Numerical Python

CS*101* lec11

Plotting

2019-10-23

Announcements

quiz: `quiz11` due on Thurs 24/10

lab: `lab` running in 100 meters race. No Lab

hw: `hw06` due Wed 30/10

exam: `exam02` coming in Nov

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. Create basic plots of several types using MatPlotLib. => Using **lec10 Numpy** type as data

B. Understand (and be able to repeat) the import process for MatPlotLib.

C. Display simulation results in an intelligible fashion as a plot. => Needed everywhere in Engineering

numpy Recap

Main point

0. In numpy, the operators and functions normally work element-wise

1. `x = np.zeros(5) = np.zeros((5)) = np.zeros([5])` creates a 1D np.array
2. You can only do `x[i]` where i = 0 to 4

Main point

0. In numpy, the operators and functions normally work element-wise

1. `x = np.zeros(5) = np.zeros((5)) = np.zeros([5])` creates a 1D np.array
2. You can only do `x[i]` where i = 0 to 4

Compare with

3. `x = np.zeros([1,5]) = np.zeros((1,5))` creates a 2D np.array of 1 row and 2 columns
4. You can do `x[i,j]` where i = 0 and j = 0 to 4

or `x[i]` where i = 0 which shows the whole row

Main point

0. In numpy, the operators and functions normally work element-wise

1. `x = np.zeros(5) = np.zeros((5)) = np.zeros([5])` creates a 1D np.array
2. You can only do `x[i]` where i = 0 to 4

Compare with

3. `x = np.zeros([1,5]) = np.zeros((1,5))` creates a 2D np.array of 1 row and 2 columns
4. You can do `x[i,j]` where i = 0 and j = 0 to 4

or `x[i]` where i = 0 which shows the whole row

** 1D vs 2D array is true for other commands like `np.array([1,2,3])` vs `np.array([[1,2,3]])`

** Use `([ ])` or `([[ ]])` to create array

numpy

```
>>> x.max(i)
#max by column if i=0, by row if i=1
#max of everything in x if nothing

>>> x.min(i)
#min by column if i=0, by row if i=1
#min of everything in x if nothing

>>> x.mean(i)
#mean by column if i=0, by row if i=1
#mean of everything in x if nothing
```

Question 1

$$x = \begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix}$$

What will produce this array?

A `np.array([[1,2,3],[1,2,3]])`

B `np.array([2,3])`

C `np.array([3,2])`

D `np.array([[1,1],[2,2],[3,3]])`

Question 1

$$x = \begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix}$$

What will produce this array?

A `np.array([[1,2,3],[1,2,3]])`

B `np.array([2,3])`

C `np.array([3,2])`

D `np.array([[1,1],[2,2],[3,3]]) ***`

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

1. x.sort(0)? (by column)

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

1. x.sort(0)? (by column)

$$x = \begin{pmatrix} 2 & 1 \\ 3 & 1 \\ 9 & 3 \end{pmatrix}$$

2. x.argsort(0)

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

1. x.sort(0)? (by column)

$$x = \begin{pmatrix} 2 & 1 \\ 3 & 1 \\ 9 & 3 \end{pmatrix}$$

2. x.argsort(0)

$$\begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{pmatrix}$$

x NOT changed!

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

3. x.sort(1)? (by row)

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

3. x.sort(1)? (by row)

$$x = \begin{pmatrix} 1 & 9 \\ 1 & 2 \\ 3 & 3 \end{pmatrix}$$

4. x.mean(1) (by row)

Question 2

$$x = \begin{pmatrix} 9 & 1 \\ 2 & 1 \\ 3 & 3 \end{pmatrix}$$

What will be

3. x.sort(1)? (by row)

$$x = \begin{pmatrix} 1 & 9 \\ 1 & 2 \\ 3 & 3 \end{pmatrix}$$

4. x.mean(1) (by row)

```
array([5., 1.5, 5.])
```

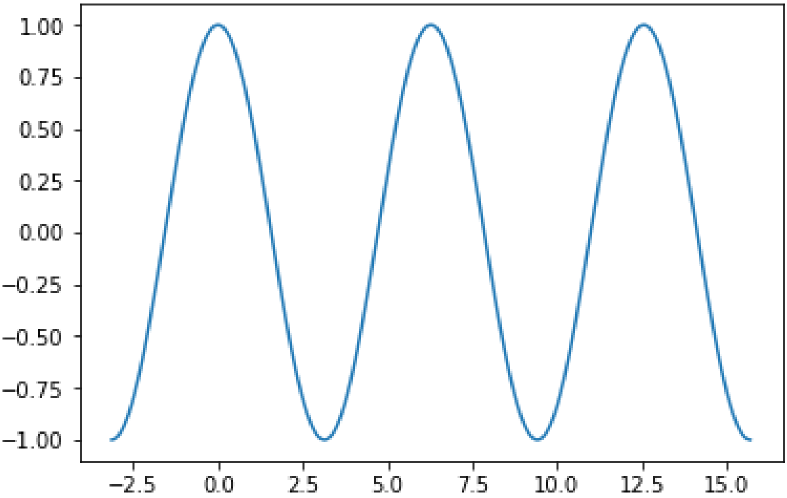
Plotting

Why plot?

Why plot?

```
X = ([-3.14159265, -3.11695271, -3.09231277, -3.067
       -3.01839294, -2.993753  , -2.96911306, -2.94
       -2.89519323, -2.87055329, -2.84591335, -2.82
      ... (1000 lines)
        2.89519323,  2.91983317,  2.94447311,  2.96
        3.01839294,  3.04303288,  3.06767283,  3.14

Y = ([-1.        , -0.99969645, -0.99878599, -0.997
       -0.99242051, -0.98909161, -0.98516223, -0.98
       -0.96979694, -0.96349314, -0.95660442, -0.94
      ... (1000 lines)
       -0.96979694, -0.97551197, -0.98063477, -0.98
       -0.99242051, -0.99514692, -0.99726917, -1.
```

Why plot?



matplotlib

```
import matplotlib.pyplot as plt
# add this for jupyter only
%matplotlib inline
```

matplotlib

```
import matplotlib.pyplot as plt
# add this for jupyter only
%matplotlib inline
```

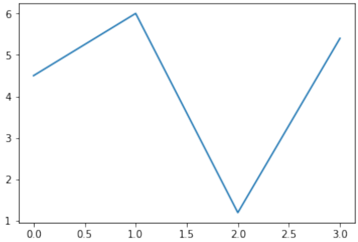
> A plotting environment similar to MATLAB.
> Can plot `list`s, `np.array`s or most containers.

matplotlib

```
import matplotlib.pyplot as plt
# add this for jupyter only
%matplotlib inline
```

> A plotting environment similar to MATLAB.
> Can plot `lists`, `np.arrays` or most containers.

```
xs = list( range(4) )
ys = [ 4.5, 6.0, 1.2, 5.4 ]
plt.plot( xs, ys )
plt.show()
```



matplotlib

One kind of plots today:

```
> plt.plot( x,y ) # for point-wise data
```

matplotlib

Basic cycle:

1. Add data to plot.
2. Plot.
3. Show plot.

plt.plot

Assuming you have a lot of data pairs X, C and X, S and X, Y

Plot

```
import matplotlib.pyplot as plt
plt.plot(X, C, color="blue", linewidth=1.0,
                linestyle="-", label="Solid")
plt.plot(X, S, color="red", linewidth=3,
                linestyle="--", label="Dot")
plt.plot(X, Y, 'ko', label="oo")
```

Set x and y limits for display

```
plt.xlim(-4.0,4.0)
plt.ylim(-1.0,1.0)
```

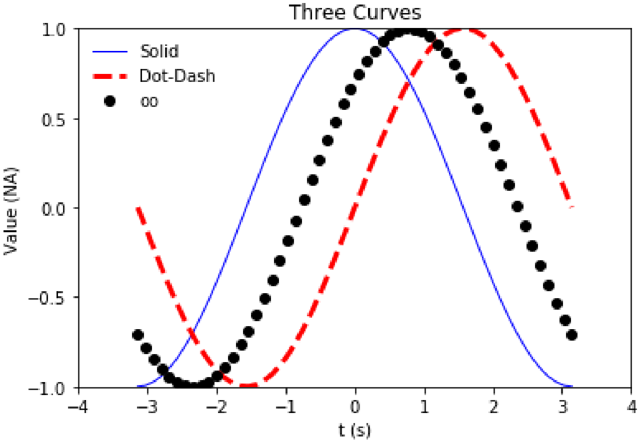
Set x and y ticks intervals

```
plt.xticks(np.linspace(-4,4,9,endpoint=True))
plt.yticks(np.linspace(-1,1,5,endpoint=True))
```

plt.plot

```
# Adding x-axis and y-axis labels and a title
plt.xlabel( 't (s)' )
plt.ylabel( 'Value (NA)' )
plt.title( 'Three Curves' )

# Adding a legend
plt.legend(loc='upper left', frameon=False)

# Save figure using 72 dots per inch
plt.savefig("filePath/ex2.png",dpi=72)

# Show result on screen
plt.show()
```

plt.plot



Where are the `xlim`, `ylim`, `legend`, `xticks`, `yticks`, `title`, `xlabel`, `ylabel`?

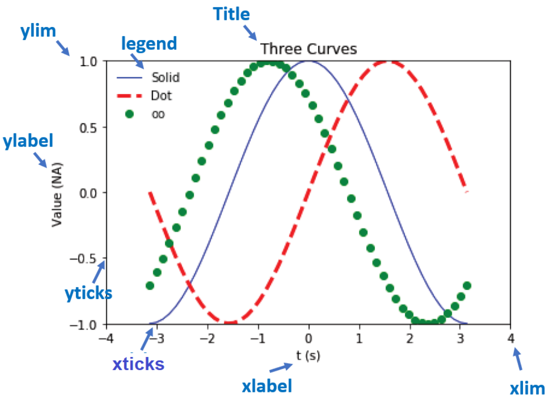
plt.plot

You have plotted an invisible graph

if you see it, you have x-ray eyes

if not, move to the next page for your answers

plt.plot



Note: The xlim and ylim refers to both ends.
xticks give the positions of interval across x-axis

plt

Always include labels:

> `plt.xlabel( 'domain (units)' )`

> `plt.ylabel( 'range (units)' )`

> `plt.title( 'topical data' )`

(We may omit this in lecture for convenience.)

```
plt.plot( xs,ys )
plt.xlabel( 'x' )
plt.ylabel( 'y' )
plt.title( 'some values' )
plt.show()
```

Why use numpy as input?

Plot $sin(x)$ **for** $x \in [0, 2\pi]$

1. Pure Python:

```
from math import pi
x = []    # can't use range easily!
for i in range(100):
    x.append( 2*pi*i/100 )
from math import sin
y = []
for j in range(100):
    y.append( sin(x[j]) )

plt.plot( x,y,'k-' )
plt.xlim( 0,2*pi )
plt.ylim( -1,1 )
plt.show()
```

Why use numpy as input?

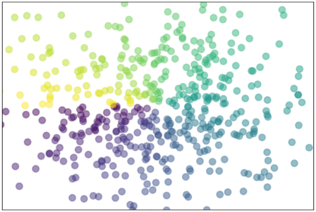
Plot $sin(x)$ **for** $x \in [0, 2\pi]$

2. numpy:

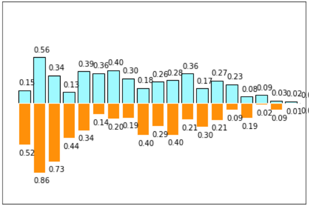
```
import numpy as np
x = np.linspace( 0,2*np.pi,101 )
y = np.sin( x )

plt.plot( x,y,'k-' )
plt.xlim( 0,2*pi )
plt.ylim( -1,1 )
plt.show()
```

Other than .plot ?

> `.scatter` - plot of points (x,y)

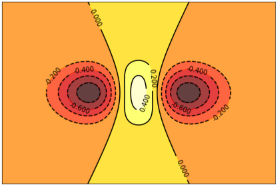


> `.bar` - bar chart

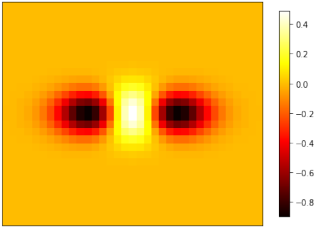


Other plot types?

> `.contour` - identical values are connected together. Like in a physical map

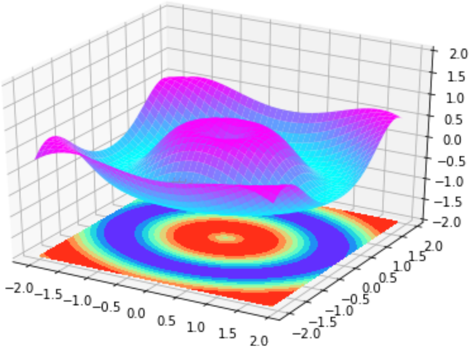


> `. imshow` - show an image or plot a collection of values in one array



Other plot types?

> `.plot_surface` - plot of a 3D surface



> animation

Modeling (next lecture)

Modeling

Help to simplify a complicated problem

Based on mathematical equations and physical laws

Gives an ”Ideal” solution

But... it is not EXACTLY correct!

Modeling

”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.

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

```
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
```

Modeling

A Use “finite difference” equation otherwise.

Modeling

A Use "finite difference" equation otherwise.

$$\frac{dy}{dt} = v(t) \approx \frac{y^{n+1} - y^n}{t^{n+1} - t^n} \rightarrow y^{n+1} = y^n + v\left(t^{n+1} - t^n\right)$$

$$\frac{dv}{dt} = a \approx \frac{v^{n+1} - v^n}{t^{n+1} - t^n} \rightarrow v^{n+1} = v^n + a\left(t^{n+1} - t^n\right)$$

$$v^{n=0} = 0 \qquad y^{n=0} = 1 \qquad a = -9.8$$

subject to

$$y(t) \geq 0$$

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

Modeling

```
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.)