Numerical Python

Simulation - State

Announcements

quiz: `quiz12` due on Tues 29/10

lab: `lab` on Fri 01/11

hw: `hw06` due 30/10

Roadmap

- **Week 1 (9/9)**: L0 Intro; L1 PY - Intro
- **Week 2 (16/9)**: L2 PY – Data& Bool; L3 PY – if & Loops
- **Week 3 (23/9)**: L4 PY - Lists; L5 PY - Funcs
- **30/9**: L6 PY - Files
- **9/10**: L7 PY - Muta
- **12/10**: Exam 01
- **Week 6 (14/10)**: L8 PY - Dict; L9 PY - IO
- **Week 7 (21/10)**: L10 PY - Numpy; L11 PY - Plot
- **Week 8 (28/10)**: L12 PY - State; L13 PY - Rand
- **Week 9 (4/11)**: L14 PY – De-bug; L15 PY - Eqn
- **Week 10 (11/11)**: L16 PYA I; Exam 02
- **Week 11 (18/11)**: L17 PY - Opt II; L18 PY - Mod
- **Week 12 (25/11)**: L19 PY – Sym I; L20 PY – Sym II
- **Week 13 (2/12)**: L21 PY – Sym III; L22 ML - Intro
- **Week 14 (9/12)**: L23 ML - IO; L24 ML - Poly
- **Week 15 (16/12)**: L25 ML - Stats; L26 ML - Fit
- **Week 16 (23/12)**: Review
- **Week 17 (30/12)**: Finals

Objectives

A. Use program state to track the evolution of models over time.
B. Construct multivariable simulations.
C. Apply the finite difference method to simulate differential equations.

plt Recap

plt.show()

1. It is a command needed if you plot from a python script.

    This means you have a file.py and run it at prompt (not inside python) `> python file.py`

In jupyter, you just need to use `plt.show()` or more often `%matplotlib inline` once in the same notebook.

In Spyder, you might not need it. There are some settings in Spyder that made this unnecessary. For some people, they still need in their Spyder.

plt.show()

1. It is a command needed if you plot from a python script.

   This means you have a file.py and run it at prompt (not inside python) `> python file.py`

In jupyter, you just need to use `plt.show()` or more often `%matplotlib inline` once in the same notebook.

In Spyder, you might not need it. There are some settings in Spyder that made this unnecessary. For some people, they still need in their Spyder.

So do you want to use `plt.show()`?

Question 1

Which of the following produces a valid plot given data in `x` and `y`?

A
```
import matplotlib.plot as plot
plot( x,y )
show( x,y )
```

B
```
import matplotlib.pyplot as plt
plt.plot( x,y )
plt.show()
```

C
```
import matplotlib.plot as plot
plot( x,y )
show()
```

Question 1

Which of the following produces a valid plot given data in `x` and `y`?

A
```
import matplotlib.plot as plot
plot( x,y )
show( x,y )
```

B ⋆
```
import matplotlib.pyplot as plt
plt.plot( x,y )
plt.show()
```

C
```
import matplotlib.plot as plot
plot( x,y )
show()
```

Question 2

Which format string produces a black dashed line?

A `'b--'`

B `'k--'`

C `'b-'`

D `'ko'`

Question 2

Which format string produces a black dashed line?

A `'b--'`

B `'k--'` ⋆

C `'b-'`

D `'ko'`

Question 3

```
plt.plot(x, y, 'bx', x, y2, 'r-')
```

What will be plotted?

A Error

B One line of 'r-'

C One line of 'bx'

D Both lines

Question 3

```
plt.plot(x, y, 'bx', x, y2, 'r-')
```

What will be plotted?

- A Error
- B One line of 'r-'
- C One line of 'bx'
- D Both lines ⋆what is 'x'?

plt.plot()

Two ways to plot:

`plt.plot(x, y, 'bx')` plot using markers

`plt.plot(x, y, linestyle='-', color='b')` plot using line, will not show which points are used

np.sqrt()

```
import numpy as np
import math
x = np.array([[1.0, 2.0, 3.0, 4.0]])
```

what will happen when:

A. `math.sqrt(x)`?

np.sqrt()

```
import numpy as np
import math
x = np.array([[1.0, 2.0, 3.0, 4.0]])
```

what will happen when:

A. `math.sqrt(x)`? Ans: `Error`

B. `np.sqrt(x)`?

np.sqrt()

```
import numpy as np
import math
x = np.array([[1.0, 2.0, 3.0, 4.0]])
```

what will happen when:

A. `math.sqrt(x)`? Ans: `Error`

B. `np.sqrt(x)`? Ans: `array([[1.0, 1.414, 1.732, 2.0 ]])`

*** `np.sqrt()` goes into array `x` and perform sqrt `individually`!!!

*** No need a loop!!!

np.sqrt()

```
import numpy as np
import math
x = np.array([[1.0, 2.0, 3.0, 4.0]])
```

what will happen when:

A. `math.sqrt(x)`? Ans: `Error`

B. `np.sqrt(x)`? Ans: `array([[1.0, 1.414, 1.732, 2.0 ]])`

*** `np.sqrt()` goes into array `x` and perform sqrt `individually`!!!

*** No need a loop!!!

*** Treat array `x` as one number when you use functions from numpy!!!

Modeling with State

Modeling

lec12:

```
"All models are wrong but some are useful"
                               ~ George Box
```

Modeling

Consider a ball falling from the edge of a table. Describe its path and time until it hits the ground.

Two approaches:

A Use analytical equation **`(if available)`**

B Use finite difference equation otherwise.

Modeling

A Use analytical equation (if available).

$$y(t) = y_0 + v_0 t + \frac{a}{2} t^2$$

$$y_0 = 1$$

$$v_0 = 0$$

$$a = -9.8$$

subject to

$$y(t) \geq 0$$

Modeling

Input

```
# Parameters of simulation
n = 100      # number of data points to plot
start = 0.0 # start time, s
end = 1.0    # ending time, s
a = -9.8     # acceleration, m*s**-2

# State variable initialization
t = np.linspace(start,end,n+1) # time, s
```

Model

```
y = 1.0 + a/2 * t**2

for i in range(1,n+1):
    if y[i] <= 0: # ball has hit the ground
        y[i] = 0
```

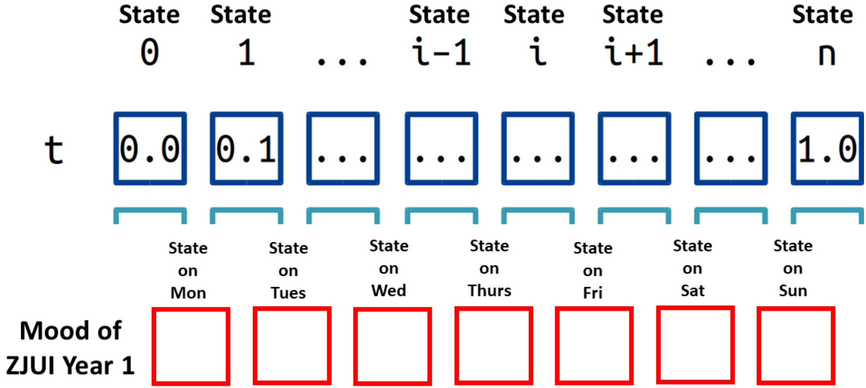
Modeling

	0	1	...	i-1	i	i+1	...	n
t	0.0	0.1	...	...	...	...	...	1.0
y	1.0	0.9	...	...	...	...	0.0	0.0
v	0.0	0.1	...	...	...	...	0.0	0.0

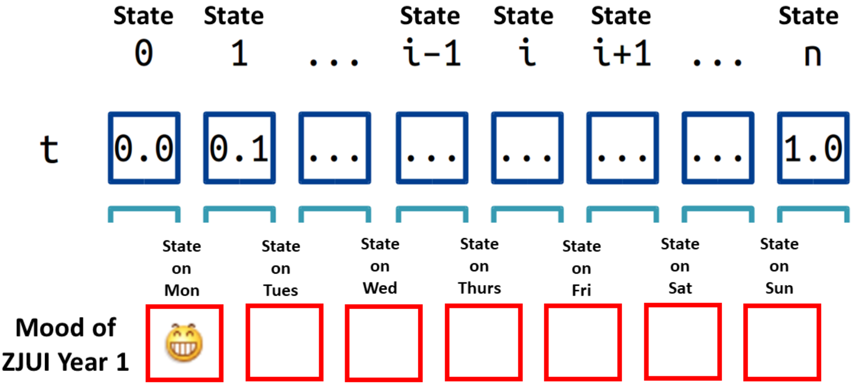
Model State

	State 0	State 1	...	State i-1	State i	State i+1	...	State n
t	0.0	0.1	...	...	...	...	...	1.0
y	1.0	0.9	...	...	...	...	0.0	0.0
v	0.0	0.1	...	...	...	...	0.0	0.0

Model State

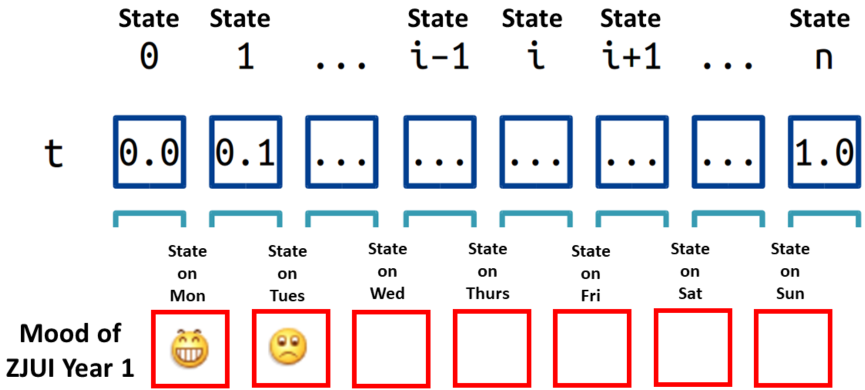

State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

Model State

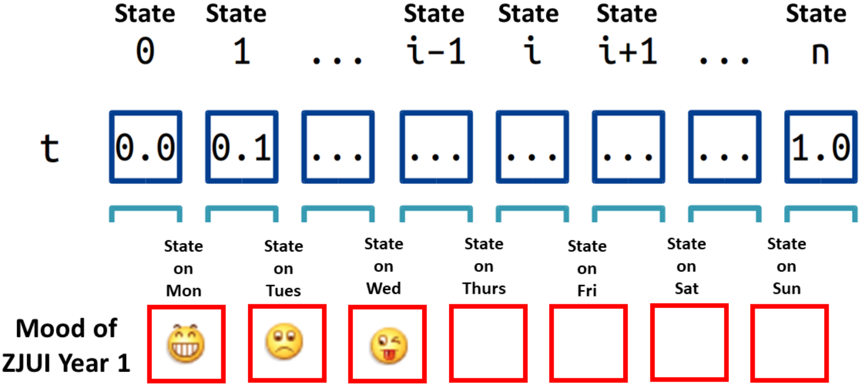


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

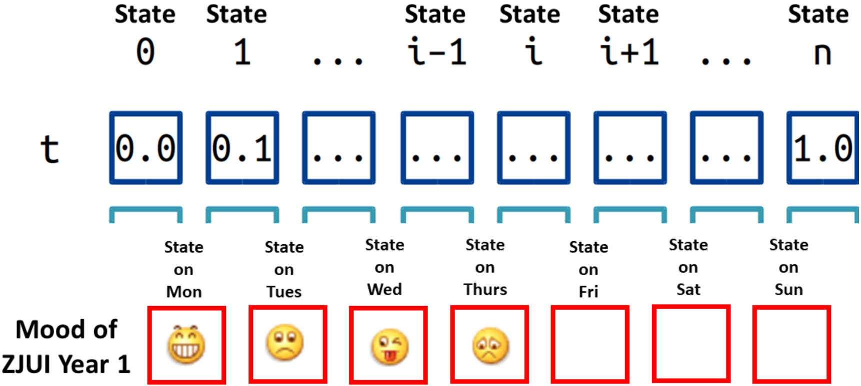
Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

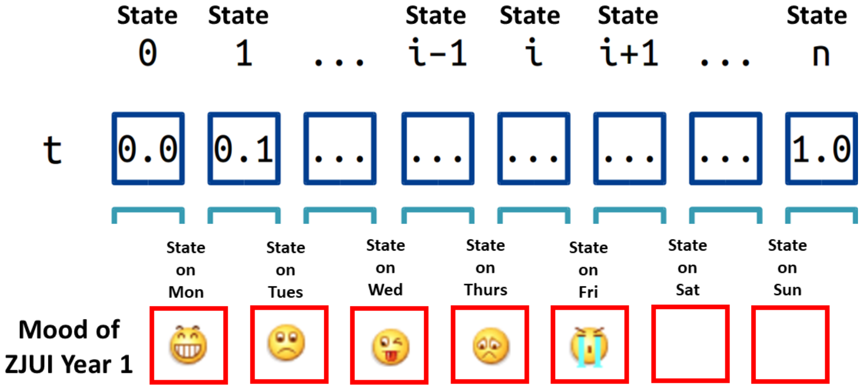
Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

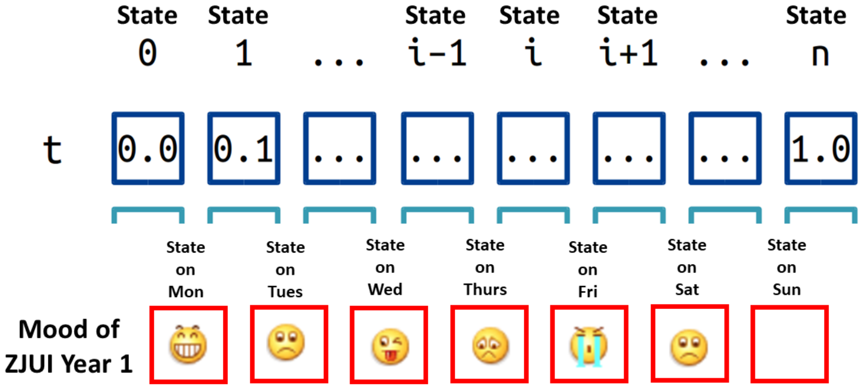
Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

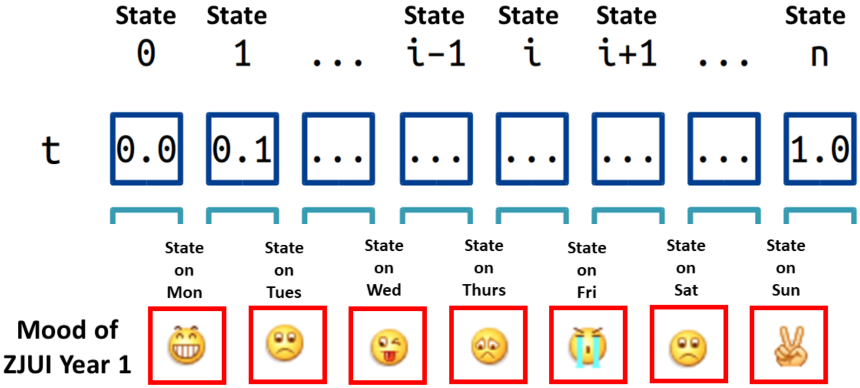
Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

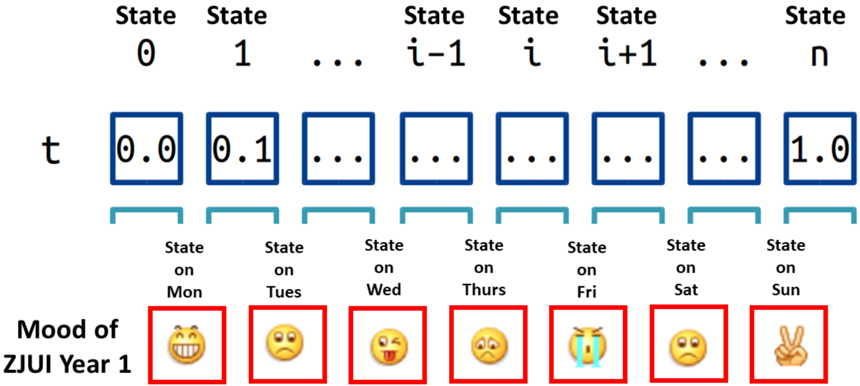
Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

Model State


State 0
State 1
...
State i-1
State i
State i+1
...
State n
t
0.0
0.1
...
...
...
...
...
1.0
State on Mon
State on Tues
State on Wed
State on Thurs
State on Fri
State on Sat
State on Sun
Mood of ZJUI Year 1

Model State



Print( ) or Plot( ) to debug and know what happens at each state!

Modeling

What if you do not have an analytical solution?

Modeling

What if you do not have an analytical solution?

B Use “finite difference” equation otherwise.

Modeling

What if you do not have an analytical solution?

B Use "finite difference" equation otherwise.

$$v_n(t) = \frac{dy_n}{dt} \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} \rightarrow y_{n+1} = y_n + v_n\left(t_{n+1} - t_n\right)$$

$$a = \frac{dv_n}{dt} \approx \frac{v_{n+1} - v_n}{t_{n+1} - t_n} \rightarrow v_{n+1} = v_n + a\left(t_{n+1} - t_n\right)$$

$$v_{n=0} = 0 \qquad y_{n=0} = 1 \qquad a = -9.8$$

subject to

$$y(t) \geq 0$$

1. We break the whole problem into n number of small parts.
2. n starts from 0 to any number we want.
3. Each n is a different state of the solution.

Numerical Methods

A. Numerical Differentiation:

Forward difference, Backward difference, and others....

I) Forward difference,

$$\frac{dy_n}{dt} \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n}$$

II) Backward difference,

$$\frac{dy_n}{dt} \approx \frac{y_n - y_{n-1}}{t_n - t_{n-1}}$$

B. Numerical Integration:

Trapezoidal Rule, Simpson Rule, and many more...

Modeling

Input

```
# Parameters of simulation
n = 100      # number of data points to plot
start = 0.0 # start time, s
end = 1.0    # ending time, s
a = -9.8     # acceleration, m*s**-2

# State variable initialization
t = np.linspace( start,end,n+1 )     # time, s
y = np.zeros( n+1 )                  # height, m
v = np.zeros( n+1 )                  # velocity, m*s**-1
y[ 0 ] = 1.0                         # initial condition, m
```

Model

```
for i in range( 1,n+1 ):
    v[ i ] = v[ i-1 ] + a*( t[ i ]-t[ i-1 ] )
    y[ i ] = y[ i-1 ] + v[ i ] * ( t[ i ]-t[ i-1 ] )

    if y[ i ] <= 0: # ball has hit the ground
        v[ i ] = 0
        y[ i ] = 0
```

Which method?

Use analytical equation

$$y(t) = y_0 + v_0 t + \frac{a}{2} t^2$$

```
import numpy as np

# Parameters of simulation
n = 100      # number of data points to plot
start = 0.0 # start time, s
end = 1.0    # ending time, s
a = -9.8     # acceleration, m*s**-2

# State variable initialization
t = np.linspace(start,end,n+1) # time, s

y = 1.0 + a/2 * t**2

for i in range(1,n+1):
    if y[i] <= 0: # ball has hit the ground
        y[i] = 0
```

VS

Use "finite difference" equation otherwise.

$$\frac{dy}{dt} = v_n(t) \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} \rightarrow y_{n+1} = y_n + v_n\,(t_{n+1} - t_n)$$

$$\frac{dv}{dt} = a \approx \frac{v_{n+1} - v_n}{t_{n+1} - t_n} \rightarrow v_{n+1} = v_n + a\,(t_{n+1} - t_n)$$

```
import numpy as np
# Parameters of simulation
n = 100     # number of data points to plot
start = 0.0 # start time, s
end = 1.0   # ending time, s
a = -9.8    # acceleration, m*s**-2

# State variable initialization
t = np.linspace( start,end,n+1 )    # time, s
y = np.zeros( n+1 )                 # height, m
v = np.zeros( n+1 )                 # velocity, m*s**-1
y[ 0 ] = 1.0                        # initial condition, m

for i in range( 1,n+1 ):
    v[ i ] = v[ i-1 ] + a*( t[ i ]-t[ i-1 ] )
    y[ i ] = y[ i-1 ] + v[ i ] * ( t[ i ]-t[ i-1 ] )

    if y[ i ] <= 0: # ball has hit the ground
        v[ i ] = 0
        y[ i ] = 0
```

Modeling

A How would you make the ball bounce?

Modeling

A How would you make the ball bounce? (Reverse the direction of the velocity at the ground; have a decay factor.)

Modeling

A How would you make the ball bounce? (Reverse the direction of the velocity at the ground; have a decay factor.)

B How would you include lateral motion?

Modeling

A How would you make the ball bounce? (Reverse the direction of the velocity at the ground; have a decay factor.)

B How would you include lateral motion? (Have separate x- and y-positions and velocities.)

Using Model State

From **hw07**
Consider a ball falling from the edge of a table.
Describe its path and how many times it bounces.

Most get bounces = 340 but model answer = 37. Why ??!!

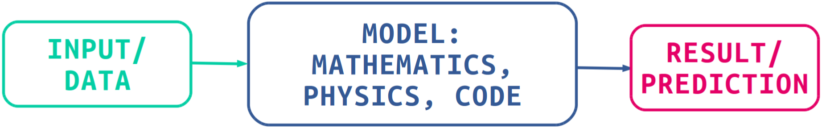
Using Model State

From **hw07**
Consider a ball falling from the edge of a table.
Describe its path and how many times it bounces.

Most get bounces = 340 but model answer = 37. Why ??!!

Check using the states!

Modeling similar to data pipeline



Modeling - Problem

PROBLEM

- Specify the problem, relevant physical constraints
- First problem statement is often ill-defined
- If too big, need to break into clear measurable sub-problems

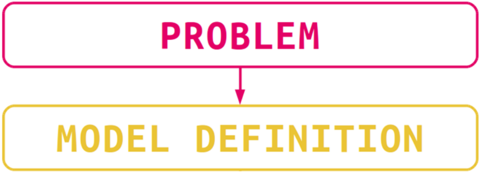
Modeling - Problem

PROBLEM

- Specify the problem, relevant physical constraints
- First problem statement is often ill-defined
- If too big, need to break into clear measurable sub-problems

- Ex. Model a ball falling
  - From where? At where?
- Ex. Model the relationship of current and voltage
  - Of what?

Modeling - Define Model



- Includes physical and mathematical equation
- Boundary conditions
- To create a program to implement the model in code

Modeling - Define Model

PROBLEM

↓

MODEL DEFINITION

- Includes physical and mathematical equation
- Boundary conditions
- To create a program to implement the model in code

- Ball bouncing off the table

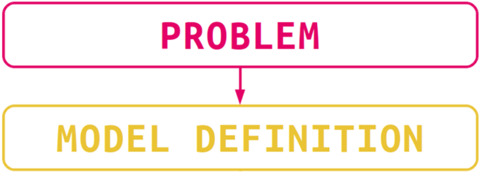
$$y_{n+1} = y_n + v_n \left(t_{n+1} - t_n\right)$$

$$v_{n+1} = v_n + a \left(t_{n+1} - t_n\right)$$

or

$$y(t) = y_0 + v_0 t + \frac{a}{2} t^2$$

Modeling - Define Model



- Includes physical and mathematical equation
- Boundary conditions
- To create a program to implement the model in code

- Ball bouncing off the table

$$y_{n+1} = y_n + v_n\,(t_{n+1} - t_n)$$

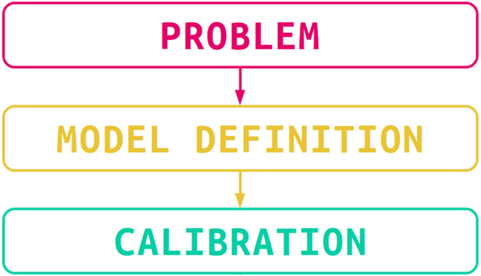
$$v_{n+1} = v_n + a\,(t_{n+1} - t_n)$$

or

$$y(t) = y_0 + v_0 t + \frac{a}{2} t^2$$

- Current vs Voltage in conducting materials
  - V = IR

Modeling - Calibration

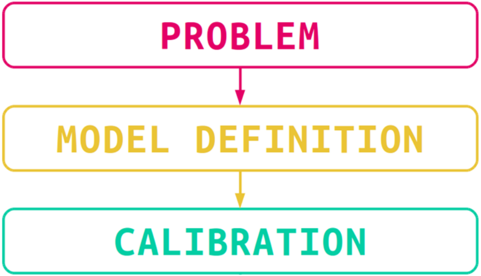


Calibration - test our model/program if it works as we think it will.

- Verification - check if you are solving the correct problem **(Are you solving the right problem?)**
- Validation - check if the problem is solved correctly **(Are you solving it right?)**

- Test with small samples with known solutions

Modeling - Calibration



Calibration - test our model/program if it works as we think it will.

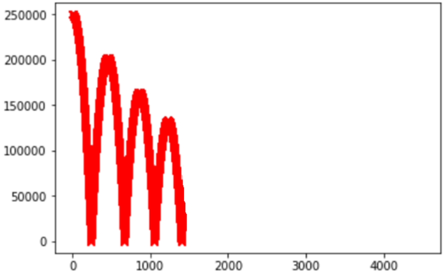
- Verification - check if you are solving the correct problem **(Are you solving the right problem?)**
- Validation - check if the problem is solved correctly **(Are you solving it right?)**

- Test with small samples with known solutions

- Verification first or Validation first?

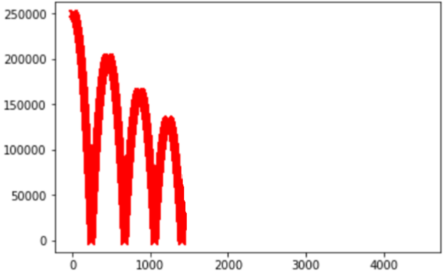
Modeling - Calibration

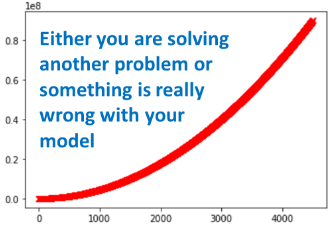
For **hw07**:



Modeling - Calibration

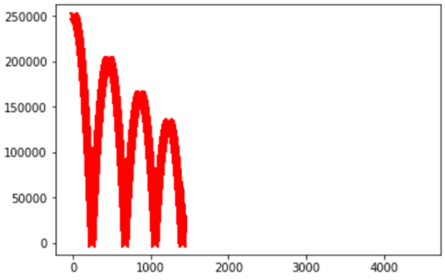
From **hw07**:

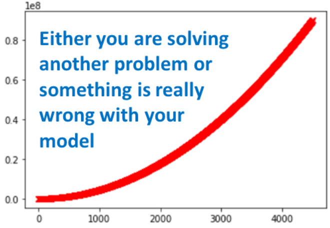


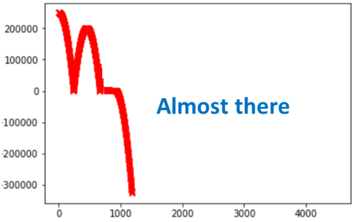


Modeling - Calibration

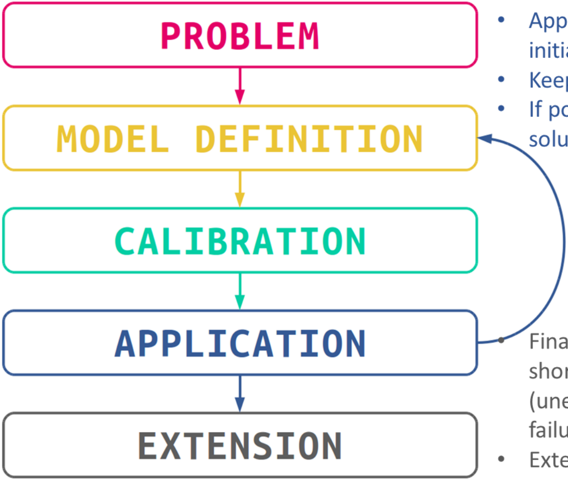
From **hw07**:







Modeling - Extend/Correct



- Apply the model to solve the initial problem.
- Keep track of errors
- If possible, perform a thorough solution analysis.

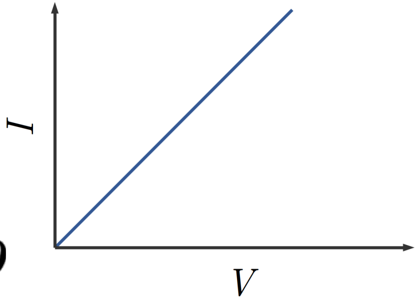
Finally, we can encounter shortcomings and surprises (unexpected discoveries or failure modes)

- Extend/Correct our model

Model Ohms Law

$$V = IR$$

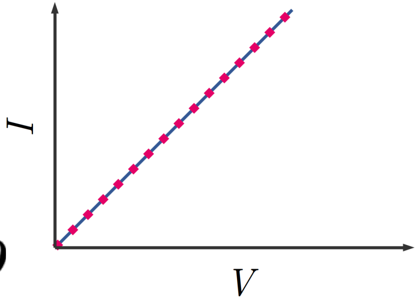
$$y = mx + b$$



Modeling - All metals?

$$V = IR$$

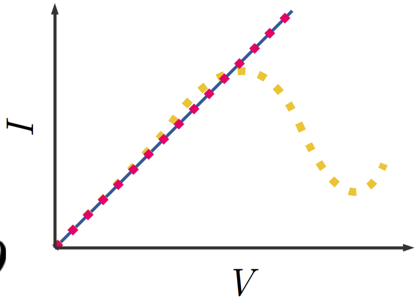
$$y = mx + b$$



Modeling - All metals?

$$V = IR$$

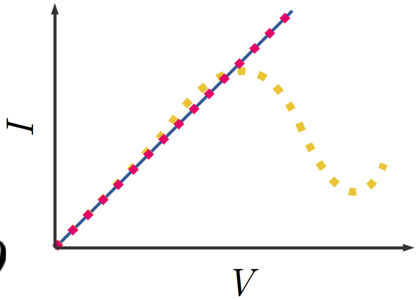
$$y = mx + b$$



Modeling - All metals?

$$V = IR$$

$$y = mx + b$$



If you can extend/correct the model to explain why, then you will probably get a **Nobel Prize in Physics**