12 Applying the Variational Principle to Impacts

(Last Update: 2020-10-21 21:59:58Z)

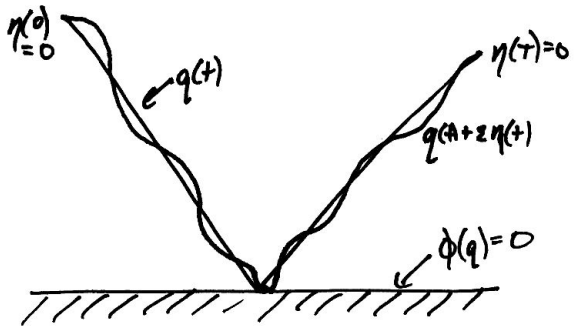


Figure 16: $q(t)$ and $\eta(t)$ have to both be in the free space where $\phi(q) \geq 0$, so the impact time and the impact location of the variations are constrained.

Suppose we have a system with Lagrangian $L(q, \dot{q})$ that is evolving over a trajectory $q(t)$ from time $t = 0$ to T. Assume, also, that the trajectory is continuous everywhere and differentiable everywhere, except at a point $\tau \in [0, T]$, at which a collision occurs, and a discontinuity in $\dot{q}$ is allowed. The action is

$$A(q, \dot{q}, \tau) = \int_0^T L(q, \dot{q})\, dt = \int_0^\tau L(q, \dot{q})\, dt + \int_\tau^T L(q, \dot{q})\, dt,$$

where, for reasons soon to become evident, we split the integral in the terms before and after the collision.

To find the equations of motion, we want to set δA to zero for all possible variations in the trajectory. To do this, we consider

$$A(\underbrace{q + \varepsilon\eta}_{q_\epsilon}, \underbrace{\dot{q} + \varepsilon\dot{\eta}}_{\dot{q}_\epsilon}, \underbrace{\tau + \varepsilon\theta}_{\tau_\epsilon}) = \int_0^{\tau_\varepsilon} L(q_\varepsilon, \dot{q}_\varepsilon)\, dt + \int_{\tau_\varepsilon}^T L(q_\varepsilon, \dot{q}_\varepsilon)\, dt,$$

and thus we have (where we will constrain η and θ to vary in a physically meaningful way later)

$$\begin{aligned}
\delta A \cdot (\eta, \theta)^{\mathrm{T}} &= \frac{d}{d\varepsilon}\left[\int_0^{\tau_\varepsilon} L(q_\varepsilon, \dot{q}_\varepsilon)\, dt + \int_{\tau_\varepsilon}^{T} L(q_\varepsilon, \dot{q}_\varepsilon)\, dt\right]_{\varepsilon=0} \\
&= \left[\int_0^{\tau^-}\left(\frac{\partial L}{\partial q_\varepsilon}\cdot\eta + \frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\dot{\eta}\right) dt + L(q_\varepsilon, \dot{q}_\varepsilon)\,\theta\Big|_{\tau^-} + \int_{\tau^+}^{T}\left(\frac{\partial L}{\partial q_\varepsilon}\cdot\eta + \frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\dot{\eta}\right) dt - L(q_\varepsilon, \dot{q}_\varepsilon)\,\theta\Big|_{\tau^+}\right]_{\varepsilon=0}
\end{aligned}$$

where the non-integral terms comes from a combination of Leibniz rule and chain rule

$$\begin{aligned}
&= \left[\int_0^{\tau^-}\left(\frac{\partial L}{\partial q_\varepsilon}\cdot\eta - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\eta\right) dt + \left(\frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\eta + L(q_\varepsilon, \dot{q}_\varepsilon)\,\theta\right)_{\tau^-}\right]_{\varepsilon=0} \\
&\qquad + \left[\int_{\tau^+}^{T}\left(\frac{\partial L}{\partial q_\varepsilon}\cdot\eta - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\eta\right) dt - \left(\frac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\eta + L(q_\varepsilon, \dot{q}_\varepsilon)\,\theta\right)_{\tau^+}\right]_{\varepsilon=0}
\end{aligned}$$

where the $\dfrac{\partial L}{\partial \dot{q}_\varepsilon}\cdot\eta$ terms *not* under the integral come from integration by parts and the fact that $\eta(\tau)$ need not be zero

$$= \int_0^{\tau^-}\left(\frac{\partial L}{\partial q}\cdot\eta - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}}\cdot\eta\right) dt - \left[\frac{\partial L}{\partial \dot{q}}\cdot\eta + L(q, \dot{q})\,\theta\right]_{\tau^-}^{\tau^+} + \int_{\tau^-}^{T}\left(\frac{\partial L}{\partial q}\cdot\eta - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}}\cdot\eta\right) dt.$$

Notice that the two integrals must be zero, as they are over intervals on which the trajectory is smooth and one can show by applying the variational principle, that the Euler-Lagrange equations must hold on those intervals. We have, then

$$\delta A \cdot (\eta, \theta)^{\mathrm{T}} = -\left[\frac{\partial L}{\partial \dot{q}}\cdot\eta + L(q, \dot{q})\,\theta\right]_{\tau^-}^{\tau^+},$$

which has to vanish for all possible (η, θ) pairs. The equation is evaluated only at the time of impact, for which we know that $\phi[q(\tau)] = 0$ — the system is on the contact manifold at the time of impact. The derivative of this relation must also hold, $\delta\phi[q(\tau)] = 0$. We calculate just as before

$$\begin{aligned}
\frac{d}{d\varepsilon}\phi[q_\varepsilon(\tau_\varepsilon)]\Big|_{\varepsilon=0} &= 0, \\
\frac{d}{d\varepsilon}\phi[q(\tau+\varepsilon\theta) + \varepsilon\eta(\tau+\varepsilon\theta)]\Big|_{\varepsilon=0} &= 0, \\
\frac{\partial\phi}{\partial q_\varepsilon(\tau_\epsilon)}\cdot\left[\frac{d}{d\epsilon}\left(q(\tau+\varepsilon\theta) + \varepsilon\eta(\tau+\varepsilon\theta)\right)\right]\Big|_{\varepsilon=0} &= 0,
\end{aligned}$$

where the "$\cdot$" symbol has been included to indicate that a row vector is being composed with a column vector.

$$\begin{aligned}
\frac{\partial\phi}{\partial q_\varepsilon(\tau_\epsilon)}\cdot\left[\dot{q}(\tau_\varepsilon)\theta + \underbrace{\eta(\tau_\varepsilon) + \varepsilon\dot{\eta}(\tau_\varepsilon)\theta}_{\text{result of product rule}}\right]\Big|_{\varepsilon=0} &= 0, \\
\frac{\partial\phi}{\partial q}\cdot[\dot{q}(\tau)\theta + \eta(\tau)] &= 0
\end{aligned}$$

Note now that the space of all possible $(\eta(\tau), \theta)$ has $n+1$ dimensions, since θ is a scalar. Imposing the previous condition reduces the dimensionality of the allowed combinations by one, unless $\nabla\phi = 0$ (which doesn't make sense if there is a constraint). Consider the pairs $(\eta(\tau), 0)$, with

$\frac{\partial \phi}{\partial q} \cdot \eta(\tau) = 0$. The space of $\eta(\tau)$ is n-dimensional, which implies that after applying the orthogonality condition we can have exactly $n-1$ linear independent pairs that satisfy it. It follows that any other pair of $(\eta(\tau), \theta)$ that is linearly independent of the previously described ones is sufficient to describe the space of allowable perturbations.[10] Let that pair be $(\dot{q}(\tau), -1)$. So the set of vectors in the space of (η, θ) pairs that spans the subspace that satisfies the constraint $\frac{\partial \phi}{\partial q}\left[\dot{q}(\tau)\theta + \eta(\tau)\right] = 0$ are:

$$(\eta, 0) \text{ such that } \frac{\partial \phi}{\partial q} \cdot \eta(\tau) = 0 \text{ and } (\dot{q}(\tau), -1)$$

[10] To see that this must be true, it is helpful to look at an example. Consider the problem of spanning the subspace of $\mathbb{R}^3$ such that $x + y = z$. The first approach would be to find by hand two vectors that are linearly independent that satisfy the constraint. For instance, setting $x = 0$ we would get the vetor $\{0, 1, 1\}$ and setting $y = 0$ we would get the vector $\{1, 0, 1\}$. These two are linearly independent and we can only be looking for a two dimensional space (since the constraint reduces the dimension by one).

That solution doesn't really look like our problem, so rewrite $x + y - z = 0$ to be $A(vx + (y, z)^T) = 0$ where $A = (1, -1)$ and $v = \{\frac{1}{2}, -\frac{1}{2}\}$. We need to construct two vectors (x_1, y_1, z_1) and (x_2, y_2, z_2) that span the space of solutions. Similar to the previous strategy, set $x_1 = 0$ and get a resulting constraint of the form $y_1 - z_1 = 0$, where a vector that spans this constraint is $(x_1, y_1, z_1) = (0, 1, 1)$. Now, we need one more vector. To get this vector, set $x_2 = -1$ and $va + b = 0$ to get $b = v$, yielding a vector of the form $(-1, v) = (-1, \frac{1}{2}, -\frac{1}{2})$. This is the same linear span as the previous solution.

13 Applying the Variational Principle to Impacts Continued

(Last Update: 2020-10-21 21:59:58Z)

The action principle becomes

$$0 = \delta A \cdot (\eta(\tau), 0)^{\mathrm{T}} = -\frac{\partial L}{\partial \dot{q}} \cdot \eta \Big|_{\tau^-}^{\tau^+}, \quad \forall \eta(\tau) \text{ s.t. } \frac{\partial \phi}{\partial q} \cdot \eta(\tau) = 0,$$
$$0 = \delta A \cdot (\dot{q}, -1)^{\mathrm{T}} = -\left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]_{\tau^-}^{\tau^+}$$

We can read the first equation as requiring that the operator $\frac{\partial L}{\partial \dot{q}}$ be orthogonal to all $\eta(\tau)$ that are orthogonal to $\frac{\partial \phi}{\partial q}$. This is equivalent to saying that the two operators lie in the same direction. The equations become

$$\frac{\partial L}{\partial \dot{q}} \Big|_{\tau^-}^{\tau^+} = \lambda \frac{\partial \phi}{\partial q},$$
$$\left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]_{\tau^-}^{\tau^+} = 0.$$

The first equation can be interpreted as restricting the change of momentum due to impact to lie perpendicular to the contact surface. The second equation simply states that the Hamiltonian (which is the energy in most cases) is conserved through the impact.

Example: Point mass in gravity.

The configuration variables are x and y, the Lagrangian is

$$L = \tfrac{1}{2} m(\dot{x}^2 + \dot{y}^2) - mgy.$$

Let the point hit a horizontal surface at $y = h$ from above, at time τ. We describe this surface by $\phi(x, y) = y - h$. The previously derived equations become

$$m\dot{x}(\tau^+) = m\dot{x}(\tau^-)$$
$$m\dot{y}(\tau^+) = m\dot{y}(\tau^-) + \lambda,$$
$$m\dot{x}^2(\tau^+) + m\dot{y}^2(\tau^+) = m\dot{x}^2(\tau^-) + m\dot{y}^2(\tau^-).$$

Here, $\dot{x}(\tau^-)$ and $\dot{y}(\tau^-)$ are the velocity of the point right before impact (known) and $\dot{x}(\tau^+)$ are $\dot{y}(\tau^+)$ the velocity right after impact (unknown). The equations have two solutions, but only one of them with a positive λ. Note that we restrict $\lambda \geq 0$ since the wall can only act away from its surface, and a negative λ would correspond to the wall sucking the point in. The solution is, unexpectedly,

$$\dot{x}(\tau^+) = \dot{x}(\tau^-),$$
$$\dot{y}(\tau^+) = -\dot{y}(\tau^-),$$
$$\lambda = -2m\dot{y}(\tau^-).$$

MS says: Example code starts here

However, the above are just analytical and symbolic solutions for the impact update, when we simulate the system, how do we include impact in the simulation? First let's simulate the particle falling in gravity without impact (which you should be familiar with now):

```python
import numpy as np
import sympy as sym
from sympy.abc import t
from sympy import Function, solve, Matrix, symbols
# define states
q = Matrix([Function('x')(t), Function('y')(t)])
qdot = q.diff(t)
qddot = qdot.diff(t)
# compute Lagrangian
m, g = 1, 9.8
ke = 0.5 * m * (qdot[0]**2+qdot[1]**2)
pe = m * g * q[1]
L = ke-pe
print('Lagrangian:')
display(L)
# compute EL-equations (here we use SymPy's built-in method for simplicity)
from sympy.calculus.euler import euler_equations
el_eqns = euler_equations(L, funcs=[q[0], q[1]], vars=[t])
el_eqns = sym.Eq(Matrix([-el_eqns[0].lhs, -el_eqns[1].lhs]), Matrix([el_eqns[0].
    rhs, el_eqns[1].rhs]))
print('EL-equation(s):')
display(el_eqns)
# numerical evaluation
qddot_soln = solve(el_eqns, [qddot[0], qddot[1]], dict=True)[0]
xddot_func = sym.lambdify([q[0], q[1], qdot[0], qdot[1]], qddot_soln[qddot[0]])
yddot_func = sym.lambdify([q[0], q[1], qdot[0], qdot[1]], qddot_soln[qddot[1]])
# define dynamics for the particle
def particle_dyn(s):
    return np.array([s[2], s[3], xddot_func(*s), yddot_func(*s)]) # argument '*s'
    is same as 's[0], s[1], s[2], s[3]'
s0 = np.array([1, 1, 0, 0])
print('[xdot, ydot, xddot, yddot] at initial condition [x={}, y={}, xdot={}, ydot
    ={}] : [{}, {}, {}, {}]'.format(*s0, *particle_dyn(s0)))
```

If there is no impact, then next step is to simulate the trajectory based on the dynamics function, below is the pseudo-code for simulation without impact:

Algorithm 1: Simulating Trajectory without Impact

Input : $dyn(s)$: first-order dynamics of the system
s_0: initial condition
N: length of simulation

Output: $traj$: vector/matrix representing the simulated trajectory

$t \leftarrow 0$ // initialize current time step
$traj \leftarrow [s_t]$ // initialize simulated trajectory
while $t < N$ **do**
  $s_{t+1} \leftarrow$ `integrate`(dyn, s_t)
  $t \leftarrow t + 1$
  `append`$(traj, s_t)$ // insert the current state at the end of $traj$
end while
return $traj$ // return results

However, if there is impact, then for some iterations of the loop in Algorithm 1, we can not simply obtain s_{t+1} as `integrate`(dyn, s_t)—before the integration, there should be an instantaneous

update to s_t caused by the impact. This update is directly related to our previous results:

$$\dot{x}(\tau^+) = \dot{x}(\tau^-),$$
$$\dot{y}(\tau^+) = -\dot{y}(\tau^-).$$

Based on this, we can write the pseudo-code for simulation with impact:

Algorithm 2: Simulating Trajectory with Impact

Input : $dyn(s)$: first-order dynamics of the system
s_0: initial condition
N: length of simulation

Output: $traj$: vector/matrix representing the simulated trajectory

$t \leftarrow 0$ // initialize current time step
$traj \leftarrow [s_t]$ // initialize simulated trajectory
while $t < N$ **do**
  if s_t *will cause impact* **then**
    $s_t \leftarrow$ impact_update(s_t) // instantaneous impact update: s_t on the right represents $s(\tau^-)$, the one on the left represents $s(\tau^+)$
    while s_t *will cause impact* **do**
      $s_t \leftarrow$ integrate(dyn, s_t) // in practice one integration should end the loop
    end while
    $s_{t+1} \leftarrow s_t$
  else
    $s_{t+1} \leftarrow$ integrate(dyn, s_t)
  end if
  $t \leftarrow t + 1$
  append$(traj, s_t)$ // insert the current state at the end of $traj$
end while
return $traj$ // return results

In practice, for the above pseudo-code to work, we first need to write a function to decide when the impact would happend based on current state s_t, in order to do it, we need to define the constraint related to the impact:

```
# define impact constraint (assuming the particle hits the ground at y=0)
phi = q[1]-0
# numerically evaluate the constraint
# note that even phi is not in terms of qdot, we still include them in lambdify
# for the simulation later (you can also exclude them)
phi_func = sym.lambdify([q[0], q[1], qdot[0], qdot[1]], phi)
# function for impact condition
# since we are numerically simulating the system, when impact happends, phi(s)
# will not be exactly zero, thus we need to either catch the change of sign in
# phi(q) or set a threshold for that
def impact_condition(s, threshold=1e-1):
    phi_val = phi_func(*s)
    if phi_val > -threshold and phi_val < threshold:
        return True
    return False
# test the impact condition function
s_test = [1, 0.01, 1, 1]
```

```
print('impact will happen at [x={}, y={}, xdot={}, ydot={}]: {}'.format(*s_test,
    impact_condition(s_test, 1e-1)))
```

After that, we need to have a (numerical) function which takes in a s_t and treats it as $s(\tau^-)$ to return $s(\tau^+)$. Recall that our equations for impact update are:

$$\frac{\partial L}{\partial \dot{q}}\Big|_{\tau^-}^{\tau^+} = \lambda \frac{\partial \phi}{\partial q},$$
$$\left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]_{\tau^-}^{\tau^+} = 0.$$

where $H(q, \dot{q}) = \left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]$ is the Hamiltonian of the system. We can explicitly re-write the equations as:

$$\frac{\partial L}{\partial \dot{q}}\left(q(\tau^+), \dot{q}(\tau^+)\right) - \frac{\partial L}{\partial \dot{q}}\left(q(\tau^-), \dot{q}(\tau^-)\right) = \lambda \cdot \frac{\partial \phi}{\partial q}\left(q(\tau^-)\right),$$
$$H\left(q(\tau^+), \dot{q}(\tau^+)\right) - H\left(q(\tau^-), \dot{q}(\tau^-)\right) = 0.$$

Based on the equations above, let's first evaluate the related expressions at τ^- and τ^+:

```
# define dummy variables for q(tau-), qdot(tau-), qdot(tau+)
# note that we don't need to define q(tau+), why?
xMinus, xdotMinus, yMinus, ydotMinus = symbols(r'x^-, \dot{x}^-, y^-, \dot{y}^-')
xdotPlus, ydotPlus = symbols(r'\dot{x}^+, \dot{y}^+')
# define substitution dictionary to evalute expressions at tau- and tau+
# note that in "subs_plus", we replace q[0] and q[1] with xMinus and yMinus (why?)
subs_minus = {q[0]:xMinus, q[1]:yMinus, qdot[0]:xdotMinus, qdot[1]:ydotMinus}
subs_plus = {q[0]:xMinus, q[1]:yMinus, qdot[0]:xdotPlus, qdot[1]:ydotPlus}
# compute dLdqdot and evaluate it at tau- and tau+
dLdqdot = Matrix([L]).jacobian(qdot)
dLdqdot_Minus = dLdqdot.subs(subs_minus)
dLdqdot_Plus = dLdqdot.subs(subs_plus)
# compute dPhidq and evaluate it ONLY at tau- (why?)
dPhidq = Matrix([phi]).jacobian(q)
dPhidq_Minus = dPhidq.subs(subs_minus)
# compute Hamiltonian and evaluate it at tau- and tau+
H = (dLdqdot*qdot)[0] - L
H_Minus = H.subs(subs_minus)
H_Plus = H.subs(subs_plus)
```

Note that here we use dummy symbolic variables to substitute the state variables previously defined as symbolic functions. After evaluation at τ^- and τ^+, we can easily write down the equations and solve them for $\dot{q}(\tau^+)$, the results will be the impact update rules:

```
# define equations for impact update
lamb = symbols('\lambda')
impact_eqns_lhs = Matrix([ dLdqdot_Plus[0]-dLdqdot_Minus[0], dLdqdot_Plus[1]-
    dLdqdot_Minus[1], H_Plus-H_Minus ])
impact_eqns_rhs = Matrix([ lamb*dPhidq_Minus[0], lamb*dPhidq_Minus[1], 0 ])
impact_eqns = sym.Eq(impact_eqns_lhs, impact_eqns_rhs)
print('equations for impact update:')
display(impact_eqns)
# solve them for qdot(tau+) and lamb
impact_solns = solve(impact_eqns, [xdotPlus, ydotPlus, lamb], dict=True)[0]
xdot_update_sol = impact_solns[xdotPlus]
ydot_update_sol = impact_solns[ydotPlus]
```

```
lamb_update_sol = impact_solns[lamb]
print('impact update solutions:')
display(sym.Eq(xdotPlus, xdot_update_sol))
display(sym.Eq(ydotPlus, ydot_update_sol))
display(sym.Eq(lamb, lamb_update_sol))
```

With the symbolic solutions for impact update, we can numerically evaluate them and define a function for the impact update in the simulation loop:

```
# numerically evaluate impact udpate solutions
xdot_update_func = sym.lambdify([xMinus, yMinus, xdotMinus, ydotMinus],
    xdot_update_sol)
ydot_update_func = sym.lambdify([xMinus, yMinus, xdotMinus, ydotMinus],
    ydot_update_sol)
# define impact update function
def impact_update(s):
    return np.array([
        s[0],
        s[1], # q will be the same after impact
        xdot_update_func(*s),
        ydot_update_func(*s)
    ])
# test impact update function
s_test = [1, 0.01, 1, 1]
print('impact update at [x={}, y={}, xdot={}, ydot={}]: [{}, {}, {}, {}]'.format(*
    s_test, *impact_update(s_test)))
```

Finally, everything is ready and all we need to do, is to combine the impact condition function and impact update function into the simulation as in Algorithm 2. Below is an example, but you can try your own implementation:

```
# define integrate function
def integrate(f, xt, dt):
    """
    RK4 integration
    """
    k1 = dt * f(xt)
    k2 = dt * f(xt+k1/2.)
    k3 = dt * f(xt+k2/2.)
    k4 = dt * f(xt+k3)
    new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
    return new_xt
# define simualate function
def simulate_with_impact(f, x0, tspan, dt, integrate):
    """
    simulate with impact
    """
    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    tvec = np.linspace(min(tspan),max(tspan),N)
    xtraj = np.zeros((len(x0),N))
    for i in range(N):
        if impact_condition(x) is True:
            x = impact_update(x)
            xtraj[:,i]=integrate(f,x,dt)
        else:
            xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return xtraj
# simulate the particle falling in gravity with impact
```

```
s0 = [0, 1, 0, 1]
traj = simulate_with_impact(particle_dyn, s0, tspan=[0,5], dt=0.01, integrate=
    integrate)
import matplotlib.pyplot as plt
plt.plot(np.arange(traj.shape[1]), traj[0:2].T)
plt.show()
```

Example: Let a pendulum have configuration θ measured from vertical on a wall, so that $\theta = 0$ at its stable equillibrium. Then we have

$$L = \frac{1}{2}mR^2\dot{\theta}^2 - gmR(1 - \cos(\theta))$$

where R is the length of the pendulum, m is its mass, and $g > 0$ is the gravitational constant. From these we get the continuous time Euler-Lagrange equations before the impact

$$\ddot{\theta} = -\frac{g}{R}\sin(\theta)$$

and momentum

$$p = \frac{\partial L}{d\dot{q}} = mR^2\dot{\theta}$$

and Hamiltonian

$$H = p\dot{q} - L = \frac{1}{2}mR^2\dot{\theta}^2 + gmR(1 - \cos(\theta)).$$

So, the impact law tells us that

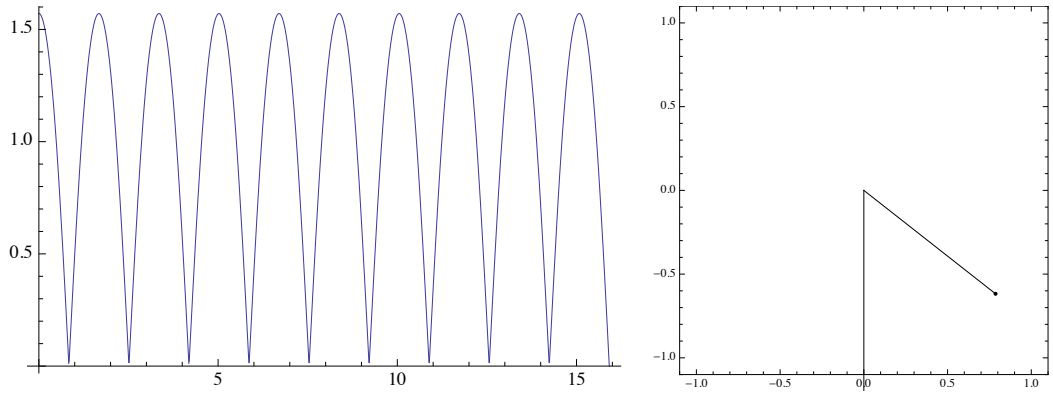
$$\left[\frac{\partial L}{d\dot{q}}\right]_{\tau^-}^{\tau^+} = \lambda(1)$$

$$\left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]_{\tau^-}^{\tau^+} = \left[mR^2\dot{\theta} \cdot \dot{\theta} - \frac{1}{2}mR^2\dot{\theta}^2 + gmR(1 - \cos(\theta))\right]_{\tau^-}^{\tau^+} = 0.$$

With these, we get an impact update law:

$$\dot{\theta}(\tau^+) = -\dot{\theta}(\tau^-)$$

when $\lambda \neq 0$. Simulating this system, we get the figures below.



(a) (b)

Figure 17: (a) simulation of θ versus t (b) a snapshot of the animation

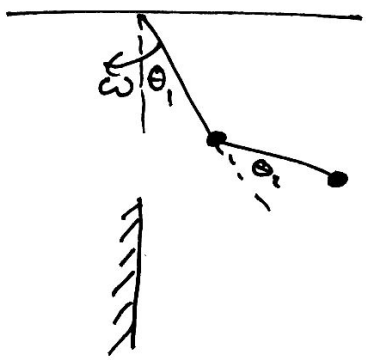


Figure 18: A driven double pendulum where $\dot{\theta}_1 = \omega$ impact a wall at $x_2 = 0$