CEE 251L. Uncertainty, Design and Optimization

Civil and Environmental Engineering
Duke University
Homework 1, due: Wednesday, January 14, 2026

This assignment asks you to run `Python` commands and to write short `Python` functions. You may prepare your `Python` code as `.py`-files using VScode or as a Jupyter notebook. It is convenient to save your `.py`-files to a GitHub repository, since GitHub saves previous versions.

Always explain your code with detailed comments. Include your name as a comment within each of your `.py`-files.

1. (5 points) engineering ethics

   Engineering Ethics Question 1

2. (5 points) Martin Luther King day reflection

   On Martin Luther King day, please enjoy time reading something by Martin Luther King, listening to one of his speeches, or participating in a Martin Luther King day event. Write a pargraph about your thoughts on your experience.

3. (4 points) `Python`

   (a) Read Using the Terminal.

   (b) Read `Python` language skills and debugging.

   (c) Install `git`, VS Code and `Python` from the instructions in section 2.

4. (6 points) multivarious
   Install multivarious using `git` and `pip` (for example, open a terminal, navigate to a folder like Desktop/UDO/Code) and clone the repository:

```
cd ~/Desktop/UDO/Code
git clone https://github.com/hpgavin/multivarious
pip install -e multivarious
```

   Verify that VS Code has access to multivarious:

   (a) VS Code > File > Open Folder ... > navigate to your `multivarious/examples` folder > Open

   (b) VS Code > File Explorer > select `verify_path_import.py` > click the right arrow in the Edit window [Run Python File]

   (c) This will open a new Terminal Window within VS Code which should display ...

```
hello
verifying that PYTHONPATH has been set ...
PYTHONPATH env: /home/USERNAME/Desktop/UDO/Code/multivarious
... and yes, yes it has. Great!
verifying that multivarious can be imported ...
... and yes, yes it can. Great!
```

   (d) Click on  in the VS Code terminal panel or CTRL-D at the >>> `Python` prompt to exit the Python Interactive mode and return to the VS Code terminal.

5. (10 points) make computing mistakes in `Python` and fix them.
   Type in each command, one at a time, and after each command check the value of the result, i.e., after typing the first line below, type `a1` to see the value of the variable `a1`

   Here is a tutorial on Python, NumPy, and MatPlotLib

```
# import package(s) and create aliases
import numpy as np                    # np instead of numpy
from np import array                  # array instead of numpy.array
from numpy.linalg import inv as inv # inv instead of numpy.linalg.inv

# create arrays and do math on them ----------------------------------------
a1 = [ 5 , 8 , 13 ]                 # ok.  but this is a "list" not a mathematical array
a = array( [ 5 , 8 , 13 ] )         # ok?  this neither a row nor a column
b1 = a.T                            # !!.  .T transpose - does not work with a 1D array
b  = np.array([a]).T                # ok.  .T transpose - works with a 2D array
c1 = a1 * 2                         # surprised?  intrigued??  correct??
c  = a  * 2                         # ok.  mathematically correct.
x  = b^3                            # ??   bitwise XOR
d  = b**3                           # ok.  use ** for exponentiation
p1 = a * b                          # ok.  element-wise multiplication
p2 = b * a                          # ok.  element-wise multiplication  a*b = b*a
p3 = a * b.T                        # ok.  element-wise multiplication (1x3) ... like .*
p4 = a.T * b                        # ok.  element-wise multiplication (3x1) ... like .*
q1 = a @ b                          # ok?  correct inner product (?x3)(3x1) not(1x1) ?
q2 = np.array([a]) @ b              # ok   correct inner product (1x3)(3x1)  = (1x1)
q3 = b @ a                          # ??    not an outer product (3x1)(?x3) not (3x3)
q4 = b @ [a]                        # ok.  correct outer product (3x1)(1x3)  =  (3x3)
q5 = [a].T @ b.T                    # ??   can not transpose a list
q4 = np.array([a]).T @ b.T          # ok.  correct transpose of outer product
r1 = a / b                          # ok.  element-wise  operation, (1x3) like "./"
r2 = a / b.T                        # ok.  element-wise  operation, (3x1) like "./"
r3 = a \ b                          # !!.  there's no "left divide" in Python
# multi-dimensional arrays (matrices)  ------------------------------------
A1 = [ a , a+21 , a**2 ]            # !!.  list of 2D arrays
A2 = array( a , a+21 , a**2 )       # !!.  syntax error
A  = np.array([ a, a+21, a**2 ]) # ok.  preferable!
x  = inv(A) @ b                     # ok.  solves A*x=b for x
z  = A @ x - b                      # ok.  ... should be very close to zero
AtA = A.T @ A                       # a symmetric non-negative definite matrix
# stacking arrays horizontally and vertically ------------------------------
B  = np.array([ b , b**2 ])         # ok. --- but it will not work with np.block ...
B  = np.hstack([ b , b**2 ])        # ok. --- this will work with np.block
C  = np.array([ c+13 , c**2 ])      # ok. ... even though the first try at B woudn't work
D  = np.zeros([2,2])                # ok.
S  = np.block([[A, B], ([C, D]) ])  #  [ A, B ; C, D ]
# indexing arrays ----------------------------------------------------------
u  = [ 0 : 10 ]                     # !!. syntax error
u  = np.linspace(1,10,10)           # ok! much better ...
v  = u[10]                          # !!. Python array indices start at zero
v  = u[0]                           # ok. Python array indices start at zero
v  = u[-1]                          # ok. ... the last element of u
v  = u[-2]                          # ok. ... the next to last element of u
v  = u[2:5]                         # ??. starting at index 2 and going to the 5th value
v  = u[2:5+1]                       # ok. starting at index 2 and going to index 5
v  = A[2,0]                         # ok. ... row 2, column 0 of A
v  = A[2][0]                        # ok. C-syntax
v  = A[1,:]                         # ok. row 1 of A ... (the second row of A "index-0")
v  = A[1:2,:]                       # ok. rows 2 and 3 and all columns
v  = A[1:2,1:2]                     # ok. rows 2 and 3 and columns 2 and 3
v  = u[6:-2:2]                      #!?!. not at all what matlab users would expect!
v  = u[[6,4,2]]                     # ok. ... what we would expect, in any case.
```

   Think of two more vector mistakes and two more matrix mistakes. Call them `y` , `z` , `E` , and `F`. And fix those mistakes

6. (10 points) the golden ratio

   pS&D section 7.11 help on Python functions and matplotlib examples

   In the Fibonacci sequence, F_j, each number is the sum of the previous two numbers. The first two values of a Fibonacci sequence are [1, 1]. The ratio of consecutive Fibonacci numbers, F_j/F_{j+1}, converges to the golden ratio ϕ as j gets large. The golden ratio solves the quadratic $1/\phi = 1 + \phi$.

   In this problem you are asked to write a `.py`-file called `fibo.py` (a.k.a. a Python *module*) that contains two function called `seqnce` and `plot`. Each of these two functions takes an argument `N` for the length of the Fibonacci sequence. A sketch of the code is ...

```python
# add comments indicating the purpose of this code and your name
import numpy as np
import matplotlib.pyplot as plt

def seqnce(N):
"""
returns a Fibonacci sequence of any length N, N greater than 1.
"""

# ... write code to
# (a) compute the Fibonacci sequence F_0 ... F_N
#     ... without typing in any Fibonacci number except for ... F_0 = 1 and F_1 = 1
# (b) compute the array of indices j = [ 0 ... N ] using np.linspace
# ... about five lines of code
    return j, F

def plot(N):

# ... write code to
#    (a) obtain the Fibonacci sequence F and the array j from the function seqnce(N),
#    (b) calculates the squence of golden ratio values from F
#    (c) plots the Fibonacci sequenc and plots the golden ratio sequence
#        where the plot axes are labled
#    ... about two lines of code for (a) and (b) plus another dozen for (c)
```

   A rational number a is the ratio of integers n and m: $a = n/m$. Not all numbers are rational, eg., π, $\sqrt{2}$, e, etc. How can one irrational number be "more irrational" than another irrational number? What is the *most irrational* number? Here are notes showing how the golden ratio is the *most irrational* number and a proof from stackexchange.

7. (10 points) sums of sinusoids without a `for`-loop or `sum` ... pS&D section 14.3

   A sum of sinusoids is called a *Fourier series*. For example,

$$y(x;n) = \frac{4}{\pi} \sum_{k=1}^{n} \frac{1}{2k-1} \sin\left((2k-1)x\right)$$

   Using a single `for`-loop calculate three vectors of y for $n = 5$, 10, and 15. Within the `for`-loop: create a row vector `k = [ 1 : N ];` and then calculate `y` in one line of `Python` by being clever about using transpose (`.T`) and vector multiplication (`*` and `@`). Why don't you need a `for`-loop or a `sum` for this calculation? On the same set of axes, make plots of $y(x;5)$, $y(x;10)$ and $y(x;15)$ with 500 values of x in the range $-\pi \le x \le \pi$. Label your axes.

8. (5 points) 3-D plots and contour plots ... pS&D section 17

   Make a surface plot and a contour plot of the saddle shape

$$f(x,y) = 2x^2 - 3xy - 4y^2$$

   for $-10 \le x \le 10$ and $-10 \le y \le 10$.

9. (5 points) make a beam slope as much as possible ... pS&D section 9

   From the beam deflection tables here: https://mechanicalc.com/reference/beam-deflection-tables, find the equation for the slope at the end of a simply supported beam, with a point load located a distance a from the left end.

   Calculate a dimensionless end-rotation, $(EI\theta_2/(FL^2))$ for 100 values of (a/L) in the range $0 \le (a/L) \le 1$, in one line, without using a for loop by being clever about using element-wise multiplication (`*`). Use the `argmax` command to find the value of the loading point (a/L) that produces the maximum rotation θ_2. Plot $(EI\theta_2/(FL^2))$ vs. (a/L) and plot a `'ro'` (red circle) on the figure at the plot coordinate $(a/L, \theta_2)$ that maximizes the end rotation θ_2.

10. (25 points) be smart with a guess-and-check approach to minimize a function

    In most optimization problems, there is no closed-form mathematical expression relating design variables $(v_1, v_2, ..., v_n)$ to the objective fuction $f(v_1, v_2, ..., v_n)$.

    Most methods of numerical optimization work something like this ...

    Given a way to calculate the design objective function $f(v_1, v_2, ..., v_n)$, in terms of n design variables $(v_1, v_2, ..., v_n)$ and an initial numerical guess for the design variables ...

    (a) Compute the value of the function $f(\boldsymbol{v}^{(0)})$ for the initial numerical guess of $\boldsymbol{v}^{(0)}$.
    (b) Somehow, find a new set of n values for $(v_1, v_2, ..., v_n)$ which reduces the value of $f(\boldsymbol{v})$, "at least a little bit."
    (c) Repeat (b) until changes in $f(\boldsymbol{v})$ are "small."

    For the objective function

    $$f(v_1, v_2) = 2 + v_1/40 + v_2/30 + \cos(v_1 v_2/20)$$

    Within a `Python` file called `udo_hw01_pr10.py` write a `.py`-function called `fv(v)` that is a function of the list `v` and computes the resulting value of the objective function $f([v_1, v_2])$. The first line of this `.py`-file is:
    `def fv(v):`
    The last line of the `.py`-file displays the result:
    `print(f' f([ v1:7.4f , v2:7.4f ]) = f:7.4f ')`
    (The six `f` characters above mean four different things!)
    (This function needs numpy for the cosine function and only a few more lines of code.)

    Within the `Python` Terminal window (`>>>`), import the Python function `fv`
    `>>> from udo_hw01_pr10 import fv`
    and compute a value for the objective function for $\boldsymbol{v}$ within the domain $-10 \le v_1 \le 10$ and $-10 \le v_2 \le 10$, for example
    `>>> fv([ 3.0 , -4.0 ])`
    Now change the values in $\boldsymbol{v}$ a little and re-run the line above.

    Did f increase or decrease? Use this to deduce your third guess and run `fv(...)` again. Repeat a few more times, and consider the logic you used to update $\boldsymbol{v}$ each time, with the goal of reducing $f(\boldsymbol{v})$. Given three consecutive guesses $\boldsymbol{v}^{(k)}$, $\boldsymbol{v}^{(k-1)}$ and $\boldsymbol{v}^{(k-2)}$ and the associated values $f^{(k)}$, $f^{(k-1)}$ and $f^{(k-2)}$. Write some logic that will give you a value $\boldsymbol{v}^{(k+1)}$ that is likely to make $f^{(k+1)}$ more negative than $f^{(k)}$.

    Now, apply that logic to this problem from some initial guess and iterate a few times.

    How does it work out?

    (It's o.k. if it doesn't.)

    Either way, describe why you think it works, or why it doesn't.