Table of Contents

Practical 9 - Cortés & García

```
clear; close all;
format long;
set(0, 'DefaultTextInterpreter', 'latex');
```

3 forces are acting on the particle of mass m due to each particle of mass m_i with $i = \{1, 2, 3\}$. From here we'll have that, naming $\vec{f}$ as the force acting on m

$$\vec{f} = m\vec{a} = m\partial_t^2 \vec{r}$$

From this, we clearly have a partial diferential equation of 2nd order. Once we have defined the force function $\vec{f}$, we'll define the variable that will be used through out the practical z and the new partial diferential equation (using f_x and f_y the x and y components of $\vec{f}$, respectively):

$$\vec{z} = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}$$

$$\dot{\vec{z}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ f_x/m \\ f_y/m \end{pmatrix}$$

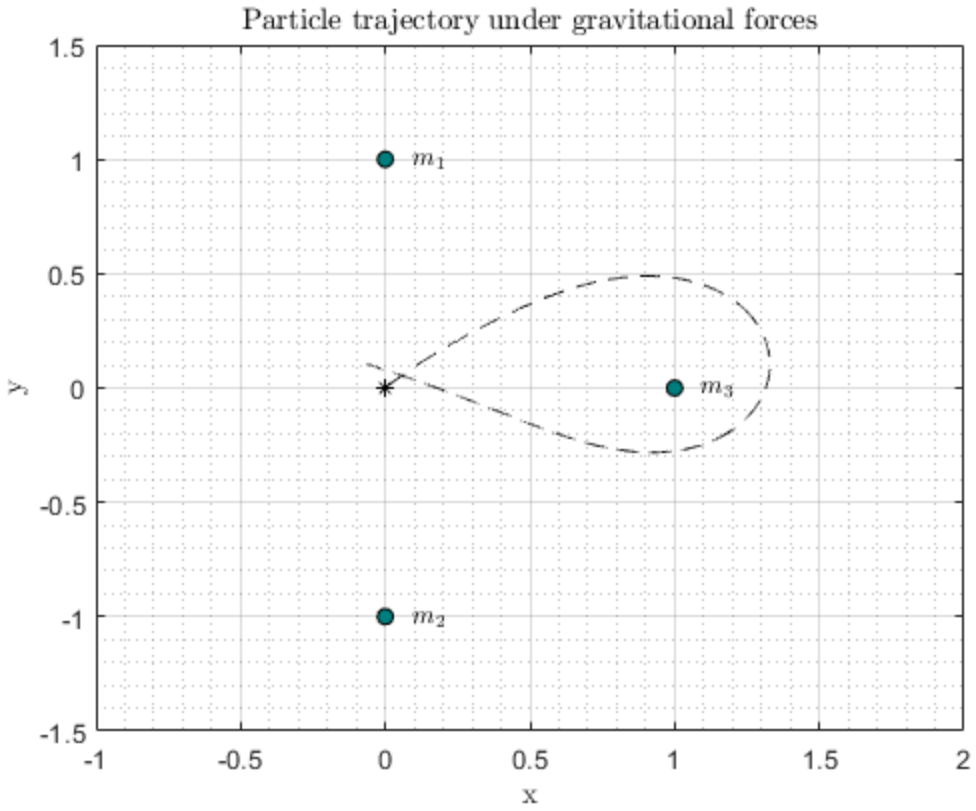
Now it is only left that we use the RK4 and AB4 methods to approximate the coordinates of the moving particle m.

```
% set parameter values: masses & positions of the masses
m1 = 1; m2 = 2; m3 = 1; m = 1;
r1 = [0 ; 1]; r2 = [0; -1]; r3 = [1 ; 0];

% define the x and y component of the force between to masses
Fx = @(mi, mj, ri, rj) (mi*mj)*(ri(1)-rj(1))/(norm(ri-rj)^3);
Fy = @(mi, mj, ri, rj) (mi*mj)*(ri(2)-rj(2))/(norm(ri-rj)^3);

% define the total force in each direction acting on m
FxTot = @(m, r) Fx(m1, m, r1, r) + Fx(m2, m, r2, r) + Fx(m3, m, r3,
 r);
```

```
FyTot = @(m, r) Fy(m1, m, r1, r) + Fy(m2, m, r2, r) + Fy(m3, m, r3,
 r);


global f % define as global to use in parts b) and c)
f = @(t,z) [z(3) ; z(4) ; FxTot(m, z(1:2))/m ; FyTot(m, z(1:2))/m];
```

(a) Error

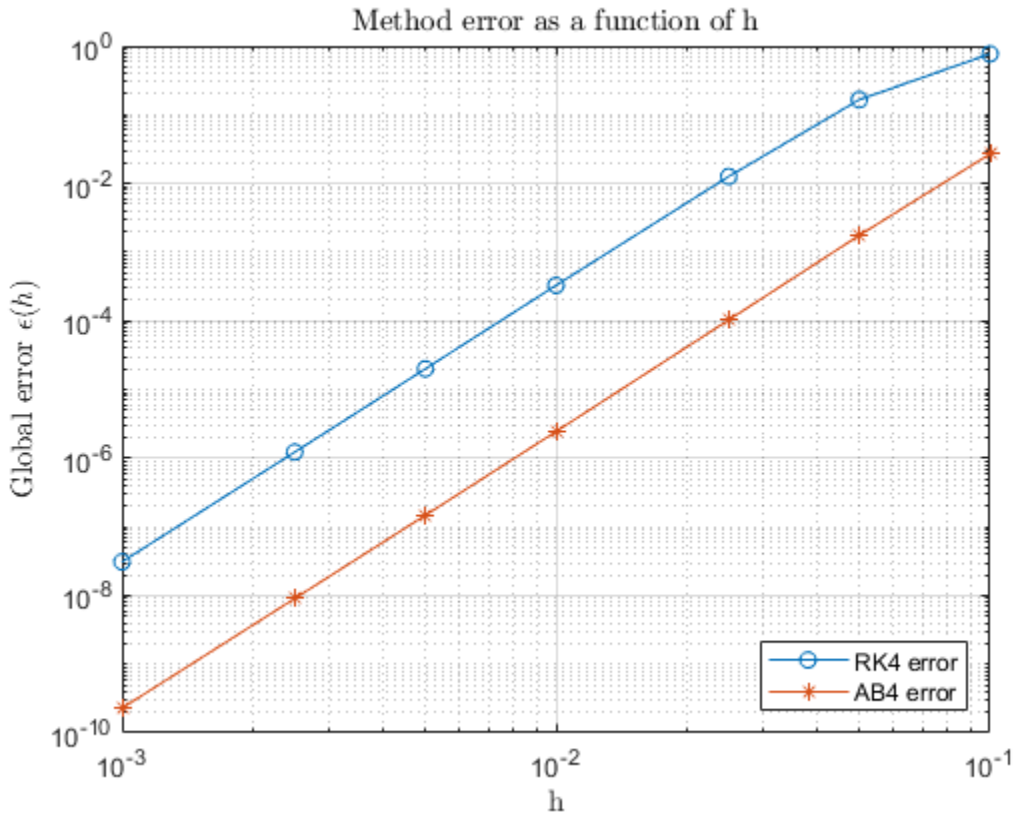
```
% initial condition z0 = [x0 y0 vx0 vy0] of the particle of mass m
z0 = [0 0 1 1]';
h = 1e-4;
steps = round(2/h);

% Compute the exact coordinates at t=2 with the RK4 method
[T, Z] = rk4(f, 0, h, z0, steps);

figure(1)
plot(Z(1,:), Z(2,:), 'k--') % plot the trajectory of the particle
hold on
% plot the masses
plot(r1(1),r1(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r1(1),r1(2),'$\quad m_1$', 'Interpreter', 'latex')
plot(r2(1),r2(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r2(1),r2(2),'$\quad m_2$', 'Interpreter', 'latex')
plot(r3(1),r3(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r3(1),r3(2),'$\quad m_3$', 'Interpreter', 'latex')
plot(0,0,'k*')
axis([-1 2 -1.5 1.5])
grid(gca,'minor')
grid on
title('Particle trajectory under gravitational forces')
xlabel('x')
ylabel('y')
```



In this first figure we can see that the obtained trajectory of the particle is pretty similar to the one displayed in the book. This makes us think that the method used (RK4) has been well implemented.

```
% Define the "exact" solution to compute error for a very smnall h
% Use the previous computation
exact_sol = Z(1:2,end);

% Define different h values with which we will compute the error
% This values have been chosen for they ensure the particle will stop
% moving at t=2 (they have been picked as to make sure that there is
 an
% integer number of steps before reaching t=2)
hVec = [0.1 0.05 0.025 0.01 0.005 0.0025 0.001];
errVec_RK4 = zeros(1, 7);
errVec_AB4 = zeros(1, 7);
for k = 1:length(hVec)
    % Define parameters
    hh = hVec(k); steps = round(2/hh);

    [~, Zh_rk4] = rk4(f, 0, hh, z0, steps);             % RK4

    [~, Zh_ab4] = ab4(f, 0, hh, Zh_rk4(:, 1:4), steps); % AB4

    % Update error vectors
    errVec_AB4(k) = norm(Zh_rk4(1:2, end) - exact_sol);
    errVec_RK4(k) = norm(Zh_ab4(1:2, end) - exact_sol);
```

```
end

% Plot the errors in a logarithmic scale
figure(2)
loglog(hVec, errVec_RK4,'o-')
hold on
loglog(hVec, errVec_AB4,'*-')
title('Method error as a function of h')
xlabel('h')
ylabel('Global error $\epsilon (h)$')
legend('RK4 error', 'AB4 error', 'Location', 'SouthEast')
grid on
hold off
```



In the previous plot we can observe that when the value of h increases by a factor of 10^2, the error increases by a factor of 10^8. This coincides with the error $O(h^4)$ that we expected. This happens for both methods more or less, even though the error for the RK4 method is greater than that obtained with the AB4 approximation.

To have an error $\epsilon(h) \leq 10{-}4$, we'll need $h \leq 7.5 \times 10^{-3}$ for the AB4 method and $h \leq 2.5 \times 10^{-2}$ for the RK4 method.

The RK4 method requires more evaluations of the field vector: whereas the AB4 method will perform 1 evaluation of f per iteration (aside from the first 4 evaluations for the initial conditions), the RK4 method needs 4 evaluations of f per iteration.

(b) Around m_3 and back to the origin

To solve this section, we have implemented another function fun (with the auxiliar codes). This function will return the final position of the particle. We'll define the input variable of the function

$$\vec{u} = \begin{pmatrix} \theta_0 \\ T_a \end{pmatrix}$$

where θ_0 is the initial angle of the velocity of the particle (we'll always have initial speed $\sqrt{2}$) and T_a is the time at which the particle arrives to this position (the output of fun). We'll want to fint $\vec{u}$ such that the final position is the origin, with the condition that $T_a < 2$.

```
% Define approximated initial condition
u0 = [pi/4 ; 1.9];
% Solve with Newton for the solution
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);

th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta); % display results

% Compute the trajectory for this solution
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
N = 1000; hh = Ta/N;
[~, Zh1] = rk4(f, 0, hh, z0, N);
```

```
   0.777454978986880

   1.911586213475094
```

We can see with these 2 displays that the obtained solution is valid (since $T_a < 2$). The trajectory computed with the solution is displayed afterwards in figure 3.

(c) Around m_1 & m_2 and back to the origin

For this part, we'll repeat the process done in (b) but changing the initial conditions. Depending on the particle we want the particle to go around, we'll need a different angle θ_0 as a starting point. We have maintained $T_a = 1.9$ as a starting point since they worked pretty well.

```
% Compute solution for m1
u0 = [3*pi/4 ; 1.9];
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);
th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta);

% Compute new trajectory for m1
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
N = 1000; hh = Ta/N;
[~, Zh2] = rk4(f, 0, hh, z0, N);
```

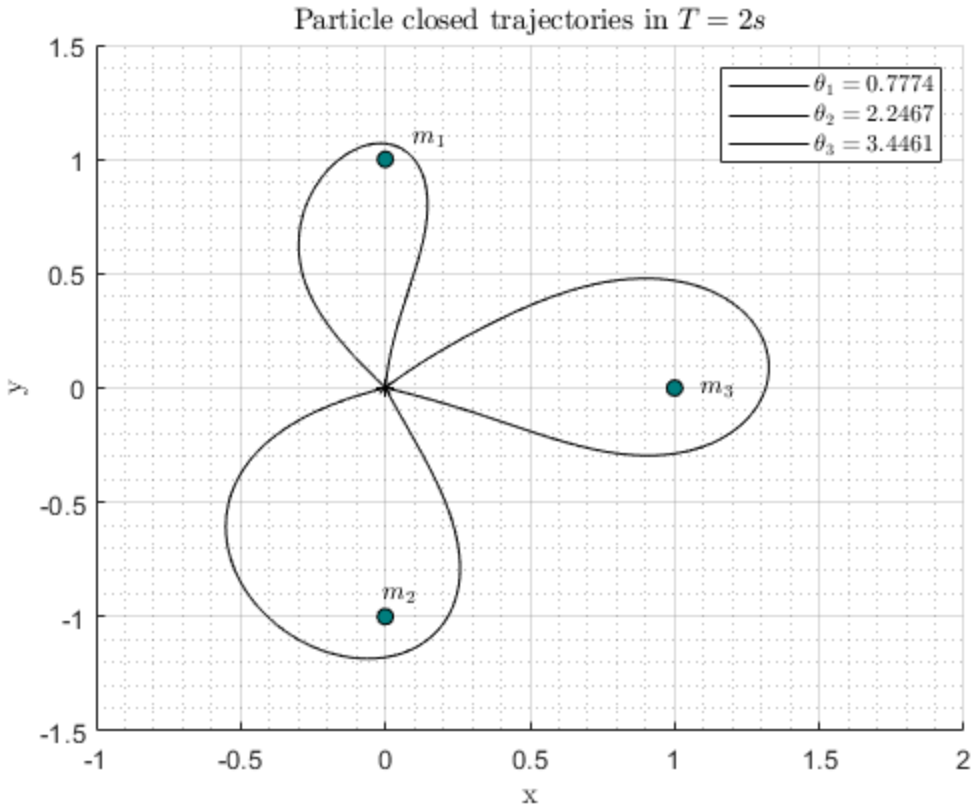
```
   2.246792185103864
```

1.436358101656933

```
% Compute solution for m2
u0 = [7*pi/6 ; 1.9];
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);
th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta);

% Compute new trajectory for m2
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
N = 1000; hh = Ta/N;
[~, Zh3] = rk4(f, 0, hh, z0, N);

figure(3)
hold on
% Plot trajectories
plot(Zh1(1,:), Zh1(2,:), 'k')
plot(Zh2(1,:), Zh2(2,:), 'k')
plot(Zh3(1,:), Zh3(2,:), 'k')
% Plot the masses
plot(r1(1),r1(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r1(1),r1(2)+0.1,'$\quad m_1$', 'Interpreter', 'latex')
plot(r2(1),r2(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r2(1)-0.1,r2(2)+0.1,'$\quad m_2$', 'Interpreter', 'latex')
plot(r3(1),r3(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r3(1),r3(2),'$\quad m_3$', 'Interpreter', 'latex')
plot(0,0,'k*')
axis([-1 2 -1.5 1.5])
grid(gca,'minor')
grid on
title('Particle closed trajectories in $T = 2s$')
legend('$\theta_1 = 0.7774$','$\theta_2 =  2.2467$','$\theta_3 =
 3.4461$', 'Interpreter','latex')
xlabel('x')
ylabel('y')
hold off
```

3.446142140790294

1.599214813041981



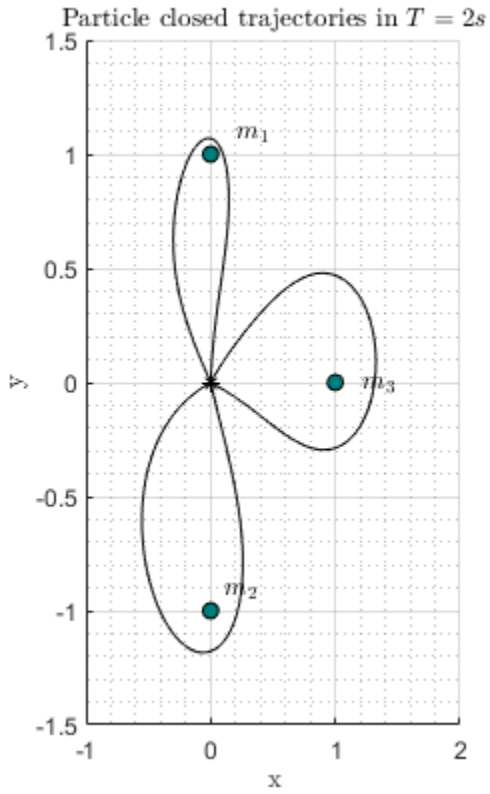
In this last figures we observe all the computed trajectories. We do observe how the particle of mass m goes arround each of the three other particles and returns to the origin. Again, all the arrival times are $T_a < 2$ so they are valid solutions and trajectories. At the same time, one trait we could say they all have in common is that their shape is more or less the same, though rotated.

We might as well use another initial condition for each trajectory and see what happens.

```
% Plot previous trajectories on one side of the figure
figure(4)
subplot(1, 2, 1)
hold on
% Plot trajectories
plot(Zh1(1,:), Zh1(2,:), 'k')
plot(Zh2(1,:), Zh2(2,:), 'k')
plot(Zh3(1,:), Zh3(2,:), 'k')
% Plot the masses
plot(r1(1),r1(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r1(1),r1(2)+0.1,'$\quad m_1$', 'Interpreter', 'latex')
plot(r2(1),r2(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r2(1)-0.1,r2(2)+0.1,'$\quad m_2$', 'Interpreter', 'latex')
plot(r3(1),r3(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r3(1),r3(2),'$\quad m_3$', 'Interpreter', 'latex')
plot(0,0,'k*')
axis([-1 2 -1.5 1.5])
grid(gca,'minor')
grid on
title('Particle closed trajectories in $T = 2s$')
```

```
xlabel('x')
ylabel('y')

% Define approximated initial condition
u0 = [-pi/8 ; 1.9];
% Solve with Newton for the solution
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);

th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta); % display results

% Compute the trajectory for this solution
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
hh = Ta/N;
[~, Zh1] = rk4(f, 0, hh, z0, N);

% Compute solution for m1
u0 = [3*pi/8 ; 1.9];
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);
th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta);

% Compute new trajectory for m1
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
hh = Ta/N;
[~, Zh2] = rk4(f, 0, hh, z0, N);

% Compute solution for m2
u0 = [7*pi/4 ; 1.9];
[ukVec, ~, ~] = newtonn(u0, 1e-6, 100, @fun);
th0 = ukVec(1, end); Ta = ukVec(2, end);
disp(th0); disp(Ta);

% Compute new trajectory for m2
z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
hh = Ta/N;
[~, Zh3] = rk4(f, 0, hh, z0, N);

figure(4)
subplot(1, 2, 2)
hold on
% Plot trajectories
plot(Zh1(1,:), Zh1(2,:), 'k')
plot(Zh2(1,:), Zh2(2,:), 'k')
plot(Zh3(1,:), Zh3(2,:), 'k')
% Plot the masses
plot(r1(1),r1(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r1(1),r1(2)+0.1,'$\quad m_1$', 'Interpreter', 'latex')
plot(r2(1),r2(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r2(1)-0.1,r2(2)+0.1,'$\quad m_2$', 'Interpreter', 'latex')
plot(r3(1),r3(2),'ko', 'MarkerFaceColor', [0 0.5 0.5])
text(r3(1),r3(2),'$\quad m_3$', 'Interpreter', 'latex')
plot(0,0,'k*')
axis([-1 2 -1.5 1.5])
```

```
grid(gca,'minor')
grid on
title('Particle closed trajectories in $T = 2s$')
xlabel('x')
ylabel('y')
hold off
```



By comparing both subplots, we can see that, as we suspected, we do see that there are two values θ_0 and T_a for which we could obtain each trajectory. We can say, then, that the found trajectories are reversible, so that is another trait they have in common. To obtain them, we only needed to find the other valid initial condition so our solution could converge to the reversed one of the one obtained previously.

We can say that these trajectories are the same, not only due to the graphic representation of it, but also because it takes the exact same time for the particle of mass m to reach the origin again.

Auxiliar codes

Due to some problems when publishing the script, if we don't comment the auxiliar codes, the last figure (figure 3) does not appear.

```
% RK4   Runge-Kutta solution for y' = f(t,y) with y(a) = ya.
% Sample call   [T,Y] = rk4('f',a,h,ya,m)
% Inputs:   f    name of the function
%           a    initial time
%           h    time step
%           ya   initial value
```

```
%           m    number of steps
% Return:   T    solution: vector ( 1 x m )          of abscissas
%           Y    solution: matrix ( length(ya) x m ) of ordinates
function [T,Y] = rk4(f,a,h,ya,m)
    T = zeros(1,m+1);
    Y = zeros(length(ya),m+1);
    T(1) = a; Y(:,1) = ya;
    for j=1:m
        tj = T(j); yj = Y(:,j);
        k1 = h*feval(f,tj,yj);
        k2 = h*feval(f,tj+h/2,yj+k1/2);
        k3 = h*feval(f,tj+h/2,yj+k2/2);
        k4 = h*feval(f,tj+h,yj+k3);
        Y(:,j+1) = yj + (k1 + 2*k2 + 2*k3 + k4)/6;
        T(j+1) = a + h*j;
    end
end

% AB4 Adams-Bashforth solution for y' = f(t,y) with with y(t0) =
 y0(1) ;
% y(t0+h) = y0(2) ; y(t0+2h) = y0(3) ; y(t0+3h) = y0(4).
% Sample call: [T,Y] = rk4('f',a,h,ya,m)
% Inputs:   f    name of the function
%           t0    initial time
%           h    time step
%           y0   initial values
%           m    number of steps
% Return:   T    solution: vector ( 1 x m )          of abscissas
%           Y    solution: matrix ( length(ya) x m ) of ordinates
function [T,Y] = ab4(f,t0,h,y0,m)
    T = zeros(1,m+1); T(1:4) = t0 + [0 h 2*h 3*h] ;
    Y = zeros(length(y0(:,1)),m+1);
    Y(:,1) = y0(:,1); F0 = feval(f,T(1),Y(:,1));
    Y(:,2) = y0(:,2); F1 = feval(f,T(2),Y(:,2));
    Y(:,3) = y0(:,3); F2 = feval(f,T(3),Y(:,3));
    Y(:,4) = y0(:,4); F3 = feval(f,T(4),Y(:,4));

    for j=4:m
      T(j+1) = t0 + h*j;
      Y(:,j+1) = Y(:,j) + h*((-3/8)*F0+(37/24)*F1-
(59/24)*F2+(55/24)*F3);
      Y(:,j-3) = Y(:,j-2); Y(:,j-2) = Y(:,j-1); Y(:,j-1) = Y(:,j);
 Y(:,j) = Y(:,j+1);
      F0 = F1 ; F1 = F2 ; F2 = F3 ; F3 = feval(f,T(j+1),Y(:,j+1));
    end
end

% This functions returns the final position of the mass after u(2)
 seconds
% with an initial u(1) angle.
function r = fun(u)
    th0 = u(1); Ta = u(2);

    global f
```

```
    z0 = [0 ; 0 ; sqrt(2)*cos(th0) ; sqrt(2)*sin(th0)];
    N = 1000;
    hh = Ta/N;
    [~, Zh] = rk4(f, 0, hh, z0, N);

    r = Zh(1:2, end);
end

% Code 19: Computation of the Jacobian J
% Input:   F(x) : R^m ---> R^n
%          x : (m x 1)-vector ; F: (n x 1)-vector
% Output: DF(x) (n x m) Jacobian matrix at x
function DF = jac(F,x)
    f1 = feval(F,x);
    n = length(f1);
    m = length(x);

    DF = zeros(n,m);
    H = sqrt(eps)*eye(m);

    for j = 1:m
        f2 = feval(F,x+H(:,j));
        DF(:,j) = (f2 - f1)/H(j,j);
    end
end

% Code 13: PA = LU factorization (partial pivoting)
% Input: A (non-singular square matrix)
% Output: L (unit lower triangular matrix)
%         U (upper triangular matrix)
%         P (reordering vector)
function [P, L, U] = pplu(A)
    [m,n] = size(A);

    if m~=n
           error('not square matrix');
    end

    U = A;
    L = eye(n);

    P = [1:n]';

    for k = 1:n-1
        [~, imax] = max(abs(U(k:end,k)));
        imax = imax+k-1;
        i1 = [k, imax];
        i2 = [imax, k];

        U(i1,:) = U(i2,:); % Column k will be column imax and column
 imax will be column k
        P(k) = imax;
```

```
        L(i1,1:k-1) = L(i2, 1:k-1);

        for jj = [k+1:n]
            L(jj, k) = U(jj, k)/U(k, k);
            U(jj, k:n) = U(jj, k:n) - L(jj, k)*U(k,k:n);
        end
    end
end

% Code 11: Forward Substitution for Lower Triangular Systems
% Input:    L: Low Triangular non-singular square matrix
%           b: column right-hand side
% Output:   x: solution of Lx=b
function x = fs(L, b)
    x = 0*b;
    n = length(b);
    x(1) = b(1)/L(1,1);

    for ii = 2:n
        x(ii) = (b(ii)-L(ii, 1:ii-1)*x(1:ii-1))/L(ii,ii);
    end
end

% Code 12: Backward Substitution for Upper Triangular Systems
% Input:    U: Upp. Triangular non-singular square matrix
%           b: column right-hand side
% Output:   x: solution of Ux=b
function x = bs(U, b)
    x = 0*b;
    n = length(b);
    x(n) = b(n)/U(n,n);

    for ii = n-1:-1:1
        x(ii) = (b(ii)-U(ii, ii+1:n)*x(ii+1:n))/U(ii,ii);
    end
end

% Code 14: PA = LU (Solver for Ax = b)
% Input:    L (unit lower triangular matrix)
%           U (upper triangular matrix)
%           P (reordering vector)
%           b (right-hand side)
% Output:   solution x
function x = plusolve(L, U, P, b)
    n = length(b);
    for k = 1:n-1
        b([k P(k)]) = b([P(k) k]);
    end
    y = fs(L, b);
    x = bs(U, y);
end

% Code 20: Newtonis method for n-dimensional systems
% Input: x0 - initial guess (column vector)
```

```
%        tol - tolerance so that ||x_{k+1} - x_{k} || < tol
%        itmax - max number of iterations
%        fun - function's name
% Output:   XK - iterated
%           resd: resulting residuals of iteration: ||F_k||
%           it:   number of required iterations to satisfy tolerance
function [XK,resd,it] = newtonn(x0,tol,itmax,fun)
    xk = [x0];
    resd = [norm(feval(fun,xk))];
    XK = [x0];
    it = 1;

    tolk = 1.0;
    n = length(x0);

    while it < itmax && tolk > tol
        Fk = feval(fun, xk);

        DFk = jac(fun, xk);
        [P,L,U] = pplu(DFk);

        dxk = plusolve(L,U,P,-Fk);

        xk = xk + dxk;
        XK = [XK xk];
        resd = [resd norm(Fk)];
        tolk = norm(XK(:, end)-XK(:, end-1));
        it = it + 1;
    end
end
```

Published with MATLAB® R2020b