```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Declare Frames, Points, and Objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NewtonianFrame N
RigidFrame A, B
Point N1(N), N2(N)
Particle Q

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Declare mathematical quantities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
variable x'', y''
constant mQ, g = 9.81 m/sec^2
constant thetaA = 20 degrees, thetaB = 45 degrees

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Rotational Kinematics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A.rotateNegativeZ(N, thetaA)
B.rotateNegativeZ(N, thetaB)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Translational Kinematics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Q.translate(N2, x*bx> + y*by> )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Q.addForce( mQ * g * (-ny>) )

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Dynamics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% If we did it by hand, we would expect to see:
%%% a_Q_N> = x'' * bx> + y'' * by>
%%% F_Q> = -mQ*g*ny>

%%% Form the vector equation of motion
ZeroNewton> = Q.getDynamics()

%%% Get scalar differential equations of motion.
system[1] = dot(ZeroNewton>, bx>)
system[2] = dot(ZeroNewton>, by>)

%%% Re-arrange those differential equations.
solve(system, x'', y'')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Prepare and Run Numerical Integration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Manually declare initial position and velocity.
r_N2_N1_i> = 3*ny>  %%% r^(N1/N2)
v_Q_N_i> = 10*bx>   %%% launch velocity

%%% Evaluate the initial conditions
input x = EvaluateAtInput(dot(r_N2_Q_i>, bx>))
input y = EvaluateAtInput(dot(r_N2_Q_i>, by>))
input x' = EvaluateAtInput(dot(v_Q_N_i>, bx>))
input y' = EvaluateAtInput(dot(v_Q_N_i>, by>))
input tFinal = 1.8, absError = 1e-5

%%% Run numerical integration
output t, x, y
ODE() ski_jump
```