

Python Lecture 3: Defining Functions, Plotting with Matplotlib, and Conditional Programming

Computing in Statistics

Dr. Yen-Yi Ho

Department of Statistics, University of South Carolina

Outline

1. Making plots with Matplotlib
2. Defining new functions
3. Conditional programming (`if` / `elif` / `else`)
4. Putting it together: piecewise functions
5. Practice

Making Plots: Getting Started

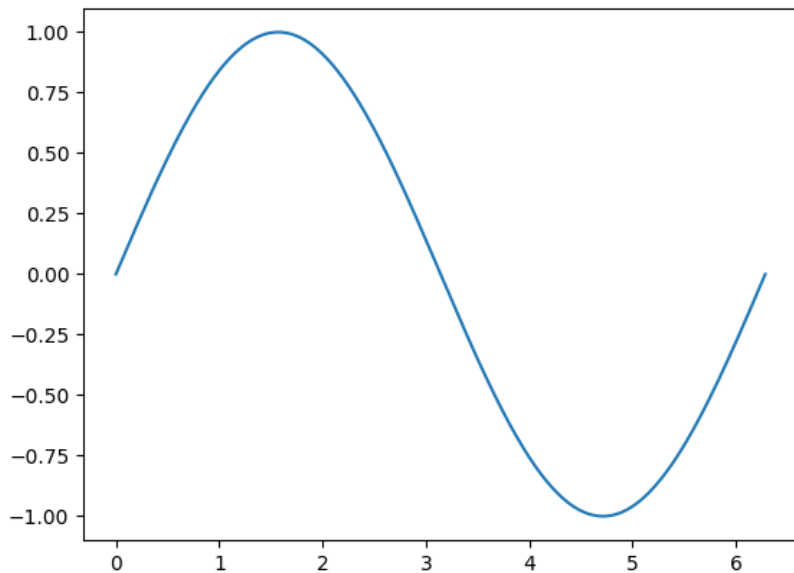
To make plots, we import **matplotlib** — the most widely used Python plotting library (matplotlib.org).

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2*np.pi, 361)
sinx = np.sin(x)
plt.plot(x, sinx)
plt.show()
```

- ▶ `plt.plot(x, y)` draws a line plot
- ▶ `plt.show()` displays the figure

Result



Adding Features and a Legend

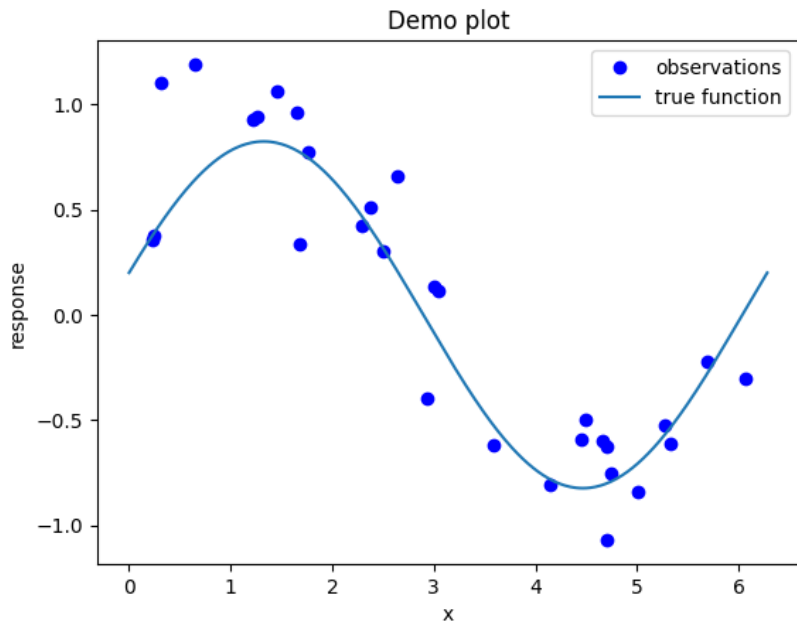
```
u = np.random.random(30) * 2*np.pi
v = 0.8*np.sin(u) + 0.2*np.cos(u) + np.random.normal(0, 1/3)

x = np.linspace(0, 2*np.pi, 361)
fx = 0.8*np.sin(x) + 0.2*np.cos(x)

plt.figure() # start a new, empty figure
plt.plot(u, v, 'bo', label='observations')
plt.plot(x, fx, label='true function')
plt.xlabel("x"); plt.ylabel("response")
plt.title("Demo plot")
plt.legend()
plt.show()
```

A label= on each plotted feature builds the legend automatically.

Result



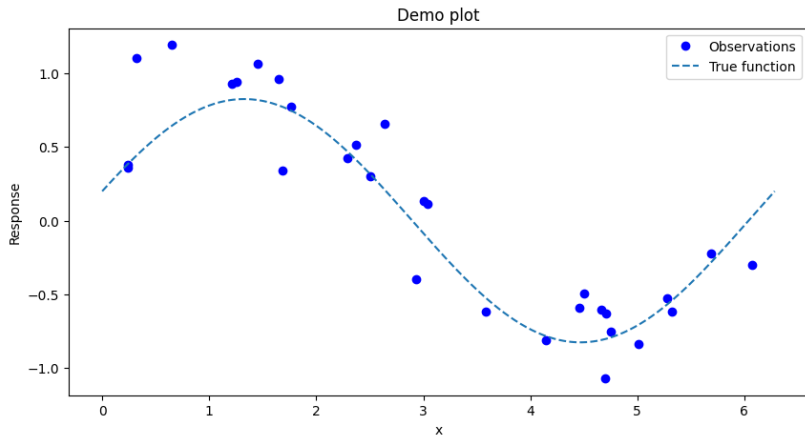
The Figure/Axes Approach

The more flexible — and more common in tutorials — way to build a plot: set up a **figure** and a set of **axes** first, then add to the axes.

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(u, v, 'bo', label='Observations')
ax.plot(x, fx, label='True function', linestyle="--")
ax.set_xlabel("x")
ax.set_ylabel("Response")
ax.set_title("Demo plot")
ax.legend()
plt.show()
```

Because a new figure is created, plotting commands never get appended to a previous plot by accident.

Result



Multi-Panel Figures

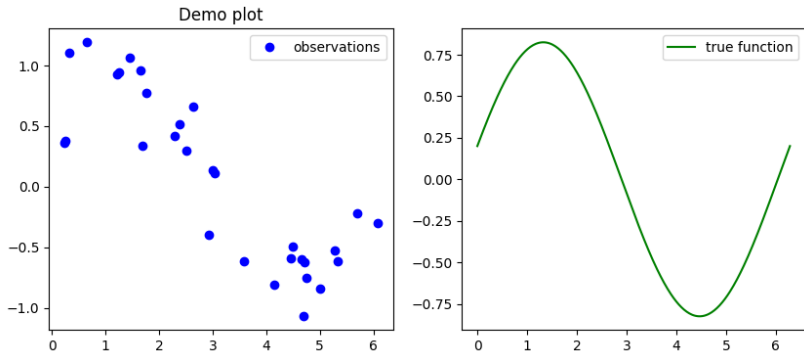
Setting up the figure first lets us build figures with several panels (each panel is an **axes**):

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

ax1.plot(u, v, 'bo', label='observations')
ax1.set_title("Demo plot"); ax1.legend()

ax2.plot(x, fx, 'g', label='true function')
ax2.legend()
plt.show()
```

Result



A “mosaic” arrangement (`plt.subplot_mosaic`) allows even more flexible panel layouts, with panels referenced by name instead of position.

Outline

1. Making plots with Matplotlib
2. **Defining new functions**
3. Conditional programming (`if` / `elif` / `else`)
4. Putting it together: piecewise functions
5. Practice

Defining a Simple Function

A one-liner function:

```
def f(x): return x * np.sin(x)
```

```
f(1)
```

0.8414709848078965

i Note

Key rule: A Python function must return a value explicitly, or it returns nothing!

Defining a Python Function

With more than one line, we **indent** the body — no curly braces as in R. The function ends when the indentation stops.

```
def hyp(a, b):  
    csq = a**2 + b**2  
    c = np.sqrt(csq)  
    return c
```

Returning Multiple Values

List every object to return, separated by commas:

```
def eucl(r, th):  
    x = r * np.cos(th)  
    y = r * np.sin(th)  
    return x, y  
  
x, y = eucl(2, np.pi/3)
```

Python automatically packages multiple return values into a tuple, which we can unpack on the left-hand side — just as we did with multiple assignment earlier.

Default Argument Values

Default values work just as they do in R:

```
def logistic(x, a=0, b=1):  
    l    = a + x*b  
    val = np.exp(l) / (1 + np.exp(l))  
    return val  
  
x    = np.linspace(-4, 4, 200)  
fx1 = logistic(x)           # uses defaults a=0, b=1  
fx2 = logistic(x, a=-2, b=3) # override the defaults
```

Arguments can be supplied positionally or by name (`a=-2`); naming them makes calls more readable.

Outline

1. Making plots with Matplotlib
2. Defining new functions
3. **Conditional programming (if / elif / else)**
4. Putting it together: piecewise functions
5. Practice

if / else

The logic of conditional programming is the same in every language — only the syntax changes.

```
def rolldice():  
    r1 = np.random.choice(np.arange(1, 7), 1)  
    r2 = np.random.choice(np.arange(1, 7), 1)  
  
    if r1 == r2:  
        print("Doubles!")  
    else:  
        print("No match.")  
  
    print(r1, r2)
```

Just as with functions and loops, the body of an if/else block is defined by **indentation**, not braces.

elif: "Else If"

Python spells "else if" as a single word, elif:

```
def rolldice():
    r1 = np.random.choice(np.arange(1, 7), 1)
    r2 = np.random.choice(np.arange(1, 7), 1)

    if r1 == r2:
        if r1 == 1:
            print("Snake-eyes!")
        elif r1 == 6:
            print("Can't beat it!")
        else:
            print("Doubles!")
    elif (r1 + r2) <= 7:
        print("Sad rollin'.")
    else:
        print("Mighty fine!")
```

Conditions can be **nested** just like in R.

Booleans as Numbers

The function `bool()` coerces its argument to True/False:

```
bool(1)    # True  
bool(0)    # False
```

Just as in R, a logical value coerces to 0 or 1 in arithmetic. This lets us write **piecewise-defined functions** compactly, without an explicit `if` statement, using indicator-style expressions.

Outline

1. Making plots with Matplotlib
2. Defining new functions
3. Conditional programming (if / elif / else)
4. **Putting it together: piecewise functions**
5. Practice

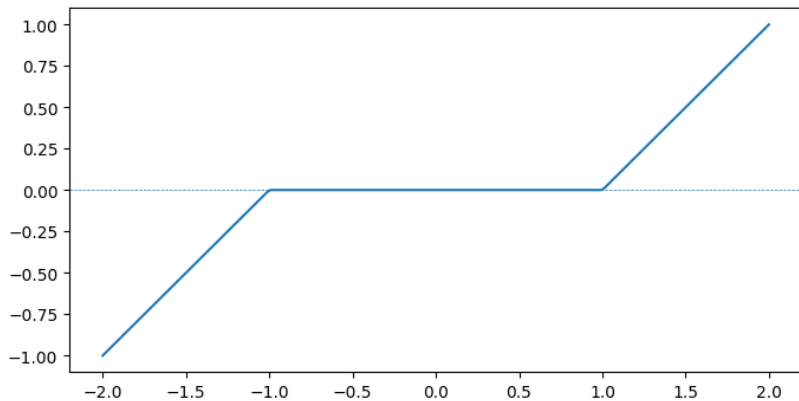
Example: Soft-Thresholding

$$S(x, \lambda) = \begin{cases} x + \lambda, & x < -\lambda \\ 0, & -\lambda \leq x \leq \lambda \\ x - \lambda, & \lambda < x \end{cases} \quad (\lambda > 0)$$

```
def softthresh(x, lam):  
    return (x + lam)*(x < -lam) + (x - lam)*(x > lam)  
  
x = np.linspace(-2, 2, 200)  
fx = softthresh(x, lam=1)  
  
fig, ax = plt.subplots(figsize=(8, 4))  
ax.plot(x, fx)  
ax.axhline(0, linestyle="--", linewidth=0.5)  
plt.show()
```

The boolean expressions $(x < -\text{lam})$ etc. act as 0/1 indicators.

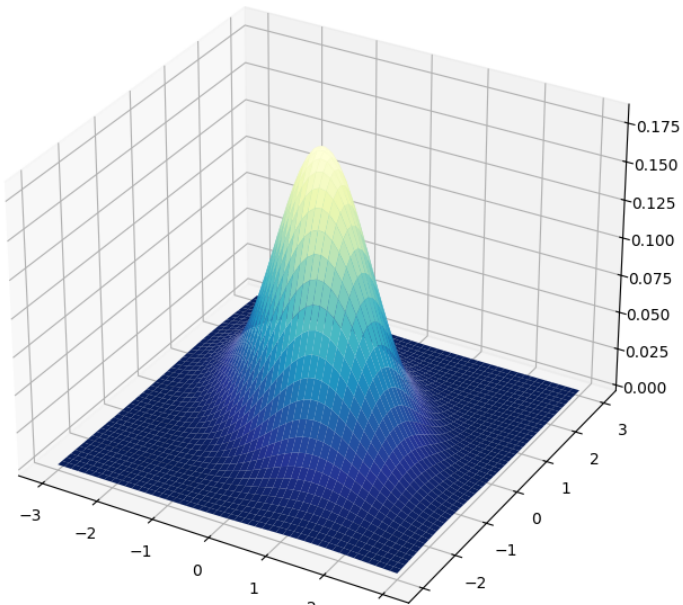
Result



Example: A Bivariate Density Surface

```
def bivn(z1, z2, rho=0):  
    r = 1 - rho**2  
    a = 1 / (2*np.pi*np.sqrt(r))  
    b = (z1**2 - 2*rho*z1*z2 + z2**2) / r  
    return a * np.exp(-b/2)  
  
gs = 50  
z = np.linspace(-3, 3, gs)  
X, Y = np.meshgrid(z, z) # build a 2-d grid from 1  
Z = bivn(X, Y, rho=-0.5) # evaluate the function on  
  
fig = plt.figure(figsize=(8, 8))  
ax = fig.add_subplot(projection='3d')  
ax.plot_surface(X, Y, Z, cmap=plt.cm.YlGnBu_r)  
plt.show()
```

Result



Outline

1. Making plots with Matplotlib
2. Defining new functions
3. Conditional programming (`if` / `elif` / `else`)
4. Putting it together: piecewise functions
5. **Practice**

Practice: Write a function for Bootstrap Median

```
def bootstrap_ci(x, n_boot=2000, seed=42, alpha=0.05):
    rng = np.random.default_rng(seed)
    x = np.asarray(x, dtype=float)
    n = x.size
    obs_median = float(np.median(x))
    boot_meds = np.empty(n_boot)          # pre-allocate storage

    for b in range(n_boot):
        idx = rng.integers(0, n, size=n) # 0-based, high exclusive
        x_star = x[idx]
        boot_meds[b] = np.median(x_star)

    lo, hi = np.quantile(boot_meds, [alpha/2, 1 - alpha/2])
    return {"estimate": obs_median, "ci_lower": float(lo),
            "ci_upper": float(hi), "boot_dist": boot_meds}

bmi = np.round(np.random.default_rng(1).gamma(30.0, 1.0, size=5000), 1)
res = bootstrap_ci(bmi, n_boot=2000, seed=42)
res
```

```
{'estimate': 29.6,
 'ci_lower': 29.4,
 'ci_upper': 29.8,
 'boot_dist': array([29.5, 29.9, 29.7, ..., 29.8, 29.5, 29.5])}
```

Practice: Read Code

Predict the output of the following code before running it.

```
def star(p, s):  
    th = np.linspace(np.pi/2, np.pi/2 + 2*np.pi*s, s*p + 1)  
    x = np.cos(th[:,s])  
    y = np.sin(th[:,s])  
    return x, y  
  
x, y = star(5, 2)  
  
fig, ax = plt.subplots(figsize=(4, 4))  
ax.set_aspect('equal')  
ax.axis('off')  
ax.plot(x, y)  
plt.show()
```

Result

