el método RK (de orden s) tiene la siguiente expresión, en su forma más general:

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \qquad [1]$$

donde h es el paso por iteración, o lo que es lo mismo, el incremento $\Delta\, t_n$ entrelos sucesivos t_n y t_{n+1}. Los coeficientes k_i son términos de aproximación intermedios, evaluados en f de manera local.

$$k_i = f\left(t_n + h\,c_i\,, y_n + h\sum_{j=1}^{s} a_{ij}k_j\right) \quad i = 1,\ldots,s. \qquad [2]$$

con a_{ij}, b_i, c_i coeficientes propios del esquema numérico elegido, dependiente de la regla de cuadratura utilizada. Los esquemas Runge-Kutta pueden ser explícitos o implícitos dependiendo de las constantes a_{ij} del esquema. Si esta matriz es triangular inferior con todos los elementos de la diagonal principal iguales a cero; es decir, $a_{ij} = 0$ para $j = i, \ldots, s$ los esquemas son explícitos.

Aplicación del Método de Runge-Kutta al Problema N-Cuerpos

En este problema se usó una variante del método de Runge-Kutta llamado método de Runge Kutta de cuarto orden (RK4). En este caso, se inicia tomando un problema de valor inicial dado por $\frac{dy}{dt} = f\big(t, y(t)\big)$ con $y(t_0) = y_0$, definiendo el término y_n como:

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4),$$

en donde se establecen las constantes s, b_i de la ecuación [1] como: $s = 4, b_1 = \frac{1}{6}, b_2 = \frac{2}{6}, b_3 = \frac{2}{6}, b_4 = \frac{1}{6}$. Los términos de aproximación k_i se definen como:

$$\begin{gathered} k_1 = f(t_n, y_n), \\ k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\ k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\ k_4 = f(t_n + h, y_n + hk_3) \end{gathered} \qquad [3]$$

con:

$$t_{n+1} = h + t_n$$

para $n = 0,1,2,\ldots$

Así, en el problema de N-cuerpos, se definieron dos PVI que han de ser resueltos por el método de Runge-Kutta de cuarto orden:

$$\begin{cases} \dfrac{d\vec{r}}{dt} = \vec{v} \\ \vec{r}(t_0) = \overrightarrow{r_0} \end{cases},$$

$$\begin{cases} \dfrac{d\vec{v}}{dt} = \vec{a} \\ \vec{v}(t_0) = \overrightarrow{v_0} \end{cases} \quad \text{con} \quad \vec{a} = \sum_i -\frac{GM_i}{r_i^3}\vec{r_i}$$

Siendo $\vec{r} = \vec{r}(t)$ la posición de uno de los cuerpos en un instante de tiempo t, $\vec{v} = \vec{v}(t)$ siendo la velocidad del mismo cuerpo en el mismo instante t, y siendo $\vec{a} = \vec{a}(t)$ la aceleración de este mismo en el instante t. La aceleración se calculó en base a la sumatoria de fuerzas gravitacionales que siente el cuerpo de estudio, debido a la masa de los demás cuerpos M_i ubicados a una distancia r_i del cuerpo, usando la ley de gravitación de Newton. Siendo $\vec{r_i}$ un vector que va del cuerpo de masa M_i al cuerpo de estudio.

Restringiendo el problema al caso bidimensional y usando notación de índices podemos definir:

$$\vec{r}_{ij} = \vec{r}_j - \vec{r}_i = (x_j - x_i)\hat{e}_x + (y_j - y_i)\hat{e}_y$$

Siendo $\overrightarrow{r_{ij}}$ el vector posición que va del cuerpo M_i al M_j. Con lo cual la ley de gravitación usada para hallar la fuerza neta sobre M_i queda como:

$$M_i\vec{a}_i = \vec{F}_{ji} = \sum_{j=0}^{n-1} -G\frac{M_jM_i}{r_{ji}^3}\vec{r}_{ji} \quad \text{con} \quad i \neq j$$

Para las posiciones en x del cuerpo de masa M_i se obtiene:

$$\begin{cases} \dfrac{dx_i}{dt} = v_{x_i} \\ \dfrac{dv_{x_i}}{dt} = a_{x_i} \quad \text{con} \quad a_{x_i} = \displaystyle\sum_{j=0,\ i\neq j}^{n-1} \frac{GM_j}{r_{ji}^3}\left(x_j - x_i\right) \quad i,j = 0,1,2,\ldots n-1 \\ x_i(t_0) = x_{i0} \\ v_x(t_0) = v_{x_{i0}} \end{cases}$$

Y para las posiciones en y del cuerpo de masa M_i:

$$\begin{cases} \dfrac{dy_i}{dt} = v_{y_i} \\ \dfrac{dv_{y_i}}{dt} = a_{y_i} \quad \text{con} \quad a_{y_i} = \displaystyle\sum_{j=0,\ i\neq j}^{n-1} \frac{GM_j}{r_{ji}^3}\left(y_j - y_i\right) \quad i,j = 0,1,2,\ldots n-1\,. \\ y_i(t_0) = y_{i0} \\ v_y(t_0) = v_{y_{i0}} \end{cases}$$

Así, a cada componente $x_i, y_i, v_{x_i}, v_{y_i}$ se le puede aplicar el algoritmo RK4, con el cual se pueden actualizar las posiciones y velocidades en cada instante de tiempo t_n.

Pseudocódigo y análisis de complejidad

Para el análisis de complejidad del algoritmo se tuvo en cuenta la cantidad de operaciones del procedimiento, en esta se consideraron operaciones como asignaciones, hallar componentes de un vector (locate), operaciones básicas matemáticas elementales, condicionales y ciclos for. Para el total de operaciones de todo el código se hizo la suma de las operaciones de todos los procedimientos y las funciones no se tomaron en cuenta en el momento de ser declaradas, pero sí se tomaron en cuenta cuando se aplicaron sus ejecuciones dentro del código. Aquello resaltado dentro del código corresponde a la complejidad algorítmica.

```
Inicio AnimarMovimiento
    //Definición de parametros y valores iniciales del código//

    //Valores inciales de posición y velocidad en listas de números reales//
    x<-(x1, x2, ..., xn)   n
    y<-(y1, y2, ..., yn)    n
    vx<-(vx1, vx2, ..., vxn)   n
    vy<-(vy1, vy2, ..., vyn)   n

    //lista con masas de los cuerpos (n en total) con números reales//
    m<-(m1, m2, ..., mn)    n

    //Parametros y constantes//
    t0<-0      1   //tiempo inicial//
    h<-10000   1   //paso de tiempo. Modificar según se necesite//
    N<-20000     1 //cantidad de pasos. Aumentar para añadir tiempo de animación//
    G<-6.67e-11  1 //constante gravitacional//
    n<-Tamaño de m 1//indica la cantida de objetos estelares//

    //listas que contendrán los datos de posición y velocidad//
    X<-Lista_de_ceros(N) N
    Y<-Lista_de_ceros(N) N
    Vx<-Lista_de_ceros(N) N
    Vy<-Lista_de_ceros(N) N

    //Inicializando las listas de posición y velocidad//
    X[0]<-x n
    Y[0]<-y n
    Vx[0]<-vx n
    Vy[0]<-vy n

    //Definición de funciones importantes para el código//

    Función aceleración (i,j,x,y)
        //Función auxiliar que halla la aceleración gravitacional del cuerpo j-ésimo
          debido al cuerpo i-ésimo//
        // i,j son números enteros que indican el subíndice de cada cuerpo
        // x,y son vectores de tamaño n, que contienen las posiciones de los n-cuerpos //
        rij<- (x[i]-x[j])^2 + (y[i]-y[j])^2//distancia al cuadrado/1+(3+2)+(3+2)+1=12
        ax<- -G*m[i]*(x[j]-x[i])/rij^(3/2) 1asignación, 3locate, 1resta, 2multiplicac
           1   1  1 1   1 1  1    1    2   1división + 2 = 10

        ay<- -G*m[i]*(y[j]-y[i])/rij^(3/2)     10
        Devolver (ax, ay) //aceleraciones en x e y     2
    Fin Función
    Total ops función aceleración = 34

    Función Fk (k, x, y, vx, vy)
        //Esta función evalúa la superposición de fuerzas para hallar la aceleración
```

```
          neta, hace las veces de f(t, y(t))//
        //k es el número entero que indica el subíndice del cuerpo al que se le halla
          la superposición de fuerzas//
        //x,y,vx,vy son listas de números con las posiciones y velocidades de los n
          Cuerpos//
        akx<-0    1 //inicializa aceleraciones en X y Y del cuerpo k-ésimo en cero//
        aky<-0    1
        Para i<-0 Hasta n-1 Con Paso 1 Hacer //itera sobre los n cuerpos//
        1asignación , n comparaciones, n-1 incrementos = 2n

            Si i Distinto De k Entonces//evita calcular sobre el cuerpo de estudio//n
                aki<-aceleracion(i,k,x,y)  //halla aceleración debido al cuerpo i//
                      n+34n

                akx<- akx+aki[0] n+n+n = 3n
                aky<- aky+aki[1] 3n

        Devolver (vx[k],vy[k],akx,aky)//velocidad y aceleración del cuerpo k-ésimo 4
    Fin Función
    Total ops función Fk = 1+1+2n+n+34n+3n+3n+4 = 43n+6

    Función siguiente_valor (x, y, vx, vy)
        //Esta función halla los siguientes valores de posición y Velocidad
          ejecutando el algoritmo de Runge-Kutta Cuarto Orden//
        // x,y,vx,vy son listas de números con las posiciones y velocidades de los n
          Cuerpos//

        k1<- Lista_de_ceros(n) n  //Lista de listas, que contendrá las listas de
                                   velocidad y aceleración de los n cuerpos//
        Para k<-0 Hasta n-1 Con Paso 1 Hacer  2n
            k1[k]<- h*Fk(k,x,y,vx,vy) n + n + n + n(43n+6) = 43n^2 + 9n
        Fin Para

        x1 <- Lista_de_ceros(n) n   //Listas que contendrán los valores asociados al
                                     término k1 del método RK4//
        y1 <- Lista_de_ceros(n) n
        vx1<- Lista_de_ceros(n) n
        vy1<- Lista_de_ceros(n) n

        Para i<-0 Hasta n-1 Con Paso 1 Hacer 2n
            x1[i] <- (x[i]+k1[i][0]/2)
              n+  n+    (n+n+n+n+n)   =  7n

            y1[i] <- (y[i]+k1[i][1]/2)   7n
            vx1[i]<- (vx[i]+k1[i][2]/2)  7n
            vy1[i]<- (vy[i]+k1[i][3]/2)  7n
        Fin Para

        k2<- Lista_de_ceros(n) n
        Para k<-0 Hasta n-1 Con Paso 1 Hacer  2n
            k2[k]<- h*Fk(k,x1,y1,vx1,vy1)     43n^2+9n
        Fin Para

        x2 <- Lista_de_ceros(n) n   //Listas que contendrán los valores asociados al
                                     término k2 del método RK4//
        y2 <- Lista_de_ceros(n) n
        vx2<- Lista_de_ceros(n) n
        vy2<- Lista_de_ceros(n) n
```

```
        Para i<-0 Hasta n-1 Con Paso 1 Hacer 2n
            x2[i] <- (x[i]+k2[i][0]/2)    7n
            y2[i] <- (y[i]+k2[i][1]/2)    7n
            vx2[i]<- (vx[i]+k2[i][2]/2)   7n
            vy2[i]<- (vy[i]+k2[i][3]/2)   7n
        Fin Para

        k3<- Lista_de_ceros(n)  n
        Para k<-0 Hasta n-1 Con Paso 1 Hacer  2n
            k3[k]<- h*Fk(k,x2,y2,vx2,vy2)   43n^2+9n
        Fin Para

        x3 <- Lista_de_ceros(n) n   //Listas que contendrán los valores asociados al
                                     término k3 del método RK4//
        y3 <- Lista_de_ceros(n) n
        vx3<- Lista_de_ceros(n) n
        vy3<- Lista_de_ceros(n) n

        Para i<-0 Hasta n-1 Con Paso 1 Hacer 2n
            x3[i] <- (x[i]+k3[i][0])         6n
            y3[i] <- (y[i]+k3[i][1])         6n
            vx3[i]<- (vx[i]+k3[i][2])        6n
            vy3[i]<- (vy[i]+k3[i][3])        6n
        Fin Para

        k4<- Lista_de_ceros(n)  n
        Para k<-0 Hasta n-1 Con Paso 1 Hacer 2n
            k4[k]<- h*Fk(k,x3,y3,vx3,vy3)    43n^2+9n
        Fin Para

        xf <- Lista_de_ceros(n)  n  //Listas que contendrán los valores del siguiente
                                     intervalo//
        yf <- Lista_de_ceros(n)  n
        vxf<- Lista_de_ceros(n)  n
        vyf<- Lista_de_ceros(n)  n

        Para i<-0 Hasta n-1 Con Paso 1 Hacer 2n
            xf[i]  <-  x[i] + (k1[i][0]+2*k2[i][0]+2*k3[i][0]+k4[i][0])/6
              n     n   n   n     2n     n  2n   2n n   2n   2n n     2n     n  = 20n

            yf[i]  <-  y[i] + (k1[i][1]+2*k2[i][1]+2*k3[i][1]+k4[i][1])/6    20n
            vxf[i] <- vx[i] + (k1[i][2]+2*k2[i][2]+2*k3[i][2]+k4[i][2])/6    20n
            vyf[i] <- vy[i] + (k1[i][3]+2*k2[i][3]+2*k3[i][3]+k4[i][3])/6    20n
            Devolver (xf, yf, vxf, vyf)  4n
        Fin Para
    Fin Función

    Total ops función siguiente_valor = 172n^2 + 236n

    //Se hallan los datos de posición, velocidad y aceleración para todos los N
    intervalos de este código//
    sig <- Lista_de_ceros(n) n
    Para t<-0 Hasta N-1 Con Paso 1 Hacer 2N
        sig  <-siguiente_valor(X[t],Y[t],Vx[t],Vy[t])
             N                 N(172n^2 + 236n) = N + 172Nn^2 + 236Nn

        X[t] <-sig[0]
```

```
          N    N    N   = 3N
        Y[t] <-sig[1]   3N
        Vx[t]<-sig[2]   3N
        Vy[t]<-sig[3]   3N


    //Se realiza la parte de la animación de las trayectorias//
    coordenas <- “centradas”  1
    //esta variable puede tomar el valor de “centradas” o el de “no centradas”. Centradas
    quiere decir que el origen de coordenadas de la animación estará centrado sobre el
    cuerpo 0 (Observador en movimiento). Y “no centradas”, que el origen estará en el
    punto (0,0) (Observador fijo)//

    Función actualizar (i)
        //esta función permite graficar la animación del frame i//
        //i es un número entero que indica el índice del frame
        Limpiar Grafica Pasada 1

        //Se declaran las variables que contendrán las posiciones desde el intervalo
          t_0 hasta el intervalo t_i//
        x<- Lista_de_ceros(i) i
        y<- Lista_de_ceros(i) i

        Si coordenadas=“centradas” Entonces
            Para t<-0 Hasta n-1 Con Paso 1 Hacer  //itera sobre cada planeta// 2n
                Para j<-0 Hasta i-1 con Paso 1 Hacer//itera sobre cada frame// n(2i)
                    x[j] <- X[j][t] – X[j][0] //centrado sobre el cuerpo 0//
                     n(i   i    2i    i   2i)  = n(7i)
                    y[j] <- Y[j][t] – Y[j][0]    n(7i)
                Fin Para
             Fin Para
        Si no Entonces
            Para t<-0 Hasta n-1 Con Paso 1 Hacer  //itera sobre cada planeta// 2n
                Para j<-0 Hasta i-1 con Paso 1 Hacer//itera sobre cada frame// n(2i)
                    x[j]<- X[j][t]  n(i+i+2i) =  n(4i)           //no centrado//
                    y[j] <- Y[j][t]   n(4i)
                Fin Para
             Fin Para
         Fin Si
         Para j<-0 Hasta n-1 con Paso 1 Hacer 2n
            Para k<-0 Hasta i-2 con Paso 1 Hacer    n(1  +  i-2+1   + i-2)  = n(2i-2)
                Graficar Segmento De (x[k], y[k]) Hasta (x[k+1], y[k+1])
                     n(i-1             i-1  i-1             i-1     i-1) = n(5i-5)
            Fin Para
            Graficar Punto (x[i-1],y[i-1])
                n(1      +      1   +  1)   = 3n
         Fin Para
    Fin Función
    Total ops función actualizar = 1 + 2i + 2n + 33ni

    //Funciones auxiliares para crear los límite en X y Y de la gráfica//
    Función máximo (x)
        //x es un vector de números//
        max<-x[0]  2
        Para i<-0 Hasta n-1 Con Paso 1 Hacer  2n
            Si x[i]>max Entonces
            n   n   n  = 2n
                max<-x   n
```

```
            Fin Si

        Devolver max  1
    Fin Función
    Total ops función máximo = 5n+3

    Función mínimo (x)
        //x es un vector de números//
        min<-x[0]  2
        Para i<-0 Hasta n-1 Con Paso 1 Hacer  2n
            Si x[i]<min Entonces  2n
                min<-x            n
            Fin Si

        Devolver min 1
    Fin Función
    Total ops función mínimo = 5n+3


    Crear gráfica vacía 1
    Fijar Límites Eje X (mínimo(x), máximo(x)) 1+(5n+3)*2
    Fijar Límites Eje Y (mínimo(y), máximo(y)) 1+(5n+3)*2
    Poner Etiqueta Eje X "Eje X" 1
    Poner Etiqueta Eje Y "Eje Y" 1

    Para i-<2 Hasta N Hacer
        actualizar(i) N-1 + (2+33n)(N(N+1)/2-1) + 2n(N-1)

Fin
Total de operaciones del código:(2N^2 + 344Nn^2 + 509Nn + 42N - 30n + 33nN^2 + 40)/2
```

$$O\left(\frac{1}{2}(2N^2 + 344Nn^2 + 509Nn + 42N - 30n + 33N^2n + 40)\right)$$

Diagramas de flujo

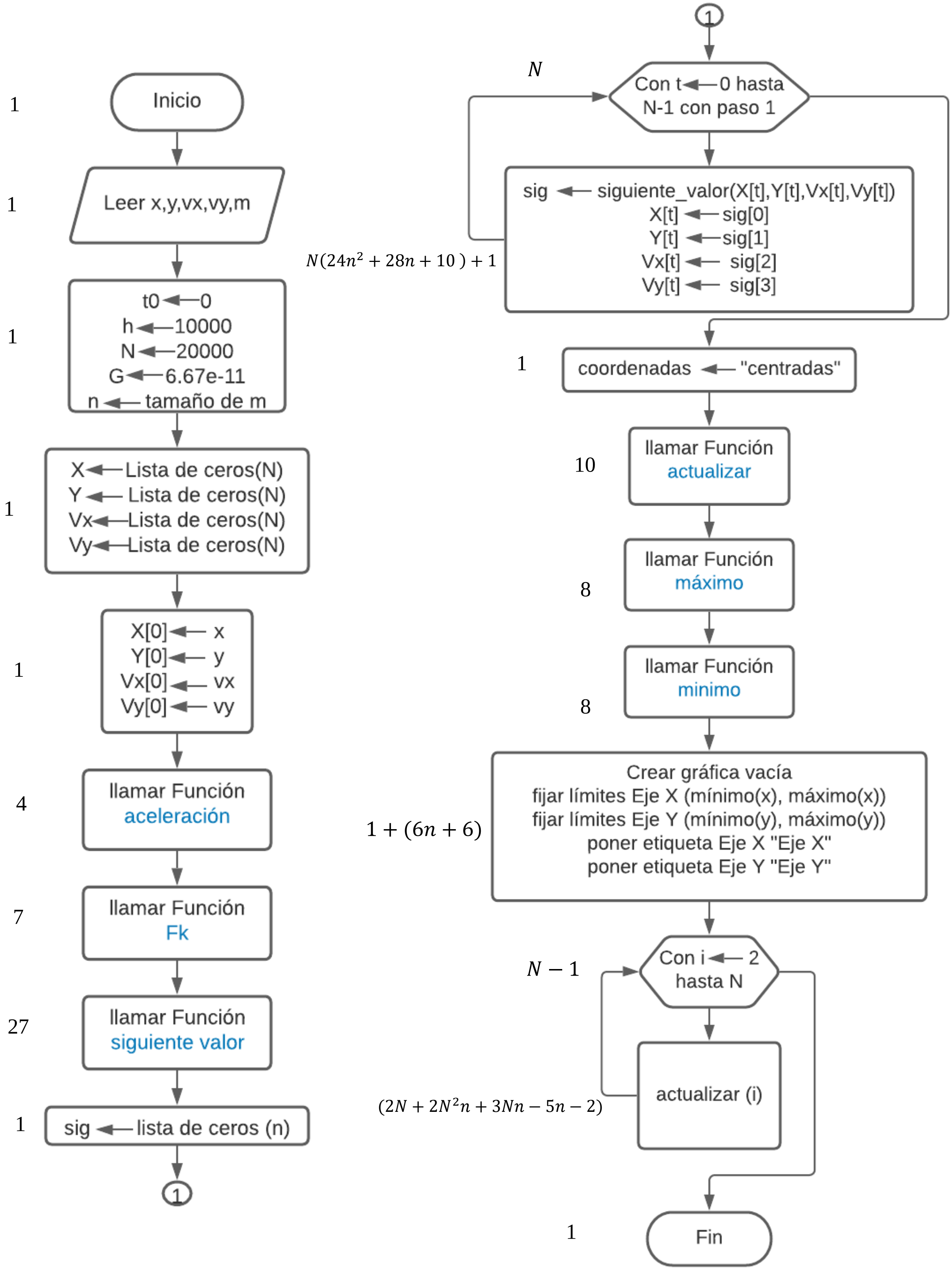
Con el fin de ilustrar de una forma más organizada el pseudocódigo presentado en este trabajo, se realizaron diferentes diagramas de flujo que se pueden encontrar en el siguiente drive[1], los cuales corresponden al desarrollo que presenta cada función a lo largo del código, estas son: función aceleración, función Fk o ecuación de movimiento, función siguiente valor, función actualizar, función máximo y función mínimo. A continuación, se tiene el diagrama de flujo del código en general.

Adicionalmente, se realizó el conteo de los bloques del diagrama de flujo, con el fin de obtener el DSPACE del código. Este es un indicador del espacio que ocupa el código en la memoria del computador, pero, no es un indicador del tiempo de compilación. En este caso, el valor obtenido fue de

$$DSPACE(2N^2n + 24Nn^2 + 31Nn + 14N + n + 76)$$

medido en bloques del diagrama de flujo.

[1] Por medio del siguiente enlace puede acceder a los diagramas de flujo https://drive.google.com/drive/folders/1_uNw-jl6k4LKu6wxAv9cgghN8-huy3mo?usp=sharing



$DSPACE(2N^2n+24Nn^2+31Nn+14N+n+76)$