ME314 Homework 0 -- Due April 07, 2021

*Please note that a **single** PDF file will be the only document that you turn in, which will include your answers to the problems with corresponding derivations, and the code used to complete the problems. Problems and deliverables that should be included with your submission are shown in **bold**.*

This Juputer Notebook file serves as a template for you to start homework, since we recommend to finish the homework using Jupyter Notebook. You can start with this notebook file with your local Jupyter environment, or upload it to Google Colab. You can include all the code and other deliverables in this notebook Jupyter Notebook supports $\LaTeX$ for math equations, and you can export the whole notebook as a PDF file. But this is not the only option, if you are more comfortable with other ways, feel free to do so, as long as you can submit the homework in a single PDF file.

If you are using Google Colab, make sure you first copy this template to your own Google driver (click "File" -> "Save a copy in Drive"), then start to edit it.

In [1]:

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

# import sympy as sym
# def custom_latex_printer(exp,**options):
#     from google.colab.output._publish import javascript
#     url = "https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.1.1/latest.js?c
#     javascript(url=url)
#     return sym.printing.latex(exp,**options)
# sym.init_printing(use_latex="mathjax",latex_printer=custom_latex_printer)
```

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.

Hint 1: As an example, below is the code solving the problem when $f(x) = x^2$ (feel free to take it as a start point for your solution).

In [2]:

```
import sympy as sym
from sympy import symbols

####################################
# Part 1: compute derivative of f

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

# define the function f
f = x**2 # if you're using Jupyter-Notebook, try "display(f)"

# compute derivative of f
# (uncomment next line and add your code)
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 = symbols(r'\epsilon, v')

# add eplision into function f
new_f = (x + v*eps)**2

# 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:

$2x$

derivative of f wrt eps:

$2v\left(\epsilon v + x\right)$

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

$2vx$

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.

In [50]:

```
# You can start your implementation here :)

x = symbols('x')
print('----------------')
print('define the function, f(x) = ')
f = sym.sin(x) #define the function
display(f) #display the symbolic function

df = f.diff(x) #get derivative
print('----------------')
print('get the derivative, df(x) = ')
display(df)

eps, v = symbols(r'\epsilon, v')

new_f = sym.sin(x + v*eps)
df_eps = new_f.diff(eps)
print('----------------')
print("original function represented with direction v: ")
print(new_f)
print()
print('----------------')
print("derivative of f wrt eps: ")
print(df_eps)
print()

#evaluate at eps = 0
new_df = df_eps.subs(eps, 0)
print('----------------')
print("directional derivative of f wrt eps, Df(x)*v: ")
display(new_df)
```

```
----------------
define the function, f(x) =
```

$\sin(x)$

```
----------------
get the derivative, df(x) =
```

$\cos(x)$

```
----------------
original function represented with direction v:
sin(\epsilon*v + x)

----------------
derivative of f wrt eps:
v*cos(\epsilon*v + x)

----------------
directional derivative of f wrt eps, Df(x)*v:
```

$y \cos(x)$

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:

In [4]:

```
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
    ====================
    '''
    import numpy as np
    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
print( integrate(func=square, xspan=[0, 1], step_size=0.01) )
# or you just call the function without indicating parameters
# print( integrate(square, [0, 1], 0.01) )
```

```
0.32835000000000014
```

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.

In [41]:

```
from math import pi, cos

def integrate(func, xspan, step_size):
    '''
    Numerical integration with Euler's Method

    Args:
    =====================
    func: Python function
    func is the function you want to perform integration on
    xspan: list
    xspan is a list of two elements, used for the integrand bounds
    (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 operation
    '''

    import numpy as np
    x = np.arange(xspan[0], xspan[1], step_size)
    int_value = 0
    for xi in x:
        int_value += func(xi) * step_size
    return int_value

def cosine(x):
    return cos(x)**2

print('')
print(integrate(cosine, [0, pi/2], 0.001))
```

```
0.7858981634134686
```

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 $v(t) = \sin(t)$. The directional derivative should be in the form of integration, evaluate the integration analytically, and verify it using numerical integration.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written analytical solution (or you can use $\LaTeX$), you need to evaluate the integration in this problem. Also, include the code used to numerically verify the integration result.

In [51]:

```
##See attached work in pdf
```

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

In [7]:

```
import sympy as sym
from sympy import symbols, integrate, Function, pi, cos, sin
from sympy.abc import t

# define function x and y
x = Function('x')(t)
y = Function('y')(t)
# define J(x(t), y(t))
J = integrate(x**2 + x*y, [t, 0, pi])
print('J(x(t), y(t)) = ')
display(J)

# take the time derivative of J(x(t))
dJdx = J.diff(x)
print('derivative of J(x(t), y(t)) wrt x(t): ')
display(dJdx)

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

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

$$\int\limits_{0}^{\pi} (x(t) + y(t))\, x(t)\, dt$$

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

$$\int\limits_{0}^{\pi} (2x(t) + y(t))\ dt$$

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

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

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

$$4.0$$

Turn in: A copy of the code you used to numerically and symbolically evaluate the solution.

In [40]:

```
from sympy import symbols, integrate, Function, pi, cos, sin
from sympy.abc import t

#define functions x and y
x = Function('x')(t) #define x as a function of t. Different than just simply
v = Function('v')(t) #again, same as above. these are time bound functions so
eps = symbols(r'\epsilon')

#define function f
print('original function: ')
f = (1/2)*(x+v*eps)**2
display(f)
print('----------------')
#take directional derivative
df = f.diff(eps)

#define the integral
print('Directional derivative of the integral: ')
J = integrate(df.subs(eps, 0), [t, 0 , pi/2]) #substitute eps = 0 and do the
print('J(x(t)) = ')
display(J)

J_subs = J.subs({x:sin(t), v:cos(t)})
print('----------------')
print('substitute for x = sin(t) and v = cos(t):')
display(J_subs)
print('----------------')
print('solve the integration for bound 0 to pi/2:')
display(sym.N(J_subs))
```

```
original function:
```

$$0.5(\epsilon v(t) + x(t))^2$$

```
----------------
Directional derivative of the integral:
J(x(t)) =
```

$$1.0 \int_0^{\frac{\pi}{2}} v(t)x(t)\,dt$$

```
----------------
substitute for x = sin(t) and v = cos(t):
```

$$1.0 \int_0^{\frac{\pi}{2}} \sin(t)\cos(t)\,dt$$

```
----------------
solve the integration for bound 0 to pi/2:
```

$$0.5$$

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):

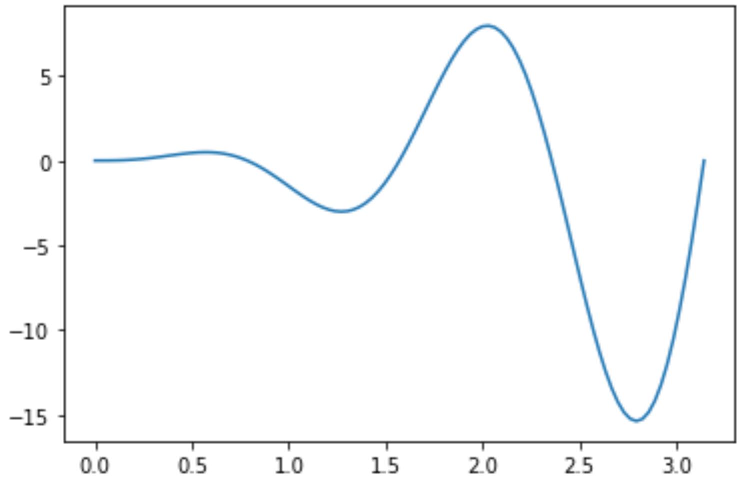
In [9]:

```
import sympy as sym
import numpy as np
from sympy import sin, cos
from sympy.abc import x, y # it's same as defining x, y using symbols()
import matplotlib.pyplot as plt

# define an equation
eqn = sym.Eq(x**3 * 2*sin(4*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
```

$$2x^3 \sin(4x) = xy$$

```
symbolic solutions:
[2*x**2*sin(4*x)]
Test: func(1.0) =  -1.5136049906158564
```



Turn in: A copy of the code used to solve for symbolic solution and evaluate it as a numerical function. Also, include the plot of the numerical function.

In [33]:

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

from math import pi
from sympy import sin, cos
from sympy.abc import x, y


eqn = sym.Eq(x*y + sin(x), x+y)
print()
print('the original equation:')
display(eqn)

y_sol = sym.solve(eqn, y) #solve for y

print()
print("the symbolic solution looks like this: ")
y_expr = y_sol[0]
print(y_expr)
print()

#convert symbolic representation to numeric representation so it can be solve
func = sym.lambdify(x, y_expr)
print('Test: func(pi) = ', func(pi)) #evaluate the equation for f(pi) using f
```

```
the original equation:
```

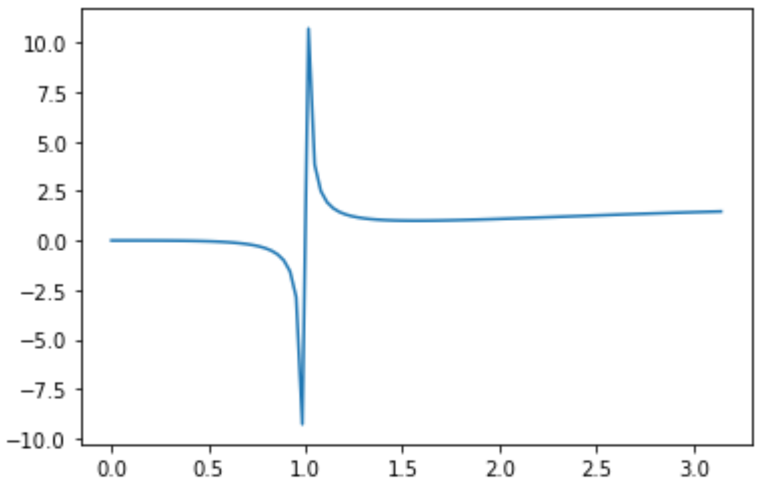
$$xy + \sin(x) = x + y$$

```
the symbolic solution looks like this:
(x - sin(x))/(x - 1)

Test: func(pi) =  1.46694220692426
```

In [35]:

```
############
#plot it
print()
print()
print('Plot y(x) for x = 0 to pi')

x_list = np.linspace(0, pi, 100) #generate a list of evenly spaced out 100 nu

f_list = func(x_list) #evaluate at those values

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

```
Plot y(x) for x = 0 to pi
```



In [ ]: