

PROBLEMA DE LOS N CUERPOS

FÍSICA COMPUTACIONAL

Diego Sarceño

201900109

Guatemala, 28 de noviembre de 2022

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1. Problema de los N Cuerpos

Las primeras ideas acerca del "problema de los n cuerpos" llegaron a comienzos del siglo XVII de la mano de Newton al decir que no es suficiente con especificar la posición inicial y la velocidad, o tampoco tres posiciones orbitales, sino que, para determinar con certeza la órbita de un planeta es necesario tomar en cuenta las fuerzas gravitatorias interactivas. Newton no lo plantea directamente, pero en sus "Principia" se deduce que el problema de los n es irresoluble.

Ya se ha visto en otros cursos (Mecánica Clásica 1 y 2) que simplificaciones de este problema son posibles de resolver o aproximar analíticamente. Estas son: El problema de los 2 cuerpos y el problema de los 3 cuerpos restringido. En este caso se tomarán todas las masas involucradas iguales ($10^{18}kg$), además de restringir el movimiento de las mismas al plano xy .

2. Implementación

2.1. Posiciones Iniciales

Para las posiciones iniciales se utilizó una distribución uniforme de números en el rango de $[-1, 1]$, en unidades astronómicas ($1UA = 1.5 \times 10^{11}m$). Para esto se utilizaron librerías como `<cstdlib>` y `<random>`; así como una semilla (seed) para que las pruebas realizadas se puedan replicar la simulación. Con esto se generan los arreglos para las coordenadas x e y .

2.2. Velocidades Iniciales

Dado que la distribución usada para colocar las partículas en la región dada es uniforme, entre más partículas se coloquen en dicha distribución, mejor será la aproximación a una distribución homogénea de masa. Teniendo esto, se supone una trayectoria circular inicial, entonces igualando la fuerza centrípeta y gravitacional, se tiene

$$\frac{m_i v_i^2}{r_i} = \frac{GMm_i}{r_i^2}, \quad r_i = \sqrt{x_{i0}^2 + y_{i0}^2}, \quad (1)$$

con M es la masa contenida en dicho círculo inicial, esta se puede calcular utilizando la densidad superficial de masa Nm_i/L^2 . Con esto se obtiene que

$$v_i = \frac{\sqrt{G\pi Nm_i r_i}}{L}.$$

Además, es necesario añadirle la dirección aleatoria la cuál está dada por un vector unitario generado por la posición de la partícula

$$\mathbf{v}_{i0} = \frac{\sqrt{G\pi Nm_i r_i}}{L} \left(-\frac{y_{i0}}{r_i}, \frac{x_{i0}}{r_i} \right). \quad (2)$$

Aparte de esto, a cada partícula se le añade un valor adicional en cada dirección.

2.3. Ecuaciones de Movimiento

Para cada partícula se tiene una fuerza neta actuando sobre ella, con la ventaja de ser un sistema aislado. Cada una de estas fuerzas incluye la interacción del resto de partículas, por ende

$$\vec{F}_i = \sum_{i \neq j} -Gm_i m_j \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3} \quad (3)$$

con $|\vec{r}_i - \vec{r}_j|^2 = (x_i - x_j)^2 + (y_i - y_j)^2$. Entonces, por segunda ley de Newton se tienen las siguientes ecuaciones de movimiento

$$\begin{aligned} \ddot{x}_i &= \sum_{i \neq j} -\frac{Gm_j}{|\vec{r}_i - \vec{r}_j|^3} (x_i - x_j), \\ \ddot{y}_i &= \sum_{i \neq j} -\frac{Gm_j}{|\vec{r}_i - \vec{r}_j|^3} (y_i - y_j). \end{aligned}$$

2.4. Energía y Momentum

Utilizando las definiciones clásicas para muchas partículas

$$\begin{aligned} E &= \frac{1}{2} \sum_i^N m_i v_i^2 - G \sum_{i < j} \frac{m_i m_j}{r_{ij}} \\ P &= \sum_i^N m_i v_i. \end{aligned}$$

2.5. Escritura de los Archivos

La parte más importante antes de las gráficas, los datos devueltos por el programa. A lo largo del desarrollo del proyecto, esta fue una de las partes en las que se presentaron más problemas, recibí ayuda para poder tener un buen output a mi programa. Se utilizó la clase *StringStream* para ello.

3. Resultados

El código compiló bien; sin embargo, no dio resultados satisfactorios. Las hojas de datos mostradas por el programa daban resultados inmanejables y/o sin sentido físico. Obviamente es un error en alguna de las funciones, pero por mala planificación del tiempo por mi parte no pude encontrar dicho error.

4. Anexos

4.1. Código

4.1.1. Prueba para 3 Cuerpos

```
1 // Librerias
2 #include <iostream>
3 #include <cmath>
4 #include <iomanip>
5 #include <fstream>
6 #include <cstdlib>
7 #include <random>
8
9
10 using namespace std;
11
12
13 std::default_random_engine generator(17); // seed
14 std::uniform_real_distribution<long double> random_interval(-1.0,1.0);
15
16
17
18
19
20
21
22 /*
23 // constantes globales
24 #define n_cuerpos (100)
25 #define n_ec (n_cuerpos*4)
26 #define G (6.67e-11)
27 */
28
29 void RK4( double *y,
30           const int n_ec,
31           const double t,
32           const double h,
33           double *y_imas1,
34           void (*derivada)( double *, const double, double * ) );
35
36
```

```

37 const int n_cuerpos = 3;
38 const int n_ec = 4*n_cuerpos;
39 const double G = 6.67e-11;
40 // Variables globales
41 double *xp, *yp; // posicion en x, y
42 double *vx, *vy; // velocidad en x, y
43 double ax[n_cuerpos], ay[n_cuerpos]; // aceleracion en x, y
44 double masa[n_cuerpos]; // masa de cada particula
45 double E, P, L;
46
47
48
49
50
51
52 // Inicializar masas
53 void init_masa(){
54     masa[0] = 1.989e30;
55     masa[1] = 5.972e24;
56     masa[2] = 7.348e22;
57 } // END masa
58
59 // Inicializar posiciones
60 void init_posicion(){
61     xp[0] = 0.0;
62     xp[1] = 1.5096e11;
63     xp[2] = 1.5134e11;
64     yp[0] = 0.0;
65     yp[1] = 0.0;
66     yp[2] = 0.0;
67 } // END init_posicion
68
69 // Inicializar velocidades
70 void init_velocidad(){
71     vx[0] = 0.0;
72     vx[1] = 0.0;
73     vx[2] = 0.0;
74     vy[0] = 0.0;
75     vy[1] = 2.979e4;
76     vy[2] = 3.080e4;
77 } // END init_velocidad
78
79
80 void calc_aceleracion(){
81     for (int i = 0; i < n_cuerpos; i++){
82         ax[i] = 0;
83         ay[i] = 0;
84         for (int j = 0; j < n_cuerpos; j++){

```

```

85         double deltaX = xp[i] - xp[j];
86         double deltaY = yp[i] - yp[j];
87         double common_term = G*masa[j]*pow(pow(deltaX,
88             2) + pow(deltaY, 2), -3/2);
89         ax[j] -= common_term*deltaX;
90         ay[i] -= common_term*deltaY;
91     } // END FOR
92 } // END calc_aceleracion
93
94
95
96
97 // Derivadas del sistemas de ecuaciones
98 void nCuerposGrav(double y[n_ec], const double t, double dydt[n_ec]){
99     xp    = y;
100    yp    = y + n_cuerpos;
101    vx    = y + 2*n_cuerpos;
102    vy    = y + 3*n_cuerpos;
103
104    // Calcular aceleraciones
105    calc_aceleracion();
106
107    for( int i=0; i<n_cuerpos; i++ ){
108        dydt[i] = vx[i];
109        dydt[i + n_cuerpos] = vy[i];
110        dydt[i + 2*n_cuerpos] = ax[i];
111        dydt[i + 3*n_cuerpos] = ay[i];
112    } // END FOR
113 } // END nCuerposGrav
114
115
116 void energia_momentos(){
117     E = 0.0;
118     L = 0.0;
119     P = 0.0;
120     double energia_potencial = 0.0;
121     double energia_cinetica = 0.0;
122     for (int i = 0; i < n_cuerpos; i++){
123         P += masa[i]*sqrt(vx[i]*vx[i] + vy[i]*vy[i]);
124         energia_cinetica += 0.5*masa[i]*(vx[i]*vx[i] + vy[i]*vy[
125             i]);
126         for (int j = 0; j < i; j++){
127             energia_potencial += G*masa[i]*masa[j]/sqrt(pow(
128                 xp[i] - xp[j],2) + pow(yp[i] - yp[j],2));
129         } // END FOR
130     } // END FOR
131     E = energia_cinetica - energia_potencial;

```

```

130 } // END energia_momentos
131
132
133
134 void archivo(ofstream &of, stringstream &ss){
135     of << ss.str();
136 }
137
138 void salidaSolucion(const double &t, stringstream &ss){
139     ss << t;
140     for (int i = 0; i < n_cuerpos; i++){
141         ss << "\t" << xp[i] << "\t" << yp[i];
142     } // END FOR
143     ss << endl;
144 } // END salidaSolucion
145
146
147 void salidaVelocidad(const double &t, stringstream &ss){
148     ss << t;
149     for (int i = 0; i < n_cuerpos; i++){
150         ss << "\t" << xp[i] << "\t" << yp[i];
151     } // END FOR
152     ss << endl;
153 } // END salidaVelocidad
154
155
156
157
158
159 void salidaEnergiaMomento(const double &t, stringstream &ss){
160     ss << t << "\t" << E << "\t" << P << endl;
161 } // END salidaEnergia
162
163
164
165 /*
166 void salidaMomentumLineal(const double tiempo, ofstream &of){
167
168 } // END salidaMomentumLineal
169
170
171
172
173 void salidaMomentumAngular(const double tiempo, ofstream &of){
174
175 } // END salidaMomentumAngular
176 */
177

```

```

178
179
180 void salidaMasa(const double &t, stringstream &ss){
181     ss << t;
182     for (int i = 0; i < n_cuerpos; i++){
183         ss << "\t" << masa[i];
184     } // END FOR
185     ss << endl;
186 } // END salidaMasa
187
188
189
190 void verificar_colisiones( double t )
191 {
192     double dist_lim = 1.0e7;
193
194     for( int i=0; i<n_cuerpos; i++ ){
195         if ( masa[i] != 0.0 ){
196             for( int j=0; j<i; j++ ){
197                 if ( masa[j] != 0.0 ){
198                     double dx = xp[i]-xp[j];
199                     double dy = yp[i]-yp[j];
200                     double distancia = sqrt( dx*dx + dy*dy );
201                     if ( distancia < dist_lim ){
202                         vx[i] = (masa[i]*vx[i] + masa[j]*vx[j] ) / (masa[i]+masa[j])
203                         ;
204                         vy[i] = (masa[i]*vy[i] + masa[j]*vy[j] ) / (masa[i]+masa[j])
205                         ;
206                         masa[i] = masa[i] + masa[j];
207
208                         // partícula j sigue la misma trayectoria que partícula i
209                         pero sin masa
210                         vx[j] = 0.0;
211                         vy[j] = 0.0;
212                         masa[j] = 0.0;
213                         cout << "Colision_" << i << "_" << j << " en t=" << t << endl
214                         ;
215                     }
216                 }
217             }
218         }
219     }
220 }
221
222 void RK4( double *y,
223     const int n_ec,

```

```

222         const double t,
223         const double h,
224         double *y_imas1,
225         void (*derivada)( double *, const double, double * ) )
226 {
227     double *k0 = new double[ n_ec ];
228     double *k1 = new double[ n_ec ];
229     double *k2 = new double[ n_ec ];
230     double *k3 = new double[ n_ec ];
231     double *z  = new double[ n_ec ];
232
233     (*derivada)( y, t, k0 );
234
235     for( int i=0; i<n_ec; i++ )
236         z[i] = y[i] + 0.5*k0[i]*h;
237
238
239     (*derivada)( z, t+0.5*h, k1 );
240
241     for( int i=0; i<n_ec; i++ )
242         z[i] = y[i] + 0.5*k1[i]*h;
243
244     (*derivada)( z, t+0.5*h, k2 );
245
246     for( int i=0; i<n_ec; i++ )
247         z[i] = y[i] + k2[i]*h;
248
249     (*derivada)( z, t+h, k3 );
250
251     for( int i=0; i<n_ec; i++ )
252         y_imas1[i] = y[i] + h/6.0 * ( k0[i] + 2*k1[i] + 2*k2[i] + k3[i] );
253
254     delete[] k0;
255     delete[] k1;
256     delete[] k2;
257     delete[] k3;
258     delete[] z;
259 }
260
261
262
263
264 int main()
265 {
266     // Parametros
267     // de 5 dias en 5 dias
268     const int Niter = 5000; // numero de iteraciones
269     const double h_step = 1; // tamaño de paso

```



```

270  const int out_cada = 10; // output cada out_cada iteraciones
271
272  double tiempo = 0.0;
273
274      /*
275  // Archivos de salida
276  ofstream of_posicion( "posicion.dat", ios::out );
277  ofstream of_velocidad( "velocidad.dat", ios::out );
278  ofstream of_energia( "energia.dat", ios::out );
279  ofstream of_momentumLineal( "momLineal.dat", ios::out );
280  ofstream of_momentumAngular( "momAngular.dat", ios::out );
281  ofstream of_masa( "masa.dat", ios::out );
282      */
283
284  // reservar espacio para y
285  double y[n_ec];
286  double y_nueva[n_ec];
287
288  // reservar espacio para posicion y velocidad
289  xp = y;
290  yp = y + n_cuerpos;
291  vx = y + 2*n_cuerpos;
292  vy = y + 3*n_cuerpos;
293
294
295  // puntero a la funcion "derivada"
296  void (*derivada)( double *, const double, double * );
297  derivada = nCuerposGrav;
298
299
300  // inicializar y_nueva
301  for( int i=0; i<n_ec; i++ ) y_nueva[i] = 0.0;
302
303
304
305  // Inicializar masa
306  init_masa();
307
308  // Inicializar posicion
309  init_posicion();
310
311  // Inicializar velocidad
312  init_velocidad();
313
314
315      // strings
316      std::stringstream ss_posicion;
317      std::stringstream ss_velocidad;

```

```

318     std::stringstream ss_energia;
319     std::stringstream ss_masa;
320
321
322     // ciclo de iteraciones
323     for( int k=0; k<=Niter; k++ ){
324         // Nombres faciles para las variables
325         xp    = y;
326         yp    = y +    n_cuerpos;
327         vx    = y + 2*n_cuerpos;
328         vy    = y + 3*n_cuerpos;
329
330
331         energia_momentos();
332         // Verificar colisiones
333         verificar_colisiones(tiempo);
334
335
336
337         // salida
338         if (k%out_cada == 0){
339             salidaSolucion(tiempo, ss_posicion);
340             salidaVelocidad(tiempo, ss_velocidad);
341             salidaEnergiaMomento(tiempo, ss_energia);
342             salidaMasa(tiempo, ss_masa);
343         }
344         RK4( y, n_ec, tiempo, h_step, y_nueva, derivada );
345
346         // Intercambiar valores
347         for( int i=0; i<n_ec; i++ ) y[i] = y_nueva[i];
348
349
350
351         // Incrementar el tiempo
352         tiempo += h_step;
353     }
354     ofstream of_posicion("posicion.dat", ios::out);
355     ofstream of_velocidad("velocidad.dat", ios::out);
356     ofstream of_energia("energia_momentos.dat", ios::out);
357     ofstream of_masa("masa.dat", ios::out);
358
359     archivo(of_posicion, ss_posicion);
360     archivo(of_velocidad, ss_velocidad);
361     archivo(of_energia, ss_energia);
362     archivo(of_masa, ss_masa);
363     return 0;
364 }

```

4.1.2. Problema de los N Cuerpos

```
1 // Librerias
2 #include <iostream>
3 #include <cmath>
4 #include <iomanip>
5 #include <fstream>
6 #include <cstdlib>
7 #include <random>
8
9
10 using namespace std;
11
12
13 std::default_random_engine generator(17); // seed
14 std::uniform_real_distribution<long double> random_interval(-1.0,1.0);
15
16
17
18
19
20
21
22 /*
23 // constantes globales
24 #define n_cuerpos (100)
25 #define n_ec (n_cuerpos*4)
26 #define G (6.67e-11)
27 */
28
29 void RK4( double *y,
30           const int n_ec,
31           const double t,
32           const double h,
33           double *y_imas1,
34           void (*derivada)( double *, const double, double * ) );
35
36
37 const int n_cuerpos = 100;
38 const int n_ec = 4*n_cuerpos;
39 const double G = 6.67e-11;
40 // Variables globales
41 double *xp, *yp; // posicion en x, y
42 double *vx, *vy; // velocidad en x, y
43 double ax[n_cuerpos], ay[n_cuerpos]; // aceleracion en x, y
44 double masa[n_cuerpos]; // masa de cada particula
45 double E, P, L;
46
```

```

47
48
49
50
51
52 // Inicializar masas
53 void init_masa(){
54     for (int i = 0; i < n_cuerpos; i++){
55         masa[i] = 10e18;
56     } // END FOR
57 } // END masa
58
59 // Inicializar posiciones
60 void init_posicion(){
61     for (int i = 0; i < n_cuerpos; i++){
62         xp[i] = random_interval(generator)*1.5e11;
63         yp[i] = random_interval(generator)*1.5e11;
64     } // END FOR
65 } // END init_posicion
66
67 // Inicializar velocidades
68 void init_velocidad(){
69     for (int i = 0; i < n_cuerpos; i++){
70         double ri = sqrt(pow(xp[i],2) + pow(yp[i],2));
71         double vi = sqrt(G*M_PI*n_cuerpos*masa[i]*ri)/(2*1.5e11)
72         ;
73         vx[i] = vi*(-1.0*(yp[i]/ri) + random_interval(generator)
74         );
75         vx[i] = vi*((xp[i]/ri) + random_interval(generator));
76     } // END FOR
77 } // END init_velocidad
78
79 void calc_aceleracion(){
80     for (int i = 0; i < n_cuerpos; i++){
81         ax[i] = 0;
82         ay[i] = 0;
83         for (int j = 0; j < n_cuerpos; j++){
84             double deltaX = xp[i] - xp[j];
85             double deltaY = yp[i] - yp[j];
86             double common_term = G*masa[j]*pow(pow(deltaX,
87             2) + pow(deltaY, 2), -3/2);
88             ax[j] -= common_term*deltaX;
89             ay[i] -= common_term*deltaY;
90         } // END FOR
91     } // END FOR
92 } // END calc_aceleracion

```

```

92
93
94
95
96 // Derivadas del sistemas de ecuaciones
97 void nCuerposGrav(double y[n_ec], const double t, double dydt[n_ec]){
98     xp    = y;
99     yp    = y +    n_cuerpos;
100    vx    = y + 2*n_cuerpos;
101    vy    = y + 3*n_cuerpos;
102
103    // Calcular aceleraciones
104    calc_aceleracion();
105
106    for( int i=0; i<n_cuerpos; i++ ){
107        dydt[i] = vx[i];
108        dydt[i +    n_cuerpos] = vy[i];
109        dydt[i + 2*n_cuerpos] = ax[i];
110        dydt[i + 3*n_cuerpos] = ay[i];
111    } // END FOR
112 } // END nCuerposGrav
113
114
115 void energia_momentos(){
116     E = 0.0;
117     L = 0.0;
118     P = 0.0;
119     double energia_potencial = 0.0;
120     double energia_cinetica = 0.0;
121     for (int i = 0; i < n_cuerpos; i++){
122         P += masa[i]*sqrt(vx[i]*vx[i] + vy[i]*vy[i]);
123         energia_cinetica += 0.5*masa[i]*(vx[i]*vx[i] + vy[i]*vy[
124             i]);
125         for (int j = 0; j < i; j++){
126             energia_potencial += G*masa[i]*masa[j]/sqrt(pow(
127                 xp[i] - xp[j],2) + pow(yp[i] - yp[j],2));
128         } // END FOR
129     } // END FOR
130     E = energia_cinetica - energia_potencial;
131 } // END energia_momentos
132
133 void archivo(ofstream &of, stringstream &ss){
134     of << ss.str();
135 }
136
137 void salidaSolucion(const double &t, stringstream &ss){

```

```

138         ss << t;
139         for (int i = 0; i < n_cuerpos; i++){
140             ss << "\t" << xp[i] << "\t" << yp[i];
141         } // END FOR
142         ss << endl;
143     } // END salidaSolucion
144
145
146 void salidaVelocidad(const double &t, stringstream &ss){
147     ss << t;
148     for (int i = 0; i < n_cuerpos; i++){
149         ss << "\t" << xp[i] << "\t" << yp[i];
150     } // END FOR
151     ss << endl;
152 } // END salidaVelocidad
153
154
155
156
157
158 void salidaEnergiaMomento(const double &t, stringstream &ss){
159     ss << t << "\t" << E << "\t" << P << endl;
160 } // END salidaEnergia
161
162
163
164 /*
165 void salidaMomentumLineal(const double tiempo, ofstream &of){
166
167 } // END salidaMomentumLineal
168
169
170
171
172 void salidaMomentumAngular(const double tiempo, ofstream &of){
173
174 } // END salidaMomentumAngular
175 */
176
177
178
179 void salidaMasa(const double &t, stringstream &ss){
180     ss << t;
181     for (int i = 0; i < n_cuerpos; i++){
182         ss << "\t" << masa[i];
183     } // END FOR
184     ss << endl;
185 } // END salidaMasa

```

```

186
187
188
189 void verificar_colisiones( double t )
190 {
191     double dist_lim = 1.0e7;
192
193     for( int i=0; i<n_cuerpos; i++ ){
194         if ( masa[i] != 0.0 ){
195             for( int j=0; j<i; j++ ){
196                 if ( masa[j] != 0.0 ){
197                     double dx = xp[i]-xp[j];
198                     double dy = yp[i]-yp[j];
199                     double distancia = sqrt( dx*dx + dy*dy );
200                     if ( distancia < dist_lim ){
201                         vx[i] = (masa[i]*vx[i] + masa[j]*vx[j] ) / (masa[i]+masa[j])
202                         ;
203                         vy[i] = (masa[i]*vy[i] + masa[j]*vy[j] ) / (masa[i]+masa[j])
204                         ;
205                         masa[i] = masa[i] + masa[j];
206
207                         // partícula j sigue la misma trayectoria que partícula i
208                         pero sin masa
209                         vx[j] = 0.0;
210                         vy[j] = 0.0;
211                         masa[j] = 0.0;
212                         cout << "Colision_"<< i <<"_"<< j << "_en_t=__" << t << endl
213                         ;
214                     }
215                 }
216             }
217         }
218     }
219
220 void RK4( double *y,
221           const int n_ec,
222           const double t,
223           const double h,
224           double *y_imas1,
225           void (*derivada)( double *, const double, double * ) )
226 {
227     double *k0 = new double[ n_ec ];
228     double *k1 = new double[ n_ec ];
229     double *k2 = new double[ n_ec ];
230     double *k3 = new double[ n_ec ];

```

```

230 double *z = new double[ n_ec ];
231
232 (*derivada)( y, t, k0 );
233
234 for( int i=0; i<n_ec; i++ )
235     z[i] = y[i] + 0.5*k0[i]*h;
236
237
238 (*derivada)( z, t+0.5*h, k1 );
239
240 for( int i=0; i<n_ec; i++ )
241     z[i] = y[i] + 0.5*k1[i]*h;
242
243 (*derivada)( z, t+0.5*h, k2 );
244
245 for( int i=0; i<n_ec; i++ )
246     z[i] = y[i] + k2[i]*h;
247
248 (*derivada)( z, t+h, k3 );
249
250 for( int i=0; i<n_ec; i++ )
251     y_imas1[i] = y[i] + h/6.0 * ( k0[i] + 2*k1[i] + 2*k2[i] + k3[i] );
252
253 delete[] k0;
254 delete[] k1;
255 delete[] k2;
256 delete[] k3;
257 delete[] z;
258 }
259
260
261
262
263 int main()
264 {
265     // Parametros
266     // de 5 dias en 5 dias
267     const int Niter = 73*5000; // numero de iteraciones
268     const double h_step = 432000; // tamano de paso
269     const int out_cada = 10000; // output cada out_cada iteraciones
270
271     double tiempo = 0.0;
272
273     /*
274     // Archivos de salida
275     ofstream of_posicion( "posicion.dat", ios::out );
276     ofstream of_velocidad( "velocidad.dat", ios::out );
277     ofstream of_energia( "energia.dat", ios::out );

```



```

278 ofstream of_momentumLineal( "momLineal.dat", ios::out );
279 ofstream of_momentumAngular( "momAngular.dat", ios::out );
280 ofstream of_masa( "masa.dat", ios::out );
281     */
282
283 // reservar espacio para y
284 double y[n_ec];
285 double y_nueva[n_ec];
286
287 // reservar espacio para posicion y velocidad
288 xp = y;
289 yp = y + n_cuerpos;
290 vx = y + 2*n_cuerpos;
291 vy = y + 3*n_cuerpos;
292
293
294 // puntero a la funcion "derivada"
295 void (*derivada)( double *, const double, double * );
296 derivada = nCuerposGrav;
297
298
299 // inicializar y_nueva
300 for( int i=0; i<n_ec; i++ ) y_nueva[i] = 0.0;
301
302
303
304 // Inicializar masa
305 init_masa();
306
307 // Inicializar posicion
308 init_posicion();
309
310 // Inicializar velocidad
311 init_velocidad();
312
313
314 // strings
315 std::stringstream ss_posicion;
316 std::stringstream ss_velocidad;
317 std::stringstream ss_energia;
318 std::stringstream ss_masa;
319
320
321 // ciclo de iteraciones
322 for( int k=0; k<=Niter; k++ ){
323     // Nombres faciles para las variables
324     xp = y;
325     yp = y + n_cuerpos;

```

```

326     vx    = y + 2*n_cuerpos;
327     vy    = y + 3*n_cuerpos;
328
329
330         energia_momentos();
331     // Verificar colisiones
332     verificar_colisiones(tiempo);
333
334
335         RK4( y, n_ec, tiempo, h_step, y_nueva, derivada );
336     // salida
337     if (k%out_cada == 0){
338         salidaSolucion(tiempo, ss_posicion);
339         salidaVelocidad(tiempo, ss_velocidad);
340         salidaEnergiaMomento(tiempo, ss_energia);
341         salidaMasa(tiempo, ss_masa);
342     }
343
344
345     // Intercambiar valores
346     for( int i=0; i<n_ec; i++ ) y[i] = y_nueva[i];
347
348
349
350     // Incrementar el tiempo
351     tiempo += h_step;
352 }
353     ofstream of_posicion("posicion.dat", ios::out);
354     ofstream of_velocidad("velocidad.dat", ios::out);
355     ofstream of_energia("energia_momentos.dat", ios::out);
356     ofstream of_masa("masa.dat", ios::out);
357
358     archivo(of_posicion, ss_posicion);
359     archivo(of_velocidad, ss_velocidad);
360     archivo(of_energia, ss_energia);
361     archivo(of_masa, ss_masa);
362     return 0;
363 }

```

Referencias

- [1] DeVries, P. L., & Wolf, R. P. (1994). *A first course in computational physics. Computers in Physics*, 8(2), 178-179.
- [2] Symon, K. R. (1971). *Mechanics*. Addison.