MATRIX ALGEBRA WITH COMPUTATIONAL APPLICATIONS

Dirk Colbry
Michigan State University

LibreTexts™

Michigan State University

Matrix Algebra with Computational Applications

Dirk Colbry

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (https://LibreTexts.org) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide and accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden born from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.

LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by NICE CXone Expert and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptions contact info@LibreTexts.org or visit our main website at https://LibreTexts.org.

This text was compiled on 06/08/2025

TABLE OF CONTENTS

21: 11 Pre-Class Assignment - Vector Spaces

22: 11 In-Class Assignment - Vector Spaces

23: 12 Pre-Class Assignment - Matrix Spaces

24: 12 In-Class Assignment - Matrix Representation

25: 13 Pre-Class Assignment - Projections

26: 13 In-Class Assignment - Projections

27: 14 Pre-Class Assignment - Fundamental Spaces

28: 14 In-Class Assignment - Fundamental Spaces

- 28.0: Introduction
- 28.1: Pre-class assignment review
- 28.2: Four Fundamental Subspaces
- 28.3: Practice Example

29: 15 Pre-Class Assignment - Diagonalization and Powers

- 29.0: Introduction
- 29.1: Eigenvalues and eigenvectors review
- 29.2: Diagonalizable Matrix
- 29.3: Assignment wrap-up

30: 15 In-Class Assignment - Diagonalization

- 30.0: Introduction
- 30.1: Pre-class Assignment Review
- 30.2: Diagonalization
- 30.3: The Power of a Matrix

31: 16 Pre-Class Assignment - Linear Dynamical Systems

- 31.0: Introduction
- 31.1: Linear Dynamical Systems
- 31.2: Markov Models
- 31.3: Ordinary Differential Equations
- 31.4: Assignment wrap up

32: 16 In-Class Assignment - Linear Dynamical Systems

- 32.0: Introduction
- 32.1: Epidemic Dynamics - Discrete Case
- 32.2: Epidemic Dynamics - Continuous Model
- 32.3: Population Dynamics

33: 17 Pre-Class Assignment - Decompositions

- 33.0: Introduction
- 33.1: Matrix Decomposition
- 33.2: Decompositions
- 33.3: Assignment wrap-up

34: 17 In-Class Assignment - Decompositions and Gaussian Elimination

- 34.0: Introduction
- 34.1: Pre-Class Review
- 34.2: Decompositions
- 34.3: Focus on LU

35: 18 Pre-Class Assignment - Inner Product

- 35.0: Introduction
- 35.1: Inner Products
- 35.2: Inner Product on Functions
- 35.3: Assignment wrap-up

36: 18 In-Class Assignment - Inner Products

- 36.0: Introduction
- 36.1: Pre-class Review
- 36.2: Minkowski Geometry
- 36.3: Function Approximation

37: 19 Pre-Class Assignment - Least Squares Fit (Regression)

- 37.0: Introduction
- 37.1: Least Squares Fit
- 37.2: Linear Regression
- 37.3: One-to-one and Inverse transform
- 37.4: Inverse of a Matrix
- 37.5: Assignment wrap-up

38: 19 In-Class Assignment - Least Squares Fit (LSF)

- 38.0: Introduction
- 38.1: LSF Pre-class Review
- 38.2: Finding the best solution in an overdetermined system
- 38.3: Pseudoinverse

39: 20 In-Class Assignment - Least Squares Fit (LSF)

- 39.0: Introduction
- 39.1: LSF Example - Tracking the Planets
- 39.2: LSF Example - Predator Pray Model
- 39.3: LSF Example - Estimating the best Ellipses

40: Pre-Class Assignment - Solve Linear Systems of Equations

- 40.0: Introduction
- 40.1: Linear Systems
- 40.2: Under Defined Systems
- 40.3: Invertible Systems
- 40.4: Overdefined systems
- 40.5: System Properties
- 40.6: Assignment wrap up

41: 21 In-Class Assignment - Solve Linear Systems of Equations using QR Decomposition

- 41.0: Introduction
- 41.1: Pre-Class Review
- 41.2: Solve Linear Systems
- 41.3: Overdetermined Systems
- 41.4: Underdetermined Systems

42: Supplemental Materials - Python Linear Algebra Packages

- 42.0: Introduction
- 42.1: Matplotlib
- 42.2: Review of Python Math Package
- 42.3: Review of Python Numpy Package

Licensing

*A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.*

About the Lead Author

Dr. Dirk Colbry is faculty member in the Department of Computational Mathematics, Science and Engineering (CMSE) at Michigan State University (MSU). Dr. Colbry has decades of experience in curriculum development and teaching across a wide range of subjects, including: numerical linear algebra, parallel programming, microprocessors, artificial intelligence, scientific image analysis, compilers, GPU programming, next generation architectures, tools for computational modeling, algorithm analysis, and professional skills for scientists and engineers.

In addition to teaching and curriculum development, Dr. Colbry is an expert in computer vision and scientific image understanding, and has collaborated on research in fields as diverse as Engineering, Toxicology, Plant and Soil Sciences, Zoology, Mathematics, Statistics, and Biology. Recent projects include research in Image Phenomics; developing a commercially-viable large scale, cloud based image pathology tool; and helping develop methods for measuring the carbon stored inside of soil. In addition to his primary research focus in Scientific Image Understanding and Machine Learning, Dr. Colbry also researches computational education and high-performance computing (HPC). Prior to joining CMSE, Dr. Colbry worked for the MSU Institute for Cyber-Enabled Research (iCER) as a computational consultant and Director of the High Performance Computing Center (HPCC).

These Open Educational Resources (OER) were developed as part of a restructuring of the Michigan State University Matrix Algebra Course (MTH314) in Fall Semester of 2017. All of the OER materials are provided as Jupyter notebooks, which are open source tools that integrate multiple resources (websites, word processors, LaTeX, math, and programming) into a digital “notebook.”

Instructors interested in using these materials are encouraged to reach out to Dr. Dirk Colbry (colbrydi@msu.edu) in advance to gain access to additional materials and examples designed to support teaching and student evaluation.

Acknowledgments

This textbook is a product of the Michigan State University Libraries' Open Educational Resource (OER) program led by Regina Gong, OER and Student Success Librarian. The content of this book is a work in progress, with contributions and collaboration from the MTH314 instructors listed below. Individual contributions range from co-authoring specific Jupyter notebooks to testing and providing feedback of the materials.

Lead Instructors

- Dr. Dirk Colbry
- Dr. Ming Yan
- Dr. Matthew Mills
- Dr. Paul Speaker
- Dr. Zhichao Peng
- Dr. Sulin Wang
- Dr. Rongrong Wang

Graduate Teaching Assistance

- Shuyang Qin
- Cullen Avery Haselby
- Haoyang Chen
- Kai Huang
- Nathan Brugnone
- Yao Li
- Thomas Chuna

Undergraduate Learning Assistance

- Amanda Bowerman
- Zachary Matson
- Noah Jankowski
- Nicholas Mouaikel
- Marv Zurek
- Ishaan Pathak
- Sam Tracht
- Dave Yonkers
- Heather Noonan
- Drew Pype

CHAPTER OVERVIEW

1: Matrix Algebra class preparation checklist

It is recommended that all students complete the following tasks before showing up for the first day of class.

1.1: Get Jupyter Working

We will be using Jupyter notebooks for most of our student/instructor communication. Please attend class with the latest version of Anaconda Python installed on your computer (Anaconda includes Jupyter). Instructions for installing and using Anaconda and Jupyter can be found here:

- Jupyter Getting Started Guide

1.2: Review Python Packages

In this class we will be using programming to teach linear algebra and therefore assume that everyone knows how to use python. the instructors provide the following reference as a review for the most common python libraries we expect you to be able to use for class (numpy, scipy, math, sympy, etc). please review this tutorial to make sure you are properly prepared for class:

- Python Library Review

1.3: Complete Pre-class Assignment

You are expected to come to class ready to complete the in-class assignments. To prepare for most classes, students are required to complete pre-class assignments (typically two pre-class assignments per week). These assignments will be in the form of jupyter notebooks and will include reading, videos and programming activities. The required time for each pre-class will vary but the instructors have tried to make them take approximately 30-minutes.

Each pre-class assignment ends with an online survey which students are required to complete in order to earn credit for the assignment. Your first weeks assignments are available on the course OER website or downloadable from the course website.

- Pre-Class Assignment: Vectors

CHAPTER OVERVIEW

2: 01 In-Class Assignment - Welcome to Matrix Algebra with computational applications

---

2.0: Introduction

What can you solve with $Ax=b$?

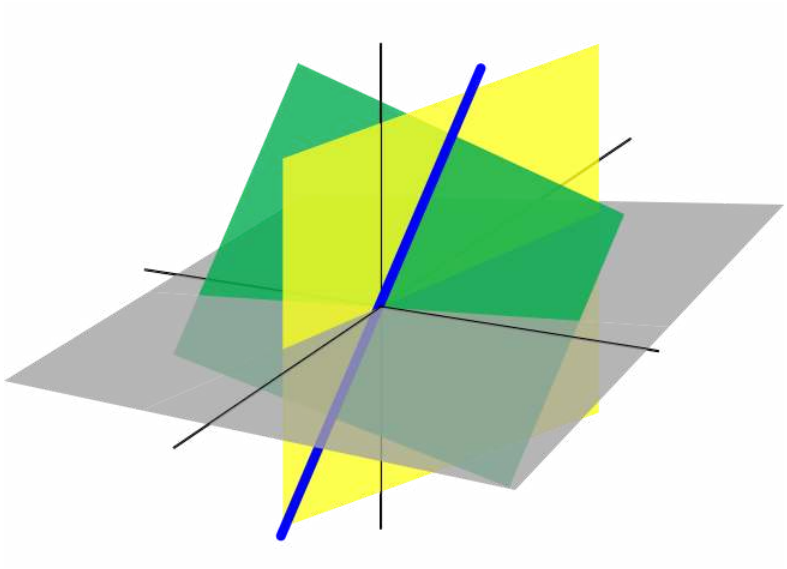


Image from http://Wikipedia.org/.

```
from IPython.display import YouTubeVideo
YouTubeVideo("-aiL8iWhQjc",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Agenda for today's class (80 minutes)

1. (10 minutes) Class Procedures
2. (10 minutes) Introductions
3. (10 minutes) Example
4. (25 minutes) Syllabus, Schedule and other Procedures
5. (5 minutes) Download and review next pre-class assignment

2.1: Class Procedures

All in-class assignments are designed such that you can get started as soon as you show up in class. This is highly recommended. See how far you can get on your own and then you will be ready when it is time for questions. Here are the basic instructions.

Step 1 - Get out your laptop

Feel free to grab one of the laptops in the classroom if you do not have your own.

Step 2 - Create a Course Assignment Folder in your home directory

> **Hint**
>
> Store all notebooks in the same folder on your computer. They will work better that way.

Step 3 - Download this jupyter notebook

Download a copy of this notebook (ipynb file which tands for ipython notebook).

Step 4 - Open this assignment in Jupyter

Open the downloaded file inside of jupyter either on your laptop or upload the file to an on-line server and open it there.

2.2: Introductions

This course appeals to students from many different backgrounds. Let us use this time to go around the room and introduce ourselves to each other.

2.3: Example

Suppose that we have three objects on a balanced beam. Also suppose we know that one has a mass of 2 kg, and we want to find the two unknown masses. Experimentation with a (assume weightless) meter stick produces these two balances. (diagram not to scale)

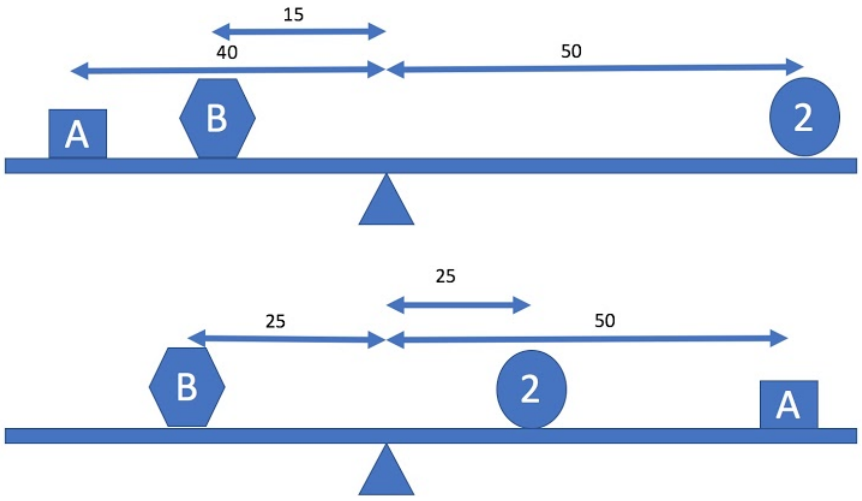


Figure 2.3.1: Image showing two balanced beams, each with three weights. In the top beam is unknown weight A is a distance of 40 to the left of the fulcrum, unknown weight B is a distance of 15 to the left of the fulcrum and a weight of 2 is 50 to the right of the fulcrum. In the bottom beam is the same unknown weights. Weight A is now a distance of 50 to the right of the fulcrum, weight B is a distance of 25 to the left of the fulcrum and the weight of 2 is a distance of 25 to the right of the fulcrum.

For the masses to balance we must have the sum of the moments on the left equal to the sum of the moments on the right, where the moment of an object is its mass times its distance from the balance point. That gives a system of two equations:

$$40A + 15B = 50 \times 2$$

$$25B = 25 \times 2 + 50A$$

? Do This

Find a solution for the above systems of equations and place your solution in the following cell. Make sure you delete the instructional text in the cell first.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Using Python as a calculator, verify that the solution you have found is correct.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Now lets consider a system where we have three unknown masses instead of two. Experimentation with a meter stick produces the two balanced states shown below (diagram not to scale). Write the equations for this system.

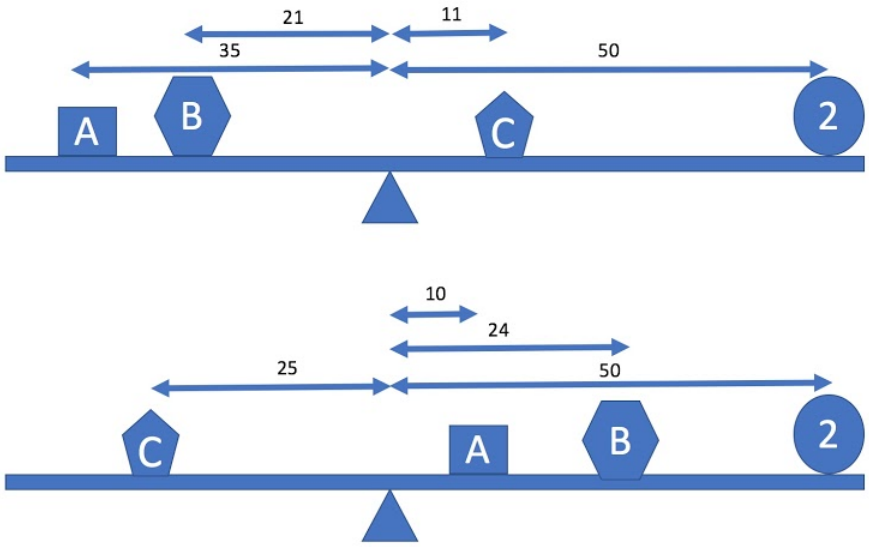


Figure 2.3.2: Image showing two balanced beams, each with four weights. In the top beam is unknown weight A which is a distance of 35 to the left of the fulcrum, unknown weight B is a distance of 21 to the left of the fulcrum, unknown weight C is a distance of 11 to the right of the fulcrum and a weight of 2 is 50 to the right of the fulcrum. In the bottom beam is the same unknown weights. Weight A is now a distance of 10 to the right of the fulcrum, weight B is a distance of 24 to the right of the fulcrum, weight C is a distance of 25 to the left of the fulcrum and the weight of 2 is still at a distance of 50 to the right of the fulcrum.

? Do This

Find a solution to the second set of equations and report the mass for objects A, B and C.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Using Python as a calculator, verify that the solution you have found is correct.

```
# Put your answer to the above question here
```

run | restart | restart & run all

2.4: Syllabus, Schedule and other Procedures

Let us now use the time to review the course Syllabus, schedule and other procedures.

2.5: Download and review next pre-class assignment

I will try my best to post each weeks pre-class assignments before the weekend (hopefully on Thursday before class). Note pre-class assignments are generally due at midnight the day before class. To get credit for the pre-class assignment you will need to do the readings, answer the questions and submit your answers to the embeded google form.

Here is next Pre-class assignments.

- Pre-Class Assignment: Vectors

CHAPTER OVERVIEW

3: 02 Pre-Class Assignment - Vectors

3.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 1.II.1 pg 35-42
- Beezer Chapter V pg 74-88
- ***Boyd Sections 1.1-1.3 pg 1-19***

Assignment Overview

1. Introducing the Course Textbooks
2. Todays Reading
3. Scalars, Vector and Tensors
4. Assignment wrap-up

3.1: Introducing the Course Textbooks

Student self guided learning through assigned readings are required for students to be successful. The course strives to use Open Educational Resources (OER) to help reduce financial burden on the students. To this end we have selected the following textbooks for reading assignments and supplemental examples:

- Introduction to Applied Linear Algebra by Boyd and Vandenberghe
- Linear Algebra by Jim Heffron
- A First Course in Linear Algebra by Robert A. Beezer

DO NOT WORRY You will not be expected to read all three textbooks in this course! In fact, we try to keep the reading at a reasonable level and focus on problem solving. However, most students benefit from seeing material in multiple ways (Reading, Lecture, Practice, etc). Different students (and instructors) also prefer different writing styles and may learn better with different text (which is why we provide more than one).

Students are encouraged to review and become familiar with the style and layout of each text to act as a common reference for the course. If you get stuck on a topic try looking it up and reviewing it in one of the other texts. If you are still stuck you can search the Internet. Do not be afraid to also ask your instructors questions and come to office hours. That is why we are here!!!

? Do This

Download a copy of each textbooks onto your preferred reading device and review the Table of Contents in each text.

As you can see each textbook approaches Linear algebra in a slightly different way. This variety reflects the different philosophies of teaching and different ways of individual learning. One way to evaluate the focus of a particular textbook is to look at it's very first chapter. For Example:

- The ***Beezer*** and ***Heffron*** texts start out with "Systems of linear Equations" and "Linear Systems." These topics are basically the same idea with the focus of defining linear systems as just sets of "linear combinations". Clearly this is a core concept and a good place to start.
- The ***Boyd and Vandenberghe*** text choose to start with "Vectors". In linear algebra the "vector" is a mathematical tool for which all of the mechanics of the math is built. Again, not a bad place to start.

In the first few assignments this course we will be looking at both concepts. You will want to learn and be able to identify linear systems and how to represent them as vectors.

? Question

Find three additional topics (Besides Linear Systems and Vectors) that seem to be common between the three textbooks. You can probably assume that these topics will be important to our course as well.

Put your answer to the above question here.

3.2: Today's Reading

Quite a bit of this pre-class assignment about vectors is motivated from Chapter 1 of the Stephen Boyd and Lieven Vandenberghe Applied Linear algebra book. This material may be review for some students and may be new for others. It is expected that students review the chapter to help them understand the material better.

? Do This

Review ***Sections 1.1, 1.2 and 1.3 in Boyd and Vandenberghe*** and become familiar with the contents and the basic terminology. If you find this material is difficult make sure you take advantage of the survey at the end of this assignment to ask your instructor questions about things that are confusing.

Hint

Many computers and smart phones have a "read to me feature". Some students find it helpful to download the pdf of the textbook and have the computer read to them out loud while they follow along.

3.3: Scalars, Vector and Tensors

The two primary mathematical entities that are of interest in linear algebra are the vector and the matrix. They are examples of a more general entity known as a tensor. The following video gives a basic introduction of scalars, vectors, and tensors. It is fine if you can not understand all of this video. We will learn most of it in this course.

> **Note**
>
> The terms vectors and tensors can get fairly confusing. For the purpose of this class, we will not use the term *Tensor* all that much. Instead we will treat everything as *vectors* (more on this later).

- Direct Link to YouTube Video

```
from IPython.display import YouTubeVideo
YouTubeVideo("ml4NSzCQobk",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Think of a *scalar* as a single number or variable that is an example of a 0th-order tensor. The following are all scalars:

\[1, \frac{1}{2}, 3.1416 \nonumber \]
Defining a *scalar* in Python is easy. For example

```
a = 8
a
```

run | restart | restart & run all

```
8
```

A *vector*, on the other hand, is an *ordered* list of values which we typically represent with lower case letters. Vectors are ordered arrays of single numbers and are an example of 1st-order tensor. The following are all vectors:
Row Vector: \(v = [v_1, v_2, \dots, v_n] \) here \(v_1\), \(v_2\), \(\dots\), \(v_n\) are single numbers.
\[f = [1,2,3,5,8] \nonumber \]
Here \(f\) in the above example is a vector of numbers, and it is the common way we think of vectors.
Note, it is often more common to write vecors vertically. These are often called column vectors:
Column Vector: \(v = \begin{bmatrix} v_1 \cr v_2 \cr \vdots \cr v_m \end{bmatrix} \) here \(v_1\), \(v_2\), \(\dots\), \(v_m\) are single numbers.

Introducing Vectors in Python

In Python, there are multiple ways to store a vector. Knowing how your vector is stored is very important (especially for debugging). Probably the easiest way to store a vector is using a list, which are created using standard square brackets as follows:

```
f = [1, 2, 3, 5, 8]
f
```

run | restart | restart & run all

```
[1, 2, 3, 5, 8]
```

Another common way to store a vector is to use a tuple.

```
b = (2, 4, 8, 16)
b
```

run | restart | restart & run all

```
(2, 4, 8, 16)
```

You can access a particular scalar in your Python object using its index. Remember that Python index starts counting at zero. For example, to get the fourth element in `f` and `b` vectors, we would use the following syntax:

```
print(f[3])
print(b[3])
```

run | restart | restart & run all

```
5
16
```

Later in this course, we may discuss which data format is better (and introduce new ones). At this point let's not worry about it too much. You can always figure out a variable's data type using the `type` function. For example:

```
type(f)
```

run | restart | restart & run all

```
list
```

```
type(b)
```

run | restart | restart & run all

```
tuple
```

Finally, I am not sure if you will need this but always remember, it is easy to convert from a tuple to a list and vice versa (this is called "casting"):

```
#Convert tuple to list
b_list = list(b)
b_list
```

run | restart | restart & run all

```
[2, 4, 8, 16]
```

```
#Convert list to tuple
f_list = tuple(f)
f_list
```

run | restart | restart & run all

```
(1, 2, 3, 5, 8)
```

Vector size

A vector can be used to represent quantities or values in an application. The size (also called dimension or length) of the vector is the number of elements it contains. The size of the vector determines how many quantities are in the vector. We often refer to the size of a vector using the variable `n`. So an `n-vector` has `n` values. A `3-vector` only has 3 values.
The length (`len`) function returns the size of a vector in Python:

```
len(f)
```

run | restart | restart & run all

```
5
```

```
len(b)
```

run | restart | restart & run all

```
4
```

Special Vectors

The following are special vectors with special names.

Standard Unit Vectors

Vectors with a single 1 and the rest of the values are zero have a special name called “Standard Unit Vectors”. The number of different standard unit vectors there are is equal to the size of the vector. For example, a 3-vector has the following standard unit vectors:

\[e_1 = \begin{bmatrix} 1 \cr 0 \cr 0 \end{bmatrix}, e_2 = \begin{bmatrix} 0 \cr 1 \cr 0 \end{bmatrix}, e_3 = \begin{bmatrix} 0 \cr 0 \cr 1 \end{bmatrix} \nonumber \]

Zero Vectors

Vectors with all values of zero also have a special name called “Zero Vectors”. Typically we just use a zero to represent the zero vector. For example: \(0 = \begin{bmatrix} 0 \cr 0 \cr 0 \end{bmatrix} \)

Examples

Vectors are used to represent all types of data that has structures. Here are some simple examples from the Boyd and Vandenberghe textbook:

Location and displacement

A 2-vector can be used to represent a position or location in a space. The first value is the distance in one direction (from the origin) and the second value is the distance in a different direction. Probably most students are famiar with the 2D Cartesian coordinate system where a location can be defined by two values in the `x` and `y` directions. Here is a simple scatter plot in python which show the concept:

```
%matplotlib inline
import matplotlib.pylab as plt
p1 = [2, 1]
p2 = [1, 3]
p3 = [1, 1]

plt.plot(p1[0],p1[1],'*k')
plt.plot(p2[0],p2[1],'*k')
plt.plot(p3[0],p3[1],'*k')

## Add some labels (offset slightly)
plt.text(p1[0]+0.1,p1[1],'$p_1$')
plt.text(p2[0]+0.1,p2[1],'$p_2$')
plt.text(p3[0]+0.1,p3[1],'$p_3$')

## Fix the axis so you can see the points
plt.axis([0,4,0,4])

## 'Run' this cell to view the plot
```

run | restart | restart & run all

```
(0.0, 4.0, 0.0, 4.0)
```

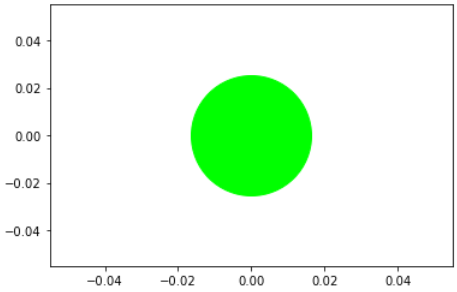
Color

A 3-vector can represent a color, with its entries giving the Red, Green, and Blue (RGB) intensity values (often between 0 and 1). The vector (0,0,0) represents black, the vector (0, 1, 0) represents a bright pure green color, and the vector (1, 0.5, 0.5) represents a shade of pink.

The Python `matplotlib` library uses this type of vector to define colors. For example, the following code plots a point at the origin of size 10000 (the size of the circle, and the value does not have exact meaning here) and color c = (0,1,0). You can change the values for `c` and `s` to see the difference.

```
import warnings
warnings.filterwarnings("ignore")

c = (0, 1, 0)
plt.scatter(0,0, color=c, s=10000);
```

run | restart | restart & run all



Just for fun, here is a little interactive demo that lets you play with different color vectors.

Note

This demo uses the `ipywidgets` Python library which works by default in Jupyter notebook (which is installed on the MSU jupyterhub)) but *NOT* in the newer jupyter lab interface which some students may have installed on their local computers. To get these types of examples working in jupyter lab requires the installation of the ipywidgets plug-in.

```
%matplotlib inline
import matplotlib.pylab as plt
from ipywidgets import interact, fixed

def showcolor(red,green,blue):
    color=(red,green,blue)
    plt.scatter(0,0, color=color, s=20000);
    plt.axis('off');
    plt.show();
    return color

color = interact(showcolor, red=(0.0,1.0), green=(0.0,1.0), blue=(0.0,1.0));
```

Vector Addition

Two vectors of the same size can be added together by adding the corresponding elements, to form another vector of the same size, called the sum of the vectors. For example:

\[\begin{bmatrix} 1 \cr 20 \end{bmatrix} + \begin{bmatrix} 22 \cr -3 \end{bmatrix} = \begin{bmatrix} 23 \cr 17 \end{bmatrix} \nonumber \]

Python Vector Addition

Here is where things get tricky in Python. If you try to add a list or tuple, Python does not do the vector addition as we defined above. In the following examples, notice that the two lists concatenate instead of adding by element:

```
## THIS IS WRONG
a = [1, 20]
b = [22,-3]
c = a+b
c
```

run | restart | restart & run all

```
[1, 20, 22, -3]
```

```
## THIS IS ALSO WRONG
a = (1, 20)
b = (22,-3)
c = a+b
c
```

run | restart | restart & run all

```
(1, 20, 22, -3)
```

To do proper vector math you need either use a special function (we will learn these) or loop over the list. Here is a very simplistic example:

```
a = (1, 20)
b = (22,-3)
c = []
for i in range(len(a)):
    c.append(a[i] + b[i])
c
```

run | restart | restart & run all

```
[23, 17]
```

For fun, we can define this operation as a function in case we want to use it later:

```
def vecadd(a,b):
    """Function to add two equal size vectors."""
    if (len(a) != len(b)):
        raise Exception('Error - vector lengths do not match')
    c = []
    for i in range(len(a)):
        c.append(a[i] + b[i])
    return c
```

run | restart | restart & run all

```
#Lets test it

vecadd(a,b)
```

run | restart | restart & run all

```
[23, 17]
```

Scalar-Vector multiplication

You can also multiply a scalar by a vector, which is done by multiplying every element of the vector by the scalar.
\[3 \begin{bmatrix} 3 \cr -7 \cr 10 \end{bmatrix} = \begin{bmatrix} 9 \cr -21 \cr 30 \end{bmatrix} \nonumber \]

Scalar-Vector Multiplication in Python

Again, this can be tricky in Python because Python lists do not do what we want. Consider the following example that just concatenates three copies of the vector.

```
##THIS IS WRONG##
z = 3
a = [3,-7,10]
c = z*a
c
```

run | restart | restart & run all

```
[3, -7, 10, 3, -7, 10, 3, -7, 10]
```

Again, in order to do proper vector math in Python you need either use a special function (we will learn these) or loop over the list.

? Do This

See if you can make a simple function with a loop to multiply a scalar by a vector. Name your function `sv_multiply` and test it using the cells below:

```
#put your sv_multiply function here

# 'Run' this cell afterwards to register your sv_multiply function
```

run | restart | restart & run all

```
#Test your function here
z = 3
a = [3,-7,10]
sv_multiply(z,a)
```

run | restart | restart & run all

Let us use the following code to test your functon further. Note that this uses the `anchsercheck` function provided by your instructors. Please review Python Linear Algebra Packages for instructions on installing and using `answercheck` .

```
from answercheck import checkanswer
checkanswer.vector(sv_multiply(10,[1,2,3,4]),'414a6fea724cafda66ab5971f542adf2')
```

```
from answercheck import checkanswer
checkanswer.vector(sv_multiply(3.14159,(1,2,3,4)),'f349ef7cafae77c7c23d6924ec1fd36e'
```

3.4: Assignment wrap-up

? Assignment-Specific Question

Are you able to get the sv_multiply function working in part 1 of this assignment? If not, where did you get stuck?

? Question

Summarize what you did in this assignment.

? Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

? Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

? Question

What was the **most** challenging part of this assignment for you?

? Question

What was the **least** challenging part of this assignment for you?

? Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

? Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

? Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

4: 02 In-Class Assignment - Vectors

4.0: Introduction

What can you solve with $Ax = b$?

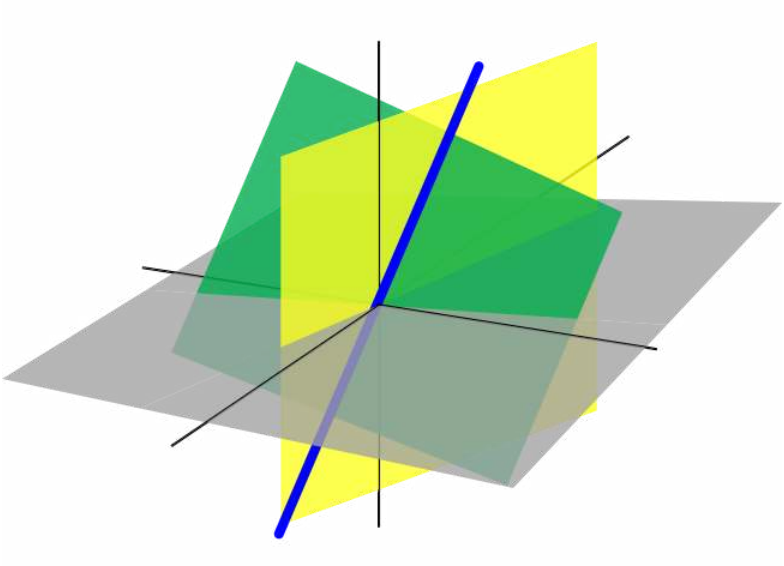


Image from http://Wikipedia.org/.

Vector mathematics are used in physics all of the time. Consider the picture at the beginning of this notebook. While the space shuttle is attached to the Airplane they have the same velocity vector. This vector likely has three components; the velocity in the x direction, y direction and z direction. In fact this velocity is just a combination of multiple components such as thrust from the engines, wind speed and drag.

In this notebook we will be discussing basic vector mathematics and practicing our Python. We will be using the commands in this notebook all semester so make sure you have have some mastery before moving on.

Agenda for today's class (80 minutes)

1. (20 minutes) Example linear System
2. (20 minutes) Review pre-class assignment
3. (20 minutes) Vectors in Python

4.1: Example linear system

Suppose that we have three objects on a balanced beam. Also suppose we know that one has a mass of 2 kg, and we want to find the two unknown masses. Experimentation with a (assume weightless) meter stick produces these two balances. (diagram not to scale)

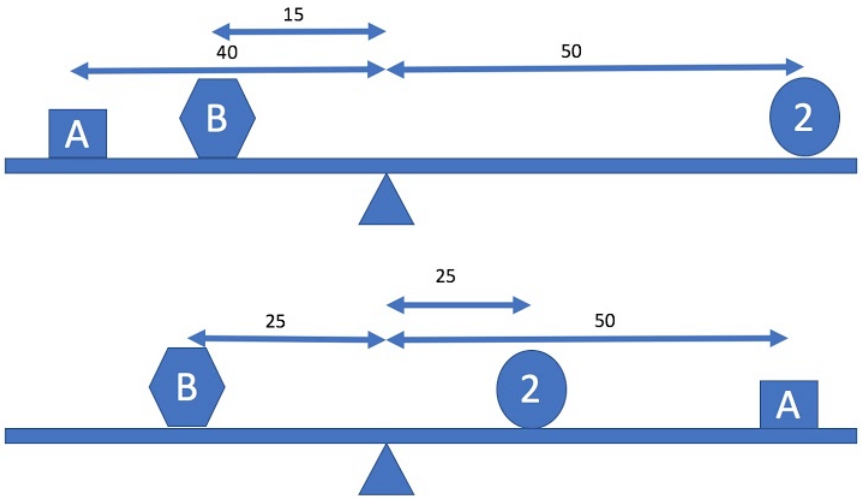


Figure 4.1.1: Image showing two balanced beams, each with three weights. In the top beam is unknown weight A is a distance of 40 to the left of the fulcrum, unknown weight B is a distance of 15 to the left of the fulcrum and a weight of 2 is 50 to the right of the fulcrum. In the bottom beam is the same unknown weights. Weight A is now a distance of 50 to the right of the fulcrum, weight B is a distance of 25 to the left of the fulcrum and the weight of 2 is a distance of 25 to the right of the fulcrum.

For the masses to balance we must have the sum of the moments on the left equal to the sum of the moments on the right, where the moment of an object is its mass times its distance from the balance point. That gives a system of two equations:

$$40A + 15B = 50 \times 2$$

$$25B = 25 \times 2 + 50A$$

? Do This

Find a solution for the above systems of equations and place your solution in the following cell. Make sure you delete the instructional text in the cell first.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Using Python as a calculator, verify that the solution you have found is correct.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Now lets consider a system where we have three unknown masses instead of two. Experimentation with a meter stick produces the two balanced states shown below (diagram not to scale). Write the equations for this system.

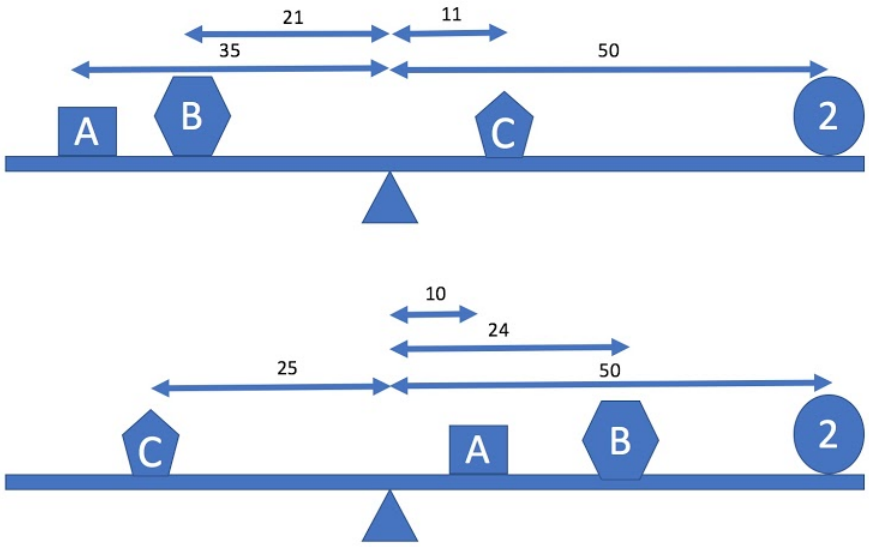


Figure 4.1.2: Image showing two balanced beams, each with four weights. In the top beam is unknown weight A which is a distance of 35 to the left of the fulcrum, unknown weight B is a distance of 21 to the left of the fulcrum, unknown weight C is a distance of 11 to the right of the fulcrum and a weight of 2 is 50 to the right of the fulcrum. In the bottom beam is the same unknown weights. Weight A is now a distance of 10 to the right of the fulcrum, weight B is a distance of 24 to the right of the fulcrum, weight C is a distance of 25 to the left of the fulcrum and the weight of 2 is still at a distance of 50 to the right of the fulcrum.

? Do This

Find a solution to the second set of equations and report the mass for objects A, B and C.

```
# Put your answer to the above question here
```

run | restart | restart & run all

? Do This

Using Python as a calculator, verify that the solution you have found is correct.

```
# Put your answer to the above question here
```

run | restart | restart & run all

4.2: Pre-class assignment review

- 02 Pre-class Assignment - Vectors

Vector Addition Properties

For any vectors x, y, and z of the same size/dimension, we have the following properties:

1. Vector addition is commutative: $x+y=y+x$.
2. Vector addition is associative: $(x+y)+z=x+(y+z)$. We can therefore write both as $x+y+z$.
3. Adding the zero vector to a vector has no effect: $x+0=0+x=x$. (This is an example where the size of the zero vector follows from the context, i.e., its size must be the same as the size of x)
4. $x-x=0$. Subtracting a vector from itself yields the zero vector. (Here too the size of 0 is the size of a.)

Scalar-Vector Multiply Properties

For any vectors x, y, and scalars a, b, we have the following properties

- Scalar-vector multiply is commutative: $ax=x*a$; This means that scalar-vector multiplication can be written in either order.
- Scalar-vector multiply is associative: $(ab)x=a(bx)$
- Scalar-vector multiply is distributive: $a(x+y)=ax+ay$, $(x+y)a=xa+ya$, and $(a+b)x=ax+bx$.

4.3: Vectors in Python

For those who are new to Python, there are many common mistakes happen in this course. Try to fix the following codes.

SyntaxError

It means that the code does not make sense in Python. We would like to define a vector with four numbers.

Do This

Fix the following code to creat three vectors with four numbers.

```
x = [1 2 3.4 4]
y = [1, 2, 3, 5]]
z = [[1, 2, 3, 6.3]
```

run | restart | restart & run all

```
    File "<ipython-input-1-cd07b469c255>", line 1
      x = [1 2 3.4 4]
             ^
  SyntaxError: invalid syntax
```

Although you may have been able to get rid of the error messages the answer to you problem may still not be correct. Throughout the semester we will be using a python program called `answercheck` to allow you to check some of your answers. This program doesn't tell you the right answer but it is intended to be used as a way to get immediate feedback and accelerate learning.

Do This

First we will need to download `answercheck.py` to your current working directory. You only really need to do this once. However, if you delete this file by mistake sometime during the semester, you can come back to this notebook and download it again by running the following cell:

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Do This

How just run the following command to see if you got \(x\), \(y\) and \(z\) correct when you fixed the code above.

```
from answercheck import checkanswer

checkanswer([x,y,z],'e80321644979a873b273aebbbcd0e450');
```

run | restart | restart & run all

Note

make sure you do not change the `checkanswer` commands. The long string with numbers and letters is the secret code that encodes the true answer. This code is also called the HASH. Feel free to look at the `answercheck.py` code and see if you can figure out how it works?

Numpy

Numpy is a common way to represent vectors, and you are suggested to use `numpy` unless otherwise specified. The benefit of `numpy` is that it can perform the linear algebra operations listed in the previous section.

For example, the following code uses `numpy.array` to define a vector of four elements.

```
import numpy as np
x_np = np.array([-1, 0, 2, 3.1])
x_np
```

run | restart | restart & run all

Scalars versus 1-vectors

In mathematics, 1-vector is considered as a scalar. But in Python, they are not the same.

```
x = 2.4
y = [2.4]
x == y
```

run | restart | restart & run all

```
False
```

```
x == y[0]
```

run | restart | restart & run all

```
True
```

Lists of vectors

We have a list of numpy arrays or a list of list. In this case, the vectors can have different dimensions.

? Do This

Modify the print statement using indexing to only print the value 3 from the `list_of_vectors` defined below.

```
x_np = np.array([-1,0, 2 , 3.1])
y_np = np.array([1,-1,3])
z_np = np.array([0,1])
list_of_vectors = [x_np,y_np,z_np]

print(list_of_vectors)
```

run | restart | restart & run all

```
[array([-1. ,  0. ,  2. ,  3.1]), array([ 1, -1,  3]), array([0, 1])]
```

Indexing

The index of a vector runs from 0 to \(n−1\) for a \(n\)-vector.

? Do This

The following code tries to get the third element of `x_np`, which is the number `2.0`. Fix this code to provide the correct answer.

```
print(x_np(3))
```

run | restart | restart & run all

? Do This

Replace only the third element of `x_np` with the number `20.0` such that the new values of `x_np` is [-1, 0, 20., 3.1]

```
# Replace the third element using 20.0, then the resulting element is
#####Start your code here #####

#####End of your code here#####
print(x_np)
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(x_np,'993d5cbc6ddeb10776ed48159780a5d3');
```

run | restart | restart & run all

There is a special index -1, which represents the last element in an array. There are several ways to get more than one consecutive elements.

- x_np[1:3] gives the 2nd and 3rd elements only. It starts with the first index and ends before the second index. So the number of element is just the difference between these two numbers.
- x_np[1:-1] is the same as x_np[1:3] for a 4-vector.
- If you want the last element also, then you do not need to put the index, e.g., x_n[1:] gives all elements except the first one. You can do the same thing as the first one.

? Do This

you are given a vector (`x_np`) of \(n\) elements, define a new vector (`d`) of size \(n−1\) such that \(d_i = x_{i+1} - x_i \) for \(i=1, \dots, n-1 \).
Hint try doing this without writing your own loop. You should be able to use simple `numpy` indexing as described above.

```
x_np = np.array([1,8,3,2,1,9,7])

## Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(d,'14205415f0ed56e608d0a87e7253fa70');
```

run | restart | restart & run all

Assignment versus copying

Take a look at the following code.

- we create one numpy array `x_np`
- we let `y_np = x_np`
- we change the third element of `y_np`
- The third element of `x_np` is also changed

This looks weired and may not make sense for those uses other languages such as MATLAB.

The reason for this is that we are not creating a copy of `x_np` and name it as `y_np`. What we did is that we give a new name `y_np` to the same array `x_np`. Therefore, if one is changed, and the other one is also changed, because they refer to the same array.

```
x_np = np.array([-1, 0, 2, 3.1])
y_np = x_np
y_np[2] = 20.0
x_np
```

run | restart | restart & run all

Do This

There is a method named `copy` that can be used to create a new array. You can search how it works and fix the code below. If this is done correctly the `x_np` vector should stay the same and the `y_np` you now be `[-1 0 2 3.1]`.

```
## modify the following code to copy the x_np instead of just giving it a new name
x_np = np.array([-1, 0, 2, 3.1])
y_np = x_np
y_np[2] = 20.0
print(x_np)
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(x_np,'0ba269d18692155ba252e6addedf37ad');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(y_np,'993d5cbc6ddeb10776ed48159780a5d3');
```

run | restart | restart & run all

Vector equality in numpy and list

The relational operator (`==` , `<` , `>` , `!=` , etc.) can be used to check whether the vectors are same or not. However, they will act differently if the code is comparing `numpy.array` objects or a `list`. In `numpy`, In `numpy` relational operators checks the equality for each element in the `array`. For `list`, relational operators check all elements.

```
x = [-1, 0, 2, 3.1]
y = x.copy()
y[2] = 20.2

x_np = np.array(x)
```

```
y_np = np.array(y)
```
run | restart | restart & run all

```
x == y
```
run | restart | restart & run all

```
False
```

```
np.array(x_np) == np.array(y_np)
```
run | restart | restart & run all

```
array([ True,  True, False,  True])
```

Zero vectors and Ones vectors in numpy

- zeros(n) creates a vector with all 0s
- ones(n) creates a vector with all 1s

Do This

Create a zero vector (called `zero_np`) with the same dimension as vector `x_np`. Create a ones vector (called `ones+np`) also with the same dimension as vector `x_np`.

```
x_np = np.array([-1, 0, 2, 3.1])
### Define zero_np and ones_np here
```
run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer([zero_np, ones_np],'7f874c2e446786655ff96f0bbae8a0c6');
```
run | restart | restart & run all

Random vectors

- random.random(n) creates a random vector with dimension \(n\).

```
random_np = np.random.random(2)
print(random_np)
```
run | restart | restart & run all

```
[0.44542519 0.09046139]
```

Vector addition and subtraction

In this section, you will understand why we use numpy for linear algebra opeartions. If x and y are numpy arrays of the same size, we can have x + y and x-y for their addition and subtraction, respectively.

```
x_np = np.array([1,2,3])
y_np = np.array([100,200,300])

v_sum = x_np + y_np
v_diff = x_np - y_np

print (f'Sum of vectors: {v_sum}')
print (f'Difference of vectors: {v_diff}')
```

run | restart | restart & run all

```
Sum of vectors: [101 202 303]
Difference of vectors: [ -99 -198 -297]
```

For comparison, we also put the addition of two lists below. Recall from the pre-class assignment, we have to define a function to add two lists for linear algebra.

Do This

Modify the following code to properly add and subtract the two lists.
HINT it is perfectly okay NOT to write your own function try you should be able to cast the lists as arrays:

```
x = [1,2,3]
y = [100,200,300]

v_sum = x + y
v_diff = x - y

print (f'Sum of vectors: {v_sum}')
print (f'Difference of vectors: {v_diff}')
```

run | restart | restart & run all

Scalar-vector addition

A scalar-vector addition means that the scalar (or a 1-vector) is added to all elements of the vector.

Do This

Add a scalar 20.20 to all elements of the following vector `x_np` and store teh result back into `x_np`

```
x_np = np.array([1.0,2.0,3.0])
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(x_np,'2f8cbcce405fa12b8608422ff28544bb');
```

run | restart | restart & run all

Scalar-vector multiplication and division

When `a` is a scalar and `x` is `numpy` array. We can express the scalar-vector multiplication as `a*x` or `x*a` .
We can also do scalar-vector division for `x/a` or `a/x` . (note that `x/a` and `a/x` are different)

Do This

Divide all elements of the following vector `x_np` by `20.20` and put it into `y_np`

```
x_np = np.array([1,2,3])
#####Start your code here #####
y_np =
#####End of your code here#####
print(y_np)
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(y_np,'90c1b8639f9d350af1d971d89209a0c6');
```

run | restart | restart & run all

Element-wise operations

As stated above relational operations on `numpy` arrays are performed element-wise. Examples we mentioned before are

- The `==` operator
- The addition `+` and subtraction `-`

Note

for this to work the two vectors have to be the same dimensions.

If they are not have the same dimension, such as a scalar and a vector, we can think about expanding the scalar to have the same dimension as the vector and perform the operations. For example.

- Vector-scalar addition and subtraction
- Vector-scalar multiplication and division

Do This

Assume that you invested three assets with initial values stored in `p_initial`, and after one week, their values are stored in `p_final`. Then what are the asset return ratio (`r`) for these three assets (i.e. price change over the initial value).

```
p_initial = np.array([22.15, 89.32, 56.77])
p_final = np.array([23.05, 87.32, 53.13])
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer(r,'0e231e6cfbef65cf178208cf377af85c');
```

run | restart | restart & run all

Linear combination

We have two vectors x and y we can get the linear combination of these two vectors as $ax+by$ where a and b are scalar coefficients.

In the following example, we are given two vectors (`x_np` and `y_np`), and two scalars (`alpha` and `beta`), we obtain the linear combination `alpha*x_np + beta*y_np` .

```
x_np = np.array([1,2])
```

```
y_np = np.array([3,4])
alpha = 0.5
beta = -0.8
c = alpha*x_np + beta*y_np
print(c)
```

run | restart | restart & run all

We can also define a function `lincomb` to performn the linear combination.

Do This

Finish the following code for lincomb and compare the results we just get.

```
def lincomb(coef, vectors):
    n = len(vectors[0])  # get the dimension of the vectors. note they have to be of t
    comb = np.zeros(n)   # initial the value with all zeros.
    ### Add code here to calculate the linear combination of the input vecotrs and the
    return comb
```

run | restart | restart & run all

```
from answercheck import checkanswer

combination = lincomb([alpha, beta], [x_np,y_np])

checkanswer(combination,'8bab7329c94f3e3cda423add411685c2');
```

run | restart | restart & run all

We can also test the functions ourselves by using values for which we know the answer. For example, the following tests are multiplying and adding by zero we know what these answers should be and can check them.

```
combination = lincomb([0, 0], [x_np,y_np])

combination == np.zeros(combination.shape)
```

run | restart | restart & run all

```
combination = lincomb([2, 2], [combination,np.zeros(combination.shape)])

combination == 2*combination
```

run | restart | restart & run all

If you want to check that all values in a `numpy.array` are the same you could convert it to a list or there is a method called `alltrue` which checks if everything is true. It is a good idea to use this method if vectors get big.

```
combination = lincomb([2, 2], [combination,np.zeros(combination.shape)])

np.alltrue(combination == 2*combination)
```

run | restart | restart & run all

CHAPTER OVERVIEW

5: 03 Pre-Class Assignment - Linear Equations

5.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 1.I, pg 1-2
- Beezer Chapter SLE pg 1-7

Assignment Overview

1. System of Linear Equations
2. Visualizing the problem
3. Multidimentional Spaces
4. Matrixes Notation
5. Assignment wrap-up

5.1: System of Linear Equations

In this course we will spend a lot of time working with systems of linear equations. A linear equation is in the form:

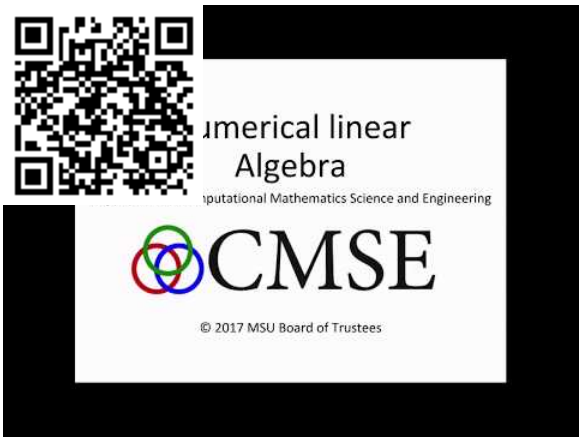
$$a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n = b$$

Where $a_1, a_2, a_3, \ldots a_n$ and b are known constants and $x_1, x_2, x_3, \ldots x_n$ are unknown values. Typically we have systems of equations with different values of as and bs but the unknowns are the same. For example. Consider the example of linear equations in the following video.

TODO

Watch the video and follow along in the notebook.


```
from IPython.display import YouTubeVideo
YouTubeVideo("CH68cc7sH4A",width=640,height=360, cc_load_policy=True)
```



> *Giselle works as a carpenter and as a blacksmith. She earns 20 dollars per hour as a carpenter and 25 dollars per hour as a blacksmith. Last week, Giselle worked both jobs for a total of 30 hours, and earned a total of 690 dollars. How long did Giselle work as a carpenter last week, and how long did she work as a blacksmith?*
>
> - *Brainly.com*

This problems gives us two equations and two unknowns:

$$c+b=30$$

$$20c+25b=690$$

How would we solve this in linear algebra?

$$c+b=30$$

$$20c+25b=690$$

First, we can multiply the first equation by -20 and add to the second equation. This is often called a “linear combination” of the two equations. The operation does not change the answer:

$$-20c-20b=-600$$

$$20c+25b=690$$

$$----$$

$$0c+5b=90$$

This is our new system of equations: $c+b=300c+5b=90$

Now we can easily divide the second equation by 5 and get the value for b:

$$b=90/5=18$$

If we substitute 18 for b into the first equation we get: $c+18=30$

And solving for c gives us c=30−18=12. Let’s check to see if this works by substituting c=18 and c=12 into our original equations:

$$12+18=30$$

$$20(12)+25(18)=690$$

Let’s check the answer using Python:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
b = 18
c = 12
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
c + b == 30
```

```
True
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
20*c + 25*b == 690
```

```
True
```

Question

The above video described three (3) elementary operators that can be applied to a system of linear equations and not change their answer. What are these three operators?

5.2: Visualizing the problem

We can visualize the solution to a system of linear equations in a graph. If we make \(b\) the "\(y\)"-axis and \(c\) the "\(x\)"-axis. For each equation, we calculate the \(b\) value for each \(c\), and two equations give us two lines.

Note

This is sometimes called the “Row Picture.” I will ask you why it has this name in class so think about it.

Question

The above video described three (3) elementary operators that can be applied to a system of linear equations and not change their answer. What are these three operators?

```
from IPython.display import YouTubeVideo
YouTubeVideo("BSxWO6FGib0",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
```

run | restart | restart & run all

```
c = np.linspace(0,20)
c
```

run | restart | restart & run all

```
array([ 0.        ,  0.40816327,  0.81632653,  1.2244898 ,
1.63265306,
        2.04081633,  2.44897959,  2.85714286,  3.26530612,
3.67346939,
        4.08163265,  4.48979592,  4.89795918,  5.30612245,
5.71428571,
        6.12244898,  6.53061224,  6.93877551,  7.34693878,
7.75510204,
        8.16326531,  8.57142857,  8.97959184,  9.3877551 ,
9.79591837,
       10.20408163, 10.6122449 , 11.02040816, 11.42857143,
11.83673469,
       12.24489796, 12.65306122, 13.06122449, 13.46938776,
13.87755102,
       14.28571429, 14.69387755, 15.10204082, 15.51020408,
15.91836735,
       16.32653061, 16.73469388, 17.14285714, 17.55102041,
17.95918367,
       18.36734694, 18.7755102 , 19.18367347, 19.59183673, 20.
])
```

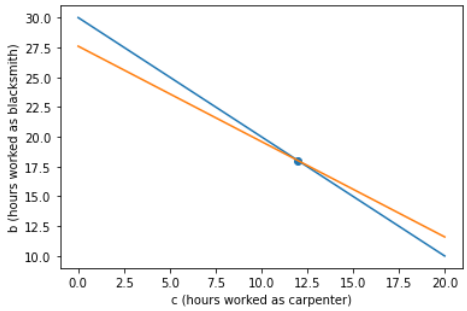
```
b1 = 30-c
b2 = (690-20*c)/25
```

run | restart | restart & run all

Row Picture

```
plt.plot(c,b1)
plt.plot(c,b2)
plt.xlabel('c (hours worked as carpenter)')
plt.ylabel('b (hours worked as blacksmith)')
plt.scatter(12,18);
```

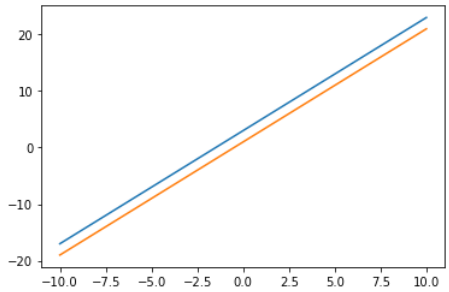
run | restart | restart & run all



Now, consider the next set of equations which do not have a solution
\[-2x+y=3 \nonumber \]
\[-4x+2y=2 \nonumber \]

```
x = np.linspace(-10,10)
y1 =  3+2*x
y2 = (2+4*x)/2
plt.plot(x,y1)
plt.plot(x,y2);
```

run | restart | restart & run all



```
from IPython.display import YouTubeVideo
YouTubeVideo("Z9gkovHDpIQ",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Consider the next set of equations which have infinite many solutions
\[4x-2y=6 \nonumber \]
\[6x-3y=9 \nonumber \]

```
x = np.linspace(-10,10)
y1 =  (4*x-6)/2
y2 = (6*x-9)/3
plt.plot(x,y1)
plt.plot(x,y2)
# 'Run' this cell to view the plot
```

run | restart | restart & run all

```
[<matplotlib.lines.Line2D at 0x7f911c4b3250>]
```

Do This

Plot the following equations from -100 to 100

$$ 18x+21y = 226 \nonumber $$
$$ 72x-3y = 644 \nonumber $$

```
# Put your python code here
```

run | restart | restart & run all

Question

Using the graph, what is a visual estimation of the solution to these two equations? Hint, you may want to change the x range to "zoom" in on the intersection.

Column Picture

I think a good programmer is a lazy person. Let's avoid writing all of the letters in the above euqation by changing it into a column vector format as follows.

$$ \begin{split} c \left[\begin{matrix} 1 \\ 20 \end{matrix} \right] + b \left[\begin{matrix} 1 \\ 5 \end{matrix} \right] = \left[\begin{matrix} 30 \\ 330 \end{matrix} \right] \end{split} \nonumber $$

Notice that this still represents the same system of equations. We just write the constants as column vectors and we only have to write the unknowns once (Since they are the same for all equations).

Let's plot this "column picture", which shows how the above equation is a "linear combination" of the two column vectors.

One way to think about this is we can only move in straight lines in two directions. The first direction is (1,20) and the second is (1,5). The solution to the problem is how far in each direction we need to move to arrive at our final destination of (30,330).

The first column is a vector in the (1,20) direction. The variable c is how far in the (1,20) direction we want to go. Then b is how far in the (1,5) direction we want to go to arrive at the point (30,330).

We will use the `matplotlib` function `arrow` to plot the vectors. The arrow function takes a starting point $[x,y]$ and a direction $[dx,dy]$ as inputs and draws an arrow from the starting point in the direction specified.

First thing to do is plot the first column as a vector. From the origin (0,0) to $c \left[\begin{matrix} 1 \\ 20 \end{matrix} \right]$ or $x=c$ and $y=20c$ with $c=12$

```
c = 12

#hack to inicialize bounds of plot (need this to get the arrows to work?)
plt.plot(0,0)
plt.plot(30,330)

# Plot the first arrow
plt.arrow(0, 0, c*1, c*20,head_width=2, head_length=10, fc='blue', ec='blue')
# 'Run' this cell to view the plot
```

run | restart | restart & run all

```
<matplotlib.patches.FancyArrow at 0x7f911c3969d0>
```

Next thing to do is plot the second column as a vector by adding it to the first. This `arrow` will start at the end of the previous vector and "add" the second column vector:

```
b = 18

#hack to inicialize bounds of plot (need this to get the arrows to work?)
plt.plot(0,0)
```

```
plt.plot(30,330)

# Plot the first arrow
plt.arrow(0, 0, c*1, c*20,head_width=2, head_length=10, fc='blue', ec='blue')

#Plot the second arrow starting at the end of the first
plt.arrow(c, c*20, b*1, b*5, head_width=2, head_length=10, fc='red', ec='red')
# 'Run' this cell to view the plot
```

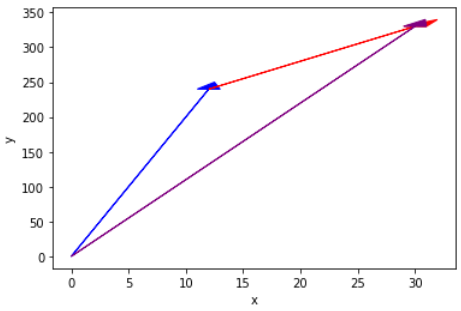
run | restart | restart & run all

```
<matplotlib.patches.FancyArrow at 0x7f911c485550>
```

The takeaway to this figure is that these two column vectors, when added together, end up at the point that represents the right hand side of the above equation (i.e. (30, 330)).

```
#hack to inicialize bounds of plot (need this to get the arrows to work?)
plt.plot(0,0)
plt.plot(30,330)

# Plot the first arrow
plt.arrow(0, 0, c*1, c*20,head_width=2, head_length=10, fc='blue', ec='blue')

#Plot the second arrow starting at the end of the first
plt.arrow(c, c*20, b*1, b*5, head_width=2, head_length=10, fc='red', ec='red')

#Plot a righthand column vector as a point.
plt.arrow(0,0, 30, 330, head_width=2, head_length=10, fc='purple', ec='purple')
plt.xlabel('x');
plt.ylabel('y');
```

run | restart | restart & run all



We say that the two column vectors “span” the \(xy\)-plane. This means that any point on the x,y plane can be represented as a linear combination of the two vectors.

Question

Give an example of two column vectors that do NOT span the \(xy\)−plane

5.3: Multidimentional Spaces

```
from IPython.display import YouTubeVideo
YouTubeVideo("A3fHytkJ010",width=640,height=320, cc_load_policy=True)
```

run | restart | restart & run all

Question

Describe in words, what the solution space would look like for three equations with three unknowns and no solutions.

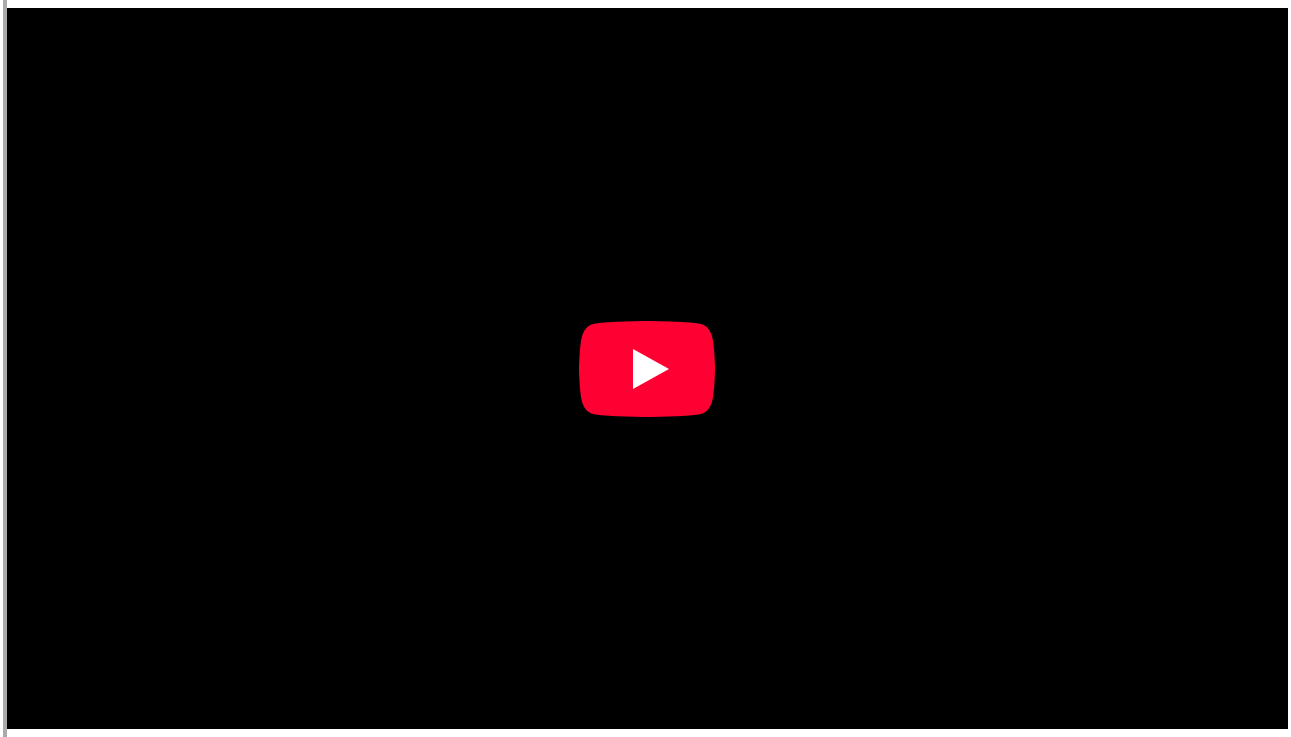
5.4: Matrix Notation

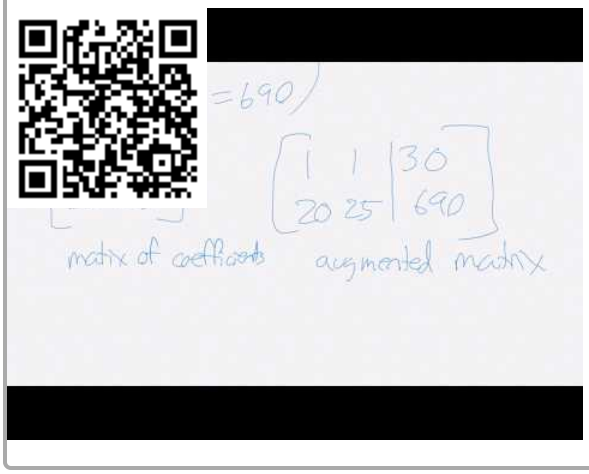
Review *Sections 6.1 - 6.3* of the Stephen Boyd and Lieven Vandenberghe Applied Linear algebra book which introduces the concept of Matrices. Some things to take away include:

- Basic Matrix composition
- Transpose Addition and norms
- Zero and Identy matrix

```
from IPython.display import YouTubeVideo
YouTubeVideo("uC46qAjdE9w",height=360,width=640, cc_load_policy=True)
```

run | restart | restart & run all





A matrix is a rectangular array of numbers typically written between rectangular brackets such as:
\[\begin{split} A = \left[\begin{matrix} 0 & -1 \\ 3 & 4 \\ 0 & 2 \end{matrix} \right]^{ 3\times 2} \end{split} \nonumber \]
The 3×2 subscript is not always included but is handy notation to remember the size of a matrix. The size of a matrix is always written \(m \times n\) where \(m\) is the number of rows and \(n\) is the number of columns. So in the above case Matrix \(A\) is a

3×2 (read “three by two”) matrix.

Question

What is the size of the following matrix?

$$\begin{split} B = \left[\begin{matrix} 0 & -1 & 0 \\ 3 & 4 & 2 \\ \end{matrix} \right] \end{split} \nonumber$$

Each element in a matrix can be referenced by it’s index location. Similar to the size of a matrix the location of an element is described by two numbers, it’s row followed by it’s column. Counting for the rows start at the top and the columns start on the left. For example, in Matrix B element $b_{1,2}$ is the number in row 1 column 2 which is -1.

Question

What is the value of element (2,1) in matrix B?

A linear system of equations can be written in matrix format. For example, the equations in the original example can be written as the following “Augmented matrix”

$$\begin{split}
\left[
\begin{matrix}
1 & 1 \\
20 & 25
\end{matrix}
\, \middle\vert \,
\begin{matrix}
30 \\
690
\end{matrix}
\right]
\end{split} \nonumber$$

And the example which included silversmith can be written as follows:

$$\left[
\begin{matrix}
1 & 1 & 1 \\
50 & 20 & 25 \\
110 & 0 & 20
\end{matrix}
\, \middle\vert \,
\begin{matrix}
30 \\
690 \\
300
\end{matrix}
\right] \nonumber$$

The above equations are represented as “augmented matrices” with the equal side represented as a vertical line.

The general matrix format for a system of linear equations can be written as follows:

$$\begin{split}
X =
\left[
\begin{matrix}
x_{11} & x_{12} & x_{13} & \dots \\
x_{21} & x_{22} & x_{23} & \dots \\
\ldots & \ldots & \ldots & \ddots \\
x_{m1} & x_{m2} & x_{m3} & \dots
\end{matrix}
\end{split}$$

\, \middle\vert \,
\begin{matrix}
x_{1n} \\ x_{2n} \\ \ldots \\ x_{mn}
\end{matrix}
\right] ^{mxn}
\end{split} \nonumber \]
where \(x_{ij}\) is a scalar element in the matrx.
Now consider the following system of linear equations:
\[x_1 = 2.14159 \nonumber \]
\[x_2 = 4 \nonumber \]
\[x_3 = -7.2 \nonumber \]
\[x_4 = 69 \nonumber \]
\[x_5 = 84 \nonumber \]
\[x_6 = 240 \nonumber \]
Lets rewrite this equation as an augmented matrix:
\[\begin{split}
X =
\left[
\begin{matrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{matrix}
\, \middle\vert \,
\begin{matrix}
2.14159 \\ 4 \\ -7.2 \\ 69 \\ 84 \\ 240
\end{matrix}
\right] ^{6x7}
\end{split} \nonumber \]
Notice the submatrix on the left hand side is just the \(I_{6}\) identity matrix and the right hand side are the solutions.

5.5: Assignment wrap-up

Assignment-Specific Question

What are the three elementary operators that can be applied to systems of linear equations that do not change their answer?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

6: 03 In-Class Assignment - Solving Linear Systems of equations

---

6.0: Introduction

In the movie Groundhog day the main character “Phil” repeats the same day over and over again. This is an iterative process where Phil learns from past mistakes until he has a “perfect” day. The Groundhog movie is a fun analogy of iterative methods for solving linear equations. In this class we will write our own iterative method.

Agenda for today’s class (80 minutes)

1. (30 minutes) Review pre-class assignment
2. (30 minutes) Jacobi Method for solving Linear Equations
3. (20 minutes) Numerical Error

6.1: Pre-class assignment review

- 03 Pre-class Assignment: Linear Equations

6.2: Jacobi Method for solving Linear Equations

During class today we will write an iterative method (named after Carl Gustav Jacob Jacobi) to solve the following system of equations:

$$6x + 2y - z = 4$$

$$x + 5y + z = 3$$

$$2x + y + 4z = 27$$

Here is a basic outline of the Jacobi method algorithm:

1. Initialize each of the variables as zero $x_0 = 0, y_0 = 0, z_0 = 0$
2. Calculate the next iteration using the above equations and the values from the previous iterations. For example here is the formula for calculating x_i from $y_{(i-1)}$ and $z_{(i-1)}$ based on the first equation: $x_i = \frac{4 - 2y_{(i-1)} + z_{(i-1)}}{6}$. Similarly, we can obtain the update for y_i and z_i from the second and third equations, respectively.
3. Increment the iteration counter $i = i + 1$ and repeat Step 2.
4. Stop when the answer "converges" or a maximum number of iterations has been reached. (ex. $i = 100$)

Important Note

A sufficient (but not necessary) condition for the method to converge is that the matrix A is strictly or irreducibly diagonally dominant. Strict row diagonal dominance means that for each row, the absolute value of the diagonal term is greater than the sum of absolute values of other terms. - From Wikipedia

In other words, the Jacobi Methid will not work an all problems.

Do This

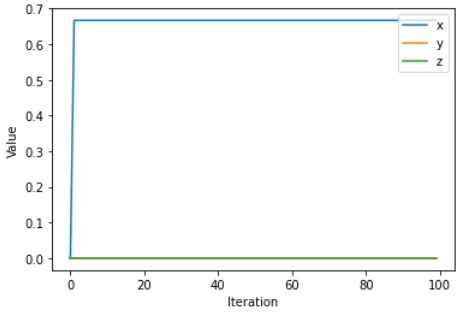
Write out the equations for x_i, y_i, and z_i based on $x_{(i-1)}$, $y_{(i-1)}$, and $z_{(i-1)}$.

Do This

Complete the following code by adding formulas for y_i and z_i to solve the above equations using the Jacobi method.


```
%matplotlib inline
import matplotlib.pylab as plt

x = []
y = []
```

```
z = []

#step 1: inicialize to zero
x.append(0)
y.append(0)
z.append(0)

for i in range(1,100):
    xi = (4 - 2*y[i-1]+ z[i-1])/6
#####Start your code here #####
    yi = 0 #Change this line
    zi = 0 #Change this line
#####End of your code here#####
    #Add latest value to history
    x.append(xi)
    y.append(yi)
    z.append(zi)

#Plot History of values
plt.plot(x, label='x')
plt.plot(y, label='y')
plt.plot(z, label='z')
plt.xlabel('Iteration')
plt.ylabel('Value')
plt.legend(loc=1);
```



Question

What are the final values for x, y, and z?

$$x =$$

$$y =$$

$$z =$$

Do This

Write out each of the above equations and show that your final result is a solution to the system of equations:

```
# Put your code here
```

Question

By inspecting the graph, how long did it take for the algorithum to converge to a solution?

Question

How could you rewrite the above program to stop earlier.

6.3: Numerical Error

Consider the following python statement when answering the questions below:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
0.1 + 0.2 == 0.3
```

```
False
```

Question

Why does Python return **False** even though the above statement is clearly true?

Do This

Let's consider another example. Run the following code which should return true.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import numpy as np
J = np.array([20])
L = [20]

pow(L[0],8) == pow(J[0],8)
```

```
True
```

If you have an older version of `numpy` installed (like 1.18.5) then the results of running the above cell may be false (did anyone get this result?). This is because `numpy` changed how it handles something called “roundoff error”. here is another cell that may help you see better what is going on:


```
import numpy as np
J = np.array([20])
L = [20]
print(pow(20,8))
print(pow(L[0],8))
print(pow(J[0],8))
```

```
25600000000
25600000000
25600000000
```

The older version of `numpy` would return the following:

```
25600000000
25600000000
-169803776
```

We could say to always upgrade to the latest stable version (generally a good idea). But some other libraries that depend on numpy may not be up to date so sometimes python will install an older version to maintain compatibility. For example, one really popular program is tensorflow, which often requires an older version of numpy.

Question

If Python is sometimes wrong, why do we use it?

Question

What are ways you can do to watch out for these types of errors?

Question

Modify the following program to return **True** if the values are within some small number (`e`) of each other.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
def checktrue(a,b,e=0.001):
    return a == b

#Test function
checktrue(0.1+0.2, 0.3)
```

```
False
```

Question

What is a good value to set for `e` and why?

Question

The errors seen in this example seem like they would be fairly common in Python. See if you can find a function in `Numpy` that has the same purpose as `checktrue` :

The class `answercheck` program will take into consideration round off error. For example, the `checkanswer.float` command would consider both of the above correct:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(0.300,'e85b79abfd76b7c13b1334d8d8c194a5');
```

```
Testing 0.3
Answer seems to be correct
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
checkanswer.float(0.1+0.2,'e85b79abfd76b7c13b1334d8d8c194a5')
```

```
Testing 0.3
Answer seems to be correct
```

CHAPTER OVERVIEW

7: 04 Pre-Class Assignment - Python Linear Algebra Packages

7.0: Introduction

Recommended further readings for this pre-class assignment.

- ***Beezer - Section RREF pg 22-44***
- Heffron - Chapter 1.I, pg 2-13

Assignment Overview

1. The Syntax for Systems of Linear Equations
2. Introduction to Gauss Jordan Elimination
3. Gauss Jordan Elimination and the Row Echelon Form
4. Gauss Jordan Practice
5. Assignment wrap up

7.1: The Syntax for Systems of Linear Equations

The following video explains the different syntax we use to describe linear systems.

```
from IPython.display import YouTubeVideo
YouTubeVideo("AQJeOg4ZoIk",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

The following is a summary of the syntax shown in the video:

Linear Equation

\[b = a_{1}x_{1} + a_{2}x_{2} + a_{3}x_{3} + \ldots + a_{n}x_{n} \nonumber \]

System of linear equations

\[b_1 = a_{11}x_1+a_{12}x_2+a_{13}x_3 + \ldots a_{1n} \nonumber \]
\[b_2 = a_{21}x_1+a_{22}x_2+a_{23}x_3 + \ldots a_{2n} \nonumber \]
\[b_3 = a_{31}x_1+a_{32}x_2+a_{33}x_3 + \ldots a_{3n} \nonumber \]
\[\vdots \nonumber \]

\[b_m = a_{m1}x_1+a_{m2}x_2+a_{m3}x_3 + \ldots a_{mn} \nonumber \]

System of linear equations (Matrix format)

\[\begin{split}
\left[
\begin{matrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_m
\end{matrix}
\right]
=
\left[
\begin{matrix}
a_{11} & a_{12} & a_{13} & & a_{1n} \\
a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\
a_{31} & a_{32} & a_{33} & & a_{3n} \\
& \vdots & & \ddots & \vdots \\
a_{m1} & a_{m2} & a_{m3} & & a_{mn}
\end{matrix}
\right]
\left[
\begin{matrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_m
\end{matrix}
\right] \end{split} \nonumber \]

System of linear equations (Augmented Form)

\[\begin{split}
\left[
\begin{matrix}
a_{11} & a_{12} & a_{13} & & a_{1n} \\
a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\
a_{31} & a_{32} & a_{33} & & a_{3n} \\
& \vdots & & \ddots & \vdots \\
a_{m1} & a_{m2} & a_{m3} & & a_{mn}
\end{matrix}
\, \middle\vert \,
\begin{matrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_m
\end{matrix}
\right]
\end{split} \nonumber \]

7.2: Introduction to Gauss Jordan Elimination

The following elementary row operations

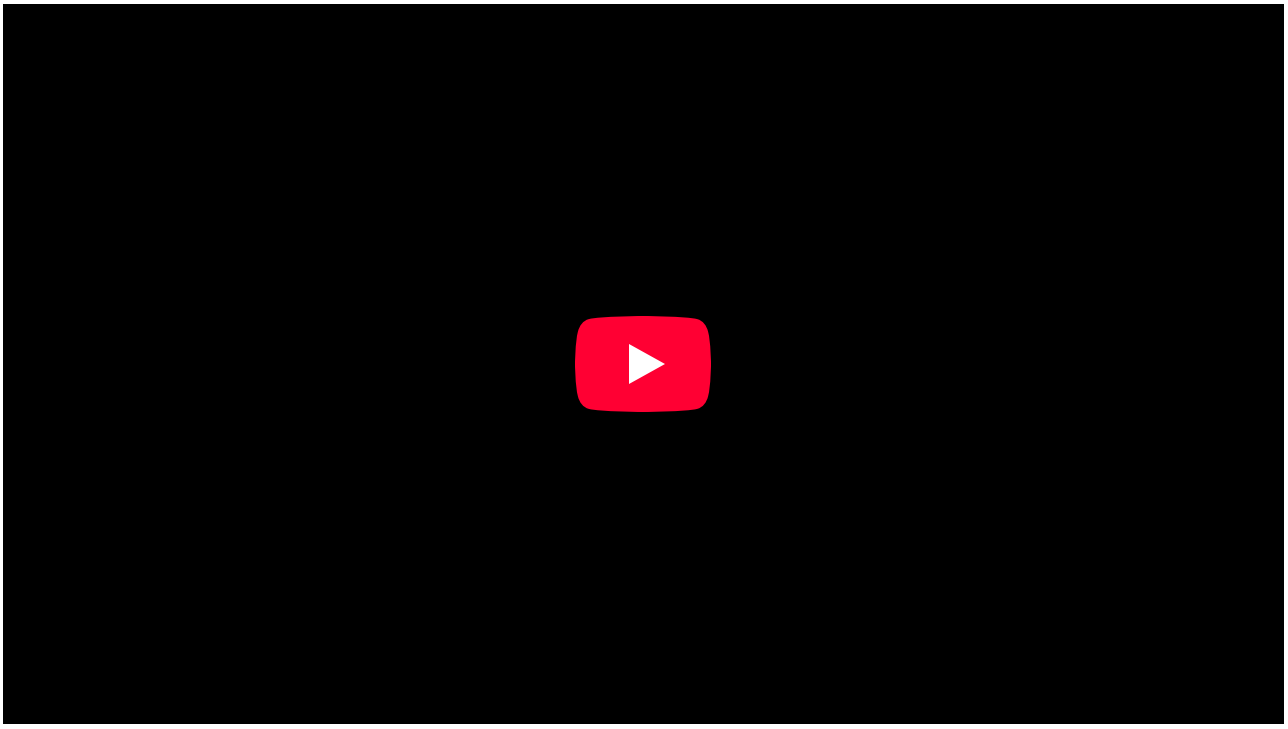
1. Interchange two rows of a matrix
2. Multiply the elements of a row by a nonzero constant
3. Add a multiple of the elements of one row to the corresponding elements of another

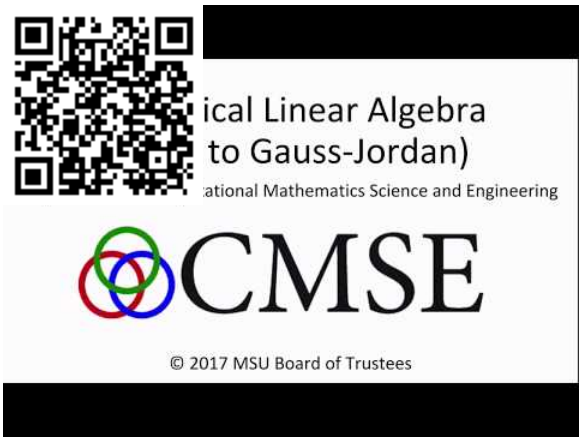
Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("iGmtmF_hm2g",width=640,height=360, cc_load_policy=True)
```





Consider the element $a_{2,1}$ in the following A Matrix.

$$A = \left[\begin{array}{cc|c} 1 & 1 & 30 \\ 20 & 25 & 690 \end{array}\right]$$

Question

Describe an elementary row operation that could be used to make element $a_{(2,1)}$ zero?

Question

What is the new matrix given the above row operation.

Modify the contents of this cell and put your answer to the above question here.

$$A = \left[\begin{array}{cc|c} 1 & 1 & 30 \\ 0 & ?? & ?? \end{array}\right]$$

The following function is a basic implementation of the Gauss-Jorden algorithm to an (m,m+1) augmented matrix:

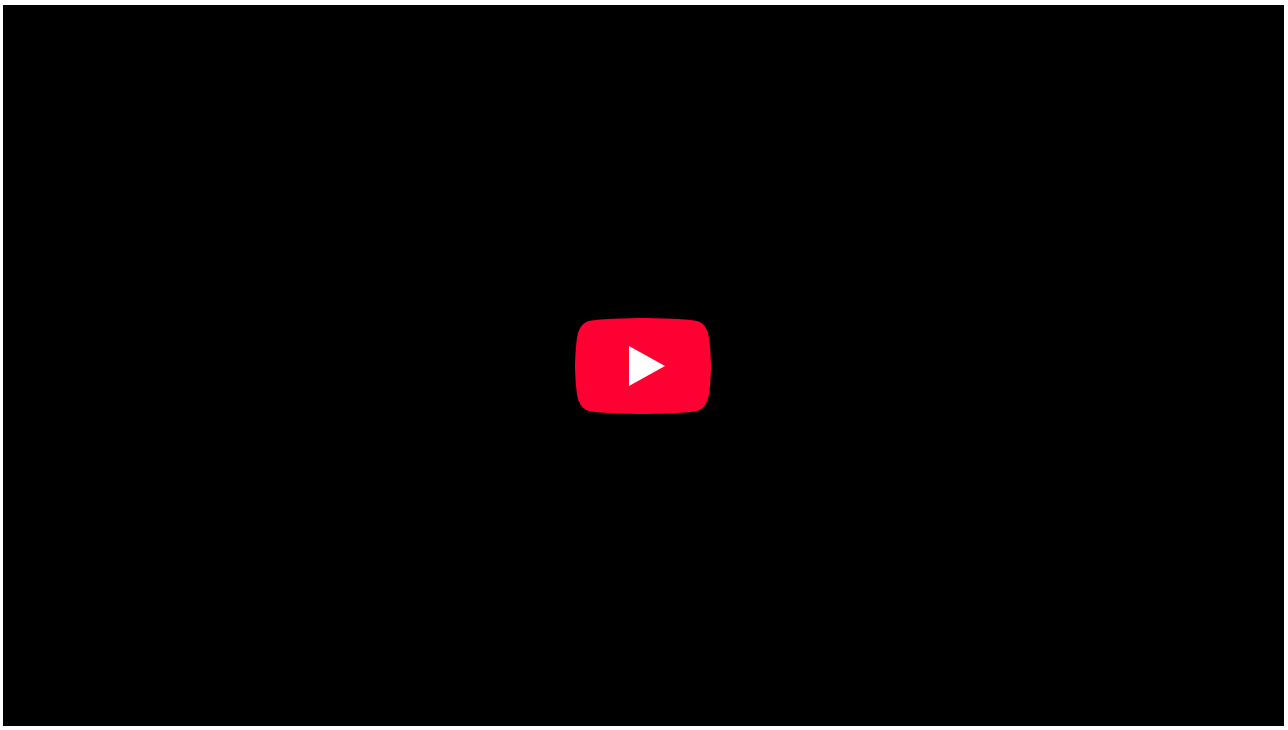
7.3: Gauss Jordan Elimination and the Row Echelon Form

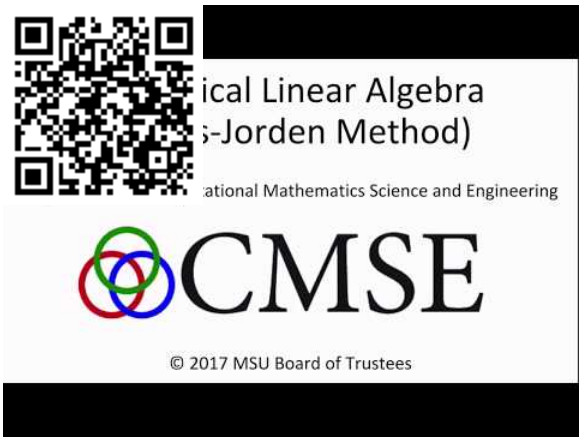
Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("v6RstFsrTJY",width=640,height=360, cc_load_policy=True)
```





The above video left out a special case for Reduced Row Echelon form. There can be non-zero elements in columns that do not have a leading one. For example, All of the following are in Reduced Row Echelon form:

$$\begin{bmatrix} 1 & 2 & 0 & 3 & 0 & 4 \\ 0 & 0 & 1 & 2 & 0 & 7 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 6 \\ 0 & 0 & 0 & 1 & 5 \end{bmatrix}$$

Question

What are the three steps in the Gauss-Jordan Elimination algorithm?

7.4: Gauss Jordan Practice

Do This

Solve the following system of linear equations using the Gauss-Jordan algorithm. Try to do this before watching the video!

$$x_1 + x_3 = 3$$

$$2x_2 - 2x_3 = -4$$

$$x_2 - 2x_3 = 5$$

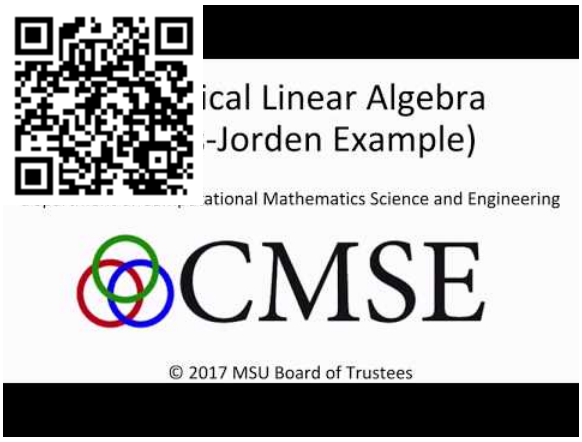
In the following video, we solve the same set of linear equations. Watch the video after trying to do this on your own. It is provided here in case you get stuck.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("xT16yIVw_KE",width=640,height=360, cc_load_policy=True)
```



Question

Something was unclear in the above videos. Describe the difference between a matrix in “row echelon” form and “reduced row echelon” form.

7.5: Assignment wrap up

Assignment-Specific Question

Describe the difference between a matrix in “row echelon” form and “reduced row echelon” form.

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that’s going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

8: 04 In-Class Assignment - Linear Algebra and Python

In order to successfully complete this assignment you need to participate both individually and in groups during class. If you attend class in-person then have one of the instructors check your notebook and sign you out before leaving class. If you are attending asynchronously, turn in your assignment using D2L no later than ***11:59pm on the day of class***. See links at the end of this document for access to the class timeline for your section.

8.0: Introduction

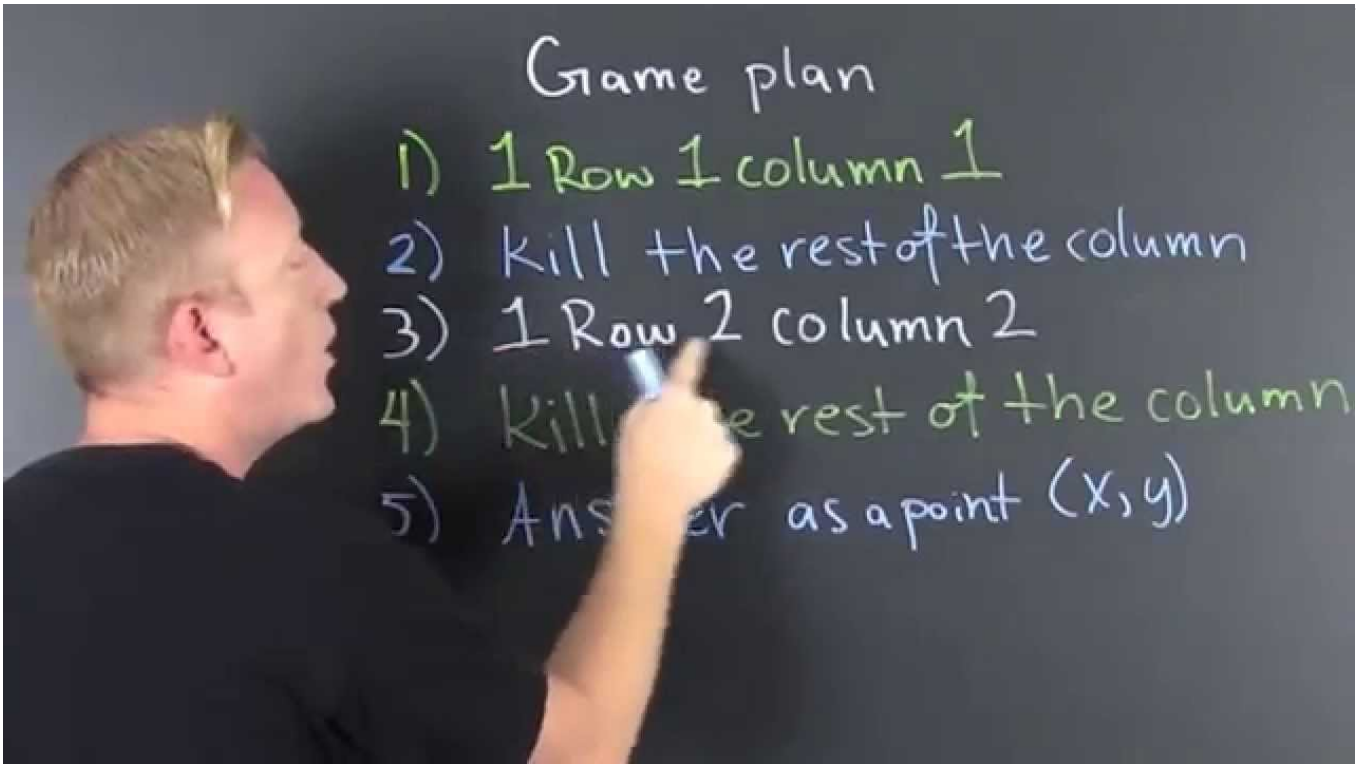


Image from: Mathbyfives @YouTube

Agenda for today's class (80 minutes)

1. (20 minutes) Pre Class Review
2. (20 minutes) Solving Systems of Linear Equations
3. (20 minutes) Underdetermined Systems
4. (20 minutes) Practice Curve Fitting Example

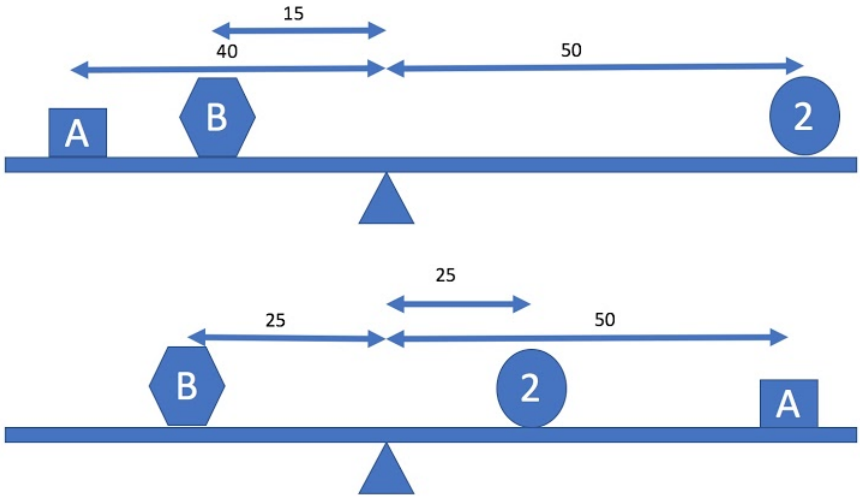
8.1: Pre Class Review

Today we will go over our quiz from last week.

```
# Here are some libraries you may need to use
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
import math
sym.init_printing()
```

8.2: Solving Systems of Linear Equations

Remember the following set of equations from the mass weight example:



$$40A + 15B = 100$$

$$25B = 50 + 50A$$

As you know, the above system of equations can be written as an Augmented Matrix in the following form:

$$\left[\begin{array}{cc|c} 40 & 15 & 100 \\ -50 & 25 & 50 \end{array}\right]$$

Question

Split apart the augmented matrix $[\,A \mid b\,]$ into it's left size (2×2) matrix A and it's right (2×1) matrix b. Define the matrices A and b as `numpy` matrices:

```
#Put your code here
```

run | restart | restart & run all

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A,'f5bfd7c52824d5ac580d0ce1ab98fe68');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(b,'c760cd470439f5db82bb165edf4dc3f8');
```

run | restart | restart & run all

Question

Solve the above system of equations using the `np.linalg.solve` function and store the value in a vector named x:

```
# Put your code to the above question here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.vector(x,'fc02fe6d0577c4452ee70252e1e17654');
```

run | restart | restart & run all

8.3: Underdetermined Systems

Sometimes we have systems of linear equations where we have more unknowns than equations, in this case we call the system "underdetermined." These types of systems can have infinite solutions. i.e., we can not find an unique x such that $Ax = b$. In this case, we can find a set of equations that represent all of the solutions that solve the problem by using Gauss Jordan and the Reduced Row Echelon form. Lets consider the following example:

$$\begin{bmatrix} 5 & -2 & 2 & 1 \\ 4 & -3 & 4 & 2 \\ 4 & -6 & 7 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Question

Define an augmented matrix M that represents the above system of equations:

```
#Put your code here
```

run | restart | restart & run all

Question

What is the Reduced Row Echelon form for A?

```
from answercheck import checkanswer

checkanswer.matrix(M,'efb9b2da0e44984a18f595d7892980e2');
```

run | restart | restart & run all

```
# Put your answer to the above question here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(RREF,'f1fa8baac1df4c378db20cff9e91ca5b');
```

run | restart | restart & run all

Notice how the above RREF form of matrix A is different from what we have seen in the past. In this case not all of our values for x are unique. When we write down a solution to this problem by defining the variables by one or more of the undefined variables. for example, here we can see that x_4 is undefined. So we say $x_4 = x_4$, i.e. x_4 can be any number we want. Then we can define x_3 in terms of x_4. In this case $x_3 = \frac{11}{15} - \frac{4}{15}x_4$. The entire solution can be written out as follows:

$$\begin{aligned} x_1 &= \frac{1}{15} + \frac{1}{15}x_4 \\ x_2 &= \frac{2}{5} + \frac{2}{5}x_4 \\ x_3 &= \frac{11}{15} - \frac{4}{15}x_4 \\ x_4 &= x_4 \end{aligned}$$

Do This

Review the above answer and make sure you understand how we get this answer from the Reduced Row Echelon form from above.

Sometimes, in an effort to make the solution more clear, we introduce new variables (typically, r,s,t) and substitute them in for our undefined variables so the solution would look like the following:

$$\begin{aligned} x_1 &= \frac{1}{15} + \frac{1}{15}r \\ x_2 &= \frac{2}{5} + \frac{2}{5}r \\ x_3 &= \frac{11}{15} - \frac{4}{15}r \\ x_4 &= r \end{aligned}$$

We can find a particular solution to the above problem by inputing any number for r. For example, set r equal to zero and create a vector for all of the x values.

$$\begin{aligned} x_1 &= \frac{1}{15} \\ x_2 &= \frac{2}{5} \\ x_3 &= \frac{11}{15} \\ x_4 &= 0 \end{aligned}$$

```
##here is the same basic math in python
import numpy as np
r = 0
x = np.matrix([1/15+1/15*r, 2/5+2/5*r, 11/15-4/15*r, r]).T
x
```

run | restart | restart & run all

Do This

Define two more matrixes A, b representing the above system of equations $Ax = b$:

```
# put your answer to the above question here.
```

run | restart | restart & run all

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A,'a600d0416a3fb9b4bde87b08caf068f1');
```

run | restart | restart & run all

```
from answercheck import checkanswer
```

```
checkanswer.vector(b,'4cfaa788e4dd6de04fdf6aea4a0e0e71');
```

run | restart | restart & run all

Now let us check our answer by multipying matrix A by our solution x and see if we get b

```
np.allclose(A*x,b)
```

run | restart | restart & run all

Do This

Now go back and pick a different value for r and see that it also produces a valid solution for $Ax = b$.

8.4: Practice Curve Fitting Example

Consider the following polynomial with constant scalars a, b, and c, that falls on the xy-plane:

$$f(x) = ax^2 + bx + c$$

Question

Is this function linear? Why or why not?

Assume that we do not know the values of a, b and c, but we do know that the points (1,2), (-1,12), and (2,3) are on the polynomial. We can substitute the known points into the equation above. For eample, using point (1,2) we get the following equation:

$$2 = a1^2 + b1 + c$$

or

$$2 = a + b + c$$

Question

Generate two more equations by substituting points (-1,12) and (2,3) into the above equation:

Question

If we did this right, we should have three equations and three unknowns (a,b,c). Note also that these equations are linear (how did that happen?). Transform this system of equations into two matrices A and b like we did above.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A,'1896041ded9eebf1cba7e04f32dd1069');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(b,'01e594bb535b4e2f5a04758ff567f918');
```

run | restart | restart & run all

Question

Write the code to solve for x (i.e., (a,b,c)) using `numpy` .

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.vector(x,'1dab22f81c2c156e6adca8ea7ee35dd7');
```

run | restart | restart & run all

Question

Given the value of your `x` matrix derived in the previous question, what are the values for a, b, and c?

```
#Put your answer to the above question here.
a = 0
b = 0
c = 0
```

run | restart | restart & run all

Assuming the above is correct, the following code will print your 2nd order polynomial and plot the original points:

```
import numpy as np
import matplotlib.pylab as plt
x = np.linspace(-3,3)
y = a*x**2 + b*x + c

#plot the function. (Transpose is needed to make the data line up).
plt.plot(x,y.transpose())

#Plot the original points
plt.scatter(1, 2)
plt.scatter(-1, 12)
plt.scatter(2, 3)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
```

run | restart | restart & run all

Question

The following program is intended to take four points as inputs ($p1$, $p2$, $p3$, $p4 \in R^2$) and calculate the coefficients a, b, c, and d so that the graph of $f(x) = ax^3 + bx^2 + cx + d$ passes smoothly through the points. Test the function with the following points (1,2), (-1,6), (2,3), (3,2) as inputs and print the values for a, b, c, and d.

```
def fitPoly3(p1,p2,p3,p4):
    A = np.matrix([[p1[0]**3, p1[0]**2, p1[0], 1],
                   [p2[0]**3, p2[0]**2, p2[0], 1],
                   [p3[0]**3, p3[0]**2, p3[0], 1],
                   [p4[0]**3, p4[0]**2, p4[0], 1]])

    b = np.matrix([p1[1],p2[1],p3[1],p4[1]]).T

    X = np.linalg.solve(A, b)
    a = X[0]
    b = X[1]
    c = X[2]
```

```
    d = X[3]
    #Try to put your figure generation code here for the next question
    #####Start your code here #####


    #####End of your code here#####
    return (a,b,c,d)
```

run | restart | restart & run all

```
#put your answer to the above question here
```

run | restart | restart & run all

Question

Modify the above `fitpoly3` function to also generate a figure of the input points and the resulting polynomial in range `x=(-3,3)` .

```
# Put the answer to the above question above or copy and paste the above function and
```

run | restart | restart & run all

Question

Give any four R^2 input points to `fitPoly3` , is there always a unique solution? Explain your answer.

CHAPTER OVERVIEW

9: 05 Pre-Class Assignment - Gauss-Jordan Elimination

9.0: Introduction

Recommended further readings for this pre-class assignment.

- ***Boyd - Section 1.4-1.5 pg 19-24***
- Beezer - Subsection IP pg 149-152
- Heffron - Chapter 1.II.2 pg 43-47

Goals for today's pre-class assignment

1. Sympy RREF function
2. Calculating Vector Length, Normalization, Distance and Dot
3. Vector spaces in R^n
4. Assignment wrap up

9.1: Sympy RREF function

```
#  Load Useful Python Libraries
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

In class we talked about the Python `sympy` library which has a "reduced row echelon form" (rref) function that runs a much more efficient version of the Gauss-Jordan function. To use the `rref` function you must first convert your matrix into a `sympy.Matrix` and then run the function. For example, lets do this for the following matrix B:

```
B = np.matrix([[ 50, 13, 30 ], [100, 26, 60 ],  [20.5, 25, 650]])
sym.Matrix(B).rref()
# 'Run' this cell to see the output
```

run | restart | restart & run all

This function outputs two values (a matrix and a tuple). For the purposes of this class we only care about the matrix. I generally use the following syntax when using `rref()`

```
sym.Matrix(B).rref()[0]
# 'Run' this cell to see the output
```

run | restart | restart & run all

Question

Although we do not use it often in this course, what does the second output of the `rref` mean (i.e. what does `(0,1)` mean?
hint: read the documentation for `rref` .

How lets consider the multi-week example from a previous assignment, where:
Week 1:
$c+b=30$
$20c+25b=690$
Week 2:
$c+b=35$
$20c+25b=750$
Week 3:
$c+b=30$
$20c+25b=650$

Do This

Write a 2×5 augmented matrix representing the 6 equations above. (you can just copy and paste this from the pre-class if you got it right there), Name your Matrix G to verify your answer using the `checkanswer` function below.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

The following function will apply the rref function to the matrix G and store it in a variable called, wait for it, `rref` :

```
rref,_ = sym.Matrix(G).rref()
rref
```

run | restart | restart & run all

Question

Given the above, How many hours did Giselle work as a capenter for the three weeks and how many hours did she work as a blacksmith. Fill in your answers below to check if you are correct:

```
#Replace the zeros with your answers
carpenter_week1 = 0
carpenter_week2 = 0
carpenter_week3 = 0
blacksmith_week1 = 0
blacksmith_week2 = 0
blacksmith_week3 = 0
```

run | restart | restart & run all

```
from answercheck import checkanswer

hours = [[carpenter_week1, carpenter_week2, carpenter_week3],
         [blacksmith_week1, blacksmith_week2, blacksmith_week3]]
hours = np.matrix(hours).astype('float')

checkanswer.matrix(hours,'b2d4a73cac3c95204f5ed743b507093a');
```

run | restart | restart & run all

9.2: Calculating Vector Length, Normalization, Distance and Dot

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#  Load Useful Python Libraries
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

In this section we will cover some of the basic vector math we will use this semester.

Do This

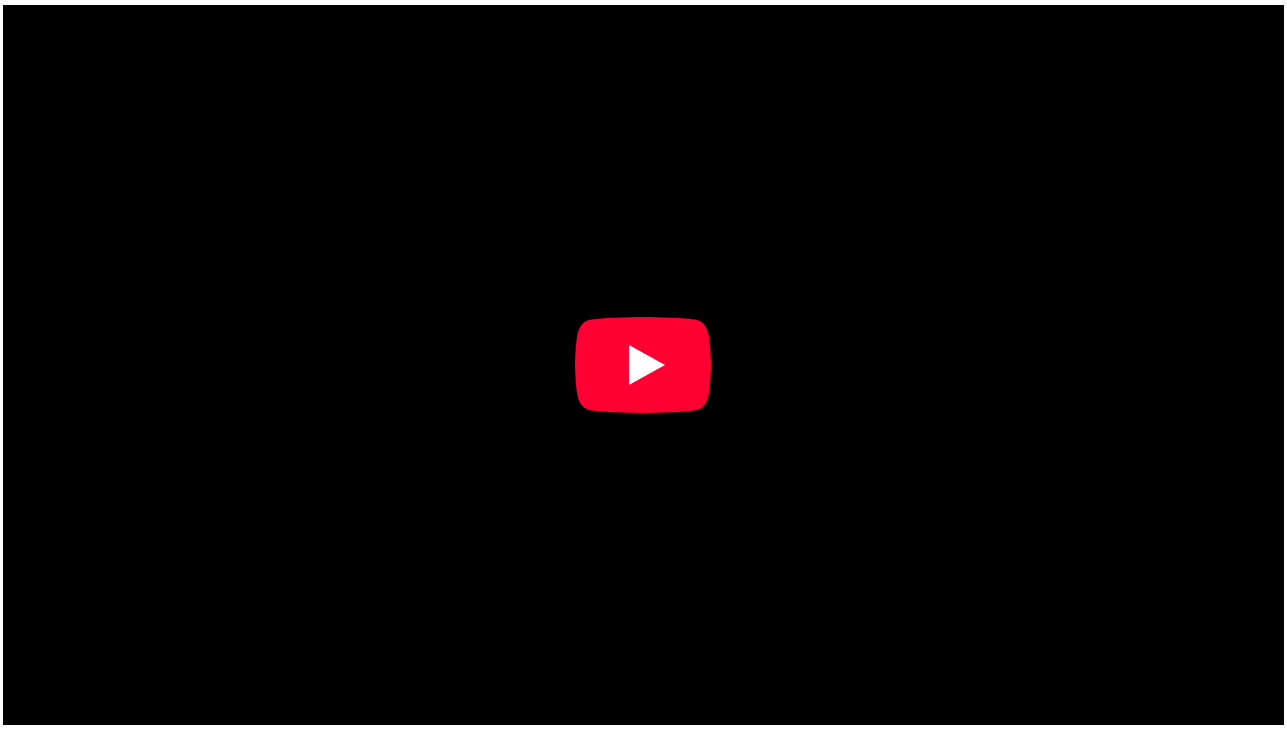
Watch the following summary video about calculation of vector length, Normalizing vectors and the distance between points then answer the questions.

Login with LibreOne to run this code cell interactively.

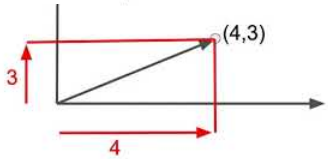
If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("S0BIhbV6reI",width=640,height=360, cc_load_policy=True)
```



$$\sqrt{4^2+3^2}=5$$



Vector:

$$(a_1, a_2, \ldots a_n)$$

$$(b_1, b_2, \ldots b_n)$$

Length:

$$length = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

Normalization:

$$\frac{1}{length}(a_1, a_2, \ldots a_n)$$

Distance:

$$distance = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

Question

Calculate length of vector (4.5, 2.6, 3.3, 4.1)?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(length,'695da96d4a240e54bd8c61e75ff5a3e2');
```

Question

What is a normalized form of the vector (4.5, 2.6, 3.3, 4.1)?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(norm,'12c94f16ba11222987ca20006790182d');
```

Question

What is the distance between (4.5, 2.6, 3.3, 4.1) and (4, 3, 2, 1)?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(distance,'d73defc9a514eb70434190e1757f5bb8');
```

Dot Product:

$$dot(a,b) = a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

Do This

Review ***Sections 1.4 and 1.5 of the Boyd and Vandenberghe*** text and answer the questions below.

Question

What is the dot product between $u = [1, 7, 9, 11]$ and $v = [7, 1, 2, 2]$ (Store the information in a variable called `uv`)?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(uv,'48044bf058c2d7d21b311b173a0ca7e5');
```

Question

What is the norm of vector u defined above (store this value in a variabled called `n`)?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(n,'96078eb552924d7bdb9e67f9ecab88c1');
```

Question

What is the distance between points u and v defined above. (put your answer in a variable named `d`)

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(d,'71f49beeb28061bc60eb3d9966497416');
```

9.3: Vector spaces in R^n

There are two properties that define a vector space these are:

- Closed under addition
- Closed under scalar multiplication

For now we will consider vector spaces in R^n which are just vectors of real numbers (ex: [10,20,3.2], [5,8,32], [8,-0.7], etc) where n is just the length of the vector (ex: 3, 3, and 2 in the earlier example). In the general case a vector does not have to be composed of real numbers but can be almost any type of object as long as it maintains the two above properties, we will get into this concept later in the semester. In the case of real number the above concepts can be described as follows:

- Closed under addition means that if we add any two real vectors vectors (i.e. u, $v \in R^n$) then the result is also in R^n). This is easy to understand if you think about adding any two real vectors there is no way to get a result that is not also a real vector. A way to say this mathematically is as follows:

$$\text{if } u, v \in R^n$$

$$\text{then } u+v \in R^n$$

- Closed under scalar multiplication means that if we have any scalar number ($s \in R$) and we multiply it by a real vector ($v \in R^n$) then the result is also a vector in R^n. Since multiplying a real number by a real number results in a real number this one is also true. Or we can say it as follows:

$$\text{if } s \in R \text{ and } v \in R^n$$

$$\text{then } sv \in R^n$$

The following are some properties of vector addition and multiplication for vectors u and v:

1. $u+v=v+u$ Commutative property
2. $u+(v+w)=(u+v)+w$ Associative property
3. $u+0=0+u=u$ Property of zero vector
4. $u+(-u)=0$ Property of the negative vector
5. $c(u+v)=cu+cv$ Distributive properties
6. $(c+d)u=cu+du$ Distributive Properties
7. $c(du)=(cd)u$ Distributed Properties
8. $1u=u$ Scalar multiplication by 1

Question

Compute the following linear combinations for $u=(1,2)$, $v=(4,-1)$, and $w=(-3,5)$.

(a) $a=u+w$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(a,'af464d466ae982f2cd4461af494e86d6');
```

(b) $a = 2u + v$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(a,'393468eff8c6ba5d27b7d0aa1b18f929');
```

(c) $a = u + 3w$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(a,'d5e5ca43a86501bcde09b1cbc0ba49b5');
```

9.4: Assignment wrap up

Assignment-Specific Question

What is the distance between (4.5, 2.6, 3.3, 4.1) and (4, 3, 2, 1)?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

10: 05 In-Class Assignment - Gauss-Jordan

10.0: Introduction

Today's Agenda (80 minutes)

1. (20 minutes) Pre-class assignment review
2. (20 minutes) Generalize the procedure
3. (20 minutes) Basic Gauss Jordan

10.1: Pre-class assignment review

- 05 Pre-Class Assignment - Gauss-Jordan Elimination

10.2: Generalize the procedure

We are going to think about Gauss-Jordan as an algorithm. First I want you to think about how you would generalize the procedure to work on any matrix. Do the following before moving on to the next section.

Do This

Use the following matrix to think about how you would solve any system of equations using the Gauss-Jordan elimination algorithm. Focus on the steps.

$$\left[\begin{array}{ccc|c} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{array}\right]$$

Question

What are the first three mathematical steps you would do to put the above equation into a reduced row echelon form using Gauss-Jordan method?

Pseudocode

Question

Write down the steps you would complete to implement the Gauss-Jordan elimination algorithm as a computer programer. Some questions to answer:

1. What are the inputs?
2. What are the outputs?
3. How many and what types of loops would you have to guarantee success of your program?

Once you have thought this though the instructor will work with you to build the algorithm.

10.3: Basic Gauss Jordan

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#  Load Useful Python Libraries
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
```

The following is implementation of the Basic Gauss-Jordan Elimination Algorithm for Matrix $A^{m \times n}$ (Pseudocode):

```
for i from 1 to m:
    for j from 1 to m
        if i ≠ j:
            Ratio = A[j,i]/A[i,i]
            #Elementary Row Operation 3
            for k from 1 to n:
                A[j,k] = A[j,k] - Ratio * A[i,k]
            next k
        endif
    next j

    #Elementary Row Operation 2
    Const = A[i,i]
    for k from 1 to n:
        A[i,k] = A[i,k]/Const
next i
```

Do This

using the Pseudocode provided above, write a `basic_gauss_jordan` function which takes a list of lists A as input and returns the modified list of lists:

Login with LibreOne to run this code cell interactively

```
# Put your answer here.
```

Lets check your function by applying the `basic_gauss_jordan` function and check to see if it matches the answer from matrix A in the pre-class video:


```
A = [[1, 1, 1, 2], [2, 3, 1, 3], [0, -2, -3, -8]]
answer = basic_gauss_jordan(A)
sym.Matrix(answer)
```



```
answer_from_video = [[1, 0, 0, -1], [0, 1, 0, 1], [0, 0, 1, 2]]
np.allclose(answer, answer_from_video)
```

The above psuedocode does not quite work properly for all matrices. For example, consider the following augmented matrix:

$$B = \left[\begin{array}{ccc|c} 0 & 1 & 33 & 30 \\ 5 & 3 & 7 & 90 \\ 6 & 69 & 4 & 420 \end{array}\right]$$

Question

Explain why doesn't the provided `basic_gauss_jordan` function work on the matrix B?

Question

Describe how you could modify matrix B so that it would work with `basic_gauss_jordan` AND still give the correct solution?

CHAPTER OVERVIEW

11: 06 Pre-Class Assignment - Matrix Mechanics

11.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 3.IV pg 224-240
- Beezer Chapter M pg 162-206 & EM 340-345
- ***Boyd Section 6.2,-3, 10.1 pg 113-118, 177-183***

Goals for today's pre-class assignment

1. Dot Product Review
2. Matrix Multiply
3. Identity Matrix
4. Elementary Matrices
5. Solving Many Systems (at the same time)
6. Assignment wrap up

11.1: Dot Product Review

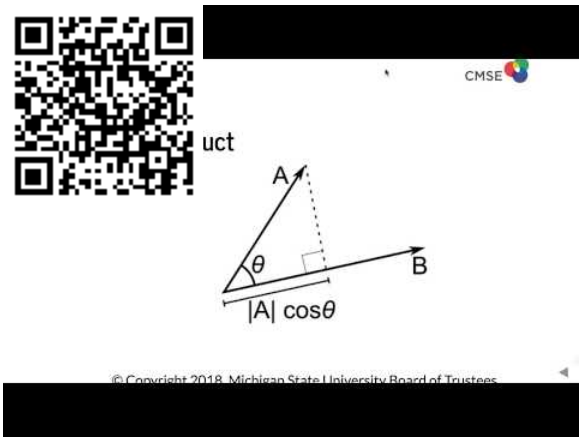
We covered inner products a while ago. This assignment will extend the idea of inner products to matrix multiplication. As a reminder, ***Sections 1.4*** of the Stephen Boyd and Lieven Vandenberghe Applied Linear algebra book covers the dot product. Here is a quick review:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("ZZjWqxKqJwQ",width=640,height=360, cc_load_policy=True)
```



Given two vectors u and v in R^n (i.e. they have the same length), the "dot" product operation multiplies all of the corresponding elements and then adds them together. Ex:

$$u = [u_1, u_2, \ldots, u_n]$$

$$v = [v_1, v_2, \ldots, v_n]$$

$$u \cdot v = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

or:

$$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

This can easily be written as python code as follows:


```
u = [1,2,3]
v = [3,2,1]
solution = 0
for i in range(len(u)):
    solution += u[i]*v[i]

solution
```

```
10
```

In `numpy` the dot product between two vectors can be calculated using the following built in function:

```
import numpy as np
np.dot([1,2,3], [3,2,1])
```

```
10
```

Question

What is the dot product of any vector and the zero vector?

Question

What happens to the `numpy.dot` function if the two input vectors are not the same size?

11.2: Matrix Multiply

Read *Section 10.1* of the Stephen Boyd and Lieven Vandenberghe Applied Linear algebra book which covers Matrix Multiplication. Here is a quick review:

Two matrices A and B can be multiplied together if and only if their "inner dimensions" are the same, i.e. A is $n \times d$ and B is $d \times m$ (note that the columns of A and the rows of B are both d). Multiplication of these two matrices results in a third matrix C with the dimension of $n \times m$. Note that C has the same first dimension as A and the same second dimension as B. i.e $n \times m$.

The (i,j) element in C is the dot product of the ith row of A and the jth column of B.

The ith row of A is:

$$[a_{i1}, a_{i2}, \ldots, a_{id}],$$

and the jth column of B is:

$$\begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{dj} \end{bmatrix}$$

So, the dot product of these two vectors is:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{id}b_{dj}$$

Consider the simple 2×2 example below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw+by & ax+bz \\ cw+dy & cx+dz \end{bmatrix}$$

Let's do an example using `numpy` and show the results using `sympy` :

```
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter
```

run | restart | restart & run all

```
A = np.matrix([[1,1], [2,2]])
sym.Matrix(A)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
B = np.matrix([[3,4], [3,4]])
sym.Matrix(B)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
sym.Matrix(A*B)
# 'Run' this cell to see the output
```

run | restart | restart & run all

Do This

Given two matrices; A and B, show that order matters when doing a matrix multiply. That is, in general, $AB \neq BA$. Show this with an example using two 3×3 matrices and `numpy` .

```
# Put your code here.
```

run | restart | restart & run all

Now consider the following set of linear equations:

$$3x_1 - 3x_2 + 9x_3 = 24$$

$$2x_1 - 2x_2 + 7x_3 = 17$$

$$-x_1 + 2x_2 - 4x_3 = -11$$

We typically write this in the following form:

$$\begin{bmatrix} 3 & -3 & 9 \\ 2 & -2 & 7 \\ -1 & 2 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 17 \\ -11 \end{bmatrix}$$

Notice how doing the matrix multiplication results back into the original system of equations. If we rename the three matrices from above to A, x, and b (note x and b are lowercase because they are column vectors) then we get the main equation for this class, which is:

$$Ax = b$$

Note the information about the equation doesn't change when you change formats. For example, the equation format, the augmented format and the $Ax = b$ format contain the same information. However, we use the different formats for different applications. Consider the `numpy.linalg.solve` function which assumes the format $Ax = b$

```
A = np.matrix([[3, -3,9], [2, -2, 7], [-1, 2, -4]])
sym.Matrix(A)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
b = np.matrix([[24], [17], [-11]])
sym.Matrix(b)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
#Calculate answer to x using numpy
x = np.linalg.solve(A,b)
sym.Matrix(x)
# 'Run' this cell to see the output
```

run | restart | restart & run all

Question

What is the size of the matrix resulting from multiplying a 10×40 matrix with a 40×3 matrix?

11.3: Identity Matrix

Read sections ***Sections 6.2 and 6.3*** of the Stephen Boyd and Lieven Vandenberghe Applied Linear algebra book covers more about matrixes.

An identity matrix is a special square matrix (i.e. $n = m$) that has ones in the diagonal and zeros other places. For example the following is a 3×3 identity matrix:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We always denote the identity matrix with a capital I. Often a subscript is used to denote the value of n. The notations $I_{n \times n}$ and I_n are both acceptable.

An identity matrix is similar to the number 1 for scalar values. I.e. multiplying a square matrix $A_{n \times n}$ by its corresponding identity matrix $I_{n \times n}$ results in itself $A_{n \times n}$.

Do This

Pick a random 3×3 matrix and multiply it by the 3×3 Identity matrix and show you get the same answer.

Question

Consider two square matrices A and B of size $n \times n$. $AB = BA$ is **NOT** true for many A and B. Describe an example where $AB = BA$ is true? Explain why the equality works for your example.

Question

The following matrix is symmetric. What are the values for a, b, and c? (***HINT*** you may want to look online or in the Boyd book for a definition of matrix symmetry)

$$\begin{bmatrix} 3 & 5 & a \\ b & 8 & 4 \\ -3 & c & 3 \end{bmatrix}$$

11.4: Elementary Matrices

```
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

NOTE: A detailed description of elementary matrices can be found here in the *Beezer text Subsection EM 340-345* if you find the following confusing.
There exist a cool set of matrices that can be used to implement Elementary Row Operations. Recall our elementary row operations include:

1. Swap two rows
2. Multiply a row by a constant (c)
3. Multiply a row by a constant (c) and add it to another row.

You can create these elementary matrices by applying the desired elementary row operations to the identity matrix.
If you multiply your matrix from the left using the elementary matrix, you will get the desired operation.
For example, here is the elementary row operation to swap the first and second rows of a 3×3 matrix:

$$E_{12} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
A = np.matrix([[3, -3,9], [2, -2, 7], [-1, 2, -4]])
sym.Matrix(A)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
E1 = np.matrix([[0,1,0], [1,0,0], [0,0,1]])
sym.Matrix(E1)
# 'Run' this cell to see the output
```

run | restart | restart & run all

```
A1 = E1*A
sym.Matrix(A1)
# 'Run' this cell to see the output
```

run | restart | restart & run all

Do This

Give a 3×3 elementary matrix named `E2` that swaps row 3 with row 1 and apply it to the A Matrix. Replace the matrix A with the new matrix.

```
# Put your answer here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(E2,'2c2d2e407389eabeb6d90894565c830f');
```
run | restart | restart & run all

Do This

Give a 3×3 elementary matrix named E3 that multiplies the first row by $c = 3$ and adds it to the third row. Apply the elementary matrix to the A matrix. Replace the matrix A with the new matrix.

```
# Put your answer here.
```
run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(E3,'55ae1f9eb21df00c59dad623b9471506');
```
run | restart | restart & run all

Do This

Give a 3×3 elementary matrix named E4 that multiplies the second row by a constant $c = 1/2$ applies this to matrix A.

```
# Put your answer here.
```
run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(E4,'3a5256840ef907a1b73ebba4471ac26d');
```
run | restart | restart & run all

If the above are correct then we can combine the three operators on the original matrix A as follows.

```
A = np.matrix([[3, -3,9], [2, -2, 7], [-1, 2, -4]])

sym.Matrix(E4*E3*E2*A)
```
run | restart | restart & run all

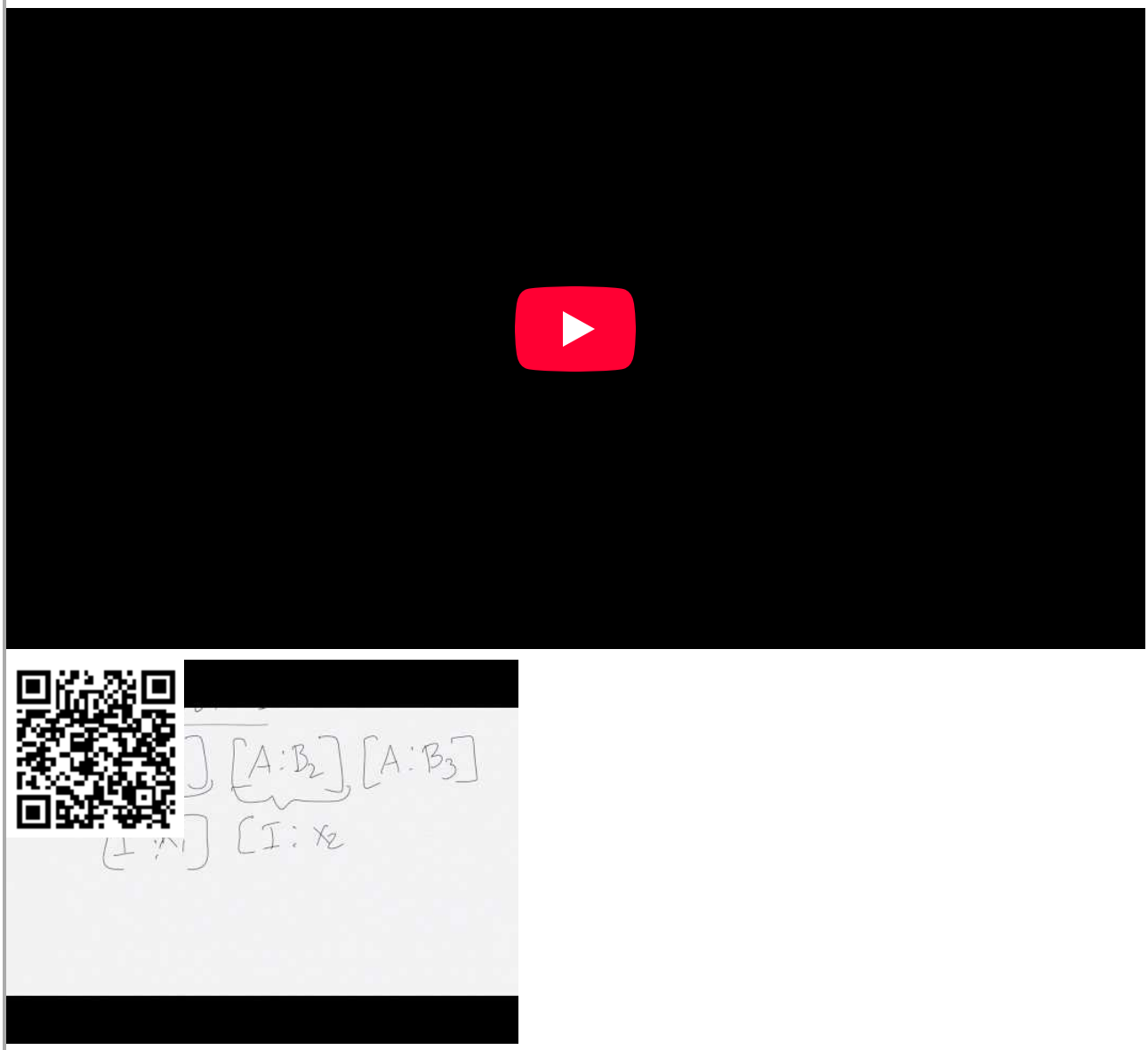
11.5: Solving Many Systems (at the same time)

```
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

```
from IPython.display import YouTubeVideo
YouTubeVideo("k5fdGS5b4OU",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



Consider the Giselle example from above. Her earnings do not change (i.e. she makes $20 per hour as a carpenter and $25 per hour as a blacksmith). However, now she has worked two more weeks. In the second week, she worked for a total of 35 hours and earned $750. In the third week, she worked for a total of 30 hours and earned $650. How much did she work as a carpenter and blacksmith for each of those weeks? In other words:

Week 1:
$c + b = 30$
$20c + 25b = 690$
Week 2:
$c + b = 35$
$20c + 25b = 750$
Week 3:
$c + b = 30$
$20c + 25b = 650$

Do This

Write a 2×5 augmented matrix representing the 6 equations above. Name your Matrix G to verify your answer using the `checkanswer` function below.

```
#Put your answer to the above quation here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(G,'a1e01de142199370be70131849fbf108');
```

run | restart | restart & run all

11.6: Assignment wrap up

Assignment-Specific Question

In the symmetric matrix shown above, what are the values for a, b, and c ?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

12: 06 In-Class Assignment - Matrix Multiply

12.0: Introduction

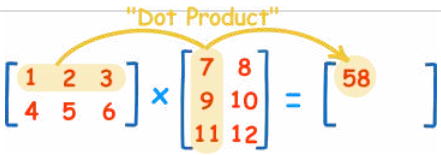


Image from: www.mathisfun.com

Agenda for today's class (80 minutes)

1. (20 minutes) Review of Pre class assignment
2. (30 minutes) Systems of Linear Equations with Many Solutions
3. (30 minutes) Matrix Multiply

12.1: Review of Pre class assignment

- 06 Pre-Class Assignment - Matrix Mechanics

12.2: Systems of Linear Equations with Many Solutions

When we solve a system of equations of the form $Ax = b$, we mentioned that we may have three outcomes:

- a unique solution
- no solution
- infinity many solutions

Assume that we have m equations and n unknowns.

- **Case 1** $m < n$, we do not have enough equations, there will be only **TWO** outcomes: no solution, or infinity many solutions.
- **Case 2** $m = n$, we may have all **THREE** outcomes. If the determinant is nonzero, we have a unique solution, otherwise, we have to decide the outcome based on the augmented matrix.
- **Case 3** $m > n$, we have more equations than the number of unknowns. That means there will be redundant equations (we can remove them) or conflict equations (no solution). We may have all **THREE** outcomes.

We talked about several methods for solving the system of equations. The most general one is the Gauss-Jordan or Gaussian elimination, which works for all three cases. Note that Jacobian and Gauss-Seidel can not work on Case 1 and Case 3.

We will focus on the Gaussian elimination. After the Gaussian elimiation, we look at the last several rows (could be zero) with all zeros except the last column.

If one element from the corresponding right hand side is not zero, we have that 0 equals some nonzero number, which is impossible. Therefore, there is no solution. E.g.,

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 5 \end{array}\right]$$

In this case, we say that the system is *inconsistent*. Later in the semester we will look into methods that try to find a "good enough" solution for an inconsistant system (regression).

Otherwise, we remove all the rows with all zeros (which is the same as removing redundant equations). If the number of remaining equations is the same as the number of unknowns, the rref is an identity matrix, and we have unique solution. E.g.,

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \end{array}\right]$$

If the number of remaining equations is less than the number of unknowns, we have infinitely many solutions. Consider the following three examples:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{array}\right] \Rightarrow x = [2, 3, x_3]^\top$$

where x_3 is a free variable.

$$\left[\begin{array}{ccc|c} 1 & 2 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right] \Rightarrow x = [2 - 2x_2, x_2, 3]$$

where x_2 is a free variable.

$$\left[\begin{array}{cccc|c} 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 & 5 \end{array}\right] \Rightarrow x = [2 - 2x_2 - x_4, x_2, 5 - 3x_4, x_4]$$

where x_2 and x_4 are free variables.

Question

Assume that the system is consistent, explain why the number of equations can not be larger than the number of unknowns after the redundant equations are removed?

Do This

If there are two solutions for $Ax = b$, that is $Ax = b$ and $Ax' = b$ while $x \neq x'$. Check that $A(cx + (1-c)x') = b$ for any real number c. Therefore, if we have two different solutions, we have infinite many solutions.

If $Ax = b$ and $Ax' = b$, then we have $A(x - x') = 0$. If x is a **particular** solution to $Ax = b$, then all the solutions to $Ax = b$ are $\{x + v : v \text{ is a solution to the homogeneous system } Av = 0\}$.

The solution for $Ax = 0$ is always a subspace.

After removing the redundant rows, if the number of equations is the same as the number of unknowns, we have a unique solution. If the difference between the number of equations and the number of unknowns is 1, all the solutions lie on a line. If the difference is 2, all the solutions lie on a 2-D plane.

Question

What is the solution to the following set of linear equations in augmented matrix form?

$$A = \left[\begin{array}{ccc|c} -2 & 4 & 8 & 0 \\ 1 & -2 & 4 & 0 \\ 4 & -8 & 16 & 0 \end{array}\right]$$

12.3: Matrix Multiply

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#some libraries (maybe not all) you will need in this notebook
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)

import random
import time
```

Do This

Write your own matrix multiplication function using the template below and compare it to the built-in matrix multiplication that can be found in `numpy`. Your function should take two "lists of lists" as inputs and return the result as a third list of lists.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
def multiply(m1,m2):
    #first matrix is nxd in size
    #second matrix is dxm in size
    n = len(m1)
    d = len(m2)
```

```
    m = len(m2[0])

    #check to make sure sizes match
    if len(m1[0]) != d:
        print("ERROR - inner dimentions not equal")

    #### put your matrix multiply code here #####

    return result
```

Test your code with the following examples

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Basic test 1
n = 3
d = 2
m = 4

#generate two random lists of lists.
matrix1 = [[random.random() for i in range(d)] for j in range(n)]
matrix2 = [[random.random() for i in range(m)] for j in range(d)]
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter
```

```
sym.Matrix(matrix1) # Show matrix using sympy
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
sym.Matrix(matrix2) # Show matrix using sympy
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Compare to numpy result
np_x = np.matrix(matrix1)*np.matrix(matrix2)

#use allclose function to see if they are numrically "close enough"
print(np.allclose(x, np_x))

#Result should be True
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Test identity matrix
n = 4

# Make a Ransom Matrix
matrix1 = [[random.random() for i in range(n)] for j in range(n)]
sym.Matrix(matrix1) # Show matrix using sympy
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#generate a 3x3 identity matrix
matrix2 = [[0 for i in range(n)] for j in range(n)]
for i in range(n):
    matrix2[i][i] = 1
sym.Matrix(matrix2) # Show matrix using sympy
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
result = multiply(matrix1, matrix2)

#Verify results are the same as the original
np.allclose(matrix1, result)
```

Timing Study

In this part, you will compare your matrix multiplication with the `numpy` matrix multiplication. You will multiply two randomly generated $n \times n$ matrices using both the `multiply()` function defined above and the `numpy` matrix multiplication. Here is the basic structure of your timing study:

1. Initialize two empty lists called `my_time` and `numpy_time`
2. Loop over values of n (100, 200, 300, 400, 500)
3. For each value of n use the time.clock() function to calculate the time it takes to use your algorithm and append that time (in seconds) to the `my_time` list.
4. For each value of n use the time.clock() function to calculate the time it takes to use the `numpy` matrix multiplication and append that time (in seconds) to the `numpy_time` list.
5. Use the provided code to generate a scatter plot of your results.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
n_list = [100, 200, 300, 400, 500]
my_time = []
numpy_time = []
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# RUN AT YOUR OWN RISK.
# THIS MAY TAKE A WHILE!!!!

for n in n_list:
    print(f"Measureing time it takes to multiply matrixes of size {n}")
    #Generate random nxn array of two lists
    matrix1 = [[random.random() for i in range(n)] for j in range(n)]
    matrix2 = [[random.random() for i in range(n)] for j in range(n)]
    start = time.time()
    x = multiply(matrix1, matrix2)
    stop = time.time()
    my_time.append(stop - start)
```

```
    #Convert the lists to a numpy matrix
    npm1 = np.matrix(matrix1)
    npm2 = np.matrix(matrix2)

    #Calculate the time it takes to run the numpy matrix.
    start = time.time()
    answer = npm1*npm2
    stop = time.time()
    numpy_time.append(stop - start)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
plt.scatter(n_list,my_time, color='red', label = 'my time')
plt.scatter(n_list,numpy_time, color='green', label='numpy time')

plt.xlabel('Size of $n x n$ matrix');
plt.ylabel('time (seconds)')
plt.legend();
```

Based on the above results, you can see that the `numpy` algorithm not only is faster but also "scales" at a slower rate than your algorithm.

Question

Why do you think the `numpy` matrix multiplication is so much faster?

CHAPTER OVERVIEW

13: 07 Pre-Class Assignment - Transformation Matrix

13.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 3.IV pg 241-243
- Beezer Subsection EM pg 340-345
- Boyd A little on page 7

Assignment Overview

1. The Inverse Matrix
2. Transformation Matrix
3. Assignment wrap-up

13.1: The Inverse Matrix (aka A^-1)

For some (not all) **square** matrices A, there exists a special matrix called the Inverse Matrix, which is typically written as A^{-1} and when multiplied by A results in the identity matrix I:

$$A^{-1}A = AA^{-1} = I$$

Some properties of an Inverse Matrix include:

1. $(A^{-1})^{-1} = A$
2. $(cA)^{-1} = \frac{1}{c}A^{-1}$
3. $(AB)^{-1} = B^{-1}A^{-1}$
4. $(A^n)^{-1} = (A^{-1})^n$
5. $(A^\top)^{-1} = (A^{-1})^\top$ here $A^\top$ is the tranpose of the matrix A.

If you know that A^{-1} is an inverse matrix to A, then solving $Ax = b$ is simple, just multiply both sides of the equation by A^{-1} and you get:

$$A^{-1}Ax = A^{-1}b$$

If we apply the definition of the inverse matrix from above we can reduce the equation to:

$$Ix = A^{-1}b$$

We know I times x is just x (definition of the identity matrix), so this further reduces to:

$$x = A^{-1}b$$

To conclude, solving $Ax = b$ when you know A^{-1} is really simple. All you need to do is multiply A^{-1} by b and you know x.

Do This

Find a Python numpy command that will calculate the inverse of a matrix and use it invert the following matrix `A` . Store the inverse in a new matirx named `A_inv`

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter

A = np.matrix([[1, 2, 3], [4, 5, 6], [7,8,7]])

sym.Matrix(A)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your answer to the above question here.
```

Lets check your answer by multiplying `A` by `A_inv` .

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
A * A_inv
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
np.allclose(A*A_inv, [[1,0,0],[0,1,0],[0,0,1]])
```

Question

What function did you use to find the inverse of matrix A?

How do we create an inverse matrix?

From previous assignments, we learned that we could string together a bunch of Elementary Row Operations to get matrix (A) into it's Reduced Row Echelon form. We now know that we can represent Elementary Row Operations as a sequence of Elementaary Matrices as follows:

$$E_n \dots E_3 E_2 E_1 A = RREF$$

If A reduces to the identity matrix (i.e. A is row equivalent to I), then A has an inverse and its inverse is just all of the Elementary Matrices multiplied together:

$$A^{-1} = E_n \dots E_3 E_2 E_1$$

Consider the following matrix.

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 6 \end{bmatrix}$$


```
A = np.matrix([[1, 2], [4,6]])
```

It can be reduced into an identity matrix using the following elementary operators

Words	Elementary Matrix
Adding -4 times row 1 to row 2.	$E_1 = \begin{bmatrix} 1 & 0 \\ -4 & 1 \end{bmatrix}$
Adding row 2 to row 1.	$E_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$
Multiplying row 2 by $-\frac{1}{2}$.	$E_3 = \begin{bmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{bmatrix}$

Login

```
E1 = np.matrix([[1,0], [-4,1]])
E2 = np.matrix([[1,1], [0,1]])
E3 = np.matrix([[1,0],[0,-1/2]])
```

We can just check that the statment seems to be true by multiplying everything out.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
E3*E2*E1*A
```

```
matrix([[1., 0.],
        [0., 1.]])
```

Do This

Combine the above elementary Matrices to make an inverse matrix named `A_inv`

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer to the above question here.
```

Do This

Verify that `A_inv` is an actual inverse and chech that $AA^{-1} = I$.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your code here.
```

Question

Is an invertible matrix is always square? Why or why not?

Question

Is a square matrix always invertible? Why or why not?

Question

Describe the Reduced Row Echelon form of a square, invertible matrix.

Question

Is the following matrix in the Reduced Row Echelon form?

$$\begin{bmatrix} 1 & 2 & 0 & 3 & 0 & 4 \\ 0 & 0 & 1 & 3 & 0 & 7 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Question

If the matrix shown above is not in Reduced Row Echelon form. Name a rule that is violated?

Question

What is the size of the matrix described in the previous QUESTION?

- 4×6
- 6×4
- 3×6
- 5×3

Question

Describe the elementary row operation that is implemented by the following matrix

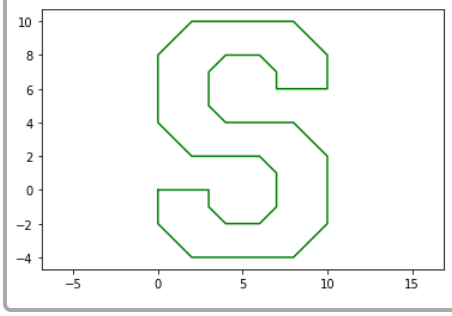
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

13.2: Transformation Matrix

Consider the following set of points:

```
%matplotlib inline
import matplotlib.pylab as plt

x = [0.0,  0.0,  2.0,  8.0, 10.0, 10.0, 8.0, 4.0, 3.0, 3.0, 4.0, 6.0, 7.0, 7.0, 10.0,
     10.0,  8.0,  2.0, 0.0, 0.0, 2.0, 6.0, 7.0,  7.0,  6.0,  4.0,  3.0, 3.0, 0.0]
y = [0.0, -2.0, -4.0, -4.0, -2.0,  2.0, 4.0, 4.0, 5.0, 7.0, 8.0, 8.0, 7.0, 6.0,  6.0,
     8.0, 10.0, 10.0, 8.0, 4.0, 2.0, 2.0, 1.0, -1.0, -2.0, -2.0, -1.0, 0.0, 0.0]

plt.plot(x,y, color='green');
plt.axis('equal');
```

run | restart | restart & run all



We can rotate these points around the origin by using the following simple set of equations:

\[x \cos(\theta) - y \sin(\theta) = x_{rotated} \nonumber \]

\[x \sin(\theta) + y \cos(\theta) = y_{rotated} \nonumber \]

This can be rewritten as the following system of matrices:

\[\begin{split}
\left[
\begin{matrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{matrix}
\right]
\left[
\begin{matrix}
x \\
y
\end{matrix}
\right]
=
\left[
\begin{matrix}
x_{rotated}\\
y_{rotated}
\end{matrix}
\right]
\end{split} \nonumber \]

We can rotate the points around the origin by \(\pi/4\) (i.e. \(45^o\)) as follows:

```
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter

points = np.matrix([x,y])
```

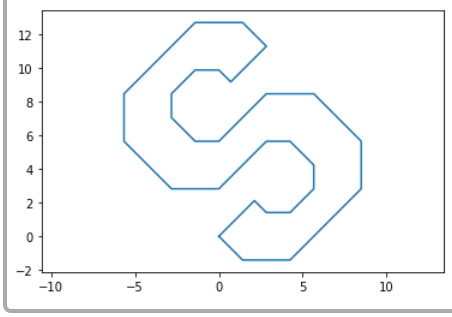
run | restart | restart & run all

```
angle = np.pi/4
R = np.matrix([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]]);
sym.Matrix(R)
```

run | restart | restart & run all

```
p=R*points

plt.plot(p[0].T,p[1].T);
plt.axis('equal');

#print(p[0].T)
```

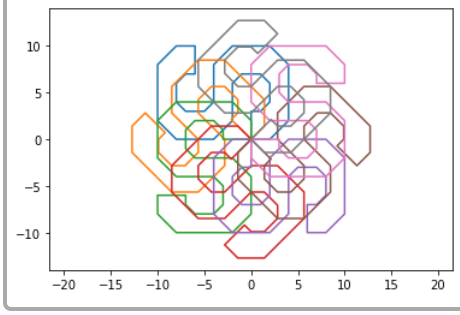
run | restart | restart & run all



We can even have a little fun and keep applying the same rotation over and over again.

```
# Apply R and plot 8 times
for i in range(0,8):
    p = R * p
    plt.plot(p[0].T,p[1].T);

plt.axis('equal');
```

run | restart | restart & run all



Question

In the above code what does the `T` call in `p[0].T` do?

13.3: Assignment wrap-up

Assignment-Specific Question

If the matrix shown above is not in Reduced Row Echelon form. Name a rule that is violated?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

14: 07 In-Class Assignment - Transformations

14.0: Introduction

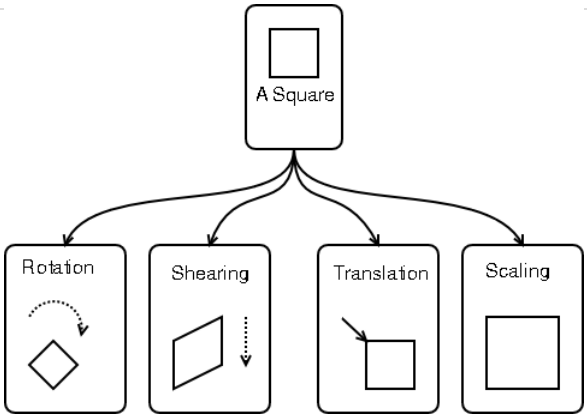


Image from: https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html

Agenda for today's class (80 minutes)

1. (20 minutes) Review of Pre-Class Assignment
2. (20 minutes) Affine Transforms
3. (20 minutes) Fractals

14.1: Review of Pre-Class Assignment

- 07 Pre-Class Assignment - Transformation Matrix

14.2: Affine Transforms

In this section, we are going to explore different types of transformation matrices. The following code is designed to demonstrate the properties of some different transformation matrices.

Do This

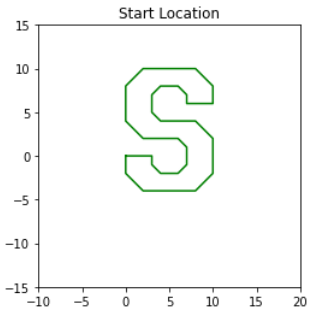
Review the following code.

```
#Some python packages we will be using
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
from mpl_toolkits.mplot3d import Axes3D #Lets us make 3D plots
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter
```

run | restart | restart & run all

```
# Define some points
x = [0.0,  0.0,  2.0,  8.0, 10.0, 10.0, 8.0, 4.0, 3.0, 3.0, 4.0, 6.0, 7.0, 7.0, 10.0,
     10.0,  8.0,  2.0, 0.0, 0.0, 2.0, 6.0, 7.0,  7.0,  6.0,  4.0,  3.0, 3.0, 0.0]
y = [0.0, -2.0, -4.0, -4.0, -2.0,  2.0, 4.0, 4.0, 5.0, 7.0, 8.0, 8.0, 7.0, 6.0,  6.0,
     8.0, 10.0, 10.0, 8.0, 4.0, 2.0, 2.0, 1.0, -1.0, -2.0, -2.0, -1.0, 0.0, 0.0]
con = [ 1.0 for i in range(len(x))]

p = np.matrix([x,y,con])


mp = p.copy()

#Plot Points
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green');
plt.axis('scaled');
plt.axis([-10,20,-15,15]);
plt.title('Start Location');
```

run | restart | restart & run all



Example Scaling Matrix

```
#Example Scaling Matrix
```

```
#Define Matrix
scale = 0.5  #The amount that coordinates are scaled.
S = np.matrix([[scale,0,0], [0,scale,0], [0,0,1]])

#Apply matrix

mp = p.copy()
mp = S*mp

#Plot points after transform
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green')
plt.axis('scaled')
plt.axis([-10,20,-15,15])
plt.title('After Scaling')

#Uncomment the next line if you want to see the original.
# plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0], color='blue',alpha=0.3);

sym.Matrix(S)
```

run | restart | restart & run all

Example Translation Matrix

```
#Example Translation Matrix

#Define Matrix
dx = 1  #The amount shifted in the x-direction
dy = 1  #The amount shifted in the y-direction
T = np.matrix([[1,0,dx], [0,1,dy], [0,0,1]])

#Apply matrix

mp = p.copy()

mp = T*mp

#Plot points after transform
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green')
plt.axis('scaled')
plt.axis([-10,20,-15,15])
plt.title('After Translation')

#Uncomment the next line if you want to see the original.
# plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0], color='blue',alpha=0.3);

sym.Matrix(T)
```

run | restart | restart & run all

Example Reflection Matrix

```
#Example Reflection Matrix
```

```
#Define Matrix
Re = np.matrix([[1,0,0],[0,-1,0],[0,0,1]]) ## Makes all y-values opposite so it reflec

#Apply matrix

mp = p.copy()

mp = Re*mp

#Plot points after transform
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green')
plt.axis('scaled')
plt.axis([-10,20,-15,15])

#Uncomment the next line if you want to see the original.
# plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0], color='blue',alpha=0.3);

sym.Matrix(Re)
```

run | restart | restart & run all

Example Rotation Matrix

```
#Example Rotation Matrix

#Define Matrix
degrees = 30
theta = degrees * np.pi / 180  ##Make sure to always convert from degrees to radians.

# Rotates the points 30 degrees counterclockwise.
R = np.matrix([[np.cos(theta),-np.sin(theta),0],[np.sin(theta), np.cos(theta),0],[0,0

#Apply matrix
mp = p.copy()

mp = R*mp

#Plot points after transform
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green')
plt.axis('scaled')
plt.axis([-10,20,-15,15])

#Uncomment the next line if you want to see the original.
# plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0], color='blue',alpha=0.3);

sym.Matrix(R)
```

run | restart | restart & run all

Example Shear Matrix

Combine Transforms

We have five transforms R, S, T, Re, and SH

Do This

Construct a (\(3 \times 3\)) transformation Matrix (called \(M\)) which combines these five transforms into a single matrix. You can choose different orders for these five matrix, then compare your result with other students.

```
#Put your code here
```

run | restart | restart & run all

```
#Plot combined transformed points
mp = p.copy()
mp = M*mp
plt.plot(mp[0,:].tolist()[0],mp[1,:].tolist()[0], color='green');
plt.axis('scaled');
plt.axis([-10,20,-15,15]);
plt.title('Start Location');
```

run | restart | restart & run all

Questions

Did you can get the same result with others? You can compare the matrix \(M\) to see the difference. If not, can you explain why it happens?

Interactive Example

```
from ipywidgets import interact,interact_manual

def affine_image(angle=0,scale=1.0,dx=0,dy=0, shx=0, shy=0):
    theta = -angle/180  * np.pi

    plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0], color='green')

    S = np.matrix([[scale,0,0], [0,scale,0], [0,0,1]])
    SH = np.matrix([[1,shx,0], [shy,1,0], [0,0,1]])
    T = np.matrix([[1,0,dx], [0,1,dy], [0,0,1]])
    R = np.matrix([[np.cos(theta),-np.sin(theta),0],[np.sin(theta), np.cos(theta),0],

    #Full Transform
    FT = T*SH*R*S;
    #Apply Transforms
    p2 =  FT*p;

    #Plot Output
    plt.plot(p2[0,:].tolist()[0],p2[1,:].tolist()[0], color='black')
    plt.axis('scaled')
    plt.axis([-10,20,-15,15])
    return sym.Matrix(FT)
```

run | restart | restart & run all

```
##TODO: Modify this line of code
interact(affine_image, angle=(-180,180), scale_manual=(0.01,2),
         dx=(-5,15,0.5), dy=(-15,15,0.5), shx = (-1,1,0.1), shy = (-1,1,0.1));
```

run | restart | restart & run all

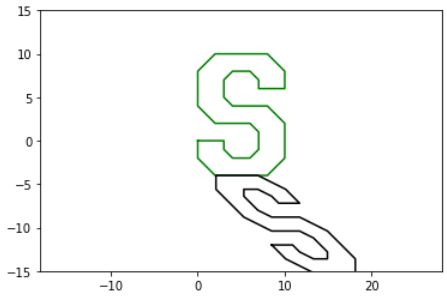
The following command can also be used but it may be slow on some peoples computers.

```
##TODO: Modify this line of code
#interact(affine_image, angle=(-180,180), scale=(0.01,2),
#          dx=(-5,15,0.5), dy=(-15,15,0.5), shx = (-1,1,0.1), shy = (-1,1,0.1));
```

run | restart | restart & run all

Do This

Using the above interactive enviornment to see if you can figure out the transformation matrix to make the following image:



Question

What where the input values?

14.3: Fractals

In this section we are going to explore using transformations to generate fractals. Consider the following set of linear equations. Each one takes a 2D point as input, applies a 2×2 transform, and then also translates by a 2×1 translation matrix

$$T_1: \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.86 & 0.03 \\ -0.03 & 0.86 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} : probability = 0.83$$

$$T_2: \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.2 & -0.25 \\ 0.21 & 0.23 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} : probability = 0.08$$

$$T_3: \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.15 & 0.27 \\ 0.25 & 0.26 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.45 \end{bmatrix} : probability = 0.08$$

$$T_4: \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0.17 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} : probability = 0.01$$

We want to write a program that use the above transformations to "randomly" generate an image. We start with a point at the origin (0,0) and then randomly pick one of the above transformation based on their probability, update the point position and then randomly pick another point. Each matrix adds a bit of rotation and translation with T_4 as a kind of restart.
To try to make our program a little easier, lets rewrite the above equations to make a system of "equivelent" equations of the form $Ax = b$ with only one matrix. We do this by adding an additional variable variable $z = 1$. For example, verify that the following equation is the same as equation for T_1 above:

$$T_1: \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.86 & 0.03 & 0 \\ -0.03 & 0.86 & 1.5 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

Please NOTE that we do not change the value for z, and it is always be 1.

Do This

Verify the $Ax = b$ format will generate the same answer as the T_1 equation above.

The following is some pseudocode that we will be using to generate the Fractals:

1. Let $x = 0$, $y = 0$, $z = 1$
2. Use a random generator to select one of the affine transformations T_i according to the given probabilities.
3. Let $(x', y') = T_i(x, y, z)$.
4. Plot (x', y')
5. Let $(x, y) = (x', y')$
6. Repeat Steps 2, 3, 4, and 5 one thousand times.

The following python code implements the above pseudocode with only the T_1 matrix:

```
%matplotlib inline

import numpy as np
import matplotlib.pylab as plt
import sympy as sym
sym.init_printing(use_unicode=True) # Trick to make matrixes look nice in jupyter

T1 = np.matrix([[0.86, 0.03, 0],[-0.03, 0.86, 1.5]])
#####Start your code here #####
T2 = T1
T3 = T1
T4 = T1
#####End of your code here#####
```

```
prob = [0.83,0.08,0.08,0.01]

I = np.matrix([[1,0,0],[0,1,0],[0,0,1]])

fig = plt.figure(figsize=[10,10])
p = np.matrix([[0.],[0],[1]])
plt.plot(p[0],p[1], 'go');
for i in range(1,1000):
    ticket = np.random.random();
    if (ticket < prob[0]):
        T = T1
    elif (ticket < sum(prob[0:2])):
        T = T2
    elif (ticket < sum(prob[0:3])):
        T = T3
    else:
        T = T4
    p[0:2,0] = T*p
    plt.plot(p[0],p[1], 'go');
plt.axis('scaled');
```

run | restart | restart & run all

Do This

Modify the above code to add in the T_2, T_3 and T_4 transforms.

Question

Describe in words for the actions performed by T_1, T_2, T_3, and T_4.

Do This

Using the same ideas to design and build your own fractal. You are welcome to get inspiration from the internet. Make sure you document where your inspiration comes from. Try to build something fun, unique and different. Show what you come up with with your instructors.

CHAPTER OVERVIEW

15: 08 Pre-Class Assignment - Robotics and Reference Frames

15.0: Introduction

Image from: www.fanuc.com

Goals for today's pre-class assignment

1. Review
2. 2D Forward Kinematics
3. Assignment Wrap-up

Reference: https://studywolf.wordpress.com/2013/08/21/robot-control-forward-transformation-matrices/

15.1: Review

Question

Matrix A is of size ($m_1 \times n_1$) and matrix B is of size ($m_2 \times n_2$). What must be true about the dimensions in order to multiply $A \times B$?

Question

The following transformation matrix will move points in R^n dimensional space. What is n?

$$\begin{bmatrix} \sin(\theta) & -\cos(\theta) & 0 & d_x \\ \cos(\theta) & \sin(\theta) & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Question

The above matrix rotates around which axis?

Question

In the above matrix, how do the scalar values d_x,d_y,d_z influence the transformation?

Question

Compute $2u + 3v$ for vectors $u = (1, 2, 6)$ and $v = (4, -1, 3)$.

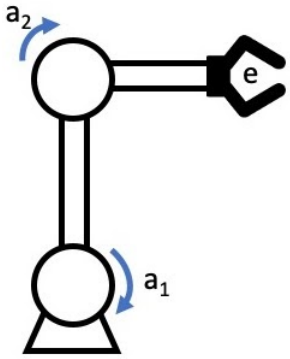
Question

What is a homogeneous system of linear equations?

15.2: 2D Forward Kinematics

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
from ipywidgets import interact
import sympy as sym
sym.init_printing(True)
```

run | restart | restart & run all



This robot can move in the \(x−y\) plane. We can represent the configuration of the robot in its "*Joint Space*" by knowing the two joint angles or \([a_1,a_2]\). However what we would like is to represent the location of the end of the robot (often called the "end effector" or "hand") in "world" coordinates (i.e. \(x−y\) coordinate system).
Today, we will use Linear Algebra and simple transformation matrices to try and calculate how to go from "joint" coordinates to "world" coordinates.

```
from IPython.display import YouTubeVideo
YouTubeVideo("aCohcLYrYcY",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Single axis Robot

The following code draws a simple single axis (single joint) robot with its joint centered at the origin and its initial angle of zero with an robot arm length of 4 “units” long.

```
plt.scatter(4,0, s=200, facecolors='none', edgecolors='r') #plot end effector
plt.scatter(0,0, s=200, facecolors='r', edgecolors='r') # plot origin
plt.plot([0,4],[0,0]) #plot blue line for arm
plt.axis('square')
plt.xlim([-5.5,5.5])
plt.ylim([-5.5,5.5])
# 'Run' this cell to see the output
```

run | restart | restart & run all

A 2D rotation matrix around the origin is defined as the following:

\[\begin{split}
\left[\begin{matrix}

x_{end} \\
y_{end}
\end{matrix}
\right]
=
\left[\begin{matrix}
\cos(a) & -\sin(a) \\
\sin(a) & \cos(qa
\end{matrix}
\right]
\left[\begin{matrix}
x_{start} \\
y_{start}
\end{matrix}
\right]
\end{split} \nonumber \]

The following rotation matrix will rotate the point \(45^o\) around the origin:

```
p = [[4],[0]]

a1=np.pi/4

R = np.matrix([[np.cos(a1), -np.sin(a1)], [np.sin(a1), np.cos(a1)]])

p2 = R*p

x1 = p2[0,0]
y1 = p2[1,0]

plt.scatter(x1,y1, s=200, facecolors='none', edgecolors='r') #plot end effector
plt.scatter(0,0, s=200, facecolors='r', edgecolors='r') # plot origin
plt.plot([0,x1],[0,y1]) #plot blue line for arm
plt.axis('square')
plt.xlim([-5.5,5.5])
plt.ylim([-5.5,5.5])
sym.Matrix(R)
```

run | restart | restart & run all

The following code uses the Jupyter `interact` function and `numpy` to make an interactive view of the above. This lets us change the value of the rotation motor and see how it changes the robot. The input to the function is the axis angle and the output is the \(x−y\) coordinates.

Note

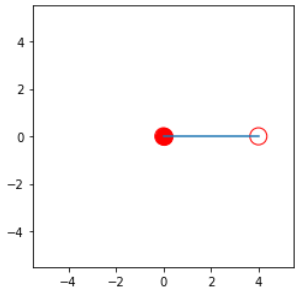
It can take some time for the interaction to catch up. Try moving the slider slowly…

```
def Robot_Simulator(q1=0):
    a1 = q1/180  * np.pi
    p0 = np.matrix([4,0]).T
    p = p0
    J1 = np.matrix([[np.cos(a1), -np.sin(a1)], [np.sin(a1), np.cos(a1)]])
    p = np.concatenate( ( J1*p, np.matrix([0,0]).T), axis=1 )
```

```
    plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=200, facecolors='none', edge
    plt.scatter(0,0, s=200, facecolors='r', edgecolors='r')
    plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
    plt.axis('square')
    plt.xlim([-5.5,5.5])
    plt.ylim([-5.5,5.5])
    ax = plt.gca()
    plt.show()
    return([ p[0,0], p[1,0] ])

p = interact(Robot_Simulator, q1=(-180,180,2));
```

run | restart | restart & run all

q1 0



$\displaystyle \left[4.0, \ 0.0\right]$

Do This

Inspect the above code.

Question

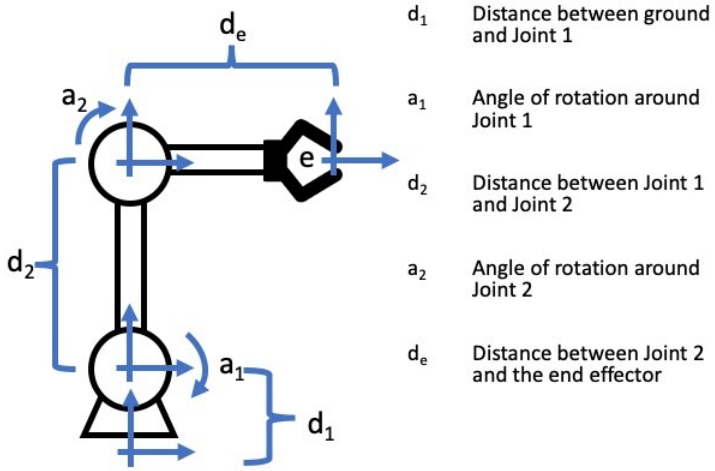
Which line of the code defines the transformation matrix for the robot joint?

Question

Which line of code defines the length of the robot arm?

Multi axis robot

Now, let's consider the robot from the video:



Notice it has two joints \(a_1\) and \(a_2\) and offset lengths of \(d_1\), \(d_2\) and \(d_e\). The joint space for this robot is just its angles \([a_1,a_2]\). However, what we want is to know the location of end effector point \(p_e\) at the gripper in the "world" reference frame, which the bottom most axies "on the ground".

At each joint, we can define a reference frame that rotates and then transforms the origin to the ealier joint. The forward transformation matrices capture the relationship between the reference frames of different links of the robot.

For example, We can move from the base motor, or \(p_1\), reference frame to the world, or \(p_w\), reference frame using the following equations:

\[\begin{split}
p_w
=
\left[\begin{matrix}
\cos(a_1) & -\sin(a_1) \\
\sin(a_1) & \cos(a_1)
\end{matrix}
\right]
p_1
+
\left[\begin{matrix}
0 \\
d_1
\end{matrix}
\right].
\end{split} \nonumber \]

The equation shown above are a little tricky to work with because the \(\left[\begin{matrix}
0 \\
d_1
\end{matrix}
\right] \) part makes the equation non-linear (if you don't believe me? Remember the rules for making a function linear and apply them and see for yourself). However, there is an easy trick in Linear Algebra to convert the above to one big linear matrix. This trick requires us to keep an extra 1 (one) for each point but makes the math work out nicely. Basically the trick works as follows:

\[\begin{split}
\left[\begin{matrix}
x_w \\
y_w \\
1
\end{matrix}

\right]
=
\left[\begin{matrix}
cos(a_1) & -sin(a_1) & 0 \\
sin(a_1) & cos(a_1) & d_1 \\
0 & 0 & 1
\end{matrix}
\right]
\left[\begin{matrix}
x_1 \\
y_1 \\
1
\end{matrix}
\right]
\end{split} \nonumber \]
Let's call the transformation matrix for Joint 1, \(J_1\), and we can rewrite the equations as follows:
\[p_1 = J_1p_2 \nonumber \]

Do This

On your own, write out the above matrix multiplication and convince yourself that it is the same as the one above.

Now, we can move from the \(p_2\) reference frame to the \(p_1\) reference frame basically the same equation:
\[\begin{split}
\left[\begin{matrix}
x_1 \\
y_1 \\
1
\end{matrix}
\right]
=
\left[\begin{matrix}
cos(a_2) & -sin(a_2) & d_2 \\
sin(a_2) & cos(a_2) & 0 \\
0 & 0 & 1
\end{matrix}
\right]
\left[\begin{matrix}
x_2 \\
y_2 \\
1
\end{matrix}
\right]
\end{split} \nonumber \]
For the last step we can do a simple linear transpose from the end effector \(p_e\) reference frame to the \(p_2\) reference frame:
\[\begin{split}
\left[\begin{matrix}
x_2 \\
y_2 \\
1
\end{matrix}
\right]
=
\left[\begin{matrix}

1 & 0 & d_2 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{matrix}
\right]
\left[\begin{matrix}
x_e \\
y_e \\
1
\end{matrix}
\right]
\end{split} \nonumber \]

If we call each transformation matrix \(J_1\),\(J_2\),\(J_e\) then hopefully you can see that we can string these transformation matrices together such that we get a single transform from the end effector all the way back to the world coordinates as follows:

\[p_w = J_1J_2J_ep_e \nonumber \]

Let's see what this looks like in Python. I am going to use `numpy` . The plotting gets a little awkward but hopefully it makes sense.

First, lets initialize the variables to some discreet numbers:

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
from ipywidgets import interact

#Inicial state
a1 = 0
a2 = 0

#Lenths of the offsets
d1 = 0.5
d2 = 3
de = 3
```

run | restart | restart & run all

Next, I am going to define a set of points in the end effector coordinate system. These points are picked to form a sort of "C" shaped designed to look sort of like an end effector. I will plot them to help show you what I mean:

```
#Points needed to define a square
pe = np.matrix([[1, 0.5, 1],[0,0.5,1],[0,-0.5, 1],[1,-0.5, 1],[0,-0.5, 1],[0,0, 1]]).T
p = pe

plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgecolors
plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
plt.axis('scaled')
plt.xlim([-5.5,5.5])
plt.ylim([-5.5,5.5])
# 'Run' this cell to see the output
```

run | restart | restart & run all

The next step is to apply the \(J_e\) transformation matrix to the gripper points which will put them in the \(p_2\) coordinate system. Once the points are transposed the code concatenates the origin (0,0) onto the list of points so we can part of the robot arm

in the plot:

```
Je = np.matrix([[1, 0, de],
                [0, 1, 0],
                [0,0,1]])

p = np.concatenate( ( Je*p, np.matrix([0,0,1]).T), axis=1 )

plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgecolors
plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
plt.axis('scaled')
plt.xlim([-5.5,5.5])
plt.ylim([-5.5,5.5])
```

run | restart | restart & run all

We do this again. Apply the \(J_2\) transformation matrix to put the points into the \(p_1\) coordinate system, concatenate the origin and plot the results.

```
J2 = np.matrix([[np.cos(a2), -np.sin(a2), 0],
                [np.sin(a2), np.cos(a2), d2],
                [0,0,1]])

p = np.concatenate( ( J2*p, np.matrix([0,0,1]).T), axis=1 )

plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgecolors
plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
plt.axis('scaled')
plt.xlim([-5.5,5.5])
plt.ylim([-5.5,5.5])
```

run | restart | restart & run all

We do it yet again. Apply the \(J_1\) transformation matrix which will put the points in the \(p_w\) coordinate system, concatenate the origin and plot the results. The result is a skeletal frame representing our robot.

```
J1 = np.matrix([[np.cos(a1), -np.sin(a1), 0],
                [np.sin(a1), np.cos(a1), d1],
                [0,0,1]])

p = np.concatenate( ( J1*p, np.matrix([0,0,1]).T), axis=1 )

plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgecolors
plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
plt.axis('scaled')
plt.xlim([-8,8])
plt.ylim([-8,8])
```

run | restart | restart & run all

Do This

Modify the rotation variables `a1` and `a2` in the above code and see if the new robot configuration looks right.
HINT make sure your angles are in radians.

The following is the same code as above but put into an interactive function to make the code easier to play with:

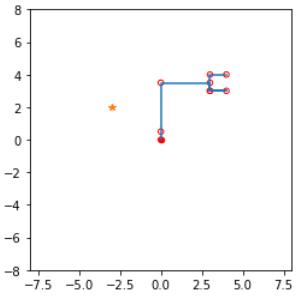
```
from ipywidgets import interact

def Robot_Simulator(q1=0,q2=-0):
    a1 = q1/180  * np.pi
    a2 = q2/180  * np.pi

    d1 = 0.5
    d2 = 3
    de = 3

    target = np.matrix([-3,2, 1])
    print(target)

    pe = np.matrix([[1, 0.5, 1],[0,0.5,1],[0,-0.5, 1],[1,-0.5, 1],[0,-0.5, 1],[0,0, 1]

    Je = np.matrix([[1, 0, de],
                    [0, 1, 0],
                    [0,0,1]])
    p = np.concatenate( ( Je*pe, np.matrix([0,0,1]).T), axis=1 )

    J2 = np.matrix([[np.cos(a2), -np.sin(a2), 0],
                    [np.sin(a2), np.cos(a2), d2],
                    [0,0,1]])
    p = np.concatenate( ( J2*p, np.matrix([0,0,1]).T), axis=1 )

    J1 = np.matrix([[np.cos(a1), -np.sin(a1), 0],
                    [np.sin(a1), np.cos(a1), d1],
                    [0,0,1]])
    p = np.concatenate( ( J1*p, np.matrix([0,0,1]).T), axis=1 )


    plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgec
    plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
    plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
    plt.plot(target[0,0], target[0,1],'*')
    plt.axis('scaled')
    plt.xlim([-8,8])
    plt.ylim([-8,8])

    plt.show()

target = interact(Robot_Simulator, q1=(-180,180), q2=(-180,180));
```

run | restart | restart & run all

```
        q1                      0
```

q2 0

```
[[-3  2  1]]
```



Note

If the `interact` plot is really choppy on your machine this is because the function is coninuously updating as you move the sliders. Consider replacing the `interact` function with `interact_manual` to eliminate the continuous updates. You will have to change the import code in line 1.

Question

Move the above robot so that the end effector is "gripping" the target (yellow/orangeish star). Notice that there is more than one point in the "joint space" that gives the same answer. This is the reverse Kinematic problem (which is harder). We know the point we want but we need to find the joints that put the robot at that point.

Do This

The code in the following cell is cut and pasted from above. Modify the code to add a third Joint to the robot.

```
from ipywidgets import interact

def Robot_Simulator(q1=0,q2=-0):
    a1 = q1/180  * np.pi
    a2 = q2/180  * np.pi
    #####Start your code here #####

    #####End of your code here#####

    d1 = 0.5
    d2 = 3
    de = 3
    #####Start your code here #####

    #####End of your code here#####


    target = np.matrix([-3,2, 1])
    print(target)

    pe = np.matrix([[1, 0.5, 1],[0,0.5,1],[0,-0.5, 1],[1,-0.5, 1],[0,-0.5, 1],[0,0, 1]

    Je = np.matrix([[1, 0, de],
                    [0, 1, 0],
```

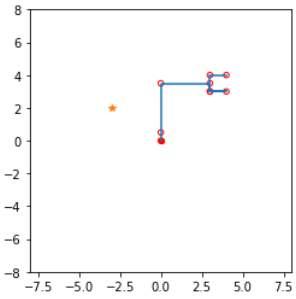
```
                    [0,0,1]])
    p = np.concatenate( ( Je*pe, np.matrix([0,0,1]).T), axis=1 )

    #####Start your code here #####

    #####End of your code here#####

    J2 = np.matrix([[np.cos(a2), -np.sin(a2), 0],
                    [np.sin(a2), np.cos(a2), d2],
                    [0,0,1]])
    p = np.concatenate( ( J2*p, np.matrix([0,0,1]).T), axis=1 )

    J1 = np.matrix([[np.cos(a1), -np.sin(a1), 0],
                    [np.sin(a1), np.cos(a1), d1],
                    [0,0,1]])

    p = np.concatenate( ( J1*p, np.matrix([0,0,1]).T), axis=1 )


    plt.scatter(p[0,:].tolist()[0],p[1,:].tolist()[0], s=20, facecolors='none', edgecc
    plt.scatter(0,0, s=20, facecolors='r', edgecolors='r')
    plt.plot(p[0,:].tolist()[0],p[1,:].tolist()[0])
    plt.plot(target[0,0], target[0,1],'*')
    plt.axis('scaled')
    plt.xlim([-8,8])
    plt.ylim([-8,8])

    plt.show()

target = interact(Robot_Simulator, q1=(-180,180), q2=(-180,180));
```

run | restart | restart & run all

q1 0

q2 0

```
[[-3  2  1]]
```



Question

Do the reverse Kinematics again, and find three angles that place the robot on the star.

15.3: Assignment wrap-up

Assignment-Specific Question

What three angles did you find that place the robot on the star.

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

16: 08 In-Class Assignment - The Kinematics of Robotics

16.0: Introduction

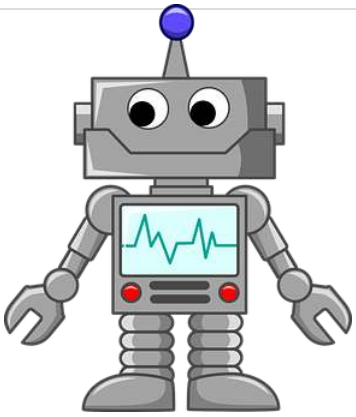


Image from: https://pixabay.com/images/search/toy robot/

Today, we will calculate the forward kinematics of some 3D robots. This means we would like to come up with a set of transformations such that we can know the x, y, z coordinates of the end effector with respect to the world coordinate system which is at the base of the robot.

Agenda for today's class (80 minutes)

1. (20 minutes) Review Pre-Class Assignment
2. (30 minutes) Robot Kinematics - Pick and Place
3. (30 minutes) Odd Clock

16.1: Review Pre-class Assignment

Like most days, your instructor will review and answer questions from your pre-class assignment. This is your opportunity to ask any lingering questions before the quiz.

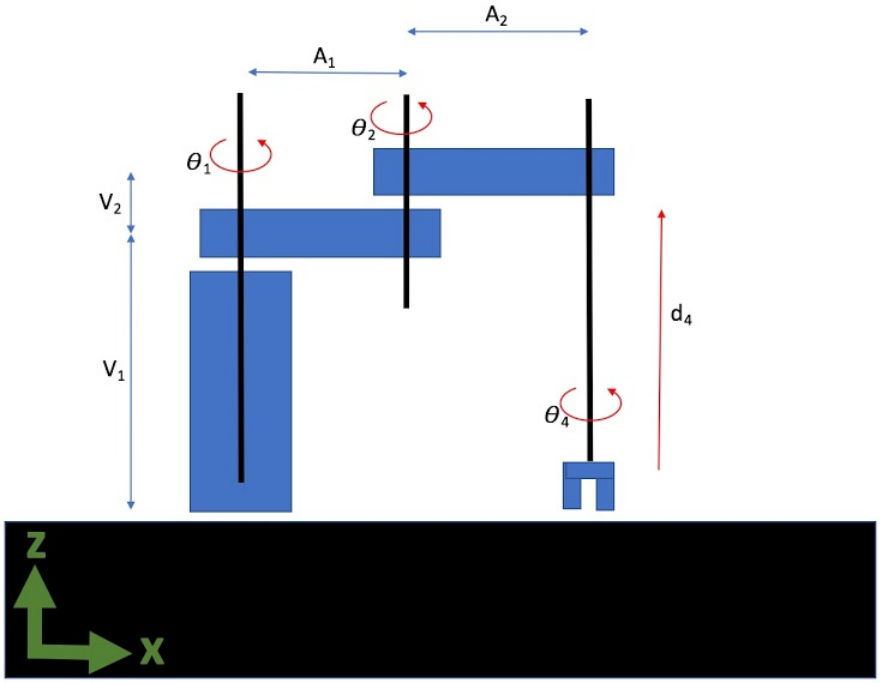
- 08 Pre-Class Assignment - Robotics and Reference Frames

16.2: Pick and Place


```
# Here are some libraries you may need to use
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
import math
sym.init_printing()
```

Consider the robot depicted in the following image.

This style of robot is often called a “pick-and-place” robot. It has two motors that rotate around the z-axis to move the end effector in the $x-y$ plane; one “linear actuator” which moves up-and-down in the z-direction; and then finally a third rotating “wrist” joint that turns the “hand” of the robot. Let’s model our robot using the following system diagram:



Note

The origin for this robot is located at the base of the first "tower" and is in-line with the first joint. The x-direction goes from the origin to the right and the z-axis goes from the origin upwards.

This is a little more tricky than the 2D case where everything was rotating around the axes that projects out of the $x-y$ plane.

In 2D we only really worry about one axis of rotation. However in 3D we can rotate around the x, y, and z axis. The following are the 3D transformation matrices that combine rotation and translations:

X-Axis rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & cos(q) & -sin(q) & dy \\ 0 & sin(q) & cos(q) & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Y-Axis rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(q) & 0 & sin(q) & dx \\ 0 & 1 & 0 & dy \\ -sin(q) & 0 & cos(q) & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Z-Axis rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(q) & -sin(q) & 0 & dx \\ sin(q) & cos(q) & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Question (a)

Construct a joint transformation matrix called J_1, which represents a coordinate system that is located at the top of the first "tower" (robot's sholder) and moves by rotating around the z-axis by θ_1 degrees. Represent your matrix using `sympy` and the provided symbols:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Use the following symbols

q1,q2,d4,q4,v1,v2,a1,a2 = sym.symbols('\Theta_1, \Theta_2, d_4, \Theta_4, V_1, V_2,A_
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your answer here
q1
```

Question (b)

Construct a joint transformation matrix called J_2, which represents a coordinate system that is located at the "elbow" joint between the two rotating arms and rotates with the second arm around the z-axis by θ_2 degrees. Represent your matrix using `sympy` and the symbols provided in question a:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your answer here
```

Question (c)

Construct a joint transformation matrix called J_3, which represents a coordinate translation from the "elbow" joint all the way to the horizontal end of the robot arm above the wrist joint. Note: there is no rotation in this transformation. Represent your matrix using `sympy` and the symbols provided in question a:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your answer here
```

Question (d)

Construct a joint transformation matrix called J_4, which represents a coordinate system that is located at the tip of the robot's "hand" and rotates around the z-axis by θ_4. This one is a little different, the configuration is such that the hand touches the table when $d_4 = 0$ so the translation component for the matrix in the z axis is $d_4 - V_1 - V_2$.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your answer here
```

Question

Rewrite the joint transformation matrices from questions a - d as numpy matrices with discrete (instead of symbolic) values. Plug in your transformations in the code below and use this to simulate the robot:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from ipywidgets import interact
from mpl_toolkits.mplot3d import Axes3D

def Robot_Simulator(theta1=0,theta2=-0,d4=0,theta4=0):

    #Convert from degrees to radians
    q1 = theta1/180 * math.pi
    q2 = theta2/180 * math.pi
    q4 = theta4/180 * math.pi

    #Define robot geomitry
    V1 = 4
    V2 = 0
    A1 = 2
    A2 = 2

    #Define your transfomraiton matrices here.
    J1 = np.matrix([[1, 0, 0, 0 ],
                    [0, 1, 0, 0 ],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])

    J2 = np.matrix([[1, 0, 0, 0 ],
                    [0, 1, 0, 0 ],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])

    J3 = np.matrix([[1, 0, 0, 0 ],
                    [0, 1, 0, 0 ],
                    [0, 0, 1, 0],
```

```
                    [0, 0, 0, 1]])

    J4 = np.matrix([[1, 0, 0, 0 ],
                    [0, 1, 0, 0 ],
                    [0, 0, 1, 0],
                    [0, 0, 0, 1]])


    #Make the rigid end effector
    p = np.matrix([[-0.5,0,0, 1], [-0.5,0,0.5,1], [0.5,0,0.5, 1], [0.5,0,0,1],[0.5,0,(

    #Propogate and add joint points though the simulation
    p = np.concatenate((J4*p, np.matrix([0,0,0,1]).T), axis=1 )
    p = np.concatenate((J3*p, np.matrix([0,0,0,1]).T), axis=1 )
    p = np.concatenate((J2*p, np.matrix([0,0,0,1]).T), axis=1 )
    p = np.concatenate((J1*p, np.matrix([0,0,0,1]).T), axis=1 )

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(p[0,:].tolist()[0],(p[1,:]).tolist()[0], (p[2,:]).tolist()[0], s=20, fa
    ax.scatter(0,0,0, s=20, facecolors='r', edgecolors='r')
    ax.plot(p[0,:].tolist()[0],(p[1,:]).tolist()[0], (p[2,:]).tolist()[0])
    ax.set_xlim([-5,5])
    ax.set_ylim([-5,5])
    ax.set_zlim([0,6])
    ax.set_xlabel('x-axis')
    ax.set_ylabel('y-axis')
    ax.set_zlabel('z-axis')

    plt.show()

target = interact(Robot_Simulator, theta1=(-180,180), theta2=(-180,180), d4=(0,6), th(
```

theta1

0

theta2

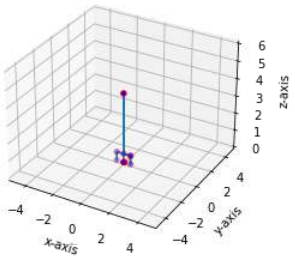
0

d4

0

theta4

0



Question

Can we change the order of the transformation matrices? Why? You can try and see what happens.

16.3: Odd Clock

Consider the clock depicted in the following image.

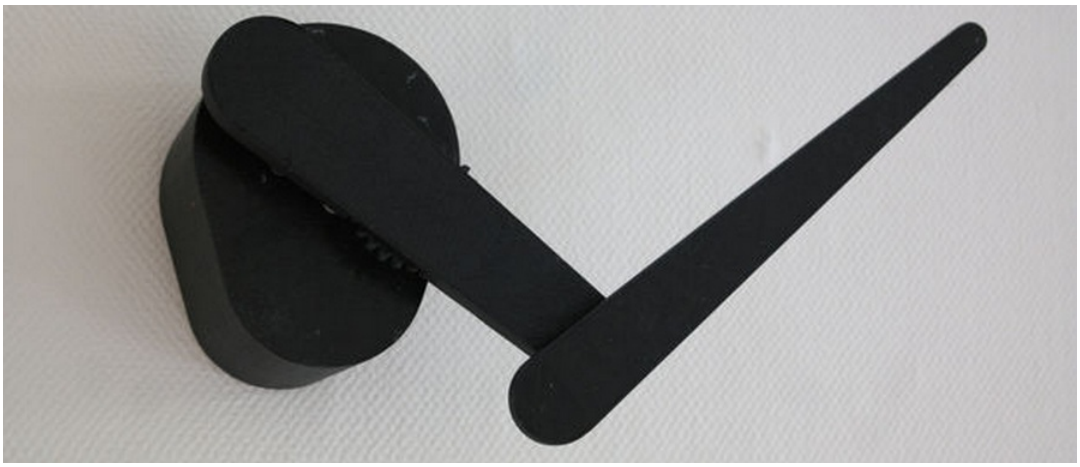
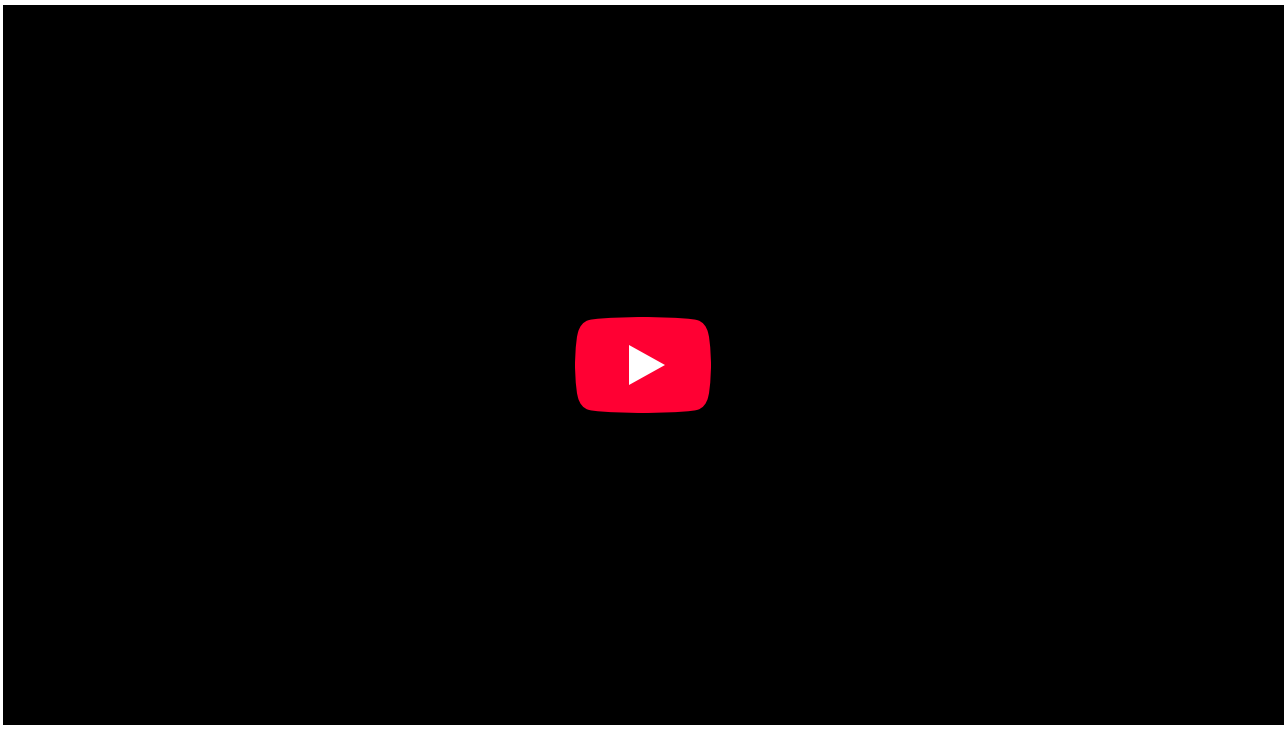
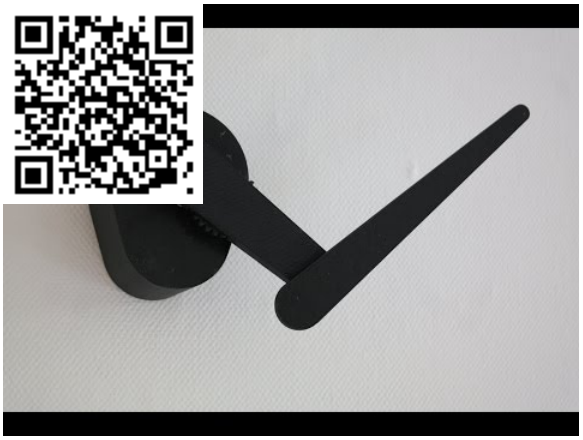


Image from: Hackaday.

Instead of a standard clock–which has independent hour and minute hands–this clock connects the minute hand at the end of the hour hand. Here is a video showing the sped-up clock motion:


```
from IPython.display import YouTubeVideo
YouTubeVideo("bowLiSlm_gA",width=640,height=360, mute=1)
```





The following code is an animated traditional clock which uses the function as a trick to animate things in jupyter:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
%matplotlib inline
import matplotlib.pylab as plt
```

```
from IPython.display import display, clear_output
import time
def show_animation(delay=0.01):
    fig = plt.gcf();
    time.sleep(delay)       # Sleep for half a second to slow down the animation
    clear_output(wait=True) # Clear output for dynamic display
    display(fig)            # Reset display
    fig.clear()             # Prevent overlapping and layered plots
```

Lets see a standard analog clock run at high speed

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import numpy as np
'''
Analog clock plotter with time input as seconds
'''
def analog_clock(tm=0):

    #Convert from time to radians
    a_minutes = -tm/(60*60) * np.pi * 2
    a_hours = -tm/(60*60*12) * np.pi * 2

    #Define clock hand sizees
    d_minutes = 4
    d_hours = 3
    arrow_width=0.5
    arrow_length=1

    # Set up figure
    fig = plt.gcf()
    ax = fig.gca();
    ax.set_xlim([-15,15]);
    ax.set_ylim([-10,10]);
    ax.scatter(0,0, s=15000, color="navy"); #Background Circle
    plt.axis('off');
```

```
    # Calculation Minute hand transformation matrix
    J2 = np.matrix([[np.cos(a_minutes), -np.sin(a_minutes)],
                    [np.sin(a_minutes), np.cos(a_minutes)]] )
    pm = np.matrix([[0,d_minutes], [-arrow_width,d_minutes], [0,arrow_length+d_minute
    pm = np.concatenate((J2*pm, np.matrix([0,0]).T), axis=1 );
    ax.plot(pm[0,:].tolist()[0],(pm[1,:]).tolist()[0], color='cyan', linewidth=2);

    # Calculation Hour hand transformation matrix
    J1 = np.matrix([[np.cos(a_hours), -np.sin(a_hours)],
                    [np.sin(a_hours), np.cos(a_hours)]] )
    ph = np.matrix([[0,d_hours], [0,d_hours], [-arrow_width,d_hours], [0,arrow_length
    ph = np.concatenate((J1*ph, np.matrix([0,0]).T), axis=1 );
    ax.plot(ph[0,:].tolist()[0],(ph[1,:]).tolist()[0], color='yellow', linewidth=2);
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Run the clock for about 5 hours at 100 times speed so we can see the hands move
for tm in range(0,60*60*5, 100):
    analog_clock(tm);
    show_animation();
# 'Run' this cell to see the animation
```

For the following few questions, consider the transformation matrix J_1 redefined below with an angle of 5 hours out of 12.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import sympy as sym
import numpy as np
sym.init_printing(use_unicode=True)

a_hours = 5/12 * 2 * np.pi
J1 = np.matrix([[np.cos(a_hours), -np.sin(a_hours)],
                [np.sin(a_hours), np.cos(a_hours)]] )

sym.Matrix(J1)
```

Question

Using code, show that the transpose of J_1 is also the inverse of J_1, then explain how the code demonstrates the answer.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer here
```

Question

Given the trigonometric identity $cos^2(\theta) + sin^2(\theta) = 1$, prove by construction–using either Python or LaTeX/ `Markdown` or `sympy` (if you are feeling adventurous)–that the transpose of the J_1 matrix is also the inverse for ANY angle `a_hours` $\in [0, 2\pi]$.

Now consider the following code which attempts to connect the hands on the clock together to make the Odd Clock shown in the video above.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import numpy as np

def odd_clock(tm=0):

    #Convert from time to radians
    #a_seconds = -tm/60 * np.pi * 2
    a_minutes = -tm/(60*60) * np.pi * 2
    a_hours = -tm/(60*60*12) * np.pi * 2

    #Define robot geomitry
    #d_seconds = 2.5
    d_minutes = 2
    d_hours = 1.5
    arrow_width=0.5
    arrow_length=1

    # Set up figure
    fig = plt.gcf()
    ax = fig.gca();
    ax.set_xlim([-15,15]);
    ax.set_ylim([-10,10]);
    plt.axis('off');

    #Define the arrow at the end of the last hand
    #p = np.matrix([[0,d_minutes,1], [0,0,1]]).T
    p = np.matrix([[0,d_minutes,1], [-arrow_width,d_minutes,1], [0,arrow_length+d_min

    # Calculation Second hand transformation matrix
    J2 = np.matrix([[np.cos(a_minutes), -np.sin(a_minutes), 0 ],
                    [np.sin(a_minutes), np.cos(a_minutes), d_hours ],
                    [0, 0, 1]])
    p = np.concatenate((J2*p, np.matrix([0,0,1]).T), axis=1 )

    J1 = np.matrix([[np.cos(a_hours), -np.sin(a_hours), 0 ],
                    [np.sin(a_hours), np.cos(a_hours), 0 ],
                    [0, 0, 1]])
    p = np.concatenate((J1*p, np.matrix([0,0,1]).T), axis=1 )

    ax.scatter(0,0, s=20, facecolors='r', edgecolors='r')
    ax.plot(p[0,:].tolist()[0],(p[1,:]).tolist()[0])
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page

```
#Run the clock for about 5 hours at 100 times speed so we can see the hands move
for tm in range(0,60*60*5, 100):
    odd_clock(tm);
    show_animation();
# 'Run' this cell to see the animation
```

Question

Using the given point (p) written in "minutes" coordinates (**on line 26 of the above code**) and the above transformation matrices (J_1,J_2), write down the equation to transform p into world coordinates p_w.

Question

Notice the above `odd_clock` function has variables `d_seconds` and `a_seconds` commented out. Use these variables and modify the above code to add a "seconds" hand on the tip of the minute hand such that the seconds hand moves around the minute hand just like the minute hand moves around the hour hand. If you have trouble, use the following cell to explain your thought process and where you are getting stuck.

CHAPTER OVERVIEW

17: 09 Pre-Class Assignment - Determinants

17.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 4.I-II pg 317-337
- ***Beezer Chapter D pg 340-366***

Goals for today's pre-class assignment

1. Introduction to Determinants
2. Properties of Determinants
3. One interpretation of determinants
4. Cramer's Rule
5. Assignment wrap-up

17.1: Introduction to Determinants

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

For a detailed overview of determinants I would recommend reviewing *Chapter D pg 340-366* of the Beezer text.
The determinant is a function that takes a $(n \times n)$ square matrix as an input and produces a scalar as an output. Determinants have been studied quite extensively and have many interesting properties. However, determinants are "computationally expensive" as the size of your matrix (n) gets bigger. This limitation makes them impractical for many real world problems.
The determinant of a 2×2 matrix can be calculated as follows:

$$det\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\right) = a_{11}a_{22} - a_{12}a_{21}$$

Question

Calculate the determinant of the following matrix by hand:

$$\begin{bmatrix} 3 & -2 \\ 1 & 2 \end{bmatrix}$$

Calculating the determinant of a larger matrix is a "recursive" problem which involves combining the determinants of smaller and smaller sub-matrices until you have a 2×2 matrix which is then calculated using the above formula. Here is some Pseudocode to calculate a determinant. To simplify the example the code assumes there is a matrix function `deleterow` which will remove the xth row from a matrix (always the first row in this example) and `deletecol` will remove the xth column from a matrix. When used together (as shown below) they will take an $n \times n$ matrix and turn it into a $(n-1) \times (n-1)$ matrix.

```
function determinant(A, n)
   det = 0
   if (n == 1)
      det = matrix[1,1]
   else if (n == 2)
      det = matrix[1,1] * matrix[2,2] - matrix[1,2] * matrix[2,1]
   else
      for x from 1 to n
          submatrix = deleterow(matrix, 1)
          submatrix = deletecol(submatrix, x)
          det = det + (x+1)**(-1) * matrix[1,x] * determinant(submatrix, n-1)
      next x
   endif

   return det
```

Notice that the combination of the determinants of the submatrixes is not a simple sum. The combination is adding the submatrices corresponding to the odd columns (1,3,5, etc) and subtracting the submatrices corresponding to the even columns (2,4,6, etc.). This may become clearer if we look at a simple 3×3 example (Let $|A|$ be a simplified syntax for writing the determinant of A):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$|A| = a_{11}\begin{vmatrix} \square & \square & \square \\ \square & a_{22} & a_{23} \\ \square & a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} \square & \square & \square \\ a_{21} & \square & a_{23} \\ a_{31} & \square & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} \square & \square & \square \\ a_{21} & a_{22} & \square \\ a_{31} & a_{32} & \square \end{vmatrix}$$

$$|A| = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$|A| = a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31})$$

Question

Calculate the determinant of the following matrix by hand:

$$\begin{bmatrix} 1 & 2 & -3 \\ 5 & 0 & 6 \\ 7 & 1 & -4 \end{bmatrix}$$

Question

Use the `numpy.linalg` library to calculate the determinant of the following matrix and stor the value in a variable called `det`

$$\begin{bmatrix} 2 & 0 & 1 & -5 \\ 8 & -1 & 2 & 1 \\ 4 & -3 & -5 & 0 \\ 1 & 4 & 8 & 2 \end{bmatrix}$$

```
#Put your answer here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.float(det,'49afb719e0cd46f74578ebf335290f81');
```

run | restart | restart & run all

17.2: Properties of Determinants

The following are some helpful properties when working with determinants. These properties are often used in proofs and can sometimes be utilized to make faster calculations.

Row Operations

Let A be an $n \times n$ matrix and c be a nonzero scalar. Let $\left| A \right|$ be a simplified syntax for writing the determinant of A:

1. If a matrix B is obtained from A by multiplying a row (column) by c then $\left| B \right| = c \left| A \right|$.
2. If a matrix B is obtained from A by interchanging two rows (columns) then $\left| B \right| = - \left| A \right|$.
3. if a matrix B is obtained from A by adding a multiple of one row (column) to another row (column), then $\left| B \right| = \left| A \right|$.

Singular Matrices

Definition: A square matrix A is said to be singular if $\left| A \right| = 0$. A is nonsingular if $\left| A \right| \neq 0$
Now, Let A be an $n \times n$ matrix. A is singular if any of these is true:

1. all the elements of a row (column) are zero.
2. two rows (columns) are equal.
3. two rows (columns) are proportional. i.e. one row (column) is the same as another row (column) multiplied by c.

Question

The following matrix is singular because of certain column or row properties. Give the reason:

$$\begin{split}
\left[
\begin{matrix}
1 & 5 & 5 \\
0 & -2 & -2 \\
3 & 1 & 1
\end{matrix}
\right]
\end{split} \nonumber$$

Question

The following matrix is singular because of certain column or row properties. Give the reason:

$$\begin{split}
\left[
\begin{matrix}
1 & 0 & 4 \\
0 & 1 & 9 \\
0 & 0 & 0
\end{matrix}
\right]
\end{split} \nonumber$$

Determinants and Matrix Operations

Let A and B be $n \times n$ matrices and c be a nonzero scalar.
Determinant of a scalar multiple: $\left| cA \right| = c^n \left| A \right|$
Determinant of a product: $\left| AB \right| = \left| A \right| \left| B \right|$
Determinant of a transpose: $\left| A^t \right| = \left| A \right|$
Determinant of an inverse: $\left| A-1 \right| = \frac{1}{\left| A \right|}$ (Assuming $A-1$ exists)

Question

If \(A\) is a \(3 \times 3\) matrix with \(\left| A \right| = 3\), use the properties of determinants to compute the following determinant:

\[\left| 2A \right| \nonumber \]

Question

If \(A\) is a \(3 \times 3\) matrix with \(\left| A \right| = 3 \), use the properties of determinants to compute the following determinant:

\[\left| A^2 \right| \nonumber \]

Question

If \(A\) and \(B\) are \(3 \times 3\) matrices and \(\left| A \right| = -3 \), \(\left| B \right| = 2\), compute the following determinant:

\[\left| AB \right| \nonumber \]

Question

If \(A\) and \(B\) are \(3 \times 3\) matrices and \(\left| A \right| = -3 \), \(\left| B \right| = 2\), compute the following determinant:

\[\left| 2 AB^{-1} \right| \nonumber \]

Triangular matrices

Definition: An upper triangular matrix has nonzero elements lie on or above the main diagonal and zero elements below the main diagonal. For example:

\[\begin{split} A =
\left[
\begin{matrix}
2 & -1 & 9 & 4 \\
0 & 3 & 0 & 6 \\
0 & 0 & -5 & 3 \\
0 & 0 & 0 & 1
\end{matrix}
\right]
\end{split} \nonumber \]

The determinant of an *upper triangle matrix* \(A\) is the product of the diagonal elements of the matrix \(A\).

Also, since the Determinant is the same for a matrix and it's transpose (i.e. \(\left| A^t \right| = \left| A \right| \), see definition above) the determinant of a *lower triangle matrix* is also the product of the diagonal elements.

Question

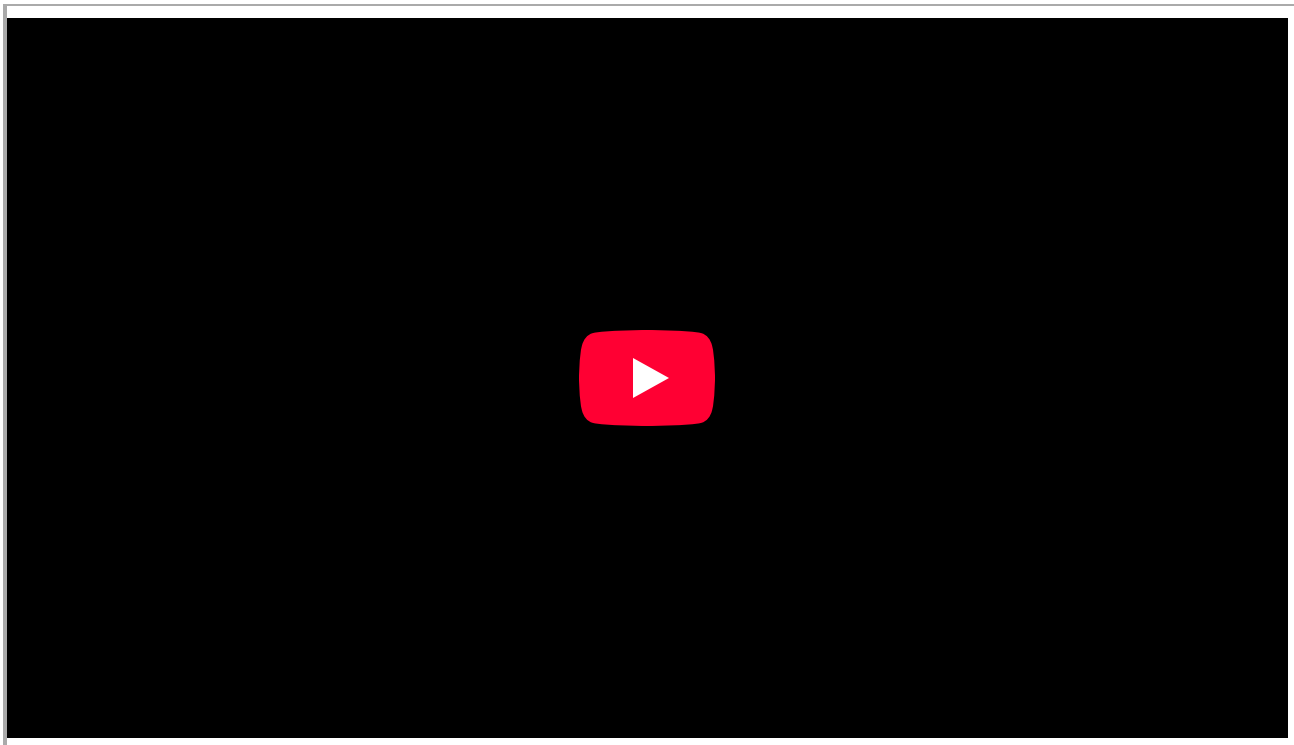
What is the determinant of matrix \(A\)?

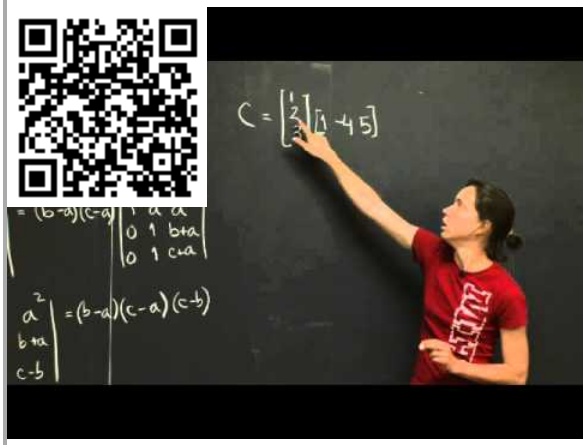
Using Properties of determinants:

Here is a great video showing how you can use the properties of determinants:

```
from IPython.display import YouTubeVideo
YouTubeVideo("aKX5_DucNq8",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all





Question (A challenging one)

Using the pattern established in the video can you calculate the determinate of the following matrix?

$$\begin{split}
\left[
\begin{matrix}
1 & a & a^2 & a^3 \\
1 & b & b^2 & b^3 \\
1 & c & c^2 & c^3 \\
1 & d & d^2 & d^3
\end{matrix}
\right]
\end{split} \nonumber$$

17.3: One interpretation of determinants

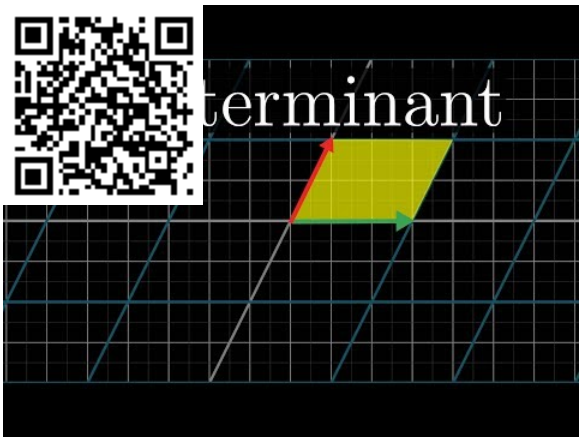
The following is an application of determinants. Watch this!

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("Ip3X9LOh2dk",width=640,height=360, cc_load_policy=True)
```



For fun, we will recreate some of the video's visualizations in Python. It was a little tricky to get the aspect ratios correct but here is some code I managed to get it work.


```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection, Line3DCollection
import numpy as np
import sympy as sym
```

```
# Lets define somme points that form a Unit Cube
points = np.array([[0, 0, 0],
                   [1, 0, 0 ],
                   [1, 1, 0],
                   [0, 1, 0],
                   [0, 0, 1],
                   [1, 0, 1 ],
                   [1, 1, 1],
                   [0, 1, 1]])

points = np.matrix(points)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

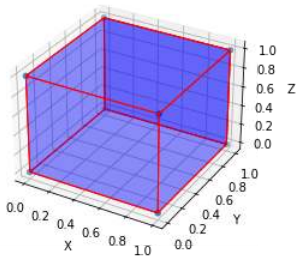
Login

```
#Here is some code to build cube from https://stackoverflow.com/questions/44881885/py

def plot3dcube(Z):

    if type(Z) == np.matrix:
        Z = np.asarray(Z)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    r = [-1,1]

    X, Y = np.meshgrid(r, r)
    # plot vertices
    ax.scatter3D(Z[:, 0], Z[:, 1], Z[:, 2])

    # list of sides' polygons of figure
    verts = [[Z[0],Z[1],Z[2],Z[3]],
     [Z[4],Z[5],Z[6],Z[7]],
     [Z[0],Z[1],Z[5],Z[4]],
     [Z[2],Z[3],Z[7],Z[6]],
     [Z[1],Z[2],Z[6],Z[5]],
```

```
     [Z[4],Z[7],Z[3],Z[0]],
     [Z[2],Z[3],Z[7],Z[6]]]

    #alpha transparency was't working found fix here:
    # https://stackoverflow.com/questions/23403293/3d-surface-not-transparent-inspite
    # plot sides
    ax.add_collection3d(Poly3DCollection(verts,
     facecolors=(0,0,1,0.25), linewidths=1, edgecolors='r'))

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    ## Weird trick to get the axpect ratio to work.
    ## From https://stackoverflow.com/questions/13685386/matplotlib-equal-unit-length
    mx = np.amax(Z, axis=0)
    mn = np.amin(Z, axis=0)
    max_range = mx-mn

    # Create cubic bounding box to simulate equal aspect ratio
    Xb = 0.5*max_range.max()*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten() + 0.5*(max_r
    Yb = 0.5*max_range.max()*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten() + 0.5*(max_r
    Zb = 0.5*max_range.max()*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten() + 0.5*(max_r
    # Comment or uncomment following both lines to test the fake bounding box:
    for xb, yb, zb in zip(Xb, Yb, Zb):
        ax.plot([xb], [yb], [zb], 'w')

    plt.show()
```



```
plot3dcube(points)
```



Question

The following the 3×3 was shown in the video (around 6'50''). Apply this matrix to the unit cube and use the `plot3dcube` to show the resulting transformed points.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
T = np.matrix([[1 , 0 ,  0.5],
               [0.5 ,1 ,1.5],
               [1 , 0 ,  1]])

#Put the answer to the above question here.
```

Question

The determinant represents how the area changes when applying a 2×2 transform. What does the determinant represent for a 3×3 transform?

17.4: Cramer's Rule

Do This

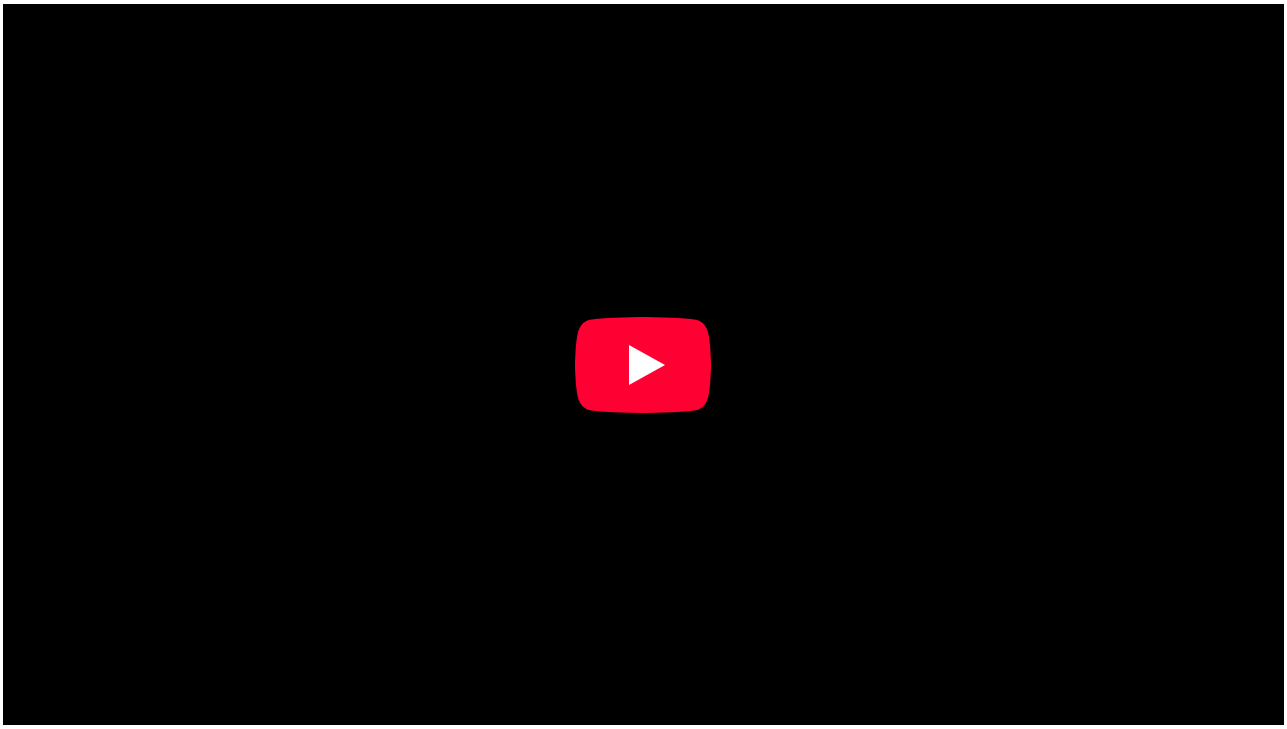
Watch the following video and come to class ready to discuss Cramer's Rule:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("BW6897HIOMA",width=640,height=360, cc_load_policy=True)
```



17.5: Assignment wrap-up

Assignment-Specific Question

What does the determinant represent for a 3×3 transform?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

18: 09 In-Class Assignment - Determinants

18.0: Introduction

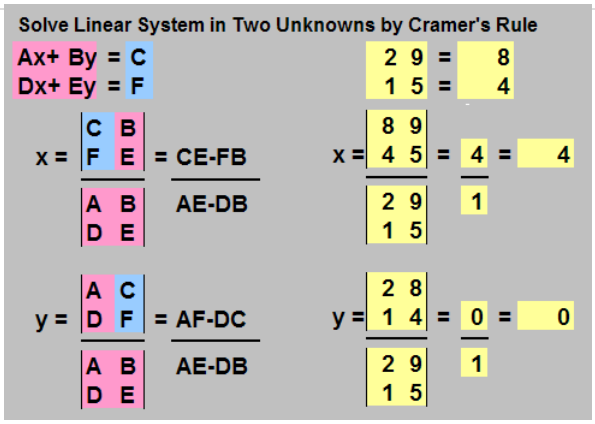


Image from: http://www.mathnstuff.com/

Agenda for today's class (80 minutes)

1. (20 minutes) Review Pre-class Assignment
2. (30 minutes) Algorithm to calculate the determinant
3. (30 minutes) Using Cramer's rule to solve $Ax = b$

18.1: Review Pre-class Assignment

- 09 Pre-Class Assignment: Determinants

18.2: Algorithm to calculate the determinant

Consider the following recursive algorithm (algorithm that calls itself) to determine the determinate of a $n \times n$ matrix A (denoted $|A|$), which is the sum of the products of the elements of any row or column. i.e.:

$$i\text{th row expansion: } |A| = a_{i1}C_{i1} + a_{i2}C_{i2} + \ldots + a_{in}C_{in}$$

$$j\text{th column expansion: } |A| = a_{1j}C_{1j} + a_{2j}C_{2j} + \ldots + a_{nj}C_{nj}$$

where C_{ij} is the cofactor of a_{ij} and is given by:

$$C_{ij} = (-1)^{i+j}|M_{ij}|$$

and M_{ij} is the matrix that remains after deleting row i and column j of A.
Here is some code that tries to implement this algorithm.

```
## Import our standard packages packages
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
sym.init_printing(use_unicode=True)
```

run | restart | restart & run all

```
import copy
import random

def makeM(A,i,j):
    ''' Deletes the ith row and jth column from A'''
    M = copy.deepcopy(A)
    del M[i]
    for k in range(len(M)):
        del M[k][j]
    return M

def mydet(A):
    '''Calculate the determinant from list-of-lists matrix A'''
    if type(A) == np.matrix:
        A = A.tolist()
    n = len(A)
    if n == 2:
        det = (A[0][0]*A[1][1] - A[1][0]*A[0][1])
        return det
    det = 0
    i = 0
    for j in range(n):
        M = makeM(A,i,j)

        #Calculate the determinant
        det += (A[i][j] * ((-1)**(i+j+2)) * mydet(M))
    return det
```

run | restart | restart & run all

The following code generates an $n \times n$ matrix with random values from 0 to 10.

Run the code multiple times to get different matrices:

```
#generate Random Matrix and calculate it's determinant using numpy
n = 5
s = 10
A = [[round(random.random()*s) for i in range(n)] for j in range(n)]
A = np.matrix(A)
#print matrix
sym.Matrix(A)
```

run | restart | restart & run all

Do This

Use the randomly generated matrix (A) to test the above `mydet` function and compare your result to the `numpy.linalg.det` function.

```
# Put your test code here
```

run | restart | restart & run all

Question

Are the answers to `mydet` and `numpuy.linalg.det` exactly the same every time you generate a different random matrix? If not, explain why.

Question

On line 26 of the above code, you can see that algorithm calls itself. Explain why this doesn't run forever.

18.3: Using Cramer's rule to solve Ax=b

Let $Ax = b$ be a system of n linear equations in n variables such that $|A| \neq 0$. The system has a unique solution given by:

$$x_1 = \frac{|A_1|}{|A|}, x_2 = \frac{|A_2|}{|A|}, \ldots, x_n = \frac{|A_n|}{|A|}$$

where A_i is the matrix obtained by replacing column i of A with b. The following function generates A_i by replacing the ith column of A with b:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
def makeAi(A,i,b):
    '''Replace the ith column in A with b'''
    if type(A) == np.matrix:
        A = A.tolist()
    if type(b) == np.matrix:
        b = b.tolist()
    Ai = copy.deepcopy(A)
    for j in range(len(Ai)):
        Ai[j][i] = b[j][0]
    return Ai
```

Do This

Create a new function called `cramersRule`, which takes A and b and returns x using the Cramer's rule.

Note: Use `numpy` and NOT `mydet` to find the required determinants. `mydet` is too slow.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Stub code. Replace the np.linalg.solve code with your answer

def cramersRule(A,b):
    detA = np.linalg.det(A)
    x = []
    #####Start of your code here#####


    #####End of your code here#####

    return x
```

Question

Test your `cramersRule` function on the following system of linear equations:

$$x_1 + 2x_2 = 3$$

$$3x_1 + x_2 = -1$$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Question

Verify the above answer by using the `np.linalg.solve` function:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Question

Test your `cramersRule` function on the following system of linear equations and verify the answer by using the `np.linalg.solve` function:

$$x_1 + 2x_2 + x_3 = 9$$
$$x_1 + 3x_2 - x_3 = 4$$
$$x_1 + 4x_2 - x_3 = 7$$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Question

Cramer's rule is a $O(n!)$ algorithm and the Gauss-Jordan (or Gaussian) elimination is $O(n^3)$. What advantages does Cramer's rule have over elimination?

CHAPTER OVERVIEW

19: 10 Pre-Class Assignment - Eigenvectors and Eigenvalues

19.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 5 II.3 pg 397-407
- Beezer Chapter E pg 367-369

Goals for today's pre-class assignment

1. Eigenvectors and Eigenvalues
2. Solving Eigenproblems - A 2x2 Example
3. Introduction to Markov Models

19.1: Eigenvectors and Eigenvalues

Understanding Eigenvector and Eigenvalues can be very challenging. These are complex topics with many facets. Different textbooks approach the problem from different directions. All have value. These facets include:

- Understanding the mathematical definition of Eigenvalues.
- Being able to calculate an Eigenvalue and Eigenvector.
- Understanding what Eigenvalues and Eigenvectors represent.
- Understanding how to use Eigenvalues and Eigenvectors to solve problems.

In this course we consider it more important to understand what eigenvectors and eigenvalues represent and how to use them. However, often this understanding comes from first learning how to calculate them.

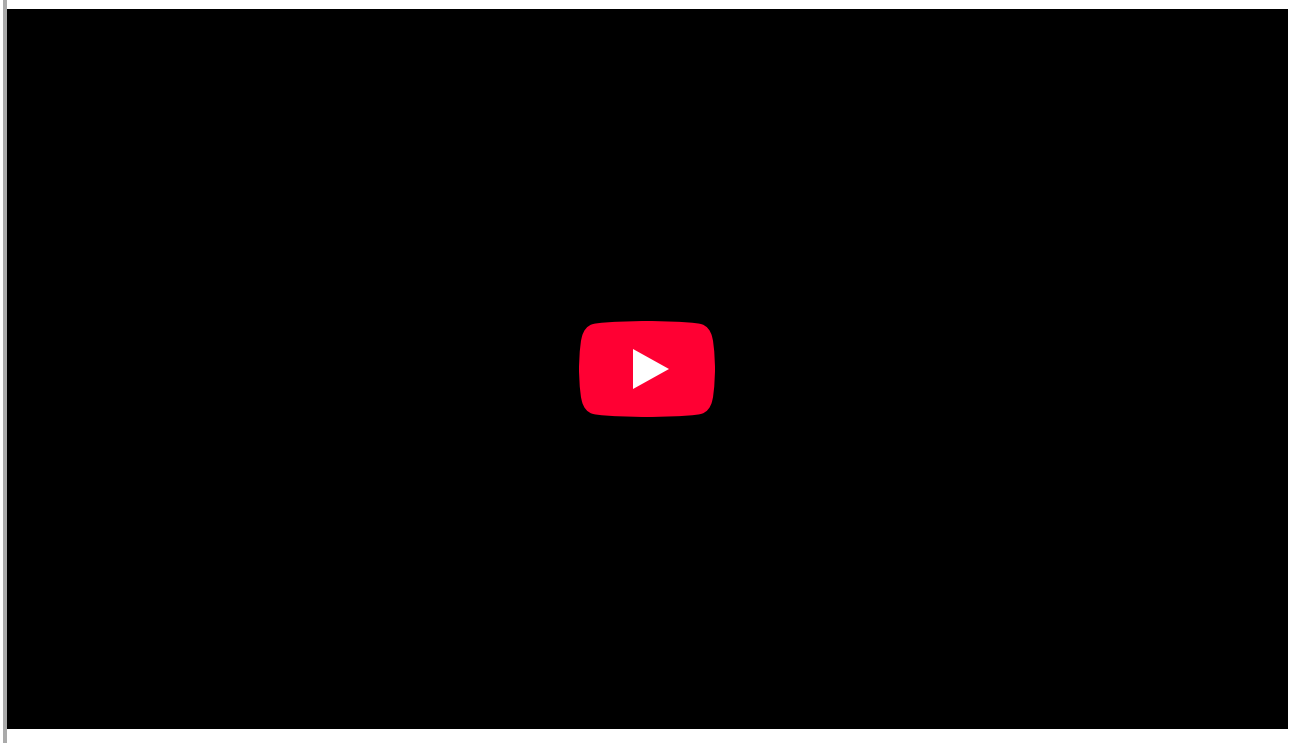
> *Eigenvalues are a special set of scalars associated with a square matrix that are sometimes also known as characteristic roots, characteristic values (Hoffman and Kunze 1971), proper values, or latent roots (Marcus and Minc 1988, p. 144).*
> *The determination of the eigenvalues and eigenvectors of a matrix is extremely important in physics and engineering, where it is equivalent to matrix diagonalization and arises in such common applications as stability analysis, the physics of rotating bodies, and small oscillations of vibrating systems, to name only a few.*
> *The decomposition of a square matrix \(A\) into eigenvalues and eigenvectors is known in this work as eigen decomposition, and the fact that this decomposition is always possible as long as the matrix consisting of the eigenvectors of \(A\) is square. This is known as the eigen decomposition theorem.*

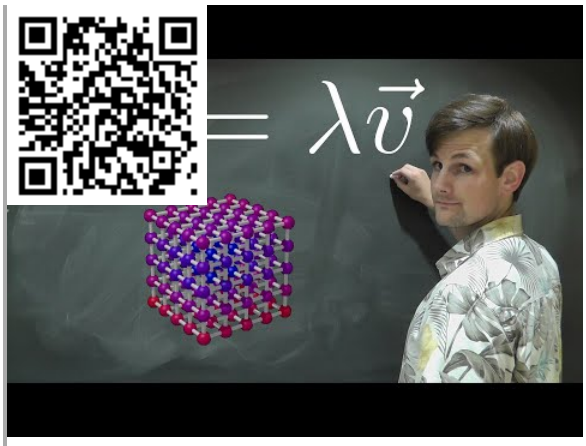
From: http://mathworld.wolfram.com/Eigenvalue.html

The following video provides an intuition for eigenvalues and eigenvectors.

```
from IPython.display import YouTubeVideo
YouTubeVideo("ue3yoeZvt8E",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all





Definition

Let \(A\) be an \(n \times n\) matrix. Find a vector \(x\) in \(R^n\) such that:
\[Ax=\lambda x \nonumber \]
The above can be rewritten as the following homogeneous equation:
\[(A-\lambda I_n)x = 0 \nonumber \]
The trivial solution is \(x=0\).
However, if we define eigenvectors to be nonzero vectors then \(|A - \lambda I_n| = 0\). Nonzero (i.e. non-trivial) solutions to this system of equations can only exist if the matrix of coefficients is singular, i.e. the determinant of \(|A - \lambda I_n| = 0 \). Therefore, solving the equation \(|A - \lambda I_n| = 0 \) for \(\lambda\) leads to all the eigenvalues of \(A\).

Note

The above logic is key. Make sure you understand. If not, ask questions.

Question

Explain why nonzero solutions to a system of homogeneous systems require the matrix to be singular.

```
from IPython.display import YouTubeVideo
YouTubeVideo("PFDu9oVAE-g",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Examples:

Here are a few more examples of how eigenvalues and eigenvectors are used (You are not required to understand all):

> *Using singular value decomposition for image compression. This is a note explaining how you can compress an image by throwing away the small eigenvalues of \(A^{T}A\). It takes an 88 megapixel image of an Allosaurus and shows how the image looks after compressing by selecting the largest singular values.*
>
> *Deriving Special Relativity is more natural in the language of linear algebra. In fact, Einstein's second postulate really states that "Light is an eigenvector of the Lorentz transform." This document goes over the full derivation in detail.*
>
> *Spectral Clustering. Whether it's in plants and biology, medical imaging, buisness and marketing, understanding the connections between fields on Facebook, or even criminology, clustering is an extremely important part of modern data analysis. It allows people to find important subsystems or patterns inside noisy data sets. One such method*

> *is spectral clustering, which uses the eigenvalues of the graph of a network. Even the eigenvector of the second smallest eigenvalue of the Laplacian matrix allows us to find the two largest clusters in a network.*
> *Dimensionality Reduction/PCA. The principal components correspond to the largest eigenvalues of \(A^{T}A\), and this yields the least squared projection onto a smaller dimensional hyperplane, and the eigenvectors become the axes of the hyperplane. Dimensionality reduction is extremely useful in machine learning and data analysis as it allows one to understand where most of the variation in the data comes from.*
> *Low rank factorization for collaborative prediction. This is what Netflix does (or once did) to predict what rating you'll have for a movie you have not yet watched. It uses the singular value decomposition and throws away the smallest eigenvalues of \(A^{T}A\).*
> *The Google Page Rank algorithm. The largest eigenvector of the graph of the internet is how the pages are ranked.*

From: https://math.stackexchange.com/questions/1520832/real-life-examples-for-eigenvalues-eigenvectors

---

19.2: Solving Eigenproblems - A 2x2 Example

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

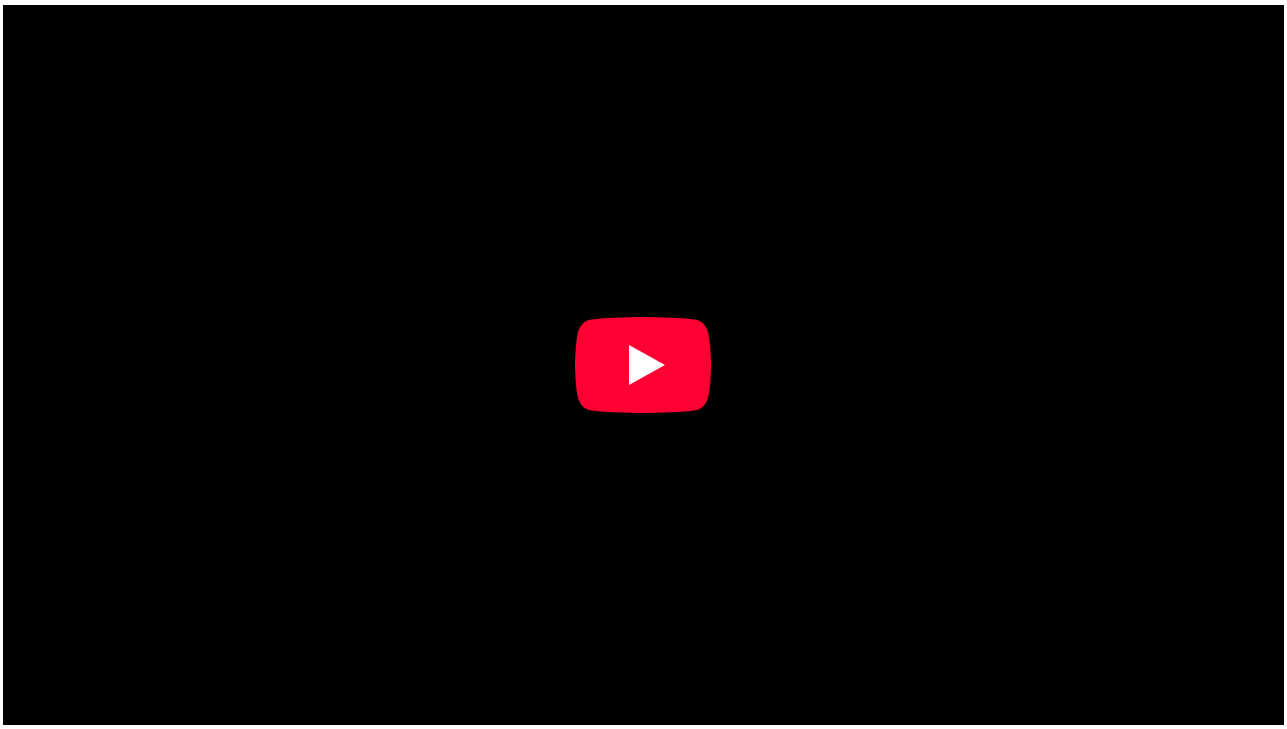
```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("0UbkMlTu1vo",width=640,height=360, cc_load_policy=True)
```



Consider calculating eigenvalues for any 2×2 matrix. We want to solve:

$$|A - \lambda I_2| = 0$$

$$\left| \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = \left| \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} \right| = 0$$

We know this determinant:

$$(a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} = 0$$

If we expand the above, we get:

$$a_{11}a_{22} + \lambda^2 - a_{11}\lambda - a_{22}\lambda - a_{12}a_{21} = 0$$

and

$$\lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{12}a_{21} = 0$$

This is a simple quadratic equation. The roots of $A\lambda^2 + B\lambda + C = 0$ can be solved using the quadratic formula:

$$\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Question

Using the above equation. What are the eigenvalues for the following 2×2 matrix. Try calculating this by hand and then store the lower value in a variable named `e1` and the larger value in `e2` to check your answer:

$$A = \begin{bmatrix} -4 & -6 \\ 3 & 5 \end{bmatrix}$$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(e1,'c54490d3480079138c8c027a87a366e3');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.float(e2,'d1bd83a33f1a841ab7fda32449746cc4');
```

Do This

Find a `numpy` function that will calculate eigenvalues and verify the answers from above.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Question

What are the corresponding eigenvectors to the matrix A? This time you can try calculating by hand or just used the function you found in the previous answer. Store the eigenvector associated with the `e1` value in a vector named `v1` and the eigenvector associated with the eigenvalue `e2` in a vector named `v2` to check your answer.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.eq_vector(v1,'35758bc2fa8ff4f04cfbcd019844f93d');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.eq_vector(v2,'90b0437e86d2cf70050d1d6081d942f4');
```

Question

Both **sympy** and **numpy** can calculate many of the same things. What is the fundamental difference between these two libraries?

19.3: Introduction to Markov Models

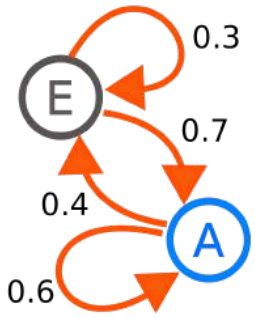
Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

> *In probability theory, a Markov model is a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it.*



A diagram representing a two-state Markov process, with the states labelled E and A. Via Wikipedia

Each number represents the probability of the Markov process changing from one state to another state, with the direction indicated by the arrow. For example, if the Markov process is in state A, then the probability it changes to state E is 0.4, while the probability it remains in state A is 0.6.

The above state model can be represented by a transition matrix.

At each time step (t) the probability to move between states depends on the previous state ($t-1$):

$$A_t = 0.6A_{(t-1)} + 0.7E_{(t-1)}$$

$$E_t = 0.4A_{(t-1)} + 0.3E_{(t-1)}$$

The above state model ($S_t = [A_t, E_t]^T$) can be represented in the following matrix notation:

$$S_t = PS_{(t-1)}$$

Do This

Create a 2×2 matrix (P) representing the transition matrix for the above Markov space.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.matrix(P,'de1c99f4b4a8d7ea541a084d836ba7e4');
```

19.4: Assignment wrap-up

Assignment-Specific Question

Both **sympy** and **numpy** can calculate many of the same things. What is the fundamental difference between these two libraries?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

20: 10 In-Class Assignment - Eigenproblems

20.0: Introduction

Image from: https://campusinvolvement.umich.edu/

Agenda for today's class (80 minutes)

1. (20 minutes) Pre Class Review
2. (20 minutes) Introduction to Markov Models
3. (20 minutes) Another Markov Model Example

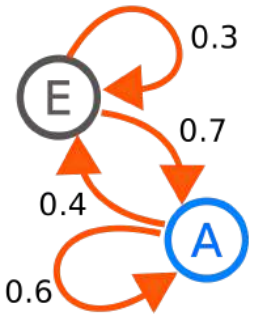
20.1: Pre Class Review

- 10 Pre-Class Assignment: Eigenvectors and Eigenvalues

20.2: Introduction to Markov Models


```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

> *In probability theory, a Markov model is a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it.*



A diagram representing a two-state Markov process, with the states labelled E and A. Via Wikipedia

Each number represents the probability of the Markov process changing from one state to another state, with the direction indicated by the arrow. For example, if the Markov process is in state A, then the probability it changes to state E is 0.4, while the probability it remains in state A is 0.6.

The above state model can be represented by a transition matrix.

$$P = \begin{matrix} \text{Current State} \\ \begin{bmatrix} p_{A\to A} & p_{E\to A} \\ p_{A\to E} & p_{E\to E} \end{bmatrix} \end{matrix} \text{Next state}$$

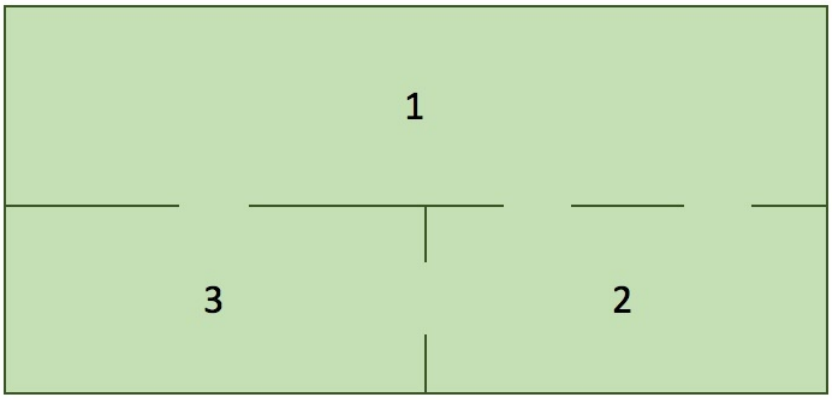
In other words we can write the above as follows

```
A = np.matrix([[0.6, 0.7],[0.4, 0.3]])
sym.Matrix(A)
```

Notice how the columns in the matrix all add to one. This is because all of the transition probabilities out of a matrix must add to 100 percent.

Now, consider the following house map with cats in each room…

At each time step, there is an equal probability of a cat staying in their current room or moving to a new room. If a cat chooses to leave a room, then there is an equal chance of that cat picking any of the doors in the room to leave.



Do This

Try to draw a Markov chain (Markov matrix) for the above system of equations. Be prepared to share your diagram with the class.

A Markov chain can be represented as a Markov transition model of the form $Ax = b$. Where A is your probability tranisition matrix (often represented as a P instead of an A). x is the state before the transition and b is the state after the transition.

Question

Generate a Markov transition model represented as a matrix P of the form:

$$P = \begin{matrix} \\ \\ \\ \text{Next Room} \end{matrix} \overset{\text{Current Room}}{\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}}$$

Where p_{ij} are probability transitions of the cat moving between rooms (from room j to room i):

```
##put your answer here
```



```
from answercheck import checkanswer

checkanswer.matrix(P,'1001a6fa07727caf8ce05226b765542c');
```

Question

Let's assume that the system starts with; 6 cats in room 1, 15 cats in room 2, and 3 cats in room 3. How many cats will be in each room after one time step (Store the values in a vector called `current_state`)?

```
#Put your answer to the above question here.
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(current_state,'98d5519be82a0585654de5eda3a7f397');
```

Question

The following code will plot the number of cats as a function of time (t). When this system converges, what is the steady state?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Define Start State
room1 = [6]
room2 = [15]
room3 = [3]

current_state = np.matrix([room1, room2, room3])

for i in range(10):
    #update Current State
    current_state = P*current_state

    #Store history for each room
    room1.append(current_state[0])
```

```
    room2.append(current_state[1])
    room3.append(current_state[2])

plt.plot(room1, label="room1");
plt.plot(room2, label="room2");
plt.plot(room3, label="room3");
plt.legend();
print(current_state)
```

Question

Calculate the eigenvalues and eigenvectors of your P transition matrix.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
##put your answer here
```

Do This

Make a new vector called `steadystate` from the eigenvector of your P matrix with a eigenvalue of 1.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Put your answer here
```

Since the `steadystate` vectors represent long term probabilities, they should sum to one (1). However, most programming libraries (ex. numpy and sympy) return "normalized" eigenvectors to length of 1 (i.e. norm(e)==1).

Do This

Correct for the normalization by multiplying the `steadystate` eigenvector by a constant such that the sum of the vector elements add to 1.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer here
```

Do This

Think about the cats problem, because one cat has to be in one of the three rooms. That means, the total number of cats will not change. If we add the number of cats at all rooms together, this number has to be the same. Therefore, if we start will 6+15+3=24 cats, there are also 24 cats at the `steadystate` . Modify the steadystate to make sure the total number of cats is 24.

Question

Why does the sum of the numbers at every stage remain the same?

20.3: Another Markov Model Example

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

> *A sports broadcaster wishes to predict how many Michigan residents prefer University of Michigan teams and how many prefer Michigan State teams. She noticed that, year after year, most people stick with their preferred team; however, about 5% of Michigan fans switch to Michigan State, and about 3% of Michigan State fans switch to Michigan each year. However, there is no noticeable difference in the state's population of 10 million's preference at large; in other words, it seems Michigan sports fans have reached a stationary distribution. What might that be?*

This problem is from Brilliant.org.

Do This

Try to draw a Markov chain for the above system of equations. Discuss your diagram with your classmate.

Question

Write a system of linear equations that represents how the populations change each year. Check your equations by writing the matrix P for the probability transitions matrix in your equations. Make sure your first row/column represents MSU and the second row/column represents UofM.

```
#Put your answer here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.vector(P,'1d3f7cbebef4b610f3b0a2d97609c81f');
```

run | restart | restart & run all

Question

Calculate the eigenvalues and eigenvectors of your P transition matrix.

```
#Put the answer to the above quesiton here.
```

run | restart | restart & run all

Question

Assuming each team starts with 500,000 fans, what is the steady state of this model? (i.e. in the long term how many Spartan and Wolverine fans will there be?).

```
#Put your answer here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.float(spartans,'06d263de629f4dbe51eafd524b69ddd9');
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.float(wolverines,'62d63699c8f7b886ec9b3cb651bba753');
```

run | restart | restart & run all

CHAPTER OVERVIEW

21: 11 Pre-Class Assignment - Vector Spaces

21.0: Introduction

Readings for this topic (Recommended in bold)

- ***Heffron Chapter 2 II pg 77-86***
- Beezer Chapter VS pg 257-269

Goals for today's pre-class assignment

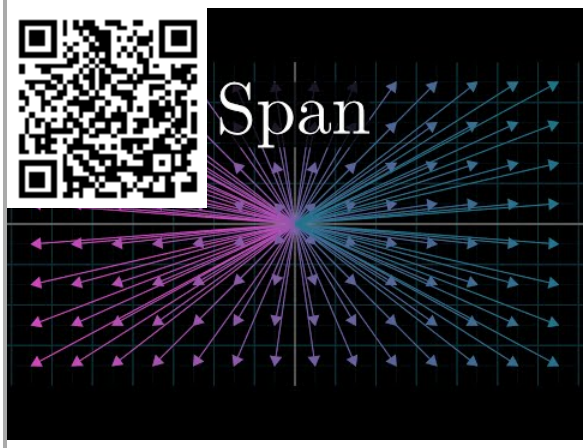
1. Basis Vectors
2. Vector Spaces
3. Lots of Things Can Be Vector Spaces
4. Assignment Wrap-up

21.1: Basis Vectors

Below is a really good review of concepts such as: Linear combinations, span, and basis vectors.

```
from IPython.display import YouTubeVideo
YouTubeVideo("k7RM-ot2NWY",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



Question

What is the technical definition of a basis?

Question

Write three basis vectors that span \(R^3\).

From the above video two terms we want you to really understand *Span* and *Linear Independent*. Understanding these two will be really important when you think about basis. Make sure you watch the video and try to answer the following questions as best you

can using your own words.

Question

Describe what it means for vectors to *Span* a space?

Question

What is the span of two vectors that point in the same direction?

Question

Can the following vectors span \(R^3\)? Why?
\((1,−2,3),(−2,4,−6),(0,6,4)\)

Question

Describe what it means for vectors to be *Linearly Independent*?

If you have vectors that *span* a space AND are *Linearly Independent* then these vectors form a *Basis* for that space.
Turns out you can create a matrix by using basis vectors as columns. This matrix can be used to change points from one basis representation to another.

21.2: Vector Spaces

Vector spaces are an abstract concept used in math. So far we have talked about vectors of real numbers (R^n). However, there are other types of vectors as well. A vector space is a formal definition. If you can define a concept as a vector space then you can use the tools of linear algebra to work with those concepts.

A **Vector Space** is a set V of elements called **vectors**, having operations of addition and scalar multiplication defined on it that satisfy the following conditions (u, v, and w are arbitrary elements of V, and c and d are scalars.)

Closure Axioms

1. The sum $u+v$ exists and is an element of V. (V is closed under addition.)
2. cu is an element of V. (V is closed under multiplication.)

Addition Axioms

1. $u+v=v+u$ (commutative property)
2. $u+(v+w)=(u+v)+w$ (associative property)
3. There exists an element of V, called a **zero vector**, denoted 0, such that $u+0=u$
4. For every element u of V, there exists an element called a **negative** of u, denoted $-u$, such that $u+(-u)=0$.

Scalar Multiplication Axioms

1. $c(u+v)=cu+cv$
2. $(c+d)u=cu+du$
3. $c(du)=(cd)u$
4. $1u=u$

21.3: Lots of Things Can Be Vector Spaces

```
from IPython.display import YouTubeVideo
YouTubeVideo("YmGWj9RrNMI",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

Consider the following two matrices \(A \in R^{3 \times 3}\) and \(B \in R^{3 \times 3} \), which consist of real numbers:

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()

a11,a12,a13,a21,a22,a23,a31,a32,a33 = sym.symbols('a_{11},a_{12}, a_{13},a_{21},a_{22}
A = sym.Matrix([[a11,a12,a13],[a21,a22,a23],[a31,a32,a33]])
```

```
A
```

run | restart | restart & run all

```
b11,b12,b13,b21,b22,b23,b31,b32,b33 = sym.symbols('b_{11},b_{12}, b_{13},b_{21},b_{22
B = sym.Matrix([[b11,b12,b13],[b21,b22,b23],[b31,b32,b33]])
B
```

run | restart | restart & run all

Question

What properties do we need to show all \(3 \times 3\) matrices of real numbers form a vector space.

Do This

Demonstrate these properties using sympy as was done in the video.

```
#Put your answer here.
```

run | restart | restart & run all

Question (assignment specific)

Determine whether \(A\) is a linear combination of \(B\), \(C\), and \(D\)?

\[\begin{split} A=
\left[
\begin{matrix}
7 & 6 \\
-5 & -3
\end{matrix}
\right],
B=
\left[
\begin{matrix}
3 & 0 \\
1 & 1
\end{matrix}
\right],
C=
\left[
\begin{matrix}
0 & 1 \\
3 & 4
\end{matrix}
\right],
D=
\left[
\begin{matrix}
1 & 2 \\
0 & 1
\end{matrix}
\right]
\end{split} \nonumber \]

```
#Put your answer to the above question here
```

run | restart | restart & run all

Question

Write a basis for all \(2 \times 3\) matrices and give the dimension of the space.

21.4: Assignment wrap-up

Assignment-Specific Question

Is matrix A is a linear combination of B, C, and D from above?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

22: 11 In-Class Assignment - Vector Spaces

22.0: Introduction

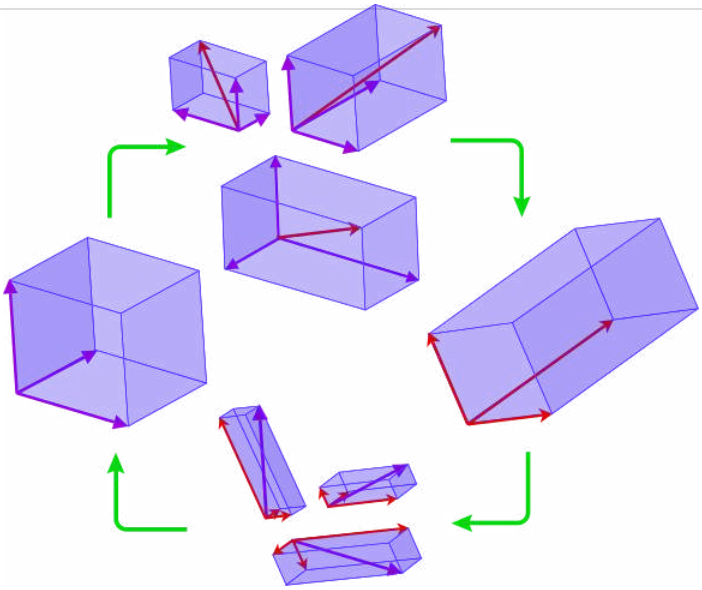


Image from: https://en.Wikipedia.org/wiki/Change_of_basis

Agenda for today's class (80 minutes)

1. (20 minutes) Review Pre-Class Assignment
2. (20 minutes) Introduction to subspaces
3. (20 minutes) Basis Vectors
4. (20 minutes) Vector Spaces

22.1: Review Pre-class Assignment

Like most days, your instructor will review and answer questions from your pre-class assignment. This is your opportunity to ask any lingering questions before the quiz.

- 11 Pre-Class Assignment: Vector Spaces

22.2: Introduction to subspaces

22.3: Basis Vectors

Consider the following example. We claim that the following set of vectors form a basis for R^3:

$$B = \{(2,1,3),(-1,6,0),(3,4,-10)\}$$

If these vectors form a basis they must be *linearly independent* and *Span* the entire space of R^3

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve
sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Do This

Create a 3×3 numpy matrix A where the columns of A form are the basis vectors.

```
#Put your answer to the above question here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A,'68b81f1c1041158b519936cb1a2e4d6b');
```

run | restart | restart & run all

Do This

Using python, calculate the determinant of matrix A.

```
# Put your answer to the above question here.
```

run | restart | restart & run all

Do This

Using python, calculate the inverse of A.

```
# Put your answer to the above question here.
```

run | restart | restart & run all

Do This

Using python, calculate the rank of A.

```
# Put your answer to the above question here.
```

run | restart | restart & run all

Do This

Using python, calculate the reduced row echelon form of A.

```
# Put your answer to the above question here.
```

run | restart | restart & run all

Do This

Using the above A and the vector $b = (1, 3, 2)$. What is the solution to $Ax = b$?

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(x,'8b0938260dfeaafc9f8e9fec0bc72f17');
```

run | restart | restart & run all

Turns out a matrix where column vectors are formed from basis vectors a lot of interesting properties and the following statements are equivalent.

- The column vectors of A form a basis for R^n
- $|A| \neq 0$
- A is invertible.
- A is row equivalent to I_n (i.e. it's reduced row echelon form is I_n)
- The system of equations $Ax = b$ has a unique solution.
- $rank(A) = n$

Not all matrices follow the above statements but the ones that do are used throughout linear algebra so it is important that we know these properties.

22.4: Vector Spaces

A **Vector Space** is a set V of elements called **vectors**, having operations of addition and scalar multiplication defined on it that satisfy the following conditions (u, v, and w are arbitrary elements of V, and c and d are scalars.)

Closure Axioms

1. The sum $u+v$ exists and is an element of V. (V is closed under addition.)
2. cu is an element of V. (V is closed under multiplication.)

Addition Axioms

1. $u+v=v+u$ (commutative property)
2. $u+(v+w)=(u+v)+w$ (associative property)
3. There exists an element of V, called a **zero vector**, denoted 0, such that $u+0=u$
4. For every element u of V, there exists an element called a **negative** of u, denoted $-u$, such that $u+(-u)=0$.

Scalar Multiplication Axioms

1. $c(u+v)=cu+cv$
2. $(c+d)u=cu+du$
3. $c(du)=(cd)u$
4. $1u=u$

Definition of a basis of a vector space

A finite set of vectors $v_1,\ldots,v_n$ is called a basis of a vector space V if the set spans V and is linearly independent. i.e. each vector in V can be expressed uniquely as a linear combination of the vectors in a basis.

Vector spaces

Do This

Let U be the set of all circles in R^2 having center at the origin. Interpret the origin as being in this set, i.e., it is a circle center at the origin with radius zero. Assume C_1 and C_2 are elements of U. Let C_1+C_2 be the circle centered at the origin, whose radius is the sum of the radii of C_1 and C_2. Let kC_1 be the circle center at the origin, whose radius is $|k|$ times that of C_1. Determine which vector space axioms hold and which do not.

Spans:

Do This

Let v, v_1, and v_2 be vectors in a vector space V. Let v be a linear combination of v_1 and v_2. If c_1 and c_2 are nonzero real numbers, show that v is also a linear combination of c_1v_1 and c_2v_2.

Do This

Let v_1 and v_2 span a vector space V. Let v_3 be any other vector in V. Show that v_1, v_2, and v_3 also span V.

Linear Independent:

Consider the following matrix, which is in the reduced row echelon form.

$$\begin{bmatrix}1&0&0&7\\0&1&0&4\\0&0&1&3\end{bmatrix}$$

Do This

Show that the row vectors form a linearly independent set:

Do This

Is the set of nonzero row vectors of any matrix in reduced row echelon form linearly independent? Discuss in your groups and include your thoughts below.

Do This

A computer program accepts a number of vectors in R^3 as input and checks to see if the vectors are linearly independent and outputs a True/False statment. Discuss in your groups, which is more likely to happen due to round-off error–that the computer states that a given set of linearly independent vectors is linearly dependent, or vice versa? Put your groups thoughts below.

CHAPTER OVERVIEW

23: 12 Pre-Class Assignment - Matrix Spaces

23.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 2 III pg 114-134
- ***Beezer Subsection CBM pg 549-549***
- Boyd Section 5.1 pg 91-95

Goals for today's pre-class assignment

1. Properties of Invertible Matrices
2. The Basis of a Vector Space
3. Change of Basis
4. Assignment wrap-up

23.1: Review the Properties of Invertible Matrices

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve
sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Let A be an $n \times n$ matrix. The following statements are equivalent.

- The column vectors of A form a basis for R^n
- $|A| \neq 0$
- A is invertible.
- A is row equivalent to I_n (i.e. it's reduced row echelon form is I_n)
- The system of equations $Ax = b$ has a unique solution.
- $rank(A) = n$

Consider the following example. We claim that the following set of vectors form a basis for R^3:

$$B = \{(2, 1, 3), (-1, 6, 0), (3, 4, -10)\}$$

Remember for these two vectors to be a basis they need to obay the following two properties:

1. They must span R^3.
2. They must be linearly independent.

Using the above statements we can show this is true in multiple ways.

The column vectors of A form a basis for R^n

Do This

Define a numpy matrix `A` consisting of the vectors B as columns:

```
#Put your answer to the above question here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A,'94827a40ec59c7d767afe6841e1723ce');
```

run | restart | restart & run all

$|A| \neq 0$

Do This

The first in the above properties tell us that if the vectors in B are truly a basis of R^3 then $|A| = 0$. Calculate the determinant of A and store the value in `det` .

```
#Put your answer to the above question here
```

run restart restart & run all

```
#Verify that the determinate is in fact zero
if np.isclose(det,0):
    print("Since the Determinate is zero the column vectors do NOT form a Basis")
else:
    print("Since the Determinate is non-zero then the column vectors form a Basis.")
```

run restart restart & run all

A is invertible.

Do This

Since the determinant is non-zero we know that there is an inverse to A. Use python to calculate that inverse and store it in a matrix called `A_inv`

```
#put your answer to the above question here
```

run restart restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(A_inv,'001aaddd4824f42ad9d2ccde21cf9d24');
```

run restart restart & run all

A is row equivalent to I_n (i.e. it's reduced row echelon form is I_n)

Do This

According to the property above the reduced row echelon form of an invertable matrix is the Identiy matrix. Verify using the python `sympy` library and store the reduced row echelone matrix in a variable called `rref` if you really need to check it.

```
#put your answer to the above question here
```

run restart restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(rref,'cde432847c1c4b6d17cd7bfacc457ed1');
```

run restart restart & run all

The system of equations $Ax = b$ has a unique solution.

Let us assume some arbitrary vector $b \in R^n$. According to the above properties it should only have one solution.

Do This

Find the solution to $Ax = b$ for the vector $b = (-10, 200, 3)$. Store the solution in a variable called `x`

```
from answercheck import checkanswer

checkanswer.vector(x,'161cfd16545b1b5fb13e35d2800f13df');
```

run restart restart & run all

$rank(A) = n$

The final property says that the rank should equal the dimension of R^n. In our example $n = 3$. Find a `python` function to calculate the rank of A. Store the value in a variable named `rank` to check your answer.

```
#Put your answer to the above quesion here
```

run | restart | restart & run all

```
#Verify that the determinate is in fact zero
if np.isclose(rank,3):
    print("Rank is 3")
else:
    print("Rank is not 3. Did we do something wrong?")
```

run | restart | restart & run all

Question (assignment-specific)

Without doing any calculations (i.e. only using the above properties), how many solutions are there to $Ax = 0$? What is(are) the solution(s)?

23.2: The Basis of a Vector Space

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve
sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

Let U be a vector space with basis $B = \{u_1, \ldots, u_n\}$, and let u be a vector in U. Because a basis "spans" the vector space, we know that there exists scalars $a_1, \ldots, a_n$ such that:

$$u = a_1 u_1 + \cdots + a_n u_n$$

Since a basis is a linearly independent set of vectors we know the scalars $a_1, \ldots, a_n$ are unique.

The values $a_1, \ldots, a_n$ are called the **coordinates of u** relative to the basis (B) and is typically written as a column vector:

$$u_B = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

We can create a *transition matrix* P using the inverse of the matrix with the basis vectors being columns.

$$P = [u_1 \ldots u_n]^{-1}$$

Now we will show that matrix P will transition vector u in the standard coordinate system to the coordinates relative to the basis B:

$$u_B = Pu$$

EXAMPLE: Consider the vector $u = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$ and the basis vectors $B = \{(1, 2), (3, -1)\}$. The following code calculates the P *transition matrix* from B and then uses P to calculate the values of u_B (a_1 and a_2):

```
u = np.matrix([[5],[3]])
sym.Matrix(u)
```



```
B = np.matrix([[1,2],[3,-1]]).T
sym.Matrix(B)
```

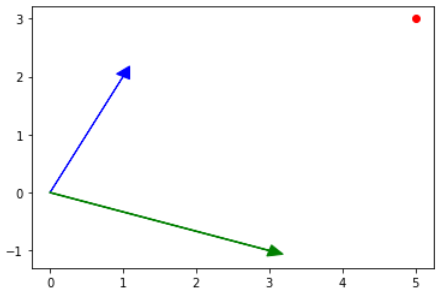


```
P = np.linalg.inv(B)
ub = P*u

sym.Matrix(ub)
```

Here we would like to view this from R^n. Let $B = [u_1 \dots u_n]$, then the values of u_B can be found by solving the linear system $u = Bu_b$. The columns of B are a basis, therefore, the matrix B is an $n \times n$ square matrix and it has an inverse. Therefore, we can solve the linear system and obtain $u_B = B^{-1}u = Pu$.

Let's try to visualize this with a plot:

```
ax = plt.axes();


#Blue arrow representing first Basis Vector
ax.arrow(0, 0, B[0,0],B[1,0], head_width=.2, head_length=.2, fc='blue', ec='blue');


#Green arrow representing Second Basis Vector
plt.plot([0,B[0,1]],[0,B[1,1]],color='green'); #Need this line to make the figure wor
ax.arrow(0, 0, B[0,1],B[1,1], head_width=.2, head_length=.2, fc='green', ec='green');

#Original point u as a red dot
ax.scatter(u[0,0],u[1,0], color='red');

plt.show()
#plt.axis('equal');
```



Notice that the blue arrow represents the first basis vector and the green arrow is the second basis vector in B. The solution to u_B shows 2 units along the blue vector and 1 units along the green vector, which puts us at the point (5,3).

This is also called a change in coordinate systems.

Question

What is the coordinate vector of u relative to the given basis B in R^3?

$$u = (9, -3, 21)$$

$$B = \{(2, 0, -1), (0, 1, 3), (1, 1, 1)\}$$

Store this coordinate in a variable `ub` for checking:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer here
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(ub,'f72f62c739096030e0074c4e1dfca159');
```

Let's look more closely into the matrix P, what is the meaning of the columns of the matrix P?

We know that P is the inverse of B, therefore, we have $BP = I$. Then we can look at the first column of the P, say p_1, we have that Bp_1 is the column vector $(1, 0, 0)$, which is exactly the first component from the standard basis. This is true for other columns.

It means that if we want to change an old basis B to a new basis B', we need to find out all the coordinates in the new basis for the old basis, and the transition matrix is by putting all the coordinates as columns.

Here is the matrix B again:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
B = np.matrix([[2,0,-1],[0,1,3],[1,1,1]]).T
sym.Matrix(B)
```

The first column of P should be the solution to $Bx = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$. We can use the `numpy.linalg.solve` function to find this solution:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# The first column of P should be
u1 = np.matrix([1,0,0]).T
p1 = np.linalg.solve(B,u1)
p1
```

We can find a similar answer for columns p_2 and p_3:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# The second column of P should be
u2 = np.matrix([0,1,0]).T
p2 = np.linalg.solve(B,u2)
p2
```

```
# The third column of P should be
u3 = np.matrix([0,0,1]).T
p3 = np.linalg.solve(B,u3)
p3
```



```
# concatenate three column together into a 3x3 matrix
P = np.concatenate((p1, p2, p3), axis=1)
sym.Matrix(P)
```



```
# Find the new coordinate in the new basis
u = np.matrix([9,-3,21]).T
```

```
UB = P*u
print(UB)
```

This should be basically the same answer as you got above.

23.3: Change of Basis

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve
sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Now consider the following two bases in R^2:

$$B_1 = \{(1,2),(3,-1)\}$$

$$B_2 = \{(3,1),(5,2)\}$$

The transformation from the "standard basis" to B_1 and B_2 can be defined as the column vectors P_1 and P_2 as follows:

```
B1 = np.matrix([[1,2],[3,-1]]).T
P1 = np.linalg.inv(B1)

sym.Matrix(P1)
```

run | restart | restart & run all

```
B2 = np.matrix([[3,1],[5,2]]).T
P2 = np.linalg.inv(B2)

sym.Matrix(P2)
```

run | restart | restart & run all

Do This

Find the transition matrix T that will take points in the B_1 coordinate representation and put them into B_2 coordinates. *NOTE* this is analogous to the robot kinematics problem. We want to represent points in a different coordinate system.

```
# Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(T,'dcc03ddff982e29eea6dd52ec9088986')
```

run | restart | restart & run all

Question

Given $u_{B_1} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ (a point named u in the B_1 coordinate system) and your calculated transition matrix T, what is the same point expressed in the B_2 basis (i.e. what is u_{B2})? Store your answer in a variable named ub2 for checking.

```
ub1 = np.matrix([[2],[1]])
sym.Matrix(ub1)
```

run | restart | restart & run all

```
##Put your code here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.vector(ub2,'9a5fe29254c07cf59ebdffcaba679917')
```

run | restart | restart & run all

There are three bases B_1, B_2, and B_3. We have the transition matrix P_{12} from B_1 to B_2 and the transition matrix P_{23} from B_2 to B_3. In R^n, we can compute the transition matrix as $P_{12} = B_2^{-1}B_1, \quad P_{23} = B_3^{-1}B_2$.
Then we can find all other transition matrices.
$P_{13} = B_3^{-1}B_1 = B_3^{-1}B_2 * B_2^{-1}B_1 = P_{23}P_{12}$
$P_{21} = B_1^{-1}B_2 = (B_2^{-1}B_1)^{-1} = P_{12}^{-1}$
$P_{32} = B_2^{-1}B_3 = (B_3^{-1}B_2)^{-1} = P_{23}^{-1}$
$P_{31} = B_1^{-1}B_3 = (B_3^{-1}B_1)^{-1} = P_{13}^{-1} = (P_{23}P_{12})^{-1} = P_{12}^{-1}P_{23}^{-1}$
The result is true for general vector spaces and can be extended to many bases.

23.4: Assignment wrap-up

Assignment-Specific Question

Without doing any calculations (i.e. only using the above properties), how many solutions are there to $Ax = 0$? What is(are) the solution(s)?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

24: 12 In-Class Assignment - Matrix Representation

24.0: Introduction

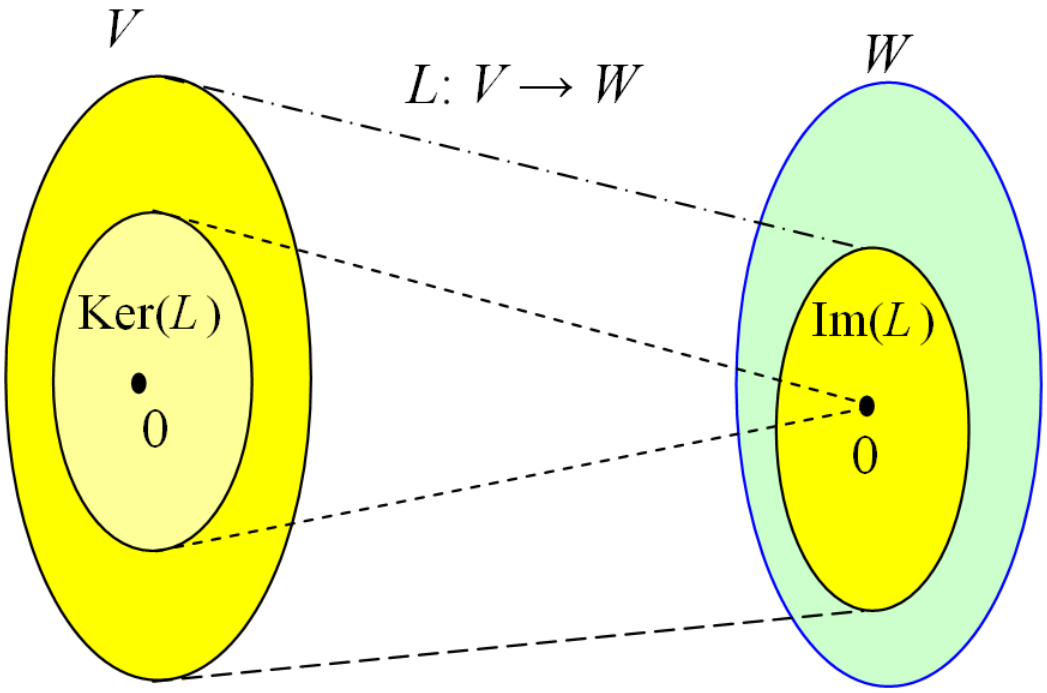


Image from: Wikipedia

Agenda for today's class (80 minutes)

1. (20 minutes) Review Pre-class assignment
2. (30 minutes) Matrix Representation of Vector Spaces
3. (30 minutes) Practice Example

24.1: Review Pre-class assignment

- 12 Pre-Class Assignment: Matrix Spaces

24.2: Matrix Representation of Vector Spaces

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve
sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Consider the following matrix A.

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 1 & 1 & 8 \end{bmatrix}$$

Question

What is the reduced row echelon form of A?

```
# Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(rref,'1731818a1555cc33a778a4eb76af945c');
```

run | restart | restart & run all

ROW SPACE The first and second (non zero) rows of the above matrix "spans" the same space as the orignal three row vectors in A. We often call this the "row space" and it can be written as a linear combination of the non-zero rows of the reduced row echelon form:

$$row(A) = r(1,0,3)^\top + s(0,1,5)^\top$$

Do This

Calculate the solutions to the system of homogeneous equations $Ax = 0$. This is often called the NULL SPACE or sometimes KERNEL of A.

```
#Put your answer here
```

run | restart | restart & run all

Do This

We introduced two subspaces. Pick one vector from the row space of A and another vector from the null space of A. Find the dot product of these two vector.

```
#Put your answer here
```

run | restart | restart & run all

Question

Did you get the same value for the dot product? Explain your answer.

Do This

What is the reduced row echelon form of A^T?

```
#Put your answer here
```

run | restart | restart & run all

COLUMN SPACE: The first and second (non zero) rows of the above matrix "spans" the same space as the original three column vectors in A. We often call this the "column space" (or "image space") of A and it can be written as a linear combination of the non-zero rows of the reduced row echelon form of A^T:

$$col(A) = a(1,0,1)^\top + b(0,1,1)^\top$$

Do This

Calculate the solutions to the system of homogeneous equations $A^T x = 0$. This is often called the NULL SPACE of A^T.

```
#Put your answer here
```

run | restart | restart & run all

Example #1

Consider the following system of linear equations.

$$x_1 - x_2 + x_3 = 3$$

$$-2x_1 + 2x_2 - 2x_3 = -6$$

Do This

What are the solutions to the above system of equations?

```
# Put your code here
```

run | restart | restart & run all

Do This

Come up with a specific arbitrary solution (any solution will do) to the above set of equations.

Do This

Now consider only the left hand side of the above matrix and solve for the kernel (null Space) of A:

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -2 & 2 & -2 \end{bmatrix}$$

```
#Put your answer here
```

run | restart | restart & run all

Do This

Express an arbitrary solution as the sum of an element of the kernel of the transformation defined by the matrix of coefficients and a particular solution.

Do This

Discuss in your group and the class your solution from above. How does the solution to $Ax = b$ relate to the solution to $Ax = 0$. If you were to plot all solutions, what shape does it take? How does this shape relate to the kernel?

24.3: Practice Nutrition

Big Annie's Mac and Cheese fans want to improve the levels of protein and fiber for lunch by adding broccoli and canned chicken. The nutrition information for the foods in this problem are

Nutrient	Mac and Cheese	Broccoli	Chicken	Shells and White Cheddar
Calories	270	51	70	260
Protein (g)	10	5.4	15	9
Fiber (g)	2	5.2	0	5

She wants to achieve the goals with exactly 400 calories, 30 g of protein, and 10 g of fiber by choosing the combination of these three or four serving. (Assume that we can have non-integer proportions for each serving.)

Question (a)

We consider all four choices of food together. Formulate the problem into a system of equations $Ax = b$. Create your matrix A and the column vector b in `np.matrix` .

```
import numpy as np
#####Start your code here #####
A = np.matrix()
b = np.matrix()
#####End of your code here#####
```

run | restart | restart & run all

Question (b)

In this and next question, we only consider three out of the four choices. What proportions of these servings of the three food (Mac and Cheese, Broccoli, and Chicken) should be used to meet the goal? (Hint: formulate it as a system of equations and solve it).

```
#Put your answer here
```

run | restart | restart & run all

Question (c)

She found that there was too much broccoli in the proportions from part (b), so she decided to switch from classical Mac and Cheese to Annie's Whole Wheat Shells and White Cheddar. What proportions of servings of the new three food should she use to meet the goals?

```
#Put your answer here
```

run | restart | restart & run all

Question (d)

Based on the solutions to parts (b) and (c), what are the possible proportions of serving for the four food that meet the goal.

Question (e)

Solve the system of equations from part (a). You need to first decide the three outcomes: One solution, None solution, Infinite many solutions. Then for *One solution*, write down the solution; for *Infinite many solutions*, write down all the solutions with free variables.

```
#Put your answer here
```

run | restart | restart & run all

CHAPTER OVERVIEW

25: 13 Pre-Class Assignment - Projections

25.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Section VI pg 267-275
- Beezer Subsections OV-GSP pg 154-161
- ***Boyd Section 5.3-5.4 pg 95-102***

Goals for today's pre-class assignment

1. Orthogonal and Orthonormal
2. Code Review
3. Gram-Schmidt
4. Assignment Wrap-up

25.1: Orthogonal and Orthonormal

Definition

A set of vectors is said to be orthogonal if every pair of vectors in the set is orthogonal (the dot product is 0). The set is orthonormal if it is orthogonal and each vector is a unit vector (norm equals 1).
Result: An orthogonal set of nonzero vectors is linearly independent.

Definition

A basis that is an orthogonal set is called an orthogonal basis. A basis that is an orthonormal set is called an orthonormal basis.
Result: Let $\{u_1, \ldots, u_n\}$ be an orthonormal basis for a vector space V. Then for any vector v in V, we have $v = (v \cdot u_1)u_1 + (v \cdot u_2)u_2 + \cdots + (v \cdot u_n)u_n$.

Definition

A *square* matrix is orthogonal if $A^{-1} = A^{\top}$.
Result: Let A be a square matrix. The following three statements are equivalent.

1. A is orthogonal.
2. The column vectors of A form an orthonormal set.
3. The row vectors of A form an orthonormal set.
4. A^{-1} is orthogonal.
5. $A^{\top}$ is orthogonal.

Result: If A is an orthogonal matrix, then we have $|A| = \pm 1$.

Consider the following vectors u_1, u_2, and u_3 that form a basis for R^3.

$$u_1 = (1, 0, 0)$$

$$u_2 = (0, \frac{1}{\sqrt{(2)}}, \frac{1}{\sqrt{(2)}})$$

$$u_3 = (0, \frac{1}{\sqrt{(2)}}, -\frac{1}{\sqrt{(2)}})$$

Do This

Show that the vectors u_1, u_2, and u_3 are linearly independent
(HINT: see the pre-class for 12 Pre-Class Assignment - Matrix Spaces).

Question 1

How do you show that u_1, u_2, and u_3 are orthogonal?

Question 2

How do you show that u_1, u_2, and u_3 are normal vectors?

Do This

Express the vector $v = (7, 5, -1)$ as a linear combination of the u_1, u_2, and u_3 basis vectors:

```
# Put your answer here
```

run | restart | restart & run all

25.2: Code Review

In the next in-class assignment, we are going to avoid some of the more advanced libraries (i.e. no `numpy` or `scipy` or `sympy`) to try to get a better understanding about what is going on in the math. The following code implements some common linear algebra functions:

```
#Standard Python Libraries only
import math
import copy
```

run | restart | restart & run all

```
def dot(u,v):
    '''Calculate the dot product between vectors u and v'''
    temp = 0;
    for i in range(len(u)):
        temp += u[i]*v[i]
    return temp
```

run | restart | restart & run all

Do This

Write a quick test to compare the output of the above `dot` function with the `numpy` dot function.

```
# Put your test code here
```

run | restart | restart & run all

```
def multiply(m1,m2):
    '''Calculate the matrix multiplication between m1 and m2 represented as list-of-l
    n = len(m1)
    d = len(m2)
    m = len(m2[0])

    if len(m1[0]) != d:
        print("ERROR - inner dimentions not equal")

    #make zero matrix
    result = [[0 for j in range(m)] for i in range(n)]
#    print(result)
    for i in range(0,n):
        for j in range(0,m):
            for k in range(0,d):
                #print(i,j,k)
                #print('result', result[i][j])
                #print('m1', m1[i][k])
                #print('m2', m2[k][j])
                result[i][j] = result[i][j] + m1[i][k] * m2[k][j]
    return result
```

run | restart | restart & run all

Do This

Write a quick test to compare the output of the above `multiply` function with the `numpy` multiply function.

```
# Put your test code here
```

run | restart | restart & run all

Question

What is the big-O complexity of the above `multiply` function?

Question

Line 11 in the provided `multiply` code initializes a matrix of the size of the output matrix as a list of lists with zeros. What is the big-O complexity of line 11?

```
def norm(u):
    '''Calculate the norm of vector u'''
    nm = 0
    for i in range(len(u)):
        nm += u[i]*u[i]
    return math.sqrt(nm)
```

run | restart | restart & run all

Do This

Write a quick test to compare the outputs of the above `norm` function with the `numpy` norm function.

```
#Put your test code here
```

run | restart | restart & run all

```
def transpose(A):
    '''Calculate the transpose of matrix A represented as list of lists'''
    n = len(A)
    m = len(A[0])
    AT = list()
    for j in range(0,m):
        temp = list()
        for i in range(0,n):
            temp.append(A[i][j])
        AT.append(temp)
    return AT
```

run | restart | restart & run all

Do This

Write a quick test to compare the output of the above `transpose` function with the `numpy` transpose function.

```
# Put your test code here
```

run | restart | restart & run all

Question

What is the big-O complexity of the above `transpose` function?

Question

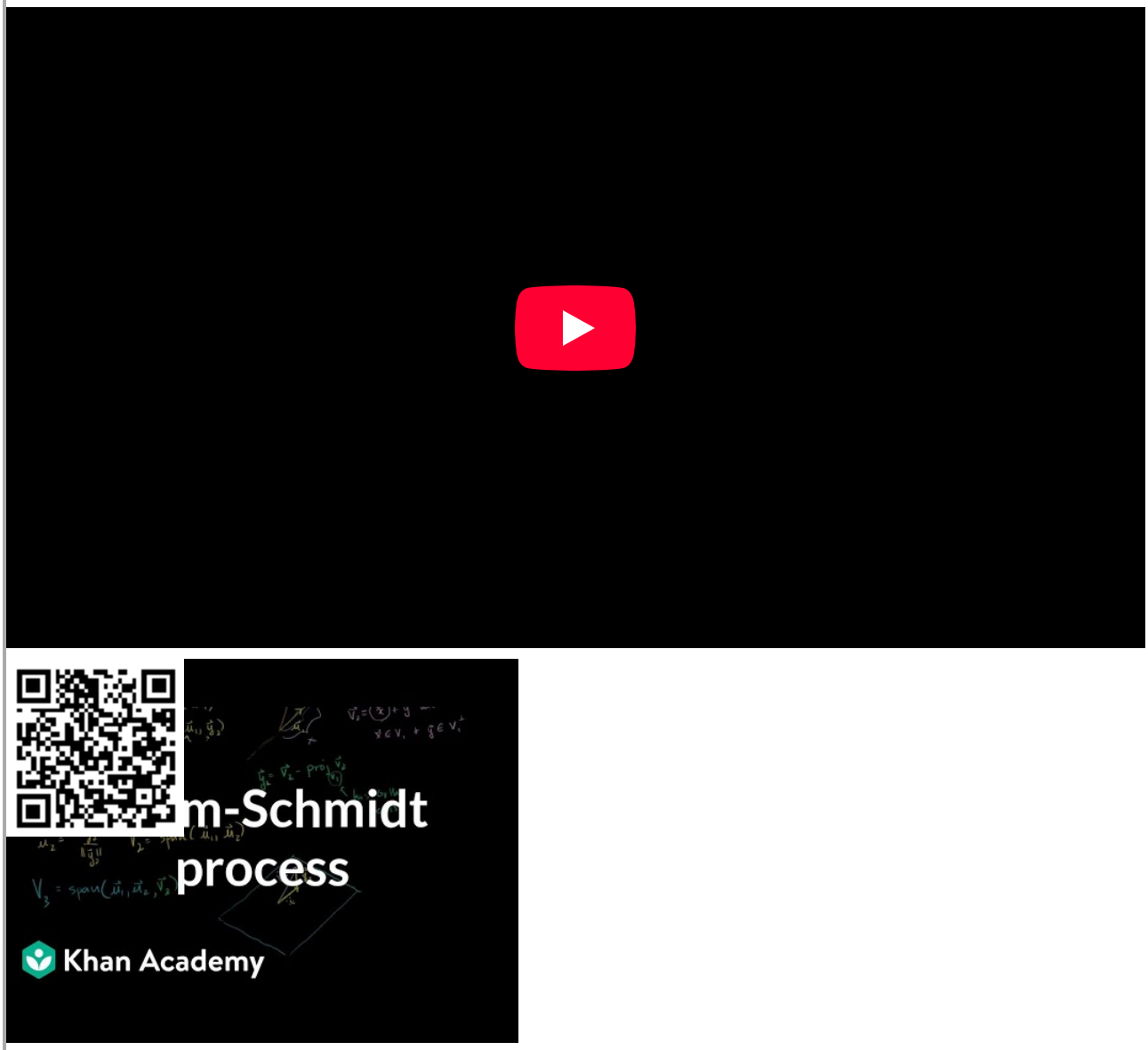
Explain any differences in results between the provided functions and their `numpy` counterparts.

25.3: Gram-Schmidt

Watch this video for the indroduction of Gram-Schmidt, which we will implement in class.

```
from IPython.display import YouTubeVideo
YouTubeVideo("rHonltF77zI",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



25.4: Assignment wrap-up

Assignment-Specific Question

How do you show that u_1, u_2, and u_3 are orthogonal?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

26: 13 In-Class Assignment - Projections

26.0: Introduction

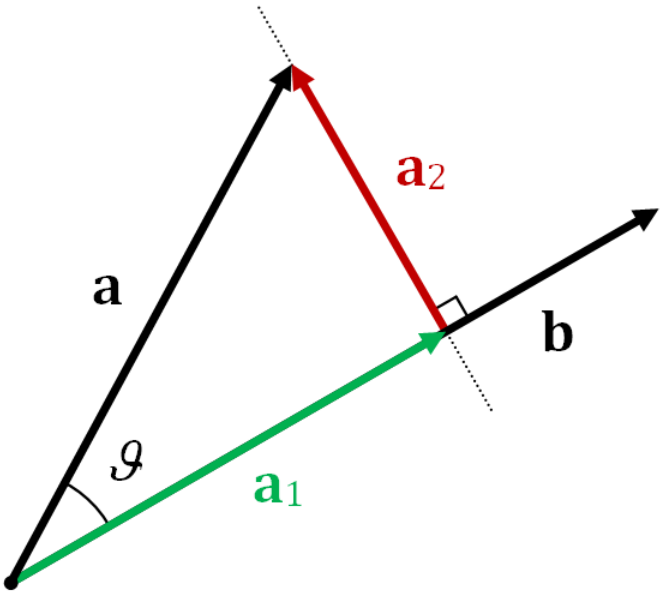


Image from: Wikipedia

Agenda for today's class (80 minutes)

1. (20 minutes) Pre-Class Review
2. (30 minutes) Understanding Projections with Code
3. (30 minutes) Gram-Schmidt Orthoganalization Process

26.1: Pre-class Review

- 13 Pre-Class Assignment: Projections

26.2: Understanding Projections With Code

In this in-class assignment, we are going to avoid some of the more advanced libraries ((i.e. no `numpy` or `scipy` or `sympy`) to try to get a better understanding about what is going on in the math. The following code implements some common linear algebra functions:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Standard Python Libraries only
import math
import copy

from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
def dot(u,v):
    '''Calculate the dot product between vectors u and v'''
    temp = 0;
    for i in range(len(u)):
        temp += u[i]*v[i]
    return temp
```

```
def multiply(m1,m2):
    '''Calculate the matrix multiplication between m1 and m2 represented as list-of-l
    n = len(m1)
    d = len(m2)
    m = len(m2[0])

    if len(m1[0]) != d:
        print("ERROR - inner dimentions not equal")
    result = [[0 for i in range(n)] for j in range(m)]
    for i in range(0,n):
        for j in range(0,m):
            for k in range(0,d):
                result[i][j] = result[i][j] + m1[i][k] * m2[k][j]
    return result
```



```
def add_vectors(v1,v2):
    v3 = []
    for i in range(len(v1)):
        v3.append(v1[i]+v2[i])
    return v3
```

```
def sub_vectors(v1,v2):
    v3 = []
    for i in range(len(v1)):
        v3.append(v1[i]-v2[i])
    return v3
```



```
def norm(u):
    '''Calculate the norm of vector u'''
    nm = 0
    for i in range(len(u)):
        nm += u[i]*u[i]
    return math.sqrt(nm)
```



```
def transpose(A):
    '''Calculate the transpose of matrix A represented as list of lists'''
```

```
    n = len(A)
    m = len(A[0])
    AT = list()
    for j in range(0,m):
        temp = list()
        for i in range(0,n):
            temp.append(A[i][j])
        AT.append(temp)
    return AT
```

Projection function

Do This

Write a function that projects vector v onto vector u. Do not use the numpy library. Instead use the functions provided above:

$$\text{proj}_u v = \frac{v \cdot u}{u \cdot u} u$$

Make sure this function will work for any size of v and u.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
def proj(v,u):
    ## Put your code here
    return pv
```

Let's test your function. Below are two example vectors. Make sure you get the correct answers. You may want to test this code with more than one set of vectors.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
u = [1,2,0,3]
v = [4,0,5,8]
print(proj(u,v))
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.vector(proj(u,v),'53216508af49c616fa0f4e9676ce3b9d');
```

Visualizing projections

Do This

See if you can design and implement a small function that takes two vectors (a and b) as inputs and generates a figure similar to the one above.

I.e. a black line from the origin to "b", a black line from origin to "a"; a green line showing the "a" component in the "b" direction and a red line showing the "a" component orthogonal to the green line. Also see section titled "Projection of One Vector onto Another Vector" in Section 4.6 on page 258 of the book.

When complete, show your solution to the instructor.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
%matplotlib inline
import matplotlib.pylab as plt
```

```
b = [3,2]
a = [2,3]

def show_projection(a,b):
    plt.plot([0,a[0]], [0,a[1]], color='black')
    plt.annotate('b', b,
            xytext=(0.9, 0.7), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            horizontalalignment='right', verticalalignment='top')
    plt.annotate('a', a,
            xytext=(0.7, 0.95), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            horizontalalignment='right', verticalalignment='top')
    plt.plot([0,b[0]], [0,b[1]], color='black')

#Finish your code here

    plt.axis('equal')

x = show_projection(a,b) ;
```

26.3: Gram-Schmidt Orthogonalization Process

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
import math
import copy
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');

def transpose(A):
    '''Calculate the transpose of matrix A represented as list of lists'''
    n = len(A)
    m = len(A[0])
    AT = list()
    for j in range(0,m):
        temp = list()
        for i in range(0,n):
            temp.append(A[i][j])
        AT.append(temp)
    return AT
```

Do This

Implement the Gram-Schmidt orthoganalization process from the Hefron textbook (page 282). This function takes a $m \times n$ Matrix A with linearly independent columns as input and return a $m \times n$ Matrix G with orthogonal column vectors. The basic algorithm works as follows:

- `AT = transpose(A)` (this process works with the columns of the matrix so it is easier to work with the transpose. Think about a list of list, it is easy to get a row (a list)).
- Make a new empty list of the same size as `AT` and call it `GT` (G transpose)
- Loop index `i` over all of the rows in AT (i.e. columns of A)
  - `GT[i] = AT[i]`
  - Loop index `j` from 0 to `i`
    - `GT[i] -= proj(GT[i], GT[j])`
- `G = transpose(GT)`

Use the following function definition as a template:

```
def GramSchmidt(A):
    return G
```

Here, we are going to test your function using the vectors:


```
A4 = [[1,4,8],[2,0,1],[0,5,5],[3,8,6]]
print(transpose(A4))
G4 = GramSchmidt(A4)
print(transpose(G4))
```

```
from answercheck import checkanswer

checkanswer.matrix(G4,'a472a81eef411c0df03ae9a072dfa040');
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
A2 = [[-4,-6],[3,5]]
print(transpose(A2))
G2 = GramSchmidt(A2)
print(transpose(G2))
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer

checkanswer.matrix(G2,'23b9860b72dbe5b84d7c598c08af9688');
```

Question

What is the Big-O complexity of the Gram-Schmidt process?

CHAPTER OVERVIEW

27: 14 Pre-Class Assignment - Fundamental Spaces

27.0: Introduction

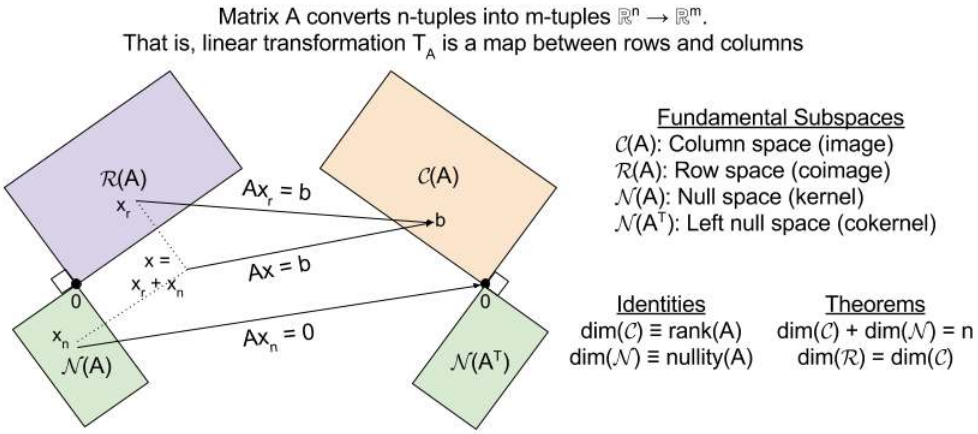


Image from: https://kevinbinz.com/2017/02/20/linear-algebra/

Readings for this topic (Recommended in bold)

- ***Heffron Section VI.3 pg 277-283***

Goals for today's pre-class assignment

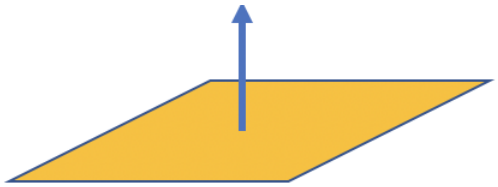
1. Orthogonal Complement
2. The Four Fundamental Spaces
3. Independent Learning
4. Assignment wrap-up

27.1: Orthogonal Complement

Definition

A vector u is **orthogonal to a subspace** W of R^n if u is orthogonal to any w in W ($u \cdot w = 0$ for all $w \in W$).

For example, consider the following figure, if we consider the plane to be a subspace then the perpendicular vector comming out of the plane is is orthoginal to any vector in the plane:



Definition

The **orthogonal complement** of W is the set of all vectors that are orthogonal to W. The set is denoted as $W_\perp$.

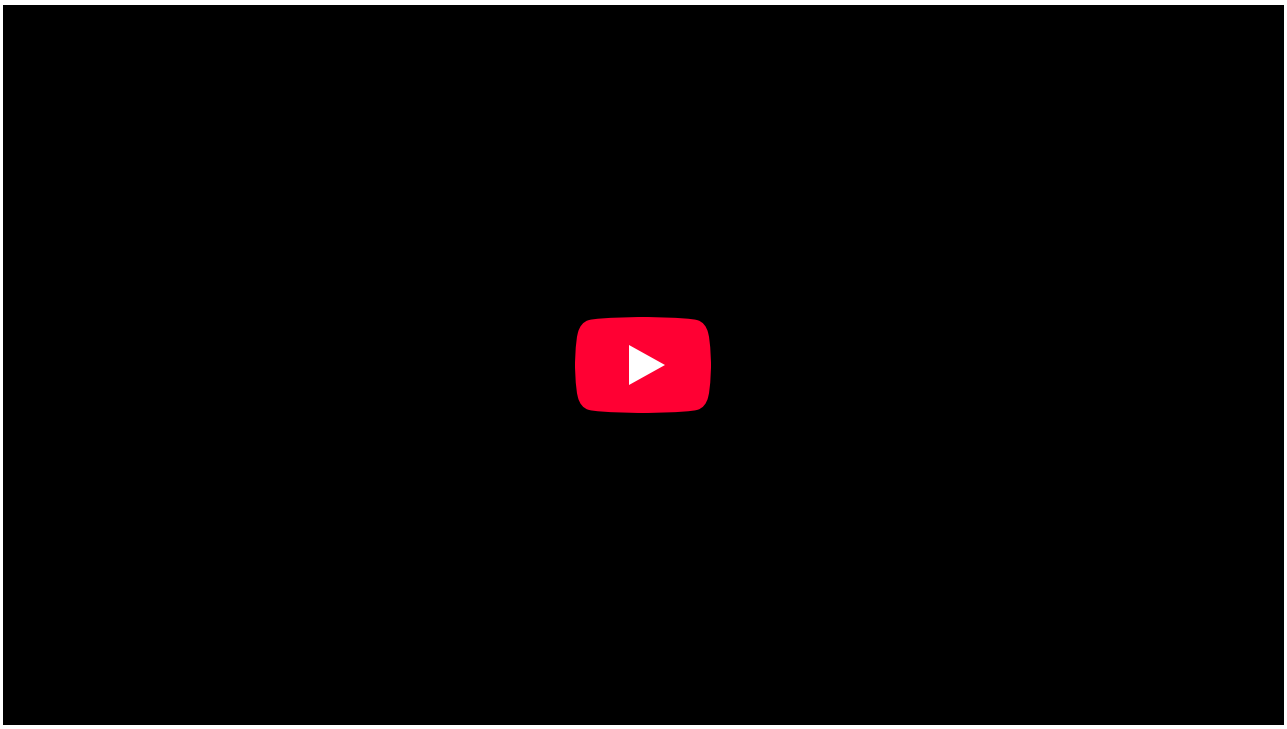
Question

Is $W_\perp$ a subspace of R^n? Justify your answer briefly.

Question

What are the vectors in both W and $W_\perp$?


```
from IPython.display import YouTubeVideo
YouTubeVideo("5B8XluiqdHM",width=640,height=360, cc_load_policy=True)
```



Projection of a Vector onto a Subspace

Think of a projection onto a subspace is analogous to a shadow on a surface. Aspects of an objects 3D space is represented in a 2D shadow but you can't take the shadow by itself and exactly recreate the 3D surface.

Image from: Wikimedia

The following is the matimatical defination of projection onto a subspace.

Definition

Let W be a subspace of R^n of dimension m. Let $\{w_1, \cdots, w_m\}$ be an orthonormal basis for W. Then the projection of vector v in R^n onto W is denoted as $\text{proj}_W v$ and is defined as $\text{proj}_W v = (v \cdot w_1)w_1 + (v \cdot w_2)w_2 + \cdots + (v \cdot w_m)w_m$.

Another way to say the above defination is that the project of v onto the W is just the sumation of v projected onto each vector in a basis of W.

Remarks:

> *Recall in the lecture on Projections, we discussed the projection onto a vector, which is the case for $m = 1$. We used the projection for $m > 1$ in the Gram-Schmidt algorithm.*

> *The projection does not depend on which orthonormal basis you choose.*

> *If v is in W, we have $\text{proj}_W v = v$.*

The Orthogonal Decomposition Theorem

Theorem

Let W be a subspace of R^n. Every vector in v in R^n can be written uniquely in the form $v = w + w_\perp$, where w is in W and $w_\perp$ is orthogonal to W (i.e., $w_\perp$ is in $W_\perp$). In addition, $w = \text{proj}_W v$ and $w_\perp = v - \text{proj}_W v$.

Definition

Let x be a point in R^n, W be a subspace of R^n. The distance from x to W is defined to be the minimum of the distances from x to any point y in W. $d(x, W) = \min\{\|x - y\| : \text{for all } y \text{ in } W\}$. The optimal y can be achieved at $\text{proj}_W x$, and $d(x, W) = |x - \text{proj}_W x|$.

Question

Let $v = (3, 2, 6)$ and W is the subspace consisting all vectors with the form (a, b, b). Find the projection of v onto W.

Question

Let $v = (3, 2, 6)$ and W is the subspace consisting all vectors with the form (a, b, b). Find the distance from v to W.

27.2: The Four Fundamental Spaces

In the lecture on *Change Basis*, we talked about four subspaces based on a matrix A:

> *Row space of A: linear combination of all rows of A*
>
> *Column space of A: linear combination of all columns of A*
>
> *Null space or kernel of A: all x such that $Ax = 0$*
>
> *Null space of $A^\top$: all y such that $A^\top y = 0$*

In this course we represent a system of linear equations as $Ax = b$. The matrix A can be viewed as taking a point x in the input space and projecting that point to b in the output space.

It turns out, everything we need to know about A is represented by four fundamental vector spaces. Two of the four spaces are easily defined as follows:

> *Row space of A: linear combination of all rows of A*
>
> *Column space of A: linear combination of all columns of A*

The other two fundamental spaces are defined by a concept called the *Null Space*. The *Null space* is calculated by finding all the solutions to the homogeneous system $Ax = 0$. The final two fundamental spaces are defined as follows:

> *Null space or kernel of A: all x such that $Ax = 0$*
>
> *Null space of $A^\top$: all y such that $A^\top y = 0$*

27.3: Independent Learning

Do This

Find a YouTube video that helps you understand the four fundamental spaces.

Question

What is the URL for your video?

Do This

Add the link to the video to the code below. Try embedding the link in the provided Python `YouTubeVideo` Function by replacing XXXXX with the video ID.

```
from IPython.display import YouTubeVideo
YouTubeVideo("XXXXXX",width=640,height=360, cc_load_policy=1)
```

run | restart | restart & run all

Question

What criteria did you use in selecting your video?

Question

How long into a video did you go before deciding if it was good or bad?

Question

What did you like about the video you selected.

Question

What didn't you like about the video?

27.4: Assignment wrap-up

Assignment-Specific Question

What is the URL for your video for the four Fundamental spaces?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

28: 14 In-Class Assignment - Fundamental Spaces

28.0: Introduction

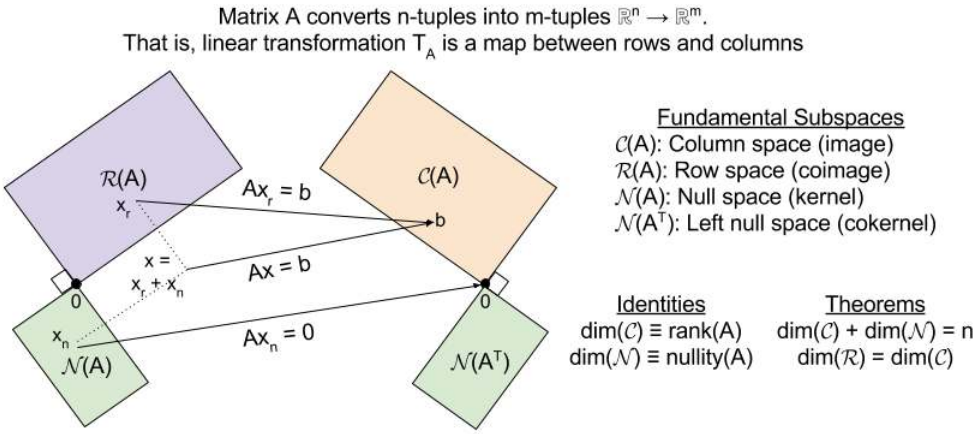


Image from: https://kevinbinz.com/2017/02/20/linear-algebra/

Agenda for today's class (80 minutes)

1. (20 minutes) Pre-class assignment review
2. (20 minutes) Four Fundamental Spaces
3. (20 minutes) Practice Example

28.1: Pre-class assignment review

- 14 Pre-Class Assignment: Fundamental Spaces

28.2: Four Fundamental Subspaces

The four fundamental subspaces

- Columnspace, $\mathcal{C}(A)$
- Nullspace, $\mathcal{N}(A)$
- Rowspaces, $R(A)$
  - All linear combinations of rows
  - All the linear combinations of the colums of $A^\top$, $\mathcal{C}(A^\top)$
- Nullspace of $A^\top$, $\mathcal{N}(A^\top)$ (the left nullspace of A)

Where are these spaces for a $m \times n$ matrix A?

- $\mathcal{R}(A)$ is in R^n
- $\mathcal{N}(A)$ is in R^n
- $\mathcal{C}(A)$ is in R^m
- $\mathcal{N}(A^\top)$ is in R^m

Calculating basis and dimension

For $\mathcal{R}(A)$

- If A undergoes row reduction to row echelon form B, then $\mathcal{C}(B) \neq \mathcal{C}(A)$, but $\mathcal{R}(B) = \mathcal{R}(A)$ (or $\mathcal{C}(B^\top) = \mathcal{C}(A^\top)$)
- A basis for the rowspace of A (or B) is the first r rows of B
  - So we row reduce A and take the pivot rows and transpose them
- The dimension is also equal to the rank r

For $\mathcal{N}(A)$

- The bases are the special solutions (one for every free variable, $n - r$)
- The dimension is $n - r$

For $\mathcal{C}(A) = \mathcal{R}(A^\top)$

- Apply the row reduction on the transpose $A^\top$.
- The dimension is the rank r

For $\mathcal{N}(A^\top)$

- It is also called the left nullspace, because it ends up on the left (as seen below)
- Here we have $A^\top y = 0$
  - $y^\top (A^\top)^\top = 0^\top$
  - $y^\top A = 0^\top$
  - This is (again) the special solutions for $A^\top$ (after row reduction)
- The dimension is $m - r$

28.3: Practice Example

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

Consider the linear transformation defined by the following matrix A.

$$A = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 2 & 3 & 1 \end{bmatrix}$$

Question

What is the reduced row echelon form of A? You can use `sympy` .

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer to the above question here.
```

Question

Now let's calculate the row space of A. Note that the row space is defined by a linear combination of the non-zero row vectors in the reduced row echelon matrix:

Question

What is the rank of matrix A? You should know the rank by inspecting the reduced row echelon form. Find a `numpy` or `sympy` function that you can use to verify your answer?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Put code here to verify your answer.
```

Question

Using the reduced row echelon form define the leading variables in terms of the free variables for the homogeneous equation.

Question

The solution to the above question defines the nullspace of A (aka the Kernel). Use the `sympy.nullspace` function to verify your answer.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your code here
```

Question

Now let's calculate the range of A (column space of A). Note that the range is spanned by the column vectors of A. Transpose A and calculate the reduced row echelon form of the transposed matrix like we did above.

```
## Put your code here
```

Question

The nonzero row vectors of the above solution will give a basis for the range (or $\mathcal{C}(A)$). Write the range of A as a linear combination of these nonzero vectors:

Question

Finally, using the reduced row echelon form for $A^\top$ define the leading variables in terms of the free variables and define the null space of $A^\top$.

CHAPTER OVERVIEW

29: 15 Pre-Class Assignment - Diagonalization and Powers

29.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 5 II1-2 pg 388-396
- ***Beezer Section SD pg 403-415***

Goals for today's pre-class assignment

1. Eigenvalues and eigenvectors review
2. Diagonalizable Matrix
3. Assignment wrap-up

29.1: Eigenvalues and eigenvectors review

Definition

A non-zero vector x in R^n is called an *eigenvector* of a $n \times n$ matrix A if Ax is a scalar multiple of x. If $Ax = \lambda x$, then λ is called the *eigenvalue* of A corresponding to x.

Steps for finding the eigenvalues and eigenvectors

We want to find λ and non-zero vector x such that $Ax = \lambda x$ for a $n \times n$ matrix.

1. We introduce an identity matrix I of $n \times n$. Then the equation becomes:

$$Ax = \lambda Ix$$

$$Ax - \lambda Ix = 0$$

$$(A - \lambda I)x = 0$$

2. This suggests that we want to find λ such that $(A - \lambda I)x = 0$ has a non-trivial solution. It is equivalent to that the matrix $A - \lambda I$ is singular, i.e., has a determinant of 0. $|A - \lambda I| = 0$
3. The determinant is polynomial in λ (called the characteristic polynomial of A) with degree n. We solve this equation (called the characteristic equation) for all possible λ (eigenvalues).
4. After finding the eigenvalues, we substitute them back into $(A - \lambda I)x = 0$ and find the eigenvectors x.

Let's calculate eigenvalues for the following matrix:

$$A = \begin{bmatrix} 0 & 0 & -2 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{bmatrix}$$

Find eigenvalues

Looking at the above recipe, let's solve the problem symbollically using `sympy`. First lets create a matrix B such that:

$$B = A - \lambda I$$


```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Most sympy requires defeing the variables as "symbols"
#Once we do this we can use the variables in place of numbers
lam = sym.symbols('lambda')

A = sym.Matrix([[0, 0 ,-2], [1, 2, 1], [1, 0, 3]])
I = sym.eye(3)

B = A - lam*I

B
```

Now, per step 2, the determinate of B must be zero. Note that `sympy` calculates the determinate symbollically as follows:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
B.det()
```

Do This

Using the `sympy.solve` function on the determinate of B to solve for `lam` (λ). Verify that the solution to the last question produces the same eigenvalues as above.

Login with LibreOne to run this code cell interactively.

Login

```
# Put your code to solve for det(B) = 0 here
```

Do This

First, let's use the built in funciton `eigenvals` function in `sympy` to calculate the eigenvalues. Find out the meaning of the output.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your code here
```

Find eigenvectors

Now we know the eigenvalues, we can substitue them back into the equation to find the eigenvectors.

We solve this symbollically using `sympy` . First let's make a vector of our eigenvalues (from above):

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
eig = [1,2]
```

Now (per step 4 above) we need to solve the equation $(A-\lambda I)x=0$. One way to do this in `sympy` is as follows:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
x1,x2,x3 = sym.symbols(['x_1','x_2','x_3'])

x = sym.Matrix([[x1],[x2],[x3]])
x
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
for lam in eig:
    vec = sym.solve((A - lam*I)*x,x)
    print(vec)
```

```
{x_1: -2*x_3, x_2: x_3}
{x_1: -x_3}
```

Question

Explain your output here. (Hint, you can also try the `rref` to find the solutions)

Do This

Next, let's use the `eigenvects` function in `sympy` to find three linear independent eigenvectors for the matrix A?

```
# Put your answer to the above question here
```

Question

Compare this answer to the eigenvectors we calculated above. Does this answer make sense? What does the syntax tell us?

Do This

Find the eigenvalues and eigenvectors of the following matrix:

$$A_2 = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

Question

What are the eigenvalues for the matrix A_2?

Question

What are the eigenvectors for the matrix A_2?

29.2: Diagonalizable Matrix

In class we will be using matrix diagonalization to solve some problems.
Matrix A is diagonalizable if there exists a diagonal matrix D that is similar similar to A:

$$D = C^{-1}AC$$

If matrix A has linearly independent eigenvectors $(v_1, \ldots, v_n)$ then A is diagonalizable with the following solution:

$$C = \left[v_1^T, \ldots, v_n^T\right]$$

In other words, each column of C is a linearly independent eigenvector of A. The diagonal matrix D is

$$D = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}$$

In other-other words, D consists of the corresponding eigenvalues.

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
sym.init_printing(use_unicode=True)
```

run | restart | restart & run all

Do This

Using `numpy` , Diagonalize (i.e. calculate C and D) the following matrix:

```
A = np.matrix([[5, -2, 2], [4, -3, 4], [4,-6,7]])
sym.Matrix(A)
```

run | restart | restart & run all

```
# Put your answer here
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(D,'56821475223b52e0b6e751da444a1441');
```

run | restart | restart & run all

Do This

Verify that A is in fact Diagonalizable by calculating $D2 = C^{-1}AC$ and comparing it to your original D using `np.allclose` .

```
#Put your verificaiton code here.
```

run | restart | restart & run all

```
np.allclose(D,D2)
```

run | restart | restart & run all

Diagonalization of Symmetric Matrices

One special case is Symmetric Matrices. It can be shown that symmetric Matrices are Diagonalizable and the resulting eigenvectors are not only linearly independent but also orthogonal. Since this is true, the equation changes to:

$$D = C^T AC$$

Question

Why do we care if C is orthogonal? What advantages does the above equation give us?

29.3: Assignment wrap-up

Assignment-Specific Question

Why do we care if C is orthogonal? What advantages does the previous equation give us?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

30: 15 In-Class Assignment - Diagonalization

30.0: Introduction

$$P^{-1}AP = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix},$$

Image from: Wikipedia

Agenda for today's class (80 minutes)

1. (20 minutes) Pre-class Assignment Review
2. (20 minutes) Diagonalization
3. (20 minutes) The Power of a Matrix

30.1: Pre-class Assignment Review

- 15 Pre-Class Assignment: Diagonalization and Powers

30.2: Diagonalization

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
from urllib.request import urlretrieve

sym.init_printing()
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Reminder

The eigenvalues of triangular (upper and lower) and diagonal matrices are easy:

- The eigenvalues for triangular matrices are the diagonal elements.
- The eigenvalues for the diagonal matrices are the diagonal elements.

Diagonalization

Definition

A square matrix A is said to be diagonalizable if there exist a matrix C such that $D = C^{-1}AC$ is a diagonal matrix.

Definition

B is a similar matrix of A if we can find C such that $B = C^{-1}AC$.

Given an $n \times n$ matrix A, can we find another $n \times n$ invertable matrix C such that when $D = C^{-1}AC$ is diagonal, i.e., A is diagonalizable?

- Because C is invertible, we have

$$C^{-1}AC = D$$

$$CC^{-1}AC = CD$$

$$AC = CD$$

- Generate C as the columns of n linearly independent vectors $(x_1 \dots x_n)$ We can compute $AC = CD$ as follows:

$$A\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = AC = CD = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \lambda_n \end{bmatrix}$$

- Then we check the corresponding columns of the both sides. We have

$$Ax_1 = \lambda_1 x_1$$

$$\vdots$$

$$Ax_n = \lambda x_n$$

- A has n linear independent eigenvectors.
- A is saied to be *similar* to the diagonal matrix D, and the transformation of A into D is called a *similarity transformation*.

A simple example

Consider the following:

$$A = \begin{bmatrix} 7 & -10 \\ 3 & -4 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & 5 \\ 1 & 3 \end{bmatrix}$$

Do This

Find the similar matrix $D = C^{-1}AC$ of A.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.matrix(D, '8313fe0f529090d6a8cdb36248cfdd6c');
```

run | restart | restart & run all

Do This

Find the eigenvalues and eigenvectors of A. Set variables `e1` and `vec1` to be the smallest eigenvalue and it's associated eigenvector and `e2, vec2` to represent the largest.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.float(e1, "e4c2e8edac362acab7123654b9e73432");
```

run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.float(e2, "d1bd83a33f1a841ab7fda32449746cc4");
```

run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.eq_vector(vec1, "d28f0a721eedb3d5a4c714744883932e", decimal_accuracy = 4)
```

run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.eq_vector(vec2, "09d9df5806bc8ef975074779da1f1023", decimal_accuracy = 4)
```

run | restart | restart & run all

Theorem

Similar matrices have the same eigenvalues.

Proof

Assume $B = C^{-1}AC$ is a similar matrix of A, and λ is an eigenvalue of A with corresponding eigenvector x. That is, $Ax = \lambda x$. Then we have $B(C^{-1}x) = C^{-1}AC(C^{-1}x) = C^{-1}Ax = C^{-1}(\lambda x) = \lambda(C^{-1}x)$. That is $C^{-1}x$ is an

eigenvector of B with eigenvalue λ.

A second example

Do This

Consider $A = \begin{bmatrix} -4 & -6 \\ 3 & 5 \end{bmatrix}$.
Find a matrix C such that $C^{-1}AC$ is diagonal.
Hint: use the function `diagonalize` in `sympy` .

```
#Put your answer to the above question here.
```
run | restart | restart & run all

```
#Check the output type
assert(type(C)==sym.Matrix)
```
run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.matrix(C,'ba963b7fef354b4a7ddd880ca4bac071')
```
run | restart | restart & run all

The third example

Do This

Consider $A = \begin{bmatrix} 5 & -3 \\ 3 & -1 \end{bmatrix}$.
Can we find a matrix C such that $C^{-1}AC$ is diagonal.
Hint: find eigenvalues and eigenvectors using `sympy` .

```
#Put your answer to the above question here.
```
run | restart | restart & run all

Dimensions of eigenspaces and diagonalization

Definition

The set of all eigenvectors of a $n \times n$ matrix corresponding to a eigenvalue λ, together with the zero vector, is a subspace of R^n. This subspace spaces is called eigenspace.

- For the third example, we have that the characteristic equation $(\lambda - 2)^2 = 0$.
- Eigenvalue $\lambda = 2$ has multiplicity 2, but the eigenspace has dimension 1, since we can not find two lineare independent eigenvector for $\lambda = 2$.

> *The dimension of an eigenspace of a matrix is less than or equal to the multiplicity of the corresponding eigenvalue as a root of the characteristic equation.*
> *A matrix is diagonalizable if and only if the dimension of every eigenspace is equal to the multiplicity of the corresponding eigenvalue as a root of the characteristic equation.*

The fourth example

Do This

Consider $A = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$.

Can we find a matrix C such that $C^{-1}AC$ is diagonal.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

30.3: The Power of a Matrix

- For a diagonalizable matrix A, we have $C^{-1}AC = D$. Then we have $A = CDC^{-1}$
- We have $A^2 = CDC^{-1}CDC^{-1} = CD^2C^{-1}A^n = CDC^{-1}\dots CDC^{-1} = CD^nC^{-1}$.
- Because the columns of C are eigenvectors, so we can say that the eigenvectors for A and A^n are the same if A is diagonalizable.
- If x is an eigenvector of A with the corresponding eigenvalue λ, then x is also an eigenvector of A^n with the corresponding eigenvalue λ^n.

```
# Here are some libraries you may need to use
%matplotlib inline
import numpy as np
import sympy as sym
import networkx as nx
import matplotlib.pyplot as plt
from urllib.request import urlretrieve

sym.init_printing(use_unicode=True)
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

Graph Random Walk

- Define the following matrices:
  - I is the identity matrix
  - A is the adjacency matrix
  - D is diagonal matrix of degrees (number of edges connected to each node)

$$W = \frac{1}{2}(I + AD^{-1})$$

- The lazy random walk matrix, W, takes a distribution vector of *stuff*, p_t, and diffuses it to its neighbors:

$$p_{t+1} = Wp_t$$

- For some initial distribution of *stuff*, p_0, we can compute how much of it would be at each node at time, t, by powering W as follows:

$$p_t = W^t p_0$$

- Plugging in the above expression yields:

$$p_t = \left(\frac{1}{2}(I + AD^{-1})\right)^t p_0$$

Do This

Using matrix algebra, show that $\frac{1}{2}(I + AD^{-1})$ is similar to $I - \frac{1}{2}N$, where $N = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$ is the normalized graph Laplacian.

Random Walk on Barbell Graph

To generate the barbell graph, run the following cell.

```
n = 60 # number of nodes
B = nx.Graph() # initialize graph
```

```
## initialize empty edge lists
edge_list_complete_1 = []
edge_list_complete_2 = []
edge_list_path = []

## generate node lists
node_list_complete_1 = np.arange(int(n/3))
node_list_complete_2 = np.arange(int(2*n/3),n)
node_list_path = np.arange(int(n/3)-1,int(2*n/3))

## generate edge sets for barbell graph
for u in node_list_complete_1:
    for v in np.arange(u+1,int(n/3)):
        edge_list_complete_1.append((u,v))

for u in node_list_complete_2:
    for v in np.arange(u+1,n):
        edge_list_complete_2.append((u,v))

for u in node_list_path:
    edge_list_path.append((u,u+1))

# G.remove_edges_from([(3,0),(5,7),(0,7),(3,5)])

## add edges
B.add_edges_from(edge_list_complete_1)
B.add_edges_from(edge_list_complete_2)
B.add_edges_from(edge_list_path)


## draw graph
pos=nx.spring_layout(B) # positions for all nodes

### nodes
nx.draw_networkx_nodes(B,pos,
                       nodelist=list(node_list_complete_1),
                       node_color='c',
                       node_size=400,
                       alpha=0.8)
nx.draw_networkx_nodes(B,pos,
                       nodelist=list(node_list_path),
                       node_color='g',
                       node_size=200,
                       alpha=0.8)
nx.draw_networkx_nodes(B,pos,
                       nodelist=list(node_list_complete_2),
                       node_color='b',
                       node_size=400,
                       alpha=0.8)


### edges
```

```
nx.draw_networkx_edges(B,pos,
                       edgelist=edge_list_complete_1,
                       width=2,alpha=0.5,edge_color='c')
nx.draw_networkx_edges(B,pos,
                       edgelist=edge_list_path,
                       width=3,alpha=0.5,edge_color='g')
nx.draw_networkx_edges(B,pos,
                       edgelist=edge_list_complete_2,
                       width=2,alpha=0.5,edge_color='b')


plt.axis('off')
plt.show() # display
```

run | restart | restart & run all

Do This

Generate the lazy random walk matrix, W, for the above graph.

```
A = nx.adjacency_matrix(B)
A = A.todense()

d = np.sum(A,0) # Make a vector of the sums.
D = np.diag(np.asarray(d)[0])
```

run | restart | restart & run all

```
#Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer
checkanswer.matrix(W, "7af4a5b11892da6e1a605c8239b62093")
```

run | restart | restart & run all

Do This

Compute the eigenvalues and eigenvectors of W. Make a diagonal matrix J with the eigenvalues on the diagonal. Name the matrix of eigenvectors V (each column is an eigenvector).

```
#Put your answer to the above question here.
```

run | restart | restart & run all

Now we make sure we constructed V and A correctly by double checking that $W = VJV^{-1}$

```
np.allclose(W, V*J*np.linalg.inv(V))
```

run | restart | restart & run all

Do This

Let your $p_0 = [1, 0, 0, \ldots, 0]$. Compute p_t for $t = 1, 2, \ldots, 100$, and plot $||v_1 - p_t||_1$ versus t, where v_1 is the eigenvector associated with the largest eigenvalue $\lambda_1 = 1$ and whose sum equals 1. (Note: $||\cdot||_1$ may be computed using `np.linalg.norm(v_1-p_t, 1)` .)

```
#Put your answer to the above question here.
```

run | restart | restart & run all

Compare to Complete Graph

If you complete the above, do the same for a complete graph on the same number of nodes.

Question

What do you notice about the graph that is different from that above?

CHAPTER OVERVIEW

31: 16 Pre-Class Assignment - Linear Dynamical Systems

31.0: Introduction

Readings for this topic (Recommended in bold)

- ***Boyd Chapter 9 pg 163-173***

Goals for today's pre-class assignment

1. Linear Dynamical Systems
2. Markov Models
3. Ordinary Differential Equations
4. Assignment wrap up

31.1: Linear Dynamical Systems

A linear dynamical system is a simple model of how a system changes with time. These systems can be represented by the following “dynamics” or “update equation”:

$$x_{(t+1)} = A_t x_t$$

Where t is an integer representing th progress of time and A_t are an $n \times n$ matrix called the dynamics matrices. Often the above matrix does not change with t. In this case the system is called “time-invariant”.

We have seen a few “time-invarient” examples in class.

Do This

Review ***Chapter 9 in the Boyd and Vandenberghe*** text and become familiar with the contents and the basic terminology.

31.2: Markov Models

This is not the first time we have used Dynamical Linear Systems.

Do This

Review markov models in 10 Pre-Class Assignment: Eigenvectors and Eigenvalues. See how this is a special type of linear dynamical systems that work with state probabilities.

Example

The dynamics of infection and the spread of an epidemic can be modeled as a linear dynamical system.
We count the fraction of the population in the following four groups:

- Susceptible: the individuals can be infected next day
- Infected: the infected individuals
- Recovered (and immune): recovered individuals from the disease and will not be infected again
- Decreased: the individuals died from the disease

We denote the fractions of these four groups in $x(t)$. For example $x(t) = (0.8, 0.1, 0.05, 0.05)$means that at day t, 80% of the population are susceptible, 10% are infected, 5% are recovered and immuned, and 5% died.
We choose a simple model here. After each day,

- 5% of the susceptible individuals will get infected
- 3% of infected inviduals will die
- 10% of infected inviduals will recover and immuned to the disease
- 4% of infected inviduals will recover but not immuned to the disease
- 83% of the infected inviduals will remain

Do This

Write the dynamics matrix for the above markov linear dynamical system. Come to class ready to discuss the matrix. (hint the columns of the matrix should add to 1).

```
# Put your matrix here
```

run | restart | restart & run all

Do This

Review how we solved for the long term steady state of the markov system. See if you can find these probabilities for your dyamics matrix.

```
# Put your matrix here
```

run | restart | restart & run all

31.3: Ordinary Differential Equations

Ordinary Differential Equations (ODEs) are yet another for of linear dynamical systems and are a scientific model used in a wide range of problems of the basic form:

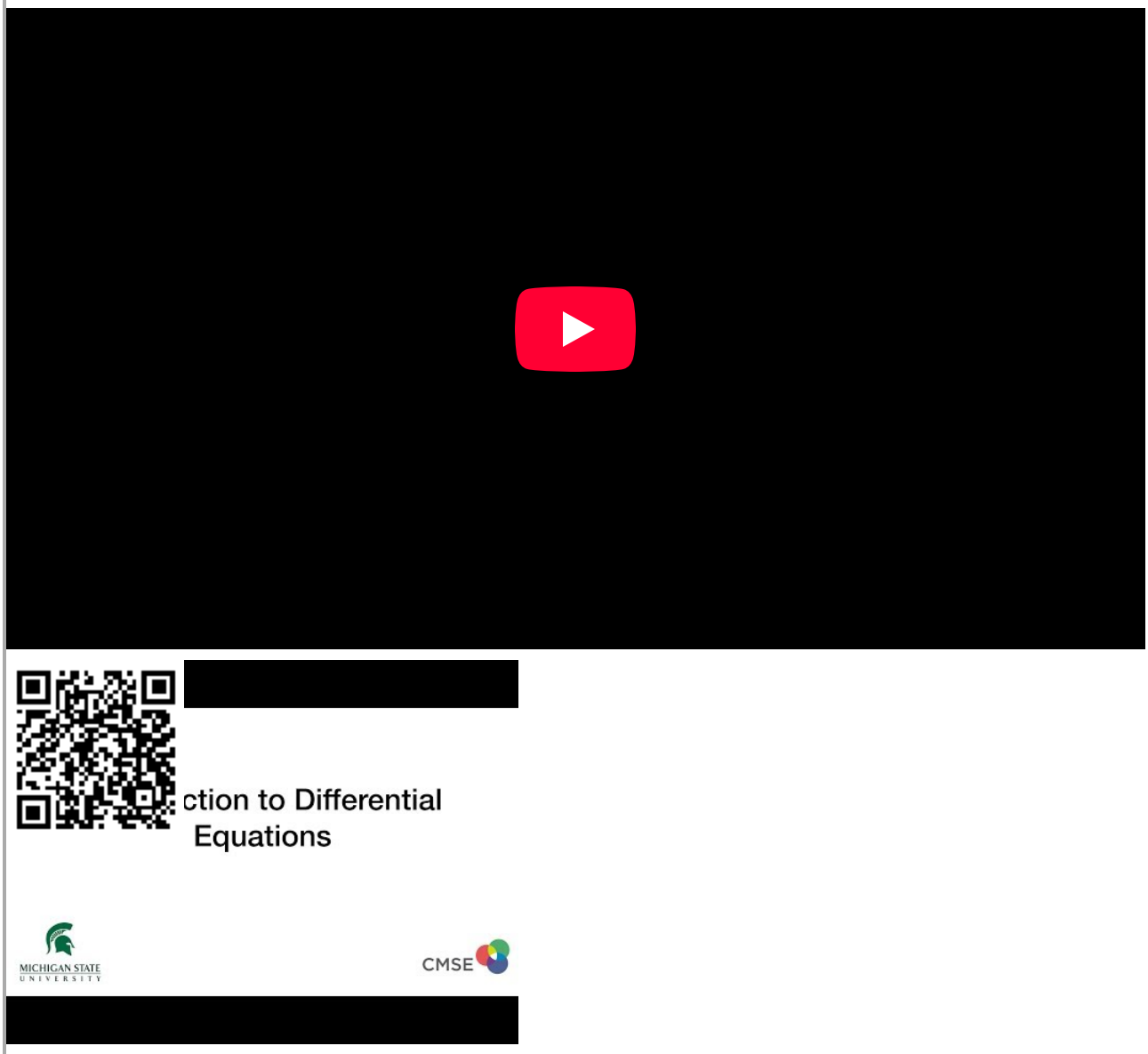
\[\dot{x} = Ax \nonumber \]

These are equations such that the is the instantaneous rate of change in \(x\) (i.e. \(\dot{x}\) is the derivative of \(x\)) is dependent on \(x\). Many systems can be modeled with these types of equations.

Here is a quick video that introduces the concepts of Differential Equations. The following is a good review of general ODEs.

```
from IPython.display import YouTubeVideo
YouTubeVideo("8QeCQn7uxnE",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



Now consider an ODE as a system of linear equations:

\[\dot{x_t} = A x_t \nonumber \]

Based on the current \(x\) vector at time \(t\) and the matrix \(A\), we can calculate the derivative at \(\dot{x}\) at time \(t\). Once we know the derivative, we can increment the time to by some small amount \(dt\) and calculate a new value of \(x\) as follows:

\[x_{t+1} = x_t + \dot{x_t}dt \nonumber \]

Then we can do the exact sequence of calculations again for \(t+2\). The following function has the transition matrix (\(A\)), the starting state vector (\(x_0\)) and a number of time steps (\(N\)) and uses the above equations to calculate each state and return all of the \(x\) statues:

The following code generates a trajectory of points starting from `x_0`, applying the matrix \(A\) to get \(x_1\) and then applying \(A\) again to see how the system progresses from the start state.

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

run | restart | restart & run all

```
def traj(A, x, n):
    dt = 0.01
    x_all = np.matrix(np.zeros((len(x),n)))   # Store all points on the trajectory
    for i in range(n):
        x_dot = A*x          # First we transform x into the derrivative
        x = x + x_dot*dt     # Then we estimate x based on the previous value and a sma
        x_all[:,i] = x[:,0]
    return x_all
```

run | restart | restart & run all

For example the following code uses the matrix \(A= \begin{bmatrix}1 & 1 \\ 1 & -2\end{bmatrix}\) and the starting point (0,0) over 50 timesteps to get a graph:

```
A = np.matrix([[1,1],[1,-2]])
x0 = np.matrix([[1],[1]])

x_all = traj(A, x0, 50)
plt.scatter(np.asarray(x_all[0,:]),np.asarray(x_all[1,:]))

plt.scatter(list(x0[0,:]),list(x0[1,:])) #Plot the start point as a refernce
```

run | restart | restart & run all

Do This

Let \(A= \begin{bmatrix}2 & 3 \\ 4 & -2\end{bmatrix}\)

Write a loop over the points \((−1.5,−1), (−1,2)(−1,2)\) and plot the results of the `traj` function:

```
A = np.matrix([[2,3],[4,-2]])
x0 = np.matrix([[1.5, -1.5, -1, 1, 2],[1, -1, 2, -2, -2]])
```

run | restart | restart & run all

```
# Put your code here
```

run | restart | restart & run all

Do This

Let $A= \begin{bmatrix}6 & -1 \\ 1 & 4\end{bmatrix}$
Write a loop over the points $((−1.5,−1), (−1,2)(−1,2)$ and plot the results of the `traj` function:

```
# Put your code here
```

run | restart | restart & run all

Do This

Let $A= \begin{bmatrix}5 & 2 \\ -4 & 1\end{bmatrix}$
Write a loop over the points $((−1.5,−1), (−1,2)(−1,2)$ and plot the results of the `traj` function:

```
# Put your code here
```

run | restart | restart & run all

31.4: Assignment wrap up

Assignment-Specific Question

Where you able to get the ODE code working in the above example. If not, where did you get stuck?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

32: 16 In-Class Assignment - Linear Dynamical Systems

32.0: Introduction

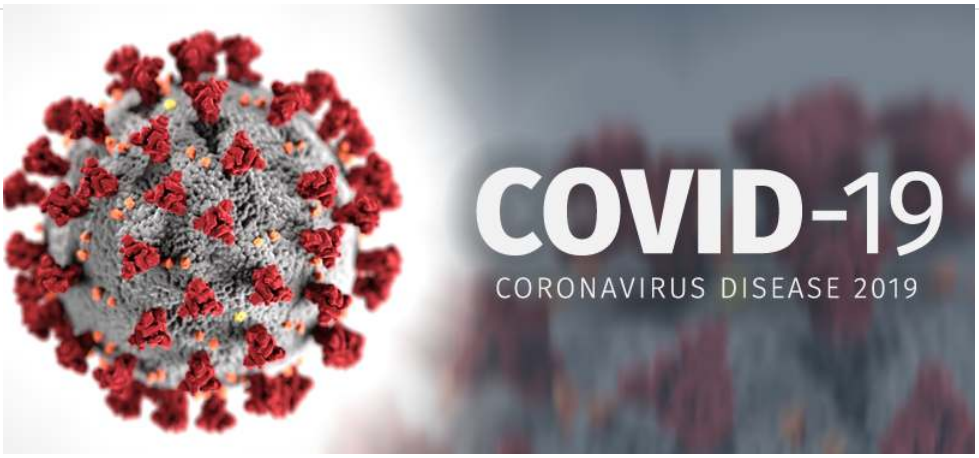


Image from: Texas Department of State Health Services

Agenda for today's class (80 minutes)

1. (20 minutes) Epidemic Dynamics - Discrete Case
2. (20 minutes) Epidemic Dynamics - Continuous Model
3. (20 minutes) Population Dynamics

32.1: Epidemic Dynamics - Discrete Case

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

run | restart | restart & run all

The dynamics of infection and the spread of an epidemic can be modeled as a linear dynamical system.
We count the fraction of the population in the following four groups:

- Susceptible: the individuals can be infected next day
- Infected: the infected individuals
- Recovered (and immune): recovered individuals from the disease and will not be infected again
- Deceased: the individuals died from the disease

We denote the fractions of these four groups in \(x(t)\). For example \(x(t)=(0.8,0.1,0.05,0.05)\) means that at day \(t\), 80% of the population are susceptible, 10% are infected, 5% are recovered and immuned, and 5% died.
We choose a simple model here. After each day,

- 5% of the susceptible individuals will get infected
- 3% of infected inviduals will die
- 10% of infected inviduals will recover and immuned to the disease
- 4% of infected inviduals will recover but not immuned to the disease
- 83% of the infected inviduals will remain

```
A = np.matrix([[0.95, 0.04, 0, 0],[0.05, 0.83, 0, 0],[0, 0.1, 1, 0],[0,0.03,0,1]])
sym.Matrix(A)
```

run | restart | restart & run all

Do This

If we start with \(x(0)=(1,0,0,0)\) for day 0. Use the `for` loop to find the distribution of the four groups after 50 days.

```
x0 = np.matrix([[1],[0],[0],[0]])
x  = x0
for i in range(50):
    x = A*x
print(x)
```

run | restart | restart & run all

```
[[0.15041595]
 [0.05576501]
 [0.61063003]
 [0.18318901]]
```

Do This

Write a program to apply the above transformation matrix for 200 iterations and plot the results.

```
#Put your answer to the above question here
```

run restart restart & run all

32.2: Epidemic Dynamics - Continuous Model

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

run | restart | restart & run all

Instead of using the discrete markov model, we can also use a continuous model with ordinary differential equations.
For example, we have that
\[\dot{x}_1 = {dx_1(t)\over dt} = -0.05x_1(t)+ 0.04 x_2(t) \nonumber \]
It means that the changes in the susceptible group depends on susceptible and infected individuals. It increase because of the recovered people from infected ones and it decreases because of the infection.
Similarly, we have the equations for all three groups.
\[\dot{x}_2 = {dx_2(t)\over dt} = 0.05x_1(t)-0.17 x_2(t) \\
\dot{x}_3 = {dx_3(t)\over dt}= 0.1 x_2(t) \\
\dot{x}_4 = {dx_4(t)\over dt} = 0.03 x_2(t) \nonumber \]

Do This

We can write it as system of ODEs as \(\dot{x}(t) = Bx(t)\). Write down the matrix \(B\) in `numpy.matrix`

```
# Put your answer to the above question here.
```

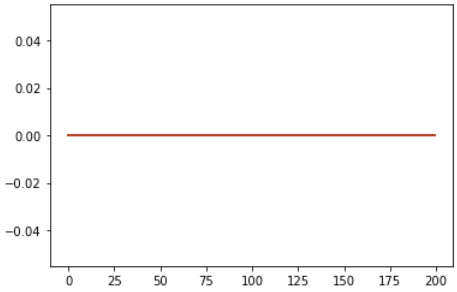
run | restart | restart & run all

Do This

Plot all the distribution for 200 days. Then compare it with the discrete version.

```
x0 = np.matrix([[1],[0],[0],[0]])
n = 200
x_all = np.matrix(np.zeros((4,n)))

### Your code starts here ###

### Your code ends here ###
for i in range(4):
    plt.plot(x_all[i].T)
```

run | restart | restart & run all



```
D2, C2 = np.linalg.eig(B)
np.allclose(C2*np.diag(D2)*C2**(-1), B)
C = C2
```

run | restart | restart & run all

32.3: Population Dynamics

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing()
```

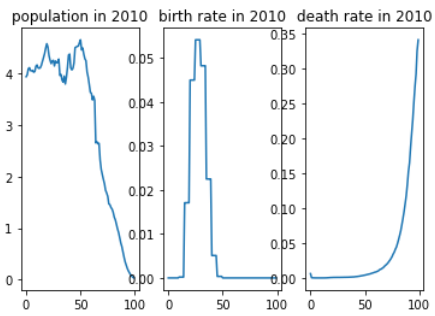
run | restart | restart & run all

In this section, we consider the distribution of a population at different ages. Let $x(t)$ be a 100-vector with $x_i(t)$ denoting the number of people with agent $i-1$.
The birth rate is given in the vector b, where b_i is the average number of births per person with age $i-1$. $b_i=0$ for $i<13$ and $i>50$.
The death rate is given by the vector d, where d_i is the portion of those aged $i-1$ who dies this year.
Then we can model the population with the dynamic systems. We consider the 0-year old first. It includes all newborn from all ages, so we have $x_1(t+1) = b^{\top}x(t)$. Then we consider all other ages for $i > 1$. $x_i(t+1) = (1-d_i)x_{i-1}(t)$

```
population = np.array([
       3.94415,       3.97807,       4.09693,       4.11904,       4.06317,       4.0
       4.04649,       4.14835,       4.17254,       4.11442,       4.10624,       4.1
       4.31614,       4.39529,       4.50085,       4.58523,       4.51913,       4.3
       4.24936,       4.26235,       4.15231,       4.24887,       4.21525,       4.2
       3.98685,       3.88015,       3.83922,       3.95643,       3.80209,       3.9
       4.38327,       4.11498,       4.07610,       4.10511,       4.21150,       4.5
       4.53880,       4.60590,       4.66029,       4.46463,       4.50085,       4.3
       4.03751,       3.93639,       3.79493,       3.64127,       3.62113,       3.4
       2.65713,       2.68076,       2.63914,       2.64936,       2.32367,       2.1
       1.86427,       1.73696,       1.68449,       1.62008,       1.47107,       1.4
       1.30851,       1.21287,       1.16142,       1.07481,       0.98572,       0.9
       0.64062,       0.53800,       0.43556,       0.34499,       0.28139,       0.2
       0.09522,       0.06814,       0.04590,       0.03227])
d = np.array([
       0.00623,       0.00044,       0.00027,       0.00020,       0.00016,       0.0
       0.00012,       0.00011,       0.00010,       0.00013,       0.00013,       0.0
       0.00037,       0.00047,       0.00064,       0.00071,       0.00076,       0.0
       0.00094,       0.00092,       0.00095,       0.00093,       0.00099,       0.0
       0.00110,       0.00114,       0.00115,       0.00120,       0.00131,       0.0
       0.00162,       0.00185,       0.00201,       0.00216,       0.00243,       0.0
       0.00351,       0.00387,       0.00413,       0.00454,       0.00494,       0.0
       0.00670,       0.00710,       0.00769,       0.00828,       0.00860,       0.0
       0.01250,       0.01282,       0.01404,       0.01515,       0.01687,       0.0
       0.02347,       0.02562,       0.02800,       0.03083,       0.03441,       0.0
       0.04964,       0.05539,       0.06149,       0.06803,       0.07673,       0.0
       0.11802,       0.13385,       0.15250,       0.16491,       0.18738,       0.2
       0.27422,       0.29239,       0.32560,       0.34157])
b = np.array([
       0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.0
       0.00000,       0.00000,       0.00020,       0.00020,       0.00020,       0.0
       0.01710,       0.01710,       0.01710,       0.01710,       0.04500,       0.0
       0.04500,       0.05415,       0.05415,       0.05415,       0.05415,       0.0
       0.04825,       0.04825,       0.04825,       0.02250,       0.02250,       0.0
```

```
        0.00510,       0.00510,       0.00510,       0.00510,       0.00510,       0.00
        0.00035,       0.00035,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000,       0.00000,       0.00
        0.00000,       0.00000,       0.00000,       0.00000])

plt.subplot(1,3,1)
plt.plot(population)
plt.title('population in 2010')
plt.subplot(1,3,2)
plt.plot(b)
plt.title('birth rate in 2010')
plt.subplot(1,3,3)
plt.plot(d)
plt.title('death rate in 2010')

plt.show()
```

run | restart | restart & run all



Do This

Find the \(100 \times 100\) matrix `A2` such that \(x(t+1) = Ax(t)\)

```
A2 =
print(A2)
```

run | restart | restart & run all

Do This

Plot the population distribution at year 2020.

CHAPTER OVERVIEW

33: 17 Pre-Class Assignment - Decompositions

33.0: Introduction

Readings for this topic (Recommended in bold)

- ***Beezer Proof Technique DC pg 600-601***

Goals for today's pre-class assignment

1. Matrix Decomposition
2. Decompositions
3. Assignment wrap-up

33.1: Matrix Decomposition

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
```

Do This

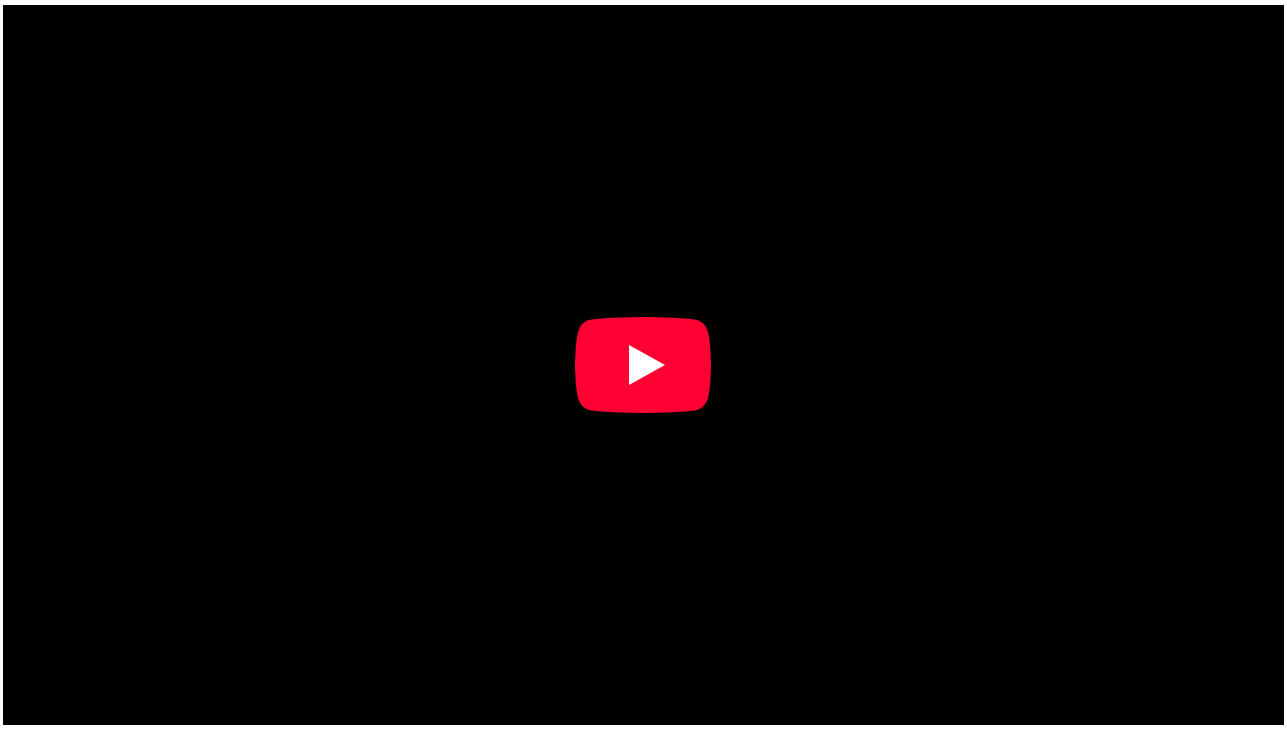
Watch the following video and answer the questions below.

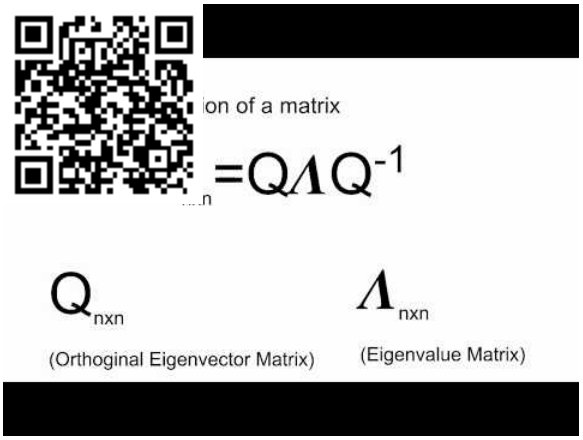
Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("-_2he4J6Xxw",width=640,height=360, cc_load_policy=True)
```





Consider the following code to calculate the $A = Q\Lambda Q^{-1}$ eivendecomposition.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Here is our input matrix
A = np.matrix([[15,7,-7],[-1,1,1],[13,7,-5]])
```

```
sym.Matrix(A)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Calculate eigenvalues and vectors using Numpy
e, Q = np.linalg.eig(A)
print(e)
sym.Matrix(Q)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Turn eigenvalues into a diagonal matrix  (there is even a function for that!)
L = np.diag(e)
sym.Matrix(L)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Calculate A again from Q and L

A2 = Q*L*np.linalg.inv(Q)

sym.Matrix(A2)
```

Do This

Using code, verify that A2 is the same as A.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your answer here
```

Do This

Turn the above code into a function called `eigendecomp` which takes in a matrix A and returns Q and L.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
# Put your code here
```

Question

What other decompositions have we covered in the class so far? Make a list and write down a short description on why we use each decomposition.

33.2: Decompositions

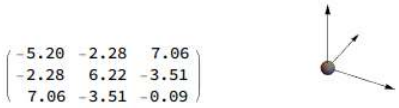


Image from: Wikipedia

In numerical linear algebra, we factorize matrices to facilitate efficient and/or accurate computations (they are also helpful in proofs). There are many possible **matrix decompositions**. Some, e.g., the eigendecomposition, require the matrix to be square, while others, e.g., the QR factorization, exist for arbitrary matrices. Among all possible decompositions (also called *factorizations*), some common examples include:

- **QR Factorization** from Gram-Schmidt orthogonization:
  - $A = QR$
  - Q has orthonormal columns and R is a upper-triangular matrix
  - If there are zero rows in R, we can reduce the number of columns in Q
  - Exists for arbitrary matrices
- **LU / LDU Decomposition** from Gauss Elimination:
  - $A = LU$ or $A = LDU$
  - L is lower-triangular, U is upper-triangular, and D is diagonal
  - Exists for all **square** matrices
  - Is *related to Gaussian Elimination*
- **Cholesky Decomposition**:
  - $A = R^T R \quad (= LDL^T)$
  - R is upper-triangular
  - Factorization of A into $R^T R$ requires A be *symmetric* and *positive-definite*. The latter simply requires $x^T Ax > 0$ for every $x \in \mathbb{R}^n$. Note that $x^T Ax$ is always a scalar value (e.g., note that $x^T A = y^T$ for some vector $y \in \mathbb{R}^n$, and $y^T x$ is the dot product between x and y and, hence, a real scalar).
- **Schur Decomposition**:
  - $A = UTU^T$
  - U is orthogonal and T is upper-triangular
  - Exists for every square matrix and says every such matrix, A, is unitarily equivalent to an upper-triangular matrix, T (i.e., there exists an orthonomal basis with respect to which A is upper-triangular)
  - Eigenvalues on diagonal of T
- **Singular Value Decomposition**:
  - $A = U\Sigma V^T$
  - U is orthogonal, V is orthogonal, and Σ is diagonal
  - Exists for arbitrary matrices
- **Eigenvalue Decomposition**:
  - $A = X\Lambda X^{-1}$
  - X is invertible and Λ is diagonal
  - Exists for square matrices with linearly independent columns (e.g., full rank)
  - Also called the eigendecomposition

Question

What decompositions have we covered in the class so far and how did we use them?

33.3: Assignment wrap-up

Assignment-Specific Question

What other decompositions have we covered in the class so far?

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

34: 17 In-Class Assignment - Decompositions and Gaussian Elimination

---

34.0: Introduction

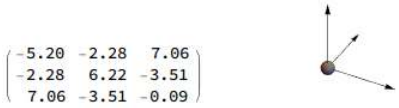
Agenda for today's class (80 minutes)

1. (20 minutes) Pre-class Review
2. (10 minutes) Decompositions
3. (50 minutes) LU Decomposition

34.1: Pre-Class Review

- 17 Pre-Class Assignment: Decompositions

34.2: Decompositions



You can visualize a matrix diagonalization as a rotation of your axis to align them with the matrix eigenvectors. (Public Domain; Jacopo Bertolotti via Wikipedia)

In numerical linear algebra, we factorize matrices to facilitate efficient and/or accurate computations (they are also helpful in proofs). There are many possible **matrix decompositions**. Some, e.g., the eigendecomposition, require the matrix to be square, while others, e.g., the QR factorization, exist for arbitrary matrices. Among all possible decompositions (also called *factorizations*), some common examples include:

- **QR Factorization** from Gram-Schmidt orthogonization:
  - $A = QR$
  - Q has orthonormal columns and R is a upper-triangular matrix
  - If there are zero rows in R, we can reduce the number of columns in Q
  - Exists for arbitrary matrices
- **LU / LDU Decomposition** from Gauss Elimination:
  - $A = LU$ or $A = LDU$
  - L is lower-triangular, U is upper-triangular, and D is diagonal
  - Exists for all **square** matrices
  - Is *related to Gaussian Elimination*
- **Cholesky Decomposition**:
  - $A = R^T R \quad (= LDL^T)$
  - R is upper-triangular
  - Factorization of A into $R^T R$ requires A be *symmetric* and *positive-definite*. The latter simply requires $x^T Ax > 0$ for every $x \in \mathbb{R}^n$. Note that $x^T Ax$ is always a scalar value (e.g., note that $x^T A = y^T$ for some vector $y \in \mathbb{R}^n$, and $y^T x$ is the dot product between x and y and, hence, a real scalar).
- **Schur Decomposition**:
  - $A = UTU^T$
  - U is orthogonal and T is upper-triangular
  - Exists for every square matrix and says every such matrix, A, is unitarily equivalent to an upper-triangular matrix, T (i.e., there exists an orthonomal basis with respect to which A is upper-triangular)
  - Eigenvalues on diagonal of T
- **Singular Value Decomposition**:
  - $A = U\Sigma V^T$
  - U is orthogonal, V is orthogonal, and Σ is diagonal
  - Exists for arbitrary matrices
- **Eigenvalue Decomposition**:
  - $A = X\Lambda X^{-1}$
  - X is invertible and Λ is diagonal
  - Exists for square matrices with linearly independent columns (e.g., full rank)
  - Also called the eigendecomposition

34.3: Focus on LU

In this assignment we will create algorithms that factorize invertible matrices (i.e., square matrices with linearly independent columns), A, into the following decomposition (**note**: the terms *decomposition* and *factorization* are used interchangeably in literature):

- LU Decomposition: $A = LU$

Each matrix in these decompositions is the *matrix product* of elementary matrices. Recall that *an elementary matrix differs from the identity matrix (the square matrix with* 1*s on the diagonal and* 0*s elsewhere) by an elementary row operation.*

The use of these matrix decompositions in Numerical Linear Algebra is motivated by computational efficiency or reduction of *computational complexity* (recall "**Big-O notation**" and **counting the loops in your matrix multiplication algorithm**) and *numerical stability*. Solving our old friend $Ax = b$ by computing the inverse of A, denoted A^{-1}, and taking the matrix-vector product $A^{-1}b = x$ is computational resource intensive and numerically unstable, in general. If the LU decomposition of A exists, then it will be less costly and more stable to:

1. Solve $Ly = b$ for y by *forward-substitution*; and then
2. Solve $Ux = y$ for x by *backward-substitution*

A final note to relate this assignment to the beginning of the course: The algorithms presented here are of a different class than the **Jacobi Algorithm** and **Gauss-Siedel Algorithm**. These are *iterative algorithms*. As you now know, this means that the algorithmic procedure is applied once, twice, and so on, until the output is within a desired tolerance, or until the process has been executed a given number of times (e.g., 100 iterations).

Gaussian Elimination & LU Decomposition

Recall that Gaussian elimination converts an arbitrary matrix, A, into its *row echelon form*. For our purposes, let's suppose that A is a square matrix and, therefore, an $n \times n$ matrix. To simplify our tasks, let us further impose the condition that A is invertible. Thus, the columns of A are linearly independent. This means that Gaussian elimination will yield an ***upper-triangular*** *matrix*. Let us denote this matrix U for ***upper-triangular***.

If there were a function, f that could take A and output U, we could think of Gaussian Elimination as the following process:

$$f(A) = U$$

With this information, we may now rewrite our equation from above as:

$$L^{-1}A = U$$

You may have noticed the superscript in L^{-1}. This just says that L^{-1} is the inverse of some matrix L. And for any invertible matrix, L, we have that the matrix products:

$$L^{-1}L = LL^{-1} = I$$

This is analogous to (for every real number $a \neq 0$):

$$a^{-1} \times a = a \times a^{-1} = 1$$

Using the rules of matrix multiplication, verify the formula above by computing the following:

$$L_1^{-1}L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -l_{21} & 1 & 0 & 0 & 0 \\ -l_{31} & 0 & 1 & 0 & 0 \\ -l_{41} & 0 & 0 & 1 & 0 \\ -l_{51} & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & 0 & 1 & 0 & 0 \\ l_{41} & 0 & 0 & 1 & 0 \\ l_{51} & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = I$$

$$L_2^{-1}L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -l_{32} & 1 & 0 & 0 \\ 0 & -l_{42} & 0 & 1 & 0 \\ 0 & -l_{52} & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & l_{32} & 1 & 0 & 0 \\ 0 & l_{42} & 0 & 1 & 0 \\ 0 & l_{52} & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = I$$

$$L_4^{-1}L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -l_{54} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & l_{54} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = I$$

To understand L^{-1} more deeply, let's turn our attention back to Gaussian elimination for a moment. Take as a given that, for a "sufficiently nice" $n \times n$ matrix A, the matrix L^{-1} that takes A to its ***upper-triangular*** or ***row echelon form***, U, has the structure:

$$L^{-1} = L_{n-1}L_{n-2}\dots L_2L_1$$

Each of the $L_i$is above is an elementary matrix that zeros out the subdiagonal entries of the i^{th} column of A. This is **the i^{th} step of *Gaussian Elimination* applied to *the entire i^{th} column of A below the i^{th} diagonal element***.

Let's show this by computation of L_i for a "nice" matrix A.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Import all necessary packages
%matplotlib inline
import scipy.sparse as sparse #this helps to speed up the algorithms, but you will no
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
sym.init_printing(use_unicode=True)

## These will allow us to see a cool simulation of the Heat Equation problem (if we c
from matplotlib import animation, rc
from IPython.display import HTML
```

Gaussian Elimination by Elementary Matrices, L_i

Let A be the following matrix:

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

Do This

Create a 4×4 **unit lower-triangular** matrix, L_1 that eliminates all of the subdiagonal entries of the first column of A. Display the matrix L_1 using SymPy.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
A = np.matrix([[2,1,1,0],[4,3,3,1],[8,7,9,5],[6,7,9,8]]) # Here is A for your conveni
As = sym.Matrix(A)
As
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Type your answer here ##
L1 = np.matrix([[1,,,],[,1,,],[,,1,],[,,,1]])
```

We should now have the following:

$$L_1 A = A^{(1)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}$$

Since our first row remained unchanged, we know that our u_{1i} (the first row entries of U) are now determined. Similarly, we have that the u_{2i} (the second row entries of U) are determined as well. The x elements are elements that have changed, but are not yet in

their final form. **Note:** Your u_{ij} will be whole, or integer ($\mathbb{Z}$), numbers.

Do This

Left-multiply A by L_1 to confirm that all of the subdiagonal entries of the first column of $A^{(1)}$ are zero. Display the result via SymPy.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Type your answer here ##
```

Our next step will be to eliminate all nonzero entries from the second column of $A^{(1)} = L_1 A$ by left multiplication of L_2. This should yield:

$$\begin{aligned} A^{(2)} &= L_2 A^{(1)} \\ &= L_2 L_1 A \\ &= \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & x & x \end{bmatrix} \end{aligned}$$

Do This

Create a 4×4 **unit lower-triangular** matrix, L_2 that eliminates all of the subdiagonal entries of the second column of $A^{(1)}$ yielding $A^{(2)}$ as above. Display the matrix L_2 using SymPy.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Type your answer here ##
L2 = np.matrix([[1,,,],[,1,,],[,,1,],[,,,1]]) # for your convenience
```

Do This

Left-multiply $A^{(1)}$ by L_2 to confirm that all of the subdiagonal entries of column 2 of $A^{(2)}$ are zero. Display the result via SymPy. **Note**: Your u_{ij} will be whole, or Integer ($\mathbb{Z}$), numbers.


```
## Type your answer here ##
```

We should now have:

$$\begin{aligned} A^{(2)} &= L_2 A^{(1)} \\ &= L_2 L_1 A \\ &= \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & x & x \end{bmatrix} \end{aligned}$$

We now want to build the final matrix L_3 that will take our matrix $A^{(2)}$ to ***upper-triangular* form**. So, let's do that!

Do This

Create a 4×4 **unit lower-triangular** matrix, L_3 that eliminates all of the subdiagonal entries of the third column of $A^{(2)}$ yielding:

$$\begin{aligned} A^{(3)} &= L_3 A^{(2)} \\ &= L_3 L_2 A^{(1)} \\ &= L_3 L_2 L_1 A \\ &= U \end{aligned}$$

Display the matrix L_3 using SymPy.

```
## Type your answer here ##
L3 = np.matrix([[1,,,],[,1,,],[,,1,],[,,,1]]) # for your convenience
```

Do This

Left-multiply $A^{(2)}$ by L_3 to confirm that all of the subdiagonal entries of column 3 of $A^{(3)}$ are zero. Display the result via SymPy. **Note**: Your u_{ij} will be whole, or integer ($\mathbb{Z}$), numbers. You should now notice that $A^{(3)} = U$ is in **row echelon form**, and, hence, U is an **upper-triangular matrix** with 0s below the diagonal!

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Type your answer here ##
```

Congratulations!

You have just decomposed your first matrix via the process below (and should now have a matrix, U, that looks like the one below):

$$\begin{aligned} L^{-1}A &= L_3L_2L_1A \\ &= L_3L_2A^{(1)} \\ &= L_3A^{(2)} \\ &= A^{(3)} \\ &= U \\ &= \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{bmatrix} \end{aligned}$$

Do This

Finally, let's explicitly generate the matrices L^{-1} and L. Then, display them using SymPy.

It will be helpful to use the following:

$$L^{-1} = L_{n-1}L_{n-2}\dots L_2L_1$$

and

$$\begin{aligned} L &= (L^{-1})^{-1} \\ &= (L_{n-1}L_{n-2}\dots L_2L_1)^{-1} \\ &= L_1^{-1}L_2^{-1}\dots L_{n-2}^{-1}L_{n-1}^{-1} \end{aligned}$$

If you're stuck, refer to the paragraph at the beginning of this section for the explicit formula. Recall: $L^{-1}L = LL^{-1} = I$

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Type your answer here ##
```

Do This

Look at all the matrices L_i and see the connections between the final L.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
print(L1)
print(L2)
print(L3)
print(L)
```

For our last bit of LU decomposition fun, let's confirm that your matrices L and U fulfill the equality:

$$A = LU$$

Indeed, there is a function in SymPy that will compute the LU decomposition for us.

Do This

Run the following function and print its outputs:

```
L_actual, U_actual, _ = As.LUdecomposition()
```

Then, compute:

```
L_actual*U_actual - As
```

and confirm that it outputs the zero matrix.


```
## Type your answer here ##
```

General LU Decomposition Algorithm

Do This

Using the scaffolded code below, complete the LU decomposition algorithm. (It may be helpful to test your code on the matrix A from above.)


```
## Type your answer here ##
C = np.matrix([[2,1,1,0],[4,3,3,1],[8,7,9,5],[6,7,9,8]]) # to test

def LU_decomp(B):
    n = len(B)
    U = B.copy()
```

```
    L = np.identity(n)
    for k in np.arange(0,n-1):
        for j in np.arange(k+1,n):
            L[j,k] =
            U[j,k:n] = U[,:] - L[,]*U[,:]
    return np.mat(L), np.mat(U)

L1,U1 = LU_dec(C) # syntax for returning matrices
np.linalg.norm(L1*U1 - A) # Test: should return 0
```

Solve $Ax = b$ via LU Decomposition

You may wish to refer to the introduction of this assignment for a general overview of how to use LU Decomposition to solve $Ax = b$.

Do This

Using the scaffolded code below, complete the LU solver algorithm. The algorithm should solve $Ly = b$ for y via Forward-Substitution and then $Ux = y$ for x by Backward-Substitution. (It may be helpful to test your code on a matrix A and vector b from homework 1 or another source.)


```
## Type your answer here ##
def LU_Axb_solve(A,b):
    L,U = LU_decomp(A)
    n = len(A)
    # Forward-Substitution: Ly = b for y
    y = np.zeros((,))
    for i in np.arange(0,n):
        y[i] = b[i].copy()
        for j in np.arange(0,i):
            y[] = y[] - L[,]*y[]

    # Backward-Substitution: Ux = y for x
    x = np.zeros((n,1))
    for i in np.arange(n-1,-1,-1):
        x[] = y[].copy()
        for j in np.arange(n-1,i,-1):
```

```
            x[] = x[] - U[,]*x[]
        x[] = x[]/U[,]

    return np.mat(x)
```

CHAPTER OVERVIEW

35: 18 Pre-Class Assignment - Inner Product

35.0: Introduction

Goals for today's pre-class assignment

1. Inner Products
2. Inner Product on Functions
3. Assignment wrap-up

35.1: Inner Products

Definition: An **inner product** on a vector space V (Remember that R^n is just one class of vector spaces) is a function that associates a number, denoted as $\langle u, v\rangle$, with each pair of vectors u and v of V. This function satisfies the following conditions for vectors u, v, w and scalar c:

- $\langle u, v\rangle = \langle v, u\rangle$ (symmetry axiom)
- $\langle u+v, w\rangle = \langle u, w\rangle + \langle v, w\rangle$ (additive axiom)
- $\langle cu, v\rangle = c\langle v, u\rangle$ (homogeneity axiom)
- $\langle u, u\rangle \geq 0$ and $\langle u, u\rangle = 0$ if and only if $u = 0$ (positive definite axiom)

The dot product of R^n is an inner product. Note that we can define new inner products for R^n.

Norm of a vector

Definition: Let V be an inner product space. The **norm** of a vector v is denoted by $\|v\|$ and is defined by:

$$\|v\| = \sqrt{\langle v, v\rangle}.$$

Angle between two vectors

Definition: Let V be a real inner product space. The **angle θ between two nonzero vectors u and v** in V is given by:

$$\cos(\theta) = \frac{\langle u, v\rangle}{\|u\|\|v\|}.$$

Orthogonal vectors

Definition: Let V be an inner product space. Two vectors u and v in V are **orthogonal** if their inner product is zero:

$$\langle u, v\rangle = 0.$$

Distance

Definition: Let V be an inner product space. The **distance between two vectors (points) u and v** in V is denoted by $d(u, v)$ and is defined by:

$$d(u, v) = \|u - v\| = \sqrt{\langle u - v, u - v\rangle}$$

Example:

Let R^2 have an inner product defined by: $\langle (a_1, a_2), (b_1, b_2)\rangle = 2a_1b_1 + 3a_2b_2$.

Question 1

What is the norm of (1,-2) in this space?

Question 2

What is the distance between (1,-2) and (3,2) in this space?

Question 3

What is the angle between (1,-2) and (3,2) in this space?

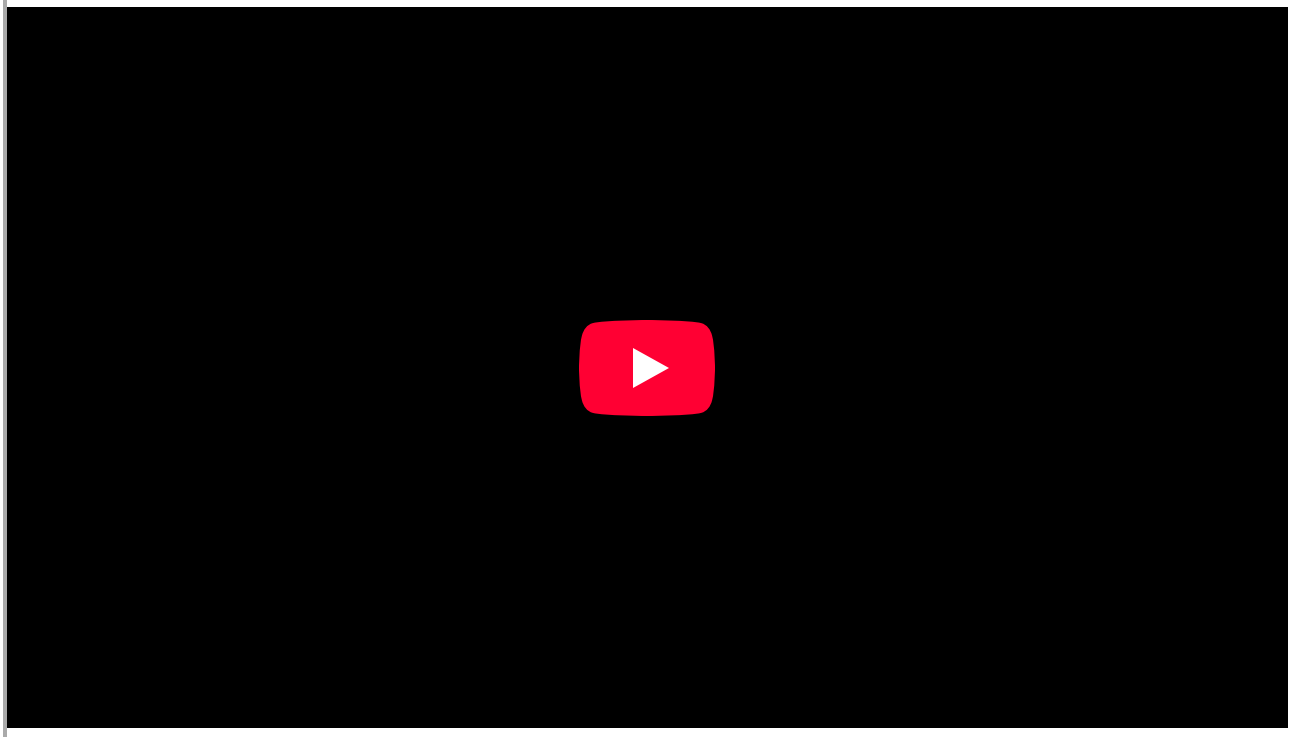
Question 4

Determine if (1,-2) and (3,2) are orthogonal in this space?

35.2: Inner Product on Functions

```
from IPython.display import YouTubeVideo
YouTubeVideo("8ZyeHtgMBjk",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



Example

Consider the following functions

\[f(x)=3x-1 \nonumber\]

\[g(x)=5x+3 \nonumber\]

\[\text{with inner product defined by }\langle f,g\rangle=\int_0^1{f(x)g(x)dx}. \nonumber\]

Question 5

What is the norm of \(f(x)\) in this space?
(Hint: you can use `sympy.integrate` to compute the integral)

Question 6

What is the norm of \(g(x)\) in this space?

Question 7

What is the inner product of \(f(x)\) and \(g(x)\) in this space?

35.3: Assignment wrap-up

Assignment-Specific Question

There is no Assignment specific question for this notebook.

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

36: 18 In-Class Assignment - Inner Products

36.0: Introduction

Image from: Wikipedia

Agenda for today's class (80 minutes)

1. (20 minutes) Pre-class Review
2. (30 minutes) Minkowski Geometry
3. (30 minutes) Function Approximation

36.1: Pre-class Review

- 18 Pre-Class Assignment: Inner Product

An inner product on a real vector space V is a function that associates a number, denoted as $\langle u, v\rangle$, with each pair of vectors u and v of V. This function satisfies the following conditions for vectors u, v, w and scalar c:

$$\langle u, v\rangle = \langle v, u\rangle \text{ symmetry axiom}$$

$$\langle u+v, w\rangle = \langle u, w\rangle + \langle v, w\rangle \text{ additive axiom}$$

$$\langle cu, v\rangle = c\langle v, u\rangle \text{ homogeneity axiom}$$

$$\langle u, v\rangle = \langle v, u\rangle \text{ Symmetry axiom}$$

$$\langle u, u\rangle \geq 0 \text{ and } \langle u, u\rangle = 0 \text{ if and only if } u = 0 \text{ positive definite axiom}$$

The dot product of R^n is an inner product. However, we can define many other inner products.

Norm of a vector

Definition: Let V be an inner product space. The **norm** of a vector v is denoted by $\|v\|$ and is defined by:

$$\|v\| = \sqrt{\langle v, v\rangle}.$$

Angle between two vectors

Definition: Let V be a real inner product space. The **angle θ between two nonzero vectors u and v** in V is given by:

$$cos(\theta) = \frac{\langle u, v\rangle}{\|u\|\|v\|}.$$

Orthogonal vectors

Definition: Let V be an inner product space. Two vectors u and v in V are **orthogonal** if their inner product is zero:

$$\langle u, v\rangle = 0.$$

Distance

Definition: Let V be an inner product space. The **distance between two vectors (points) u and v** in V is denoted by $d(u, v)$ and is defined by:

$$d(u, v) = \|u - v\| = \sqrt{\langle u - v, u - v\rangle}$$

36.2: Minkowski Geometry

Consider the following pseudo inner-product which is used to model special relativity in R^4:

$$\langle X, Y\rangle = -x_1y_1 - x_2y_2 - x_3y_3 + x_4y_4$$

It has the following norms and distances:

$$\|X\| = \sqrt{|\langle X, X\rangle|}$$

$$d(X,Y) = \|X - Y\| = \|(x_1 - y_1, x_2 - y_2, x_3 - y_3, x_4 - y_4)\|$$

$$= \sqrt{|-(x_1 - y_1)^2 - (x_2 - y_2)^2 - (x_3 - y_3)^2 + (x_4 - y_4)^2|}$$

Question

The Minkowski Geometry is called pseudo inner product because it violates one of the inner product axioms. Discuss the axioms in your group and decide which one it violates.

The Physical Interpretation of Minkowski Geometry

> *The distance between two points on the path of an observer in Minkowski geometry corresponds to the time recorded by that observer in traveling between the two points.*

We assume that Alpha Centauri lies in the x_1 direction from the Earch. The twin on Earth advances in time x_4. There is no motion in either the x_2 or x_3 directions. Twin 2 on board the rocket advances in time and moves toward Alpha Centauri and back to the Earth.

Let $P = (0,0,0,0)$, $R = (4,0,0,5)$, and $Q = (0,0,0,10)$.

- $d(P,Q) = 10$ means that Twin 1 ages 10 years from P to Q. Because x_1 does not change and only the time x_4 changes. Twin 1 does not travel and stay on Earth for 10 years.
- $d(P,R) = 3$ means that Twin 2 ages 3 years in traveling from P to R. When Twin 2 arrives at the R, the time on the earth has passed 55 years, though the recored time by Twin 2 is only 33 years.
- $d(R,Q) = 3$ means taht Twin 2 ages 3 years in traveling from R to Q. When Twin 2 travels back to the Earth P, it records 3 years but the time at the Earch has passed 5 years.
- The time from $P \to R \to Q$ is shorter than $P \to Q$.

Question

The star cluster Pleiades in the constellation Taurus is 410 light years from Earth. A generational spaceship to the cluster traveling at constant speed ages 850 years on a round trip. By the time the spaceship returns to Earth, how many centuries will have passed on Earth?

Question

How fast was the spaceship going relative to earth?

36.3: Function Approximation

Definition

Let $C[a,b]$ be a vector space of all possible continuous functions over the interval $[a,b]$ with inner product: $\langle f,g \rangle = \int_a^b f(x)g(x)\,dx.$

Now let f be an element of $C[a,b]$, and W be a subspace of $C[a,b]$. The function $g \in W$ such that $\int_a^b \left[f(x) - g(x) \right]^2 dx$ is a minimum is called the least-squares approximation to f.
The least-squares approximation to f in the subspace W can be calculated as the projection of f onto W:

$$g = proj_Wf$$

If $\{g_1, \ldots, g_n\}$ is an orthonormal basis for W, we can replace the dot product of R^n by an inner product of the function space and get:

$$prog_Wf = \langle f,g_1 \rangle g_1 + \ldots + \langle f,g_n \rangle g_n$$

Polynomial Approximations

An orthogonal bases for all polynomials of degree less than or equal to n can be computed using Gram-schmidt orthogonalization process. First we start with the following standard basis vectors in W

$$\{ 1, x, \ldots, x^n \}$$

The Gram-Schmidt process can be used to make these vectors orthogonal. The resulting polynomials on $[-1,1]$ are called Legendre polynomials. The first six Legendre polynomial basis are:

$$1$$
$$x$$
$$x^2 -\frac{1}{3}$$
$$x^3 - \frac{3}{5}x$$
$$x^4 - \frac{6}{7}x^2 + \frac{3}{35}$$
$$x^5 - \frac{10}{9}x^3 + \frac{5}{12}x$$

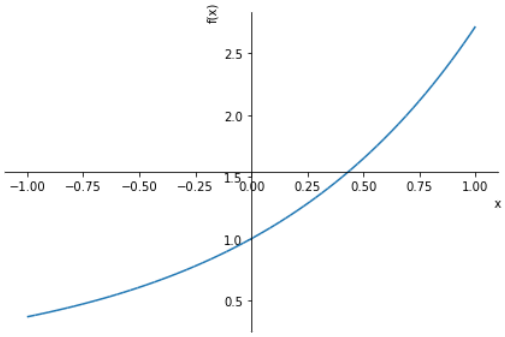
Question

What is the least-squares linear approximations of $f(x)=e^x$ over the interval $[-1,1]$. In other words, what is the projection of f onto W, where W is a first order polynomal with basis vectors $(\{1, x\}$ $(\textit{i.e. } n=1)$.
(Hint: You can give the answer in integrals without computing the integrals. Note the Legendre polynomials are not normalized.)

Here is a plot of the equation $f(x)=e^x$:

```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np

#px = np.linspace(-1,1,100)
#py = np.exp(px)
#plt.plot(px,py, color='red');
import sympy as sym
from sympy.plotting import plot
x = sym.symbols('x')
f = sym.exp(x)
plot(f,(x,-1,1))
```

run | restart | restart & run all



We can use `sympy` to compute the integral. The following code compute the definite integral of \(\int_{-1}^1 e^x dx\). In fact, `sympy` can also compute the indefinite integral by removing the interval.

```
sym.init_printing()
x = sym.symbols('x')
sym.integrate('exp(x)',(x, -1, 1))
#sym.integrate('exp(x)',(x))
```

run | restart | restart & run all

Use `sympy` to compute the first order polynomial that approximates the function \(e^x\). The following calculates the above approximation written in `sympy` :

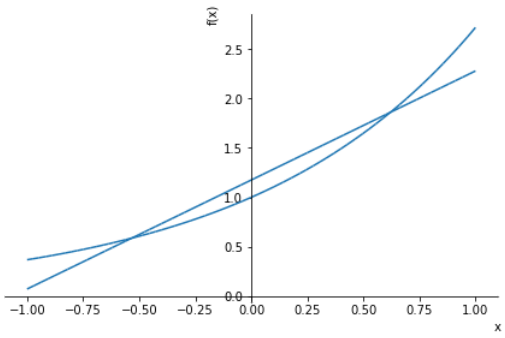
```
g_0 = sym.integrate('exp(x)*1',(x, -1, 1))/sym.integrate('1*1',(x,-1,1))*1
g_1 = g_0 + sym.integrate('exp(x)*x',(x,-1,1))/sym.integrate('x*x',(x,-1,1))*x
g_1
```

run | restart | restart & run all

Plot the original function \(f(x)=e^x\) and its approximation.

```
p2 = plot(f, g_1,(x,-1,1))
```

run | restart | restart & run all




```
#For fun, I turned this into a function:
x = sym.symbols('x')

def lsf_poly(f, gb = [1,  x], a =-1, b=1):
    proj = 0
    for g in gb:
```

```
#        print(sym.integrate(g*f,(x,a,b)))
        proj = proj + sym.integrate(g*f,(x,a,b))/sym.integrate(g*g,(x,a,b))*g
    return proj

lsf_poly(sym.exp(x))
```

run | restart | restart & run all

Question

What would a second order approximation look like for this function? How about a fifth order approximation?

```
#####Start your code here #####
x = sym.symbols('x')
g_2 =
g_2
#####End of your code here#####
```

run | restart | restart & run all

```
p2 = plot(f, g_2,(x,-1,1))
```

run | restart | restart & run all

CHAPTER OVERVIEW

37: 19 Pre-Class Assignment - Least Squares Fit (Regression)

37.0: Introduction

Readings for this topic (Recommended in bold)

- Heffron Chapter 3 pg 287-292
- ***Boyd Chapter 13 pg 225-239***

Goals for today's pre-class assignment

1. Least Squares Fit
2. Linear Regression
3. One-to-one and Inverse transform
4. Inverse of a Matrix
5. Assignment Wrap-up

37.1: Least Squares Fit

Review Chapters Chapter 13 pg 225-239 of the Boyd textbook.

In this first part of this course, we try to solve the system of linear equations $Ax = b$ with an $m \times n$ matrix A and a column vector b.

There are three possible outcomes: an unique solution, no solution, and infinite many solutions. (Review the material on this part if you are no familiar with when the three types of outcomes happen.)

When $m < n$, we call the matrix A underdeterminated, because we can not have an unique solution for it. When $m > n$, we call the matrix A overdeterminated, becasue we may not have a solution with high probability.

However, if we still need to find a best x, even when there is no solution or infinite many solutions we use a technique called least squares fit (LSF). Least squares fit find x such that $\|Ax - b\|$ is the smallest (i.e. we try to minimize the estimation error).

- When there is no solution, we want to find x such that $Ax - b$ is small (here, we want $\|Ax - b\|$ to be small).
- If the null space of A is just $\{0\}$, we can find an unique x to obtain the smallest $\|Ax - b\|$.
  - If there is a unique solution x^* for $Ax = b$, then x^* is the optimal x to obtain the smallest $\|Ax - b\|$, which is 0.
  - Because the null space of A is just $\{0\}$, you can not have infinite many solutions for $Ax = b$.
- If the null space of A is not just $\{0\}$, we know that we can always add a nonzero point x_0 in the null space of A to a best x^*, and $\|A(x^* + x_0) - b\| = \|Ax^* - b\|$. Therefore, when we have multiple best solutions, we choose to find the x in the rowspace of A, and this is unique.

Question 1

Let $A = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $b = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix}$. Find the best x such that $\|Ax - b\|$ has the smallest value.

Question 2

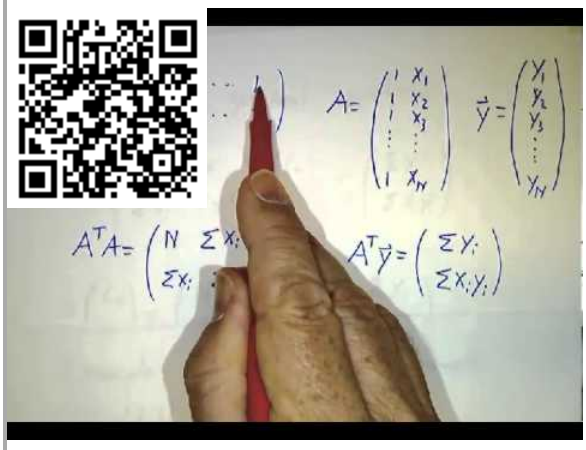
Compute $(A^\top A)^{-1} A^\top b$.

37.2: Linear Regression

Watch the video for using Least Squares to do linear regression.

```
from IPython.display import YouTubeVideo
YouTubeVideo("Lx6CfgKVIuE",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



Question 3

How to tell it is a good fit or a bad one?

37.3: One-to-one and Inverse transform

Read Section 4.9 of the textbook if you are not familiar with this part.

Definition

A transformation $T : U \mapsto V$ is said to be *one-to-one* if each element in the range is the image of just one element in the domain. That is, for two elements (x and y) in U. $T(x) = T(y)$ happens only when $x = y$.

Theorem

Let $T : U \mapsto V$ be a one-to-one linear transformation. If $\{u_1, \ldots, u_n\}$ is linearly independent in U, then $\{T(u_1), \ldots, T(u_n)\}$ is linearly independent in V.

Definition

A linear transformation $T : U \mapsto V$ is said to be *invertible* if there exists a transformation $S : V \mapsto U$, such that $S(T(u)) = u, \qquad T(S(v)) = v,$ for any v in V and any u in U.

Question 4

If linear transformation $T : U \mapsto V$ is invertible, and the dimension of U is 2, what is the dimension of V? Why?

37.4: Inverse of a Matrix

- Recall the four fundamental subspaces of a $m \times n$ matrix A
  - The rowspace and nullspace of A in R^n
  - The columnspace and the nullspace of $A^\top$ in R^m
- The two-sided inverse gives us the following $AA^{-1} = I = A^{-1}A$
  - For this we need $r = m = n$, here r is the rank of the matrix.
- For a left-inverse, we have the following
  - Full column rank, with $r = n \leq m$ (but possibly more rows)
  - The nullspace contains just the zero vector (columns are independent)
  - The rows might not all be independent
  - We thus have either no or only a single solution to $Ax = b$.
  - $A^\top$ will now also have full row rank
  - From $(A^\top A)^{-1}A^\top A = I$ follows the fact that $(A^\top A)^{-1}A^\top$ is a left-sided inverse
  - Note that $(A^\top A)^{-1}A^\top$ is a $n \times m$ matrix and A is of size $m \times n$, theire mulitiplication $(A^\top A)^{-1}A^\top A$ results in a $n \times n$ identity matrix
  - The $A(A^\top A)^{-1}A^\top$ is a $m \times m$ matrix. BUT $A(A^\top A)^{-1}A^\top \neq I$ if $m \neq n$. The matrix $A(A^\top A)^{-1}A^\top$ is the projection matrix onto the column space of A.

Question 5

What is the projection matrix that projects any vector onto the subspace spanned by $[1, 2]^\top$. (What matrix will give the same result as projecting any point onto the vector $[1, 2]^\top$.)

Question 6

If $m = n$, is the left inverse the same as the inverse?

Theorem

For a matrix A with $r = n < m$, the columnspace of A has dimension $r(= n)$. The linear transfrom $A : R^n \mapsto R^m$ is one-to-one. In addition, the linear transformation A from R^n to the columnspace of A is one-to-one and onto (it means that for any element in the columnspace of A, we can find x in R^n such that it equals Ax.) Then the left inverse of A is a one-to-one mapping from the columnspace of A to R^n, and it can be considered as an inverse transform of A.

37.5: Assignment wrap-up

Assignment-Specific Question

There is no Assignment specific question for this notebook.

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

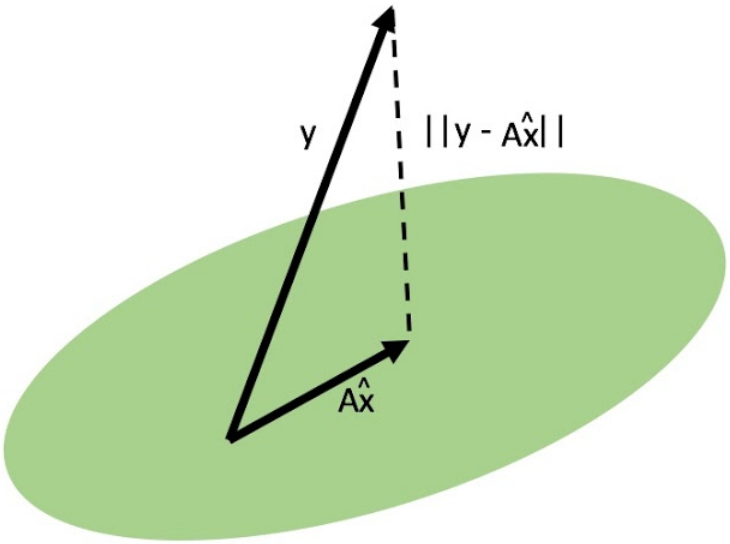
Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

38: 19 In-Class Assignment - Least Squares Fit (LSF)

38.0: Introduction



Agenda for today's class (80 minutes)

1. (20 minutes) LSF Pre-class Review
2. (30 minutes) Finding the best solution in an overdetermined system
3. (30 minutes) Pseudoinverse

38.1: LSF Pre-class Review

- 19 Pre-Class Assignment: Least Squares Fit (Regression)

38.2: Finding the best solution in an overdetermined system


```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
import time
sym.init_printing(use_unicode=True)
```

Let $Ax = y$ be a system of m linear equations in n variables. A *least squares solution* of $Ax = y$ is an solution $\hat{x}$ in R^n such that:

$$\min_{\hat{x}} \|y - A\hat{x}\|$$

Note we substitute y for our typical variable b here because we will use b later to represent the intercept to a line and we want to try and avoid confusion in notation. It also consistent with the picture above.

In other words, $\hat{x}$ is a value of x for which Ax is as close as possible to y. From previous lectures, we know this to be true if the vector $y - A\hat{x}$ is orthogonal (perpendicular) to the column space of A.

We also know that the dot product is zero if two vectors are orthogonal. So we have $a \cdot (Ax - y) = 0$, for all vectors a in the column space of A.

The columns of A span the column space of A. Denote the columns of A as $A = [a_1, \cdots, a_n]$. Then we have $a_1^\top (Ax - y) = 0$, it is the same as taking the transpose of A and doing a matrix multiply $A^\top (Ax - y) = 0$.

$$\begin{aligned} a_1^\top (Ax - y) &= 0, \\ a_2^\top (Ax - y) &= 0 \\ &\vdots \\ a_n^\top (Ax - y) &= 0 \end{aligned}$$

That is:

$$A^\top Ax = A^\top y$$

The above equation is called the *least squares solution* to the original equation $Ax = y$. The matrix $A^\top A$ is symmetric and invertable. Then solving for $\hat{x}$ can be calculated as follows:

$$x = (A^\top A)^{-1} A^\top y$$

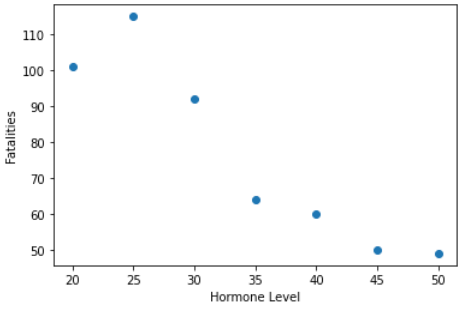
The matrix $(A^\top A)^{-1} A^\top$ is also called the left inverse.

Example

A researcher has conducted experiments of a particular Hormone dosage in a population of rats. The table shows the number of fatalities at each dosage level tested. Determine the least squares line and use it to predict the number of rat fatalities at hormone dosage of 22.

Hormone level	20	25	30	35	40	45	50
Fatalities	101	115	92	64	60	50	49


```
H = [20,25,30,35,40,45,50]
f = [101,115, 92,64,60,50,49]

plt.scatter(H,f)
plt.xlabel('Hormone Level')
plt.ylabel('Fatalities')
f = np.matrix(f).T
```



We want to determine a line that is expressed by the following equation

$$f = aH + b,$$

to approximate the connection between Hormone dosage (H) and Fatalities f. That is, we want to find a (slope) and b (y-intercept) for this line. First we define the variable $x = \begin{bmatrix} a \\ b \end{bmatrix}$ as the column vector that needs to be solved.

Do This

Rewrite the system of equations to the form $Ax = y$ by defining your `numpy` matrices `A` and `y` using the data from above:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your code here
```

Question

Calculate the square matrix $C = A^\top A$ and the modified right hand side vector as $A^\top y$ (Call it `Aty`):

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your code here
```

Question

Find the *least squares solution* by solving $Cx = A^\top y$ for x.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

Login

```
# Put your code here
```

Question

Given the solution above, define the two scalars slope `a` and y-intercept `b` .

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#put your code here
```

The following code will Plot the original data and the line estimated by the coefficients found in the above quation.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
H = [20,25,30,35,40,45,50]
f = [101,115, 92,64,60,50,49]
plt.scatter(H,f)

H2 = np.linspace(np.min(H), np.max(H))

f2 = a * H2 + b

plt.plot(H2, f2)
```

Question

Repeat the above analysis but now with a eight-order polynomial.

Question

Play with the interactive function below by adjusting the degree of the least-squares fit approximation. Then extend the `x_min` and `x_max` parameters. Do you think that an eight-order polynomial is a good model for this dataset? Why or why not?

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from ipywidgets import interact, fixed
import ipywidgets as widgets

@interact(x=fixed(H), y=fixed(f), degree=widgets.IntSlider(min=1, max=8, step=1, valu
def graphPolyN(x, y, x_min, x_max, degree):
    p = np.polyfit(x, y, degree)
    f = np.poly1d(p)

    x_pred = np.linspace(x_min, x_max, 1000)
    y_pred = f(x_pred)

    plt.scatter(x, y, color="red")
    plt.plot(x_pred, y_pred)
```

Question

Check the rank of $C = A^\top A$ for the previous case. What do you get? Why?

38.3: Pseudoinverse


```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
import time
sym.init_printing(use_unicode=True)
```

In this class we often talk about solving problems of the form:

$$Ax = b$$

Currently we have determined that this problem becomes very nice when the $n \times n$ matrix A has an inverse. We can easily multiply each side by the inverse:

$$A^{-1}Ax = A^{-1}b$$

Since $A^{-1}A = I$ the solution for x is simply:

$$x = A^{-1}b$$

Now, let us consider a a more general problem where the $m \times n$ matrix A is not square, i.e. $m \neq n$ and its rank r maybe less than m and/or n. In this case we want to find a Pseudoinverse (which we denote as A^+) which acts like an inverse for a non-square matrix. In other words we want to find an A^+ for A such that:

$$A^+A \approx I$$

Assuming we can find the $n \times m$ matrix A^+, we should then be able to solve for x as follows:

$$Ax = b$$

$$A^+Ax = A^+b$$

$$x \approx A^+b$$

How do we know there is a Pseudoinverse

Assuming the general case of a $m \times n$ matrix A where its rank r maybe less than m and/or n ($r \leq m$ and $r \leq n$). We can conclude the following about the fundamental spaces of A:

- The rowspace of A is in R^n with dimension r
- The columnspace of A is in R^m also with dimension r.
- The nullspace of A is in R^n with dimension $n - r$

- The nullspace of $A^\top$ is in R^m with dimension $m-r$.

Because the rowspace of A and the column space A have the same dimension then A is a the one-to-one mapping from the row space to the columnspace. In other words:

- For any x in the rowspace, we have that Ax is one point in the columnspace. If x' is another point in the row space different from x, we have $Ax \neq Ax'$ (The mapping is one-to-one).
- For any y in the columnspace, we can find x in the rowspace such that $Ax = y$ (The mapping is onto).

The above is not really a proof but hopefully there is sufficient information to convince yourself that this is true.

How to compute pseudoinverse

We want to find the $n \times m$ matrix that maps from columnspace to the rowspace of A, and $x = A^+Ax$, if x is in the rowspace.

- Let's apply SVD on A: $A = U\Sigma V^\top$, where U is a $m \times m$ matrix, $V^\top$ is a $n \times n$ matrix, and Σ is a diagonal $m \times n$ matrix.
  We can decompose the matrices as $A = \begin{bmatrix} \vdots & \vdots \\ U_1 & U_2 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \cdots & V_1^\top & \cdots \\ \cdots & V_2^\top & \cdots \end{bmatrix}$. Here U_1 is of $m \times r$, U_2 is of $m \times (m-r)$, Σ_1 is of $r \times r$, $V_1^\top$ is of $r \times n$, and $V_2^\top$ is of $(n-r) \times n$.
  - The columnspace of U_1 is the columnspace of A, and columnspace of U_2 is the nullspace of $A^\top$.
  - The rowspace of V_1 is the rowspace of A, and rowspace of V_2 is the nullspace of A.
- If x is in the rowspace of A, we have that $V_2^\top x = 0$. We have $Ax = U_1\Sigma_1 V_1^\top x$.
  - If we define a matrix $B = V_1\Sigma_1^{-1}U_1^\top$, we have that $BAx = V_1\Sigma_1^{-1}U_1^\top U_1\Sigma_1 V_1^\top x = V_1V_1^\top x$. That is $BAx = x$ is x is in the rowspace of A.
- The matrix B is the pseudoinverse of matrix A. $A^+ = V_1\Sigma_1^{-1}U_1^\top$ $A^+ = \begin{bmatrix} \vdots & \vdots \\ V_1 & V_2 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \cdots & U_1^\top & \cdots \\ \cdots & U_2^\top & \cdots \end{bmatrix}$.

Example

Let $A = [1, 2]$, we know that $r = m = 1$ and $n = 2$.


```
A = np.matrix([[1,2]])
```

Todo

Calculate the pseudoinverse A^+ of A using the `numpy.linalg` function `pinv`:

```
#put your code here
```

Do This

Compute AA^+ and A^+A


```
#put your code here
```

Question

If x is in the nullspace of A what is the effect of A^+Ax?

Question

If x is in the rowspace of A what is the effect of A^+Ax?

Left inverse is pseudoinverse

We can compute the left inverse of A if $r = n \leq m$. In this case, we may have more rows than columns, and the matrix A has full column rank.

In this case, the SVD of A is $A = U\Sigma V^\top$. Here U is of $m \times n$, Σ is of $n \times n$ and nonsingular, $V^\top$ is of $n \times n$. The pseudoinverse of A is $A^+ = V\Sigma^{-1}U^\top$.

The left inverse of A is $(A^\top A)^{-1}A^\top = (V\Sigma U^\top U\Sigma V^\top)^{-1}V\Sigma U^\top = V(\Sigma\Sigma)^{-1}V^\top V\Sigma U^\top = V\Sigma^{-1}U^\top = A^+$.

Example

Let $A = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, we know that $r = n = 1$ and $m = 2$. Then we have the left inverse.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
A = np.matrix([[1],[2]])
A
```

Do This

Calculate the pseudoinverse $A^\top$ of A.

Do This

Calculate the left inverse of A, and verify that it is the same as $A^\top$.

CHAPTER OVERVIEW

39: 20 In-Class Assignment - Least Squares Fit (LSF)

39.0: Introduction

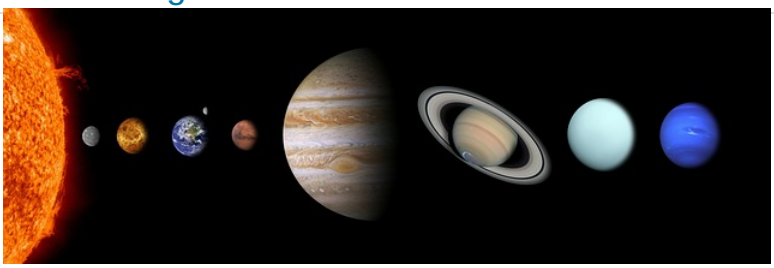
Todays in-class assignment includes multiple Least Squares Fit models. The goal is to see the types of models that can be solved using least squares fit. Even though this is a Linear Algebra Method the models do not need to be linear.

As soon as you get to class, download and start working on this notebook: the instructor will go over solutions but make sure you try to understand and solve them on your own.

Agenda for today's class (80 minutes)

1. LSF Example: Tracking the Planets
2. LSF Example: Predator Pray Model
3. LSF Example: Estimating the best Ellipses

39.1: LSF Example - Tracking the Planets



The following table lists the average distance from the sun to each of the first seven planets, using Earth's distance as a unit of measure (AUs).

Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus
0.39	0.72	1.00	1.52	5.20	9.54	19.2

The following is a plot of the data:


```
# Here are some libraries you may need to use

%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
import math
sym.init_printing()
```

```
distances = [0.39, 0.72, 1.00, 1.52, 5.20, 9.54, 19.2]
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Satern','Uranus']
ind = [1.0,2.0,3.0,4.0,5.0,6.0,7.0]

plt.scatter(ind, distances);
plt.xticks(ind,planets)
plt.ylabel('Distance (AU)')
```

Note

That the above plot does not look like a line, and so finding the line of best fit is not fruitful. It does, however look like an exponential curve (maybe a polynomial?). The following step transforms the distances using the numpy `log` function and generates a plot that looks much more linear.


```
log_distances = np.log(distances)

plt.scatter(ind,log_distances)
plt.xticks(ind,planets)
plt.ylabel('Distance (log(AU))')
```

For this question we are going to find the coefficients (c) for the best fit line of the form $c_1 + c_2 i = \log d$, where i is the index of the planet and d is the distance.

The following code constructs this problem in the form $Ax = b$ and define the A matrix and the b matrix as `numpy` matrices

```
A = np.matrix(np.vstack((np.ones(len(ind)),ind))).T
b = np.matrix(log_distances).T
sym.Matrix(A)
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
sym.Matrix(b)
```

Do This

Solve for the best fit of $Ax = b$ and define a new variable c which consists of the of the two coefficients used to define the line ($\log d = c_1 + c_2 i$)

Do This

Modify the following code (as needed) to plot your best estimates of c_1 and c_2 against the provided data.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
## Modify the following code

est_log_distances = (c[0] + c[1]*np.matrix(ind)).tolist()[0]
plt.plot(ind,est_log_distances)

plt.scatter(ind,log_distances)
```

```
plt.xticks(ind,planets)
plt.ylabel('Distance (log(AU))')
```

We can determine the quality of this line fit by calculating the root mean squared error between the estimate and the actual data:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
rmse = np.sqrt(((np.array(log_distances)  - np.array(est_log_distances)) ** 2).mean()
rmse
```

Finally, we can also make the plot on the original axis using the inverse of the log (i.e. the exp function):

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
est_distances = np.exp(est_log_distances)
plt.scatter(ind,distances)
plt.plot(ind,est_distances)
plt.xticks(ind,planets)
plt.ylabel('Distance (AU)')
```

The asteroid belt between Mars and Jupiter is what is left of a planet that broke apart. Let's the above calculation again but renumber so that the index of Jupyter is 6, Saturn is 7 and Uranus is 8 as follows:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

```
distances = [0.39, 0.72, 1.00, 1.52, 5.20, 9.54, 19.2]
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Satern','Uranus']
ind = [1,2,3,4,6,7,8]

log_distances = np.log(distances)
```

Do This

Repeat the calculations from above with the updated model. Plot the results and compare the RMSE.


```
## Copy and Paste code from above
```



```
## Copy and Paste code from above
est_log_distances = (c[0] + c[1]*np.matrix(ind)).tolist()[0]

est_distances = np.exp(est_log_distances)
plt.scatter(ind,distances)
```

```
plt.plot(ind,est_distances)
plt.xticks(ind,planets)
plt.ylabel('Distance (AU)')

rmse = np.sqrt(((np.array(log_distances)  - np.array(est_log_distances)) ** 2).mean()
rmse
## Copy and Paste code from above
```

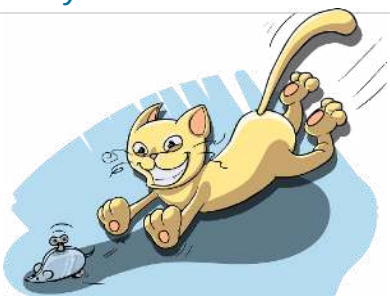
This model of planet location was used to help discover Neptune and prompted people to look for the “missing planet” in position 5 which resulted in the discovery of the asteroid belt. Based on the above model, what is the estimated distance of the asteroid belt and Neptune (index 9) from the sun in AUs?

Hint

You can check your answer by searching for the answer on-line.


```
#Put your prediction calcluation here
```

39.2: LSF Example - Predator Pray Model



The following example example data comes from Mathematica Stack Exchange and represents some experimental data `time` , `x` and `y` .

\[dx = ax - bxy \nonumber \]

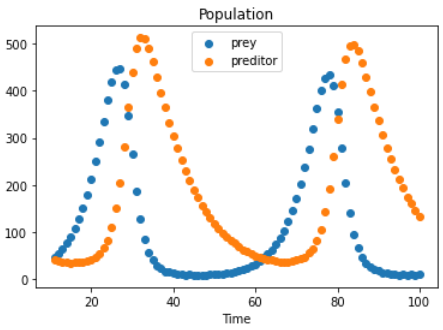
\[dy = -cy + dxy \nonumber \]

The following code plots the data

```
# (* The first column is time 't', the second column is coordinate 'x', and the last
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np

data=[[11,45.79,41.4],
      [12,53.03,38.9],[13,64.05,36.78],
      [14,75.4,36.04],[15,90.36,33.78],
      [16,107.14,35.4],[17,127.79,34.68],
      [18,150.77,36.61], [19,179.65,37.71],
      [20,211.82,41.98],[21,249.91,45.72],
      [22,291.31,53.1],[23,334.95,65.44],
      [24,380.67,83.],[25,420.28,108.74],
      [26,445.56,150.01],[27,447.63,205.61],
      [28,414.04,281.6],[29,347.04,364.56],
      [30,265.33,440.3],[31,187.57,489.68],
      [32,128.,512.95],[33,85.25,510.01],
      [34,57.17,491.06],[35,39.96,462.22],
      [36,29.22,430.15],[37,22.3,396.95],
      [38,16.52,364.87],[39,14.41,333.16],
      [40,11.58,304.97],[41,10.41,277.73],
      [42,10.17,253.16],[43,7.86,229.66],
      [44,9.23,209.53],[45,8.22,190.07],
      [46,8.76,173.58],[47,7.9,156.4],
      [48,8.38,143.05],[49,9.53,130.75],
      [50,9.33,117.49],[51,9.72,108.16],
      [52,10.55,98.08],[53,13.05,88.91],
      [54,13.58,82.28],[55,16.31,75.42],
      [56,17.75,69.58],[57,20.11,62.58],
      [58,23.98,59.22],[59,28.51,54.91],
      [60,31.61,49.79],[61,37.13,45.94],
      [62,45.06,43.41],[63,53.4,41.3],
      [64,62.39,40.28],[65,72.89,37.71],
      [66,86.92,36.58],[67,103.32,36.98],
      [68,121.7,36.65],[69,144.86,37.87],
      [70,171.92,39.63],[71,202.51,42.97],
```

```
      [72,237.69,46.95],[73,276.77,54.93],
      [74,319.76,64.61],[75,362.05,81.28],
      [76,400.11,105.5],[77,427.79,143.03],
      [78,434.56,192.45],[79,410.31,260.84],
      [80,354.18,339.39],[81,278.49,413.79],
      [82,203.72,466.94],[83,141.06,494.72],
      [84,95.08,499.37],[85,66.76,484.58],
      [86,45.41,460.63],[87,33.13,429.79],
      [88,25.89,398.77],[89,20.51,366.49],
      [90,17.11,336.56],[91,12.69,306.39],
      [92,11.76,279.53],[93,11.22,254.95],
      [94,10.29,233.5],[95,8.82,212.74],
      [96,9.51,193.61],[97,8.69,175.01],
      [98,9.53,160.59],[99,8.68,146.12],[100,10.82,131.85]]
data = np.array(data)
t = data[:,0]
x = data[:,1]
y = data[:,2]
plt.scatter(t,x)
plt.scatter(t,y)
plt.legend(('prey', 'preditor'))
plt.xlabel('Time')
plt.title('Population');
```

run | restart | restart & run all



Do This

Use Numerical Differentiation to calculate \(dx\) and \(dy\) from \(x\) and \(y\). See if you can plot \(x\),\(dx\) and \(y\),\(dy\) on a couple of plots. Use the plots to try and check to make sure your results make senes.

```
# Put your answer here
```

run | restart | restart & run all

Do This

Formulate two linear systems (\(Ax=b\)) and solve them using LSF as we did in the pre-class. Use one to solve the first ODE and the second to solve the second ODE. Remember, we are trying to estimate values for \(a\),\(b\),\(c\),\(d\)

```
#Put your answer here.
```

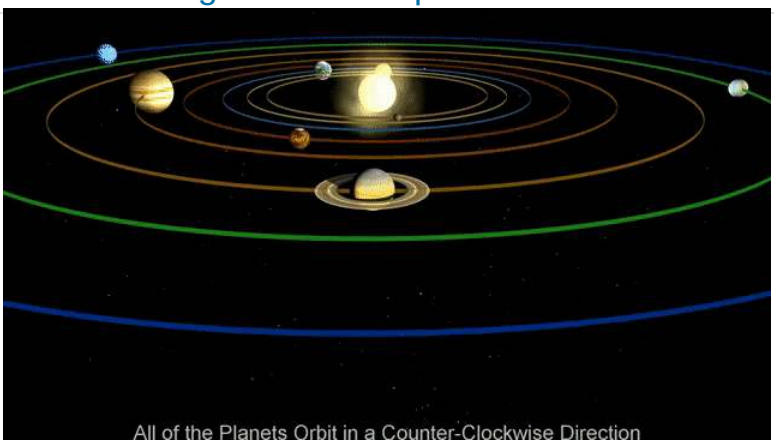
run | restart | restart & run all

Assuming everything worked the following should plot the result.

```
from scipy.integrate import odeint
# The above ODE model sutiable for ODEINT
def deriv(position,t,a,b,c,d):
    x = position[0]
    y = position[1]
    dx = a*x - b*x*y
    dy = -c*y + d*x*y
    return (dx,dy)

# Integrate equations over the time grid, t.
ret = odeint(deriv, (data[0,1],data[0,2]), t, args=(a,b,c,d))

#Plot the model on the data
plt.plot(t,ret)
plt.scatter(t, data[:,1])
plt.scatter(t, data[:,2]);
plt.legend(('x est', 'y est', 'x', 'y'))
plt.xlabel('Time');
```

run | restart | restart & run all

39.3: LSF Example - Estimating the best Ellipses




```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import sympy as sym
sym.init_printing(use_unicode=True)
```

run | restart | restart & run all

Now consider the following sets of points. Think of these as observations of planet moving in an elliptical orbit.

```
x=[0, 1.0, 1.1, -1.1, -1.2, 1.3]
y =[2*1.5, 2*1.0, 2*-0.99, 2*-1.02, 2*1.2, 2*0]

plt.scatter(x,y)
plt.axis('equal')
```

run | restart | restart & run all

In this problem we want to try to fit an ellipse to the above data. First lets look at a general equation for an ellipse:

$$\frac{(u+x)^2}{a^2}+\frac{(v+y)^2}{b^2}=1 \tag{39.3.1}$$

Where u and v are the x and y coordinates for the center of the ellipse and a and b are the lengths of the axes sizes of the ellipse. A quick search on how to plot an ellipse in python comes up with the following example:

```
# Code from: https://stackoverflow.com/questions/10952060/plot-ellipse-with-matplotlil

u=1.     #x-position of the center
v=0.5    #y-position of the center
a=2.     #radius on the x-axis
b=1.5    #radius on the y-axis

t = np.linspace(0, 2*np.pi, 100)
plt.plot( u+a*np.cos(t) , v+b*np.sin(t) )
plt.grid(color='lightgray',linestyle='--')
plt.show()
```

run | restart | restart & run all

Notice this example uses equations of the form:

$$t = [0, \ldots, 2\pi]$$

$$x = u + a\cos(t)$$

$$y = v + b\sin(t)$$

Turns out that this form of the equation is easier to plot and the variables u,v,a,b are the same as our original equation.
Now lets expand the original equation (equation $39.3.1$ from above) and we get the following:

$$x^2 - 2ux - u^2 + y^2 - 2vy + v^2 = r^2 \tag{39.3.2}$$

Question

Why can't we convert equation $39.3.2$ into the form $Ax = b$ and solve using Least Squares Fit? Discuss with your group and be prepared to share your thought with the class.

If we look at our data more closely we can simplify equations $39.3.1$ and $39.3.2$ by assuming the the centroid (u, v) is at the origin. This assumption results in the following equation:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{39.3.3}$$

Notice we can rewrite this into a standard linear set of equations by defining $c_o = \frac{1}{a^2}$ and $c_1 = \frac{1}{b^2}$ and rewriting the equation as follows:

$$c_0 x^2 + c_1 y^2 = 1 \tag{39.3.4}$$

Do This

Given that we know the x and y values of our point observations, equation $39.3.4$ is now linear and can be solved using Least Squares Fit. Using the observation points from above construct `A` and `b` as numpy matrixes for the overdefined system $Ax = b$

```
sym.Matrix(A)
```

run | restart | restart & run all

```
sym.Matrix(b)
```

run | restart | restart & run all

Do This

Solve the above over defined system of linear equations for c_0 and c_1 using LSF.

```
# Put your answer to the above question here
```

run | restart | restart & run all

Assuming we have c in the correct format, we can now calculate a and b from the solution for c_0 and c_1 calculated in the previous step and plot using our plotting code:

```
c = 1/np.sqrt(np.abs(c))
b=c[1,0]
a=c[0,0]
print(a,b)
```

run | restart | restart & run all

```
u=0     #x-position of the center
v=0     #y-position of the center

t = np.linspace(0, 2*np.pi, 100)
plt.plot(u+a*np.cos(t) , v+b*np.sin(t) )
plt.scatter(x,y)
plt.grid(color='lightgray',linestyle='--')
plt.axis('equal');
```

run | restart | restart & run all

CHAPTER OVERVIEW

40: Pre-Class Assignment - Solve Linear Systems of Equations

40.0: Introduction

Goals for today's pre-class assignment

1. Linear Systems
2. Under Defined Systems
3. Invertible Systems
4. Overdefined systems
5. System Properties
6. Assignment wrap up

40.1: Linear Systems

In this course, we learned how to represent linear systems which basically consists of equations added sums of multiple numbers in the form:

$$b = a_1x_1 + a_2x_2 + a_3x_3 + \dots a_mx_m$$

Systems of linear equations are multiple equations of the above form with basically the same unknowns but different values of a and b.

$$b_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots a_{1n}x_n$$

$$b_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots a_{2n}x_n$$

$$b_3 = a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \dots a_{3n}x_n$$

$$\vdots$$

$$b_m = a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \dots a_{mn}x_n$$

The above equations can be represented in matrix form as follows:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ & \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Which can also be represented in “augmented” form as follows:

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & & a_{3n} & b_3 \\ & \vdots & & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & & a_{mn} & b_m \end{array}\right]$$

The above systems can be modified into equivelent systems using combinations of the following operators.

1. Multiply any row of a matrix by a constant
2. Add the contents of one row by another row.
3. Swap any two rows.

Often the 1st and 2nd operator can be combined where a row is multipled by a constanet and then added (or subtracted) from another row.

Question

Consider the matrix $A = \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix}$. What operators can you use to put the above equation into it’s reduced row echelon form?

40.2: Under Defined Systems

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
sym.init_printing()
```

run | restart | restart & run all

An under-defined system is one that is non-invertible and the number of unknowns is more than the number of knowns. These system often have infinite numbers of possible solutions and solving them involves finding a set of simplified equations that represent all solutions.
Often the simplest way to solve an under-defined systems of equations is to extract the solution directly from the reduced row echelon form.

Question

What is the reduced row echelon form of the matrix $A = \begin{bmatrix} 1 & 3 \\ 2 & 6 \end{bmatrix}$.

```
##ADD example of solving an underdefined system.
```

run | restart | restart & run all

Question

What are the solutions to the above systems of equations if $b = \begin{bmatrix} 10 \\ 3 \end{bmatrix}$?

```
#put your answer to the above quesiton here.
```

run | restart | restart & run all

40.3: Invertible Systems

An invertible system has a square A that is invertible such that all the following properties are true:

1. $A^{-1}A = AA^{-1} = I$
2. $(A^{-1})^{-1} = A$
3. $(cA)^{-1} = \frac{1}{c}A^{-1}$
4. $(AB)^{-1} = B^{-1}A^{-1}$
5. $(A^n)^{-1} = (A^{-1})^n$
6. $(A^\top)^{-1} = (A^{-1})^\top$ here $A^\top$ is the transpose of the matrix A.

Consider the following system of equations:

$$\begin{bmatrix} 5 & -2 & 2 \\ 4 & -3 & 4 \\ 4 & -6 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
import numpy as np
import sympy as sym
```

run | restart | restart & run all

```
A = np.matrix([[5, -2, 2], [4, -3, 4], [4,-6,7]])
b = np.matrix([[1],[2],[3]])
display(sym.Matrix(A))
display(sym.Matrix(b))
```

run | restart | restart & run all

Iterative algorithms (Gauss-Seidel and Jacobian):

- They may require many iterations
- Gauss-Seidel is faster than Jacobian
- They do not work for all square invertible systems.

Non-iterative algorithms:

- Gauss elimination (rref)
- LU decomposition
- Find the inverse of the matrix A and $x = A^{-1}b$

Do This

Pick at least two methods to solve the system of equations and compare them.

```
#put your answer here
```

run | restart | restart & run all

40.4: Overdefined systems

```
import numpy as np
import sympy as sym
sym.init_printing()
```

run | restart | restart & run all

We also learned solutions to overdefined systems (more equations than unknowns) often do not exist. However, we can estimate a solution using Least Squares fit.
Consider the following system of equations:

$$\begin{bmatrix} 5 & -2 & 2 \\ 4 & -3 & 4 \\ 4 & -6 & 7 \\ 6 & 3 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ -1 \end{bmatrix}$$

Do This

Solve the above using LSF.

```
#Put your answer to the above question here.
```

run | restart | restart & run all

40.5: System Properties

The above methods for solving systems of linear equations is only part of the story. We also explored ways to understand properties of linear systems. Properties such as rank, determinate, eigenvectors and eigenvalues all provide insight into the matrices that are at the core of the systems.

One problem is that as systems get really large the computational cost of finding a solution can also become large and intractable (i.e. difficult to solve). We use our understanding of matrix properties and "decompositions" to transform systems into simpler forms so that solving the problem also becomes simple.

In class tomorrow we will review all of these concepts by looking at methods to solve linear systmes of the form $Ax = b$ using QR decomposition. When we solve for $Ax = b$ with QR decomposition. We have the following steps:

- Find the QR decomposition of A such that:
  - R is square upper-triangular matrix
  - The Columns of Q are orthonormal
- From $QRx = b$, we obtain $Rx = Q^\top b$
- Solve for x using back substitution.

Do This

Search for a video describing the QR decomposition of a matrix. Try to pick a video that you think does a good job in a short amount of time.

40.6: Assignment wrap up

Assignment-Specific Question

What is the URL you found that describes how to do a QR decomposition.

Question

Summarize what you did in this assignment.

Question

What questions do you have, if any, about any of the topics discussed in this assignment after working through the jupyter notebook?

Question

How well do you feel this assignment helped you to achieve a better understanding of the above mentioned topic(s)?

Question

What was the **most** challenging part of this assignment for you?

Question

What was the **least** challenging part of this assignment for you?

Question

What kind of additional questions or support, if any, do you feel you need to have a better understanding of the content in this assignment?

Question

Do you have any further questions or comments about this material, or anything else that's going on in class?

Question

Approximately how long did this pre-class assignment take?

CHAPTER OVERVIEW

41: 21 In-Class Assignment - Solve Linear Systems of Equations using QR Decomposition

---

41.0: Introduction

Image from: Wikipedia

Agenda for today's class (80 minutes)

1. (20 minutes) Pre-class Review
2. (20 minutes) Solve Linear Systems
3. (30 minutes) Overdetermined Systems
4. (30 minutes) Underdetermined Systems

In this assignment, we try to solve the linear systems $Ax = b$ in three different categories.

- A is a square matrix. Unique solution when A is invertible
- overdetermined (more equations than unknowns): If A has full column rank, the system has an unique solution when b is in the column space of A, otherwise no solution.
- underdetermined (more unknowns than equations): If A has full row rank, there are infinite many solutions.

41.1: Pre-Class Review

- 21 Pre-Class Assignment - Solve Linear Systems of Equations

41.2: Solve Linear Systems

```
import numpy as np
import sympy as sym
```

run | restart | restart & run all

When we have the same number of equations as unknowns, we have the following system $Ax = b$ with a square matrix A. In this section, we assume that the matrix A has full rank. That is the system has an unique solution. We talked about many ways to solve this system of equations. Some examples are:

- Jacobian iteration/ Gauss-Seidel iteration
- $x = A^{-1}b$
- Gaussian elimination
- LU decomposition

In this assignment, we will show that we can solve it by QR decomposion.
Consider the following system of equations:

$$\begin{bmatrix} 5 & -2 & 2 \\ 4 & -3 & 4 \\ 4 & -6 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
A = np.matrix([[5, -2, 2], [4, -3, 4], [4,-6,7]])
b = np.matrix([[1],[2],[3]])
display(sym.Matrix(A))
display(sym.Matrix(b))
```

run | restart | restart & run all

Back substitution

Let's first implement the back substitution in Python. The back substitution function `back_subst` solves the system $Rx = b$ where R is an upper-triangular matrix.

When we solve for x, we start with x_n: $x_n = \frac{b_n}{R_{n,n}}$. Then we solve for x_{n-1} as $x_{n-1} = \frac{b_{n-1} - R_{n-1,n}x_n}{R_{n-1,n-1}}$. Then we can find $x_{n-2}, \cdots, x_1$: $x_{n-2} = \frac{b_{n-2} - R_{n-2,n-1}x_{n-1} - R_{n-2,n}x_n}{R_{n-2,n-2}}$. We can solve for all components of x in the reversed order. So this is called back substitution.

Do This

Complete the following code for back substitution.

```
def back_subst(R,b):
    n = R.shape[0]; x = np.zeros(n);
    for i in reversed(range(n)):
        x[i] = b[i]
        for j in range(i+1,n):
## Your code starts ##
            x[i] =   # Complet this line of code.
## Your code ends ##
        x[i] = x[i]/R[i,i]
    return np.matrix(x).T
```

run | restart | restart & run all

Do This

When we solve for $Ax = b$ with QR decomposition. We have the following steps:

- Find the QR decomposition of A
- From $QRx = b$, we obtain $Rx = Q^\top b$
- Solve for x using back substitution.

Use these steps to solve $Ax = b$ with the given A and b. Compare the result with `np.linalg.solve` .

```
## Your code starts ##
x =
## Your code ends ##
print(type(x))   # x is a column vector in the np.matrix type
np.allclose(x, np.linalg.solve(A,b))
```

run | restart | restart & run all

41.3: Overdetermined Systems

```
import numpy as np
import sympy as sym
```
run | restart | restart & run all

When we have more equations than unknowns, we have the overdetermined system $Ax = b$. In this assignment, we assume that the matrix A has full column rank. Therefore, the system may not be feasible, *i.e.*, we cannot find x such that $Ax = b$. Then we want to find the x to minimize $\|Ax - b\|^2$, which is the least squares problem. We mentioned in previous assignments that we can solve this least squares problem by finding the left inverse of the matrix A. That is $x = (A^\top A)^{-1} A^\top b$.
In this assignment, we show that we can solve it by QR decomposion.
Consider the following system of equations:

$$\begin{bmatrix} 5 & -2 & 2 \\ 4 & -3 & 4 \\ 4 & -6 & 7 \\ 6 & 3 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ -1 \end{bmatrix}$$

Do This

We solve the least squares problem in the following steps

- Find the QR decomposition of the matrix A such that R is a square upper-triangular matrix. (Q is not a square matrix any more)
- Use the `back_subst` function we defined before to solve $Rx = Q^\top b$

```
A = np.matrix([[5, -2, 2], [4, -3, 4], [4,-6,7], [6,3,-3]])
b = np.matrix([[1],[2],[3],[-1]])
display(sym.Matrix(A))
display(sym.Matrix(b))
```
run | restart | restart & run all

```
## Your code starts ##
x =
## Your code ends ##
print(type(x))   # x is a column vector in the np.matrix type
print(x)
```
run | restart | restart & run all

We can not use the `np.linalg.solve` because the matrix A is not a square matrix (you can try if you do not believe it). However, we can use the `np.linalg.lstsq` function to find the least squares solution to minimize $\|Ax - b\|^2$. The next cell compares the result from `lstsq` and our result from the QR decomposition. (If everything is correct, you will expect a `True` result.)

```
np.allclose(x, np.linalg.lstsq(A,b)[0])
```
run | restart | restart & run all

Do This

Explain why we can use the QR decomposition to solve the least squares problem.

41.4: Underdetermined Systems

```
import numpy as np
import sympy as sym
```
run | restart | restart & run all

When we have more unknowns than equations, we have the underdeterminated system $Ax = b$. In this assignment, we assume that the matrix A has full row rank. This system has infinitely many solutions, *i.e.*, we cannot find a unique x such that $Ax = b$. Then we want to find the smallest x (by smallest, we mean the smallest $\|x\|^2$) such that $Ax = b$, which is also the least squares problem. In this assignment, we show that we can also solve it by QR decomposion.
Consider the following system of equations:

$$\begin{bmatrix} 5 & -2 & 2 & 1 \\ 4 & -3 & 4 & 2 \\ 4 & -6 & 7 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Do This

We solve the least squares problem in the following steps

- Find the QR decomposition of the matrix $A^\top$ such that R is a square upper-triangular matrix.
- Use the `forward_subst` function defined below to solve $x = Q(R^\top)^{-1}b$

```
A = np.matrix([[5, -2, 2, 1], [4, -3, 4, 2], [4,-6,7, 4]])
b = np.matrix([[1],[2],[3]])
display(sym.Matrix(A))
display(sym.Matrix(b))
```
run | restart | restart & run all

```
def forward_subst(L,b):  # This function solves $L x= b$ when $L$ is the lower-trigul
    n = L.shape[0]; x = np.zeros(n);
    for i in range(n):
        x[i] = b[i]
        for j in range(i):
            x[i] = x[i] - L[i,j]*x[j]
        x[i] = x[i]/L[i,i]
    return np.matrix(x).T
```
run | restart | restart & run all

```
## Your code starts ##
x =
## Your code ends ##
print(type(x))   # x is a column vector in the np.matrix type
print(x)
```
run | restart | restart & run all

We can not use the `np.linalg.solve` because the matrix A is not a square matrix. However, we can use the `np.linalg.lstsq` function to find the least squares solution to minimize $\|Ax - b\|^2$ with underdeterminated systems. The

next cell compares the result from `lstsq` and our result from the QR decomposition. (If everything is correct, you will expect a `True` result.)

```
np.allclose(x, np.linalg.lstsq(A,b)[0])
```

run | restart | restart & run all

Do This

Explain why we can use the QR decomposition to solve the least squares problem.
(HINT: you may need the orhogonal decomposition to the four fundamental spaces of Q.)

CHAPTER OVERVIEW

42: Supplemental Materials - Python Linear Algebra Packages

42.0: Introduction

This course uses Python to help students gain a practical understanding of how to use Linear Algebra to solve problems. Although students will likely become better programmers this course does not teach programming and assumes that that students have a basic understanding of Python.

This notebook is designed to provide a review of the major Python Packages we will be using in this course and includes some common techniques you can use to avoid problems.

I hope all students will learn something from the videos. However, feel free to run them at a faster speed and/or skip ahead if you feel you know what you are doing.

Overview

1. Matplotlib
2. Review of Python Math Package
3. Review of Python Numpy Package
4. Advanced Python Indexing
5. LaTeX Math
6. Jupyter Tips and Tricks

42.1: Matplotlib

We will be using the `matplotlib` library quite a bit to visualize the concepts in this course. This is a very big library with a lot of components. Here are some basics to get you started.

First, in order to see the figures generated by the `matplotlib` library in a jupyter notebook you will need to add the following like to a code cell somewhere near the top of the notebook. This like of code must run before any figures will display.

```
%matplotlib inline
```

run | restart | restart & run all

Next, we typically we import either the `pylab` or `pyplot` packages from the `matplotlib` library using one of the following import statements. In most cases these statements are interchangeable, however, in this class we will generally stick to using `pyplot` because it has a little more functionality.

```
import matplotlib.pylab as plt
```

run | restart | restart & run all

or

```
import matplotlib.pyplot as plt
```

run | restart | restart & run all

The basic way to plot values is to use the `plot` function as follows:

```
y = [0,1,4,9,16,25,36]
plt.plot(y);
```

run | restart | restart & run all

The `matplotlib` library is big!!! There is no way we can cover all of the topics in this notebook. However, it is not that hard to use and there are plenty of tutorials and examples on the Internet.

Do This

Review the `matplotlib` examples in the Matplotlib Example Gallery.

42.2: Review of Python Math Package

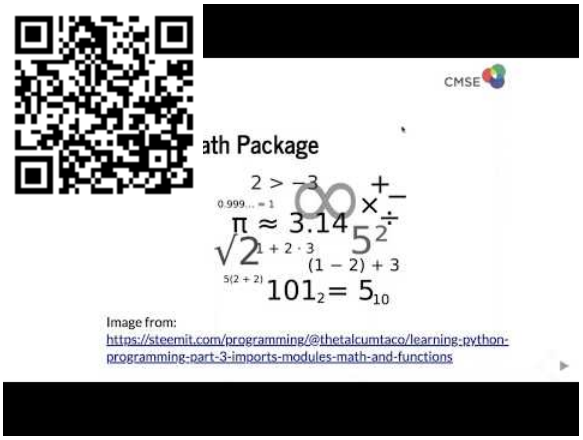
Direct Link to the Youtube video.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("PBlKeuzUf5g",width=640,height=320, cc_load_policy=True)
```



Do This

In the following cell, load the math package and run the `hypot` function with inputs (3,4).

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer here
```

Question

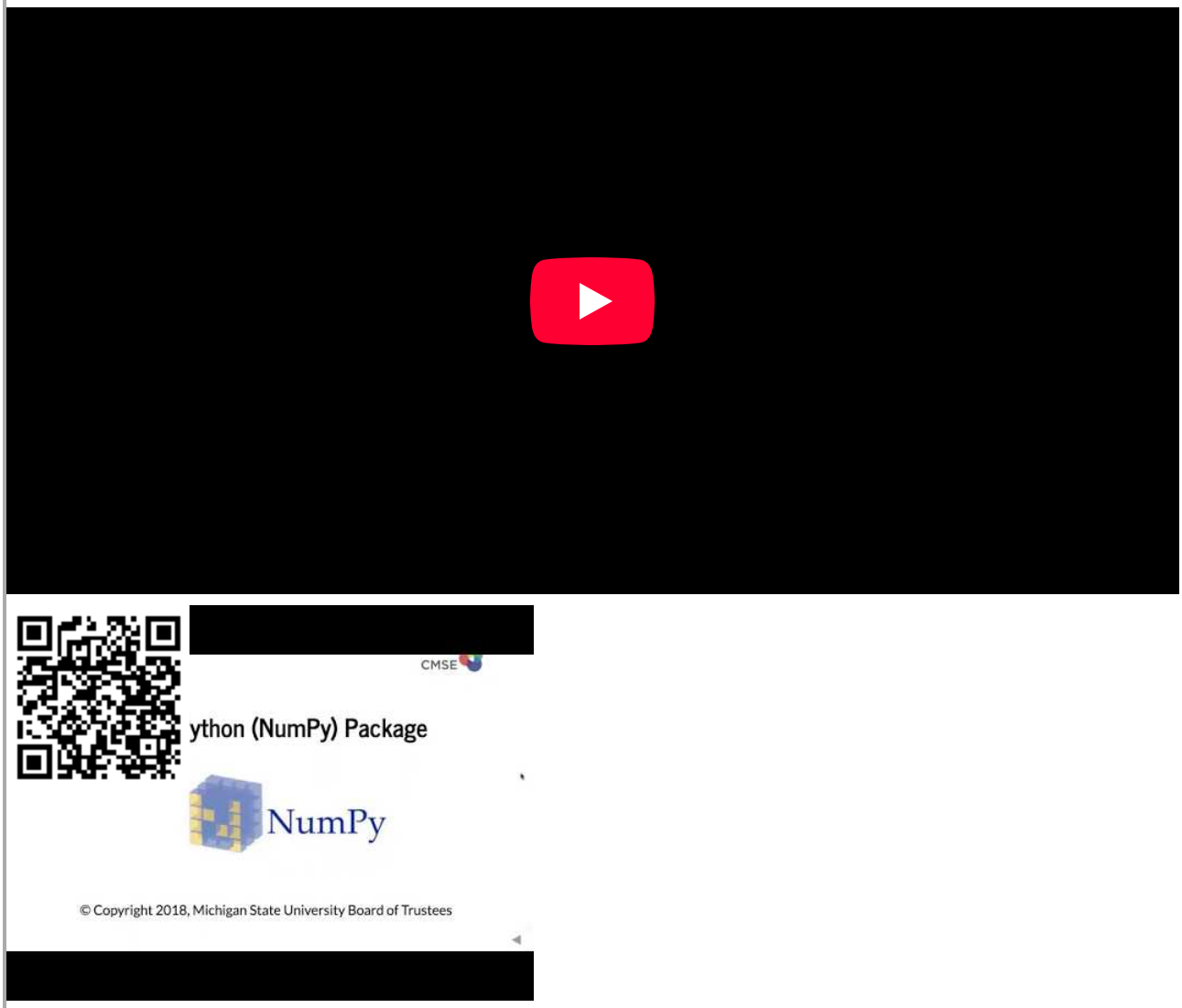
What does the `hypot` function do?

42.3: Review of Python Numpy Package

Direct Link to the Youtube video.

```
from IPython.display import YouTubeVideo
YouTubeVideo("_hbWtNgstlI",width=640,height=320, cc_load_policy=True)
```

run | restart | restart & run all



The Python Numpy library has a “Matrix” object which can be initialized as follows:

```
import numpy as np
A = np.matrix([[1,1], [20,25]])
b = np.matrix([[30],[690]])
print("A="+str(A))
print("b="+str(b))
```

run | restart | restart & run all

```
A=[[ 1  1]
 [20 25]]
b=[[ 30]
 [690]]
```

Python can solve equations in the $Ax = b$ format with the `numpy.linalg` library. For example:

```
import numpy as sp

x = sp.linalg.solve(A, b)
print("X="+str(x))
```

run | restart | restart & run all

```
X=[[12.]
 [18.]]
```

The `numpy.linalg` library is just a subset of the `scipy.linalg` library. Oddly you can't load the SciPy library the same way. Instead you can call it as follows:

```
import scipy.linalg as la

x = la.solve(A, b)
print("X="+str(x))
```

run | restart | restart & run all

```
X=[[12.]
 [18.]]
```

Do This

Convert the following system of linear equations to numpy matrices and solve using a python linear algebra solver $18x + 21y = 22672x - 3y = 644$

```
##Put your answer to the above question here.
```

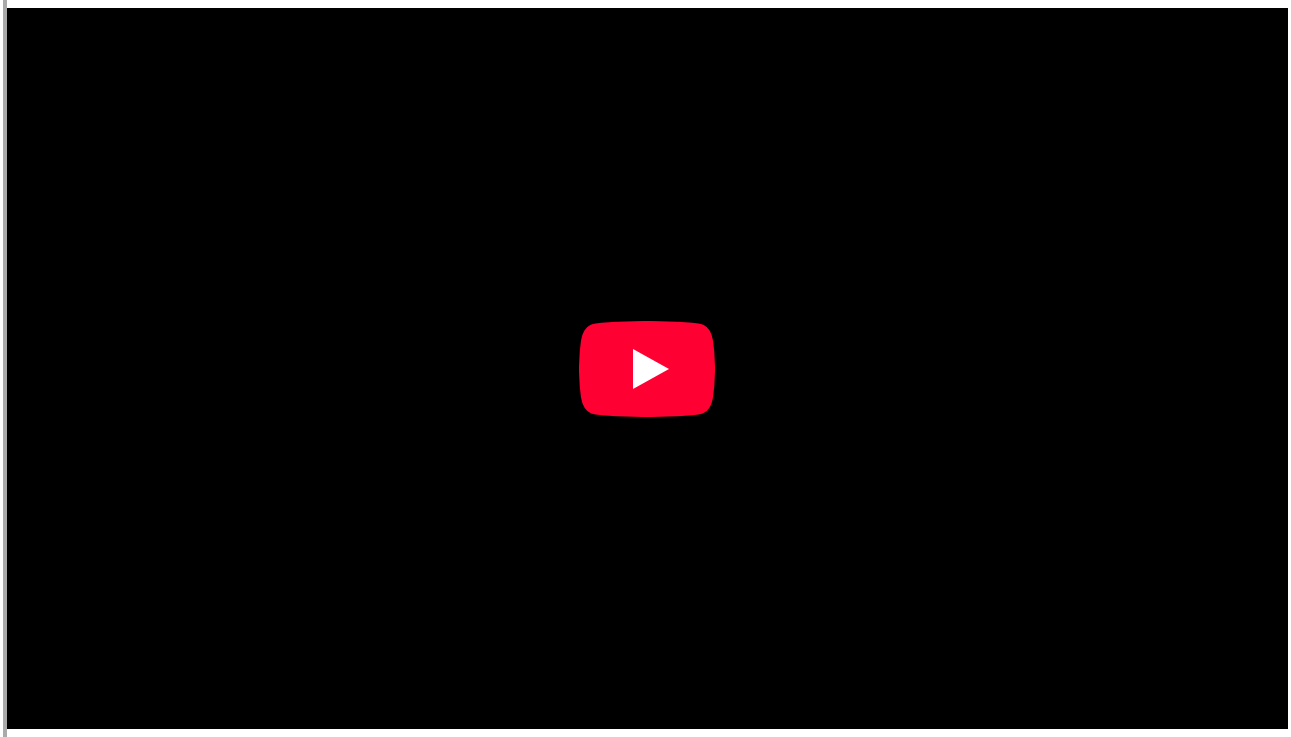
run | restart | restart & run all

42.4: Advanced Python Indexing

This one is a little long and reviews some of the information from the last video. However, I really like using images as a way to talk about array and matrix indexing in `Numpy` .
Direct Link to the Youtube video.

```
from IPython.display import YouTubeVideo
YouTubeVideo("XSyiafkKerQ",width=640,height=360, cc_load_policy=True)
```

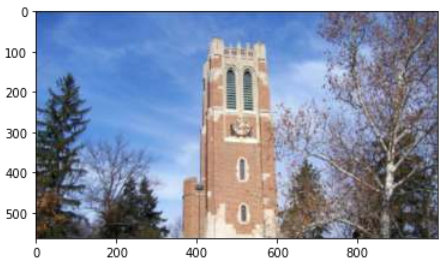
run | restart | restart & run all



```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import imageio

#from urllib.request import urlopen, urlretrieve
```

```
from imageio import imsave

url = 'https://res.cloudinary.com/miles-extranet-dev/image/upload/ar_16:9,c_fill,w_10(
im = imageio.imread(url)

im[10,10,0] = 255
im[10,10,1] = 255
im[10,10,2] = 255

#Show the image
plt.imshow(im);
```

run | restart | restart & run all




```
im[20,20,:] = 255
plt.imshow(im)
```

run | restart | restart & run all

```
cropped = im[0:50,0:50,:]
plt.imshow(cropped)
```

run | restart | restart & run all

```
cropped = im[50:,350:610,:]
plt.imshow(cropped)
```

run | restart | restart & run all

```
red = im[:,:,0]
plt.imshow(red)
plt.colorbar()
```

run | restart | restart & run all

```
#Note python changed slightly since the making of the video.
# We added the astype funciton to ensure that values are between 0-255
red_only = np.zeros(im.shape).astype(int)
red_only[:,:,0] = red

plt.imshow(red_only)
```

run | restart | restart & run all

```
green_only = np.zeros(im.shape).astype(int)
green_only[:,:,1] = im[:,:,1]

plt.imshow(green_only)
```

run | restart | restart & run all

```
blue_only = np.zeros(im.shape).astype(int)
blue_only[:,:,2] = im[:,:,2]

plt.imshow(blue_only)
```

run | restart | restart & run all

Do This

Modify the following code to set all of the values in the blue channel to zero using only one simple line of indexing code.

```
no_blue = im.copy()

#put your code here

plt.imshow(no_blue)
```

run | restart | restart & run all

Question

What was the command you use to set all of the values of blue inside `no_blue` to zero?

42.5: LaTeX Math

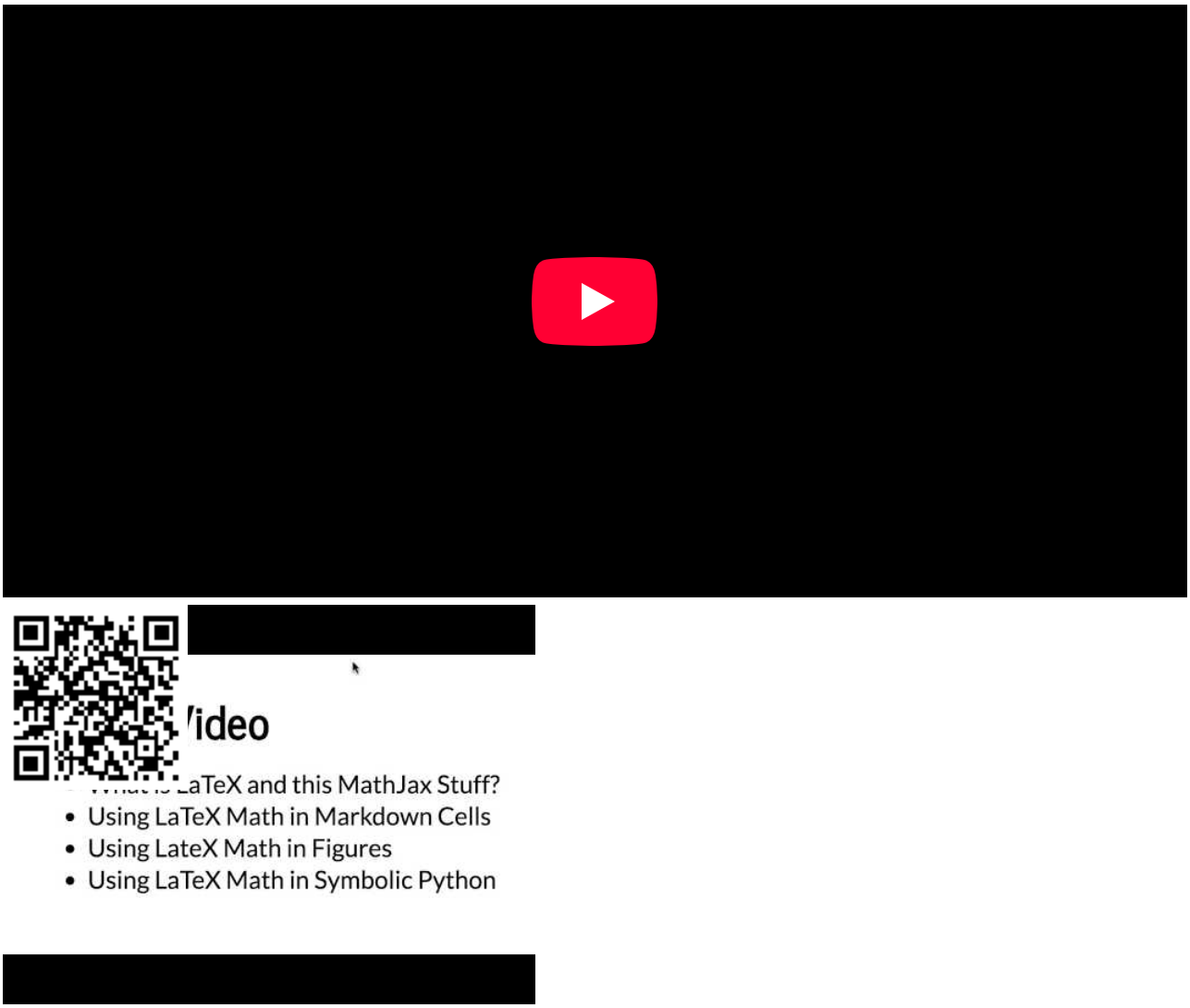
Direct Link to the Youtube video.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("qgSa7n_zQ3A",width=640,height=320, cc_load_policy=True)
```



Since this is a “Matrix Algebra” course, we need to learn how to do ‘matrices’ in LaTeX. Double click on the following cell to see the LaTeX code to build a matrix:

Basic matrix notation:

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 2 & -2 \\ 0 & 1 & 2 \end{bmatrix}$$

Augmented matrix notation:

$$\left[\begin{array}{ccc|c} 1 & 0 & 4 & -10 \\ 0 & 2 & -2 & 3 \\ 0 & 1 & 2 & 1 \end{array}\right]$$

Do This

Using LaTeX, create an augmented matrix for the following system of equations:

$$4x + 2y - 7z = 3$$

$$12x + z = 10$$

$$-3x - y + 2z = 30$$

Question

In LaTeX, what special characters is used to separate elements inside a row?

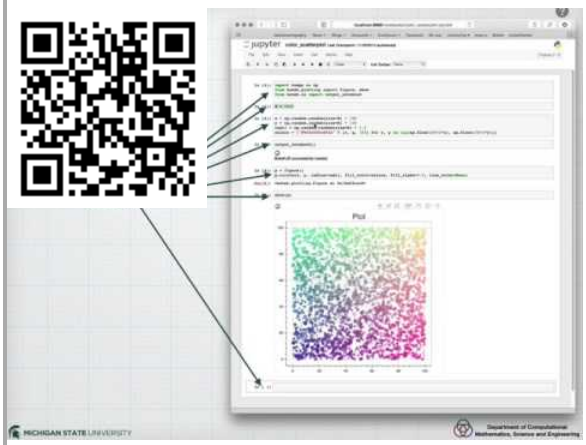
42.6: Jupyter Tips and Tricks

For those of you new to Jupyter notebooks you may want to consider watching the following video as well. More about Jupyter Notebooks.

Direct Link

```
from IPython.display import YouTubeVideo
YouTubeVideo("zSDfRY8-3QE",width=640,height=320, cc_load_policy=True)
```

run | restart | restart & run all



CHAPTER OVERVIEW

43: Jupyter Getting Started Guide

This guide is designed to help students new to Jupyter notebooks get started.

> *The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.*
>
> *From: https://jupyter.org/*

Jupyter works best as a communication tool. Notebooks will be used throughout this class as a way for instructors to communicate with students and for students to communicate with instructors. We will use Jupyter notebooks extensively for pre-class assignments, in-class assignments, homework and Exams.

43.1: Getting Jupyter Working

The first thing you will need to do is to get Jupyter running. There are two basic methods we will be using Jupyter in class. The first is to install it on your computer using Anaconda Python distribution and the second is to use the Web based JupyterHub server put together for the class. The following instructions can help you get started. We recommend learning how to use both methods for class in case there is a problem on one of the systems.

Instructions for downloading Anaconda (Python 3.x.x):

(These instructions are also available via YouTube video: https://youtu.be/3BiLPXAGINA)

1. Go to the Anaconda Download web page: https://www.continuum.io/downloads
2. Use the "Jump to: Windows | OS X | Linux" to pick your operating system.
3. Download the Python 3.x version (64 bit recommended).
4. Follow the directions at the bottom of the page to install Python on your specific operating system.
5. Open the command line program on your computer
    - On windows, type CMD in the run box in the Start menu.
    - On Mac, type "terminal" and hit enter in the Finder window
    - On Linux, open up the console application
6. Type `jupyter notebook` in the command line and hit enter

If everything goes correctly, a browser window should open up with the Jupyter interface running. If things do not work, do not worry; we will help you get started.

Instructions for connecting to the engineering JupyterHub server:

(These instructions are also available via a YouTube video)

Every student enrolled in this class will be given an engineering computing account. If this is your first time using your Engineering account you will need to activate the account by going to the following website:

https://www.egr.msu.edu/decs/myaccount/?page=activate

Enter your MSU NetID. The initial password will be your APID with an @ on the end (example: A12345678@) and then they have to set a password that meets the requirements listed on the page. Verify the password. Then agree to the terms and Activate.

- Once your account is activated you can access the classroom Jupyterhub server using the following instructions:
    1. Open up a web browser and go to the following URL: jupyterhub.egr.msu.edu
    2. Type your engineering login name. This will be your MSU NetID.
    3. Your engineering password.

If everything is working properly you will see the main "Files" windows in the Jupyter interface.

Instructions for getting Jupyter notebook files into Jupyter:

Once you have Jupyter running you will need a notebook file to try out. Jupyter notebooks (also referred to as iPython notebooks) are files that end with the .ipynb extension. We will give you these files for all of your assignments, you will edit them and turn in the edited files in using the course website.

You can download the ipynb assignment files from the course website (http://d2l.msu.edu). Once you have an ipynb file you can load it into Jupyter using the "upload" button on the main "Files" tab in the Jupyter web interface. Hitting this button will cause a file browser window to open. Just navigate to your ipynb file, select it and hit the open button.

Once you see your filename in the jupyter window you can just click on that name to start using that file.

43.2: Example running python code in Jupyter Notebooks

One of the most unique and defining features of Jupyter notebooks is the ability to run code inside of this notebook. This ability makes Jupyter Notebooks especially useful in classes that teach or use programming concepts.

Jupyter notebooks are separated into different types of “cells”. The two major types of cells are; Markdown cells and code cells. Markdown cells (such as this one) consist of formated text, images and equations much like your favorite word processor.

The following are two code cells written in the Python programming language. This simple code is a tool to make it easy to search your jupyter notebooks which can be handy if you are looking for something from a previous class. The example searches for an exact string in your notebook files in the current directory and displays links to the files as output.

To run the code, first select the code cell with your mouse and then hold down the “Shift” key while hitting the “enter” key. You will have to hit the enter key twice to run both cells.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Search string
search_string = "rocket"

#Search current directory
directory ='.'
```

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from pathlib import Path
from IPython.core.display import display, HTML

search_string = search_string.lower()
links=[]
```

```
folder = Path(directory)

files = folder.glob('*.ipynb')
files = sorted(files)
for file in files:
    fn = str(file)
    with open(fn,'r', encoding='utf-8') as fp:
        for line in fp:
            line = line.lower()
            if search_string in line:
                links.append("<a href="+fn+" target=\"_blank\" >"+fn+"</a></br>")
                break
if links:
    display(HTML(' '.join(links)))
else:
    print('string ('+search_string+') not found.')
```

```
string (rocket) not found.
```

43.3: Video review of Python, IPython, and IPython notebooks

Much of this course will be taught in a “flipped” style. This means we give you notebooks to review outside of class and we use in-class time to work on meaningful problems. Many of our pre-class assignments notebooks use videos to help communicate ideas (in lieu of lecture time in class).

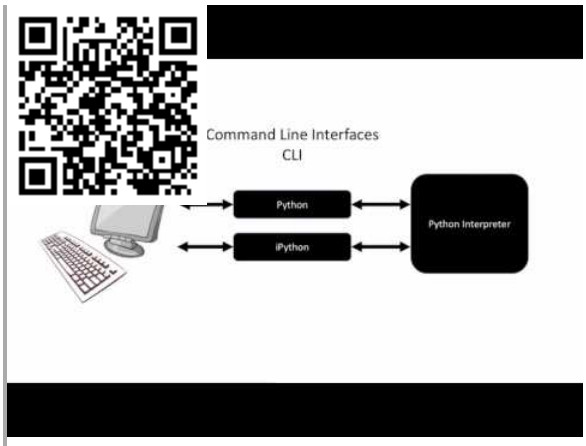
The following two cells will embed the lectures in the notebooks. Run the cells using the “Shift-Enter” key combination described above. Once the video appears just click on the “Play” triangle.

These videos explain Python and Jupyter in more detail.

- Direct link to “Python, iPython, Jupyter” video: https://youtu.be/L03BzGmLUUE
- Alternative Link: https://mediaspace.msu.edu/media/t/0_wxpceyi6

```
# The command below this comment actually displays a specific YouTube video,
# with a given width and height.  You can watch the video in full-screen (much higher
# resolution) mode by clicking the little box in the bottom-right corner of the video
from IPython.display import YouTubeVideo
YouTubeVideo("L03BzGmLUUE",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all

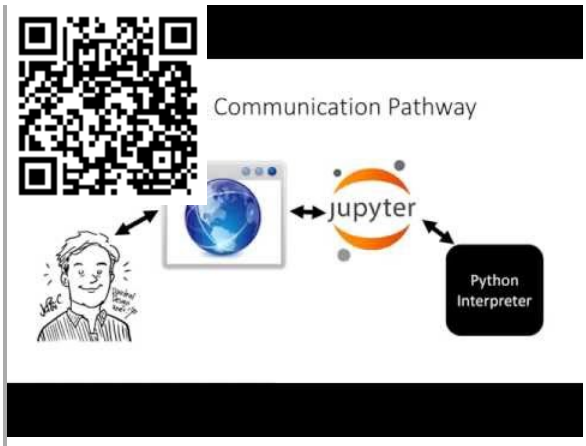


- Direct link to “Working with Jupyter and ipynb files” video: https://youtu.be/5WSQnGmz3IA.
- Alternative Link: https://mediaspace.msu.edu/media/t/0_hkqjufix

Note that the download URL in this video is a little out of date. See the next video or google “Anaconda Python Download”

```
#Python code to display embeded video in jupyter notebook
from IPython.display import YouTubeVideo
YouTubeVideo("5WSQnGmz3IA",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



43.4: Installing Anaconda Python

The following video will introduce you to install Anaconda Python on your personal computer. For this class, make sure you install the latest version (the version in the video is probably old). Also the websites may have been updated. Hopefully that will not be confusing.

- Direct link to "**Install Anaconda**" video: https://youtu.be/3BiLPXAGINA
- Alternative Link:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Python code to display embeded video in jupyter notebook
from IPython.display import YouTubeVideo
YouTubeVideo("3BiLPXAGINA",width=640,height=360, cc_load_policy=True)
```

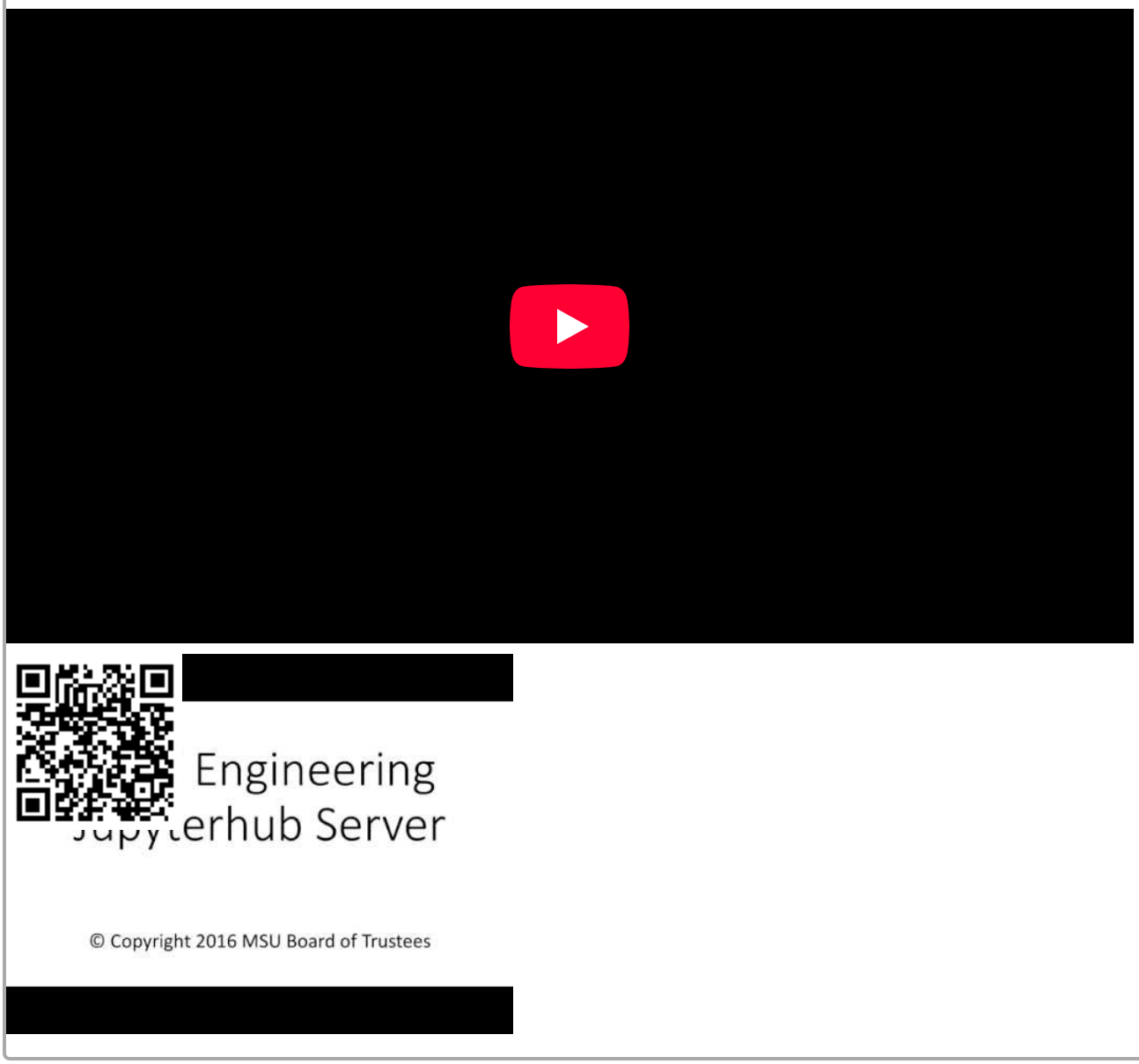
43.5: Introduction to the Engineering Jupyter Account

The following video will introduce you to the Engineering JupyterHub Interface. Please watch this video and answer the questions. Log onto the engineering JupyterHub account using the following link:

- http://jupyterhub.egr.msu.edu
- Direct link to “MSU Engineering Jupyterhub Server” video: https://youtu.be/l7mhi4ww6tY
- Alternative Link: https://mediaspace.msu.edu/media/t/0_brafne0e

```
#Python code to display embeded video in jupyter notebook
from IPython.display import YouTubeVideo
YouTubeVideo("l7mhi4ww6tY",width=640,height=360, cc_load_policy=True)
```

run | restart | restart & run all



43.6: More Information

There are lots of resources on the web for using Python and Jupyter notebooks. The following are some recommended websites for getting more information. If these sites do not work consider using your favorite search engine.

- https://software-carpentry.org/lessons/
- https://docs.python.org/3/tutorial/
- http://pythontutor.com/

CHAPTER OVERVIEW

44: Python Linear Algebra Packages

44.0: Introduction

This tutorial is designed to provide a review of the Python packages we will be using in this course.

I think even experienced Python programmers may learn something from the videos. However, feel free to run them at a faster speed.

Assignment Overview

1. AnswerCheck tool
2. Review of Python Math Package
3. Review of Python Numpy Package
4. Advanced Python Indexing
5. LaTeX Math
6. Assignment wrap-up

44.1: AnswerCheck tool

The `jupytercheck` Package is intended to provide students with immediate feedback to check answers inside of a Jupyter notebook. This was written with a Linear Algebra class in mind so it tries to do a robust comparison and take into consideration different object types as well as round off errors.

It works by providing a function called `answercheck` that takes in a variable to be checked and a "hash" which is a one-way function encoding the answer. The program generates a new hash based on the input variable and compares the two hash values. An output is provided that the answer appears correct or incorrect.

The program is also designed to run without installing anything in python. However, it does require the download of the correct file.

Do This

Two use `answercheck` we will need to download `answercheck.py` to your current working directory. You only really need to do this once. However, if you delete this file by mistake sometime during the semester, you can come back to this notebook and download it again by running the following cell:

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from urllib.request import urlretrieve

urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

Verify you have `answercheck` installed by running the following cell

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from answercheck import checkanswer
checkanswer("Check Answer",'bd8337cd5327e54b2b4b15c6ec3703ed');
```

```
Testing Check Answer
Answer seems to be correct
```

For more information about how `answercheck` works watch the following video:

Direct Link to the Youtube video.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("d4a9Xag-yc8",width=640,height=320, cc_load_policy=True)
```

Note

Make sure you do not change the `checkanswer` commands. The long string with numbers and letters is the secret code that encodes the true answer. This code is also called the HASH. Feel free to look at the `answercheck.py` code and see if you can figure out how it works?

44.2: Review of Python Math Package

Do This

Watch the following video about the python Math package.

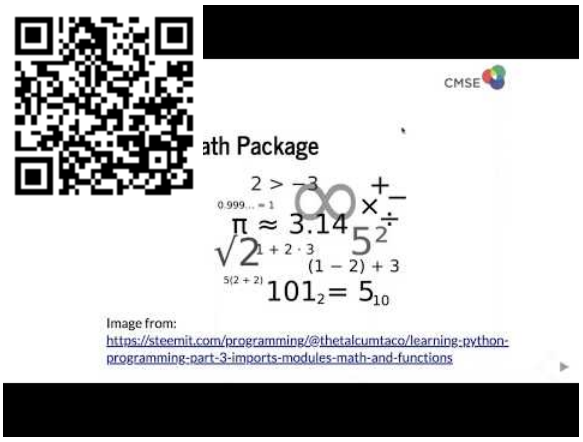
Direct Link to the Youtube video.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("PBlKeuzUf5g",width=640,height=320, cc_load_policy=True)
```



Do This

In the following cell, load the math package and run the `hypot` function with inputs (3,4).

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
#Put your answer here
```

Question

What does the `hypot` function do?

44.3: Review of Python Numpy Package

```
from urllib.request import urlretrieve
urlretrieve('https://raw.githubusercontent.com/colbrydi/jupytercheck/master/answerche
            'answercheck.py');
```

run | restart | restart & run all

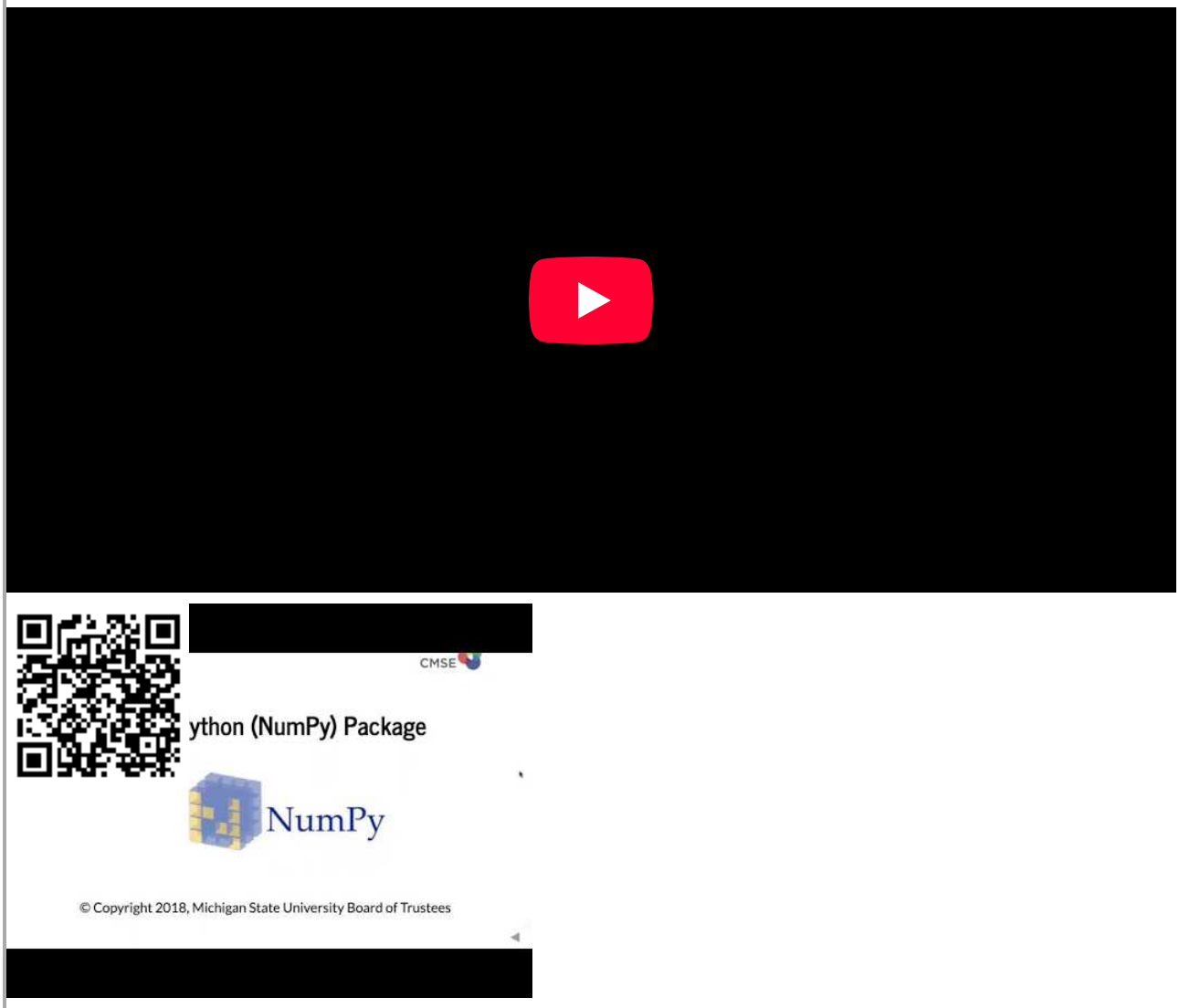
Do This

Watch the following video about the python Numpy package.

Direct Link to the Youtube video.

```
from IPython.display import YouTubeVideo
YouTubeVideo("_hbWtNgstlI",width=640,height=320, cc_load_policy=True)
```

run | restart | restart & run all



The Python Numpy library has a `matrix` object which can be initialized as follows:

```
import numpy as np
A = np.matrix([[1,1], [20,25]])
b = np.matrix([[30],[690]])
print("A="+str(A))
print("b="+str(b))
```

run | restart | restart & run all

```
A=[[ 1  1]
 [20 25]]
b=[[ 30]
 [690]]
```

Python can solve equations in the \(Ax=b\) format with the `numpy.linalg` library. For example:

```
import numpy as np

x = np.linalg.solve(A, b)
print("x="+str(x))
```

run | restart | restart & run all

```
x=[[12.]
 [18.]]
```

The `numpy.linalg` library is just a subset of the `scipy.linalg` library.

```
import scipy.linalg as la

x = la.solve(A, b)
print("X="+str(x))
```

run | restart | restart & run all

```
X=[[12.]
 [18.]]
```

Do This

Convert the following system of linear equations to numpy matrices and solve it sing a Python linear algebra solver (Store the solutions in a vector named `x`).
\(18x + 21y = 22672x - 3y = 644\)

```
##Put your answer to the above question here.
```

run | restart | restart & run all

```
from answercheck import checkanswer

checkanswer.vector(x,'756ca9fa3951fad0e623b2a8315d5fd7');
```

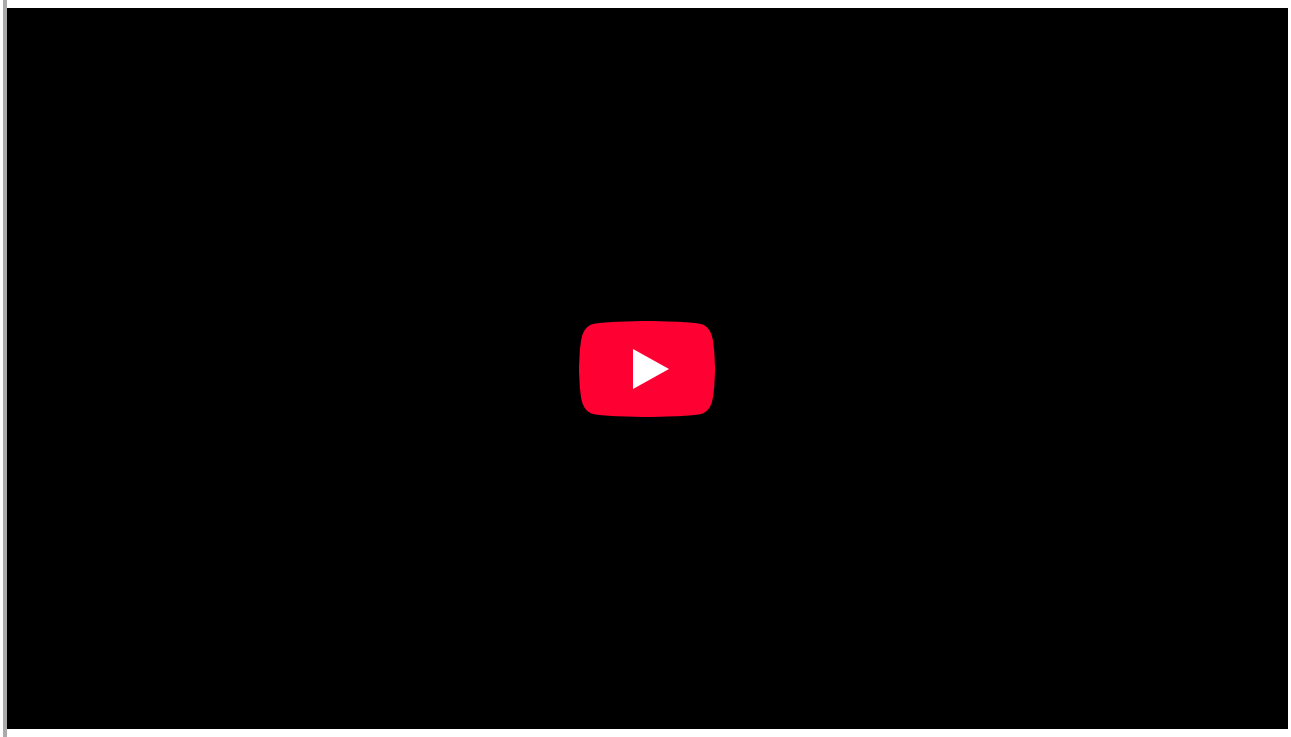
run | restart | restart & run all

44.4: Advanced Python Indexing

This one is a little long and reviews some of the information from the last video. However, I really like using images as a way to talk about array and matrix indexing in `Numpy` .
Direct Link to the Youtube video.

```
from IPython.display import YouTubeVideo
YouTubeVideo("XSyiafkKerQ",width=640,height=360, cc_load_policy=True)
```

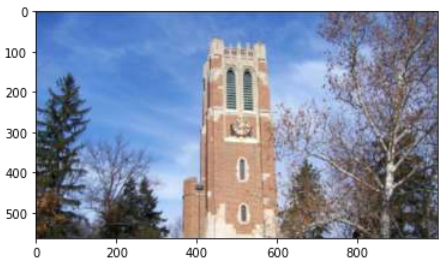
run | restart | restart & run all



```
%matplotlib inline
import matplotlib.pylab as plt
import numpy as np
import imageio

#from urllib.request import urlopen, urlretrieve
```

```
#from scipy.misc import imsave

url = 'https://res.cloudinary.com/miles-extranet-dev/image/upload/ar_16:9,c_fill,w_10(
im = imageio.imread(url)

im[10,10,0] = 255
im[10,10,1] = 255
im[10,10,2] = 255

#Show the image
plt.imshow(im);
```

run | restart | restart & run all




```
im[20,20,:] = 255
plt.imshow(im)
```

run | restart | restart & run all

```
cropped = im[0:50,0:50,:]
plt.imshow(cropped)
```

run | restart | restart & run all

```
cropped = im[50:,350:610,:]
plt.imshow(cropped)
```

run | restart | restart & run all

```
red = im[:,:,0]
plt.imshow(red)
plt.colorbar()
```

run | restart | restart & run all

```
#Note python changed slightly since the making of the video.
# We added the astype funciton to ensure that values are between 0-255
red_only = np.zeros(im.shape).astype(int)
red_only[:,:,0] = red

plt.imshow(red_only)
```

run | restart | restart & run all

```
green_only = np.zeros(im.shape).astype(int)
green_only[:,:,1] = im[:,:,1]

plt.imshow(green_only)
```

run | restart | restart & run all

```
blue_only = np.zeros(im.shape).astype(int)
blue_only[:,:,2] = im[:,:,2]

plt.imshow(blue_only)
```

run | restart | restart & run all

Do This

Modify the following code to set all of the values in the blue channel to zero using only one simple line of indexing code.

```
no_blue = im.copy()
#####Start your code here #####

#####End of your code here#####
plt.imshow(no_blue)
```

run | restart | restart & run all

Question

What was the command you use to set all of the values of blue inside `no_blue` to zero?

44.5: LaTeX Math

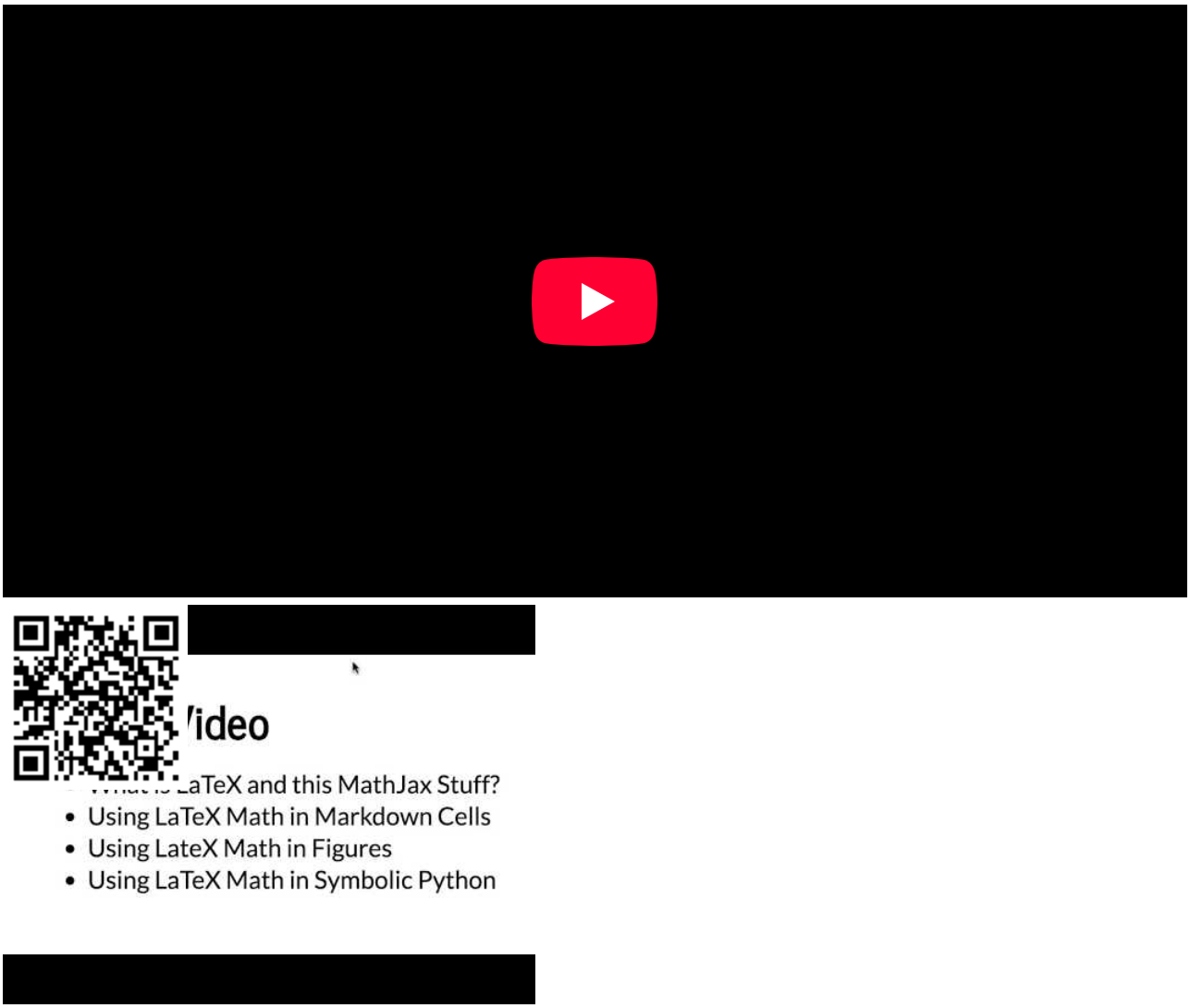
Direct Link to the Youtube video.

Login with LibreOne to run this code cell interactively.

If you have already signed in, please refresh the page.

Login

```
from IPython.display import YouTubeVideo
YouTubeVideo("qgSa7n_zQ3A",width=640,height=320, cc_load_policy=True)
```



Since this is a “Matrix Algebra” course, we need to learn how to do ‘matrices’ in LaTeX. Double click on the following cell to see the LaTeX code to build a matrix:

Basic matrix notation:

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 2 & -2 \\ 0 & 1 & 2 \end{bmatrix}$$

Augmented matrix notation:

$$\left[\begin{array}{ccc|c} 1 & 0 & 4 & -10 \\ 0 & 2 & -2 & 3 \\ 0 & 1 & 2 & 1 \end{array}\right]$$

Do This

Using LaTeX, create an augmented matrix for the following system of equations:

$$4x + 2y - 7z = 3$$

$$12x + z = 10$$

$$-3x - y + 2z = 30$$

Question

In LaTeX, what special characters is used to separate elements inside a row?

Index

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: Matrix Algebra with Computational Applications (Colbry)

Webpages: 265

Applicable Restrictions: Noncommercial

All licenses found:

- CC BY-NC 4.0: 98.9% (262 pages)
- Undeclared: 1.1% (3 pages)

By Page

- Matrix Algebra with Computational Applications (Colbry) - *CC BY-NC 4.0*
  - Front Matter - *CC BY-NC 4.0*
    - TitlePage - *CC BY-NC 4.0*
    - InfoPage - *CC BY-NC 4.0*
    - Table of Contents - *Undeclared*
    - Licensing - *Undeclared*
    - About the Lead Author - *CC BY-NC 4.0*
    - Acknowledgments - *CC BY-NC 4.0*
  - 1: Matrix Algebra class preparation checklist - *CC BY-NC 4.0*
    - 1.1: Get Jupyter Working - *CC BY-NC 4.0*
    - 1.2: Review Python Packages - *CC BY-NC 4.0*
    - 1.3: Complete Pre-class Assignment - *CC BY-NC 4.0*
  - 2: 01 In-Class Assignment - Welcome to Matrix Algebra with computational applications - *CC BY-NC 4.0*
    - 2.0: Introduction - *CC BY-NC 4.0*
    - 2.1: Class Procedures - *CC BY-NC 4.0*
    - 2.2: Introductions - *CC BY-NC 4.0*
    - 2.3: Example - *CC BY-NC 4.0*
    - 2.4: Syllabus, Schedule and other Procedures - *CC BY-NC 4.0*
    - 2.5: Download and review next pre-class assignment - *CC BY-NC 4.0*
  - 3: 02 Pre-Class Assignment - Vectors - *CC BY-NC 4.0*
    - 3.0: Introduction - *CC BY-NC 4.0*
    - 3.1: Introducing the Course Textbooks - *CC BY-NC 4.0*
    - 3.2: Today's Reading - *CC BY-NC 4.0*
    - 3.3: Scalars, Vector and Tensors - *CC BY-NC 4.0*
    - 3.4: Assignment wrap-up - *CC BY-NC 4.0*
  - 4: 02 In-Class Assignment - Vectors - *CC BY-NC 4.0*
    - 4.0: Introduction - *CC BY-NC 4.0*
    - 4.1: Example linear system - *CC BY-NC 4.0*
    - 4.2: Pre-class assignment review - *CC BY-NC 4.0*
    - 4.3: Vectors in Python - *CC BY-NC 4.0*
  - 5: 03 Pre-Class Assignment - Linear Equations - *CC BY-NC 4.0*
    - 5.0: Introduction - *CC BY-NC 4.0*
    - 5.1: System of Linear Equations - *CC BY-NC 4.0*
    - 5.2: Visualizing the problem - *CC BY-NC 4.0*
    - 5.3: Multidimentional Spaces - *CC BY-NC 4.0*
    - 5.4: Matrix Notation - *CC BY-NC 4.0*
    - 5.5: Assignment wrap-up - *CC BY-NC 4.0*
  - 6: 03 In-Class Assignment - Solving Linear Systems of equations - *CC BY-NC 4.0*
    - 6.0: Introduction - *CC BY-NC 4.0*
    - 6.1: Pre-class assignment review - *CC BY-NC 4.0*
    - 6.2: Jacobi Method for solving Linear Equations - *CC BY-NC 4.0*
    - 6.3: Numerical Error - *CC BY-NC 4.0*
  - 7: 04 Pre-Class Assignment - Python Linear Algebra Packages - *CC BY-NC 4.0*
    - 7.0: Introduction - *CC BY-NC 4.0*
    - 7.1: The Syntax for Systems of Linear Equations - *CC BY-NC 4.0*
    - 7.2: Introduction to Gauss Jordan Elimination - *CC BY-NC 4.0*
    - 7.3: Gauss Jordan Elimination and the Row Echelon Form - *CC BY-NC 4.0*
    - 7.4: Gauss Jordan Practice - *CC BY-NC 4.0*
    - 7.5: Assignment wrap up - *CC BY-NC 4.0*
  - 8: 04 In-Class Assignment - Linear Algebra and Python - *CC BY-NC 4.0*
    - 8.0: Introduction - *CC BY-NC 4.0*
    - 8.1: Pre Class Review - *CC BY-NC 4.0*
    - 8.2: Solving Systems of Linear Equations - *CC BY-NC 4.0*
    - 8.3: Underdetermined Systems - *CC BY-NC 4.0*
    - 8.4: Practice Curve Fitting Example - *CC BY-NC 4.0*
  - 9: 05 Pre-Class Assignment - Gauss-Jordan Elimination - *CC BY-NC 4.0*
    - 9.0: Introduction - *CC BY-NC 4.0*