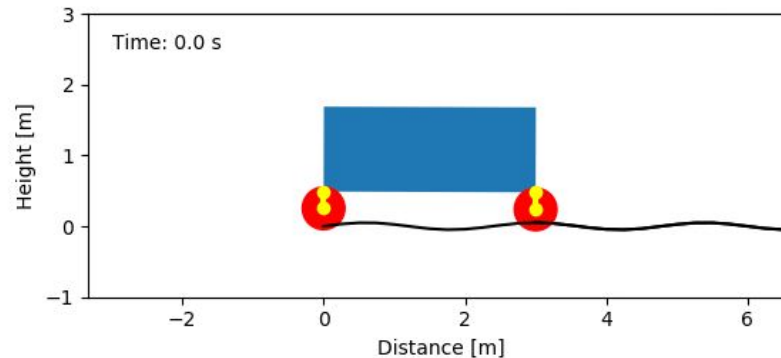


Resonance

Learning Mechanical Vibrations Through Computational Thinking



Car Traversing A Bumpy Road

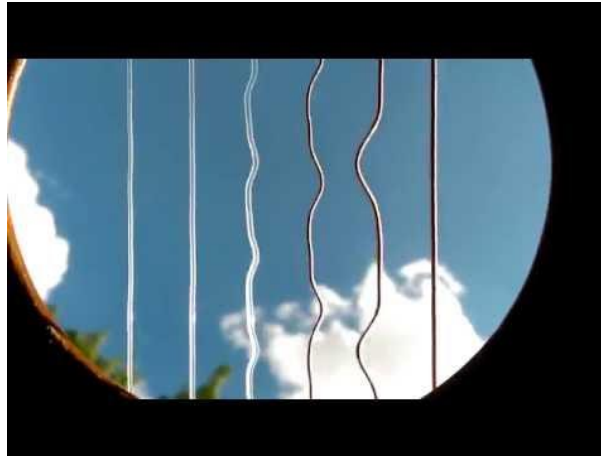
SciPy 2018, Friday, July 13, 2018

Jason K. Moore, **Kenneth R. Lyons**
Mechanical and Aerospace Engineering Department
University of California, Davis

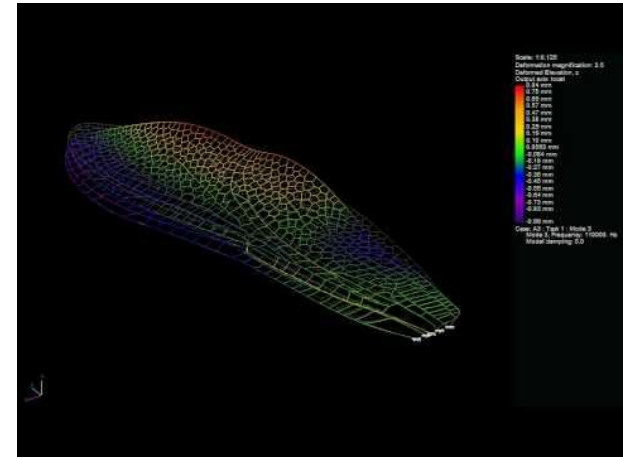
What are Mechanical Vibrations?



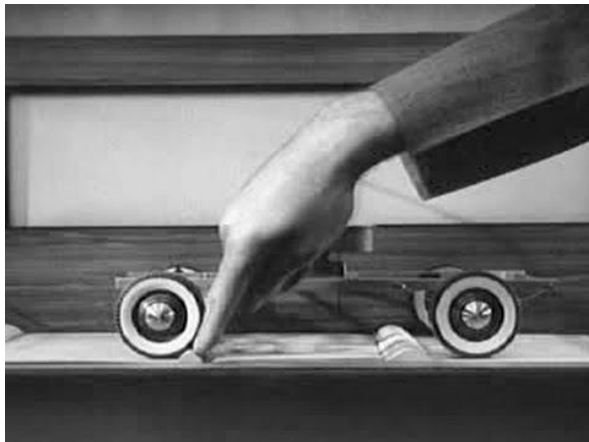
Unbalanced Washing Machine



Plucked Guitar Strings



Flapping Insect Wing



Car Suspension

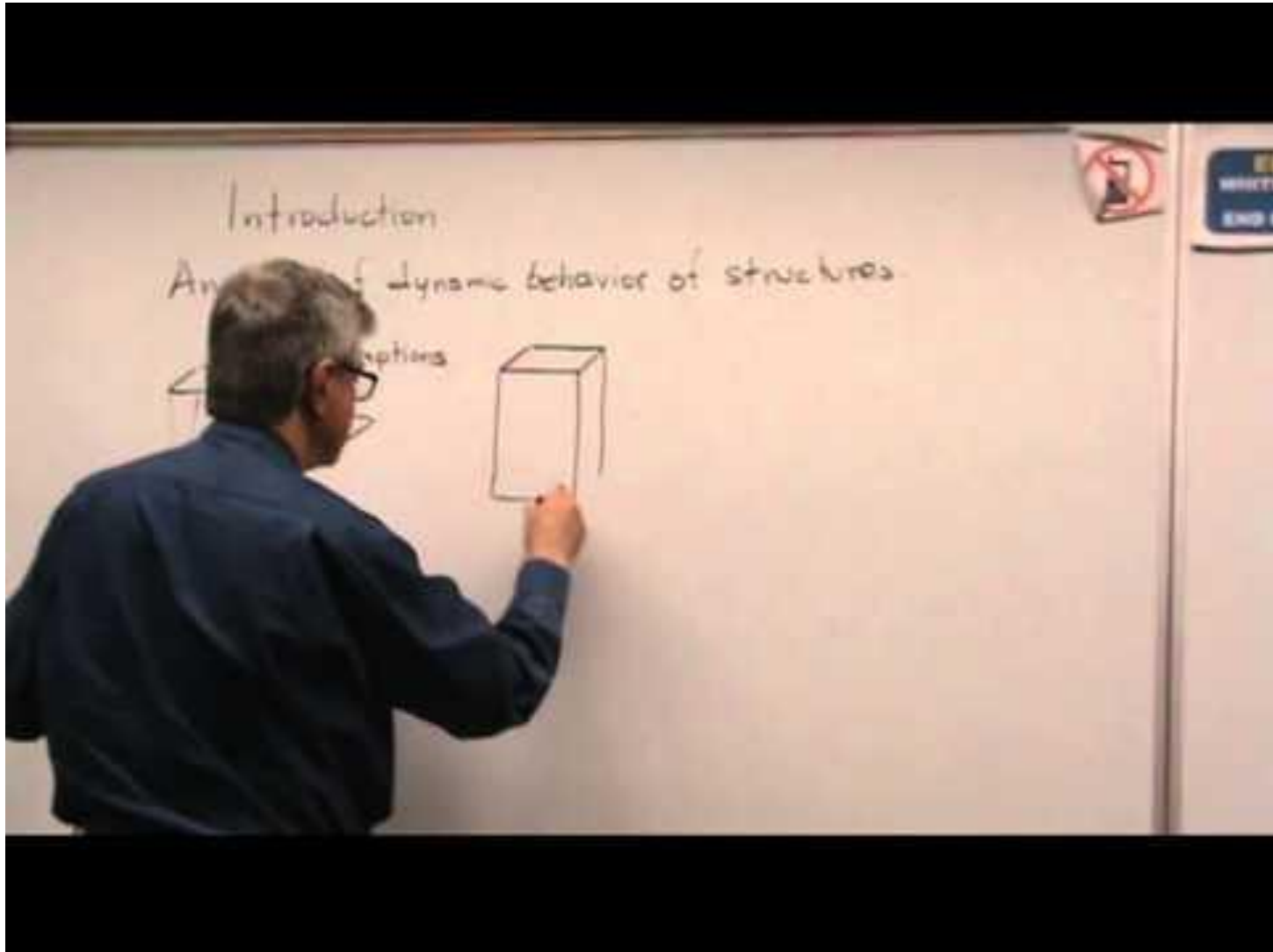


Bridge in the Wind

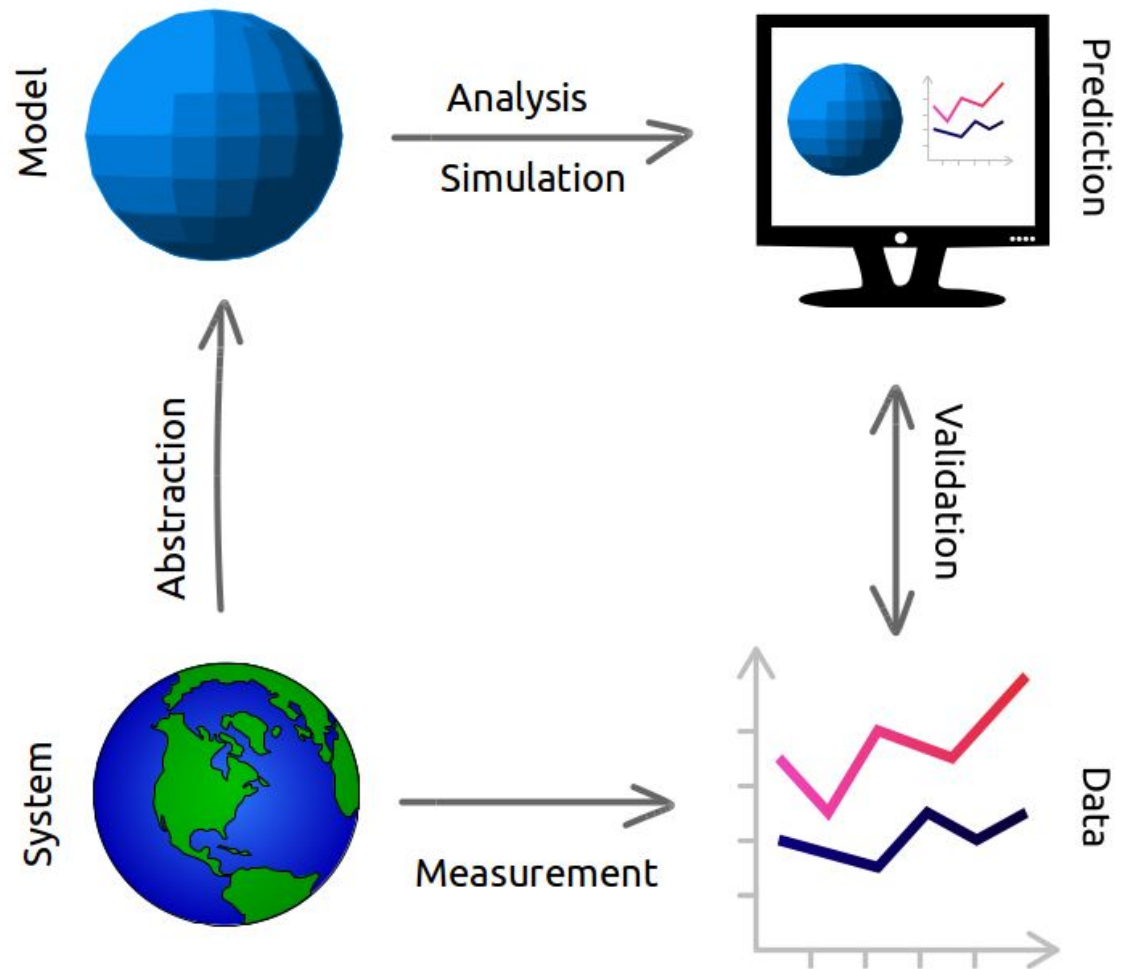


Rotorcraft

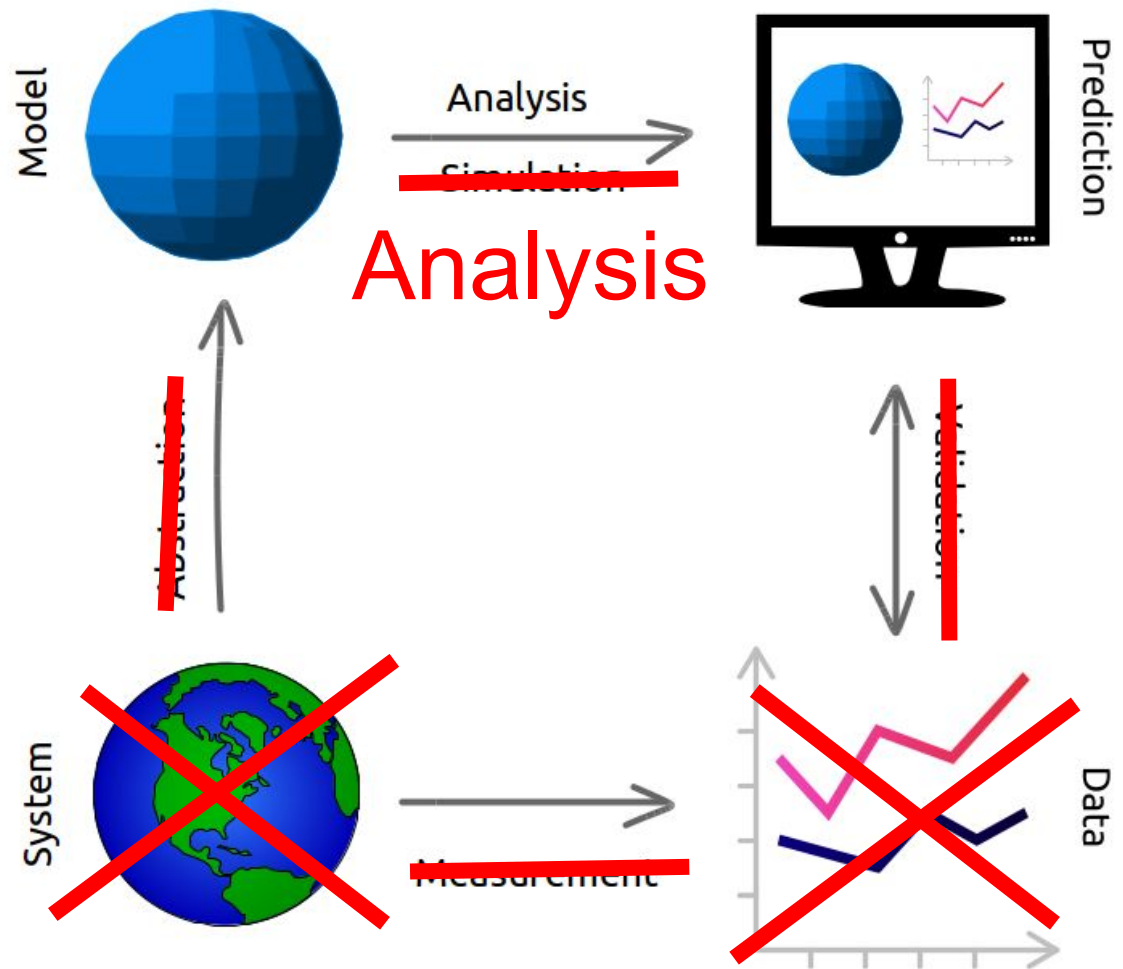
Typical First Day Of A Vibrations Course



This is what
mathematical
modeling
should be



This is what it looks like in most classes.



Computational Thinking

How To Define Computational Thinking

Before computers, engineering reasoning primarily based on:

- physical experimentation
- analytical mathematics

Post computers, new way of reasoning:

“Computational Thinking” is a way to think and solve problems using the constructs and abstractions in a programming language instead of or alongside of analytical mathematics and experimentation.

Now there are three primary languages used for understanding in engineering:

- natural language
- mathematical notation
- programming language

Probability Example: Experimenter's Method

What is the probability of rolling at least two 3's if you roll a 6-sided die 10 times?

Experimental solution: Roll ten dice thousands of times and tabulate how many times two or more 3's appear.

Result: 0.52...

Skills needed:

- bookkeeping
- counting
- division
- patience

Time to compute: Too many hours



Probability Example: Statistician's Method

What is the probability of rolling at least two 3's if you roll a 6-sided die 10 times?

Mathematical solution: remember the binomial theorem?

$$P(A) = \sum_{i=2}^{10} \binom{10}{i} \left(\frac{1}{6}\right)^i \left(\frac{5}{6}\right)^{10-i}$$

Result: 0.5233...

Mathematical concepts required:

- probability distributions
- conditional probability
- binomial theorem
- Factorials
- summations

The above abstractions seem distant from question at hand.

Probability Example: Programmer's Method

What is the probability of rolling at least two 3's if you roll a 6-sided die 10 times?

Computational solution: write some code

```
from random import choice
num_trials = 10000
dice_sides = [1, 2, 3, 4, 5, 6]
count = 0
for trial in range(num_trials):
    ten_rolls = [choice(dice_sides) for roll in range(10)]
    if ten_rolls.count(3) > 1:
        count += 1
print(count / num_trials)
```

Result: 0.5236...

Time to compute: < 1 second

Abstractions map closely
to experimenter's method!

Concepts:

- variables
- loops
- flow control

7 Ways To Learn By Computational Thinking

1. Translation
2. Proof by Example
3. Connect to the real world
4. Top-down sequence of topics
5. Two bites at every apple
6. Symbolic computation
7. Use and design APIs

Computation in STEM

Allen Downey, Olin College
Jason Moore, UC Davis

Learn more from workshop Co-Developed
with Olin College on Youtube



UCD MAE Course Description

- ENG 122: Introduction to Mechanical Vibrations
- Course website: <https://moorepants.github.io/eng122/>
- Upper level mechanical engineering elective
- Typically ~30 juniors and seniors
- 4 units: two 110 minutes sessions per week
- Primary prerequisites:
 - Dynamics (engineering application of classical mechanics)
 - Engineering programming
- Covers vibration of linear single and multi-degree of freedom systems

Course Learning Objectives

Ordered to **maximize motivation, focus usable on skills, and de-emphasize the mathematical abstractions:**

1. **Analyze:** Students will be able to analyze vibration measurement data to draw conclusions about a system's vibrational nature and characterize how the system behaves in terms of engineering vibration concepts.
2. **Model:** Students will be able to create simple mathematical and computational models of real vibrating systems that can be used to answer questions about the system by demonstrating appropriate vibration phenomena.
3. **Design:** Students will be able to design a mechanical structure that has desirable vibrational behavior.

Course Changes

- “Computational thinking” focused learning
- Reworked learning objectives
- Learn by doing: in and out of class
- All lessons motivated by real vibration problems
- Open access textbook and course materials
- Open-ended weekly computational homeworks
- Project instead of exams

Started With A New First Day Experience

- Goal: students solve a non-trivial vibrations problem end to end
- All three primary learning objectives are presented from top down perspective

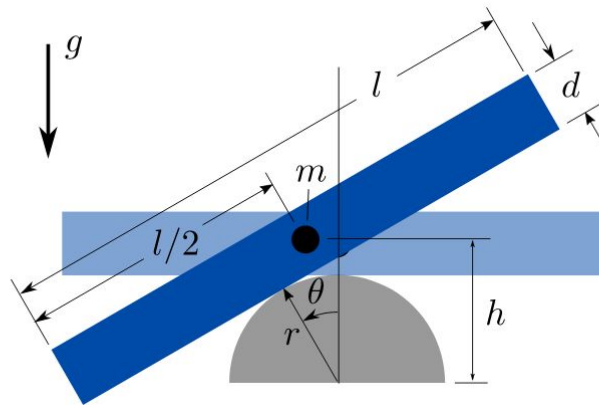
Modeling Step 1: Create a simplified free body diagram of the system

This step is not a trivial step, but for the purposes of the introduction we will assume you have come up with a simplified version of the conceptual mechanics that govern the system's motion. In this case there are many several assumptions made.

Exercise

If we want to learn specifically about how the book oscillates on the cup, what might be some good assumptions to make to simplify reality so that we can create a mathematical model?

Below you will find a pre-determined **free body diagram** that can likely capture the essential vibratory motion of this system.



Open in binder to get a feel for the materials:

<https://tinyurl.com/vib-first-day>

Software Design For Computational Thinking

Resonance: The Software Library

- Open Source: <https://github.com/moorepants/resonance> (CC-BY 4.0)
- Docs: <http://resonance.readthedocs.io>
- Conda: <https://anaconda.org/conda-forge/resonance>
- Pip: <https://pypi.org/project/resonance/>

Design Principles Derived From Computational Thinking

- Students only create functions, no need to understand classes and objects.
- Hide the simulation details (linear/nonlinear ODE solutions).
- Centered around the “System” object. Systems represent real things: a car, a bridge, a bicycle, an airplane wing.
- Students can use built-in systems construct their own.
- Easy visualizations (time history plots and animations of systems)
- Extra informative and lots of error messages (try to predict student mistakes)
- Don't teach programming for the sake of teaching programming. Show them how to solve problems and introduce programming along the way to solve those problems.

Primary Class: System

Represents a vibrating object, machine, mechanism, etc.

Stores constants, coordinates, measurements, equations of motion.

Allows students to simulate a system quickly and to explore behavior in just a few lines of code.

```
from resonance.linear_systems import SimpleQuarterCarSystem
```

```
sys = SimpleQuarterCarSystem()
```

```
sys.coordinates['car_vertical_position'] = -0.05 # meters
```

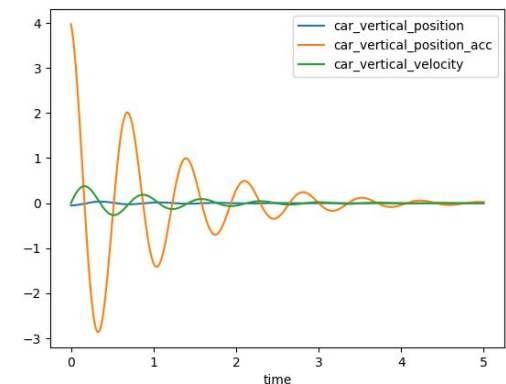
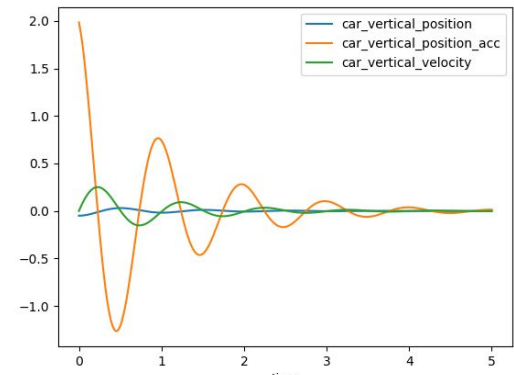
```
trajectory = sys.free_response(5.0)
```

```
trajectory.plot()
```

```
sys.constants['suspension_stiffness'] *= 2.0
```

```
trajectory = sys.free_response(5.0)
```

```
trajectory.plot()
```



System: Measurements

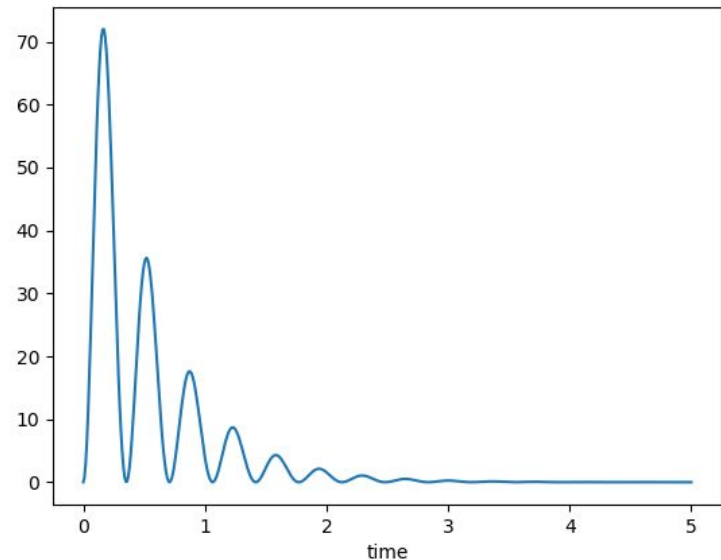
Students develop functions that represent measurements they may be interested in examining.

```
def calc_kinetic_energy(car_vertical_velocity, sprung_mass):  
    return 0.5 * sprung_mass * car_vertical_velocity**2
```

```
sys.add_measurement('kinetic_energy', calc_kinetic_energy)
```

```
trajectory = sys.free_response(5.0)
```

```
trajectory['kinetic_energy'].plot()
```

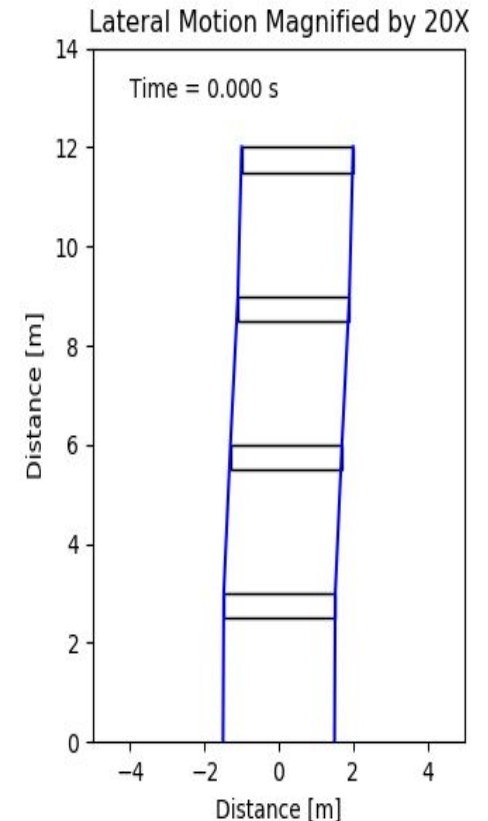


System: Configuration Plots

Visualize the system's configuration with matplotlib.

```
def plot_config(x1, x2, x3, x4):  
    fig, ax = plt.subplots(1, 1)  
    xi = scale * np.array([x1, x2, x3, x4])  
    rects = []  
    for i in range(4):  
        rect = Rectangle((-width / 2 + xi[i], 3 - height + i * 3),  
                          width, height, fill=False)  
        rects.append(rect)  
        ax.add_patch(rect)  
    left_walls = ax.plot(-width / 2 * np.ones(5) + np.hstack((0, xi)),  
                          [0, 3, 6, 9, 12], color='blue')[0]  
    right_walls = ax.plot(width / 2 * np.ones(5) + np.hstack((0, xi)),  
                          [0, 3, 6, 9, 12], color='blue')[0]  
    return fig
```

```
sys.config_plot_func = plot_config  
sys.plot_configuration()
```

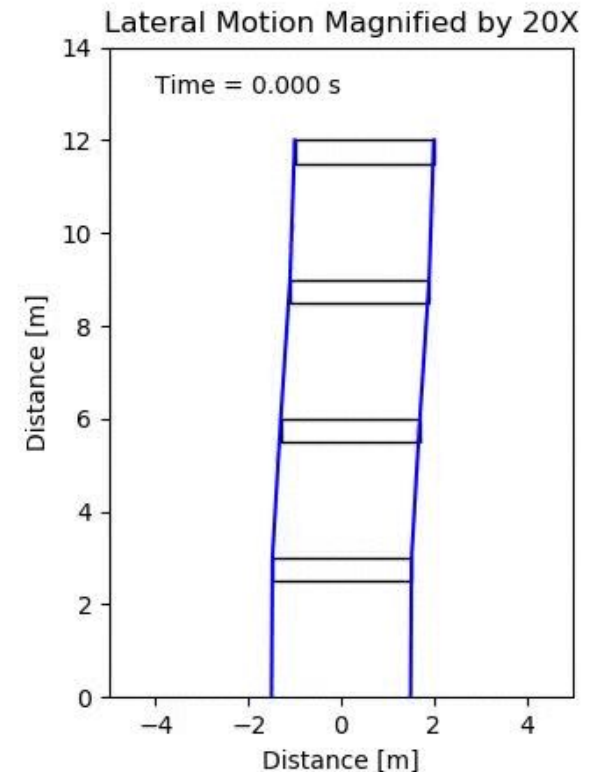


Building Swaying in Earthquake

System: Animations

The student defines a function that updates the configuration.

```
def update(x1, x2, x3, x4, rects, left_walls, right_walls):  
    # grab the ith coordinate vector  
    xi = scale * np.array([x1, x2, x3, x4])  
  
    # move each rectangle laterally by the associated coordinate  
    for i, rect in enumerate(rects):  
        rect.set_xy([-width / 2 + xi[i], 3 - height + i * 3])  
  
    left_walls.set_xdata(-width / 2 * np.ones(5) + np.hstack((0, xi)))  
    right_walls.set_xdata(width / 2 * np.ones(5) + np.hstack((0, xi)))  
  
sys.config_plot_update_func = update  
  
sys.animate_configuration()
```



Building Swaying in Earthquake

System: Equations of Motion

Students develop models of systems and derive the equations of motion (non-linear and linear) using Lagrange's approach (energy methods). All of the mathematics are done with symbolic algebra using **SymPy**.

The Lagrangian:

```
In [13]: L = T - U
         L
```

Out[13]:

$$\frac{I}{2} \left(\frac{d}{dt} \theta(t) \right)^2 - gm \left(l - \sqrt{l^2 - 4r^2 \sin^2 \left(\frac{1}{2} \theta(t) \right)} \right) + \frac{mr^4 \sin^2 (\theta(t)) \left(\frac{d}{dt} \theta(t) \right)^2}{2 (l^2 + 2r^2 \cos (\theta(t)) - 2r^2)}$$

```
In [14]: zero = L.diff(theta.diff(t)).diff(t) - L.diff(theta)
         zero
```


Out[14]:

$$I \frac{d^2}{dt^2} \theta(t) + \frac{2gmr^2 \sin \left(\frac{1}{2} \theta(t) \right) \cos \left(\frac{1}{2} \theta(t) \right)}{\sqrt{l^2 - 4r^2 \sin^2 \left(\frac{1}{2} \theta(t) \right)}} + \frac{mr^6 \sin^3 (\theta(t)) \left(\frac{d}{dt} \theta(t) \right)^2}{(l^2 + 2r^2 \cos (\theta(t)) - 2r^2)^2} + \frac{mr^4 \sin^2 (\theta(t)) \frac{d^2}{dt^2} \theta(t)}{l^2 + 2r^2 \cos (\theta(t)) - 2r^2} + \frac{mr^4 \sin (\theta(t)) \cos (\theta(t)) \left(\frac{d}{dt} \theta(t) \right)^2}{l^2 + 2r^2 \cos (\theta(t)) - 2r^2}$$

System: Simulation

The symbolic equations of motion are **translated** into a function that computes the right hand side of the first order ordinary differential equations.

$$\frac{d}{dt}\omega(t) = -\frac{mr^2}{2(I(l^2 + 2r^2 \cos(\theta(t)) - 2r^2) + mr^4 \sin^2(\theta(t)))(l^2 + 2r^2 \cos(\theta(t)) - 2r^2)^{\frac{7}{2}}}\left(2g(l^2 + 2r^2 \cos(\theta(t)) - 2r^2)^4 \sin(\theta(t)) + 2r^4(l^2 + 2r^2 \cos(\theta(t)) - 2r^2)^{\frac{5}{2}}\omega^2(t) \sin^3(\theta(t)) + r^2(l^2 + 2r^2 \cos(\theta(t)) - 2r^2)^{\frac{7}{2}}\omega^2(t) \sin(2\theta(t))\right)$$



```
def eval_rhs(theta, omega, I, m, r, l, g):
    theta_dot = omega
    omega_dot = (-m*r**2*(2*g*(l**2 + 2*r**2*np.cos(theta) -
                                2*r**2)**4*np.sin(theta) +
                                2*r**4*(l**2 + 2*r**2*np.cos(theta) -
                                2*r**2)**(5/2)*omega**2*np.sin(theta)**3 +
                                r**2*(l**2 + 2*r**2*np.cos(theta) -
                                2*r**2)**(7/2)*omega**2*np.sin(2*theta)))/
                (2*(I*(l**2 + 2*r**2*np.cos(theta) - 2*r**2) +
                    m*r**4*np.sin(theta)**2)*(l**2 + 2*r**2*np.cos(theta) -
                    2*r**2)**(7/2)))
    return theta_dot, omega_dot
```

```
sys.diff_eq_func = eval_rhs
```

Course Materials and Assessment

Textbook

- Open Access (CC-BY)
- Written in Jupyter Notebooks: mixes prose, math, videos, graphics, code, widgets
- <https://moorepants.github.io/resonance>
- All chapters should have context: a real problem to solve
- 8 NBGrader style homework sets
- Notebooks used live in class, ala Software Carpentry
- Version 1.0 coming soon!

Energy

At any given configuration dynamic systems potentially have both potential energy and kinetic energy. Recall that the potential energy of a mass that is lifted above ground in a gravitational field is:

$$PE = mgh$$

where m is the mass of the object, g is the acceleration due to gravity, and h is the height above the ground. Additionally, the translational and rotational kinetic energy of a planar rigid body can be expressed as:

$$KE = \frac{mv^2}{2} + \frac{I\omega^2}{2}$$

where m is the mass of the body, v is the magnitude of the linear velocity of the center of mass, I is the centroidal moment of inertia of the rigid body, and ω is the angular velocity of the rigidbody.

For example the bob of the pendulum has:

$$PE_{bob} = m_{bob}g(l(1 - \cos\theta))$$

and

$$KE_{bob} = m_{bob}(\dot{\theta})^2 \frac{l^2}{2} + \frac{(m_{bob}l^2/2 + m_{bob}I^2)\dot{\theta}^2}{2}$$

You can add a measurement for this that looks like:

```
def kinetic_energy(bob_mass, bob_radius, rod_length, bob_height, rod_mass, angle_vel):  
  
    v_bob = rod_length * angle_vel  
    I_bob = bob_mass * bob_radius**2 / 2.0 + bob_mass * rod_length**2  
    KE_bob = bob_mass * v_bob**2 / 2.0 + I_bob * angle_vel**2 / 2.0  
  
    v_rod =  
    I_rod =  
    KE_rod =  
  
    return KE_rod + KE_bob
```

Exercise

Add a measurement to the system that outputs the kinetic energy in the trajectory data frame from `free_response`.

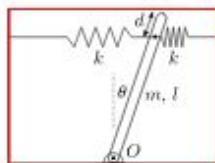
```
In [34]: def kinetic_energy(bob_mass, bob_radius, rod_length, bob_height, rod_mass, angle_vel):  
    v_bob = rod_length * angle_vel  
    I_bob = bob_mass * bob_radius**2 / 2.0 + bob_mass * rod_length**2  
    KE_bob = bob_mass * v_bob**2 / 2.0 + I_bob * angle_vel**2 / 2.0  
  
    v_rod = rod_length / 2 * angle_vel  
    I_rod = rod_mass * rod_length**2 / 3  
    KE_rod = rod_mass * v_rod**2 / 2.0 + I_rod * angle_vel**2 / 2  
  
    return KE_rod + KE_bob
```

Weekly Open-Ended Homeworks via NBGrader

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Problem 1

Pictured below is a slender rod of length l and mass m which pivots without friction about a point O . Two springs are connected a distance d from the end of the rod. You may assume the springs are deflected linearly in the horizontal direction and apply their restoring force perpendicular to the rod.



1. Derive the full nonlinear equation of motion for this system, then use the small angle approximation to find a linearized equation of motion.
2. What is the minimum k that ensures the linearized system stiffness is positive?
3. Use the linear equation of motion to find the natural frequency ω_n .
4. Use a `SingleDoFLinearSystem` from `resonance.linear_systems` to plot the system's free response to an initial angle when the stiffness is positive and when it is negative. Use whatever values for m , l , and d that you want.
5. Verify that your expression for ω_n is correct by comparing to the simulated system's period.

```
In [2]: import sympy as sm
sm.init_printing()
```

PART I:

In [3]:

```
Students answer
```

Full credit No credit 4.5 / 5.0 (extra credit)

```
#Create symbols
m, l, g, d, k, t = sm.symbols('m, l, g, d, k, t')
theta = sm.Function('theta')(t)
theta_dot = theta.diff()

#Solve for Kinetic Energy (T) and Potential Energy (U)
T = (1/2) * m * (theta_dot * (l / 2))**2 + (1/2) * I * theta_dot**2
U = k * ((l - d) * sm.sin(theta))**2 + m * g * (l/2 * sm.cos(theta) - l/2)

#Lagrange's eqn:
L = T - U
```

The natural frequency is the square root of the equivalent spring over the equivalent mass, not the mass of the pendulum. The difference in the two periods in question 5 should have tipped you off about the equation for the natural frequency.

In [4]:

```
#Nonlinear equation of motion for this system.
zero = L.diff(theta_dot).diff(t) - L.diff(theta)

print('Nonlinear equation of motion for this system:')
zero
```

Nonlinear equation of motion for this system:

```
Out[4]: 1.0*I*(d**2 - l**2)*sin(theta)**2 + 2*k*(-d + l)**2*sin(theta)*cos(theta) + 0.25*I*m*(d**2 - l**2)
```

To linearize the equation of motion, make assumptions for small θ :

- $\sin\theta \approx \theta$
- $\cos\theta \approx 1$

In [5]:

```
#Linearized equation of motion with small angle approximation:
zero_lin = zero.subs({sm.sin(theta): theta,
                    sm.cos(theta): 1})

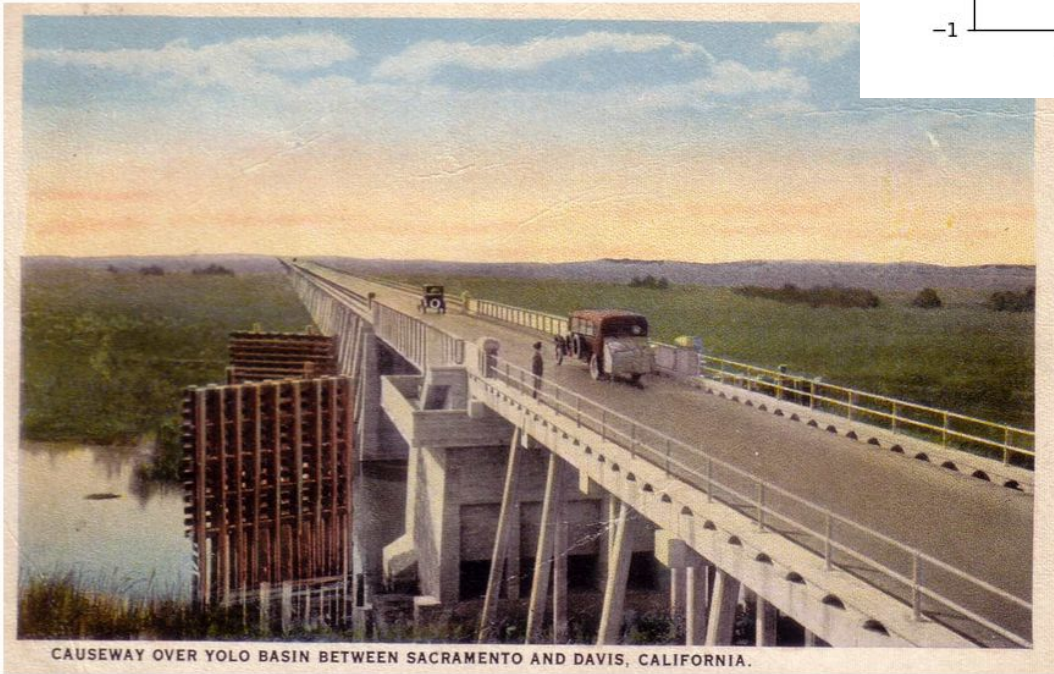
print('Linearized equation of motion with small angle approximation:')
zero_lin
```

Linearized equation of motion with small angle approximation:

```
Out[5]: 1.0*I*(d**2 - l**2)*theta**2 + 2*k*(-d + l)**2*theta + 0.25*I*m*(d**2 - l**2)
```

Design Project

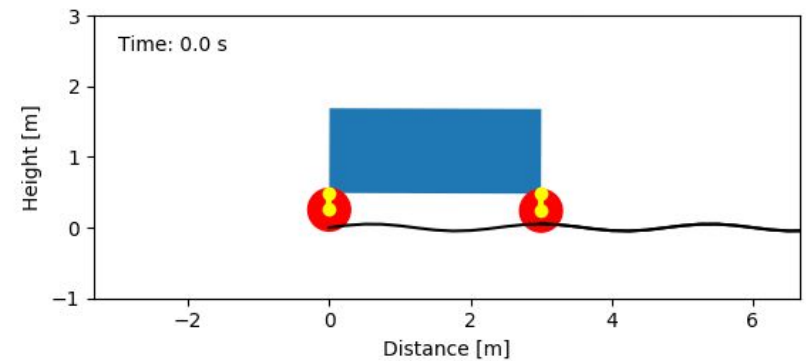
Car Suspension Design for the Yolo Causeway



The old [Yolo Causeway](#) (pre-1962) was constructed by simply supported beams across 30 meter spans. These beams deflected due to their own weight with a maximum deflection at the center of the spans of 7 centimeters. This design worked well for cars of the era, but as cars improved over time and speeds increased it was discovered that at certain speeds, some cars vibrated to both uncomfortable and unsafe levels.

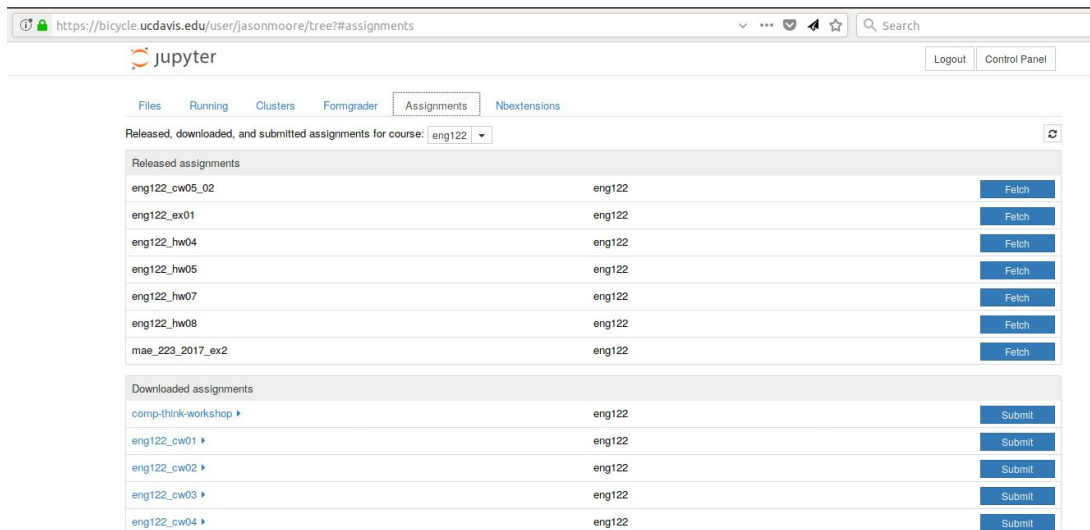
You work for a major car manufacturer at the time of this discovery and management is very concerned that the suspension designs of the past era will no longer cut it for higher travel speeds with new car models. Management has put together a team of engineers to solve this problem. You are part of the suspension design team. You have been provided with a Python (it would have actually be Fortran in ~1960) file, `half_car.py`, that contains a simulation model of a half car system. This model was developed by a previous engineer and does a very good job at modelling the existing car suspension. You can use this model to obtain various "measurements" from a typical car designed by your company driving over arbitrary road profiles at a range of speeds.

Your primary goal is to design a new suspension for the car so that it has suitable displacement, rotational, and force transmissibility ratios for speeds from 0 to 130 km/h while riding over the causeway. You must decide what is suitable and explain why in your notebook.



Delivered on Self Hosted JupyterHub

- Multi user cloud server for Jupyter notebooks.
- JupyterHub allows us the freedom to set things up the way we want.
- Many options for deployment, customization, etc.
- Free and open source (BSD).
- Friendly and active community, consisting of people that work on Jupyter itself.



Released, downloaded, and submitted assignments for course: eng122		
Released assignments		
eng122_cw05_02	eng122	Fetch
eng122_ex01	eng122	Fetch
eng122_hw04	eng122	Fetch
eng122_hw05	eng122	Fetch
eng122_hw07	eng122	Fetch
eng122_hw08	eng122	Fetch
mae_223_2017_ex2	eng122	Fetch
Downloaded assignments		
comp-think-workshop ▶	eng122	Submit
eng122_cw01 ▶	eng122	Submit
eng122_cw02 ▶	eng122	Submit
eng122_cw03 ▶	eng122	Submit
eng122_cw04 ▶	eng122	Submit

Goals for our setup:

- run persistently for several years with low cost
- install/update packages as we see fit (sometimes right before class starts)
- simple to maintain*
- restrict access to specific UCD students

Lessons Learned

Negative

- The overhead of introducing a programming language still a bit high (most are new to Python)
- resonance software needs to expose the right details and pace the introduction
- Classwork can move too fast
- Not cultured to ask questions in active learning
- nbgrader workflow isn't ideal for our homework style
- Students haven't quite grokked good notebook presentation style yet
- Need to assess more acutely whether the learning objectives have been met
- It was an extreme amount of work to fully convert the class in one go

Positive

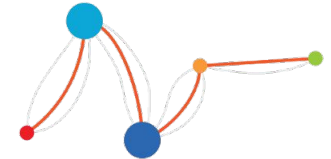
- Students seem motivated to learn about vibrations (they want to know how the simulations work)
- Students are able to work at different levels of abstraction for solving problems
- Students can approach fairly complex systems and run the entire analysis process
- Impressive project solutions!
- They like Python, especially the symbolics.

Thanks

- Co-instructor: Benjamin Margolis (MAE Grad Student)
- ENG 122 Students
- Center of Educational Effectiveness
- Mechanical and Aerospace Engineering Department
- Data Science Initiative
- Allen Downey and Olin College



Data Science Initiative



More Info

- <http://www.moorepants.info/blog/introducing-resonance.html>
- @ixjlyons, @moorepants

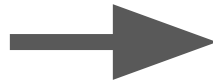
Extras

1. Translation

Students learn mathematical ideas and algorithms by translating them into code.

From natural language, math notation, or another programming language.

$$y = \sum_{i=0}^{10} x_i$$



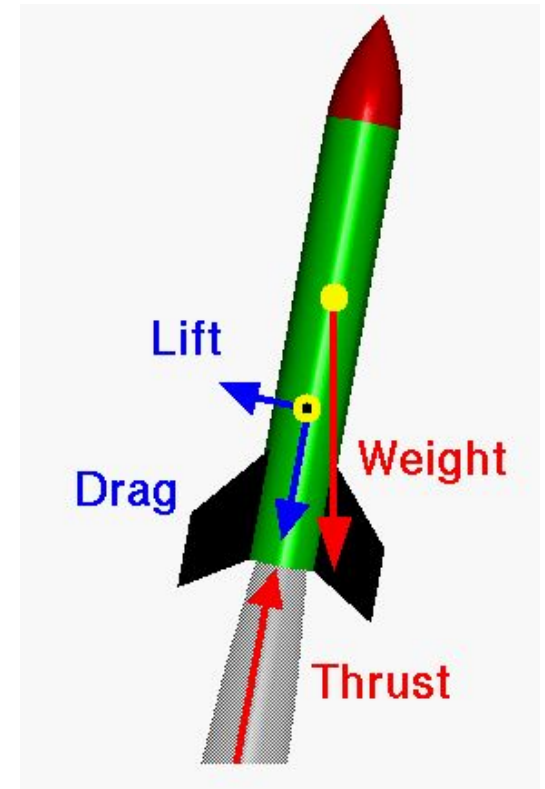
```
y = 0
for xi in range(11):
    y += xi
```

2. Proof by example

Run experiments to see if theoretical results are true.

And when they break.

Example: Try using Newton's second law, in $F = ma$ for a rocket.



3. Connect to the real world

With basic programming skills, you can pull data from almost anywhere.

Or collect it yourself.

Engage with real problems, real data.

4. Top-down sequence of topics

When algorithms are available in libraries,
you can use them before you know how they work.

Example: Start with vibrations simulation instead of ordinary
differential equations

5. Two bites at every apple

Some people love mathematical notation, and learn by reading formulas.

Many people benefit from seeing these ideas expressed several different ways: math, natural language, programming language.

6. Symbolic computation

When students do symbolic manipulation on paper, they often lose the forest for the (dead) trees.

With symbolic computation, you can separate the ends from the means.

$$y = \sum_{i=0}^{10} x_i$$



```
In [1]: from sympy import *  
        from sympy.abc import x, y, i
```

```
In [2]: init_printing()
```

```
In [3]: rhs = Sum(Indexed('x', i), (i, 0, 10))  
        Eq(y, rhs)
```

```
Out[3]: y = \sum_{i=0}^{10} x_i
```

```
In [4]: Eq(y, rhs.doit())
```

```
Out[4]: y = x_0 + x_{10} + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9
```

7. Use and design APIs

Abstract mathematical entities are...
abstract.

Representing entities as
computational objects and operations
clarifies what these entities ARE...

... by showing what they DO.

API: Application Programming
Interface

```
In [1]: %matplotlib notebook
```

```
In [2]: from resonance.linear_systems import BicycleSystem
```

```
In [3]: sys = BicycleSystem()
```

```
In [4]: traj = sys.free_response(5.0)
```

```
In [5]: traj[['phi', 'delta']].plot();
```

Figure 1

