Object moving in scene subject to gravity and bouncing

Assume no air resistance. Here we consider the change in the vertical component of velocity, and assume that the horizontal component of velocity is constant.

For use of image or imshow functions to display image, the positive vertical position coordinate *y* points down and is proportional to the image array row number, and the value of *g* is positive. An object moving upward in scene has a negative value of velocity.

For use of plot function to display image, the positive *y* coordinate points up and the value of *g* is negative. An object moving upward in scene has a positive value of velocity.

We proceed from the definitions of velocity and acceleration:

velocity (vertical component) $= v = \frac{dy}{dt}$

acceleration $= \frac{dv}{dt} = \frac{d^2 y}{dt^2}$

$\frac{dv}{dt} = g$ for gravitational acceleration

$\int_{v_0}^{v} d\tilde{v} = g \int_{0}^{t} d\tilde{t}$ separate variables and integrate

$v = v_0 + g\,t$

$v = \frac{dy}{dt} = v_0 + g\,t$ Ordinary Differential Equation (ODE)

$\int_{y_0}^{y} d\tilde{y} = \int_{0}^{t} (v_0 + g\,t)\, d\tilde{t}$ separate variables and integrate

$y = y_0 + v_0\,t + \frac{1}{2} g\,t^2$ analytical solution is possible for this simple case

For computing vertical coordinate *y* values during an animation with constant time step *dt*, we can use the following algorithm at each time step in a repeat structure:

- compute new *y* value for the current value of *t*
- increment the value of *t* by the time step *dt*: that is, *t = t + dt*

The new y value can be computed in several ways, e.g.,

- Using Euler's method of solution of $dy/dt = v_0 + g\,t$
- Using the analytical solution for this case, $y = y_0 + v_0\,t + \frac{1}{2} g\,t^2$

For computing vertical coordinate y values during an animation with constant time step dt, we don't have to keep track of the time values in an array, since we know how many positions we compute and we know the time step. For more solution of more complex cases, we may need to vary the time step and, thus, would want to keep track of the t values in an array.

Consider what happens when a falling object hits a surface and bounces. For an "elastic collision" in which no energy is lost, the sign of the object's vertical velocity simply changes sign. For an "inelastic collision" in which energy is transferred from the object to the surface, we can set the absolute value of the new velocity to some fraction of the absolute value of the velocity before the collision.

During the repeat that is stepping in time, we can add a check to see if the y coordinate has reached the coordinate position of a surface. When this happens, we can do the following:

- set the new value of v_0 to the negative of the current v value, or to some fraction (v can be computed from the current v_0 value and the current t).
- set the value of t to zero
- set the value of y_0 to the current y value
- continue repeating this series of actions until the next surface is hit

For a ball bouncing down a flight of stairs, you could define a stair coordinate array that contains the vertical coordinate values of each step. Update a counter variable when each step is hit so that you can check for collision with the next element in the stair coordinate array. This way you only need one repeat structure.

PART 2 - Example program using image or imshow functions

Object initially is falling in scene. Assumptions:

- initial vertical velocity = 0
- elastic collision when object hits the floor and bounces
- no air resistance or other friction

The positive y coordinate points down and is proportional to the image array row number. An object moving downward in scene has a positive value of velocity.

t_1 = time when object starts to fall
y_1 = vertical (row) coordinate when object starts to fall
$v_1 = 0$ = vertical velocity when object starts to fall
t_2 = time when object hits floor
y_2 = vertical coordinate when object hits floor
t_3 = time when object reaches maximum height of trajectory
$y_3 = y_1$ = vertical coordinate when object is at maximum height of trajectory
t_4 = time when object hits floor the second time
$y_4 = y_2$ = vertical coordinate when object hits floor the second time

Force = mass * acceleration = mg

$$\text{velocity} = v = \frac{dy}{dt}$$

$$\text{acceleration} = \frac{dv}{dt} = \frac{d^2 y}{dt^2}$$

$$\frac{dv}{dt} = g$$

$$dv = g\, dt$$

$$\int_{v_1}^{v} d\tilde{v} = g \int_{t_1}^{t} d\tilde{t}$$

$$v - v_1 = g(t - t_1) \quad ; \quad v = g(t - t_1)$$

$$v = \frac{dy}{dt} = g(t - t_1)$$

$$\int_{y_1}^{y} d\tilde{y} = g \int_{t_1}^{t} (\tilde{t} - t_1)\, d\tilde{t}$$

$$y = y_1 + \frac{1}{2} g(t^2 - t_1^2) - g\, t_1 (t - t_1)$$

Given g, t_1, y_1, and $y_2 = y$ of floor - height of object, we can solve for t_2, the time when the bottom of the object hits the floor...

... or we can just have our program detect the collision and set $t_2 = t$ at $y = y_2$.

The instant before the object hits the floor, its velocity = $v_{2-} = g(t_2 - t_1) > 0$. The instant after the object hits the floor, its velocity = $v_{2+} = -v_{2-} = -g(t_2 - t_1) < 0$. That is, the object reverses direction in the assumed elastic collision, with no loss of energy. If you wish to have a loss of energy, then reduce the velocity after the collision by some factor. Remember that kinetic energy is proportional to the square of velocity.

v – $v_{2+} = g(t - t_2)$

$$v = -g(t_2 - t_1) + g(t - t_2)$$

Note that $v = 0$ when $(t_3 - t_2) = (t_2 - t_1)$, that is, the object reaches the top of its trajectory after bouncing, and that the time to fall equals the time to rise to the top.

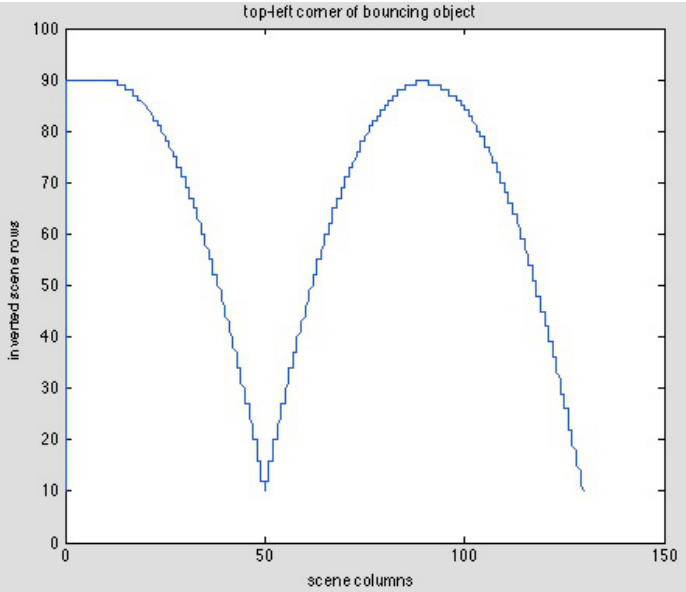
$$v = \frac{dy}{dt} = -g(t_2 - t_1) + g(t - t_2)$$

$$\int_{y_2}^{y} d\tilde{y} = -g(t_2 - t_1) \int_{t_2}^{t} d\tilde{t} + g \int_{t_2}^{t} (\tilde{t} - t_2)\, d\tilde{t}$$

$$y - y_2 = -g(t_2 - t_1)(t - t_2) + \frac{1}{2} g(t^2 - t_2^2) - g t_2 (t - t_2)$$

This works from t_2 and y_2 at the bounce, through the maximum height at t_3 and $y_3 = y_1$, to t_4 and $y_4 = y_2$ at the second collision with the floor.

The Matlab plot and code below shown below suggests the above equations are correct. Feel free to derive yourself and double-check my work.




```
% Experimental code for developing bouncing ball.
% This is one alternative method of computing the path.
% In this alternative, we already had integrated
% the differential equations of motion and have algebraic
% solutions. Thus, we could use array expressions
% to compute results and not repeats.
% However, since we may want to do things along the
% way such as show images, we use repeats here.
% With either array equations or results of our repeats,
% we have the coordinates in arrays, so we could do the
% animation of the images in a separate repeat.
% Another alternative would be to integrate the
% differential equations approximately numerically
% each time through a repeat loop as we step in time, e.g.,
% using Euler's method: y(i+1) = y(i) + g*t(i)*dt, which is obtained
% after integrating dv/dt = d^2y/dt^2 = g once analytically
% to obtain dy/dt = g*t

% Dimensional units are not shown at this stage of development.

clear all
t1 = 10; % time at drop
y1 = 10; % initial y (row) position, object top-left
x1 = 0; % initial x (column) position, object top-left
ymax = 100; % row height of of scene
yobj = 10; % row height of object
y2 = ymax-yobj; % y position of bottom of object on floor
```

```
g = 0.1; % vertical acceleration (length per time^2)
dt = 0.1; % time increment
xv = 1; % x velocity (length per time)
i = 1; % index i is our frame counter
t(i) = 0;
y(i) = y2;

% ball is rolling before dropping off bench
while t(i) < t1
    i = i+1;
    t(i) = t(i-1) + dt;
    y(i) = y1;
end

% now ball drops off bench
while y(i) < y2
    i = i+1;
    t(i) = t(i-1) + dt;
    y(i) = y1 + 0.5*g*(t(i)^2-t1^2)-g*t1*(t(i)-t1);
    y(i) = round(y(i)); % need integer for column
end
t2 = t(i);
% check to make sure don't go past floor
if y(i) > y2
    y(i) = y2;
end

% now ball bounces
while y(i) <= y2
    i = i+1;
    t(i) = t(i-1) + dt;
    y(i) = y2 + -g*(t2-t1)*(t(i)-t2) + ...
        0.5*g*(t(i)^2-t2^2)-g*t2*(t(i)-t2);
    y(i) = round(y(i));
end
t4 = t(i);
% since y(i) > y2 to get out of repeat
% we have gone past floor
if y(i) > y2
    y(i) = y2;
end

% compute x position from time
% and constant x-velocity
x = round(x1 + t*xv); % need integer for col

% need to add check that object stays
% on scene horizontally

% use plot() here to show results
% but plot() y points up and image
% rows point down, so here plot ymax-y vs. t
plot(x,ymax-y)
axis([0 max(x)+20 0 ymax])
title('top-left corner of bouncing object')
ylabel('inverted scene rows')
xlabel('scene columns')
```