Javier de León **201603068**

Problema de los 100 Cuerpos

1. Introducción

El problema de los n cuerpos se refiere de forma usual a la solución del sistema de movimiento de n masas las cuales interactúan entre sí por medio de la fuerza gravitacional en 3D. Este problema no está limitado a esas restricciones y pueden añadirse otros factores o estudiar el mismo comportamiento con otras fuerzas que se comportan de manera similar como la fuerza eléctrica [1].
Este sistema deja de tener solución analítica en el caso general para $n > 2$, ya que la cantidad de grados de libertad es menor a la cantidad de ecuaciones disponibles para resolver el sistema. A pesar de esto existen casos particulares de los cuales se conocen las soluciones analíticas para más dimensiones, en especial $n = 3$. La solución con condiciones arbitrarias de este sistema para los casos con $n > 2$ puede ser encontrada mediante métodos numéricos, aproximaciones o casos reducidos de geometrías particulares.

Nosotros estudiamos el caso particular donde se tienen 100 masas puntuales y simétricas distribuidas aleatoriamente en un cuadrado 2D, afectadas únicamente por la fuerza gravitacional. Las velocidades iniciales se obtienen mediante una simplificación asumiendo una distribución de masa uniforme y se les es agregado un factor aleatorio proporcional a la misma. Además si 2 de estas masas llegasen a acercarse de forma que la distancia entre ellas fuese menor a un threshold dado, estas participarían en una colisión inelástica perfecta.

Para modelar este sistema se realizó una simulación en c++, utilizando el método de RK4 para resolver el sistema de ecuaciones durante 5000 mil años de evolución. Se realizaron modelos con distintos intervalos de aumento en el tiempo y se encontró la evolución de las posiciones, velocidades, aceleraciones, energía, momento y momento angular total del sistema.
Los resultados fueron graficados por medio de gnuplot y se proporciona una animación realizada por medio de ffmpeg.

2. Métodos

2.1. Generalidades:

Para resolver el problema se realizaron dos scripts distintos de simulaciones computacionales en c++ bajo el estándar stdc++20 y fue compilado utilizando flags de optimización como Ofast, se realizaron pruebas para comprobar que no ocurrieran problemas de precisión numérica. Estas simulaciones realizan la aproximación de la solución del sistema de movimiento empleando el método de RK4, este funciona tomando las condiciones iniciales dadas por nuestra solución particular y obtiene las variables físicas en el siguiente intervalo de tiempo. Este algoritmo se repite tomando como los valores obtenidos las nuevas condiciones iniciales y se realiza hasta que se llegue al tiempo de duración deseado.

2.2. Condiciones iniciales:

Para obtener las posiciones iniciales se utilizó el generador de números pseudo aleatorios por defecto de la librería random de c++ y se distribuyeron las masas en un cuadrado centrado en el origen de lado 2UA.

Para obtener las velocidades iniciales se utilizó la expresión obtenida en [3].

$$r_i = \sqrt{x_i^2 + y_i^2}, \tag{1}$$

$$v_{i0} = \frac{\sqrt{G\pi N m_i r_i}}{L}(-\frac{y_{i0}}{r_i}, \frac{x_{i0}}{r_i}), \tag{2}$$

Además se agregó un término aleatorio que oscila entre $\pm\frac{\sqrt{G\pi N m_i r_i}}{L}$.

Para encontrar la aceleraciones iniciales se despejó el sistema obtenido por ley de Newton y se llegó resultado como fue dado en [3] .

$$|r_i - r_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{3}$$

$$\ddot{x_i} = \sum_{i \neq j} -\frac{G m_j}{|r_i - r_j|^3}(x_i - x_j), \tag{4}$$

$$\ddot{y_i} = \sum_{i \neq j} -\frac{G m_j}{|r_i - r_j|^3}(y_i - y_j), \tag{5}$$

Finalmente se coloca el valor de todas las masas a $m_i = 10^{18} kg$. Es importante agregar que nuestra simulación se generalizó para aceptar valores distintos de masas puntuales.

2.3. Cálculo de las variables físicas:

Para encontrar el momento total se utilizó la ecuación del momento para un sistema de partículas, la derivación detallada de estas ecuaciones pueden ser consultadas en [2].

$$P = \sum m_i v_i, \tag{6}$$

Para encontrar la energía total se utiliza la ecuación de la energía para un sistema de partículas:

$$E = \frac{P^2}{2M} + \sum \frac{m_i v_i^2}{2} + V(r_i) + \sum_{i \neq j} V(r_{ij}), \tag{7}$$

Donde M es la masa total y P es el momento del centro de masa del sistema. Ya que nuestra simulación calcula de manera separada el momento del centro de masa, nosotros graficaremos la resta de la energía total con esta, esto quiere decir que nuestro resultado estará en términos de

$$E' = E - \frac{MP^2}{2}, \tag{8}$$

Para encontrar el momento angular total se utilizó la ecuación del momento angular para un sistema de partículas

$$L = R \times P + \sum r_i \times m_i v_i, \tag{9}$$

Donde R es el radio del centro de masa.

Para calcular la evolución de las variables de movimiento utilizamos la implementación del algoritmo de RK4 y repetimos el proceso.

2.4. Cálculo de las colisiones:

Se utiliza un threshold de colisión bastante pequeño comparado con el tamaño de la geometría simulación pero basado físicamente en un radio de planeta ligeramente mayor al radio de la tierra $th = 10^7$.

Por este motivo las colisiones son poco comunes y puede existir un traslape en las animaciones donde parece que ocurre una colisión porque el tamaño animado es más grande que el threshold, o sea su tamaño físico dado. Al ocurrir una colisión se resuelve el sistema de ecuaciones para una colisión elástica perfecta con respecto de las variables físicas finales y se unen los 2 cuerpos sumando sus masas asignándole el valor a una de ellas y descartando la masa de la otra.

2.5. Validez de los modelos:

Para verificar la validez de nuestra implementación se realizó una simulación del sistema de 3 cuerpos Tierra Luna Sol utilizando como condiciones iniciales los datos de la geometría y velocidad conocidas de los mismos. Además se midieron las variables de Energía, Momento y Momento angular con respecto del tiempo y se obtuvo que el comportamiento de las mismas era constante. Ya que esto se encuentra de acuerdo con la teoría, podemos tener cierto grado de seguridad de la fidelidad del mismo. Estas simulaciones corrieron en un periodo de 42 años con un avance en el tiempo de un día.

2.6. Diferencias entre las simulaciones:

La primera implementación se utiliza como base una optimización de el script proporcionado por, este realiza el proceso del método de RK4 directamente en memoria sin protección del compilador y está construido con un paradigma de programación funcional. Esta versión obtiene resultados más precisos, pero conlleva mayor tiempo de ejecución.
La segunda implementación utiliza un enfoque mixto empleando propiedades del OOP e intenta proteger un poco más a posibles errores durante la ejecución del programa. Aún así versión diverge con mayor facilidad a la solución esperada bajo las mismas condiciones, pero tiene un menor tiempo de ejecución.

2.7. Parámetros de la simulación:

- El largo del cuadrado es de 2UA.
- Las masas se configuraron a $m = 10^{18}kg$.
- Las velocidades iniciales están acotadas entre $[-2\frac{\sqrt{G\pi N m_i r_i}}{L}, 2\frac{\sqrt{G\pi N m_i r_i}}{L}]$ y distribuidas de aleatoria.
- El período de duración de la simulación es de 5000 años.
- El avance en el tiempo de la simulación es de un día.
- Se miden las variables del sistema cada 18000 días.

2.8. Parámetros de la gráficas y animación:

- Las gráficas se obtienen por medio de gnuplot.
- Las gráficas de las variables del sistema se imprimen en consola y se guardan en formato png para que sea posible visualizarse en github.
- Las gráficas de la animación se escriben en memoria.
- La animación utiliza 301 frames y los muestra en 12 segundos.

3. Resultados

3.1. Funciones Simulación Script Original

3.1.1. Masas iniciales:

```
// Inicializar masas
void init_masa() {
  // nuestra simulación toma todas las masas con el mismo valor
  for (int i = 0; i < n_cuerpos; i++) {
    masa[i] = 10e18;}}
```

3.1.2. Posiciones iniciales:

```
std::random_device rd;
std::default_random_engine generator(rd()); // rd() random seed
std::uniform_real_distribution<long double> random_menos1_a_1(-1.0, 1.0);
// Inicializar posiciones
void init_posicion() {
// valores aleatorios entre [-1,1]UA en un cuadrado xy.

for (int i = 0; i < n_cuerpos; i++) {
  xp[i] = random_menos1_a_1(generator) * 1.5e11;
  yp[i] = random_menos1_a_1(generator) * 1.5e11;}}
```

3.1.3. Velocidades iniciales:

```
// Inicializar velocidades
void init_velocidad() {
// utilizamos la simplificación para una distribución uniforme de masas
// puntuales a estas se les agrega luego un valor entre [-1,1] la magnitud
// del valor obtenido por la distribución de masas.

for (int i = 0; i < n_cuerpos; i++) {
  long double r = sqrt(pow(xp[i], 2) + pow(yp[i], 2));
  long double r_inverso = pow(r, -1);
  long double vel_compartida =
      sqrt(G * M_PI * n_cuerpos * masa[i] * r) / (3.0e11);

  vx[i] =
      vel_compartida * (-1*yp[i] * r_inverso + random_menos1_a_1(generator));
  vy[i] = vel_compartida * (xp[i] * r_inverso + random_menos1_a_1(generator));}}
```

3.1.4. Cálculo de las aceleraciones:

```
void calc_aceleracion() {
  // consideramos solamente la fuerza de la gravedad.
  for (int i = 0; i < n_cuerpos; i++) {
    ax[i] = 0;
    ay[i] = 0;
    for (int j = 0; j < n_cuerpos; j++) {
      if (i == j)
        continue;
      long double delta_pX = xp[i] - xp[j];
      long double delta_pY = yp[i] - yp[j];
      long double G_masa_abs_ri_menos_rj_cubo =
          masa[j] * pow(pow(delta_pX, 2) + pow(delta_pY, 2), -1.5) * G;
      ax[i] -= delta_pX * G_masa_abs_ri_menos_rj_cubo;
      ay[i] -= delta_pY * G_masa_abs_ri_menos_rj_cubo; }}}
```

3.1.5. Verificación de las Colisiones:

```
//Se agregó animación
void verificar_colisiones(long double t) {
long double dist_lim = 1e8;
for (int i = 0; i < n_cuerpos; i++) {
   if (masa[i] != 0.0) {
     for (int j = 0; j < i; j++) {
       if (masa[j] != 0.0) {
         long double dx = xp[i] - xp[j];
         long double dy = yp[i] - yp[j];
         long double distancia = sqrt(pow(dx, 2) + pow(dy, 2));
         if (distancia < dist_lim) {
           long double nueva_masa = masa[i] + masa[j];
           vx[i] = (masa[i] * vx[i] + masa[j] * vx[j]) / nueva_masa;
           vy[i] = (masa[i] * vy[i] + masa[j] * vy[j]) / nueva_masa;
           masa[i] = nueva_masa;
           // particula j sigue la misma trayectoria que particula i pero sin masa
           xp[j] = (xp[i] + xp[j]) / 2;
           yp[j] = (yp[i] + yp[j]) / 2;
           vx[j] = vx[i];
           vy[j] = vy[i];
           ax[j] = ax[i];
           ay[j] = ay[i];
           masa[j] = 0.0;
           cout << "Colision " << i << " " << j << " en t = " << t << endl;}}}}}}
```

3.1.6. Cálculo de E, P y L totales:

```
void Energia_Pmomento_Lmomemnto() {

    // obtenemos E, P, L de todo el sistema no analizamos las partículas por
    // separado.

    E = 0.0;
    P = 0.0;
    L = 0.0;
    long double V = 0.0;
    long double PCM = 0.0;
    long double mtot = 0.0;
    long double Eloc = 0.0;
    long double RCM = 0.0;
    long double Lloc = 0.0;
    long double V_cruzado = 0.0;
    for (int j = 0; j < n_cuerpos; j++) {

      long double sqrt_x2_mas_y2 = sqrt(pow(xp[j], 2) + pow(yp[j], 2));
      long double vx2_mas_vy2 = (pow(vx[j], 2) + pow(vy[j], 2));
      long double sqrt_vx2_mas_vy2 = sqrt(vx2_mas_vy2);

      V += G * masa[j] / sqrt_x2_mas_y2;
      Eloc += masa[j] * vx2_mas_vy2 * 0.5;

      PCM += masa[j] * sqrt_vx2_mas_vy2;
      mtot += masa[j];
      RCM += masa[j] * sqrt(pow(xp[j], 2) + pow(yp[j], 2));
      Lloc = RCM * sqrt_vx2_mas_vy2;
      vx2_mas_vy2 = 0.0;
      sqrt_vx2_mas_vy2 = 0.0;

      // potencial cruzado
      for (int i = 0; i < j; i++) {
        long double x2_menos_y2 =
            pow((xp[i] - xp[j]), 2) + pow((yp[i] - yp[j]), 2);
        long double sqrt_x2_menos_y2 = sqrt(x2_menos_y2);
        V_cruzado += G * masa[i] * masa[j] / sqrt_x2_menos_y2;
      }
    }
    RCM = RCM / mtot;
    P = PCM;
    E = Eloc - V - V_cruzado;
    L = RCM * PCM + Lloc;}
```

3.1.7. Salida de Archivos:

```
void Energia_Pmomento_Lmomemnto() {
 void escribirArchivo(std::ofstream &of, std::stringstream &ss) {
    of << ss.str();
  }

  void salidaSolucion(const long double &t, std::stringstream &ss) {
    // escribimos en un archivo en la columna 0 el valor de t
    // en las siguientes n_cuerpos px y en las consiguientes py
    ss << t;
    for (int i = 0; i < n_cuerpos; ++i) {
      ss << "\t" << xp[i];
    }
    for (int i = 0; i < n_cuerpos; ++i) {
      ss << "\t" << yp[i];
    }
    ss << std::endl;  }
```

Se ejemplificó la escritura de las variables total del sistema y se omiten el resto de funciones de escritura por su similitud.

3.1.8. Resultados de la Simulación Tierra Luna Sol :

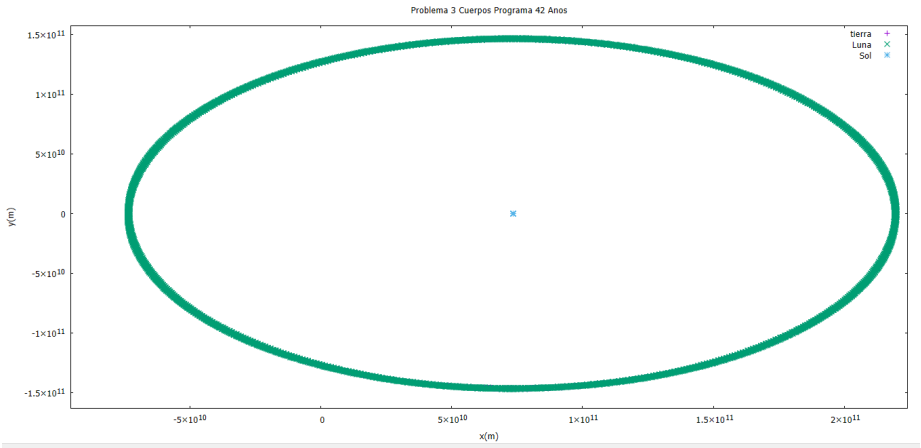

Figura 1: Evolución de La posición Tierra Luna Sol

Debido a la resolución no es posible apreciar la órbita de la tierra con el sol y las sutilezas de la órbita de la luna con la tierra, pero es posible reproducir fácilmente las imágenes utilizando las bases de datos proporcionadas a continuación.

3.2. Resultados de la Simulación de 100 cuerpos:

3.2.1. Momento Total:

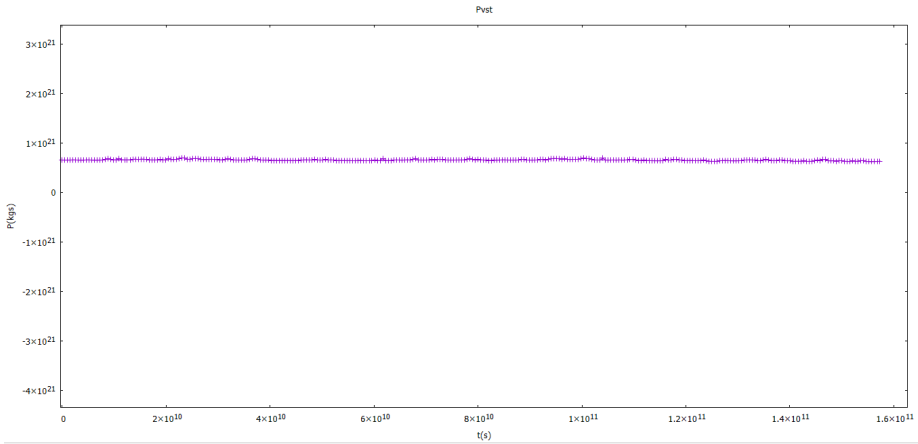

Figura 2: Momento Total 100 Cuerpos

3.2.2. Momento Angular Total:

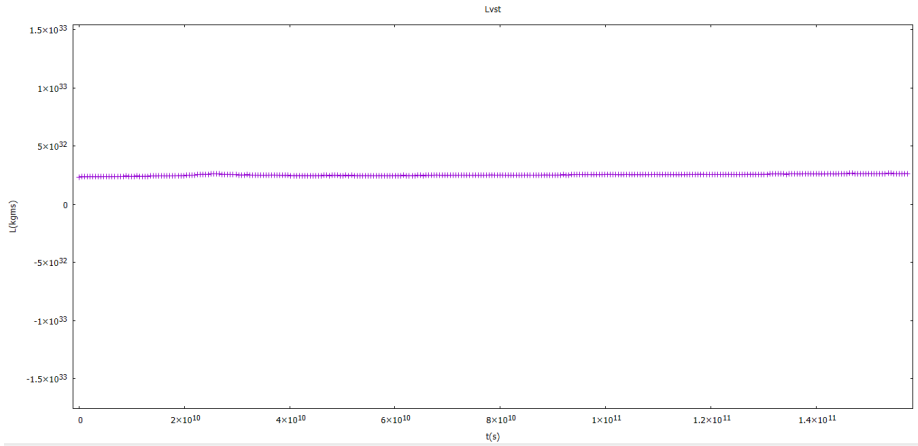

Figura 3: Momento Angular Total 100 Cuerpos

3.2.3. Momento Energía′ Total:

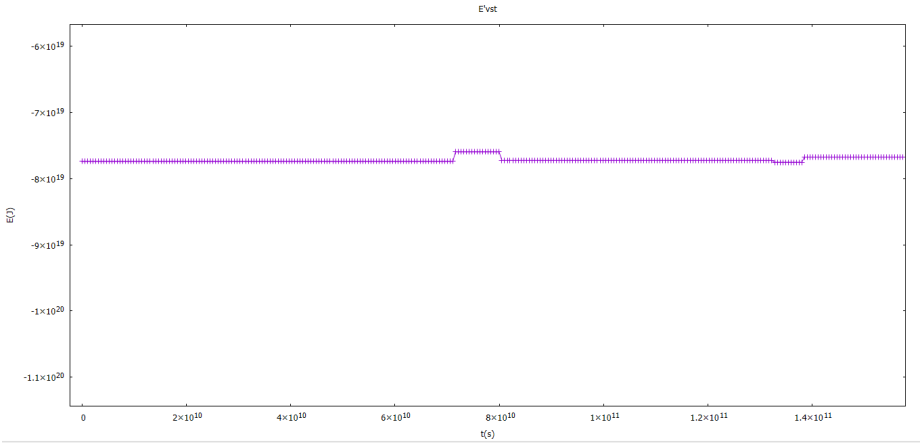

Figura 4: Energía′ Total 100 Cuerpos

3.2.4. Resumen de Movimiento:

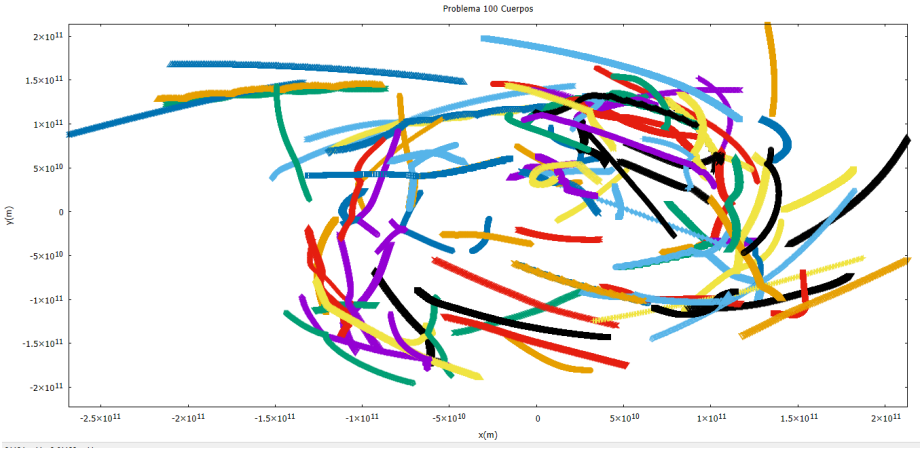

Figura 5: Evolución de La posición 100 Cuerpos

3.2.5. Frame de Animación:

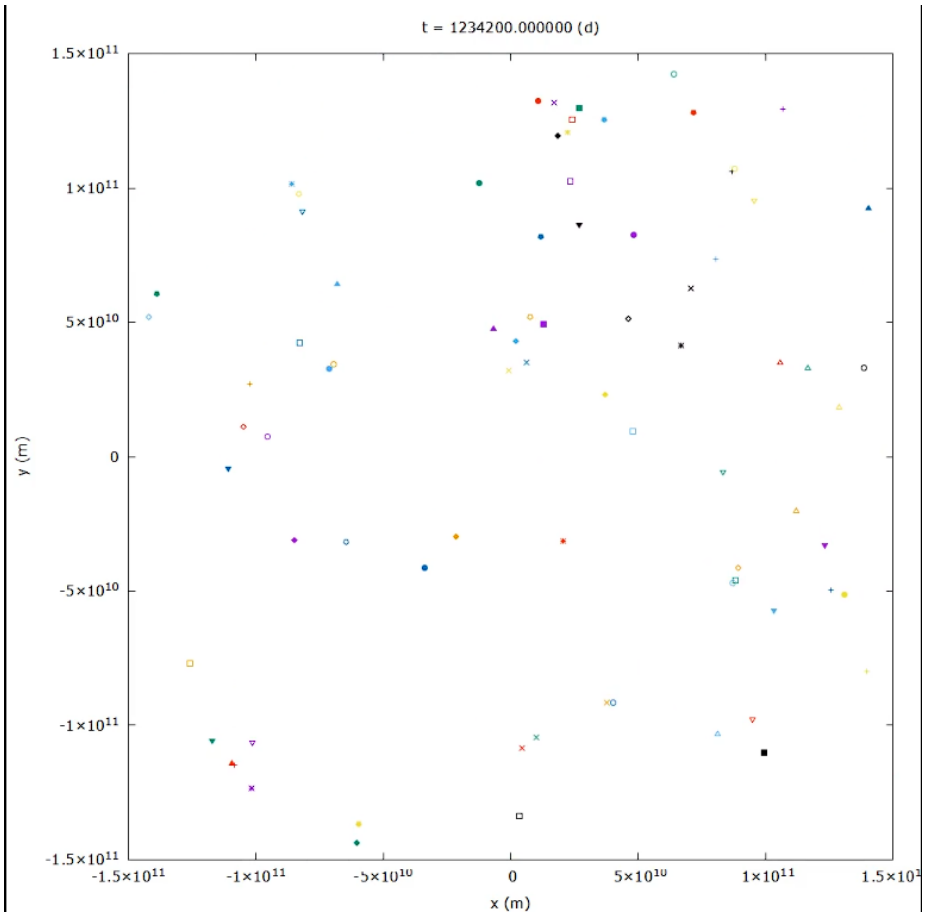

Figura 6: Frame de Animación Simulación 100 Cuerpos

Notemos que las variables físicas del sistema se comportan de manera constante durante el tiempo de evolución del sistema. La única variable que presenta cambios considerables es E′, esto será discutido más adelante.
La animación puede ser consultada en este enlace.

3.3. Funciones Simulación Script Nuevo

3.3.1. Estructura de Cuerpos para la Simulación:

```
 struct cuerpo cuerpos[n_cuerpos];

// Variables del sistema
long double E, P, L; // energia momento lineal y momento angular del sistema

// Inicializar masas
void init_masa() {
  // nuestra simulación toma todas las masas con el mismo valor
  for (int i = 0; i < n_cuerpos; i++) {
    cuerpos[i].masa = 10.0e18L;}}
```

3.3.2. Masas iniciales:

```
 // Inicializar masas
void init_masa() {
  // nuestra simulación toma todas las masas con el mismo valor
  for (int i = 0; i < n_cuerpos; i++) {
    cuerpos[i].masa = 10.0e18L;}}

    //Nuestro programa puede trabajar con
    //un sistema de masas distintas si es deseado
```

3.3.3. Posiciones iniciales:

```
 std::random_device rd;
std::default_random_engine generator(rd()); // rd() random seed
std::uniform_real_distribution<long double> random_menos1_a_1(-1.0, 1.0);


 // Inicializar posiciones
void init_posicion() {
  // valores aleatorios entre [-1,1]UA en un cuadrado xy.

  for (int i = 0; i < n_cuerpos; i++) {
    cuerpos[i].pX = random_menos1_a_1(generator) * 1.5e11;
    cuerpos[i].pY = random_menos1_a_1(generator) * 1.5e11;
  }
}
```

3.3.4. Velocidades iniciales:

```
 // Inicializar velocidades
void init_velocidad() {
  // utilizamos la simplificación para una distribución uniforme de masas
  // puntuales a estas se les agrega luego un valor entre [-1/2,1/2] la magnitud
  // del valor obtenido por la distribución de masas.

  for (int i = 0; i < n_cuerpos; i++) {
    long double r = sqrt(pow(cuerpos[i].pX, 2) + pow(cuerpos[i].pY, 2));
    long double r_inverso = pow(r, -1);
    long double vel_compartida =
        sqrt(G * M_PI * n_cuerpos * cuerpos[i].masa * r) *(3e-11);

    cuerpos[i].vX = -vel_compartida *
                    (cuerpos[i].pY * r_inverso + random_menos1_a_1(generator));
    cuerpos[i].vY = vel_compartida *
                    (cuerpos[i].pX * r_inverso + random_menos1_a_1(generator));
  }
}
```

3.3.5. Cálculo de las aceleraciones:

```
 void calc_aceleracion() {
  // consideramos solamente la fuerza de la gravedad.

  for (int i = 0; i < n_cuerpos; i++) {
    cuerpos[i].aX = 0;
    cuerpos[i].aY = 0;
    for (int j = 0; j < n_cuerpos; j++) {
      if (i == j)
        continue;
      long double delta_pX = cuerpos[i].pX - cuerpos[j].pX;
      long double delta_pY = cuerpos[i].pY - cuerpos[j].pY;
      long double G_masa_abs_ri_menos_rj_cubo =
          cuerpos[j].masa * pow(pow(delta_pX, 2) + pow(delta_pY, 2), -1.5) * G;
      cuerpos[i].aX -= delta_pX * G_masa_abs_ri_menos_rj_cubo;
      cuerpos[i].aY -= delta_pY * G_masa_abs_ri_menos_rj_cubo;
    }
  }
}
```

3.3.6. Verificación de las Colisiones:

```
//Se agregó animación.
void verificar_colisiones(long double &t) {
  for (int i = 0; i < n_cuerpos; i++) {
    if (cuerpos[i].masa != 0.0L) {
      for (int j = 0; j < i; j++) {
        if (cuerpos[j].masa != 0.0L) {
          long double deltaX = cuerpos[i].pX - cuerpos[j].pX;
          long double deltaY = cuerpos[i].pY - cuerpos[j].pY;
          long double distancia = sqrt(pow(deltaX, 2) + pow(deltaY, 2));
          if (distancia < distancia_colision) {
            long double nueva_masa = cuerpos[i].masa + cuerpos[j].masa;
            cuerpos[i].vX = (cuerpos[i].masa * cuerpos[i].vX +
                             cuerpos[j].masa * cuerpos[j].vX) /
                            nueva_masa;
            cuerpos[i].vY = (cuerpos[i].masa * cuerpos[i].vY +
                             cuerpos[j].masa * cuerpos[j].vY) /
                            nueva_masa;
            cuerpos[i].masa = nueva_masa;

          // particula j sigue la misma trayectoria que particula i pero sin
          // masa
          cuerpos[j].pX = (cuerpos[i].pX + cuerpos[j].pX) / 2;
          cuerpos[j].pY = (cuerpos[i].pY + cuerpos[j].pY) / 2;
          cuerpos[j].vX = cuerpos[i].vX;
          cuerpos[j].vY = cuerpos[i].vY;
          cuerpos[j].aX = cuerpos[i].aX;
          cuerpos[j].aY = cuerpos[i].aY;
          cuerpos[j].masa = 0.0L;
          std::cout << "Colision " << i << " " << j << " en t = " << t
                    << std::endl;
        }
      }
    }
  }
    }
}
```

La definición del resto de variables físicas y la escritura de archivos se omiten por su similitud con las funciones presentadas anteriormente.

Además se reescribió el método para calcular las derivadas, aceleración y la ejecución del RK4, se incluye un snippet mostrando las nuevas definiciones de estos métodos el cálculo de K0.

3.3.7. Diferencia entre los Scripts de Simulación:

```
void Energia_Pmomento_Lmomemnto() {


  // Derivadas del sistemas de ecuaciones
  void nCuerposGrav(struct cuerpo (&y)[n_cuerpos],
                    struct cuerpo (&dydt)[n_cuerpos]) {
    for (int i = 0; i < n_cuerpos; i++) {
      cuerpos[i] = y[i];
    }
    // Calcular aceleraciones
    calc_aceleracion();

    for (int i = 0; i < n_cuerpos; i++) {
      dydt[i].pX = cuerpos[i].vX;
      dydt[i].pY = cuerpos[i].vY;
      dydt[i].vX = cuerpos[i].aX;
      dydt[i].vY = cuerpos[i].aY;
    }
  }

  void RK4() {
    struct cuerpo k0[n_cuerpos];
    struct cuerpo k1[n_cuerpos];
    struct cuerpo k2[n_cuerpos];
    struct cuerpo k3[n_cuerpos];

    struct cuerpo z[n_cuerpos];
    for (int i = 0; i < n_cuerpos; i++) {
      z[i] = cuerpos[i];
    }

    nCuerposGrav(cuerpos, k0);

    for (int i = 0; i < n_cuerpos; i++) {
      z[i].pX = cuerpos[i].pX + k0[i].pX * h_step;
      z[i].pY = cuerpos[i].pY + k0[i].pY * h_step;
      z[i].vX = cuerpos[i].vX + k0[i].vX * h_step;
      z[i].vY = cuerpos[i].vY + k0[i].vY * h_step;
    }
```

El código fuente puede ser visto aquí.

3.3.8. Resultados de la Simulación Tierra Luna Sol :

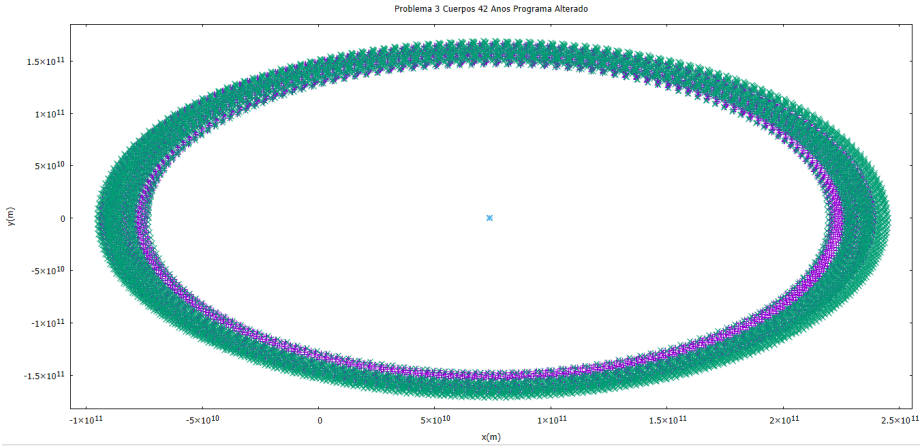

Figura 7: Evolución de La posición Tierra Luna Sol

3.4. Resultados de la Simulación de 100 cuerpos:

3.4.1. Momento Total:

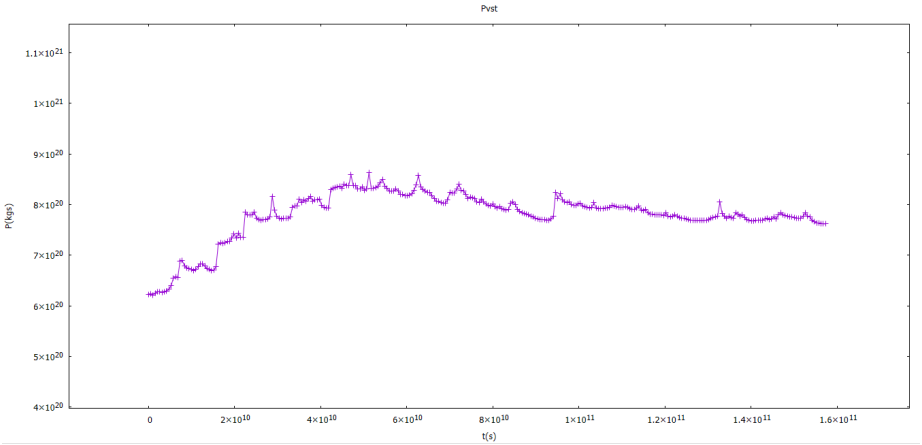

Figura 8: Momento Total 100 Cuerpos

3.4.2. Momento Angular Total:

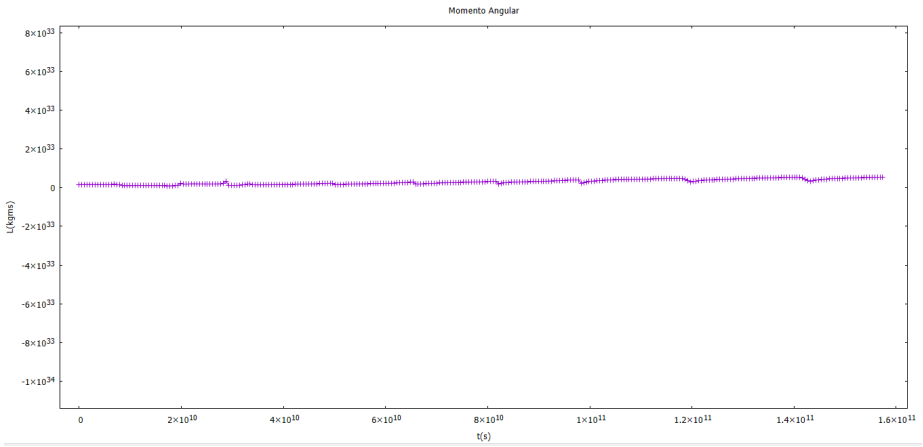

Figura 9: Momento Angular Total 100 Cuerpos

3.4.3. Momento Energía′ Total:

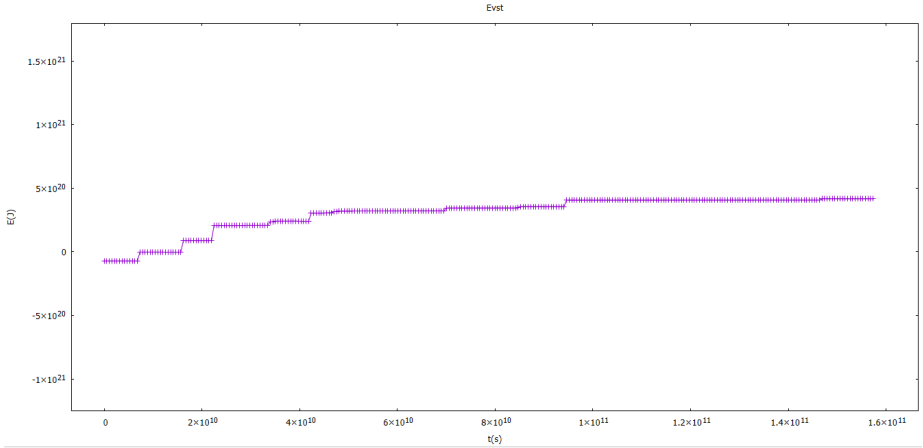

Figura 10: Energía’ Total 100 Cuerpos

3.4.4. Resumen de Movimiento:

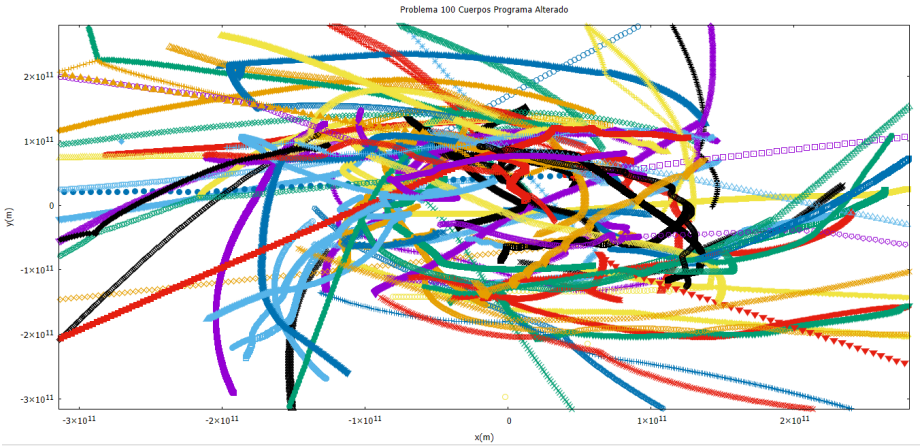

Figura 11: Evolución de La posición 100 Cuerpos

4. Discusión de Resultados

4.1. Script original

4.1.1. Respecto del Script Original:

- Se realizaron ciertos cambios en la estructura del programa base para priorizar la utilización de arreglos.
- Se definieron las funciones de forma que puedan producirse distribuciones de masas puntuales generalizadas, pero esto debe realizarse con cuidado ya que la velocidad inicial depende bajo una distribución uniforme.
- Se implementó una animación al momento de realizar una colisión por parte de la partícula a la cual se le es reducida su masa.
- Se utiliza la librería sstream que ya no es necesaria de incluir en la última versión de gcc, pero se coloca para tener retro compatibilidad. Esta librería contiene un tipo de variable llamada sstring que se utiliza para escribir en memoria el archivo utilizando en lugar de llamar al disco duro en cada iteración para optimizar el tiempo de ejecución.
- Respecto a las variables físicas como se menciona anteriormente no se calcula la energía total del sistema si no la diferencia de esta con el término del momento del centro de masa llamada E'.

4.1.2. Respecto de las gráficas:

Todas las gráficas se comportan de acuerdo a lo esperado salvo pequeños movimientos que aparecen por la aproximación numérica especialmente en la gráfica de E′. Existe una correlación entre los saltos de E′ y las colisiones entre las masas, o entre un error de la simulación que ocurre cuando 2 masas se acercaron y en vez de ocurrir una colisión, la masa resultante fue repelida con una enorme velocidad y la masa a la que se le quita su valor quedó estática. Esto ocurre debido a que existió un overflow por dividir valores muy pequeños y al evaluar la velocidad resultante en la colisión y el potencial en la energía.

4.1.3. Respecto de la validez:

Ya que este sistema puede modelar efectivamente el problema de lo 3 cuerpos durante 42 años, podemos tener seguridad que para casos con menor complejidad e intervalos de tiempo nuestra simulación produce efectivamente los resultados esperados por un sistema físico utilizando incrementos de tiempo durante un día. Pero no es posible concluir exactamente que tan precisa es la simulación del sistema final, ya que ocurren un errores que aparecen como un grandes saltos de ganancia de energía. Para realizar este tipo de simulación se sugiere utilizar un incremento de tiempo más pequeño aunque este no asegura la reproducción de una simulación exitosa, se realizaron mediciones de manera semi formal y el incremento de complejidad obtenido es de orden lineal.

4.2. Script Nuevo

4.2.1. Respecto del código:

- Se creó una estructura que contiene todos los datos que se desean medir para la simulación
- Se reescribió el método de la derivada, cálculo de la aceleración y RK4.
- Se realizan pequeñas optimizaciones priorizando realizar cálculos en memoria además de las ya realizadas en la modificación del script original
- Respecto a las variables físicas como se menciona anteriormente no se calcula la energía total del sistema si no la diferencia de esta con el término del momento del centro de masa llamada E′.

4.2.2. Respecto de las gráficas:

Notamos que en esta iteración de la simulación ninguna de las variables se comportan de manera constante que es lo esperado por la teoría. Esto se puede interpretar por el comportamiento divergente que se observa en la gráfica que resume las posiciones de los objetos del sistema, en vez de formar un sistema que orbita al rededor de un par de ejes.

4.2.3. Respecto de la validez:

Ya que este sistema puede modelar con mediana efectividad el problema de lo 3 cuerpos durante 42 años, ya que los cuerpos orbitan exitosamente entre sí pero la órbita de la tierra se va alejando con el paso de los años. No podemos tener mucha certeza que para casos con mayor complejidad e intervalos de tiempo nuestra simulación podrá reproducir resultados efectivos utilizando un incrementos de tiempo de un día. Se realizaron distintas simulaciones con distintos grados de éxito respecto a los resultados esperados por la teoría, pero no es posible reproducir de manera consistente un sistema que reproduzca los resultados esperados por la teoría. Para realizar este tipo de simulación se sugiere utilizar un incremento de tiempo mucho más pequeño aunque este no asegura la reproducción de una simulación exitosa, se realizaron mediciones de manera semi formal y el incremento de complejidad obtenido es de orden lineal. Debido a estas consideraciones para simular el sistema deseado se aconseja utilizar el script original.

4.3. Conclusiones

1. Nuestra implementación del método de RK4 no es lo suficientemente precisa para reproducir de manera consistente la evolución temporal de un sistema de 100 masas puntuales distribuidas aleatoriamente en un cuadrado durante 5000 mil años de evolución. Pero puede reproducir resultados moderadamente confiables con cierta frecuencia.

2. Un problema fundamental de esta simulación es que se debe encontrar un balance entre la precisión empleada en el modelo y el intervalo de tiempo que se desea estudiar. Para este caso particular se encontró que hacer intervalos de evolución de 1 día reproduce resultados más con mayor probabilidad de éxito sin comprometer tanto el tiempo de ejecución, para hacer una evolución de hasta 5000 mil años.

3. Las variables físicas de un sistema como la Energía, Momento y Momento angular total nos dan una pauta de hasta qué punto tiene sentido físico el comportamiento de un sistema simulado.

4. Puede realizarse a futuro un estudio con un menor tiempo de paso para tratar de obtener una simulación que pueda reproducir efectivamente el sistema físico, se aconseja utilizar el script original.

Referencias

[1] Luis Aguilar. The n-body problem - unam.

[2] Martin Houde. Chapter 7. dynamics of systems of particles.

[3] Enrique Pazos. Proyecto: Simulación gravitacional de n-cuerpos.