HW_0

September 23, 2025

Collaborators: Saif Ahmad

Link to Collab: https://colab.research.google.com/drive/1GjvcJQnzACsuACrtnT46IqUC1gmXIZuJ?usp=sharing

1 Numpy, Sympy, and Matplotlib Tutorial for Fall 2025 ME 314

[1]:
```
#IMPORT ALL NECESSARY PACKAGES AT THE TOP OF THE CODE
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Markdown, display
from IPython.display import IFrame
```

[2]:
```
#␣
 ↪###########################################################################################
# # If you're using Google Colab, uncomment this section by selecting the whole␣
 ↪section and press
# # ctrl+'/' (Linux/Windows) or cmd+'/' (MacOS) on your and keyboard. Run it␣
 ↪before you start
# # programming, this will enable the nice LaTeX "display()" function for you.␣
 ↪If you're using
# # the local Jupyter environment, leave it alone
#␣
 ↪###########################################################################################


try:
    # Only works in Google Colab
    from google.colab.output._publish import javascript

    def custom_latex_printer(exp, **options):
        url = "https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.1.1/latest.js?
 ↪config=TeX-AMS_HTML"
        javascript(url=url)
        return sym.printing.latex(exp, **options)
    sym.init_printing(use_latex="mathjax", latex_printer=custom_latex_printer)
```

```
except ModuleNotFoundError:
    # Fallback for Jupyter or other environments
    sym.init_printing(use_latex="mathjax")
```

1.1 Problem 1 (20pts)

Given a function $f(x) = \sin(x)$, find the derivative of $f(x)$ and find the directional derivative of $f(x)$ in the direction v. Moreover, compute these derivatives using Pythons's SymPy package.

[3]:
```
####################################
# Part 1: compute derivative of f

# define your symbolic variable here
x = sym.symbols('x')

# define the function f
f = sym.sin(x)

# compute derivative of f
df = f.diff(x)

# output resutls
print("derivative of f: ")
display(df)

####################################
# Part 2: compute directional derivative of f

# define dummy variable epsilon, and the direction v
# note 1: here the character 'r' means raw string
# note 2: here I define the symbol for epsilon with
#          the name "\epsilon", this is for LaTeX printing
#          later. In your case, you can give it any other
#          name you want.
eps, v = sym.symbols(r'\epsilon, v')

# add epsilon into function f
new_f = sym.sin(x + v*eps)

# take derivative of the new function w.r.t. epsilon
df_eps = new_f.diff(eps)

# output this derivative
print("derivative of f wrt eps: ")
display(df_eps)

# now, as you've seen the class, we need evaluate for eps=0 to ...
# ... get the directional derivative. To do this, we need to ...
# ... use SymPy's built-in substitution method "subs()" to ...
# ... replace the epsilon symbol with 0
new_df = df_eps.subs(eps, 0)

# output directional derivative
print("directional derivative of f on v: ")
display(new_df)
```

```
derivative of f:
```

$\cos(x)$

```
derivative of f wrt eps:
```

$v\cos(\epsilon v + x)$

```
directional derivative of f on v:
```

$v\cos(x)$

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution for both derivatives (or you can use LaTeX, instead of hand writing). Also, turn in the code used to compute the symbolic solutions for both derivatives and the code output.

```
[4]: # Specify the path to your PDF file and desired dimensions
pdf_path = "Hw0_Q1.pdf"  # Replace with the actual path to your PDF
width = 800  # Adjust as needed
height = 600  # Adjust as needed

# Create and display the IFrame
IFrame(pdf_path, width=width, height=height)
```

```
[4]: <IPython.lib.display.IFrame at 0x7626ac1c3740>
```

1.2 Problem 2 (20pts)

Given a function of trajectory:

$$J(x(t)) = \int_0^{\pi/2} \frac{1}{2} x(t)^2 dt$$

Compute the analytical solution when $x = \cos(t)$, verify your answer by numerical integration.

The code for numerical integration is provided below:

```
[5]: def integrate(func, xspan, step_size):
    '''
    Numerical integration with Euler's method

    Parameters:
    ====================
    func: Python function
        func is the function you want to integrate for
    xspan: list
        xspan is a list of two elements, representing
        the start and end of integration
    step_size:
        a smaller step_size will give a more accurate result

    Returns:
    int_val:
        result of the integration
    ====================
    '''
    x = np.arange(xspan[0], xspan[1], step_size)
    int_val = 0
    for xi in x:
        int_val += func(xi) * step_size
    return int_val

# a simple test
def square(x):
    return x**2

def doCos(x):
    return (0.5) * np.cos(x)**2
#print( integrate(func=square, xspan=[0, 1], step_size=0.01) )

print("Answer is")
print( integrate(func=doCos, xspan=[0, np.pi/2], step_size=0.01) )
# or you just call the function without indicating parameters
# print( integrate(square, [0,1], 0.01) )
```

```
Answer is
0.3951990765638034
```

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written analytical solution (or you can use LaTeX). Also, turn in the code you used to numerically evaluate the result and the code output.

[6]:
```
# Specify the path to your PDF file and desired dimensions
pdf_path = "Hw0_Q2.pdf"  # Replace with the actual path to your PDF
width = 800  # Adjust as needed
height = 600  # Adjust as needed

# Create and display the IFrame
IFrame(pdf_path, width=width, height=height)
```

[6]:
```
<IPython.lib.display.IFrame at 0x762694101f70>
```

1.3 Problem 3 (20pts)

For the function $J(x(t))$ in Problem 2, compute and evaluate the analytical solution for the directional derivative of J at $x(t) = \cos(t)$ in the direction $v(t) = \sin(t)$. Note that the directional derivative should be in the form of integration. Evaluate this integral analytically, and verify your answer using the same numerical integration function as in Problem 2.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written (or you can use LaTeX) derivation and evaluation of the directional derivative. Also, include the code used to numerically verify the integration result.

[7]:
```
# Put this in your import cell.
from IPython.display import Markdown, display

# How to print in bold, you could wrap the "display(Markdown())""
# in a function if you want a more consice alternative.
answer = 42
display(Markdown("**Hello World: {}**".format(answer)))

def doSinCos(x):
    return np.sin(x) * np.cos(x)
#print( integrate(func=square, xspan=[0, 1], step_size=0.01) )

print("Answer is: ")
print( integrate(func=doSinCos, xspan=[0, np.pi/2], step_size=0.01) )
# or you just call the function without indicating parameters
# print( integrate(square, [0, 1], 0.01) )
```

Hello World: 42

```
Answer is:
0.4999869977969663
```

[8]:
```
# Specify the path to your PDF file and desired dimensions
pdf_path = "Hw0_Q3.pdf"  # Replace with the actual path to your PDF
width = 800  # Adjust as needed
height = 600  # Adjust as needed

# Create and display the IFrame
IFrame(pdf_path, width=width, height=height)
```

[8]:
```
<IPython.lib.display.IFrame at 0x762663502ea0>
```

1.4 Problem 4 (20pts)

Verify your answer in Problem 3 symbolically using Python's SymPy package, this means you need to compute the directional derivative and evaluate the integration all symbolically.

Hint 1: Different from computing directional derivative in Problem 1, this time the function includes integration. Thus, instead of defining x as a symbol, you should define x as a function of symbol *t. An example of defining function and taking the derivative of the function integration is provided below.*

```
[9]: t = sym.symbols('t')
     eps = sym.symbols('epsilon')

     # define function x and y
     x = sym.Function('x')(t)
     v = sym.Function('v')(t)


     # define J(x(t), y(t))
     J = sym.integrate((1/2)*(x+eps*v)**2, [t, 0, sym.pi/2])
     print('J(x(t), y(t)) = ')
     display(J)

     # take the time derivative of J(x(t))
     dJdepsilon = J.diff(eps)
     print('derivative of J(x(t), v(t)) wrt x(t): ')

     dJdepsilon = dJdepsilon.subs(eps, 0)
     display(dJdepsilon)

     # now, we have x(t)=sin(t) and y(t)=cos(t), we substitute them
     # in, and evaluate the integration
     dJdepsilon_subs = dJdepsilon.subs({x:sym.sin(t), v:sym.cos(t)})
     print('derivative of J, after substitution: ')
     display(dJdepsilon_subs)
     print('evaluation of derivative of J, after substitution: ')
     display(sym.N(dJdepsilon_subs))
```

```
J(x(t), y(t)) =
```

$$0.5 \int\limits_{0}^{\frac{\pi}{2}} \epsilon^2 v^2(t)\, dt + 0.5 \int\limits_{0}^{\frac{\pi}{2}} 2\epsilon v(t) x(t)\, dt + 0.5 \int\limits_{0}^{\frac{\pi}{2}} x^2(t)\, dt$$

```
derivative of J(x(t), v(t)) wrt x(t):
```

$$0.5 \int\limits_{0}^{\frac{\pi}{2}} 2 v(t) x(t)\, dt + 0.5 \int\limits_{0}^{\frac{\pi}{2}} 0\, dt$$

```
derivative of J, after substitution:
```

$$0.5 \int\limits_{0}^{\frac{\pi}{2}} 0\, dt + 0.5 \int\limits_{0}^{\frac{\pi}{2}} 2 \sin(t) \cos(t)\, dt$$

```
evaluation of derivative of J, after substitution:
```

0.5

1.5 Problem 5 (20pts)

Given the equation:

$$xy + \sin(x) = x + y$$

Use Python's SymPy package to symbolically solve this equation for y, thus you can write y as a function of x. Transfer your symbolic solution into a numerical function and plot this function for $x \in [0, \pi]$ with Python's Matplotlib package.

In this problem you will use two methods in SymPy. The first is its symbolic sovler method **solve()**, which takes in an equation or expression (in this it equals 0) and solve it for one or one set of variables. Another method you will use is **lambdify()**, which can transfer a symbolic expression into a numerical function automatically (of course in this problem we can hand code the function, but later in the class we will have super sophisticated expression to evaluate.

Below is an example of using these two methods for an equation $2x^3 \sin(4x) = xy$ (feel free to take this as the start point for your solution):

[10]:
```python
# from sympy.abc import x, y # it's same as defining x, y using symbols() OR
x, y = sym.symbols(r'x,y')

# define an equation
eqn = sym.Eq(x*y + sym.sin(x), x+y)
print('Original equation')
display(eqn)

# solve this equation for y
y_sol = sym.solve(eqn, y) # this method returns a list,
                          # which may include multiple solutions
print('Symbolic solutions')
print(y_sol)
y_expr = y_sol[0] # in this case we just have one solution

# lambdify the expression wrt symbol x
func = sym.lambdify(x, y_expr)
print('Test: func(1.0) = ', func(1.0))

##############
# now it's time to plot it from 0 to pi

# generate list of values from 0 to pi
x_list = np.linspace(0, np.pi, 100)

# evaluate function at those values
f_list = func(x_list)

# plot it
```

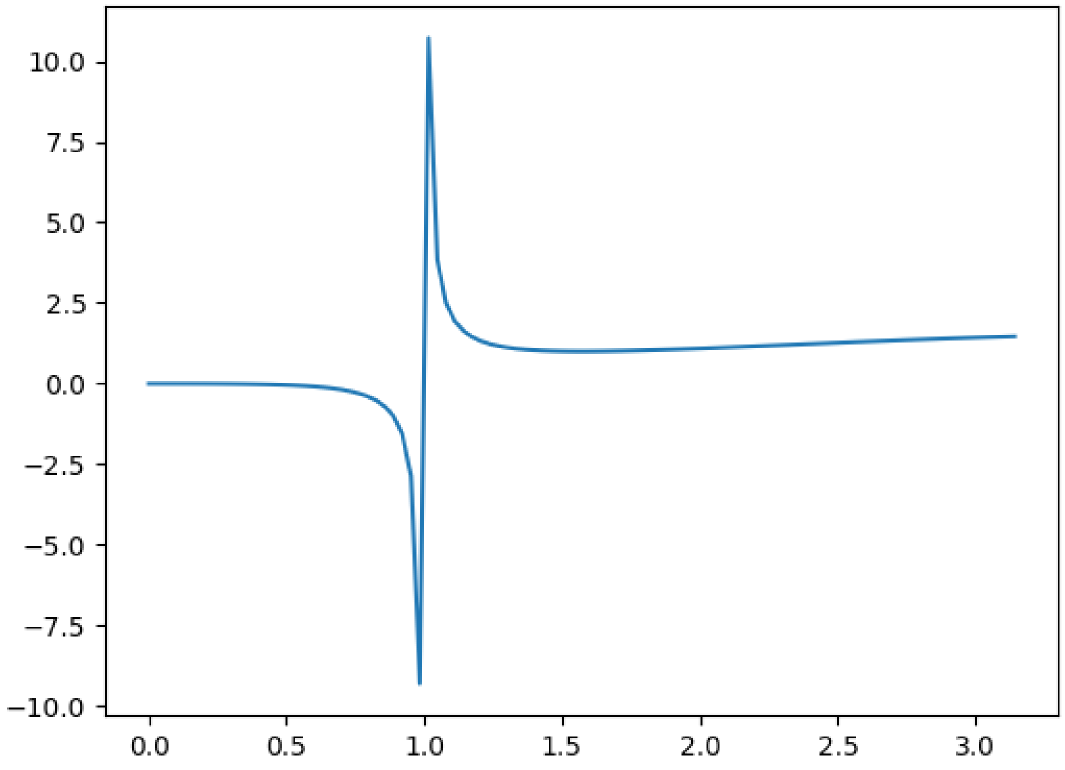
```
plt.plot(x_list, f_list)
plt.show()
```

```
Original equation
```

$xy + \sin(x) = x + y$

```
Symbolic solutions
[(x - sin(x))/(x - 1)]
Test: func(1.0) =  inf
```

```
<lambdifygenerated-1>:2: RuntimeWarning: divide by zero encountered in scalar
divide
  return (x - sin(x))/(x - 1)
```



```
[ ]:
```