Prepared for submission to JHEP

Órbitas periódicas sin colisiones en el problema de caída libre de los tres cuerpos

Diego Zinker Rivera
José Manuel Martínez del Campo
Román Morales Castillo

Instituto Tecnológico Autónomo de México
Temas Selectos de Matemáticas Aplicadas

Índice

1. Introducción

El problema de los tres cuerpos consiste en determinar cómo interactúan tres cuerpos bajo las leyes de movimiento y gravitación de Newton. Por ejemplo, con el problema de tres cuerpos podríamos hacer una representación matemática de cómo interactúan el Sol, la Luna y la Tierra.

Newton fue el primero en estudiarlo, pero este problema ha atraído a muchas de las mentes más brillantes en matemáticas y su estudio ha resultado en grandes avances de conocimiento. Por ejemplo, Henri Poincaré descubrió que las soluciones del problema de tres cuerpos son muy sensibles a las condiciones iniciales. Es decir, si tomamos dos condiciones iniciales muy cercanas, con valores ligeramente distintos, podemos esperar que la solución de cada condición inicial sea sumamente diferente a la otra. Así es como nace la teoría del caos.

El comportamiento caótico es también lo que hace que este problema sea tan difícil de resolver. De hecho, sabemos que no existe una forma cerrada de resolverlo, es decir, que no es resoluble con un número finito de operaciones estándar, constantes, variables y funciones. Esta característica combinada con el avance de los métodos numéricos ha hecho que el problema de los tres cuerpos se traté de resolver de forma numérica en lugar de analítica

En particular, en el artículo que se estará revisando en este trabajo, los investigadores han encontrado 313 nuevas familias de órbitas periódicas para el problema de tres cuerpos "free-fall". Una órbita periódica es aquella solución en la que el movimiento de los tres cuerpos cumple cierta trayectoria que se repite a lo largo de un intervalo de tiempo al cual se le llama periodo. Por otro lado, el problema de tres cuerpos "free-fall" es un caso especial del problema de tres cuerpos en el cual los tres cuerpos comienzan en un estado de reposo, por ello las soluciones a este problema no son circuitos sino trayectorias que se recorren en ambos sentidos.

Los investigadores usaron una nueva técnica de análisis numérico llamada "Clean Numerical Simulation" (CNS). Debido a que los métodos numéricos tradicionales de precisión doble siempre incurren en errores de redondeo o truncamiento, y debido a la naturaleza caótica del problema de tres cuerpos, fue necesario introducir este nuevo método numérico. Por el comportamiento caótico del problema un ligero error de redondeo podría generar una solución completamente distinta a la correcta, así que con este nuevo método fue que los investigadores lograron su sorprendente resultado. A continuación lo analizaremos con más detalle.

2. Desarrollo del problema de las órbitas periódicas en el problema de los tres cuerpos

2.1. El problema de los tres cuerpos

2.1.1. Ecuaciones de movimiento.

Las ecuaciones del problema de los tres cuerpos se generan a partir de las ecuaciones de movimiento de Newton, en donde todos los cuerpos interactúan gravitacionalmente entre sí. Entonces, la posición del cuerpo i al tiempo t está dada por $\mathbf{r_i}(t) = (x_i, y_i, z_i)$, en donde cada componente de $\mathbf{r_i} = (x_i, y_i, z_i)$ depende a su vez del tiempo t.

$$\begin{aligned}
\ddot{\mathbf{r_1}} &= -Gm_2\frac{r_1 - r_2}{|r_1 - r_2|^3} - Gm_3\frac{r_1 - r_3}{|r_1 - r_3|^3} \\
\ddot{\mathbf{r_2}} &= -Gm_1\frac{r_2 - r_1}{|r_2 - r_1|^3} - Gm_3\frac{r_2 - r_3}{|r_2 - r_3|^3} \\
\ddot{\mathbf{r_3}} &= -Gm_1\frac{r_3 - r_1}{|r_3 - r_1|^3} - Gm_2\frac{r_3 - r_2}{|r_3 - r_2|^3}
\end{aligned} \tag{2.1}$$

La masa del cuerpo i es constante, y está dada por m_i. G es la constante gravitacional de Newton. En general, el movimiento de los cuerpos en este problema no es periódico, sin embargo, existen casos especiales en los que la razón de las masas entre los cuerpos del problema, así como la posición inicial, nos permiten observar soluciones periódicas. Como solo nos importa la proporción de masas entre los diferentes cuerpos, podemos asumir sin pérdida de generalidad que $G = 1$ y que $m_i \in (0, 1]$.

2.1.2. Descripción Hamiltoniana.

Es posible describir el problema de los tres cuerpos usando ecuaciones de Hamilton. Se generan las siguientes ecuaciones parciales:

$$\frac{d\mathbf{r_i}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p_i}}, \qquad \frac{d\mathbf{p_i}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{r_i}}$$

En donde $\mathbf{p_i}$ es el momento angular del cuerpo i. El sistema Hamiltoniano $\mathcal{H}$ quedará definido como:

$$\mathcal{H} = -\frac{Gm_1m_2}{|\mathbf{r_1} - \mathbf{r_2}|} - \frac{Gm_2m_3}{|\mathbf{r_3} - \mathbf{r_2}|} - \frac{Gm_3m_1}{|\mathbf{r_3} - \mathbf{r_1}|} + \frac{\mathbf{p_1}^2}{2m_1} + \frac{\mathbf{p_2}^2}{2m_2} + \frac{\mathbf{p_3}^2}{2m_3} \tag{2.2}$$

2.1.3. Posición inicial de los cuerpos en $\mathbb{R}^2$.

Consideramos las posiciones iniciales fijas de A y B en las coordenadas $(-0{,}5, 0)$, $(0{,}5, 0)$ respectivamente. Estos dos puntos están a una distancia de 1. Luego, consideramos el tercer cuerpo C en las coordenadas $(x, y) \in D$. Como las distancias que estamos considerando son arbitrarias en nuestro problema real, siempre podremos reparametrizar la distancia más larga como la distancia entre A y B, y así encontrar familias de trayectorias. De esta

manera, encontrar una órbita periódica en el plano equivale a encontrar los valores de $(x, y) \in D$ que generen una órbita periódica.

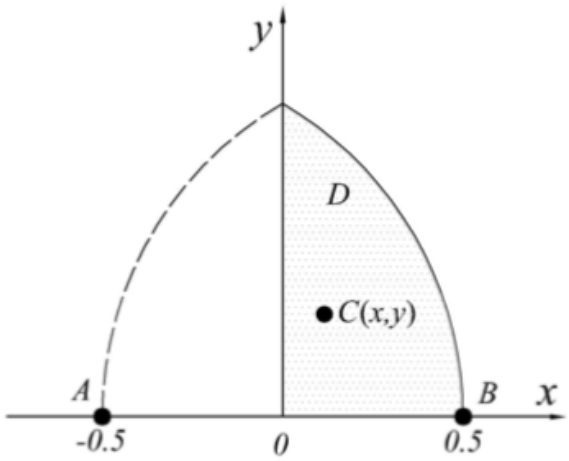


Figura 1: Posición inicial de los cuerpos A, B y C

2.2. Encontrando órbitas periódicas en el problema de los tres cuerpos

Acercamiento a la solución. Para encontrar soluciones periódicas en el problema de tres cuerpos recurrimos a métodos numéricos dado la complejidad de la forma analítica. Por más de 100 años se han intentado encontrar órbitas periódicas, y gracias al avance tecnológico hoy se cree que virtualmente se pueden encontrar una infinidad de éstas.

Planteando el problema. Definimos la función $\varphi(t) = (\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}, \dot{\mathbf{r_1}}, \dot{\mathbf{r_2}}, \dot{\mathbf{r_3}})$. Para solucionar el problema es necesario resolver la siguiente ecuación para algún $T > 0$, mientras se cumpla que los objetos no tengan colisiones entre sí. Esto es que $\forall t \in [0, T]$, $\mathbf{r_2(t)} \neq \mathbf{r_1(t)} \neq \mathbf{r_3(t)}$.

$$||\varphi(0) - \varphi(T)||_2 = 0 \tag{2.3}$$

Lo que esta ecuación nos dice es que, para algún tiempo $T > 0$, las condiciones iniciales se tienen que repetir. Esto es que las posiciones de cada uno de los objetos en el espacio y las velocidades de éstos al tiempo T sean las mismas que al tiempo 0. T es el período de la órbita.

3. Cálculo numérico de órbitas

3.1. Integración numérica

Para calcular las órbitas periódicas del problema de tres cuerpos en caída libre es necesario recurrir a métodos numéricos dada la complejidad del sistema de ecuaciones diferenciales. Para la integración numérica de las ecuaciones de movimiento (2.1.1), buscamos aproximar con distintos métodos la solución del problema en el tiempo t.

Los valores de G, y de m_1, m_2, m_3 se determinan para cada trayectoria. También, Consideramos solamente las coordenadas en el plano $\mathbb{R}^2$.

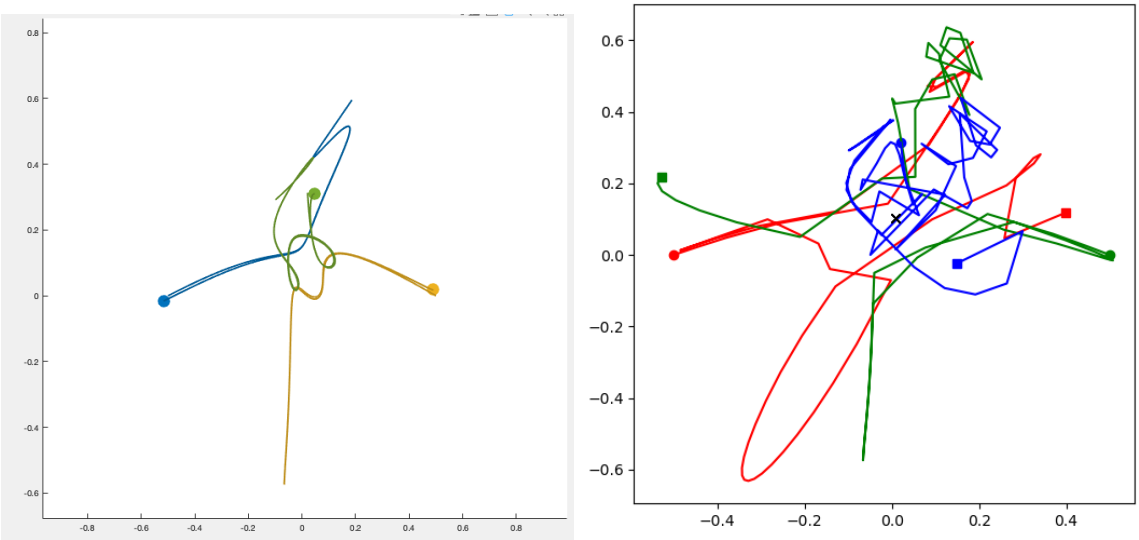
En el caso de MatLab, empleamos el método de solución de ecuaciones diferenciales *ode*45. Este método sirve para resolver EDOs no rígidas, con una buena precisión.

En el caso de Julia, nos apoyamos en el paquete Sundials. Este es una versión general de *solvers*, que engloba a otros como *ode*45 u *ode*15*s*.

3.2. Resultados Obtenidos

Los resultados obtenidos fueron los siguientes para algunas de las trayectorias estudiadas:

Condiciones iniciales:
$\mathbf{G} = 1, \quad \mathbf{m1} = 1, \quad \mathbf{m2} = 1, \quad \mathbf{m3} = 1$
$[\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}] = [-0{,}5, 0, 0{,}5, 0, 0{,}0207067154, 0{,}3133550361]$



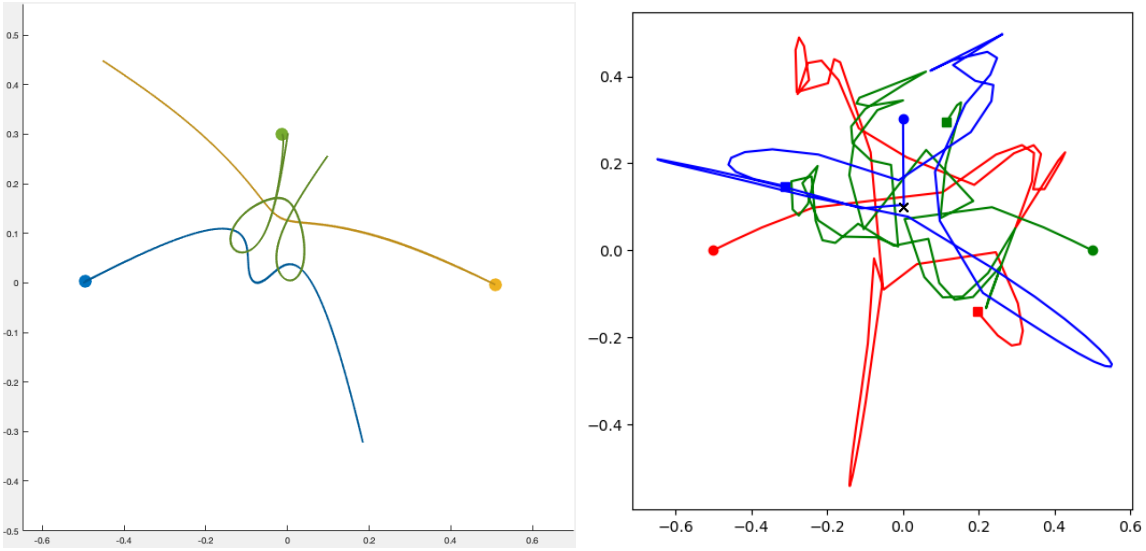
(a) Matlab (b) Julia

Figura 2: Trayectoria ejemplo 1

Condiciones iniciales:

$\mathbf{G} = 1, \quad \mathbf{m1} = 1, \quad \mathbf{m2} = 0{,}8, \quad \mathbf{m3} = 0{,}8$

$[\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}] = [-0{,}5, 0, 0{,}5, 0, 0{,}0009114239, 0{,}3019805958]$



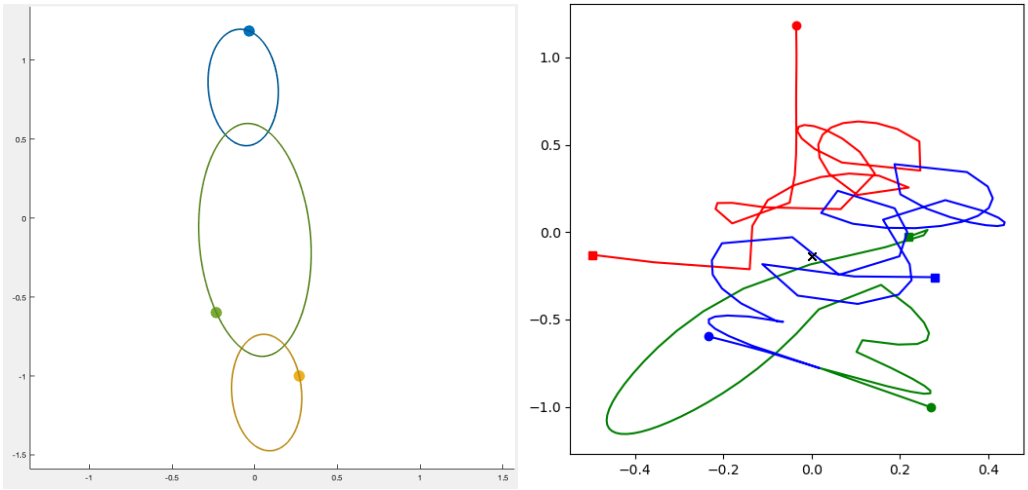
(a) Matlab (b) Julia

Figura 3: Trayectoria ejemplo 2

Condiciones iniciales:

$\mathbf{G} = 1, \quad \mathbf{m1} = 1, \quad \mathbf{m2} = 1, \quad \mathbf{m3} = 1$

$[\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}] = [-0{,}0347, 1{,}1856, 0{,}2693, -1{,}0020, -0{,}2328, -0{,}5978]$



(a) Matlab (b) Julia

Figura 4: Trayectoria de tres elipses

4. Conclusiones y Perspectivas

A pesar de que el problema de caída libre se había estudiado numéricamente durante todo el siglo pasado, no se habían descubierto más que 4 órbitas periódicas. El poder de computo de la modernidad nos permite seguir explorando problemas que se han estudiado por siglos. La Mecánica Celeste se seguirá viendo muy beneficiada de las nuevas técnicas de análisis numérico y simulación. Como sucedió en el artículo estudiado, se seguirán desarrollando técnicas para mejorar la precisión numérica de nuestras simulaciones y se aplicarán a la Mecánica Celeste. Así se seguirá aumentando el conocimiento que tenemos de nuestro universo y su física.

5. Apéndice: Códigos

5.1. Códigos en Matlab

Main.m

close all; clear all NumeroSol = 2 ;
p.G = 1; p.m1 = 1; p.m2 = 1; p.m3 = 1;
CI = 1;
switch CI case 1 [tend,inits] = getSolutionNum(NumeroSol);
case 2 tend = 2.1740969264; inits = [-0.5,0,0.5,0,0.0207067154,0.3133550361, 0,0,0,0,0,0];
case 3 tend = 1.8286248401; inits = [-0.5,0,0.5,0,0.0009114239,0.3019805958, 0,0,0,0,0,0];
p.m2=.8; p.m3=.8;
otherwise error('Solucion invalida'); end
p.tfactor = .5; NumeroPeriodos = 5;
p.odeOpts = odeset; p.odeOpts.RelTol = 1e-10; p.odeOpts.AbsTol = 1e-10;
[t,x] = ode45(@RHS,[0,NumeroPeriodos*tend],inits,p.odeOpts,p);
figure(1) plot(x(:,1),x(:,2)) xlabel('x') ylabel('y') hold on plot(x(:,3),x(:,4)) xlabel('x')
ylabel('y') hold on plot(x(:,5),x(:,6)) xlabel('x') ylabel('y') hold off
animate(t,x,p);

RHS.m

function zdot = RHS(t, z,p)
a = accel(p.G,p.m1,p.m2,p.m3,z(1),z(3),z(5),z(2),z(4),z(6));
zdot = [z(7:12);a];
end

accel.m

function a = accel(G,m1,m2,m3,x1,x2,x3,y1,y2,y3)
p1 = [x1;y1]; p2 = [x2;y2]; p3 = [x3;y3];
$r1_2 = p2 - p1; r1_3 = p3 - p1; r2_3 = p3 - p2;$
$f1_3 = G * m1 * m3/dot(r1_3, r1_3) * r1_3/norm(r1_3); f3_1 = -f1_3;$
$f2_3 = G * m2 * m3/dot(r2_3, r2_3) * r2_3/norm(r2_3); f3_2 = -f2_3;$
$f1_2 = G * m1 * m2/dot(r1_2, r1_2) * r1_2/norm(r1_2); f2_1 = -f1_2;$
$a1 = (f1_2 + f1_3)/m1; a2 = (f2_1 + f2_3)/m2; a3 = (f3_1 + f3_2)/m3;$
a = [a1;a2;a3]; end

getSolutionNum.m

function [tend,inits] = getSolutionNum(solNum)
switch solNum case 1 disp('Triple rings lined up solution returned.'); inits = [-0.0347
1.1856 0.2693 -1.0020 -0.2328 -0.5978 0.2495 -0.1076 0.2059 -0.9396 -0.4553 1.0471]'; tend
= 2.9623;
case 2 disp('Flower in circle.'); inits = [-0.602885898116520 1.059162128863347-1 0.252709795391000
1.058254872224370-1 -0.355389016941814 1.038323764315145-1 0.122913546623784 0.747443868604908
-0.019325586404545 1.369241993562101 -0.103587960218793 -2.116685862168820]'; tend =
2.246101255307486; case 3 disp('Montgomery 8 returned.'); inits = [-0.97000436,0.24308753,
0.97000436,-0.24308753, 0, 0,... -0.5*0.93240737,-0.5*0.86473146,-0.5*0.93240737,-0.5*0.86473146,0.93240737,

0.86473146]; tend = 5; case 4 disp('Ovals with flourishes. Takes very high tolerances to work.'); inits = [0.716248295712871 0.384288553041130 0.086172594591232 1.342795868576616 0.538777980807643 0.481049882655556 1.245268230895990 2.444311951776573 -0.675224323690062 -0.962879613630031 -0.570043907205925 -1.481432338146543]'; tend = 8.094721472532424; case 5 disp('Three diving into middle. Can tune velocity multiplier for more solutions.'); initPos = [1,0, -0.5,sqrt(3)/2, -0.5,-sqrt(3)/2]; initVel = 0.5*[0,1, -sqrt(3)/2,-0.5, sqrt(3)/2,-0.5]; inits = [initPos,initVel]; tend = 12; case 6 disp('Two ovals (2 on one, 1 on other).'); inits = [0.486657678894505 0.755041888583519 -0.681737994414464 0.293660233197210 -0.022596327468640 -0.612645601255358 -0.182709864466916 0.363013287999004 -0.579074922540872 -0.748157481446087 0.761784787007641 0.385144193447218]'; tend = 4.012156415940929; case 7 disp('Oval, catface, and starship.'); inits = [0.536387073390469 0.054088605007709 -0.252099126491433 0.694527327749042 -0.275706601688421 -0.335933589317989 -0.569379585580752 1.255291102530929 0.079644615251500 -0.458625997341406 0.489734970329286 -0.796665105189482]'; tend = 5.026055004808709;

case 8 disp('3 lined-up ovals, but with extra phase.'); inits = [0.517216786720872 0.556100331579180 0.002573889407142 0.116484954113653 -0.202555349022110 -0.731794952123173 0.107632564012758 0.681725256843756 -0.534918980283418 -0.854885322576851 0.427286416269208 0.173160065733631]'; tend = 5.179484537709100; case 9 disp('Skinny pineapple.'); inits = [0.419698802831451 1.190466261251569 0.076399621770974 0.296331688995343 0.100310663855700 -0.729358656126973 0.102294566002840 0.687248445942575 0.148950262064203 0.240179781042517 -0.251244828059670 -0.927428226977280]'; tend = 5.095053913455357; case 10 disp('Hand-in-hand-in-oval.'); inits = [0.906009977920936 0.347143444586515 -0.263245299491018 0.140120037699958 -0.252150695248079 -0.661320078798829 0.242474965162160 1.045019736387070 -0.360704684300259 -0.807167979921595 0.118229719138103 -0.237851756465475]'; tend = 6.868155929188273; case 11 disp('Beautiful loop-ended triangles inside an oval.'); inits = [1.666163752077218-1 -1.081921852656887+1 0.974807336315507-1 -0.545551424117481+1 0.896986706257760-1 -1.765806200083609+1 0.841202975403070 0.029746212757039 0.142642469612081 -0.492315648524683 -0.983845445011510 0.462569435774018]'; tend = 3.820761325134286; case 12 disp('Lined-up 3 ovals with nested ovals inside. Looks like newtons cradle type effect.'); inits = [1.451145020734434 -0.209755165361865 -0.729818019566695 0.408242931368610 0.509179927131025 0.050853900894748 0.424769074671482 -0.201525344687377 0.074058478590899 0.054603427320703 -0.498827553247650 0.146921917372517]'; tend = 3.550943103767421; case 13 disp('Oval and crossed triple loop.');

inits = [0.654504904492769 1.135463234751087 -0.008734462769570 -1.481635584087405 -0.487782115417060 0.236845409927442 0.294339951803385 -0.605376046698418 0.038678643994549 0.434105189236202 -0.333018595797923 0.171270857462206]'; tend = 5.377356578372074;

case 14 disp('Hard to describe squigglies PT1.') inits = [0.708322208567308 0.473311928206857 0.167739458699109 -0.057913961029346 -0.506578687023757 -0.306825234531109 0.824266639919723 0.522197827478425 -0.077017015655090 -0.167288552679470 -0.747249624264592 -0.354909274799606]'; tend = 5.403010724273298;

case 15 disp('Hard to describe squigglies PT2.')

inits = [0.865355422048074 0.629488893636092 0.085036793558805 -0.013305780703954 -0.090983494772311 -0.892179296799402 0.288687787606716 0.171289709266963 -0.220256752038501

0.090375753070602 -0.068431035568003 -0.261665462337436]'; tend = 5.427986085939034;
case 16 disp('Gear inside larger oval. VERY VERY STIFF WARNING. SEE CODE COMMENTS.'); inits = [0.335476420318203 -0.243208301824394 0.010021708193205 0.363104062311693 0.030978712523174 0.423035485079015 1.047838171160758 0.817404215288346 -0.847200907807940 -0.235749148338353 -0.200636552532016 -0.581655492859626]'; tend = 5.956867234319493;
case 17 disp('3 bouncing in a line (almost).'); inits = [-0.015958157830872 0.829387672711051 -0.023912093175726 0.211010340157083 -0.035305026035053 -0.728544174709096 0.008894140366887 0.735934914230558 -0.012693330102595 -1.001493752025633 0.003743585058501 0.265560077637935]'; tend = 1.253088248568183; otherwise tend = []; inits = []; error('Solucion invalida.'); end

end

animate.m

function fig = animate(tarray,zarray,p)

fig = figure; fig.Position = [1000,100,800,800];

hold on p1 = plot(0,0,'.','MarkerSize',50); p1cola = plot(0,0,'-','LineWidth',2,'Color',0.8*p1.Color); p2 = plot(0,0,'.','MarkerSize',50); p2cola = plot(0,0,'-','LineWidth',2,'Color',0.8*p2.Color); p3 = plot(0,0,'.','MarkerSize',50); p3cola = plot(0,0,'-','LineWidth',2,'Color',0.8*p3.Color); hold off

axis([-2.2,2.2,-2.2,2.2]);

tact = 0; tic;

while tact*p.tfactor <tarray(end)

zact = interp1(tarray,zarray,tact*p.tfactor);

p1.XData = zact(1); p1.YData = zact(2);

p2.XData = zact(3); p2.YData = zact(4);

p3.XData = zact(5); p3.YData = zact(6);

tind = find(tarray<tact*p.tfactor,1,'last');

p1cola.XData = zarray(1:tind,1); p1cola.YData = zarray(1:tind,2); p2cola.XData = zarray(1:tind,3); p2cola.YData = zarray(1:tind,4); p3cola.XData = zarray(1:tind,5); p3cola.YData = zarray(1:tind,6);

drawnow;

tact = toc; end end

5.2. Códigos en Julia

using PyPlot
using Sundials
using LinearAlgebra
using Printf
Definimos función para sistema
function f(t, g, gdot)
Obtenemos la posición inicial y la velocidad inicial
r0, v0 = g[1:2], g[3:4]
r1, v1 = g[5:6], g[7:8]
r2, v2 = g[9:10], g[11:12]

Calculamos las derivadas de la posición que en realidad son las velocidades
dr0 = v0
dr1 = v1
dr2 = v2

Calculamos la aceleración que en física es la derivada de la velocidad
Obtenemos distancia entre cada uno de los cuerpos
Solo hacemos renombramiento
d0 = (r2 - r1) / (norm(r2 - r1)^3,0)
$d1 = (r0 - r2)/(norm(r0 - r2)^3$,0)
$d2 = (r1 - r0)/(norm(r1 - r0)^3$,0)

dv0 = m1*d2 - m2*
dv1 = m2*d0 - m0*d2
dv2 = m0*d1 - m1*d0

Construimos vector de derivadas final
gdot[:] = [dr0; dv0; dr1; dv1; dr2; dv2]
end;

Masas de los cuerpos (asumimos todas igual a 1)
m0 = 1.0
m1 = 1.0
m2 = 1.0

Vector de posiciones y velocidades inicial de cada uno de los cuerpos (x0, y0, vx0, vy0)
gi0 = [-0.50; 0.0; 0.0; 0.0]
gi1 = [0.50; 0.0; 0.0; 0.0]
gi2 = [0.0207067154 ; 0.3133550361 ; 0.0; 0.0] Sí es solución del escrito
gi2 = [0.0009114239; 0.3019805958 ; 0.0; 0.0] Sí es solución del escrito

gi2 = [0.0726030609 ; 0.3037940331 ; 0.0; 0.0]
gi2 = [0.0075986006; 0.1288965886 ; 0.0; 0.0]
gi2 = [0.1400273104; 0.0081786581 ; 0.0; 0.0]

Hacemos trayectoria o simulación
tf = 10.0
dt = 6.0
t = collect(range(5.0, tf, round(Int,tf*dt)))
g0 = [gi0; gi1; gi2]
res = Sundials.cvode(f, g0, t, reltol=1e-10)

Obtenemos velocidad y posición de resultado
r0, v0, r1, v1, r2, v2 = res[:,1:2], res[:,3:4], res[:,5:6], res[:,7:8], res[:,9:10], res[:,11:12]

Solo para tener el centro de masa ->grafica
cx = [(r0[i,1]*m0 + r1[i,1]*m1 + r2[i,1]*m2) / (m0 + m1 + m2) for i=1:length(t)]
cy = [(r0[i,2]*m0 + r1[i,2]*m1 + r2[i,2]*m2) / (m0 + m1 + m2) for i=1:length(t)]

Graficar

function plot$_t rajectory(t1, t2)$
$t1i = round(Int, (length(t) - 1) * t1/tf) + 1$
$t2i = round(Int, (length(t) - 1) * t2/tf) + 1$

Graficamos vectores iniciales y finales
Primera coordenada X y segunda Y
X = 1
Y = 2

figure(figsize=(6,6))
plot(r0[t1i,X], r0[t1i,Y], ro")
plot(r0[t2i,X], r0[t2i,Y], rs")
plot(r1[t1i,X], r1[t1i,Y], "go")
plot(r1[t2i,X], r1[t2i,Y], "gs")
plot(r2[t1i,X], r2[t1i,Y], "bo")
plot(r2[t2i,X], r2[t2i,Y], "bs")

Graficamos trayectorias
plot(r0[t1i:t2i,X], r0[t1i:t2i,Y], r-")
plot(r1[t1i:t2i,X], r1[t1i:t2i,Y], "g-")
plot(r2[t1i:t2i,X], r2[t1i:t2i,Y], "b-")

Centro de masa del sistema (no es necesario)
plot(cx[t1i:t2i], cy[t1i:t2i], "kx")

Definimos los ejes
xmin = minimum([r0[t1i:t2i,X]; r1[t1i:t2i,X]; r2[t1i:t2i,X]]) * 1.10
xmax = maximum([r0[t1i:t2i,X]; r1[t1i:t2i,X]; r2[t1i:t2i,X]]) * 1.10
ymin = minimum([r0[t1i:t2i,Y]; r1[t1i:t2i,Y]; r2[t1i:t2i,Y]]) * 1.10
ymax = maximum([r0[t1i:t2i,Y]; r1[t1i:t2i,Y]; r2[t1i:t2i,Y]]) * 1.10

axis([xmin, xmax, ymin, ymax])
title(@sprintf "3-body simulation for t=[end;

plot$_t rajectory(0, 10)$;

6. Referencias

1. Xiaoming Li, Shijun Liao, Collisionless periodic orbits in the free-fall three- body problem, New Astronomy (2019), doi: https://doi.org/10.1016/j.newast.2019.01.003.

2. Marques, P., *Julia Modelling the World: 3 Body Problem*, 15 de Febrero de 2016, https://github.com/pjpmarques/Julia-Modeling-the-World/blob/master/Three-Body%20Problem.ipynb, consultado: 10 de Diciembre de 2021.