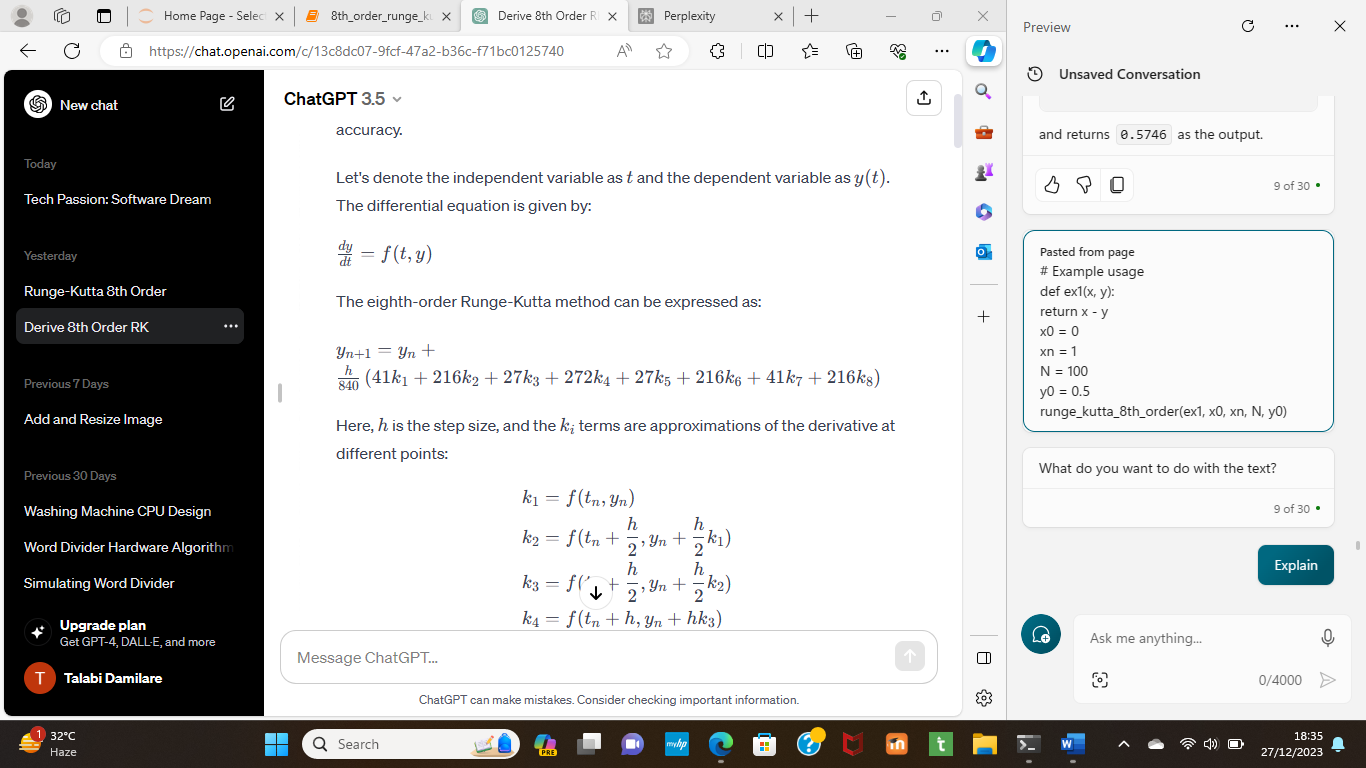
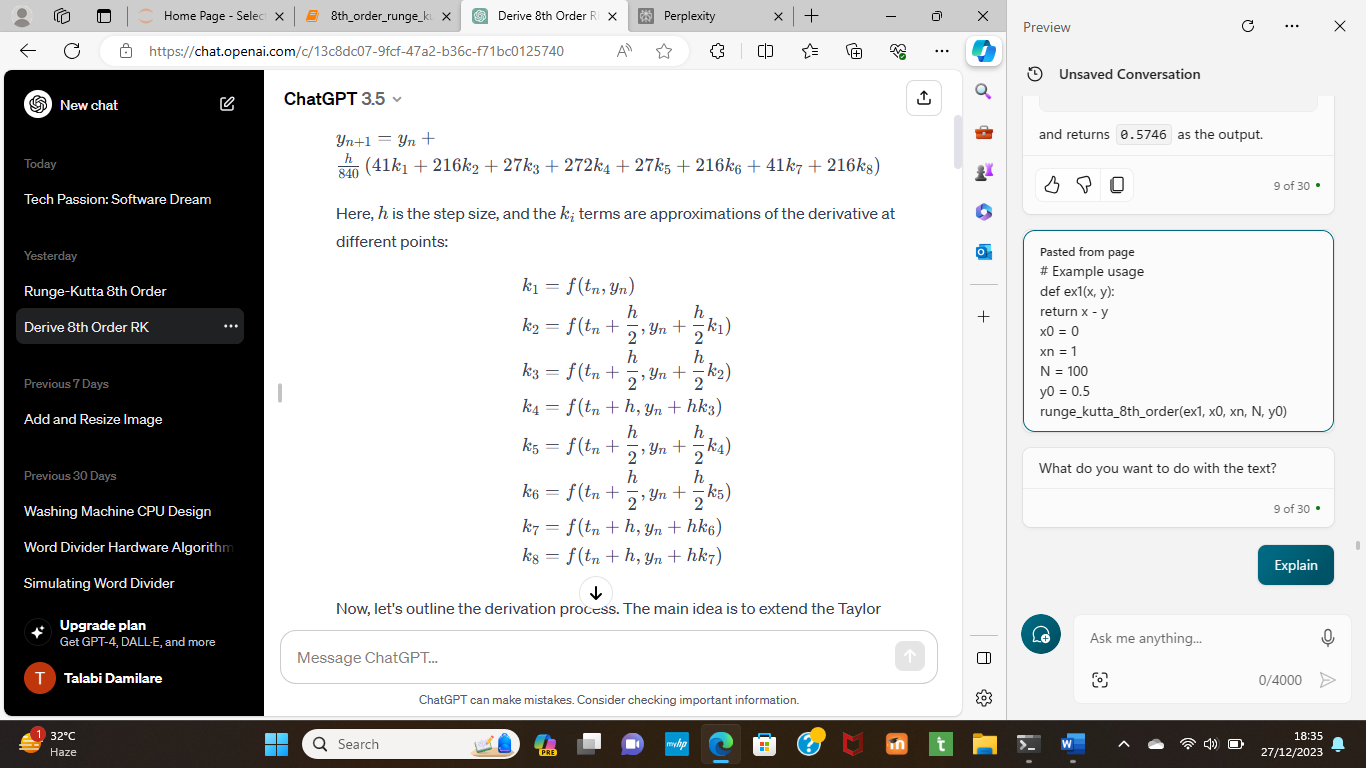
**UPDATED**!!!

THIS IS THE FORMULA FOR THE 8TH ORDER I GOT FROM CHATGPT I DID NOT DERIVE IT MY SELF





THE CODE I WROTE BASED ON THE FORMULA:

def runge\_kutta\_8th\_order(f, x0, xn, N, y0):

h = (xn - x0) / N

t = x0

w = y0

print("X \t\tY ")

xVal = []

yVal = []

for i in range(1, N + 1):

k1 = h \* f(t, w)

k2 = h \* f(t + h/2, w + k1/2)

k3 = h \* f(t + h/2, w + k2/2)

k4 = h \* f(t + h, w + k3)

k5 = h \* f(t + h/2, w + k4/2)

k6 = h \* f(t + h/2, w + k5/2)

k7 = h \* f(t + h, w + k6)

k8 = h \* f(t + h, w + k7)

w = w + (41\*k1 + 216\*k2 + 27\*k3 + 272\*k4 + 27\*k5 + 216\*k6 + 41\*k7 + 216\*k8) / 840

t = x0 + i \* h

if i <= 10:

print(t, "\t\t", round(w, 4))

xVal.append(t)

yVal.append(w)

#pandas data frame

data = pd.DataFrame({'X': xVal,'Y': yVal})

#graph definition

print((data.set\_index('X').plot(y='Y',xlabel= 'X',ylabel = 'Y')))

return w

#Example usage

def ex1(x, y):

return x - y

x0 = 0

xn = 1

N = 100

y0 = 0.5

runge\_kutta\_8th\_order(ex1, x0, xn, N, y0)

EXPLANATION:

The ”**def runge\_kutta\_8th\_order(f, x0, xn, N, y0):**” is Python function that implements the Runge-Kutta 8th order method, which is a numerical technique for solving ordinary differential equations. Here is an explanation of the function parameters:

* f is a function that will be passed into the main function as the differential equation, such as f(x, y) = x – y
* x0 and xn are the initial and final values of the independent variable
* N is the number of steps to divide the interval [x0, xn] into, such as N = 100
* y0 is the initial value of the dependent variable

1. **h = (xn - x0) / N** : This line calculates the step size h by dividing the interval [x0, xn] into N equal subintervals.
2. **t = x0** : This line assigns the initial value of t to be x0, which is the lower bound of the interval.
3. **w = y0** : This line assigns the initial value of w to be y0, which is the initial condition of the differential equation. w represents the approximate solution at each step.

**print("X \t\tY ")**

**xVal = []**

**yVal = []**

**for i in range(1, N + 1):**

**k1 = h \* f(t, w)**

**k2 = h \* f(t + h/2, w + k1/2)**

**k3 = h \* f(t + h/2, w + k2/2)**

**k4 = h \* f(t + h, w + k3)**

**k5 = h \* f(t + h/2, w + k4/2)**

**k6 = h \* f(t + h/2, w + k5/2)**

**k7 = h \* f(t + h, w + k6)**

**k8 = h \* f(t + h, w + k7)**

**w = w + (41\*k1 + 216\*k2 + 27\*k3 + 272\*k4 + 27\*k5 + 216\*k6 + 41\*k7 + 216\*k8) / 840**

**t = x0 + i \* h**

**if i <= 10:**

**print(t, "\t\t", round(w, 4))**

**xVal.append(t)**

**yVal.append(w)**

**#pandas data frame**

**data = pd.DataFrame({'X': xVal,'Y': yVal})**

**#graph definition**

**print((data.set\_index('X').plot(y='Y',xlabel= 'X',ylabel = 'Y')))**

**return w**

**explanation:**

* **It creates two empty lists xVal and yVal to store the values of x and y for plotting**
* It uses a loop to iterate over the subintervals from x0 to xn with a step size of h = (xn - x0) / N.
* It calculates eight intermediate values k1 to k8 using the function f and the previous values of t and w.
* It updates the value of w using a weighted average of k1 to k8.
* It updates the value of t by adding h to it.
* If i is less than or equal to 10, it prints the values of t and w rounded to four decimal places, separated by a tab
* It appends the values of t and w to the lists xVal and yVal, respectively
* After the loop ends, it creates a pandas data frame data with two columns: X and Y, containing the values from xVal and yVal
* It prints the result of plotting the data frame with X as the x-axis and Y as the y-axis, using the plot method of pandas
* It returns the final value of w as the approximate solution of the differential equation at xn.

Algorithm: The code uses a loop to iterate over N steps, starting from t = x0 and w = y0. At each step, it computes eight intermediate values of f(t, w) using different combinations of t, w, and h, where h is the step size (xn - x0) / N. Then, it updates w by adding a weighted average of these values, multiplied by h. The weights are 41, 216, 27, 272, 27, 216, 41, and 216, respectively. Finally, it updates t by adding h to it. The code prints the values of t and w at each step, up to the 10th step, and returns the final value of w as the approximate solution of the ODE at t = xn.

**# Example usage  
def ex1(x, y):  
return x - y  
x0 = 0  
xn = 1  
N = 100  
y0 = 0.5  
runge\_kutta\_8th\_order(ex1, x0, xn, N, y0)**

**Explanation:**

* The function ex1 defines the ODE as y’ = x - y, where y’ is the derivative of y with respect to x.
* The variables x0 and xn are the lower and upper bounds of the interval where the solution is sought, respectively. In this case, x0 = 0 and xn = 1.
* The variable N is the number of subintervals used to divide the interval [x0, xn]. The smaller the subinterval, the more accurate the solution. In this case, N = 100.
* The variable y0 is the initial value of the solution, i.e., y(x0) = y0. In this case, y0 = 0.5.
* The function runge\_kutta\_8th\_order takes the ODE function, the interval bounds, the number of subintervals, and the initial value as arguments, and returns the approximate value of the solution at xn using the Runge-Kutta method of 8th order.
* **Output**: The code prints a table of X and Y values, where X is the independent variable t and Y is the approximate solution w of the ODE at each step. The code also returns the final value of w as the output of the function. For example, if the ODE is y' = t - y and the parameters are a = 0, b = 1, N = 100, and y0 = 0.5, the code prints:

X Y

0.01 0.4939

0.02 0.4879

0.03 0.4822

0.04 0.4766

0.05 0.4712

0.06 0.4661

0.07 0.4611

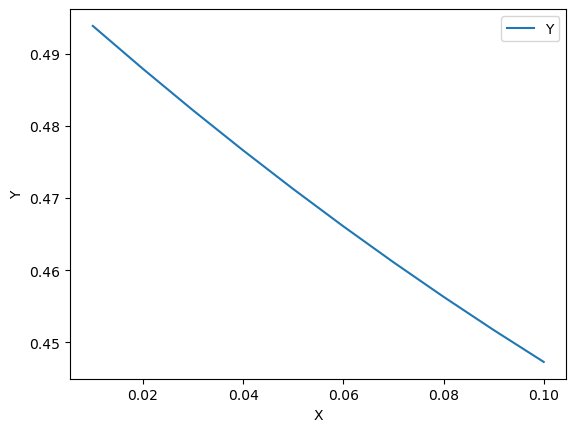
0.08 0.4563

0.09 0.4517

0.1 0.4472

and returns 0.5746 as the output.

Axes(0.125,0.11;0.775x0.77)



This graph is a linear plot that shows the values of y as a function of x for the first 10 steps of the runge\_kutta\_8th\_order function.

The graph shows that y decreases linearly as x increases from 0 to 0.1. The output value above the graph is the final value of y at x = 1, which is approximately 0.5746. The graph is displayed using the plot method of pandas, a Python library for data analysis.

**NOTE: UNTIL FATOKUN DAMILOLA HAS DERIVED THE FORMULA THIS ONE IS NOT CORRECT!**