Modelado de sistemas físicos - Caída libre

Segunda ley de Newton $F = m\,a$

$$a = \frac{F}{m} = \frac{F_d - F_u}{m}$$

F_d : fuerza hacia abajo debida a la gravedad

F_d : fuerza hacia arriba debida a la resistencia del aire (fricción)

Ecuación diferencial:

$$\frac{\mathrm{d}}{\mathrm{d}t}v = \frac{F_d - F_u}{m} = \frac{m\,g - c_d v^2}{m} = g - \frac{c_d}{m}v^2$$

Condición inicial:

$$v(0) = 0$$

Ecuación diferencial ordinaria (ODE) de primer orden, no lineal (v^2), no homogenea (g)

Solución analítica (exacta)

```
syms v(t) g cd m;
eqn = diff(v,t) == g - (cd/m)*v^2;
cond = v(0) == 0;
vSym(t) = dsolve(eqn,cond)      % tanh x = (e^x - e^-x)/(e^x + e^-x)
g = 9.81;        % m/s2; acceleration due to gravity
m = 68.1;        % kg; jumper's mass
cd = 0.25;       % kg/m; second-order drag coefficient

% evaluate vSym taking into account numerical values for g, m and cd
vel = subs(vSym)
tf = 12;                % tiempo de simulación
fplot(vel, [0 tf]);     % plots over the specified interval
title('Bungee Jumper Problem');
ylabel('v (m/s)');
xlabel('t (s)');
grid on;
hold on;
% convert symbolic expression to function handle
% velocity = matlabFunction(vel);
% fplot(velocity, [0,tf]);
```

Solución numérica (aproximada)

Euler method (Euler-Cauchy or point-slope)

First-order approximation for $v(t)$ (use of first derivative only)

The Taylor series of a function *f*(*x*) that is infinitely differentiable at a number *a* is the power series

$$f(x) = f(a) + \frac{d}{dx} f(a) \frac{(x-a)}{1!} + \frac{d^2}{dx^2} f(a) \frac{(x-a)^2}{2!} + \cdots$$

$$v(t) = v(t-h) + \frac{d}{dt} v(t-h) \frac{h}{1!} + \frac{d^2}{dt^2} v(t-h) \frac{h^2}{2!} + \cdots$$

$v_{i+1} = v_i + \frac{d}{dt} v_i * h$ (truncation)

Local trucation error: $O(h^2)$

Halving the step size will halve the global error $O(h)$

Approximation of $\frac{d}{dt}v$ with a (first forward) finite difference:

$$\frac{d}{dt} v = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$

$$v_{i+1} = v_i + \frac{d}{dt} v_i * (t_{i+1} - t_i)$$

(El valor de la derivada de una función en un punto es la pendiente de la recta tangente a la curva en ese punto: tangente = opuesto / adyacente, opuesto = tangente * adyacente)

$$v_{i+1} = v_i + \frac{d}{dt} v * (t_{i+1} - t_i)$$

new value = old value + slope*h; slope = dv/dt = a (ecuación algebraica)

$$v_{i+1} = v_i + \left(g - \frac{c_d}{m} v^2\right) * h$$

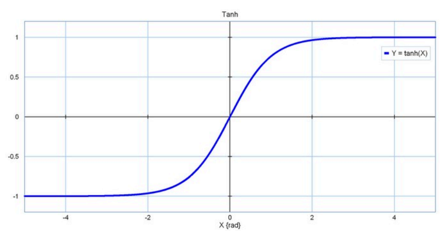
```
% Las funciones anónimas pueden utilizar cualquier variable que esté
% disponible en el espacio de trabajo actual.
a = @(v) g - (cd/m)*v^2;    % aceleración
h = 2;                      % (ti+1 - ti); step size s; probar con h=1 y 0.1
t = 0:h:tf;
v = zeros(size(t));     % preallocation of memory, v(0) = 0

for i=1:1:length(v)-1
    v(i+1) = v(i) + a(v(i))*h;
end
```

```
% Usando la funcion freefall
% [t,v] = freefall(h, tf, g, m, cd);

% Usando ivpsV(f, y0, t0, tf, h, solver)
% f = @(t,v) g - (cd/m)*v^2;
% [t,v] = ivpsV(f, 0, 0, tf, h, 'rk4');

plot(t,v);
% applies the formatSpec to all elements of array res in column order
res = [t; v];
fprintf('\tt        v\n');

fprintf('%5d %10.3f\n',res);
```

Estado estable

Físicamente, cuando la fuerza de arrastre (*drag*) es igual al peso ($c_d = m\,g$), la aceleración es cero, y la velocidad se vuelve constante.

Matemáticamente, cuando t aumenta, la función *tanh* tiende a 1 y la velocidad tiende a la velocidad terminal



$\sqrt{\frac{m\,g}{c_d}}$

```
terminalVelocity = sqrt(m*g/cd);
terminalVelocitykmh = terminalVelocity*(3600/1000)  % km/h
v = terminalVelocity*ones(size(t));
plot(t, v, '--b');
axis([0 tf 0 terminalVelocity*1.1]);
legend('exacta', 'aproximada', 'velocidad terminal');
legend('Location','southeast');
hold off;
```

Fórmulas clásicas de Cinemática:

```
% x0 posición inicial; v0 velocidad inicial
syms v(t) v0 a x(t) x0;
eqn = diff(x,t,2)==a;
Dx = diff(x,t);
cond = [x(0)==x0, Dx(0)==v0];
xSym(t) = dsolve(eqn,cond)
vSym(t) = diff(xSym(t),t)
% ¿En cuánto tiempo llega a x=0?
eqn = xSym(t)==0
xRoot = solve(eqn,t)
a = -9.81;
v0 = 0;
x0 = 45;
```

```
subs(xRoot)
res = double(subs(xRoot))
t = res(2)                      % s
v = double(subs(vSym))          % m/s
vKmH = v*(3600/1000)
```