ME314_HW0_Solution_Spring22

April 4, 2022

1 ME314 Homework 0 (Solution)

1.0.1 Submission instructions

Deliverables that should be included with your submission are shown in **bold** at the end of each problem statement and the corresponding supplemental material. Your homework will be graded IFF you submit a **single** PDF and a link to a Google colab file that meet all the requirements outlined below.

- List the names of students you've collaborated with on this homework assignment.
- Include all of your code (and handwritten solutions when applicable) used to complete the problems.
- Highlight your answers (i.e. **bold** and outline the answers) and include simplified code outputs (e.g. .simplify()).
- Enable Google Colab permission for viewing
- Click Share in the upper right corner
- Under "Get Link" click "Share with..." or "Change"
- Then make sure it says "Anyone with Link" and "Editor" under the dropdown menu
- Make sure all cells are run before submitting (i.e. check the permission by running your code in a private mode)
- Please don't make changes to your file after submitting, so we can grade it!
- Submit a link to your Google Colab file that has been run (before the submission deadline) and don't edit it afterwards!

NOTE: This Juputer Notebook file serves as a template for you to start homework. Make sure you first copy this template to your own Google driver (click "File" -> "Save a copy in Drive"), and then start to edit it.

[1]:
```
#Import cell
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
##########################################################################################
# If you're using Google Colab, uncomment this section by selecting the whole␣
 ↪section and press
# ctrl+'/' 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
##########################################################################################
# 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?
 ↪config=TeX-AMS_HTML"
#     javascript(url=url)
#     return sym.printing.latex(exp,**options)
# sym.init_printing(use_latex="mathjax",latex_printer=custom_latex_printer)
```

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.

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).

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

# define your symbolic variable here
x = sym.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 results
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 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.

Analytical solution:

$$\begin{aligned}\frac{df(x)}{dx}v &= \frac{d}{d\epsilon}\big(\sin(x+\epsilon v)\big)\Big|_{\epsilon=0} \\ &= \big(\cos(x+\epsilon v)v\big)\Big|_{\epsilon=0} \\ &= \cos(x)v\end{aligned}$$

[4]:
```
# You can start your implementation here :)
#####################################
# 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
# (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 = sym.symbols(r'\epsilon, v')

# add eplision 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:`

$v\cos(x)$

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]:
```python
def euler_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( euler_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.

Analytical solution:

When $x = \cos(t)$, we have

$$\begin{aligned} J(x(t)) &= \int_0^{\pi/2} \frac{1}{2}\cos(t)^2 dt \\ &= \frac{1}{4}\int_0^{\pi/2} (2\cos(t)^2 - 1)dt + \frac{1}{4}\int_0^{\pi/2} dt \\ &= \frac{1}{4}\int_0^{\pi/2} \cos(2t)dt + \frac{1}{4}\int_0^{\pi/2} dt \\ &= \frac{1}{8}\int_0^{\pi} \cos(2t)d(2t) + \frac{\pi}{8} \\ &= \frac{\pi}{8} \end{aligned}$$

[6]:
```
# You can start your implementation here :)
def func(t):
    return 0.5 * np.cos(t)**2

Jx = euler_integrate(func, [0, np.pi/2], 0.01)
print('J(cos(t)) = ', Jx)
print('pi / 8 = ', np.pi/8)
```

```
J(cos(t)) =  0.3951990765638034
pi / 8 =  0.39269908169872414
```

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 $v(t) = \sin(t)$. 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.

Analytical solution:

$$
\begin{aligned}
\frac{\partial J(x)}{\partial x} \cdot v &= \frac{d}{d\epsilon}\left(J(x+\epsilon v)\right)\Big|_{\epsilon=0} \\
&= \frac{d}{d\epsilon}\left[\int_0^{\pi/2} \frac{1}{2}\big(x(t)+\epsilon v(t)\big)^2 dt\right]\Big|_{\epsilon=0} \\
&= \left[\int_0^{\pi/2} \big(x(t)+\epsilon v(t)\big)v(t)dt\right]\Big|_{\epsilon=0} \\
&= \int_0^{\pi/2} x(t)v(t)dt \\
&= \int_0^{\pi/2} \sin(t)\cos(t)dt \\
&= \frac{1}{2}\int_0^{\pi/2} \sin(2t)d(t) \\
&= \frac{1}{4}\int_0^{\pi} \sin(2t)d(2t) \\
&= -\frac{1}{4}\cos(t)|_0^{\pi/2} \\
&= 0.5
\end{aligned}
$$

[7]:
```
# You can start your implementation here :)
def func(t):
    return np.sin(t) * np.cos(t)

Jx = euler_integrate(func, [0, np.pi/2], 0.01)
print('J(cos(t)) = ', Jx)
```

```
J(cos(t)) =  0.4999869977969663
```

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.

[8]:
```
# define function x and y
t = sym.symbols('t')
x = sym.Function('x')(t)
y = sym.Function('y')(t)
# define J(x(t), y(t))
J = sym.integrate(x**2 + x*y, [t, 0, sym.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:sym.sin(t), y:sym.cos(t)})
print('derivative of J, after substitution: ')
display(dJdx_subs)
print('evaluation of derivative of J, after substitution: ')
display(sym.N(dJdx_subs))
display(sym.integrate((t)**2, [t, -sym.pi, sym.pi]))
```

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

$$\int_0^{\pi} \left(x(t) + y(t)\right) x(t)\, dt$$

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

$$\int_0^{\pi} \left(2x(t) + y(t)\right)\, dt$$

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

$$\int_0^{\pi} \left(2\sin\left(t\right) + \cos\left(t\right)\right)\, dt$$

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

$$4.0$$

$$\frac{2\pi^3}{3}$$

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

[9]:
```
# You can start your implementation here :)
t = sym.symbols('t')
# define x as function of t
x = sym.Function('x')(t)
# define J(x) as function of x(t)
J = sym.integrate(0.5*x**2, [t, 0, sym.pi/2])
print('J(x(t)) = ')
display(J)
```

```
# define direction to be taken derivative with
v = sym.Function('v')(t)
# define dummy variable epsilon as a symbol
eps = sym.symbols(r'\epsilon')
# add epsilon and v(t) into J(x(t))
J_eps = sym.integrate(0.5*(x+v*eps)**2, [t, 0, sym.pi/2])
# take derivative wrt epsilon
dJ_deps = J_eps.diff(eps)
# evaluate at epsilon=0
dJ_deps_eps0 = dJ_deps.subs(eps, 0)
# substitute x(t)=cos(t), v(t)=sin(t)
dJ_dv = dJ_deps_eps0.subs({x:sym.cos(t), v:sym.sin(t)})
print('directional derivative of J with x=cos(t) and y=sin(t): ')
display(dJ_dv)
print('evaluation of directional derivative of J with x=cos(t) and y=sin(t): ')
display(sym.N(dJ_dv))
```

```
J(x(t)) =
```

$$0.5 \int_0^{\frac{\pi}{2}} x^2(t)\, dt$$

```
directional derivative of J with x=cos(t) and y=sin(t):
```

$$0.5 \int_0^{\frac{\pi}{2}} 0\, dt + 0.5 \int_0^{\frac{\pi}{2}} 2 \sin(t) \cos(t)\, dt$$

```
evaluation of directional derivative of J with x=cos(t) and y=sin(t):
```

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.

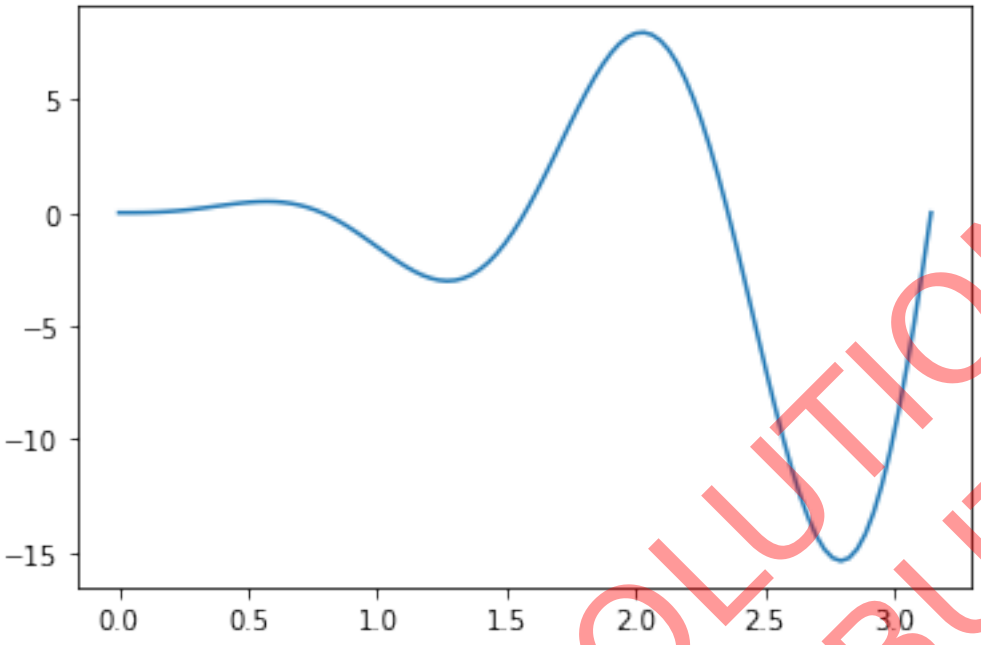
Below is an example of using these two methods for an equation $2x^3 \sin(4x) = xy$ (feel free to take this as the start point for your solution):

[10]:
```
# from sympy.abc import x, y # it's same as defining x, y using symbols()
#its good coding practice to import everything at the top
x = sym.symbols('x')
y = sym.symbols('y')

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

[11]:
```
# You can start your implementation here :)
# define an equation
#its good coding practice to import everything at the top
x = sym.symbols('x')
y = sym.symbols('y')

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)) # a warning message is expected because␣
 ↪we are dividing something by zero

###############
```

```
# 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

<string>:2: RuntimeWarning: divide by zero encountered in double_scalars
```