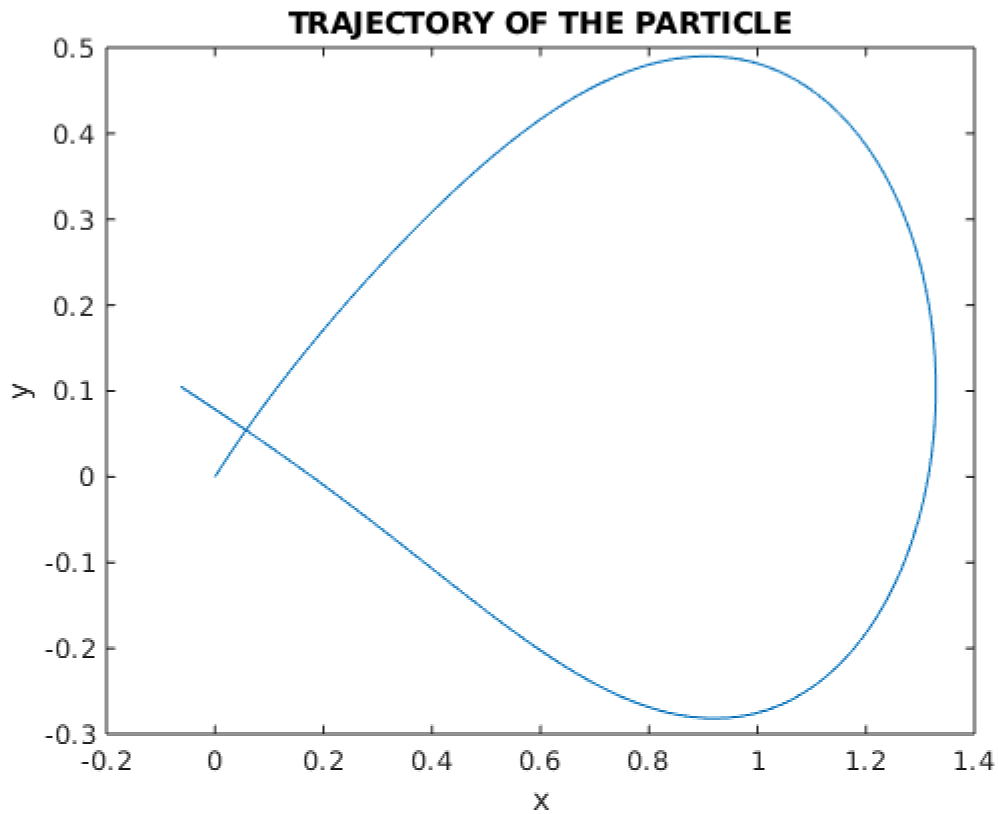
Practica_19_Casas_Mercade

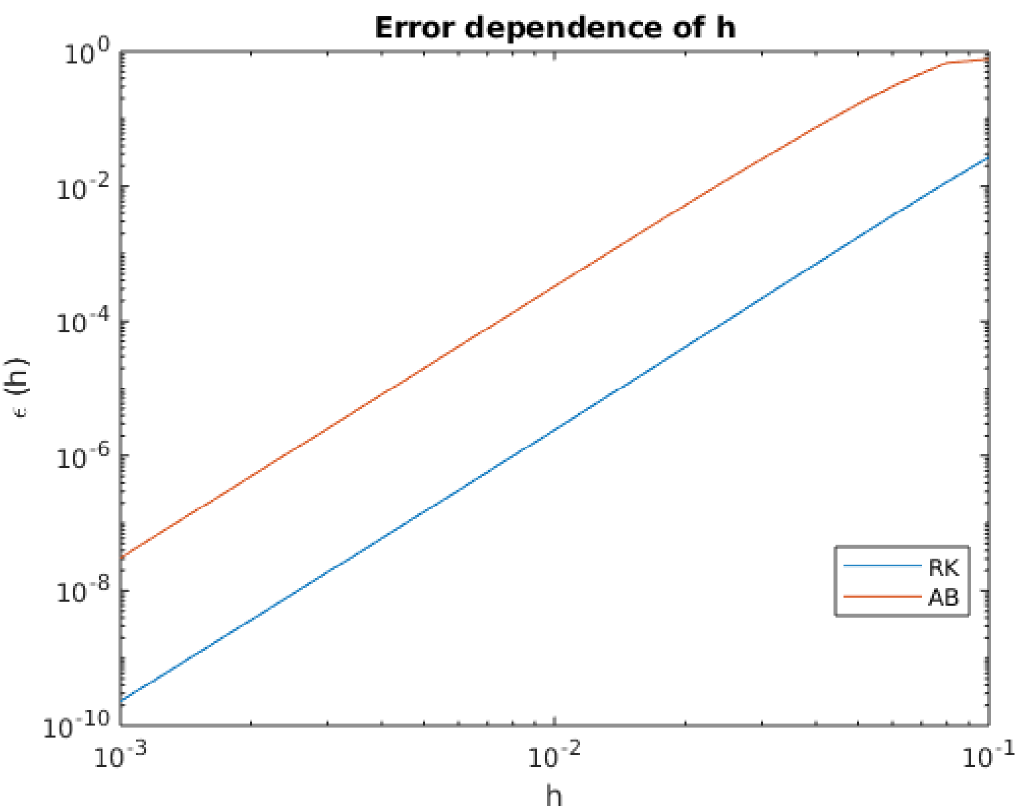
Table of Contents

Section A

```
% x = (v,r)
% fun: (v,r) --> (dr/dt, dv/dt)
format long g;
close all;
clc;
clear all;

h = 1e-2;
time = 2;
points = time / h + 1;
initial = [0, 0, 1, 1]';

%With RK4 we get the addition steps for the AB4
solutionRK = RK4(initial, h, @gravFunctionV, points);
aditionalSteps = solutionRK(:, 1:4);

%With AB4 we get the the positions and velocities of m during the
 first two
%seconds since its launch
solutionAB = AB4(aditionalSteps, h, @gravFunctionV, points);
figure;
plot(solutionAB(1, :), solutionAB(2, :));
title('TRAJECTORY OF THE PARTICLE')
xlabel('x')
ylabel('y')

% To calucate the error we first calculate the exact solution, which
 we
% will consider that is the one obtained for h=1e-4
hext = 1e-4;
points = time / hext +1;
exactSolutionRK = RK4(initial, hext, @gravFunctionV, points);
rext = exactSolutionRK(1:2, end);

%Now we find the position at t=2 for values of h bigger than
 hext=1e-4, and
%calculate the diffrence with the exact solution. We do this with RK4
 and
%AB4 to see the order of the error obtained with each method.
```

```
errorsRK = [];
errorsAB = [];
hs = [];
haches = 1e-3:0.0001:0.1;

for h = haches
    points = time / h + 1;
    %RK i AB gives us the n+1 point so in order to obatin the one
    %corresponding to t=2 we have to add this plus one

    if floor(points) == points
        % The number of points must be a natural number and for some
 values
        %of h it is a decimal one, we one use those values that gives
 a
        %natural number of points
        hs = [hs h];
        solutionRK = RK4(initial, h, @gravFunctionV, points);
        r = solutionRK(1:2, end);
        errorsRK = [errorsRK norm(r - rext)];

        %As done before, we use the first three steps provided by RK
 plus
        %the intial condition
        solutionAB = AB4(solutionRK(:, 1:4), h, @gravFunctionV,
 points);
        r2 = solutionAB(1:2, end);
        errorsAB = [errorsAB norm(r2 - rext)];
    end

end

figure;
loglog(hs, errorsRK)
hold on
loglog(hs, errorsAB)
hold off
title('Error dependence of h')
xlabel('h')
ylabel('\epsilon (h)')
legend('RK', 'AB', 'Location', 'best');

% Questions:
% Looking at the logaritmic plot of the error we can see that
% AB4 requires h = 0.008 to reach the precision of 10^-4 and
% RK4 needs h 0.25.

% RK4 should be slower because it requires 4 evaluations of the
 function while
% AB4 only needs one evaluation of the function with the
 optimitzations done.
```


TRAJECTORY OF THE PARTICLE
y
x
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4


Error dependence of h
ϵ (h)
h
RK
AB

Section B)

Since for theta=pi/4 it was seen that the time taken was less than t=2 we keep the angle but reduce the time for the inital guess

```
z0 = [pi / 4, 1.8]';
%funForNewton takes the inital angle and the time and returns de
 distance
%to the origin at the final point.
% We use the tolerance specified on newtonn te get the tolerance
 required
% by the problem
[XK, resd, it] = newtonn(z0, 1e-8, 100, @funForNewton);
disp('The inital angle is')
disp(XK(1, end))
disp('The time of arrival is')
disp(XK(2, end))

%{
function r = funForNewton(z)
    % Function to launch newtonn
    % Input: z is a vector of two components, the first one
 corresponds to
    % the launch angle of the particle and the second one to the time
 that it
    % takes to get to the origin
    % Output: the distance to the origin. Newton will make it 0.
    initial = [0; 0; sqrt(2) .* cos(z(1)); sqrt(2) .* sin(z(1))]; %
 initial point to launch RK4
    steps = 20000; %h will be smaller than 0.0001 since we the time to
 be < 2

    if 0 < z(2) < 2
        h = z(2) / steps;
        sol = RK4(initial, h, @gravFunctionV, steps + 1);
        r = sol(1:2, end);
    else % If time is greater than 2, we will introduce an "artificial
 slope" to help newton to converge to r = (0, 0)
        % If we launch newton to the correct point we will not reach
 this code:
        disp('Surpasing t = 2');
        r = [1; 1] * z(2);
    end

end

%}

%{
function [XK, resd, it] = newtonn(x0, tol, itmax, fun)
    % This code is the newton method for nonlionear systems, is an
 iterative
```

```
   % method that allows you to approximate the solution of the system
with a
   % precision tol

   % INPUTS:
   % x0 = initial guess  --> column vector
   % tol = tolerance so that ||x_{k+1} - x_{k} || < tol
   % itmax = max number of iterations allowed
   % fun = @ ffunction_handler
   % OUTPUT:
   % XK = matrix where the xk form 0 to the last one are saved (the
last
   % one is the solution) --> saved as columns
   % Resd = resulting residuals of iteration: ||F_k||, we want it to
be 0,
   % as we are looking for f(x)=0
   % it = number of required iterations to satisfy tolerance

   xk = [x0];
   XK = [x0];
   resd = [norm(feval(fun, xk))];
   it = 1;
   tolk = 1;

   while it < itmax && tolk > tol
       J = jaco(fun, xk); % Jacobia en la posicio anterior
       fk = feval(fun, xk);
       [P, L, U] = PLU(J);
       Dx = pluSolve(L, U, P, (-fk)'); %Solucio de la ecuacio J*Dx =
-fk
       %Matlab linear sistem solving
       %Dx = J\(-fk)';
       xk = xk + Dx;
       XK = [XK, xk];
       resd = [resd, norm(fk)];
       tolk = norm(XK(:, end) - XK(:, end - 1));
       it = it + 1;
   end

   %}
```

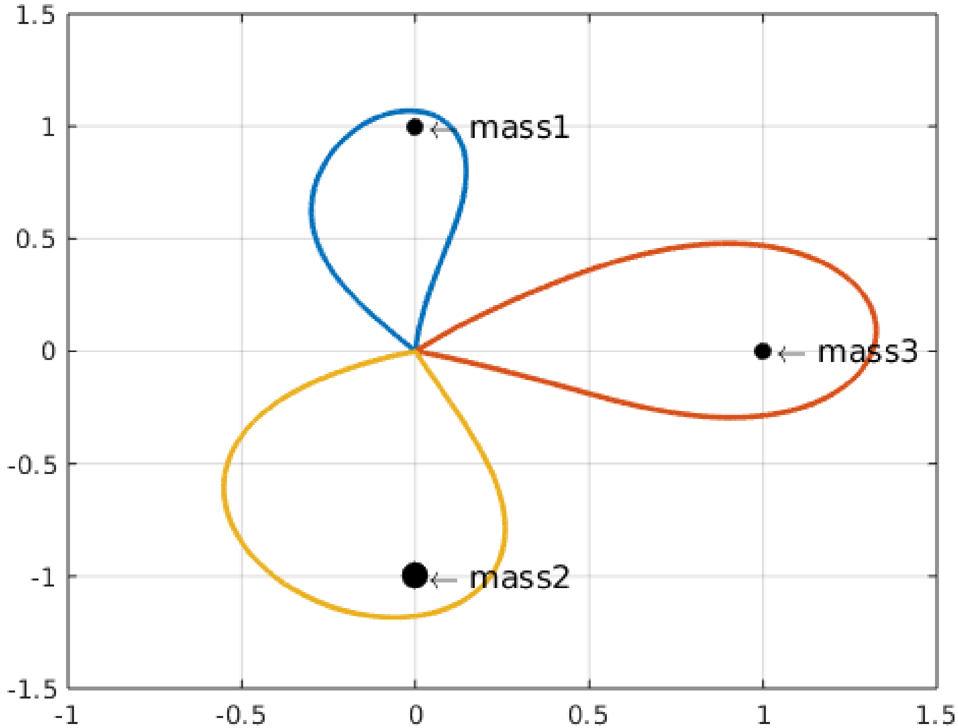
The inital angle is
0.77745497923704

The time of arrival is
1.91158621395717

Section C

```
   h = 1e-3;
   v = sqrt(2);
   iTime = 1.8;
```

```
   thetas = [pi / 2, 0, -pi / 2];
   %The intial theta guess is based on the problem's symmetry

   figure;

   for ii = thetas
       z0 = [pi / 4 + ii, 1.8]';
       %We use newton to find which is the launch angle and the
required
       %to get to the origin again
       [XK, resd, it] = newtonn(z0, 1e-8, 100, @funForNewton);
       %Once it has been obatined we use those results to know how
many steps
       %are required and the components of the inital velocity, and
with this
       %information we can proced as in section A in order to do the
plot of
       %the tragectory followed by the particle
       points = floor(XK(2, end) / h) + 2;
       initial = [0, 0, v * cos(XK(1, end)), v * sin(XK(1, end))]';
       solutionRK = RK4(initial, h, @gravFunctionV, points);
       plot(solutionRK(1, :), solutionRK(2, :), 'LineWidth', 2)
       grid on
       hold on
   end

   massPositions = [0, 1; 1, 0; 0, -1];
   plot(massPositions(1:2, 1), massPositions(1:2,
2), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'k')
   plot(massPositions(3, 1), massPositions(3, 2), '.r', 'MarkerSize',
35, 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'k')
   text(massPositions(1, 1), massPositions(1, 2), ' \leftarrow
mass1', 'FontSize', 12)
   text(massPositions(3, 1), massPositions(3, 2), ' \leftarrow
mass2', 'FontSize', 12)
   text(massPositions(2, 1), massPositions(2, 2), ' \leftarrow
mass3', 'FontSize', 12)

   % The three diferent trajectories found make a turn around one of
the masses.
   % All of the trajectores have a teardrop shape.
   % Finally, another property in common is that they do the loop
anti-clockwise
   % around the masses-
```



Published with MATLAB® R2019b