12-770 Spring 2026: Assignment #1: Building Physics

Mario Bergés

2026-01-22

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.
```

Setting Up Your Development Environment [25%]

Before we can do any data analysis or building physics calculations, we need to set up a proper development environment. In this section, you'll create a Python project using `uv` (a modern Python package manager), install the packages we need, verify everything works, and learn to track your work with version control using `git`.

By the end of this section, you will have experience with the workflow that you'll use for all future assignments in this course.

Running Shell Commands from Jupyter [2%]

Jupyter notebooks allow you to run shell commands directly from code cells by prefixing them with `!`. This is incredibly useful for managing your development environment without leaving the notebook.

Try running these commands to see where you are and what files exist. Note that we are assuming you have access to a Linux-based shell under the hood. If you are on Windows, you might need to modify the commands or find a way to access a Linux shell. I'd be curious if you find the answer for how to do this, so please share!

```
# See your current working directory
!pwd
```

```
# List files in the current directory
!ls -la
```

Task: In the cell below, use a shell command to display the current date and time. (Hint: the command is `date`)

```
# Your code goes here
```

Creating a Python Project with uv [5%]

`uv` is a modern, fast Python package manager that helps you create reproducible project environments. Instead of installing packages globally (which can lead to version conflicts), we'll create an isolated project for this assignment.

First, let's create a directory for this assignment and initialize a uv project:

```
# Create a directory for your assignment (adjust the path as needed for your system)
!mkdir -p ~/tmp/courses/12-770/assignment1
```

```
# Initialize a uv project in that directory
%cd ~/tmp/courses/12-770/assignment1
!uv init
```

Bonus: In the above code *why did we use `%` as opposed to `!` to change the directory?*

Now let's explore what uv created:

```
# List the contents of your new project
!ls -la ~/tmp/courses/12-770/assignment1
```

```
# View the pyproject.toml file - this is where your project configuration lives
!cat ~/tmp/courses/12-770/assignment1/pyproject.toml
```

The `pyproject.toml` file is the modern standard for Python project configuration. It tracks your project's dependencies, Python version requirements, and metadata. This file is what makes your environment *reproducible* — anyone with this file can recreate the exact same environment.

Task: In a markdown cell below this one, briefly explain (in your own words) why having a `pyproject.toml` file is better than just installing packages globally with `pip install`.

Adding Dependencies [3%]

Now let's add the packages we'll need for this course. With uv, adding dependencies is straightforward:

```
# Add numpy, matplotlib, and pandas to your project
%cd ~/tmp/courses/12-770/assignment1
!uv add numpy matplotlib pandas
```

```
# Verify the packages are now listed in pyproject.toml
!cat ~/tmp/courses/12-770/assignment1/pyproject.toml
```

Notice how uv automatically resolved the versions and added them to your project file. It also created a `uv.lock` file that pins exact versions for reproducibility.

Task: Add the `scipy` package to your project using the cell below, then verify it appears in `pyproject.toml`:

```
# Your code goes here
```

Setting Up the Jupyter Kernel [5%]

Now we need to make this uv environment available as a Jupyter kernel. This allows you to run notebooks using the exact packages in your project.

First, let's add the `ipykernel` package (needed to register Jupyter kernels):

```
%cd ~/tmp/courses/12-770/assignment1
!uv add ipykernel
```

Now register the kernel:

```
%cd ~/tmp/courses/12-770/assignment1
!uv run python -m ipykernel install --user --name=assignment1 --display-name="12-770 Assignme
```

You should see a message confirming the kernel was installed.

Important Next Steps:

1. **Save this notebook** and note its current location
2. **Copy or move this notebook** to your project folder (`~/tmp/courses/12-770/assignment1/`)
3. **Close this Jupyter session**
4. **Open a terminal**, navigate to your project folder, and launch JupyterLab from within the project:

```
cd ~/tmp/courses/12-770/assignment1
uv run --with jupyter jupyter lab
```

5. **Open this notebook** from JupyterLab and select the "12-770 Assignment 1" kernel (Kernel → Change Kernel)

From this point forward, you should be working in the notebook from within your uv project. This is the workflow you'll use for all assignments.

Task: After relaunching from your project folder, run the cell below to verify you're using the correct environment. The path should point to your project's virtual environment:

```
import sys
print(f"Python executable: {sys.executable}")
print(f"Python version: {sys.version}")
```

Testing Your Environment [5%]

Now let's verify that your packages are working correctly. We'll import them and do some quick tests.

```python
# Import the packages we added
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy
```

```python
# Quick numpy test: compute eigenvalues of a matrix
A = np.array([[1, 2], [3, 4]])
eigenvalues = np.linalg.eigvals(A)
print(f"Eigenvalues of A: {eigenvalues}")
```

```python
# Quick matplotlib test: plot a sine wave
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
plt.figure(figsize=(8, 4))
plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.show()
```

```python
# Quick pandas test: create a simple DataFrame
data = {
    'Room': ['Living Room', 'Bedroom', 'Kitchen', 'Bathroom'],
    'Area_m2': [25.0, 15.0, 12.0, 6.0],
    'Temperature_C': [21.5, 20.0, 22.0, 23.0]
}
df = pd.DataFrame(data)
print(df)
```

Task: In the cell below, use numpy to compute and print the eigenvalues of the following 3x3 matrix:

$$B = \begin{bmatrix} 4 & -2 & 1 \\ 1 & 1 & 1 \\ 2 & -1 & 3 \end{bmatrix}$$

```python
# Your code goes here
```

Task: In the cell below, create a plot showing **three complete periods** of a sine wave:

```
# Your code goes here
```

Markdown and LaTeX in Jupyter [2%]

Jupyter notebooks support rich text formatting using Markdown, including mathematical equations using LaTeX syntax. This is essential for documenting your analysis and presenting results clearly.

Here's an example of inline math: The thermal resistance is $R = \frac{l}{kA}$ where l is thickness, k is conductivity, and A is area.

Here's an example of a displayed equation — the heat transfer through a wall:

$$Q = \frac{A \cdot \Delta T}{R} \cdot t$$

where Q is total heat transferred (Joules), A is area (m^2), ΔT is temperature difference (K), R is thermal resistance ($m^2 \cdot K/W$), and t is time (seconds).

Task: Create a new Markdown cell below this one and write the general **heat conduction equation** (Fourier's Law) in LaTeX. Include a brief explanation of what each variable represents. (Hint: Look up Fourier's Law if you're not familiar with it.)

Python Fundamentals Review [3%]

Let's do a quick review of Python fundamentals that you'll use throughout the course.

Lists and List Comprehensions

Lists are ordered collections that can hold any type of data:

```
# Creating and manipulating lists
temperatures = [18.5, 19.2, 21.0, 22.5, 20.1, 19.8, 21.3]

# Accessing elements (0-indexed)
print(f"First temperature: {temperatures[0]}°C")
print(f"Last temperature: {temperatures[-1]}°C")
print(f"First three: {temperatures[:3]}")
```

List comprehensions provide a concise way to create lists:

```python
# Convert Celsius to Fahrenheit using list comprehension
fahrenheit = [t * 9/5 + 32 for t in temperatures]
print(f"Temperatures in Fahrenheit: {fahrenheit}")
```

Dictionaries

Dictionaries store key-value pairs, perfect for structured data:

```python
# A dictionary representing a building zone
zone = {
    'name': 'Office 101',
    'area_m2': 45.0,
    'volume_m3': 135.0,
    'setpoint_C': 21.0,
    'occupancy': 4
}

print(f"Zone: {zone['name']}")
print(f"Heat gain per occupant at 100W: {zone['occupancy'] * 100}W")
```

Loops

```python
# Iterating over a dictionary
print("Zone properties:")
for key, value in zone.items():
    print(f"  {key}: {value}")
```

Task: Given the list of R-values (thermal resistances) for different wall layers below, use a **for loop** to compute the total R-value (sum of all layers) and print it:

```python
r_values = [0.12, 2.5, 0.18, 0.04]  # R-values in m²·K/W
# Your code goes here
```

Task: Using **list comprehension**, create a new list containing the U-value (thermal transmittance) of each layer. The U-value is the reciprocal of R-value: $U = 1/R$

```python
# Your code goes here
```

Version Control with Git [5%]

Now that you have a working project, let's set up version control using `git`. Version control allows you to track changes to your code, collaborate with others, and revert to previous versions if needed.

Initializing a Repository

```
# Initialize a git repository in your project folder
%cd ~/tmp/courses/12-770/assignment1
!git init
```

Configuring Git

Git needs to know who you are for commit messages. Run these commands with your own name and email:

```
# Configure your identity (replace with your actual name and email)
!git config --global user.name "Your Name"
!git config --global user.email "your.email@example.com"
```

Checking Status

The `git status` command shows you what files have been modified, added, or are untracked:

```
%cd ~/tmp/courses/12-770/assignment1
!git status
```

Staging and Committing

To save a snapshot of your work, you first "stage" files with `git add`, then "commit" them with a message:

```
# Stage all files
%cd ~/tmp/courses/12-770/assignment1
!git add .
```

```
# Create your first commit
%cd ~/tmp/courses/12-770/assignment1
!git commit -m "Initial project setup with dependencies"
```

Viewing History

```
# View the commit history
%cd ~/tmp/courses/12-770/assignment1
!git log --oneline
```

Task: Make a small change to demonstrate the git workflow:

1. Run the cell below to create a simple README file
2. Check the git status
3. Stage and commit the new file with an appropriate message
4. View the updated commit history (you should see 2 commits)

```
# Create a README file
!echo "# Assignment 1 - Building Physics\n\nThis project contains my solutions for 12-770 Ass
```

```
# Check status - you should see README.md as untracked
# Your code goes here
```

```
# Stage the README file
# Your code goes here
```

```
# Commit with a descriptive message
# Your code goes here
```

```
# View the commit history - should show 2 commits now
# Your code goes here
```

Congratulations! You now have a properly configured development environment with version control. This workflow is what you may want to use for all future assignments in this course.

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/m³, 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
```

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
```

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), c
```

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
```