12-770 Spring 2025: Assignment #1: Building Physics

Mario Bergés

2025-01-21

Instructions

Follow the same format for the example below when answering this assignment. In general, this will require to create new cells below the **Task** cell.

Example [0%]

Using a markdown cell, like this one, you can explain your results and further elaborate on the results of your answer.

```
# Your code will go inside code cells, like this one.
# When necessary add comments in this format explaining your reasoning.
# Make sure you run your code before submitting to make sure everything works properly.
```

Warming Up [25%]

We are going to be using Jupyter Notebooks and Python for many things in the course. Given this, we will start this assignment with some warm-up exercises that are meant to make you comfortable in this environment.

Jupyter Notebooks consist of cells. This cell is a Markdown cell. Try double-clicking this cell. You can write pretty text and even Latex is supported!

A short Latex example: $\sum_i x_i = 42$

You will probably always start the notebook loading some simple libraries and setting up some of the magic that enables them to work well for you, e.g., something like this:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

After that, you will start implementing your code. Try running the code cell below.

```
#Try running this code cell
x = 5+15+1000
print(x)
```

```
1020
```

Quick exercise [5%]:

Create a new cell under this cell. Change the cell type to 'Markdown'.
Make the cell display the State-Space Representation of a linear dynamic system in LaTeX.
Do not forget to run the cell you just created.

Another quick exercise [5%]:

Create a cell under this one. Make sure the type of the cell is 'Code'.
Compute the eigenvalues of the following matrix and print them to the console:

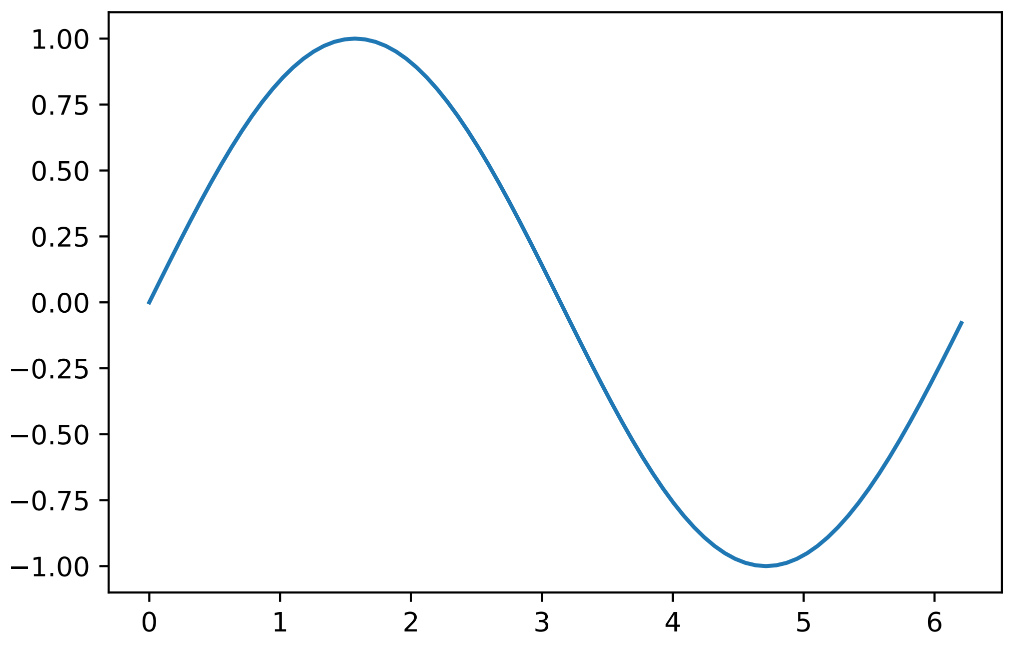
~~~
A = np.array([[1, 2], [3, 4]])

Hint: np.linalg.eigvals will be useful.

### Plotting exercise \[5%\]:

Jupyer notebooks allow you to interactively explore data. Let's plot a
sine-wave.

::: {.cell execution_count=4}
``` {.python .cell-code}
x = np.arange(0,2*np.pi,2*np.pi/80.0)
y = np.sin(x)
plt.plot(x,y)
~~~



:::

Write numpy code that plots three periods of a sine-wave in the cell below.

```
#Your code goes here
```

Data structures

The two most widely used data structures in Python are lists and dictionaries.

Lists

Here are some simple examples how to use lists.

Adding data to lists:

```
l = [] #creating an empty list
print('Empty list:')
print(l,'\n')

l.append(5) #appending 5 to the end of the list
print('List containing 5:')
print(l,'\n')
```

```
l = [1,2,3,'hello','world'] #creating a list containing 5 items
print('List with items:')
print(l,'\n')

l.extend([4,5,6]) #appending elements from another list to l
print('List with more items:')
print(l,'\n')
```

```
Empty list:
[]

List containing 5:
[5]

List with items:
[1, 2, 3, 'hello', 'world']

List with more items:
[1, 2, 3, 'hello', 'world', 4, 5, 6]
```

Accessing data in list:

```
print('Printing third element in list:')
print(l[3],'\n') #counting starts at 0

print('Printing all elements up until third element in list:')
print(l[:3],'\n')

print('Print the last 3 elements in list:')
print(l[-3:],'\n')
```

```
Printing third element in list:
hello

Printing all elements up until third element in list:
[1, 2, 3]

Print the last 3 elements in list:
[4, 5, 6]
```

Dictionaries

Dictionaries are key-value pairs. We will give some short examples on how to used dictionaries.

```
d = {} #creating empty dictionary
print('Empty dictionary:')
print(d,'\n')

d['author'] = 'Shakespeare' #adding an item to the dictionary
print('Dictionary with one element')
print(d,'\n')

#adding more items:
d['year'] = 1596
d['title'] = 'The Merchant of Venice'

#Accessing items in dictionary:
print_string = d['title'] + ' was written by ' + d['author'] + ' in the year ' + str(d['year
print(print_string)
```

```
Empty dictionary:
{}

Dictionary with one element
{'author': 'Shakespeare'}

The Merchant of Venice was written by Shakespeare in the year 1596
```

Loops

A couple of example on how to use loops.

Hint: List comprehension is the shorter syntax to loop through lists

```
list_of_numbers = [1.,2.,3.,4.,5.,4.,3.,2.,1.]

incremented_list_of_numbers = []
for i in range(len(list_of_numbers)):
    number = list_of_numbers[i]
    incremented_list_of_numbers.append(number+1)
print('Incremented list:')
```

```python
print(incremented_list_of_numbers,'\n')

#More elegantly
incremented_list_of_numbers2 = []
for number in list_of_numbers:
    incremented_list_of_numbers2.append(number+1)
print('Second incremented list:')
print(incremented_list_of_numbers2,'\n')

#We can express the for-loop above using a one-liner with list comprehensions:
#Most elegantly
incremented_list_of_numbers3 = [number + 1 for number in list_of_numbers]
print('Third incremented list:')
print(incremented_list_of_numbers3,'\n')

#looping over dictionaries
print('Print key/value pairs in dictionary:')
for key in d:
    value = d[key]
    print(key,value)
```

```
Incremented list:
[2.0, 3.0, 4.0, 5.0, 6.0, 5.0, 4.0, 3.0, 2.0]

Second incremented list:
[2.0, 3.0, 4.0, 5.0, 6.0, 5.0, 4.0, 3.0, 2.0]

Third incremented list:
[2.0, 3.0, 4.0, 5.0, 6.0, 5.0, 4.0, 3.0, 2.0]

Print key/value pairs in dictionary:
author Shakespeare
year 1596
title The Merchant of Venice
```

In the cells provided below, complete the following tasks:

Using a for loop and `len()`, compute the sum of squares of `list_of_numbers` [2%]

```
# your code goes here
```

Using list comprehension, create a list that contains the square root of each number [3%]

```
# your code goes here
```

Using list comprehension, create a list containing all values of d [2%]

```
# your code goes here
```

Using list comprehension, create a list containing all keys of d [3%]

```
# your code goes here
```

The Energy Use of Buildings (35%)

World Energy Demand

Let us now analyze the world energy demand data from EIA for the latest year. To do so, we will first import the data using another commonly used python structure *pandas*. Pandas is a data structure for designed manipulation and analysis. Learning how to use Pandas structures is essential for data science!

To start, make sure you have the `eia_data.csv` file in the same directory as your Jupyter Notebook. The code below loads the data into a pandas dataframe. This data contains the energy consumption per country in Mtoe for the last 40 years as well as the population for 2019. Familiarize yourself with the data: uncomment sections of the code below to better understand its contents (use "ctrl + /" to quickly comment/uncommment lines).

```
df = pd.read_csv('eia_data.csv',index_col=0)
# df.columns                        #View all the countries
# df['Honduras']                    #View all the data for one country
# df['Honduras']['2010']            #View the data for one country and one year
# df['Honduras']['pop_2019']        #View the data for one country and one year
# df.loc['2010']                    #View the data for one year for all countries
```

As we have seen in class, only a handful of countries account for a large portion of the total energy consumption in the world. Use the data for 2019 to answer the following questions: [15%]

- Which countries are in the top 10% of energy consumers globally?
- What proportion of the world population is contained in the list of countries you previously obtained?
- Calculate the energy consumption per capita for these countries, and comment on what you observe.

Hint: This function might be helpful `sort_values`

```
# your code goes here
```

Now let's turn our attention to a slightly different issue about world energy demand, namely our projections about the future demand. It is reasonable to expect that given growing wealth, the emergence of a middle class, access to modern conveniences, the "emerging" countries of the world may continue to experience exponential growth in energy consumption over the short to medium term. From data introduced earlier, these countries consumed $3.77x10^{13}$ kWh of energy in 1990 and $5.5x10^{13}$ kWh in 2004.

Using the top 2 highest energy consumption countries, the median energy consumption country, and the bottom 2 energy consumption countries in the world for 2019, fit a simple exponential model to predict energy consumption. For these selected countries, answer the following questions: [10%]

- Generate plots of your fitted models and historical data with projections up to 2050.
- What is the predicted energy consumption for 2025?
- What is the predicted energy consumption for 2050?
- Discuss the implications of this prediction and the validity of this claim, and compare them with the projections by official sources.

Hint: To fit an exponential model you can follow this tutorial

```
# your code goes here
```

Select any one of the countries you listed previously and answer the following questions: [10%]

- Briefly describe a few defining characteristics of the energy sector in your selected country (use IEA as your source).

- How would you describe the influence of climate in your country on building energy consumption? How will climate change affect this?
- What policies does your country have for addressing energy efficiency in the **building sector**?

Your answer goes here

Building Thermodynamics [40%]

To facilitate the calculation of the amount of heat loss per heating season, a quantity known as the number of Heating Degree Days (HDD) has been tabulated for various locations. This number is taken as a measure of the severity of the winter, but it does not consider wind or sunshine or several other weather factors. Read about it online to understand how to compute it. For a period of time characterized by a known number of HDD, the heat loss of a wall or any other surface built up of a layered series of n materials ($i = [1...n]$) with known thickness (l_i) and conductivity (k_i), the heat loss is given by:

Fill out the formula in the following cell [5%]:

Your answer goes here:

$Q = $

Now let's talk about windows. The effective R-value for any window is obtained by adding the inner and outer air layer R-values to that of the window and all its layers. Assuming that the inner and outer air layers have R-values of 0.75 and 0.2, respectively, what is the heat loss (in Wh) through a 1.5m x 3m single-pane window of 4.0mm thickness during a 140-day heating season with an average outside temperature of -5°C and an indoor temperature of 68°F?

Compute Q using a simple Python script [5%]:

```
# Your code goes here
```

Now imagine that instead of a single-pane window, we use a triple-pane window with Argon gas in-between, separated by 2cm for each layer. What would be the heat loss (Wh) in this scenario?

Recompute Q using another simple Python script [5%]:

```
# Your code goes here
```

If the efficiency of a gas furnace is 90% and the natural gas price is \$0.40/m^3, how much would the yearly savings be if the owner of the house changed the windows to the triple-pane option described in the previous task? Do we have sufficient information to estimate the cost savings? What assumptions are needed?

Answer the above question using a combination of Python code and prose [5%]

```
# Your answer goes here
```

Now let's switch gears and work on a more complete case of heat loss calculations. In particular, let's work towards finishing the spreadsheet that we will work on last during class. Instead of using a spreadsheet, though, I thought it would be a better learning opportunity to rely on Python Objects (and Classes) to model the problem.

I am providing a starting point for this idea. Below you will find sample code that creates five classes from which objects can be instantiated: Window, Wall, Furnace, House and Environment. These contain pretty much all the information there is in the spreadsheet, aside from costs and green house gas emission factors. I have also taken the liberty to modify some calculations (e.g., using 100W of heat gain per occupant) and make some corrections (e.g., the calculated area for the South facing wall was missing).

We will start by defining the classes, with the relevant attributes and methods. I will do most of the heavy lifting for you, but there are many methods (and possibly attributes) that will still need to be extended later.

The first class will be Window, which is simple and only contains information about the area and thermal resistance of the assembly.

```
class Window:
  """A window assembly"""

  def __init__(self, area, rsi):
    self.area = area
    self.rsi = rsi
```

Then we will create a similar class for Wall, but in this case we will have to segment information about the part of the wall that is below and above grade, as well as information about the windows it contains.

```
class Wall:
  """A general wall assembly, including windows"""

  def __init__(self, name, height, length, aboveGradeRSI, belowGradeRSI, belowGradeDepth, so
    self.name = name
    self.height = height
    self.length = length
    self.window = []
    self.belowGradeDepth = belowGradeDepth
    self.belowGradeRSI = belowGradeRSI
    self.aboveGradeRSI = aboveGradeRSI
    self.solarGain = solarGain

  def aboveGradeArea(self):
    return self.length * self.height - sum([w.area for w in self.window])

  def belowGradeArea(self):
    return self.length * self.belowGradeDepth

  def addWindow(self, window):
    self.window.append(window)

  def calculateHeatLossRate(self, deltaT):
    """Calculates heat loss rate through assembly, assumes SI units and outputs Watts"""
    # Fix this, if needed
    return (self.aboveGradeArea() / self.aboveGradeRSI +
      self.belowGradeArea() / self.belowGradeRSI +
      sum([w.area / w.rsi for w in self.window])) * deltaT
```

Not truly necessary but I thought it would be fun to have the Furnace have its own class:

```
class Furnace:
  """A generic furnace for our house"""

  def __init__(self, efficiency):
    self.efficiency = efficiency
    #self.fuelEmissionFactor = fuelEmissionFactor
```

And now we can put all of these together into a House, and add additional attributes about the house including internal heat gains:

```
class House:
  """A generic house composed of walls, windows and other properties"""

  def __init__(self, ceilingArea, volume, uncontrolledAirExchanges, controlledAirExchanges, m
    self.ceilingArea = ceilingArea
    self.volume = volume
    self.uncontrolledAirExchanges = uncontrolledAirExchanges
    self.controlledAirExchanges = controlledAirExchanges
    self.maxAirExchanges = maxAirExchanges
    self.heatExchangerEfficiency = heatExchangerEfficiency
    self.soilRSI = soilRSI
    self.ceilingRSI = ceilingRSI
    self.occupants = occupants
    self.averageOccupancy = averageOccupancy
    self.dailyElectricityUse = dailyElectricityUse
    self.indoorTemperature = indoorTemperature
    self.furnace = [] # List of furnaces
    self.walls = {} # Dictionary of walls

  def addFurnace(self, furnace):
    self.furnace.append(furnace)

  def calculateTotalHeatTransfer(self, deltaT, duration):
      # Fix the code below so that you compute the total heat transfer (in Joules)
      # for the house (ceiling, floor, walls, windows) over the given duration (in seconds).
      # The result can later be converted to kWh if needed.
      return 0

  def calculateDailyInternalHeatGains(self):
    return self.dailyElectricityUse + 0.1 * 24 * self.averageOccupancy * self.occupants

  def addWalls(self, wall):
    self.walls[wall.name] = wall
```

Last, but not least, let's create a class for objects that contain environmental information (i.e., the outdoor environment in which the house will be). We can also use it to store some basic physical properties of materials (e.g., air density).

```
class Environment:
  """A generic object to capture environmental conditions to which the house will be subjecte

  def __init__(self, avgOutsideTemp = 3.2, minOutdoorTemp = -25, heatingDays = 270):
    self.avgOutsideTemp = avgOutsideTemp
    self.minOutdoorTemp = minOutdoorTemp
    self.heatingDays = heatingDays
    specificHeatAir = 1000
    airDensity = 1.25
```

Now let's move on to assemble an instance of a House object. We start by creating instances of each of the Walls:

```
north = Wall(
    name = 'North',
    length = 20,
    height = 6.25,
    aboveGradeRSI = 1.4,
    belowGradeRSI = 1/0.67,
    belowGradeDepth = 2,
    solarGain = 25)

east = Wall(
    name = 'East',
    length = 5.5,
    height = 5,
    aboveGradeRSI = 1.4,
    belowGradeRSI = 1/0.67,
    belowGradeDepth = 2,
    solarGain = 50)

south = Wall(
    name = 'South',
    length = 20,
    height = 6.25,
    aboveGradeRSI = 1.4,
    belowGradeRSI = 1/0.67,
    belowGradeDepth = 2,
    solarGain = 100)

west = Wall(
    name = 'West',
    length = 5.5,
    height = 7.5,
    aboveGradeRSI = 1.4,
    belowGradeRSI = 1/0.67,
    belowGradeDepth = 2,
    solarGain = 50)
```

And then we add window objects to the walls:

```
north.addWindow(Window(7,1/2.8))
east.addWindow(Window(6,1/2.8))
#south wall does not have windows
west.addWindow(Window(7.5,1/2.8))
```

Then we can put them all together to create an instance of a House object:

```
house = House(
    ceilingArea = 110,
    volume = 907.5,
    uncontrolledAirExchanges = 1.1,
    controlledAirExchanges = 0,
    maxAirExchanges = 3.3,
    heatExchangerEfficiency = 0.75,
    ceilingRSI = 3.5,
    soilRSI = 1/0.13,
    occupants = 4.5,
    averageOccupancy = 0.5,
    dailyElectricityUse = 15,
    indoorTemperature = 21)

house.addFurnace(Furnace(efficiency = 0.71))
house.addWalls(north)
house.addWalls(east)
house.addWalls(south)
house.addWalls(west)
```

Lastly, we create an environment keeping in mind that if we pass no arguments it will assume default values for all of the attributes (which we defined to be the ones in the Harvey problem):

```
environment = Environment()
```

Now we are ready to work. To test things out, you may want to calculate daily internal heat gains and verify that it is close to what Harvey has on the problem (i.e., close to 18.8 kWh/day):

```
print(f"The internal heat gains for a house with {house.occupants} occupants and daily electr
```

```
The internal heat gains for a house with 4.5 occupants and daily electricity use of 15 kWh is
```

That seems right. Now let's move on to calculate the heat loss rate through one of the Walls to check that it makes sense:

```
print(f"The North wall of the house has an above-grade area of {house.walls['North'].aboveGra
```

```
The North wall of the house has an above-grade area of 118.0 m^2 and an area of 40 m^2 with
```

Given that we now know how to calculate heat loss rates through walls and windows, let's compute the total heat loss over a 140-day heating season for the entire thermal envelope (walls, ceiling, and floor). Update the calculateTotalHeatTransfer method in the House class so that it computes the total heat loss for the given duration in Joules or kWh. Ensure the method accounts for the entire thermal envelope and provides the correct result.

```
house.calculateTotalHeatTransfer(abs(environment.avgOutsideTemp - house.indoorTemperature),
```

Fix the class definitions and methods so that the code snippet above outputs the correct answer [10%]

Given that we now know how to calculate the total heat loss through walls, ceiling, and floor, update the code to compute the heat loss rate across the heating season. Using the updated calculateTotalHeatTransfer method, - calculate the average heat loss rate (in Watts). - calculate the Peak Heating Requirement based on conductive losses

Write a simple Python script to Compute the Total Conductive Heat Loss over the heating season and the Peak Heating Requirement based on conductive losses [5%]:

```
# Your code goes here
```

And that's it! Well, is it? Are there things we discussed in class about thermodynamics that this simple model is not considering? If so, please modify the classes according to what you think is missing and re-calculate the total heat loss and peak heating requirements.

Extend the classes so that they account for other thermodynamic properties and/or phenomena that were covered in class, and recalculate your answer to the previous question [5%]:

```
# Your code goes here
```