Jan 28, 2026

2D Mass-Spring System

Simulation and Modeling (CSCI 3010U)

Faisal Qureshi

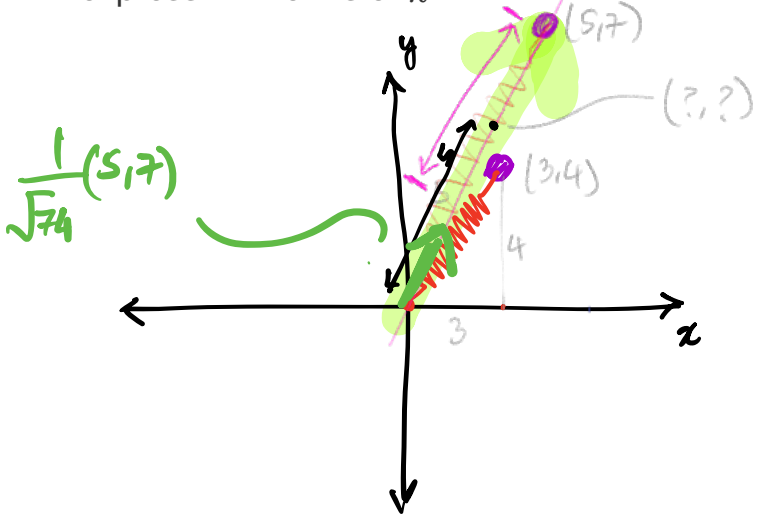
Faculty of Science, Ontario Tech University

http://vclab.science.ontariotechu.ca

Discuss in class

Consider a mass-spring system that lives in a flat (2D) world. One end of the spring is connected to a fixed hinge sitting at location $(0, 0)$. The other end of the spring is connected to a mass of value 1 kg. When the spring is at its rest length, the the mass is sitting at location $(3, 4)$. Consider that you've moved the mass to a new location $(5, 7)$, and answer the following questions. Let k represents the spring constant.

1. Is the spring extended or squished?
2. What do you think will happen if you release the mass at this point?
3. Compute the force acting on the mass when it is sitting at location $(5, 7)$. You'll need to express it in terms of k.



(i) Rest length of the spring

$= \sqrt{3^2+4^2}$

$= \sqrt{9+16}$

$= \sqrt{25}$

$= 5$

(ii) Current length of the spring

$= \sqrt{5^2+7^2}$

$= \sqrt{35+49}$

$= \sqrt{74}$

1. Extended.

2. Mass will oscillate along the line joining (0,0) and (5,7). The mass will oscillate around point where the distance from origin is 5.

3. Force on the mass due to a spring with spring constant k.

$$F = -kx$$

x = deformation

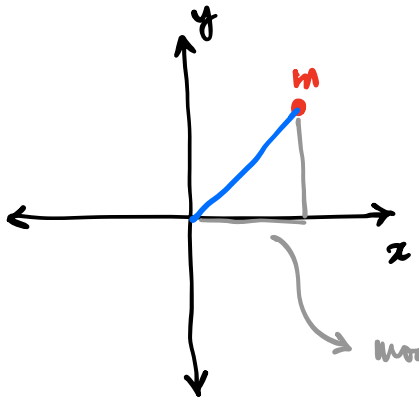
spring deformation = $\sqrt{74} - 5$

Magnitude of the force = $-k(\sqrt{74} - 5)$

Direction of the force = 

Unit vector = $\frac{1}{\sqrt{74}}(5, 7)$

2D Elastic Band



(i) Rest length is $\emptyset$

(ii) Hook's law describes the relationship between deformation and force.

$$\vec{F} = -k\vec{x}$$

model as two elastic bands.

$$F_x = -kx \quad (1)$$

$$F_y = -ky \quad (2)$$

(1) $m\frac{d^2x}{dt^2} = -kx$: $m\frac{dv_x}{dt} = -kx$ & $\frac{dx}{dt} = v_x$

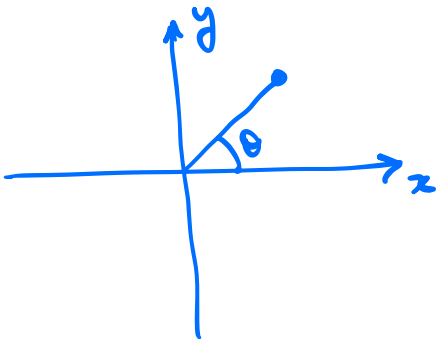
(2) $m\frac{d^2y}{dt^2} = -ky$: $m\frac{dv_y}{dt} = -ky$ & $\frac{dy}{dt} = v_y$

State variables : 4 state variables (x, y, v_x, v_y)

An alternate scheme :

$$F_x = -k\sqrt{x^2+y^2}\cos(\theta)$$

$$F_y = -k\sqrt{x^2+y^2}\sin(\theta)$$



* A 2D elastic bound with damping.

$$F_x = -kx - cv_x$$

$$F_y = -ky - cv_y$$

drag is proportional to velocity but in the opposite direction.

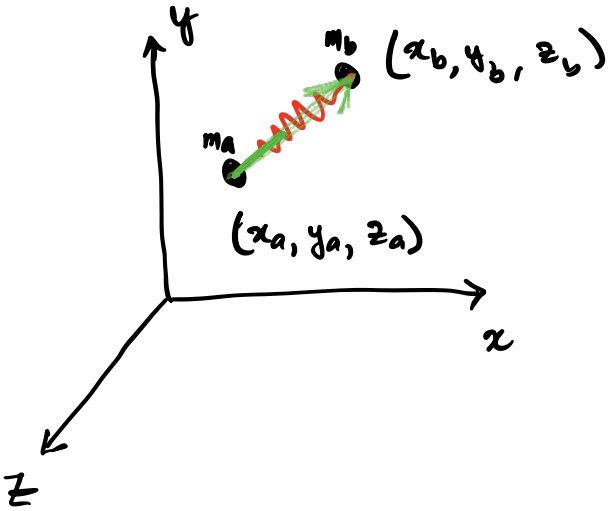
$$F_x = -kx + cv_x$$

$$F_y = -ky + cv_y$$

X

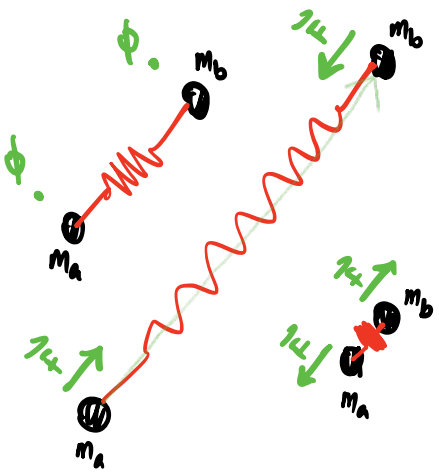
To solve this system, we need to setup the system of ODEs.

* Springs in 3D



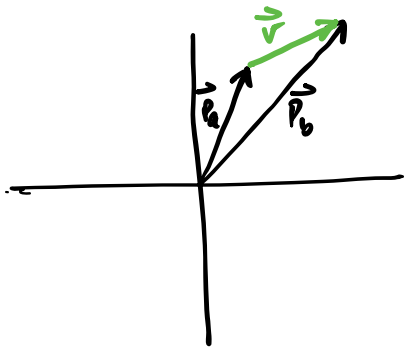
(i) Spring exerts force on both masses m_a m_b.

(ii) The force on both masses is the same but opposite.



(i) Spring axis vector :

$$\vec{v} = \vec{p}_b - \vec{p}_a$$
$$= (x_b - x_a,\ y_b - y_a,\ z_b - z_a)$$



(ii) current length of the spring :

$$\ell_{current} = \|v\| = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}$$

(iii) deformation :

$$d = \ell_{current} - \ell_{rest}$$

↳ positive d suggests that the spring is extended.

(iv) Unit vector along spring axis.

$$\hat{v} = \frac{\vec{v}}{\|v\|}$$

(v) Force on mass m_b :

$$\vec{F}_b = -kd\hat{v} \quad \in \mathbb{R}^3$$

(vi) Force on mass m_a :

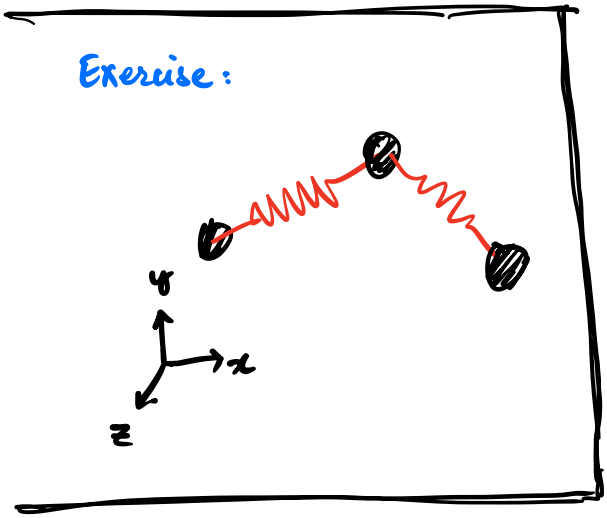
$$\vec{F}_a = kd\hat{v} \quad \in \mathbb{R}^3$$

Aside: # state variables = 12

$x^a, y^a, z^a, v_x^a, v_y^a, v_z^a$ $\quad$ $x^b, y^b, z^b, v_x^b, v_y^b, v_z^b$

mass m_a $\quad$ mass m_b

Exercise:



Write out the system of ODEs.

$F = ma.$

$\downarrow$

For m_a: $\vec{F}_a = m_a \vec{a}_a$

m_b: $\vec{F}_b = m_b \vec{a}_b$ } 2nd order eq.s

For m_a:

$$\frac{dx^a}{dt} = v_x^a$$

$$\frac{dy^a}{dt} = v_y^a$$

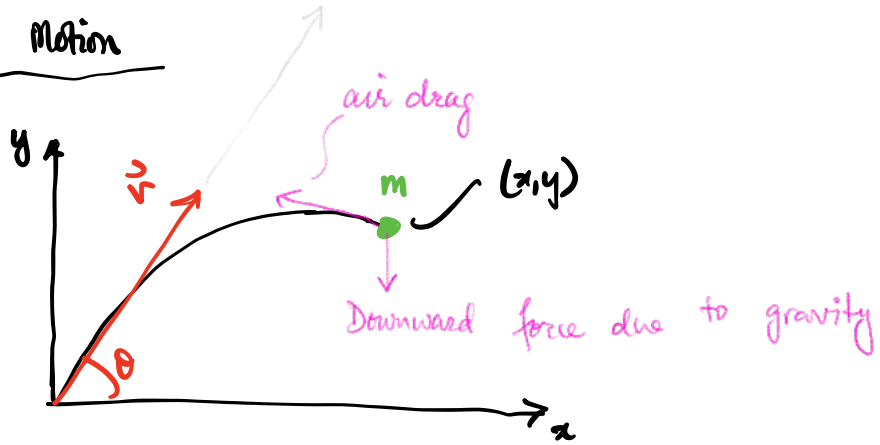
$$\frac{dz^a}{dt} = v_z^a$$

$$m_a \frac{dv_x^a}{dt} = F_{a,x}$$

$$m_a \frac{dv_y^a}{dt} = F_{a,y}$$

$$m_a \frac{dv_z^a}{dt} = F_{a,z}$$

Projectile Motion



Force in x-direction: $-\gamma x'$

Force in y-direction: $-\gamma y' - mg$

We construct the equations as follows:

$$m x'' = -\gamma x'$$
$$m y'' = -\gamma y' - mg$$

$v_x = x'$

$$m \frac{dv_x}{dt} = -\gamma v_x$$
$$\frac{dx}{dt} = v_x$$

$v_y = y'$

$$m \frac{dv_y}{dt} = -\gamma v_y - mg$$
$$\frac{dy}{dt} = v_y .$$

State variables: 4

x, y, v_x, v_y.

* Newton's Law of Gravitation

$$F = \frac{GMm}{(R+y)^2}$$

Force of gravity between two masses M and m at distance R+y between their centers.

$g = \frac{GM}{R^2}$ → mass of Earth.
↑
Earth's radius

Approximation.

* $$m x'' = -\gamma x'$$
$$m y'' = -\gamma y' - \frac{GMm}{(R+y)^2}$$

No analytical solution
Need to solve it numerically.

2D Mass Spring System Implementation

Simulation and Modeling (CSCI 3010U)

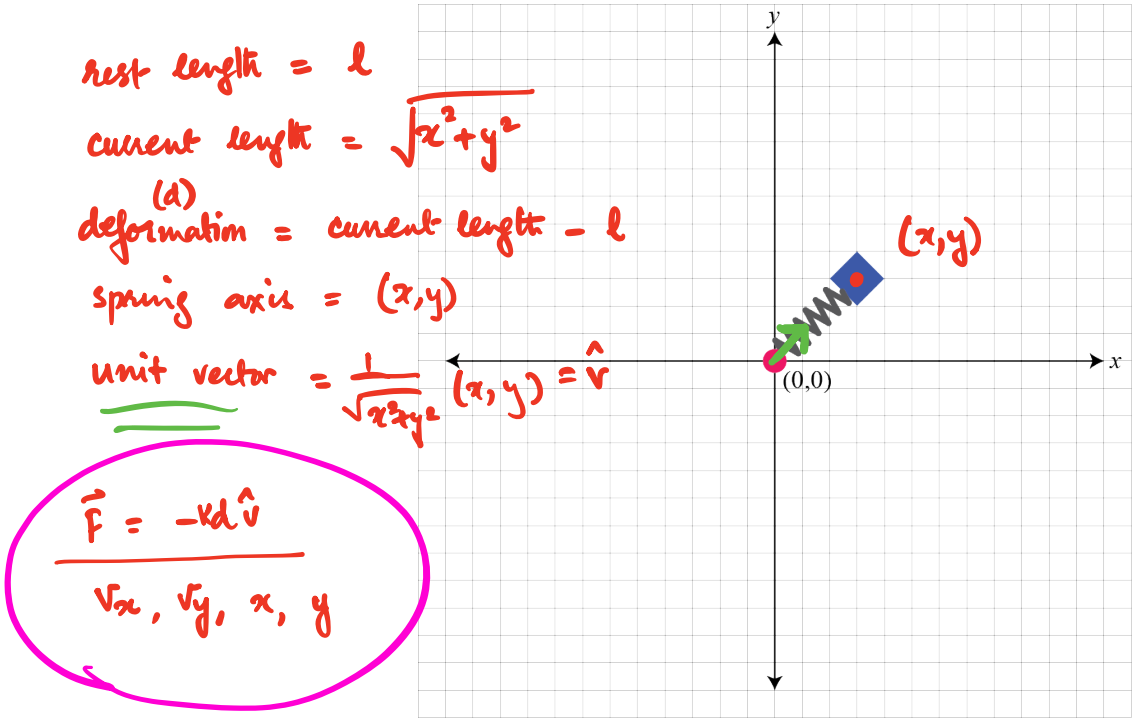
Faisal Qureshi

Faculty of Science, Ontario Tech University

http://vclab.science.ontariotechu.ca

Discuss in class

Consider a mass-spring system that lives in a flat (2D) world. One end of the spring is connected to a fixed hinge sitting at location $(0, 0)$. The other end of the spring is connected to a mass m. The rest length of this spring is l, and the Hooke's coefficient is k. Provide the `Simulation` class that simulates this mass spring system.



The following files are provided for you, which will use the `Simulation` class provided by you. It assumes that your class is available in file `sim.py`.

util.py

Includes routines used by mass-spring-2d.py file.

```
        """
        author: Faisal Z. Qureshi
        email: faisal.qureshi@uoit.ca
        website: http://www.vclab.ca
        license: BSD
        """

        import pygame

        # set up the colors
        BLACK = (0, 0, 0, 255)
        WHITE = (255, 255, 255, 0)
        RED = (255, 0, 0, 255)
        GREEN = (0, 255, 0, 255)
        BLUE = (0, 0, 255, 255)

        def load_image(name):
            image = pygame.image.load(name)
            return image

        class MyCircle(pygame.sprite.Sprite):
            def __init__(self, color, width, height, alpha=255):
                pygame.sprite.Sprite.__init__(self)

                self.image = pygame.Surface([width, height],
flags=pygame.SRCALPHA)
                self.rect = self.image.get_rect()
                cx = self.rect.centerx
                cy = self.rect.centery
                pygame.draw.circle(self.image, color, (width/2, height/2), cx, cy)
        #        self.rect = self.image.get_rect()

                self.picked = False

            def set_pos(self, pos):
                self.rect.x = pos[0] - self.rect.width//2
                self.rect.y = pos[1] - self.rect.height//2

            def update(self):
                pass
```

```
        class MyRect(pygame.sprite.Sprite):
            def __init__(self, color, width, height, alpha=255):
                pygame.sprite.Sprite.__init__(self)

                self.image = pygame.Surface([width, height],
flags=pygame.SRCALPHA)
                self.rect = self.image.get_rect()
                pygame.draw.rect(self.image, color, self.rect)

                self.picked = False

            def set_pos(self, pos):
                self.rect.x = pos[0] - self.rect.width//2
                self.rect.y = pos[1] - self.rect.height//2

            def update(self):
                pass

        def to_screen(x, y, win_width, win_height):
            return win_width//2 + x, win_height//2 - y

        def from_screen(x, y, win_width, win_height):
            return x - win_width//2, win_height//2 - y

        class MyText():
            def __init__(self, color, background=WHITE, antialias=True,
fontname="comicsansms", fontsize=16):
                pygame.font.init()
                self.font = pygame.font.SysFont(fontname, fontsize)
                self.color = color
                self.background = background
                self.antialias = antialias

            def draw(self, str1, screen, pos):
                text = self.font.render(str1, self.antialias, self.color,
self.background)
                screen.blit(text, pos)
```

mass-spring-2d.py

Calls your Simulation class to do the heaving lifting.

```
        """
        author: Faisal Z. Qureshi
```

```
email: faisal.qureshi@uoit.ca
website: http://www.vclab.ca
license: BSD
"""

import pygame, sys
import matplotlib.pyplot as plt
import numpy as np

# import sim_rk4 as Simulation
import sim as Simulation
import util


def main():
    # sim title
    title = 'Mass-Spring System'

    # initializing pygame
    pygame.init()

    # clock object that ensure that animation has the same
    # on all machines, regardless of the actual machine speed.
    clock = pygame.time.Clock()

    # fonts
    text = util.MyText(util.BLACK)

    # setting up a sprite group, which will be drawn on the
    # screen
    ball = util.MyRect(color=util.BLUE, width=32, height=32)
    center = util.MyRect(color=util.RED, width=4, height=4)
    x_axis = util.MyRect(color=util.BLACK, width=620, height=1)
    y_axis = util.MyRect(color=util.BLACK, width=1, height=460)
    my_group = pygame.sprite.Group([x_axis, y_axis, ball, center])

    # set up drawing canvas
    # top left corner is (0,0) top right (640,0) bottom left (0,480)
    # and bottom right is (640,480).
    win_width = 640
    win_height = 480
    screen = pygame.display.set_mode((win_width, win_height))
    pygame.display.set_caption(title)

    # setting up simulation
    sim = Simulation.Simulation(title)
    # sim.init(state=np.array([200,200,0,0], dtype='float32'), mass=100.,
```

```
k=.01, l=200.) Try some other values
            sim.init(state=np.array([200,200,0,0], dtype='float32'), mass=10.,
k=10, l=200.)
            sim.set_time(0.0)
            sim.set_dt(0.1)

            print ('---------------------------------')
            print ('Usage:')
            print ('Press (r) to start/resume simulation')
            print ('Press (p) to pause simulation')
            print ('Press (q) to quit')
            print ('Press (space) to step forward simulation when paused')
            print ('Use mouse left button down to move mass around (only when
simulation paused)')
            print ('---------------------------------')

            # Transformation to screen coordinates
            # Here 0,0 refers to simulation coordinates
            center.set_pos(util.to_screen(0, 0, win_width, win_height))
            x_axis.set_pos(util.to_screen(0, 0, win_width, win_height))
            y_axis.set_pos(util.to_screen(0, 0, win_width, win_height))


            while True:
                # 30 fps
                clock.tick(30)

                # update sprite x, y position using values
                # returned from the simulation
                ball.set_pos(util.to_screen(sim.state[0], sim.state[1], win_width,
win_height))

                event = pygame.event.poll()
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit(0)

                if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
                    sim.pause()
                    continue
                elif event.type == pygame.KEYDOWN and event.key == pygame.K_r:
                    if not ball.picked:
                        sim.resume()
                    continue
                elif event.type == pygame.KEYDOWN and event.key == pygame.K_q:
                    break
                elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1: #
```

```
LEFT=1
                    if sim.paused:
                        if ball.rect.collidepoint(event.pos):
                            ball.picked = True
                elif event.type == pygame.MOUSEMOTION:
                    if ball.picked:
                        x, y = util.from_screen(event.pos[0], event.pos[1],
win_width, win_height)
                        sim.set_state(np.array([x, y, 0, 0], dtype='float32'))
                elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:
                    if ball.picked:
                        ball.picked = False
                        sim.set_state(np.array([x, y, 0, 0], dtype='float32'))
                else:
                    pass

                # clear the background, and draw the sprites
                screen.fill(util.WHITE)
                my_group.update()
                my_group.draw(screen)
                text.draw("Time = %f" % sim.cur_time, screen, (10,10))
                text.draw("x = %f" % sim.state[0], screen, (10,40))
                text.draw("y = %f" % sim.state[1], screen, (10,70))
                if ball.picked:
                    text.draw("Picked. (Simulation disabled)", screen, (10,100))
                pygame.display.flip()

                # update simulation
                if not sim.paused:
                    sim.step()
                elif not ball.picked and event.type == pygame.KEYDOWN and
event.key == pygame.K_SPACE:
                        sim.step()
                else:
                    pass

            pygame.quit()
            sys.exit(0)

        if __name__ == '__main__':
            main()
```

sim.py

Starter code for your simulation class.

```
"""
author: Faisal Qureshi
email: faisal.qureshi@uoit.ca
website: http://www.vclab.ca
license: BSD
"""

import numpy as np

class Simulation:
    def __init__(self, title):
        self.paused = True # starting in paused mode
        self.title = title
        self.cur_time = 0
        self.dt = 0.033 # 33 millisecond, which corresponds to 30 fps
        # Fix this

    def init(self, state, mass, k, l):
        # Fix this
        pass

    def set_state(self, state):
        # Fix this
        pass

    def set_time(self, cur_time=0):
        self.cur_time = cur_time

    def set_dt(self, dt=0.033):
        self.dt = dt

    def step(self):
        # Fix this
        pass

    def pause(self):
        self.paused = True

    def resume(self):
        self.paused = False

    def save(self, filename):
        # Ignore this
        pass

    def load(self, filename):
```

position, velocity
x, y, 0, 0

$\Delta x + x \rightarrow x_{new}$
$\Delta y + y \rightarrow y_{new}$
$\Delta v_x + v_x \rightarrow v_{x,new}$
$\Delta v_y + v_y \rightarrow v_{y,new}$

```
        # Ignore this
        pass
```

Code execution

We will run your code as follows:

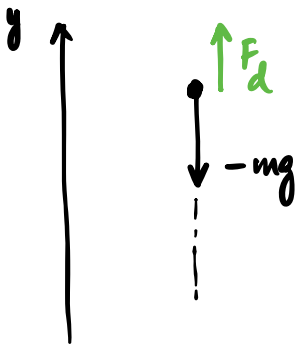
```
$ python mass-spring-2d.py
```

Submission

Nothing to submit. Please show your work to the instructor. Due in class.

---

* Free-falling particle.



$F_{total} = -mg + F_d$

drag force that depends upon the velocity of this mass.

(i) As the velocity increases, F_d will also increase.

(ii) At some point F_d will be equal to mg.

(iii) No force is acting on the object.

Object will continue to fall at a constant velocity.

(iv) The velocity at which the drag force is equal to mg is called terminal velocity.

↓

keep this aspects in mind when simulating

phenomenon.

* Models of Drag

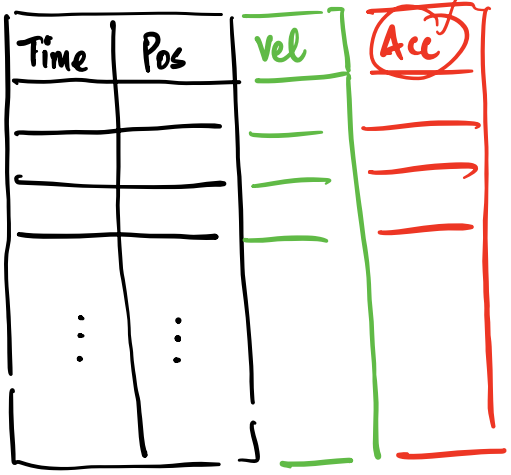
(i) Linear model:

$$F_{1,d} = c_1 v = mg \frac{v}{v_{1,t}}$$

(ii) Quadratic:

$$F_{2,d} = c_2 v^2 = mg \left(\frac{v}{v_{2,t}}\right)^2$$

* Modeling a coffee filter



$F = ma$

- Estimate velocity using finite differences
- Estimate acceleration
- Figure out the relationship between velocity & acceleration. Is it linear or quadratic?
- Compute drag force
- Run simulation.