section_1

February 7, 2025

1 Section 1 - Python code review

Question (i) **(i)** This is a copy of the Python Code Review notebook

[7]:
```
#Code CELL #1
import sympy
def compute_tangent_line(my_func, x0):
  x = sympy.Symbol('x')
  # Convert the lambda function to a symbolic expression
  func_expr = sympy.sympify(my_func(x))
  # Evaluate my_func at the given point
  y0 = my_func(x0)
  # Compute the derivative of my_func symbolically
  m = sympy.diff(func_expr, x).subs(x, x0)
  # Define function object for equation of tangent line
  tangent_line = lambda x_val: m * (x_val - x0) + y0
  return tangent_line
```

Question (ii) **(ii)** We're defining a function 'compute_tangent_line', it's return values and types aren't explicitly declared, but given the naming we can assertain that the first argument is a function that takes and returns a single floating point value, that the second is a point x_0, a value x from which to cast the tangent line.

Reading further we can assertain the return type to be a function, a line, that takes and returns singular floating points

Question (iii) **(iii)** We're modeling the function

$$f(x) = \frac{1}{2}x(x-2)(x+2)$$

And calling 'compute_tangent_line' with $x_0 = -1.5$ and the function defined input

[8]:
```
#Code CELL #2

cubic_function = lambda x : (1/2)*x*(x-2)*(x+2)
tangent_line_func = compute_tangent_line(cubic_function, -1.5)
```

Question (iv) **(iv)** Now were going to evaluate at $x = 10$

[9]:
```
#Code CELL #3

tangent_line_func(10)
```

[9]: 17.125

Question (v) **(v)** This illustrates the use of numpy's linspace to generate a range

[10]:
```
#Code CELL #4
import numpy as np
import matplotlib.pyplot as plt

x_vals = np.linspace(0,2,5)

print(x_vals)
```

```
[0.  0.5 1.  1.5 2. ]
```

Question (vi) **(vi)** Lastly, we're going to take what's done so far and apply it to a different set of numbers - We're generate a new set of x values, between -3, 3, 100 samples - We generate a line on our plot using the initial intercept of $x = -1.5$ - We then compute a new tangent line from $x = -3$ - Then plot that over the same range of x values

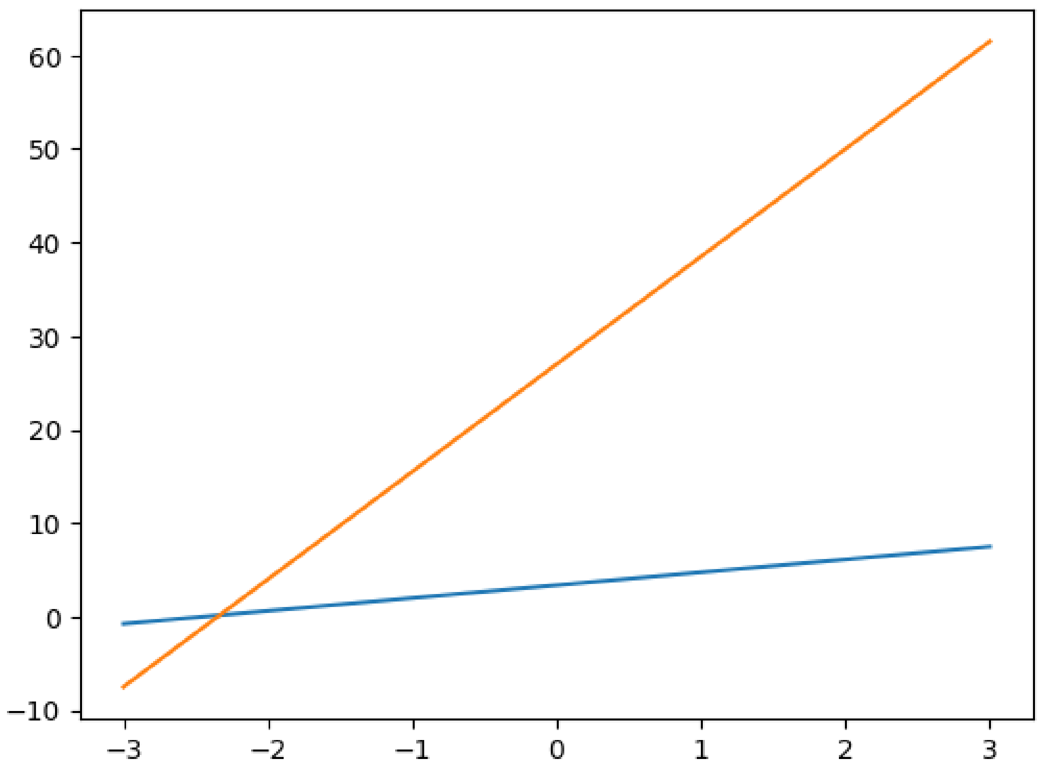
[11]:
```
x_vals = np.linspace(-3, 3, 100)

tan_y_vals = tangent_line_func(x_vals)

plt.plot(x_vals, tan_y_vals)

#Computing the tangent line instead now for -3
tangent_line_func = compute_tangent_line(cubic_function, -3)
plt.plot(x_vals, tangent_line_func(x_vals))
```

[11]:
```
[<matplotlib.lines.Line2D at 0x777ca1fe3c50>]
```


60
50
40
30
20
10
0
−10
−3
−2
−1
0
1
2
3