variantvariantukbritish

My sample book

The Jupyter Book Community

Oct 03, 2024

CONTENTS

This is a small sample book to give you a feel for how book content is structured. It shows off a few of the major file types, as well as some sample content. It does not go in-depth into any particular topic - check out the Jupyter Book documentation for more information.

Check out the content pages bundled with this sample book to see more.

- Setup
  - *Homework 01*
  - *Exploring data*
  - *Fixing WSL*
  - *August 28*
- Measures of Closeness
  - *Bayes' Theorem*
  - *Quick Jupyter Intro and Python Loops*
  - *Linear Regression*
  - *Linear Least Squares*
  - *Correlation Coefficient*
- Linear Algebra in Python
  - *Linear Algebra and Python Lists*
  - *Introduction to Matrices in NumPy*
  - *Matrix indices warmup*
  - *Gaussian Elimination*
  - *Log-Log Regression*
  - *Strassen's Algorithm (Optional Extension)*
  - *Gauss Presentation*
  - *Matrix Inversion*
  - *Running Time Analysis*
- Next Steps
  - *Reading for Research*
  - *Hobo Data, Student Version*
  - *ML Book Club*

Part I

Setup

CHAPTER

ONE

HOMEWORK 01

Goals for the day

- Set up python dev environment on school or personal laptop
- Raspberry Pi Project
- Money List

1.1 Set up dev environment

You need a decent amount of software to do real machine learning and your life will be much easier if we all have the same dev environment, to a certain degree. Here are steps we can take to try to get on the same page (these instructions are not perfect and if you get stuck, try to figure it out or we'll fix it in class)

1. Install a real text editor (VSCode, Atom, Sublime, `vim`, `emacs`, …). If you don't know or care, do VSCode.
2. On a Windows machine, install `WSL` (Windows Subsystem for Linux) following these instructions. Be sure to select Ubuntu 22 for your distribution (unless you are sure you will never need to ask me for help).
3. Inside linux (WSL or Mac/Linux bash shell), install `asdf` (a version manager for executables) according to these instructions.
4. Set up python
    1. Restart your shell (in WSL or Mac/Linux bash shell)
    2. `asdf plugin add python`
    3. `asdf install python 3.11.9`
    4. `asdf global python 3.11.9` or `asdf local python 3.11.9`
    5. Make a directory for your ML projects `mkdir ~/ml` then open the directory `cd ~/ml` and set up a virtual environment (next)
5. Set up a virtual environment (a directory with specific version of python and libraries to use in your projects.)
    1. In your ml directory (`cd ~/ml`)
    2. Ensure you're running python 3.11.9: `python --version` (if not redo step 4.4 above)
    3. Create a virtual environment `python -m venv env` (This creates a directory named `env` with a local install of python)
    4. Activate the environment `source env/bin/activate`
    5. Copy this file to your folder `~/ml` as `requirements.txt`.
    6. Install the libraries `pip install -r requirements.txt`

7. Everything should install with no errors (warnings are OK.)
8. See if jupyter is working: `jupyter-lab` (should open a browser)

Action Item: (As much as possible…) Set up a functioning python 3.11.9 dev environment with Jupyter and a text editor. Use the provided `requirements.txt` file to create a virtual environment with the packages needed for class.

1.1.1 Optional (last resort)

If the WSL route is too complicated for now or just not working, Anaconda will get you up and running. The reason this is not recommended is that Anaconda installs a LOT of software and does a LOT of setup behind the scenes. And when you want to change something, or bypass Anaconda for a reason, it can be tricky to disable it. That said, it is very popular and good at what it does.

Install Anaconda for Windows

1.2 Raspberry Pi Project

I have about 25 Raspberry Pi 3s and would like you to do *something* with one. Go home, read about them, find a fun project or something to install or make and come tell me what it is. If I approve, then I'll give you a pi for a few days to make it work. In addition to the Pi we have sensors and peripherals and things so if you want to do something fancy we might be able to find the stuff for it. Just note: this is not a *big* project. Just a quick thing to get something up and running.

You will need a monitor, HDMI cable, keyboard and wired mouse to complete the initial Pi setup. After that it can run "headless" on your home network. If you don't have these things at home, we can probably get you to do the initial setup here at school.

Action Item: come to class with some ideas for projects. Be ready to discuss with friends and/or the teacher.

1.3 The Money List

Your money list is a list of ideas that will make you money! You'll add to it all year. It starts off simple: write down things that annoy you and think of ways to fix them. The annoying things can be literally anything (that's the point of brainstorming) and the "ways to fix them" can be outlandish. But, the goal is to occasionally stumble on an idea that is generally useful to lots of people and whose solution is something you can work on. If your idea is good enough and your solution works, then sell it. Make money!

Action Item: Start the list somewhere semi-permanent. In a notebook, on your computer, Google Drive, or your phone. (the best might be a cloud file shared to multiple devices). Try to find 3 ideas by next class. Don't be too picky

CHAPTER

TWO

EXPLORING DATA

The file *weather-daylight.csv* contains observational weather data for Leesburg, VA. We want to analyze the hypothesis "the fall of 2023 had more cloudy and rainy weekends than normal." As a class, let's look at the data and talk about our ideas for processing it. Open this file in Microsoft Excel, or something like it.

What questions do you have? What pre-processing is relevant? What types of calculations would support or refute the claim? Do you know how to make those calculations in Python or another language or tool?

2.1 Look at It! LOOK at IT!

(Bonus points if you know the Seinfeld reference). The first thing to do is just look at the data set. What do you see? Here are some questions you should ask?

- How many rows, how many columns?
- Is the data rectangular (are all rows the same length?)
- What types of data? (Numerical, categorical?)
- What domain of data (numerical: min and max, precision, mean, variance; categorical: number of categories)
- Any missing data?
- Is the file clean (read/write errors? paragraphs of text before or after? anything else weird?)
- Find meaning: What do the columns are rows mean? Are there headers? Are they defined?
- What is the source? Is this data reliable?
- ***Weather columns***

2.2 Analyse it

- What question are you asking?
- What does the data say about the question?
- Repeat the last 2 steps as needed!

2.3 Jupyter Notebook

- Follow this link to a jupyter notebook
- I believe you can save and open notebooks from this interface, as long as you are using the same Chrome profile and history (it uses local storage to save state).

2.4 Homework

- Open the jupyter notebook above from class on mybinder.org (or you can run it locally as `jupyter-lab` in your `ml` folder on WSL. Make sure to copy the notebook to your `ml` folder first – download it).
- Devise a different way to analyse the question about the weather in Leesburg and try to analyse it using pandas. What conclusion can you draw?
- Make some interesting plots from this dataset. You will need to read up on plotting in dataframes using pandas and possibly some things about pyplot.
- Consider the *London Weather dataset*. Investigate the question "Has the weather in London gotten worse in the last 50 years?" Analyse the data and make a claim that you can support. Source for dataset: https://www.kaggle.com/datasets/emmanuelfwerr/london-weather-data, which retrieved the data from https://www.ecad.eu/dailydata/index.php.
- Be prepared to discuss your findings and present to the class if asked to! (Your research doesn't need to be profound but I do need to see you're learning how to use `pandas`).
- To be safe, you should download any notebooks you create on `mybinder.org` because I don't know how reliable its storage system is.
- WSL Problems: *Fix posted!*
- **In general** I'm always happy to answer homework questions. *Remind* is the best way to reach me in the evening so don't shy away from asking. If I can't help, I'll say so. Otherwise I'll try my best!

CHAPTER

THREE

FIXING WSL

If you tried my first instructions and ran into some kind of python packaging error (looks like mysql is a culprit), here's a patch

First, you should switch to your `ml` folder in WSL and delete the old environment.

```
$ cd ml # or whatever
$ deactivate # may not do anything if you didn't activate the env
$ rm -rf env # this deletes the old environment
$ asdf local python 3.11.9 # make sure you're on the right python
$ python -m venv env # make the virtual environment
$ source env/bin/activate # enable the new environment
$ pip install jupyterlab numpy scikit-learn matplotlib pandas
$ jupyter-lab # make sure it works
```

When I tried this on one student machine it worked. Jupyterlab runs in a browser and you may have to click on a link that appears in your console to get it to run. (the link will contain `localhost` or `127.0.0.1`)

The problem seems to be an incompatibility with the packages I defined in `requirements.txt`. The Python Package Manager (`pip`) tries to resolve dependencies in a consistent manner but does not always succed (i.e. it usually fails on big projects.) I was hopeful the environment I carefully curated would work on windows AET machines but it doesn't. This short `pip install` command installs what we need right now and will be fine until I can properly debug with my own machine.

CHAPTER

FOUR

AUGUST 28

Today in class

- 5 minutes for computer bugfixing
- Present findings for weather data
- Upload HW to server
  - wsl: `cd` into the `ml` directory where your jupyter notebook is located (download it if it's online)
    - wifi: AET-3142
    - use wsl `ftp` and `ftp username@ubuntu`
    - your username is `llllllll`, (first 7 letters of last name, then first initial, according to phoenix)
    - passwd is your student ID
    - to save the file: `put filename` (tab completion should probably work here)
    - to check it's there: `ls`
    - to disconnect `ctrl-D` or (I think) `exit`
- Discuss Bayes' Theorem, including Jupyter notebook and python concepts
  - lists, arrays and comprehensions in python
  - for loops in python
  - add, delete cells in Jupyter
  - Markdown cells and syntax\

Homework is complete the Bayes' Theorem Jupyter notebook.

Part II

Measures of Closeness

CHAPTER

FIVE

BAYES' THEOREM

Bayes' Theorem gives us a way to *invert* conditional probabilities. The formula comes from the definition of conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

this implies the following

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

Solving for $P(A|B)$ we get

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Though this is the final form, in practice you will need to compute $P(B)$ using the following

$$P(B) = P(B|A)P(A) + P(B|\overline{A})P(\overline{A})$$

which says the probability of B is the sum of the probability of B given A and B given not A. (A is either true or false so these are the only two options)

5.1 Exercise 1: Plot a Venn Diagram

Using matplotlib, draw a simple Venn diagram representing two sets A, B with a non-null intersection.

```
## Code here. Add cells as needed
```

5.2 Exercise 2: Compute Bayes' Probabilities

We want to replicate the computation carried out in class. If a doctor performs a test that has a given accuracy, for a disease with a given incidence rate, determine the probability that a randomly selected person with a positive test result has the disease. You are given *accuracy* and *incidence* as input, both in the range $(0, 1]$

```
def get_bayes_probability(acc, inc):
    ## code here
    return bayes
```

Check some results below. The first one comes from class

```
get_bayes_probability(0.97,0.001)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 get_bayes_probability(0.97,0.001)

Cell In[2], line 3, in get_bayes_probability(acc, inc)
      1 def get_bayes_probability(acc, inc):
      2     ## code here
----> 3     return bayes

NameError: name 'bayes' is not defined
```

```
get_bayes_probability(0.97,0.01)
```

```
get_bayes_probability(0.97,0.1)
```

```
get_bayes_probability(0.99,0.001)
```

```
get_bayes_probability(0.50,0.001)
```

5.3 Exercise 3: Plot

You will create two plots in the section. For a fixed incidence rate, plot the bayes probability as the accuracy of the test ranges from 0 to 100%.

Then, for a fixed accuracy, plot the bayes probability as the incidence rate increases.

Note, to avoid 1/0 errors you'll probably want to *not* go all the way to 0 or 1.

State a conclusion about the results. What's the correlation? What do you observe? What do you think about accuracy measures for tests now?

Hint: create two arrays `X, Y` (python lists) of the same length containing the X values in one array and the Y values in another. List comprehensions are the best way to do this in python, though a for loop is fine too (append to an initially empty list)

then use `plt.plot(X,Y)`

```
from matplotlib import pyplot as plt
```

```
# code here. add as needed
```

5.4 Exercise 4: Beautify plots

Now go back and beautify your plots. Add a title and a legend. Some axis labels. Maybe read about matplotlib styles and change up the colors. Try a different type of plot. Just experiment some. Results below.

CHAPTER

SIX

QUICK JUPYTER INTRO AND PYTHON LOOPS

Jupyter notebook is arranged into cells. Cells can contain Input (computations/code), Output and Markdown (text). A cell can be *selected* or *active*. To *select* a cell, click to the left of it and it should highlight with a colored border. To activate a cell, click inside of it so the cursor is visible. To run a code cell, or render a text cell, type `shift-return`.

When a cell is selected you can do cell operations by typing single letters such as

- `A` add a cell above this one
- `B` add a cell below this one
- `X` cut the cell to the clipboard
- `C` copy the cell to the clipboard
- `V` paste the cell on the clipboard
- `D` delete the cell
- `M` convert a cell to markdown style
- `Y` convert a markdown cell back to Input style

You can also drag a selected cell around the notebook with the mouse. You should also get familiar with the menu bar and toolbars. There are several useful operations hidden there. For example, *rerun all cells* is handy as is *Clear Outputs of all cells*

6.1 Python Loops

I'll show some basic examples of how to do things in python. You may want to run these in Jupyter to verify.

- Print the numbers 0-99 on separate lines

```
for i in range(100):
        print(i)
```

- Print the numbers 10-99 on separate lines

```
for i in range(10,100):
        print(i)
```

- Print the *even* numbers 10-19 on separate lines

```
for i in range(10,20,2):
        print(i)
```

- Print all characters in "hello computer", all on one line

```
for c in "the word":
        print(c, end="")
```

- Compute the largest Fibonacci number less than 1000 (note the use of parallel assignment)

```
a,b = 1,0
while a<1000:
                a,b = a+b,a
```

6.2 Python Lists

Python's regular list data type is not a true array, such as you would find in C or Java. An array is, properly, fixed length and single type. Python uses *lists* which are dynamic (auto-resizing) and can contain *any* type. For example

```
l1 = [1,2,3]
l2 = ["bird",2,-10.3,"cow"]
l3 = [l1,l2,3]
print(l1)
print(l2[2])
print(l3[1][1])
```

```
[1, 2, 3]
-10.3
2
```

You see lists can even contain lists. Here are some familiar tropes in Python

```
l = [0]*10 # a list of 10 zeros
for i in range(len(l)):
        l[i] = i  # replace the zeros with the index
for i in range(10,20):
        l.append(i) # add more elements to l
for i in range(20,30):
        l = l + [i] # another way to add
print(l)
```

This prints the integers from 0 to 29 inclusive.

Lists are very flexible and there are many operations that are easy to do such as sort, find, replace, merge, delete. If you want to perform a list operation, look it up in online and see if it is already a built-in operation

6.3 Comprehensions

Comprehensions are beautiful and lovely ways to build lists quickly. I'll just give a couple examples here

This code

```
l = []
for i in range(10):
        l.append(i**2)
```

can be replaced with this comprehension

```
l = [i**2 for i in range(10)]
```

It is quite lovely, isn't it. It reads like a math set definition

$$L = \{i^2 \mid 0 \leq i < 10\}$$

And make great tools for plotting, say

```
from math import sin
from matplotlib import pyplot as plt
X = [i/100 for i in range(628)]
Y = [sin(i/100) for i in range(628)]
plt.plot(X,Y);
```

will plot one period of a sine curve.

In practice, numerical algorithms will use `numpy arrays` instead of python lists because they are much much faster and work like traditional C arrays. But lists still come in extremely handy when you need to collect data and speed is not a priority.

CHAPTER

SEVEN

LINEAR REGRESSION

Given n points $(x_1, y_1) \ldots (x_n, y_n)$ and an assumed relation $y = f(x) + \epsilon, \epsilon \sim N(\mu, \sigma)$ we want to find a model $\tilde{y_i} = ax_i + b$ such that the residual squared error

$$\text{RSS}(a, b) = \sum_{i=1}^{n} \left(\tilde{y_i} - y_i\right)^2$$

is minimized.

RSS is a function of the line parameters a and b. To minimize it we set both partial derivatives to zero. (This could technically find a maximum – but it's reasonably clear this function has no maximum value because the error can always be increased.)

Take partial derivatives

$$\begin{aligned}
\frac{\partial RSS}{\partial a} &= 2 \sum \left(\tilde{y}_c - y_i\right) \frac{\partial}{\partial a} \left(\tilde{y}_i - y_i\right) \\
&= 2 \sum \left(\tilde{y}_i - y_i\right) (x_i) \\
\frac{\partial RSS}{\partial b} &= 2 \sum \left(\tilde{y}_i - y_i\right) \frac{\partial}{\partial b} \left(\tilde{y}_i - y_i\right) \\
&= 2 \sum \left(\tilde{y}_i - y_i\right) (1)
\end{aligned}$$

Since

$$\frac{\partial}{\partial a} \tilde{y}_i = \frac{\partial}{\partial a} \left(a x_i + b\right) = x_i$$

$$\frac{\partial}{\partial b} \tilde{y}_c = \frac{\partial}{\partial b} \left(a x_c + b\right) = 1$$

And solve

$$\begin{cases} \dfrac{\partial RSS}{\partial a} = 0 \\ \dfrac{\partial RSS}{\partial b} = 0 \end{cases} \Rightarrow \begin{cases} \sum (\tilde{y}_i - y_i)\, x_i = 0 \\ \sum \left(\tilde{y}_i - y_i\right) = 0 \end{cases}$$

Since $\tilde{y}_i = a x_i + b$ \$$\sum \left(a x_i + b - y_i\right) x_i = 0 \Rightarrow a \sum x_i^2 + b \sum x_i = \sum y_i x_j and \sum \left(a x_i + b - y_i\right) = 0 \Rightarrow a \sum x_i + b \sum 1 = \sum y_i$\$

by Cramer's rule

$$a = \begin{vmatrix} \sum x_i y_i & \sum x_i \\ \sum y_i & n \end{vmatrix} \Big/ \begin{vmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{vmatrix}$$

$$b = \begin{vmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i & \sum y_i \end{vmatrix} \Big/ \begin{vmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{vmatrix}$$

since $\sum_{i=1}^{n} 1 = n$

Taking determinants, $a = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$

$$b = \frac{\sum y_i \sum x_i^2 - \sum x_i \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

7.1 Interpretation as a ratio of variances

Students of statistics may appreciate the following manipulations

Definition of covariance $E(xy) - E(x)E(y) = \text{Cov}(x, y)$

Definition of variance

$$\text{Var}(x) = E\left[(x - \mu)^2\right]$$

Lemma

$$\begin{aligned}
\text{Var}(x) &= E\left[(x - \mu)^2\right] \\
&= E\left(x^2\right) - 2\mu E[x] + E[\mu]^2 \\
&= E\left[x^2\right] - 2E[x]^2 + \mu^2 \\
&= E\left[x^2\right] - E[x]^2
\end{aligned}$$

Manipulating the denominator of the equation for a on the previous page, $n\sum x_i^2 - \left(\sum x_i\right)^2 = n^2\left(\frac{1}{n}\sum x_i^2 - \left(\frac{\sum x_i}{n}\right)^2\right)$

$$\begin{aligned}
&= n^2\left(E\left[x^2\right] - E[x]^2\right) \\
&= n^2\,\text{Var}(x)
\end{aligned}$$

And the numerator $n\sum x_i y_i - \sum x_i \sum y_i$

$$\begin{aligned}
&= n^2\left(\frac{1}{n}\sum x_i y_i - \frac{1}{n}\sum x_i \cdot \frac{1}{n}\sum y_i\right) \\
&= n^2\left(E\left[xy\right] - E[x]E[y]\right) \\
&= n^2\left(E[xy] - \mu_y\mu_y\right) \\
&= n^2\,\text{Cov}(x, y)
\end{aligned}$$

so

$$a = \frac{E[xy] - \mu_x\mu_y}{E[x] - \mu_x^2} = \frac{\text{Cov}(x, y)}{\text{Vav}(x)}$$

CHAPTER

EIGHT

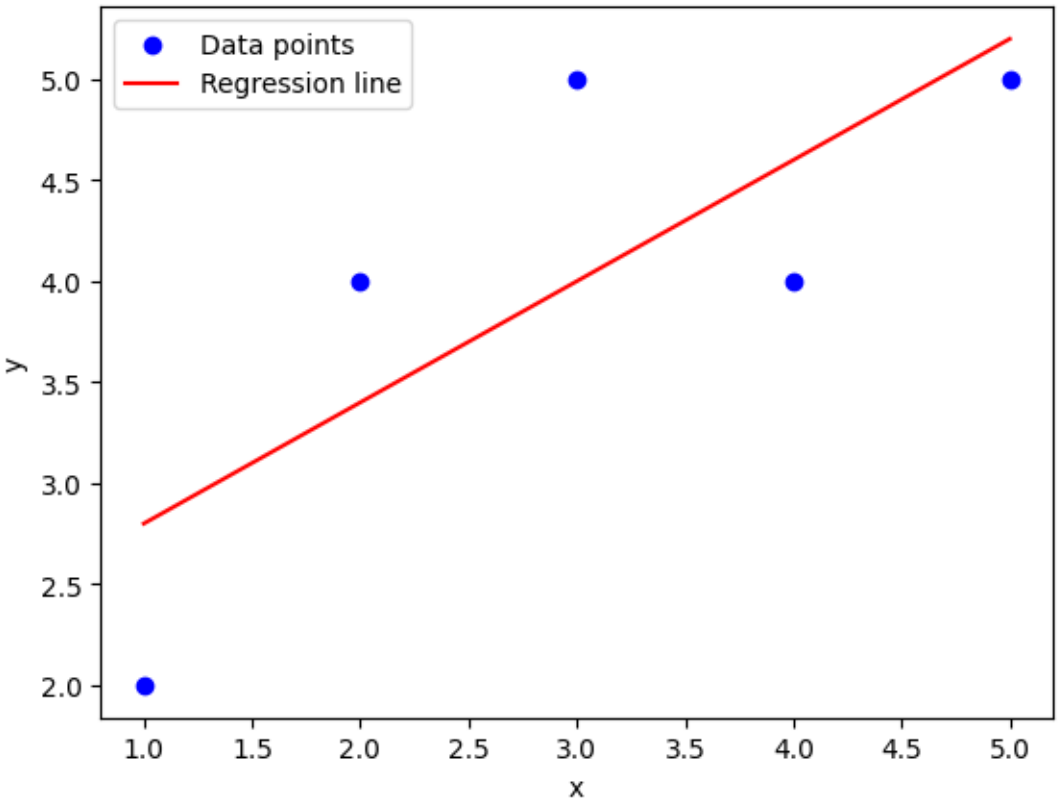
LINEAR LEAST SQUARES

In class we derived the formula for linear least squares of one variable. In this notebook you will learn a bit of the numerical library numpy, use numpy to compute linear regression, and then compute it yourself using formulas from class

Begin by running the cell below. Then go back and carefully read through all the code. There is a lot of new stuff here. Note how to create numpy arrays/matrices and how to compute a linear least squares regression.

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

# Step 1: Prepare your data
# x: Independent variable (input)
# y: Dependent variable (output)
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

# Step 2: Perform linear regression using the least squares method

# Add a column of ones to the input data for the intercept (bias term)
X = np.vstack([x, np.ones(len(x))]).T

# Calculate the slope (m) and intercept (b)
a, b = np.linalg.lstsq(X, y, rcond=None)[0]

print(f"Slope (a): {a:.4f}")
print(f"Intercept (b): {b:.4f}")

# Step 3: Predict y values using the regression line
y_pred = a * x + b

# Optional: Plot the data and the regression line
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Slope (a): 0.6000
Intercept (b): 2.2000
```



8.1 An aside about numpy matrices

What happened to x? Here's the original x, which is an array

```
x
```

```
array([1, 2, 3, 4, 5])
```

We add a row of 1s after it and take the transpose to get the input matrix X

```
X
```

```
array([[1., 1.],
       [2., 1.],
       [3., 1.],
       [4., 1.],
       [5., 1.]])
```

Breaking this down into pieces, first let's make a python list that contains x and an array of ones

```
[x,np.ones(len(x))]
```

```
[array([1, 2, 3, 4, 5]), array([1., 1., 1., 1., 1.])]
```

Now let's use numpy to make a vertical stack. The first element in the list becomes the first row

```
np.vstack([x, np.ones(len(x))])
```

```
array([[1., 2., 3., 4., 5.],
       [1., 1., 1., 1., 1.]])
```

And now take the transpose

```
np.vstack([x, np.ones(len(x))]).T
```

```
array([[1., 1.],
       [2., 1.],
       [3., 1.],
       [4., 1.],
       [5., 1.]])
```

8.1.1 Practice with matrices

Make a numpy matrix that is a row of 5 zeros followed by a row of 5 ones, then 5 zeros, then 5 ones again. Use built in functions and `vstack` (don't just type a bunch of 0 and 1 – can you guess the name of a function that makes an array of zeros?)

Now make a similar matrix that is a row of all 1s followed by all 2s in the second row, then 3s then 4s. Again use built in function `np.ones`. Name this matrix `M`. Hint: $[2,2,2,2,2] = 2 \cdot [1,1,1,1,1]$

compute M times M transpose and M transpose times M (MM^T and M^TM). In `numpy` AB can be computed with `A @ B` for matrices `A` and `B`

A matrix M is *symmetric* if $M = M^T$. This also implies $M_{ij} = M_{ji}$ for all indices (i, j). Write a python function `is_symmetric(M)` which returns `true` if and only if M is symmetric

Test your function. Make a 5 by 5 random integer matrix (see `np.random.randint`) called M. It is a fact that MM^T is always symmetric. Check that your function return `true` for MM^T and `false` for M. Repeat this trial 100 times and verify all 100 are correct.

8.2 Linear Least Squares Regression

You can create a vector of normally distributed samples with mean μ and standard deviation σ by using the numpy function `np.random.normal(mu, sigma, n)`. Try creating a vector with 10 random samples, with a mean of 100 and a standard deviation of 5.

Now create some data for linear regression. Make a vector x of ints over the range $[0, 9]$ and let y be a linear function of x, $y = 3x + 2 + \epsilon(x)$ where $\epsilon(x)$ is a random Gaussian noise function $\epsilon(x) \sim N(0, 1)$. Make a scatter plot of y vs. x and label it

Compute the correct linear regression coefficients using numpy as above. Check they are resonable.

Now compute the regression coefficients using the formulas from class. Begin by defining some very helpful variables: `Sx, Sy` will be $\sum_i x_i$ and $\sum_i y_i$ respectively. Next `Sxx` and `Syy` are the sum of squares: $\sum_i {x_i}^2$ and $\sum_i {y_i}^2$. Finally the inner product `Sxy` = $\sum_i x_i y_i$. The quickest way to do this involves using comprehensions and the `sum` function, but you can use loops for now if you need to.

```
# Print your results
Sx, Sy, Sxx, Syy, Sxy
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 2
      1 # Print your results
----> 2 Sx, Sy, Sxx, Syy, Sxy

NameError: name 'Sx' is not defined
```

Finally determine a, b as in class. Display the absolute errors between your calculations and the ones numpy returned. (They should be close to machine precision, which is 10^{-15} give or take.

8.3 Least squares function

Did you know python can return two values? Here's an example.

```
def two_numbers():
    a = 1
    b = 10
    return a,b
```

```
A, B = two_numbers()
print(A,B)
```

Write a function `linear_least_squares(x,y)` which takes input vectors x,y and returns a,b as above. (

```
def linear_least_squares(x,y):
```

8.4 Application

Now, using $a = 5, b = -15$, run linear least squares 100 times on 100 vector pairs (x, y), where each of the 100 x are the same but the $y = ax + b + \epsilon$ each have different amounts of Gaussian noise. Plot the resulting best fit lines all on the same graph.

- Use `np.arange` to make your input vector x cover the domain $[-5, 5]$ with a step-size of 0.01
- Create arrays to store all the computed a and b values (you'll use this later)
- If you call `plt.plot()` in a loop, it will keep adding to the same plot
- Give your plot a title!

Determine the average of the as and bs returned above. Compare these to the true a, b. Explain your result.(There is an `np.mean` function)

Make two histogram plots of the calculated a and b values `plt.hist` works nicely and adding a semicolon suppressed the nasty text output (you'll see)

CHAPTER

NINE

CORRELATION COEFFICIENT

We assume, as usual, a ground truth model $y = f(x) + \epsilon$ where f is usually unknown, a (possibly random) sample of points $(x_1, y_1), \cdots, (x_n, y_n)$ and a linear model $\tilde{y} = ax + b$. In this setting we usually need to know *how good* the linear model is – how well does if capture the ground truth $f(x)$?

One obvious measure is the sum of squared errors, which we minimized last class to derive the linear regression equations.

$$SSE = \sum_{i=1}^{n} (\tilde{y}_i - y_i)^2$$

While this literally captures the error in the model on each point, it is hard to interpret, it scales with the number of points, and is in different units from the given data. We can normalize it to the mean sum of squared errors:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\tilde{y}_i - y_i)^2$$

which at least doesn't scale with the number of points but is in different units. Thus by taking a radical

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\tilde{y}_i - y_i)^2}$$

we get the root mean sum of squared errors. This at least scales with the magnitude of the y values, so you can interpret it somewhat. It is also similar to a standard deviation, which is familiar to many people. (Note some texts would divide by $n - 2$ instead of n to create a truly unbiased estimator for the standard deviation, but this simpler version aggrees with other data science presentations, including kaggle.)

9.1 Pearson's Correlation Coefficient

While variants of SSE have their place, one cannot escape the use of r, the Pearson's correlation coefficient. Students learn in algebra classes that a linear regression coefficient $r = 1$ is a perfect positive correlation and $r = -1$ is a perfect negative correlation and $r = 0.5$ is a weak correlation, for example. We will take a more precise approach.

One formular for r is

$$r^2 = \frac{SS_{reg}}{SS_{tot}} = \frac{\sum_i (\tilde{y}_i - \overline{y})^2}{\sum_i (y_i - \overline{y})^2}$$

Let's unpack this. SS_{reg} is the sum of squared-error due to regression and SS_{tot} is the sum of squared-error total (due to the original data). Here $\overline{y} = \frac{1}{n} \sum_i y_i$ is the mean of the observed y_i values. SS_{tot}, then, is the variance of the observed y_i values – it is the sum of the squared deviations of the observations from their mean.

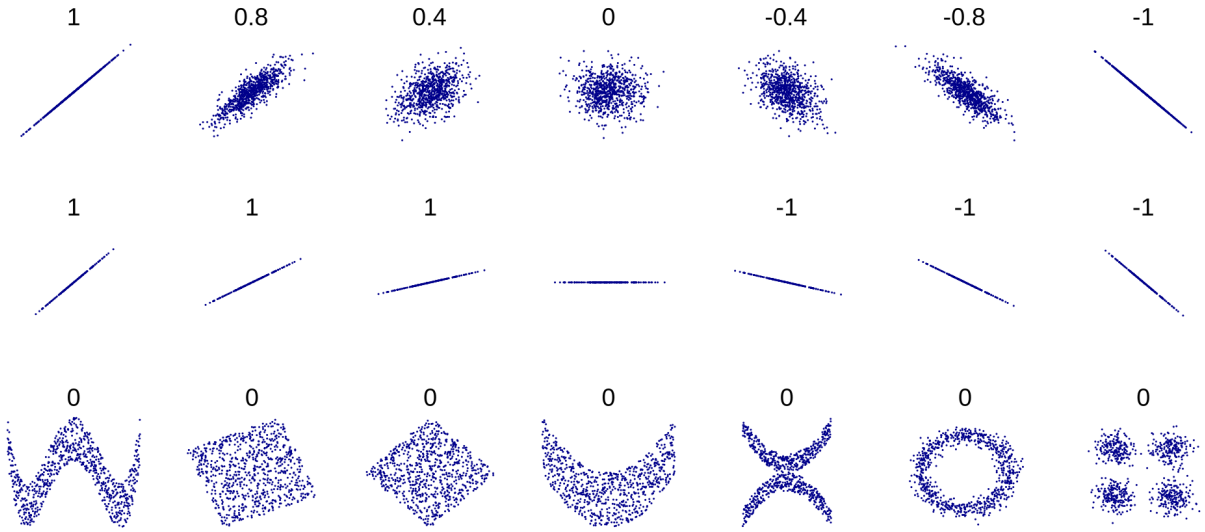
SS_{reg}, on the other hand, is the variance of the predicted $\tilde{y}_i$ values, relative to the same observed mean.

The ratio of the two is the ratio of the "explained variance" to the "total variance." There is always variance in the original dataset. If our linear model very closely fits the data, then it will explain most of that original variance. That would correspond to a high r^2 value. On the other hand a low r^2 indicates that there is variance in the data that is not capture by the linear model. Something else is happening to create this data shape.

It can be helpful to think of r^2 as the percent of "explained variance". You'll notice this formula is for r^2, not r. Obviously both are < 1 but they are not identical. We may see more details of the various ways to interpret r vs r^2 but honestly for most cases this explanation is quite good enough and better than what most people understand!

9.2 r=0

You may have been taught that $r = 0$ implies no correlation between x and y pairs. This is often indicated in math books with an amorphous cloud of points, wandering lonely across the page, enigmatic and unknowable. Actually a number of highly correlated datasets can claim to possess $r = 0$ values as this helpful chart shows[1]



To be correct, $r = 0$ implies no **linear** correlation between x and y. If it so happens that every predicted $\tilde{y}_i$ value is identical to the mean $\overline{y}$, then $r^2 = 0$. Datasets with perfect vertical symmetry can have this property.

[1] By DenisBoigelot, original uploader was Imagecreator - Own work, original uploader was Imagecreator, CC0, https://commons.wikimedia.org/w/index.php?curid=15165296

Part III

Linear Algebra in Python

CHAPTER

TEN

LINEAR ALGEBRA AND PYTHON LISTS

This notebook will review some important concepts in linear algebra while helping you practice working with lists and nested lists in Python. You will provide code cells as needed to complete the sections below. Testing code is provided for you to help check that your methods perform as expected.

```
import numpy as np
from matplotlib import pyplot as plt
import math
```

10.1 Dot product of two vectors

$\vec{a} \cdot \vec{b} = \sum_i a_i b_i$ is the dot product (or inner product) between $\vec{a}$ and $\vec{b}$. Write a python function `dot_product(a,b)` that returns a dot product of two input python lists (do not use numpy arrays yet). If the input types do not match, raise a ValueError.

```
## Your code goes here. Insert cells as needed.
```

The following cells will check your dot_product method for correctness and for catching errors.

```
assert(dot_product([1,3],[2,6])==20)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 assert(dot_product([1,3],[2,6])==20)

NameError: name 'dot_product' is not defined
```

```
try:
    dot_product([1,2,3],[4,5])
    print("Shouldn't get here")
except ValueError:
    print("Size mismatched caught")
```

10.2 Cross Product of two vectors

Given two real 3-vectors **a** and **b**:

$$\mathbf{A} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

The cross product of **a** and **b** is a new vector **C** defined as:

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

This resulting vector **C** is orthogonal (perpendicular) to both **a** and **b**, and its magnitude is equal to the area of the parallelogram formed by **a** and **b**.

The direction of **C** is determined by the right-hand rule.

The cross product of **a** and **b** can be expressed as the determinant of a 3x3 matrix:

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

Where **i**, **j**, and **k** are the unit vectors along the x, y, and z axes, respectively.

To compute the determinant, expand along the first row (using cofactor expansion):

$$\mathbf{a} \times \mathbf{b} = \mathbf{i} \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} - \mathbf{j} \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} + \mathbf{k} \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}$$

Each of these 2x2 determinants is computed as follows:

- For the **i** component: $\mathbf{i} \cdot (a_2 b_3 - a_3 b_2)$
- For the **j** component (note the negative sign): $-\mathbf{j} \cdot (a_1 b_3 - a_3 b_1)$
- For the **k** component: $\mathbf{k} \cdot (a_1 b_2 - a_2 b_1)$

Thus, the cross product is:

$$\mathbf{a} \times \mathbf{b} = (a_2 b_3 - a_3 b_2)\,\mathbf{i} - (a_1 b_3 - a_3 b_1)\,\mathbf{j} + (a_1 b_2 - a_2 b_1)\,\mathbf{k}$$

Or, written in vector form:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

This is the cross product of the vectors **a** and **b**, derived from the determinant of a 3x3 matrix.

Write a python method which returns the cross product of 2 3D vectors, and throws an error in case of invalid input

```
## Your code goes here. Insert cells as needed.
```

These cells will check your method

```
assert(cross_product([1,3,2],[5,7,2])==[-8,8,-8])
```

```
try:
    cross_product([1,2,3],[4,5])
    print("Shouldn't get here")
except ValueError:
    print("Size mismatched caught")
```

10.3 Vector magnitude

The magnitude, or length, of a vector is defined as

$||\vec{x}|| = \sqrt{\vec{x} \cdot \vec{x}} = \sqrt{x_0^2 + x_1^2 + \cdots + x_{n-1}^2}$. Notice the exponent on each element is 2 and the radical exponent is $\frac{1}{2}$. This is an L_2 norm. If we compute the value $\sqrt[3]{x_0^3 + x_1^3 + \cdots + x_{n-1}^3}$, this is an L_3 norm. The sum of absolute values $|x_1| + |x_2| + \cdots + |x_n|$ is called the L_1 norm. All of these are p-norms, with the $p = 2$ being the most familiar, and the $p = 1$ norm being useful in several settings.

Write a python method `norm(a)` which returns the 2-norm (length) of a vector, by calling `dot_product`. Then write `p_norm(a,p)` which evaluates the p-norm for any integer $p \geq 1$ and throws an error if needed.

```
## Your code goes here. Insert cells as needed.
```

This cell will check your method

```
assert(norm([1,-1,1,-1])==2)
assert(norm([0])==0)
assert(norm([30,40,50])==np.sqrt(5000))
```

```
## Your code goes here. Insert cells as needed.
```

These cells will check your `p_norm`

```
assert(p_norm([1,-1,0],1)==2)
assert(p_norm([1,-1,1,-1],2)==2)
assert(p_norm([2,-2,4,-4],3)==0)
```

```
try:
    p_norm([1,2,3],0.5)
    print("Shouldn't get here")
except ValueError:
    print("Invalid p caught")
```

10.4 Angle between vectors

A very important result from linear algebra used in machine learning relates the angle between two vectors. You will derive this yourself. Given **a** and **b**, you can form a triangle where the vectors share the same tail. The vector forming the third side is the vector **a-b**. Find the length of this third side in terms of **a** and **b**. Hint: use the law of cosines. Hint: dot product distributes like 'times'

Using your result write a function `angle_between` that returns the smallest angle between two vectors **a** and **b**. Make sure your result is in the range $[0, \pi]$

```
## Your code goes here. Insert cells as needed.
```

```
assert(abs(angle_between([1,1],[1,0])-math.pi/4)<1e-9)
assert(abs(angle_between([1,0],[1,0]))<1e-9)
assert(abs(angle_between([1,0],[0,1])-math.pi/2)<1e-9)
assert(abs(angle_between([1,2,3,4],[-4,-3,-2,-1])-2.300523983021863)<1e-9)
```

```
try:
    angle_between([1,2,3],[4,5])
    print("Shouldn't get here")
except ValueError:
    print("Size mismatch caught")
```

10.5 Matrix Operations

Matrix addition and subtraction are componentwise and require matrices of the same size: $m_{ij} = a_{ij} + b_{ij}$ or $m_{ij} = a_{ij} - b_{ij}$

Matrix multiplication involves one entire row and one entire column of each matrix to determine an entry in the product. It can be written as

$$m_{ij} = a_{ik} b^{kj}$$

Where we are using *Einstein notation*. Each repeated index (here k) is an index of summation. Thus

$$a_{ik} b^{kj} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

Matrix multiplication requires an $(m_1 \times n_1)$ matrix multiplied on the right by an $(m_2 \times n_2)$ matrix where $n_1 = m_2$. The result is an $(m_1 \times n_2)$ matrix.

Write three methods: `mat_add`, `mat_sub`, `mat_mul`. You should throw an error if dimensions do not match. You should store all matrices as 2D lists in python, i.e. lists of lists. The i, j element is referenced by `M[i][j]`, both starting at 0.

```
## Your code goes here. Insert cells as needed.
```

```
A=[[1]]
B=[[2]]
assert(mat_add(A,B)==[[3]])

A=[[-4,3],[1,-10]]
B=[[2,-5],[-9,1]]
assert(mat_add(A,B)==[[-2,-2], [-8,-9]])
assert(mat_add(B,A)==[[-2,-2],[-8,-9]])

A = [[1,2,3],[5,3,-1],[6,5,2]]
B = [[4,-2,2],[2,4,3],[6,2,2]]
assert(mat_add(A,B)==[[5,0,5],[7,7,2],[12,7,4]])
```

```
try:
    mat_add([[1,2,3],[4,5,6]],[[1,2],[3,4]])
    print("Shouldn't get here")
except ValueError:
    print("Size mismatch caught")
```

```
## Your code goes here. Insert cells as needed.
```

```
A=[[1]]
B=[[2]]
assert(mat_transpose(mat_transpose(A))==A)
assert(mat_transpose(mat_transpose(B))==B)

A=[[-4,3],[1,-10]]
B=[[2,-5],[-9,1]]
assert(mat_transpose(mat_transpose(A))==A)
assert(mat_transpose(mat_transpose(B))==B)

A = [[1,2,3],[5,3,-1],[6,5,2]]
B = [[4,-2,2],[2,4,3],[6,2,2]]
assert(mat_transpose(mat_transpose(A))==A)
assert(mat_transpose(mat_transpose(B))==B)
```

```
## Your code goes here. Insert cells as needed.
```

```
A=[[1]]
B=[[2]]
assert(mat_mul(A,B)==[[2]])

A=[[-4,3],[1,-10]]
B=[[2,-5],[-9,1]]
assert(mat_mul(A,B)==[[-35, 23], [92, -15]])
assert(mat_mul(B,A)==[[-13, 56], [37, -37]])

A = [[1,2,3],[5,3,-1],[6,5,2],[-10,2,3]]
B = [[4,-2,3,2],[2,4,3,-1],[6,3,2,2]]
C = [[2,-1,-3,4],[3,0,0,-2],[2,-3,7,-9]]

assert(mat_mul(A,B)==[[26, 15, 15, 6], [20, -1, 22, 5], [46, 14, 37, 11], [-18, 37, -
↪18, -16]])

assert(mat_transpose(mat_mul(A,B)) == mat_mul(mat_transpose(B),mat_transpose(A)))
assert(mat_add(mat_mul(A,B),mat_mul(A,C)) == mat_mul(A, mat_add(B,C)))
```

```
try:
    mat_mul(mat_transpose(C),A)
    print("Shouldn't get here")
except ValueError:
    print("Matrix incompatibility caught")
```

CHAPTER

ELEVEN

INTRODUCTION TO MATRICES IN NUMPY

Matrices are a fundamental data structure in scientific computing, and `numpy` provides powerful tools for creating and manipulating matrices. This guide will introduce the basics of working with matrices in `numpy` by covering the following topics:

1. **Creating Matrices**
2. **Accessing Elements**
3. **Basic Matrix Operations**
4. **Extracting Rows and Columns**
5. **Row and Column Operations**

You should read the guide carefully and run the code cells. Feel free to edit individual cells for experimentation.

(This guide was created by ChatGPT and edited by Dr. White)

11.1 1. Creating Matrices

In `numpy`, matrices are created as 2D arrays. Here are several ways to create them:

```
import numpy as np
```

11.1.1 a. Creating a Matrix from a List of Lists

```
# Create a 3x3 matrix of ints
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
matrix
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
# Create a 3x3 matrix with 64-bit float data types
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]], np.float64)
matrix
```

```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

11.1.2 b. Creating Matrices with Built-in Functions

```
# Create a 3x3 matrix of zeros
zeros_matrix = np.zeros((3, 3))

# Create a 3x3 matrix of ones
ones_matrix = np.ones((3, 3))

# Create a 3x3 identity matrix
identity_matrix = np.eye(3)

# Create a 3x3 matrix with random values
random_matrix = np.random.rand(3, 3)

zeros_matrix, ones_matrix, identity_matrix, random_matrix
```

```
(array([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]),
 array([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]]),
 array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]),
 array([[0.18434837, 0.58114038, 0.26121578],
        [0.67994115, 0.74174803, 0.8308242 ],
        [0.1445397 , 0.07585411, 0.81144162]]))
```

11.2 2. Accessing Elements

You can access elements, rows, or columns of a matrix using indexing and slicing:

11.2.1 a. Accessing Individual Elements

```
# Access element at row 1, column 2 (remember, indexing starts from 0)
element = matrix[1, 2]
element
```

```
np.float64(6.0)
```

11.2.2 b. Accessing Rows and Columns

```
# Access the second row (index 1)
second_row = matrix[1, :]

# Access the third column (index 2)
third_column = matrix[:, 2]

second_row, third_column
```

```
(array([4., 5., 6.]), array([3., 6., 9.]))
```

11.2.3 c. Modifying parts of a matrix

```
matrix_2 = matrix
matrix_2[1,:] = matrix_2[1,:] + 1
matrix, matrix_2
```

```
(array([[1., 2., 3.],
        [5., 6., 7.],
        [7., 8., 9.]]),
 array([[1., 2., 3.],
        [5., 6., 7.],
        [7., 8., 9.]]))
```

11.2.4 d. Matrix references

Notice the above matrix_2 copied a reference to matrix so any change to one changed both. To duplicate a matrix you need to create a new matrix. Notice the two are different in the following cell.

```
matrix_2 = np.array(matrix)
matrix_2[1,:] = matrix_2[1,:] / 100
matrix, matrix_2
```

```
(array([[1., 2., 3.],
        [5., 6., 7.],
        [7., 8., 9.]]),
 array([[1.  , 2.  , 3.  ],
        [0.05, 0.06, 0.07],
        [7.  , 8.  , 9.  ]]))
```

11.3 3. Basic Matrix Operations

NumPy allows you to perform various matrix operations easily.

11.3.1 a. Matrix Addition and Subtraction

```
# Add two matrices
result_add = matrix + identity_matrix

# Subtract two matrices
result_subtract = matrix - identity_matrix

result_add, result_subtract
```

```
(array([[ 2.,  2.,  3.],
        [ 5.,  7.,  7.],
        [ 7.,  8., 10.]]),
 array([[0., 2., 3.],
        [5., 5., 7.],
        [7., 8., 8.]]))
```

11.3.2 b. Matrix Multiplication

```
# Element-wise multiplication
element_wise_product = matrix * matrix

# Matrix multiplication, two ways
matrix_product = np.dot(matrix, identity_matrix)
mp = matrix @ identity_matrix

element_wise_product, matrix_product, mp
```

```
(array([[ 1.,  4.,  9.],
        [25., 36., 49.],
        [49., 64., 81.]]),
 array([[1., 2., 3.],
        [5., 6., 7.],
        [7., 8., 9.]]),
 array([[1., 2., 3.],
        [5., 6., 7.],
        [7., 8., 9.]]))
```

11.3.3 c. Transpose of a Matrix

```
# Transpose the matrix
transpose_matrix = matrix.T
transpose_matrix
```

```
array([[1., 5., 7.],
       [2., 6., 8.],
       [3., 7., 9.]])
```

11.3.4 d. Determinant and Inverse

```
# Calculate the determinant
determinant = np.linalg.det(matrix)

# Calculate the inverse (if the matrix is invertible)
inverse_matrix = np.linalg.inv(matrix) if determinant != 0 else None

determinant, inverse_matrix
```

```
(np.float64(-5.329070518200744e-15),
 array([[ 3.75299969e+14, -1.12589991e+15,  7.50599938e+14],
        [-7.50599938e+14,  2.25179981e+15, -1.50119988e+15],
        [ 3.75299969e+14, -1.12589991e+15,  7.50599938e+14]]))
```

11.4 4. Extracting Rows and Columns

11.4.1 a. Extracting Rows

```
# Extract the first row (index 0)
first_row = matrix[0, :]
first_row
```

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

11.4.2 b. Extracting Columns

```
# Extract the second column (index 1)
second_column = matrix[:, 1]

# Extract the last column (index -1)
last_column  = matrix[:,-1]

second_column, last_column
```

```
(array([2., 6., 8.]), array([3., 7., 9.]))
```

11.4.3 c. Extracting Submatrices

```
# Extract a 2x2 submatrix from the top-left corner
submatrix = matrix[0:2, 0:2]
submatrix
```

```
array([[1., 2.],
       [5., 6.]])
```

11.5 5. Row and Column Operations

11.5.1 a. Sum, Mean, and Other Operations on Rows/Columns

```
# Sum of each row
row_sum = np.sum(matrix, axis=1)

# Mean of each column
column_mean = np.mean(matrix, axis=0)

row_sum, column_mean
```

```
(array([ 6., 18., 24.]), array([4.33333333, 5.33333333, 6.33333333]))
```

11.5.2 b. Adding Rows/Columns

```
# Add a row (shape should match)
new_matrix_row = np.vstack([matrix, [10, 11, 12]])

# Add a column (shape should match)
new_matrix_column = np.hstack([matrix, [[10], [11], [12]]])

new_matrix_row, new_matrix_column
```

```
(array([[ 1.,  2.,  3.],
        [ 5.,  6.,  7.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]]),
 array([[ 1.,  2.,  3., 10.],
        [ 5.,  6.,  7., 11.],
        [ 7.,  8.,  9., 12.]]))
```

11.5.3 c. Deleting Rows/Columns

```
# Delete the first row (index 0)
matrix_without_first_row = np.delete(matrix, 0, axis=0)

# Delete the second column (index 1)
matrix_without_second_column = np.delete(matrix, 1, axis=1)

matrix_without_first_row, matrix_without_second_column
```

```
(array([[5., 6., 7.],
        [7., 8., 9.]]),
 array([[1., 3.],
        [5., 7.],
        [7., 9.]]))
```

11.5.4 d. Replacing Rows/Columns

```
# Replace the first row
matrix[0, :] = [10, 11, 12]

# Replace the third column
matrix[:, 2] = [13, 14, 15]

matrix
```

```
array([[10., 11., 13.],
       [ 5.,  6., 14.],
       [ 7.,  8., 15.]])
```

CHAPTER

TWELVE

MATRIX INDICES WARMUP

12.1 First, About python `range(a,b)`

Run the following and make sure you understand the output

```
a = range(10)
b = range(2,10)
c = range(2,10,3)
d = range(10,2,-1)
e = range(10,2,-3)
print(list(a))
print(list(b))
print(list(c))
print(list(d))
print(list(e))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7, 8, 9]
[2, 5, 8]
[10, 9, 8, 7, 6, 5, 4, 3]
[10, 7, 4]
```

12.2 Now for the matrix challenge

```
import numpy as np
```

Write nested loops to create each of the following 5x5 matrices, starting from a zero matrix. Use 'c' and 'r' as the names of your index variables (this really helps your brain.) Use a counter, too! Your counter should *always* start at 1, and your code should always look like this, with no added lines

```
for c in ....
  for r in ......
    A[r][c] = ...
```

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 6 & 0 & 0 & 0 \\ 3 & 7 & 10 & 0 & 0 \\ 4 & 8 & 11 & 13 & 0 \\ 5 & 9 & 12 & 14 & 15 \end{bmatrix}$$

```
A = np.zeros((5,5), np.int32)
# Solution 1 Here
print(A)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 & 0 \\ 3 & 6 & 8 & 0 & 0 \\ 4 & 7 & 9 & 10 & 0 \end{bmatrix}$$

```
B = np.zeros((5,5), np.int32)
# Solution 2 Here
print(B)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

$$C = \begin{bmatrix} 15 & 14 & 12 & 9 & 5 \\ 0 & 13 & 11 & 8 & 4 \\ 0 & 0 & 10 & 7 & 3 \\ 0 & 0 & 0 & 6 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
C = np.zeros((5,5), np.int32)
# Solution 3 Here
print(C)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

$$D = \begin{bmatrix} 0 & 10 & 9 & 7 & 4 \\ 0 & 0 & 8 & 6 & 3 \\ 0 & 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
D = np.zeros((5,5), np.int32)
# Solution 4 Here
print(D)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

CHAPTER

THIRTEEN

GAUSSIAN ELIMINATION

Gaussian elimination is an algorithm for solving a linear system by matrix operations. You will write code that performs Gaussian elimination on a solvable system and returns the solution. First, we work through an instructive example.

$$\begin{aligned} x + 2y - z &= 1 \\ 2x - y + 3z &= 5 \\ 3x + y + 3z &= 10 \end{aligned}$$

Now, we'll solve this system using reduced row echelon form (RREF).

13.1 Step 1: Write the Augmented Matrix

The augmented matrix for this system is: $$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 2 & -1 & 3 & 5 \\ 3 & 1 & 3 & 10 \end{array}\right)$$

13.2 Step 2: Apply Row Operations to Achieve RREF

We'll perform row operations to transform the matrix into reduced row echelon form (RREF).

1. **Make the pivot in the first row (top-left corner) a 1.**
   This is already 1, so no changes are needed.
2. **Make the first column below the pivot zeros.**
   - Add -2 times the first row to the second row: $R_2 \to R_2 - 2R_1$

     $$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & -5 & 5 & 3 \\ 3 & 1 & 3 & 10 \end{array}\right)$$

   - Add -3 times the first row to the third row: $R_3 \to R_3 - 3R_1$

     $$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & -5 & 5 & 3 \\ 0 & -5 & 6 & 7 \end{array}\right)$$

3. **Make the pivot in the second row a 1.**
   - Divide the second row by -5: $R_2 \to \frac{1}{-5}R_2$ $$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & -1 & -\frac{3}{5} \\ 0 & -5 & 6 & 7 \end{array}\right)$$

4. **Make the second column below the pivot zeros.**
   - Add 5 times the second row to the third row: $R_3 \to R_3 + 5R_2$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & -1 & -\frac{3}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

5. **Make the third column above the pivot zeros.**
   - Add 1 times the third row to the second row: $R_2 \to R_2 + 1R_3$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

   - Add 1 times the third row to the first row: $R_1 \to R_1 + 1R_3$

$$\left(\begin{array}{ccc|c} 1 & 2 & 0 & 5 \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

6. **Make the second column above the pivot zeros.**
   - Add -2 times the second row to the first row: $R_1 \to R_1 - 2R_2$

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{9}{5} \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

13.3 Step 3: Extract the solution

The system has a unique solution, given by the final column.

$$x = -\frac{9}{5}, \quad y = -\frac{17}{5}, \quad z = 4$$

13.4 Coding

Write a method `system_solve(A,b)` that solves the linear system **Ax = b** using Gaussian elimination. You may assume the solution exists and is unique. **A** should be an $n \times n$ coefficient matrix and **b** is an $n \times 1$ column vector. Some example input is given for you.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300 ## for high-dpi displays. edit as needed
```

```
A = np.array([[1,2,-1],[2,-1,3],[3,1,3]], np.float64)
b = np.array([[1,3,10]], np.float64).T

## The following line creates the augmented matrix $(A | b)$. You should use this in␣
 ↪your method
AA= np.hstack([A,b])
```

```
## Your code goes here. Insert cells as needed.
```

The following cell will check your code on 10 random 5x5 matrices

```
for i in range(10):
    A = np.random.rand(5,5)*10
    x = np.random.rand(5,1)*10
    b = A@x
    assert((np.abs(system_solve(A,b)-x)<1e-10).all())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 5
      3 x = np.random.rand(5,1)*10
      4 b = A@x
----> 5 assert((np.abs(system_solve(A,b)-x)<1e-10).all())

NameError: name 'system_solve' is not defined
```

13.5 Measuring Error

The Gaussian Elimination algorithm has an error that increases with the size of the input matrix. In this section you will approximate the rate at which that error grows. Assume the error can be modeled by a polynomial $$err(N) \sim N^k$$

where N is the number of unknowns in the linear system and k is a constant to be determined. It should be noted that in general the error term depends on the relative sizes of the elements in the coefficient matrix. We are choosing them to be random from $[0, 1]$ so they will usually behave reasonably the same and so the problem is simplified in our case.

13.5.1 Approach

You will calculate the error in your linear system solver on several systems up to size $N = 1000$. For each size N you will solve 10 random systems and average the error $e(N)$ over the 25 runs. You will then find a polynomial fit for the dataset N vs. $e(N)$.

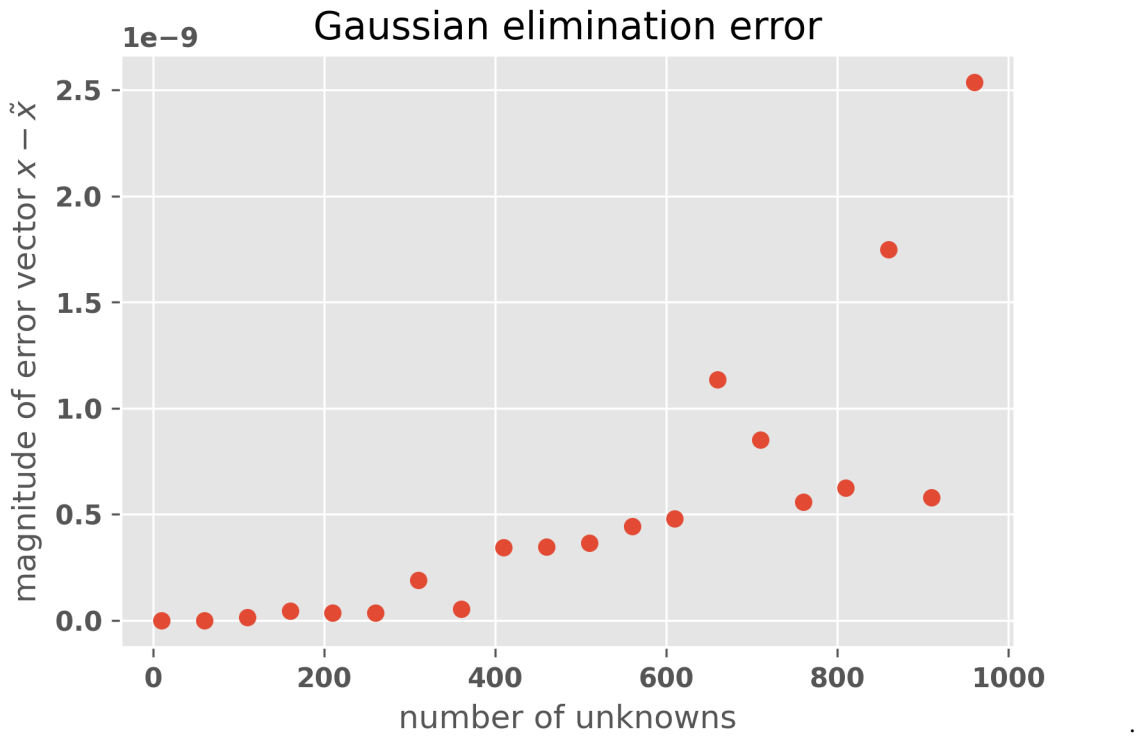
Write a method that takes as parameters the matrix size N and the number of repetitions to compute. Create two random uniform matrices: A and x (uniformly random over $[0, 1]$, by using `np.random.rand`. Compute $b = Ax$. Use your linear solver to find $\tilde{x}$ given A, b and determine the length of the error vector $x - \tilde{x}$. Do this for each repetition and return the *median* error (length of the error vector). We are using the **median** instead of the mean because the mean is too sensitive to outliers and this investigation is rife with outliers!

```
## Your code goes here. Insert cells as needed.
```

Now collect data on various values of N up to 1000. Be judicious: this problem takes a while to solve for large matrices. You should end up with a vector X that contains matrix sizes (N) up to 1000 and Y that contains the average error $e(N)$.

```
## Your code goes here. Insert cells as needed.
```

Now create a scatter plot of N vs $e(N)$. An example plot is shown here




```
## Your code goes here. Insert cells as needed.
```

13.6 Regression

You have written a regression routine before, but we will use some built into numpy. Here's sample code for doing a quadratic regression

```
x = # x data as numpy array or python list
y = # y data as numpy array or python list
coefficients = np.polyfit(x, y, 2)  # 2 indicates quadratic
a2, a1, a0 = coefficients # in decreasing order of powers
quadratic_model = np.poly1d(coefficients) # make a2x^2 + a1x + a0
y_fit = quadratic_model(x) # now y_fit is a vector
```

First you should compute a quadratic regression and superimpose the resulting parabola on a scatterplot of the data

```
## Your code goes here. Insert cells as needed.
```

13.6.1 A regression problem and a solution

Depending on your data the parabola you've plotted may or may not look like a decent fit. But it has one glaring problem. By finding a quadratic fit we are assuming $O(N^2)$ growth of our error term. But it could be $O(n^3)$ or $O(n^{2.3})$. What we want to find is the best **exponent** $O(n^k)$ for polynomial growth. This is the perfect time to use a log-log plot. If you transform your data $x_{log} = \log(x)$ and $y_{log} = \log(y)$ and perform a linear fit, the slope of the best fit line tells you the order of growth k (derivation of this is discussed in class.)

You should find the best log-log plot slope. Then make a scatter plot of the values `x_log` and `y_log` along with the best fit line. Above the graph print the coefficients of the plot and state your best estimate of the growth rate $err(N) \sim N^k$

```
## Your code goes here. Insert cells as needed.
```

Finally, find the correlation coefficient for `x_log` and `y_log`. What amount of the variance is explained by your linear model? (There are several built in methods than can find this value)

```
## Your code goes here. Insert cells as needed.
```

CHAPTER

FOURTEEN

LOG-LOG REGRESSION

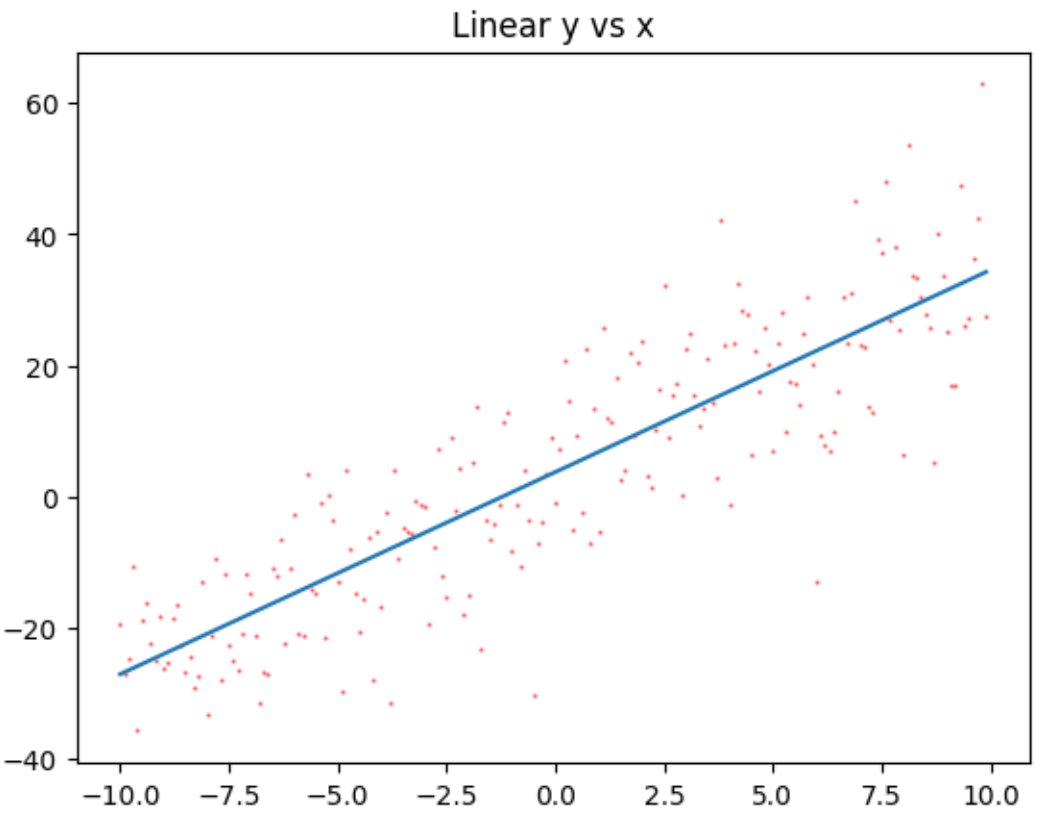
Linear Regression is quite capable of solving non-linear problems if you know how to properly pre-process your data. Let's look at a few types of datasets we can regress by using logarithmic transformations.

14.1 Regular Linear Regression

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

First let's analyze a typical linear dataset

```
x = np.arange(-10,10,0.1)
y = 3*x + 5 + np.random.normal(0,10,len(x))

m,b = np.polyfit(x,y,1)
y_fit = np.poly1d((m,b))(x)
plt.scatter(x,y, color="red", alpha=0.5, s=0.5)
plt.plot(x,y_fit); # note the semicolon here. what does it do?
plt.title("Linear y vs x");
```



And find r and m

```
print(m)
np.corrcoef(x,y)
```

```
3.0845960050150274
```

```
array([[1.        , 0.86180876],
       [0.86180876, 1.        ]])
```

14.2 Exponential Regression

If we believe $y = Ca^x$ then by regressing x against $\ln y$ we can determine a.

$$\begin{aligned} y &= \\ Ca^x & \\ \ln y &= \\ \ln C + x \ln a & \end{aligned}$$

This is a line with slope $\ln a$ and intercept $\ln C$

```
x = np.arange(3,5,0.01)
y = 0.25*3**x

# add noise, but keep y > 0
```

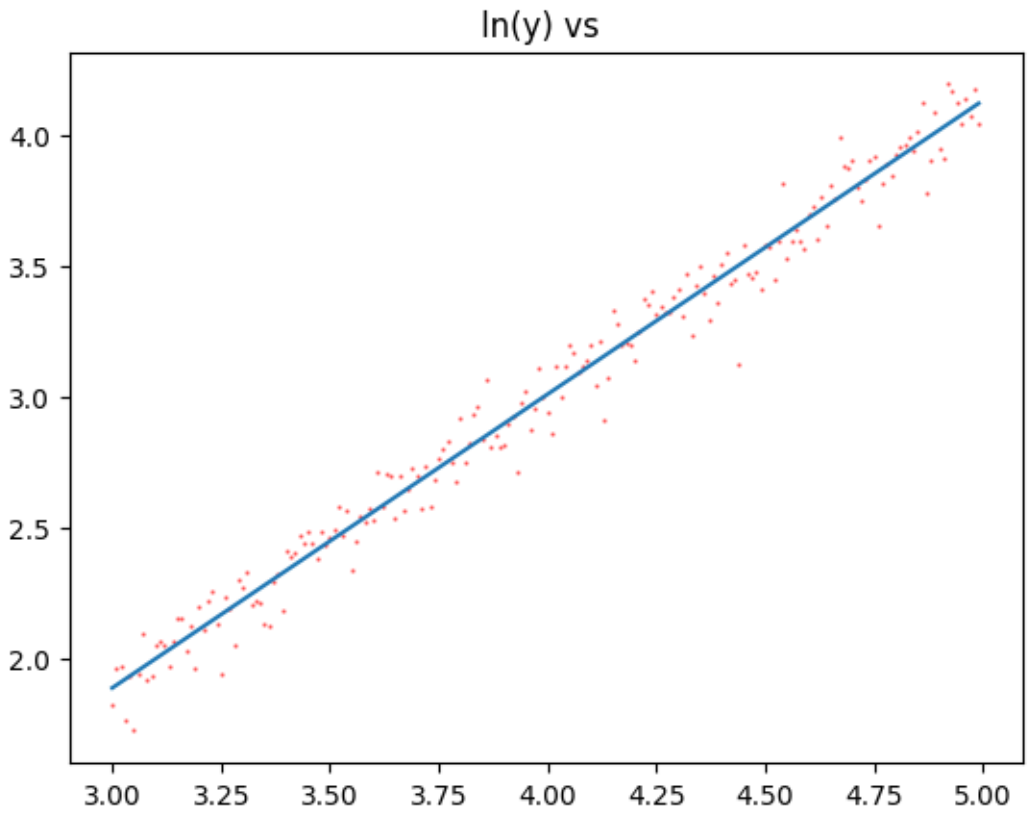
(continues on next page)

(continued from previous page)

```
for i in range(len(y)):
    while True:
        noise  = random.gauss(0,y[i]/10)
        if (y[i]+noise > 0):
            break
    y[i] += noise

plt.scatter(x,y,s=0.5);
plt.title("Exponential correlation");
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 7
      5 for i in range(len(y)):
      6     while True:
----> 7         noise  = random.gauss(0,y[i]/10)
      8         if (y[i]+noise > 0):
      9             break

NameError: name 'random' is not defined
```

```
# transform y
y_t = np.log(y) ## this is ln

m,b = np.polyfit(x,y_t,1)
y_fit = np.poly1d((m,b))(x)
plt.scatter(x,y_t, color="red", alpha=0.5, s=0.5)
plt.plot(x,y_fit);
plt.title("ln(y) vs ");
```



And find r and a and C

```
print("base = " , np.exp(m))
print("C = ", np.exp(b))
print(f"r = {np.corrcoef(x,y)[1,0]}")
```

```
base =  2.9975765382313955
C =  0.24812105741218246
r = 0.95069612414204
```

14.3 Log-Log Regression

If we believe $y = Cx^k$ then by regressing $\ln x$ against $\ln y$ we can determine k.

$$\begin{aligned} y &= \\ Cx^k & \\ \ln y &= \\ \ln C + k \ln x & \end{aligned}$$

This is a line with slope k and intercept $\ln C$

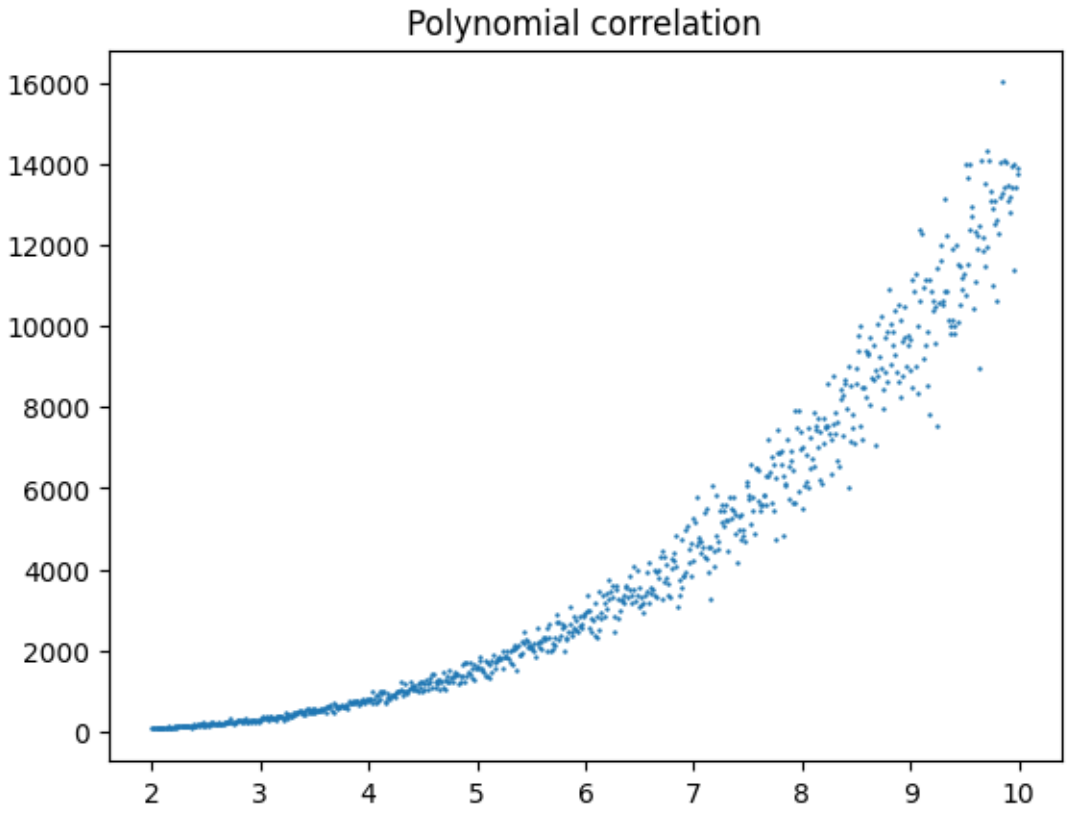
```
import random
```

```
x = np.arange(2,10,0.01)
y = 10*x**3.14
```

(continues on next page)

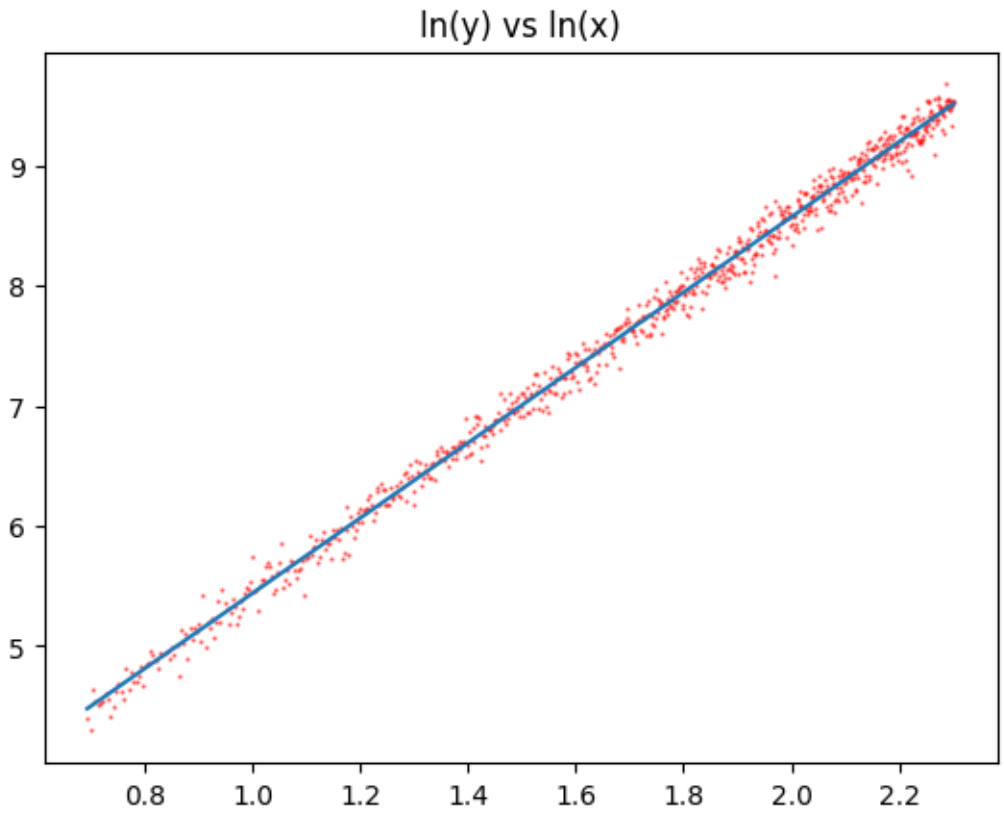
(continued from previous page)

```
for i in range(len(y)):
    while True:
        noise  = random.gauss(0,y[i]/10)
        if (y[i]+noise > 0):
            break
    y[i] += noise
plt.scatter(x,y,s=0.5);
plt.title("Polynomial correlation");
```




```
# check y for 0
print(np.min(y))

# transform y and x
x_t = np.log(x)
y_t = np.log(y)

m,b = np.polyfit(x_t,y_t,1)
y_fit = np.poly1d((m,b))(x_t)
plt.scatter(x_t,y_t, color="red", alpha=0.5, s=0.5)
plt.plot(x_t,y_fit);
plt.title("ln(y) vs ln(x)");
```

```
73.2177132494846
```



And find r and a and C

```
print("degree = " , m)
print("C = ", np.exp(b))
print(f"r = {np.corrcoef(x,y)[1,0]}")
```

```
degree =  3.133547086987462
C =  10.03662528625072
r = 0.9376281240390777
```

CHAPTER

FIFTEEN

STRASSEN'S ALGORITHM (OPTIONAL EXTENSION)

Matrix multiplication in the naive implementation requires $O(n^3)$ floating point operations on a matrix with n^2 entries. In fact it seems unlikely one could do better. Each of the n^2 entries in the product requires a dot product of two length n vectors. Yet in 1969, Volken Strassen shocked numerical analysists with his algorithm requiring $O(n^{\log_2 7}) \approx O(n^{2.73})$ floating point operations. The search for a lower bound on the complexity of matrix multiplication continues today, with recent results hovering around $O(n^{2.3})$.

15.1 Goal

You will implement Strassen's algorithm using Python and numpy. You will then perform tests to verify the achieved asymptotic running time of Strassen is actually lower than the running time of your previously implemented (non-numpy) matrix multiplication.

15.2 Details

Read up on Strassen online (wikipedia has a good article, for example) and implement his algorithm using numpy. **Your algorithm can assume** $n = 2^k$. Making Strassen work otherwise requires really messy memory management best left to C-type languages. Test correctness of your code on several random matrices by comparing your output to numpy ($A@B$). To simplify things, you should work with **integer** matrices only (floating-point introduces rounding errors that we don't want to deal with.)

Once you know it works, do some regression analysis on your original algorithm and Strassen and compare their running times. The easiest way to do this is in Python is with `timeit.timeit`. Here's an example

```
import timeit

i = 4
A = np.random.randint(-10,10,(2**i, 2**i))
timeit.timeit('A@A', number=25, globals = globals())/25
```

The `timeit` function evaluates the first string argument. If the string references values defined elsewhere you need to pass in the current state of the python interpreter which is what the `globals` argument is for. Note we divide by 25 to get an average time per operation. (Running times are usually very centrally distributed so you need not worry about outliers – the mean is a perfectly good measure of center.)

Warning – these algorithms run pretty quickly on $n = 2^6$ or $n = 2^7$ but after that become *very* slow. You will need to proceed judiciously but also gather enough data for a valid conclusion. (In my testing I let my Lenovo laptop run for about 30 minutes).

N.B.: Strassen is recursive and has a base case. Please use the base case $n = 4$ and return `A@B`. This will ensure we have similar results.

15.3 Results

You should graph the averaged running times of Strassen and Naive multiplication on the same graph. Then perform an appropriate regression analysis to determine the respective orders of growth to prove Strassen is asymptotically faster. Support your conclusions with calculations and graphs. (See the Gaussian elimination lab for regression operations).

Finally, just for humility's sake, print out a nicely formatted table comparing the above two algorithms with numpy's built in multiply operation. (Don't feel too bad, it's using a compiled C library).

CHAPTER

SIXTEEN

GAUSS PRESENTATION

Carl Friedrich Gauss was sickeningly brilliant. Prepare a 10-15 minute lecture on the accomplishments of Gauss, focusing primarily on mathematics. Give a moderately technical presentation that your classmates will understand that highlight a few of his greatest hits. This should be a whiteboard presentation – not powerpoint! (If you need a couple slides to show complex diagrams that would be OK, but not needed). This should be presented next Tuesday or Thursday and the beginning of class.

CHAPTER

SEVENTEEN

MATRIX INVERSION

You can invert a matrix A by creating the $n \times 2n$ matrix $[A \quad I]$. This means you stack A right next to the $n \times n$ identity matrix. Then perform Gaussian Elimination on A just like you were solving a system of equations. You will end up with the matrix $[I \quad A^{-1}]$. Sounds easy. But matrix inversion is highly unstable numerically. You will investigate in this lab.

1. The Hilbert matrix $a_{ij} = 1/(i + j + 1)$ is notoriously unstable. (Read about it online; note that our definition uses "+1" because python and math are different sometimes.)
2. Implement matrix inversion and check it on some small ~~Hilbert matrices~~ **random real matrices**. (Check by verifying $AA^{-1} = I$. This will *not* be exact so account for rounding errors).
3. Make a plot of matrix size vs. error. Error is defined as the Frobenius norm of the matrix $(AA^{-1} - I)$ so $err = ||(AA^{-1} - I)||_F$. (It's trivial to do in numpy so look it up!)
4. Implement GEPP (Gaussian elimination with partial pivoting) and make a similar plot to part 2. You should see the errors decrease with GEPP.
5. Finally, plot the error in the inherent `np.linalg.inv` function and compare it to yours.

You can seen an example of GEPP here. Note they do NOT make the diagonal all 1's like we do – but you can keep doing it our way. The main idea is the pivoting.

I wrote this after a short investigation of Hilbert matrices which are known to be poorly conditioned (the condition number grows quite predictably, if you want to look it up). I assumed that GEPP would behave poorly in direct correlation to the condition number but it does *not*. Random matrices provide a much cleaner pattern.

CHAPTER

EIGHTEEN

RUNNING TIME ANALYSIS

Use numpy and pandas for this lab. Download the csv file "matrix_multiply_times.csv" and put it in this folder

```
!wget https://aet-cs.github.io/white/ML/lessons/matrix_multiply_times.csv
```

```
--2024-10-03 20:02:17--  https://aet-cs.github.io/white/ML/lessons/matrix_multiply_
↪times.csv
Resolving aet-cs.github.io (aet-cs.github.io)... 185.199.108.153, 185.199.111.153,␣
↪185.199.110.153, ...
Connecting to aet-cs.github.io (aet-cs.github.io)|185.199.108.153|:443...
```

```
connected.
```

```
HTTP request sent, awaiting response...
```

```
200 OK
Length: 897 [text/csv]
Saving to: 'matrix_multiply_times.csv.5'


          matrix_mu   0%[                    ]       0  --.-KB/s
```

```
matrix_multiply_tim 100%[===================>]     897  --.-KB/s    in 0.07s

2024-10-03 20:02:18 (13.3 KB/s) - 'matrix_multiply_times.csv.5' saved [897/897]
```

Read the csv as a DataFrame and display it (`pd.read_csv` will help)

Drop the column "Matrix row size" and save it as a *new* dataframe (I like to use the names df,df2,df3,etc as I do this kind of work) (read about `df.drop`)

Plot the data frame data, `x` is the 'exp' column, the `y` values are the other columns (times in seconds). Use pandas e.g.

```
df2.plot(...)
```

Add new columns that represent the log transformations of the existing time columns, because we want a log-log analysis eventually. Here's an example

```
df2['strass_log'] = np.log(df['Strassen Time']) ## PLEASE use tab-completion for this!
```

Then *drop* the old (non-log) columns. Store the result in a *new* data frame that has only 5 columns (one 'exp' and four log-times)

Plot the new dataframe line plots: x vs. log-times

Now we want to do a log-log regression. You will need numpy. Luckily pandas and numpy play nicely together. For example you could say

```
df = #some dataframe
x = df['x']
y = df['y']
coeffs = np.polyfit(x,y,1)
```

And you can even add in things like

```
y = np.log(y)
x = 2**x
x = x**2
y = y + np.sqrt(y)
```

for example

Do the log-log regressions and print out a table in this format

```
   algorithm name, rate of growth (degree), corrcoeff
```

Recall you can get corrcoeff from

```
np.corrcoeff(x,y)[0,1]
```

You should notice some problems in your results, and the problems will depend on what you did above. What did you expect and how does it compare to what you got? Now your job is to fix it. Here are some things to consider

- Some regression may not even work because it doesn't like the data
- Are the units of your "x" column in the regression correct? (They should be log of the row length)
- What base are you using with the logs and are you consistent?
- Small values of n might be skewing the data. Filter them out (see below)
- Rows with NA value are also problematic. Drop them judiciously
- Make sure you understand how to interpret the regression equations and numpy results

Work through the Pandas filtering tricks section and then do your analysis afterwards.

18.1 Pandas filtering tricks

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame({'x': [1,2,3,10,20,-10,np.nan], 'y':[5,4,3,4,1,0,9]})
df['log_y'] = np.log2(df['y'])
```

```
/home/pewhite/github/aet-cs/aet-cs.github.io/white/ML/env/lib/python3.11/site-
↪packages/pandas/core/arraylike.py:399: RuntimeWarning: divide by zero␣
↪encountered in log2
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

```
df
```

```
      x  y     log_y
0   1.0  5  2.321928
1   2.0  4  2.000000
2   3.0  3  1.584963
3  10.0  4  2.000000
4  20.0  1  0.000000
5 -10.0  0      -inf
6   NaN  9  3.169925
```

You can index columns by boolean comparisons

```
df[df.x>5]
```

```
      x  y  log_y
3  10.0  4    2.0
4  20.0  1    0.0
```

This just returns a list of T/F values

```
df.x>5
```

```
0    False
1    False
2    False
3     True
4     True
5    False
6    False
Name: x, dtype: bool
```

Here's a different syntax

```
df['x']<5
```

```
0     True
1     True
2     True
3    False
4    False
5     True
6    False
Name: x, dtype: bool
```

```
df['log_y']==-np.inf
```

```
0    False
1    False
2    False
3    False
4    False
5     True
6    False
Name: log_y, dtype: bool
```

Boolean 'not' is ~ in pandas

```
df[~(df['log_y']==-np.inf)]
```

	x	y	log_y
0	1.0	5	2.321928
1	2.0	4	2.000000
2	3.0	3	1.584963
3	10.0	4	2.000000
4	20.0	1	0.000000
6	NaN	9	3.169925

```
df[~df['x'].isna()]
```

	x	y	log_y
0	1.0	5	2.321928
1	2.0	4	2.000000
2	3.0	3	1.584963
3	10.0	4	2.000000
4	20.0	1	0.000000
5	-10.0	0	-inf

18.2 Analysis

Complete your final analysis here of the 4 running times by using techniques you have learned in this notebook. Print out your final table of the the running time comparisons, based on your best justifiable arguments

Part IV

Next Steps

CHAPTER

NINETEEN

READING FOR RESEARCH

I believe there are three pillars of research: specific content expertise, broad knowledge of the field, and an ability to identify relevant problems that need to be solved. (I'm still thinking for a catchy three-word phrase for this.)

You're developing specific expertise in Machine Learning by taking this course. You are learning to identify relevant problems by adding to your Money List. And now we will work on broad knowledge by regularly reading publications in the field of computer science.

I've collected several good links on the class GitHub Page under the heading Research. For this assignment (which will repeat regularly, maybe every 2 weeks or so), you should read some articles and choose one to present to the class.

The presentation will usually be very informal – we'll just go around the room and ask you to stay at your seat and give us a quick summary. Tell us what the article is about, what problem they're solving, what motivated them, and what their solution is. Know the article well enough to give specifics and respond to questions.

I know that this kind of self-directed reading is one of the very best ways to get inspired to learn more about a field and to find new problems to solve. I also hope that your brief presentations to the class will help inspire the rest of the class. We'll all hear about a dozen different new results in computer science. It should be lots of fun!

Specifically for next week, please select one article from the links above and be ready to tell us about it on probably Monday. All the links are good, but the first one (Communications of the ACM) might be the best one-stop-shop. It depends on what you like, though.

CHAPTER

TWENTY

HOBO DATA, STUDENT VERSION

```
import pandas as pd
import matplotlib.pyplot as plt
import scipy
import sklearn as sk
import cv2
import numpy as np
```

We'll have three groups working on different parts of the HOBO data (described below).

- Group 1 will extract sensor coordinates from a satellite image screenshot. You will produce a funtion that returns screen cooriadte for x,y pairs as in:
  - pos("A","30") = 143,299
- Group two will ingest and clean the all the relevant data and store it in arrays indexed by <x,y,t> (horizontal, vertical, time, so that a user can query
  - "A, 30, 01/27/2022 17:28:08" and return the reading -0.95 34.40
- Group 3 will explor 3D plots in matplot lib specifically contour and surface plots, and then animate the plots as parameters change. You can start with variants of $y = e^{-x^2}\sin(t)$ and $z = 4\sin(t)e^{-x^2-y^2}\cos\left(4x^2 + 4y^2\right)$

20.1 Part 1: Extracting Coordinates

There's some data being collected here Read about the hobo data first. Then identify the data for the "red" grid of sensors for ACL. in part 1 you will approximate the coordinates of these sensors by analyzing a screenshot. You're using the openCV library "cv2" which should be installed in this environment.

```
pic = cv2.imread("red_sensor_image.png", cv2.IMREAD_COLOR)
```

```
[ WARN:0@0.024] global loadsave.cpp:241 findDecoder imread_('red_sensor_image.png
↪'): can't open/read file: check file path/integrity
```

Find the shape and then select a submatrix that is square but doesn't cut off any data

```
pic.shape
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 pic.shape
```

(continues on next page)

(continued from previous page)

```
AttributeError: 'NoneType' object has no attribute 'shape'
```

```
pic = pic[:,0:1180,:]
pic.shape
```

```
(1180, 1180, 3)
```

Convert the image from BGR to RGB (openCV uses a different ordering of color data) and then display. Use the commands `cv2.cvtColor` and `plt.imshow()`

To find the red dots, we choose to analyse just the red layer of pixel information. Split the image into RGB channels using `cv2.split` and dislay the red channel only, as grayscale.

You see the red pixels are nearly white and the background is nearly black. We want to crush this image into 0s and 255s for black and white, so the red dots really pop. Use matplotlib filtering syntax (much like pandas filtering). Any pixel with a red above 200 should become 255. Anything below should be 0. (Be sure to **copy** your red matrix first in case you mess up.)

```
## print the red matrix here as a sanity check
# and its size
```

Plot your black and white matrix here

20.1.1 Machine Learning – KMeans

We need to know the location of the centers of these dots, in the coordinate space of the picture. the kMeans algorithm is perfect for this. It finds local centers of clusters. I'll get you started with a list of all the pixels that have white centers. The KMeans algorithm will determine all the centers and list them for you as coorindate pairs. You should tell it how many centers you're looking for. Then create a plot with (a) the original image and (b) the centers plotted as white 'x' symbols (look at plt.scatter(marker = …))

```
from sklearn.cluster import KMeans

# Extract the coordinates of the white pixels (where red == 255)
white_pixels = np.column_stack(np.where(red2 == 255))
white_pixels
```

```
array([[  30,  828],
       [  30,  829],
       [  30,  830],
       ...,
       [1162,  192],
       [1162,  193],
       [1162,  194]])
```

```
# Apply KMeans to find the centers of the white dots
```

20.2 Part 2: Ingesting Data

Use pandas to read spreadsheet csv files into data frames. You will probably want to merge frames to get one big one. Then think about how to filter or aggregate the data so it's ready to go when someone asks for a location and time – you should give it to them. (Assume their input is valid at first, but then handle the case when it isn't)

20.3 Part 3: Visualization

Read up on matplotlib 3d plotting. There are four different things here

- 3d plots and contour plots (related)
- animation
- animation with 3d plots
- saving the animation as a file

Thare are plenty of tutorials on saving a simple 2d animation. I'd do that first. Then play around with 3d contour and surface plots. Finally try to get an exported video of a 3d animation. (you might need ffmpeg, it's easy to install on the unix side of WSL)

The goal here is to be able to take grids of data (x,y,t,z) corresponding to location and time and an output like temperature, and plot that as a 3d surface and/or contour and then animate it for a range of times.

CHAPTER

TWENTYONE

ML BOOK CLUB

You are a member of research book club (OK so it's an 'article club' but that sounds ridiculous). Working together please pick 3 articles from the resources on this page that are interesting to the whole group. The articles do not have to have any common topic or thread, unless this is what the group chooses. Each group member will read one article and take notes. Note: the article should be "lengthy." Some of the things you'll find here are just overviews or summaries. A journal article, or something of similar heft written for a technical audience, is the target.

Next week we will meet in book club groups. Each group member will report about their article to the group. Group members can discuss the three articles and what they found interesting and inspiring about them.

As you read your chosen article, keep the following questions in mind. Try to address each of these points in your book club meeting. You may want to take some written notes to help guide your processing of the material.

- What inspired this article?
- What problem does it solve?
- What is the primary creative insight that led to the solution they're proposing?
- What are extensions or applications that you can think of for this research?
- When you think about doing research, and posing research questions of your own, how has this article helped you better understand the process? (This may be difficult at first but I believe every good article teaches us about the research process.)