```
% Computing the evolution of the coordinates of two disks falling
% un1der the action of gravity and against drag forces while colliding
% with each other and the walls.
%
% The velocity vector (u, v) obeys the following ODE:
% du/dt =     - cd/m * velocityMagnitude1 * u
% dv/dt = - g - cd/m * velocityMagnitude1 * v
%
% dx/dt = u
% dy/dt = v
%
% The size of the domain should be set with variables xmax and ymax. We
% assume that the origin is at (0, 0).

clear variables; close all; clf;    % Clear all variables so we do not
 consider
                    % values from previous simulations.

tn = 0; tfinal = 5;
dtGlobal = 0.02;              % Time step.
un1 = -10;              % Initial x-component of the velocity vector for disk
 1.
vn1 = 50;               % Initial y-component of the velocity vector for disk
 1.
un2 = 11;               % Initial x-component of the velocity vector for disk
 2.
vn2 = 20;               % Initial y-component of the velocity vector for disk
 2.
r  = 0.05;              % Radius of the disk.
xmax = 1;               % Computational domain range in x.
ymax = 1;               % Computational domain range in y.
xn1 = 0.75;             % Initial x-coordinate of the disk 1.
yn1 = 5*r;              % Initial y-coordinate of the disk 1.
xn2 = 0.25;             % Initial x-coordinate of the disk 2.
yn2 = 5.5*r;            % Initial y-coordinate of the disk 2.
g  = 9.81;              % Gravity constant.
m  = 70;                % Mass of the object.
cd = 0.25;              % Drag coefficient in the air.
alpha = 0.80;           % Damping factor  (normal  to the wall).
beta  = 0.98;           % Friction factor (tangent to the wall).

%%%%%%%%%%%%%%%%%%%%%% SIMULATION %%%%%%%%%%%%%%%%%%%%%%%
dt = dtGlobal;      % Record the global time step so it can be reset
                    % at every time step.

xn1p1 = xn1;
yn1p1 = yn1;
un1p1 = un1;
vn1p1 = vn1;

xn2p1 = xn2;
yn2p1 = yn2;
```

```
un2p1 = un2;
vn2p1 = vn2;


% Plot initial position:
drawDisk(xn1, yn1, r, 'b'); hold on;
drawDisk(xn2, yn2, r, 'r');
s = sprintf('Simulation of a dropping disk at time %2.2f', tn);
title(s);
axis equal; axis([0 xmax 0 ymax]); fixfig;
pause();

% Advance in time
while tn < tfinal
    if tn + dt > tfinal         % Reaching the exact final time.
        dt = tfinal - tn;       % Reaching the exact final time.
    end                         % Reaching the exact final time.
    clf;
    % Apply the Euler rule to predict the velocity at time tnp1 = tn + dt:
    velocityMagnitude1 = sqrt( un1 * un1 + vn1 * vn1 );

    velocityMagnitude2 = sqrt( un2 * un2 + vn2 * vn2 );

    un1p1Prediction = un1 + dt * (     - cd/m * velocityMagnitude1 * un1 );
    un2p1Prediction = un2 + dt * (     - cd/m * velocityMagnitude2 * un2 );

    vn1p1Prediction = vn1 + dt * ( - g - cd/m * velocityMagnitude1 * vn1 );
    vn2p1Prediction = vn2 + dt * ( - g - cd/m * velocityMagnitude2 * vn2 );

    % Apply the Euler rule to predict the new location:
    xn1p1Prediction = xn1 + dt*un1;
    yn1p1Prediction = yn1 + dt*vn1;

    xn2p1Prediction = xn2 + dt*un2;
    yn2p1Prediction = yn2 + dt*vn2;

    isWallCollision = false;


    % Treat collision for both disks with the right wall:
    if ( xn1p1Prediction > xmax - r )      % collision occurs

        isWallCollision = true;

        % Shrink time step to land the disk on the right wall:
        dt = abs(xmax - xn1 - r) / abs(un1);

        % Put the disk on the right wall
        xn1p1 = xn1 + dt * un1;
        yn1p1 = yn1 + dt * vn1;

        % Compute the velocity when the disk reaches the right wall:
        velocityMagnitude1 = sqrt( un1 * un1 + vn1 * vn1 );
        un1p1 = un1 + dt * (     - cd/m * velocityMagnitude1 * un1 );
```

```
    vn1p1 = vn1 + dt * ( - g + cd/m * velocityMagnitude1 * vn1 );

    % Prepare for the next step by accoun1ting for the reboun1d:
    un1p1 = - alpha * un1p1;       % Damping
    vn1p1 =   beta  * vn1p1;       % Friction

end

if ( xn2p1Prediction > xmax - r )       % collision occurs

    isWallCollision = true;

    % Shrink time step to land the disk on the right wall:
    dt = abs(xmax - xn2 - r) / abs(un2);

    % Put the disk on the right wall
    xn2p1 = xn2 + dt * un2;
    yn2p1 = yn2 + dt * vn2;

    % Compute the velocity when the disk reaches the right wall:
    velocityMagnitude2 = sqrt( un2 * un2 + vn2 * vn2 );
    un2p1 = un2 + dt * (     - cd/m * velocityMagnitude2 * un2 );
    vn2p1 = vn2 + dt * ( - g + cd/m * velocityMagnitude2 * vn2 );

    % Prepare for the next step by accoun1ting for the reboun1d:
    un2p1 = - alpha * un2p1;       % Damping
    vn2p1 =   beta  * vn2p1;       % Friction

end

% Treat collision with the left wall:
if ( xn1p1Prediction < r )       % collision occurs

    isWallCollision = true;

    % Shrink time step to land the disk on the left wall:
    dt = abs(xn1 - r) / abs(un1);

    % Put the disk on the left wall
    xn1p1 = xn1 + dt * un1;
    yn1p1 = yn1 + dt * vn1;

    % Compute the velocity when the disk reaches the left wall:
    velocityMagnitude1 = sqrt( un1 * un1 + vn1 * vn1 );
    un1p1 = un1 + dt * (     - cd/m * velocityMagnitude1 * un1 );
    vn1p1 = vn1 + dt * ( - g + cd/m * velocityMagnitude1 * vn1 );

    % Prepare for the next step by accoun1ting for the reboun1d:
    un1p1 = - alpha * un1p1;       % Damping
    vn1p1 =   beta  * vn1p1;       % Friction

end

if ( xn2p1Prediction < r )       % collision occurs
```

```
        isWallCollision = true;

        % Shrink time step to land the disk on the left wall:
        dt = abs(xn2 - r) / abs(un2);

        % Put the disk on the left wall
        xn2p1 = xn2 + dt * un2;
        yn2p1 = yn2 + dt * vn2;

        % Compute the velocity when the disk reaches the left wall:
        velocityMagnitude2 = sqrt( un2 * un2 + vn2 * vn2 );
        un2p1 = un2 + dt * (     - cd/m * velocityMagnitude2 * un2 );
        vn2p1 = vn2 + dt * ( - g + cd/m * velocityMagnitude2 * vn2 );

        % Prepare for the next step by accoun1ting for the reboun1d:
        un2p1 = - alpha * un2p1;      % Damping
        vn2p1 =   beta  * vn2p1;      % Friction

    end

    % Treat collision with the bottom wall:
    if ( yn1p1Prediction < r )      % collision occurs

        isWallCollision = true;

        % Shrink time step to land the disk on the bottom wall:
        dt = abs(yn1 - r) / abs(vn1);

        % Put the disk on the bottom wall
        xn1p1 = xn1 + dt * un1;
        yn1p1 = yn1 + dt * vn1;

        % Compute the velocity when the disk reaches the bottom wall:
        velocityMagnitude1 = sqrt( un1 * un1 + vn1 * vn1 );
        un1p1 = un1 + dt * (     - cd/m * velocityMagnitude1 * un1 );
        vn1p1 = vn1 + dt * ( - g + cd/m * velocityMagnitude1 * vn1 );

        % Prepare for the next step by accoun1ting for the reboun1d:
        un1p1 =   beta  * un1p1;      % Friction
        vn1p1 = - alpha * vn1p1;      % Damping

    end

    if ( yn2p1Prediction < r )      % collision occurs

        isWallCollision = true;

        % Shrink time step to land the disk on the bottom wall:
        dt = abs(yn2 - r) / abs(vn2);

        % Put the disk on the bottom wall
        xn2p1 = xn2 + dt * un2;
        yn2p1 = yn2 + dt * vn2;
```

```
        % Compute the velocity when the disk reaches the bottom wall:
        velocityMagnitude2 = sqrt( un2 * un2 + vn2 * vn2 );
        un2p1 = un2 + dt * (     - cd/m * velocityMagnitude2 * un2 );
        vn2p1 = vn2 + dt * ( - g + cd/m * velocityMagnitude2 * vn2 );

        % Prepare for the next step by accoun1ting for the reboun1d:
        un2p1 =   beta  * un2p1;       % Friction
        vn2p1 = - alpha * vn2p1;       % Damping

    end

    % Treat collision with the top wall:
    if ( yn1p1Prediction > ymax - r )      % collision occurs

        isWallCollision = true;

        % Shrink time step to land the disk on the top wall:
        dt = abs(ymax - yn1 - r) / abs(vn1);

        % Put the disk on the top wall
        xn1p1 = xn1 + dt * un1;
        yn1p1 = yn1 + dt * vn1;

        % Compute the velocity when the disk reaches the top wall:
        velocityMagnitude1 = sqrt( un1 * un1 + vn1 * vn1 );
        un1p1 = un1 + dt * (     - cd/m * velocityMagnitude1 * un1 );
        vn1p1 = vn1 + dt * ( - g + cd/m * velocityMagnitude1 * vn1 );

        % Prepare for the next step by accoun1ting for the reboun1d:
        un1p1 =   beta  * un1p1;       % Friction
        vn1p1 = - alpha * vn1p1;       % Damping

    end

    if ( yn2p1Prediction > ymax - r )      % collision occurs

        isWallCollision = true;

        % Shrink time step to land the disk on the top wall:
        dt = abs(ymax - yn2 - r) / abs(vn2);

        % Put the disk on the top wall
        xn2p1 = xn2 + dt * un2;
        yn2p1 = yn2 + dt * vn2;

        % Compute the velocity when the disk reaches the top wall:
        velocityMagnitude2 = sqrt( un2 * un2 + vn2 * vn2 );
        un2p1 = un2 + dt * (     - cd/m * velocityMagnitude2 * un2 );
        vn2p1 = vn2 + dt * ( - g + cd/m * velocityMagnitude2 * vn2 );

        % Prepare for the next step by accoun1ting for the reboun1d:
        un2p1 =   beta  * un2p1;       % Friction
        vn2p1 = - alpha * vn2p1;       % Damping
```

```
    end


    % Disk collision between two disks:
    length = sqrt((xn1-xn2)^2+(yn1-yn2)^2);
    unitNormalVector = [(xn1-xn2)/length;(yn1-yn2)/length];

    unitTangentVector = [(yn1-yn2)/length; -(xn1-xn2)/length];

    predictionLength = sqrt((xn1p1Prediction-
xn2p1Prediction)^2+(yn1p1Prediction-yn2p1Prediction)^2);

    if (predictionLength <= 2*r)

    isWallCollision = true;
    % shrink dt step:
    dt = abs((length-2*r)/(sqrt((un1-un2)^2+(vn1-vn2)^2)));

    % Collide disks:
    xn1 = xn1 + dt * un1;
    xn2 = xn2 + dt * un2;
    yn1 = yn1 + dt * vn1;
    yn2 = yn2 + dt * vn2;

    % New velocity vectors:
    v1In = [un1; vn1];
    v2In = [un2; vn2];

    v1In = dot(v1In,unitNormalVector)*unitNormalVector + dot(v1In,
 unitTangentVector)*unitTangentVector;
    v2In = dot(v2In,unitNormalVector)*unitNormalVector + dot(v2In,
 unitTangentVector)*unitTangentVector;

    v1Out = dot(v2In,unitNormalVector)*unitNormalVector + dot(v1In,
 unitTangentVector)*unitTangentVector;
    v2Out = dot(v1In,unitNormalVector)*unitNormalVector + dot(v2In,
 unitTangentVector)*unitTangentVector;

    un1p1 = dot(v1Out,[1;0]);
    vn1p1 = dot(v1Out,[0;1]);

    un2p1 = dot(v2Out,[1;0]);
    vn2p1 = dot(v2Out,[0;1]);

    xn1p1 = xn1 + un1p1*dt;
    yn1p1 = yn1 + vn1p1*dt;

    xn2p1 = xn2 + un2p1*dt;
    yn2p1 = yn2 + vn2p1*dt;


    end
```

```
    % If the disk is not colliding with a wall, our predictions were
    % correct:
    if ( ~isWallCollision )
        xn1p1 = xn1p1Prediction;
        yn1p1 = yn1p1Prediction;
        un1p1 = un1p1Prediction;
        vn1p1 = vn1p1Prediction;

        xn2p1 = xn2p1Prediction;
        yn2p1 = yn2p1Prediction;
        un2p1 = un2p1Prediction;
        vn2p1 = vn2p1Prediction;
    end

    % Update the simulation time:
    tnp1 = tn + dt;
    dt = dtGlobal;

    % Plot the disk at its new location:
    drawDisk(xn1p1, yn1p1, r, 'b'); hold on;
    drawDisk(xn2p1, yn2p1, r, 'r');
    s = sprintf('Simulation of a dropping disk at time %2.2f', tnp1);
    title(s);
    axis equal; axis([0 xmax 0 ymax]); fixfig; pause(dtGlobal);

    % Prepare for the next time step.
    un1 = un1p1;
    vn1 = vn1p1;
    xn1 = xn1p1;
    yn1 = yn1p1;

    un2 = un2p1;
    vn2 = vn2p1;
    xn2 = xn2p1;
    yn2 = yn2p1;

    tn = tnp1;

end
```