Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

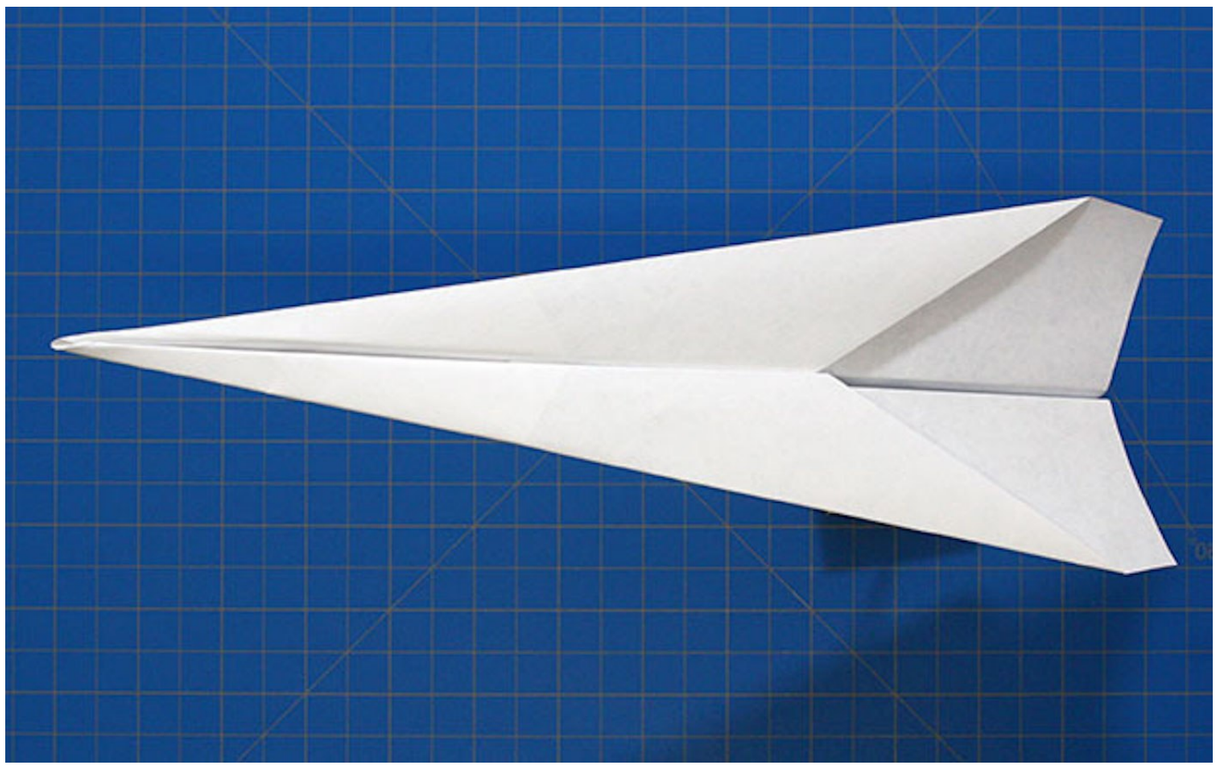
Lecture 20: Data analysis (Curve Fitting)

Goals for today

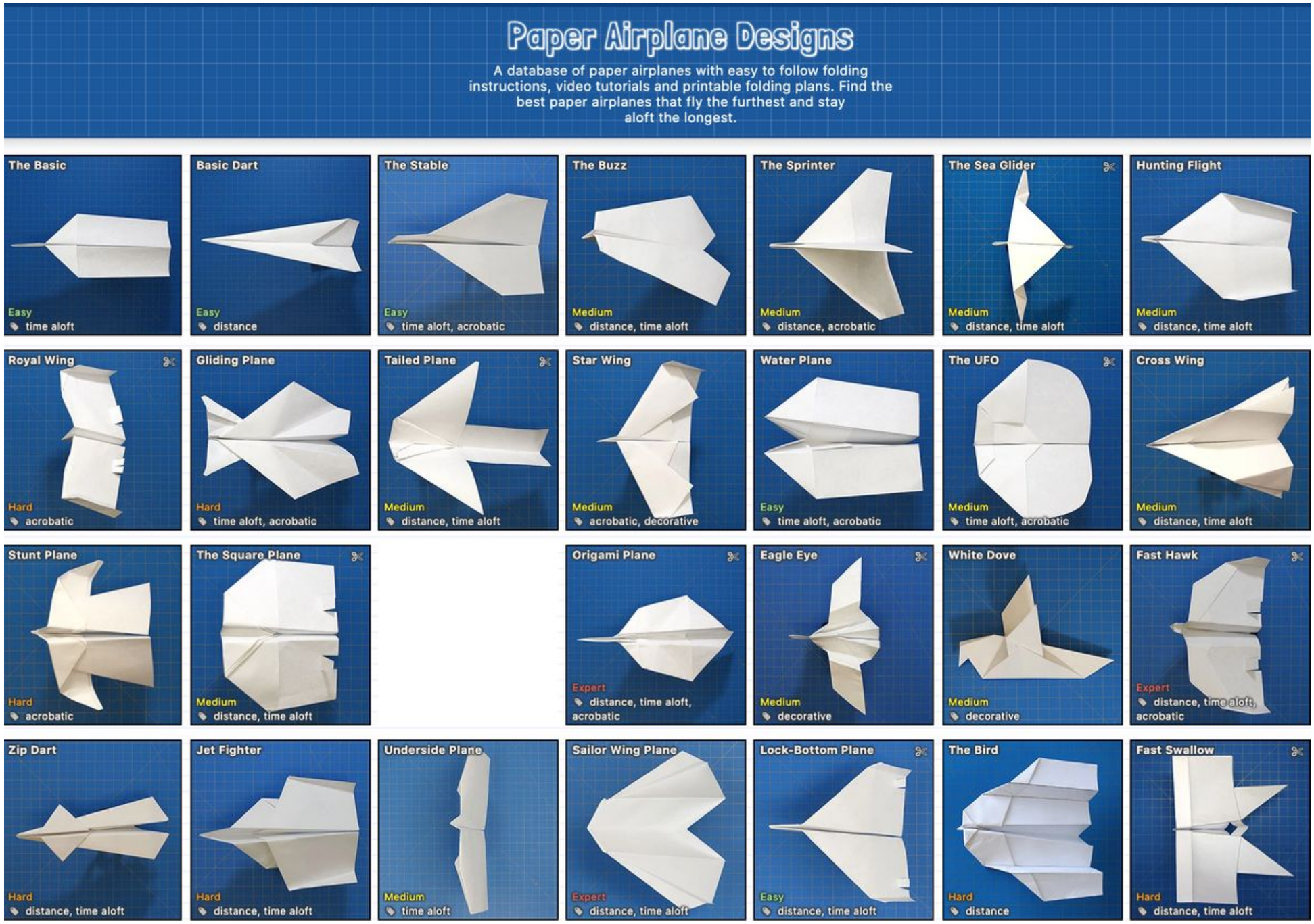
- Use the `numpy`, `datascience` and `matplotlib` libraries.
- Apply variety of methods for generating line of best fit.
- Translate mathematical expressions into Python code.

Usual module imports for our applications:

```
import numpy as np
import datascience as ds
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
```



Challenge: work in groups of 2-3 to make a paper airplane to fly as far as possible!



We'll now analyze flight distance versus wingspan.

- Measure distance from wingtip to wingtip (in cm).
- We'll measure the distance your airplane flies (in m).

Download the starter script on the course website: `paper-airplanes.py`. We can load data from the internet (and also a Google spreadsheet)!

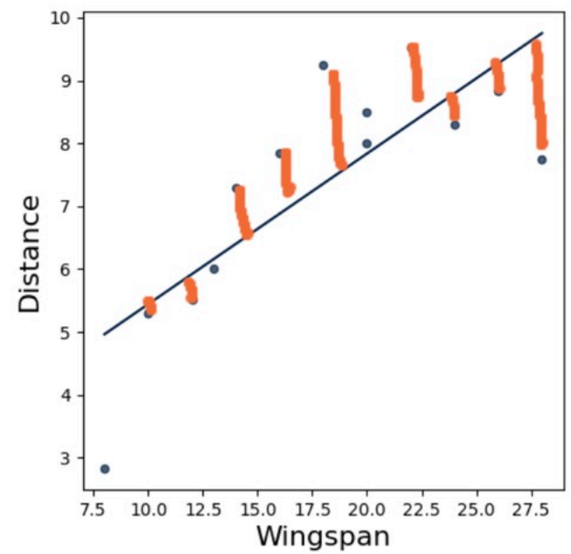
```
import numpy as np
import datascience as ds
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

# read existing flight data
data = ds.Table.read_table("https://philipclaude.github.io/csci146f25/classes/paper-airpla

# add our in-class experimental data
sheet_name = "PaperAirplaneExperiment"
sheet_url = "https://docs.google.com/spreadsheets/d/1t4GBPuhqjnxeNWBHBy1gRUsmkmW86MEn3OmSD
exp_data = ds.Table.read_table(sheet_url)
data.append(exp_data)

# plot the results
data.scatter("Wingspan", fit_line=True)
plt.show()
```

What is `fit_line=True` doing?



Let's make our own line of best fit:

1. $\text{slope} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$
2. $\text{intercept} = \bar{y} - \text{slope} \times \bar{x}$

$y = \text{slope} \cdot X + \text{intercept}$

$\bar{x}$: mean of X

Question 1: Which of the following snippets is equivalent to this expression for **numpy** arrays **x** and **y** ($\bar{x}$ is the mean of x).

$$\text{slope} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

A.

```
x_m = np.mean(x)
y_m = np.mean(y)
num = np.sum(x-x_m)*np.sum(y-y_m)
slope = num / np.sum((x-x_m)**2)
```

B.

```
x_m = np.mean(x)
y_m = np.mean(y)
ratio = (x-x_m)*(y-y_m) / (x-x_m)**2
slope = np.sum(ratio)
```

C.

```
x_m = np.mean(x)
y_m = np.mean(y)
num = np.sum((x-x_m)*(y-y_m))
slope = num / np.sum((x-x_m)**2)
```

D.

```
x_m = np.mean(x)
y_m = np.mean(y)
num = np.sum((x-x_m)*(y-y_m))
slope = num / np.sum(x-x_m)**2
```

Putting this together, we can calculate our line of best fit and plot it.

```
def regression_line(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    slope = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean) ** 2)
    intercept = y_mean - slope * x_mean
    return np.array([slope, intercept])

rline = regression_line(data["Wingspan"], data["Distance"])

x = data["Wingspan"]
y = rline[0] * x + rline[1] # evaluate the line at the existing x-values
data.scatter(data["Wingspan"], fit_line=True)
plt.plot(x, y, 'ro--') # should be overlaid on fit_line
plt.show()
```

The `slope` and `intercept` can also be derived from a "minimization" problem.

Trying to minimize the mean squared error:

$$\frac{1}{N}\sum_{i=1}^{N}\left(y_{i,\text{fit}} - y_{i,\text{known}}\right)^2$$

- For a straight line fit, $y_{i,\text{fit}} = \text{slope} \times x_i + \text{intercept}$.
- But we might also come up with something better (later).
- Then we can use a built-in optimizer (in `datascience`) to find the line parameters.

```
line_params = ds.minimize(fit_mse)

# Example: find the `x` that minimizes the following function
def my_fun_to_minimize(x):
  return x ** 2 - 2 * x + 1

x_min = ds.minimize(my_fun_to_minimize) # x_min = 1
```

Question 2: Which of the following snippets is equivalent to this expression for numpy arrays x and y?

$$\text{mse} = \frac{1}{N}\sum_{i=1}^{N}(\text{slope} \times x_i + \text{intercept} - y_{i,\text{known}})^2$$

A.

```
fit = slope*x + intercept
mse = np.mean((y-fit)**2)
```

B.

```
fit = slope*x + intercept
mse = np.sum((y-fit)**2) / len(slope)
```

C.

```
fit = slope*x + intercept
mse = np.mean(y-fit)**2
```

D.

```
fit = slope*x + intercept
mse = np.mean(y-fit**2)
```

Putting this together, we can give `datascience` a function that describes the MSE (to minimize).

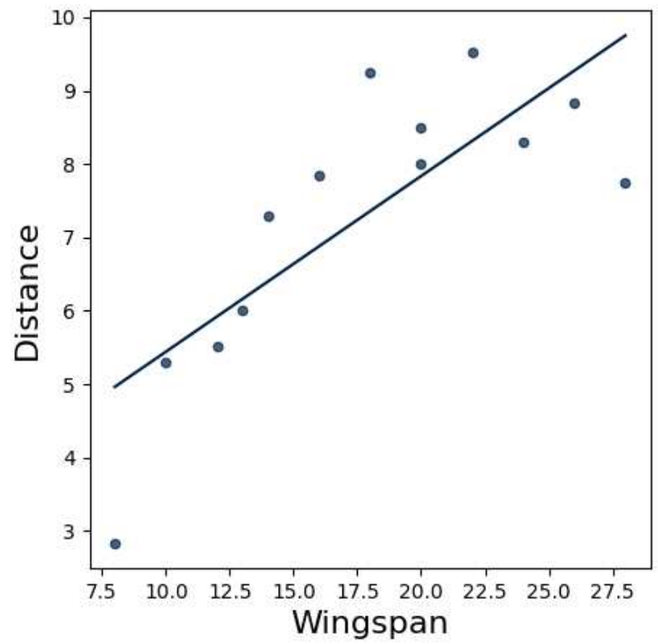
```
def mse_line(x, y):
    # Using a "nested" function here since we only need
    # it in the scope of mse_line.
    # ds will find the arguments to this function that minimize it
    # (i.e. it will find the slope and intercept)
    def fit_mse(slope, intercept):
        fitted = slope * x + intercept
        return np.mean((y - fitted) ** 2)

    # Use the minimize function to find the parameters that minimize
    # the output of the fit_mse function (passed as an argument)
    return ds.minimize(fit_mse)
```

Then:

```
>>> mse_line(data["Wingspan"], data["Distance"])
```

But this line doesn't really capture the data. Maybe a *quadratic* function will work better.



Same idea, but this time find a, b, c that minimizes (now $y_{i,\text{fit}} = ax_i^2 + bx_i + c$):

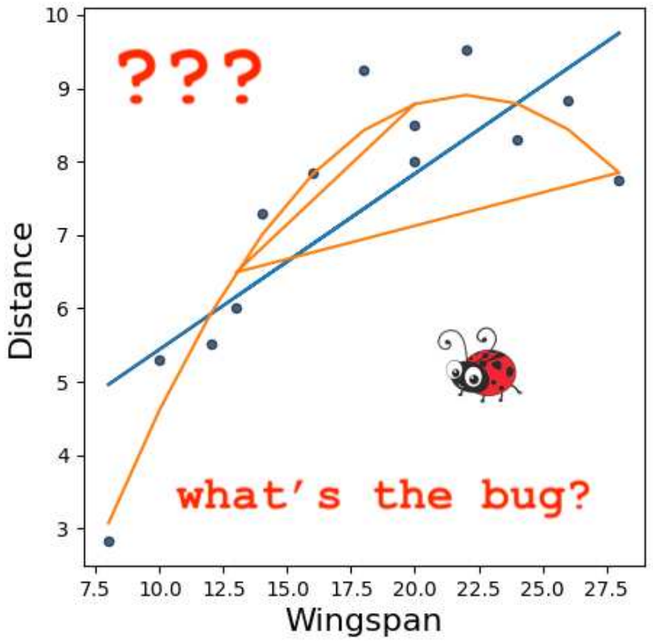
$$\frac{1}{N}\sum_{i=1}^{N}\left(y_{i,\text{fit}} - y_{i,\text{known}}\right)^2$$

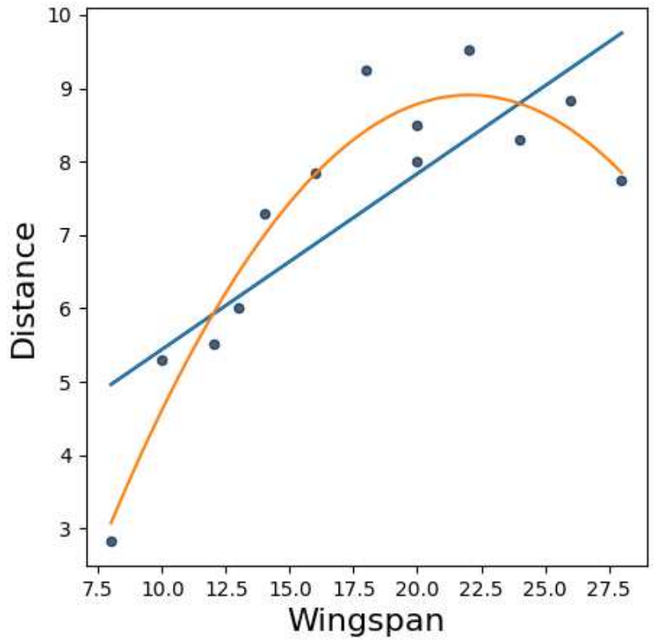
Exercise: extend the approach we used to do this in a function called `mse_quadratic`.

There are even built-in `numpy` functions to fit a general polynomial.

```
x = data["Wingspan"] # include in-class experimental data now
y = data["Distance"]
line = np.polyfit(x, y, 1)
quad = np.polyfit(x, y, 2)

plt.plot(x, line[0] * x + line[1])
plt.plot(x, quad[0] * x ** 2 + quad[1] * x + quad[2])
plt.show()
```





Reminders

- **Quiz 9** on Friday: see `PrairieLearn` problems for practice.
- Use vectorized code and built-in library functions whenever possible.
- **Test Project** is available (initial due date 12/08, final due date: 12/15).
  - No collaboration.
  - No Google searching or trying to find any parts of the solution online.
  - Please treat this as a "programming test". More details on the website.
- Programming Assignment 7 final due date this Thursday.
- Programming Assignment 9 initial due date this Friday.