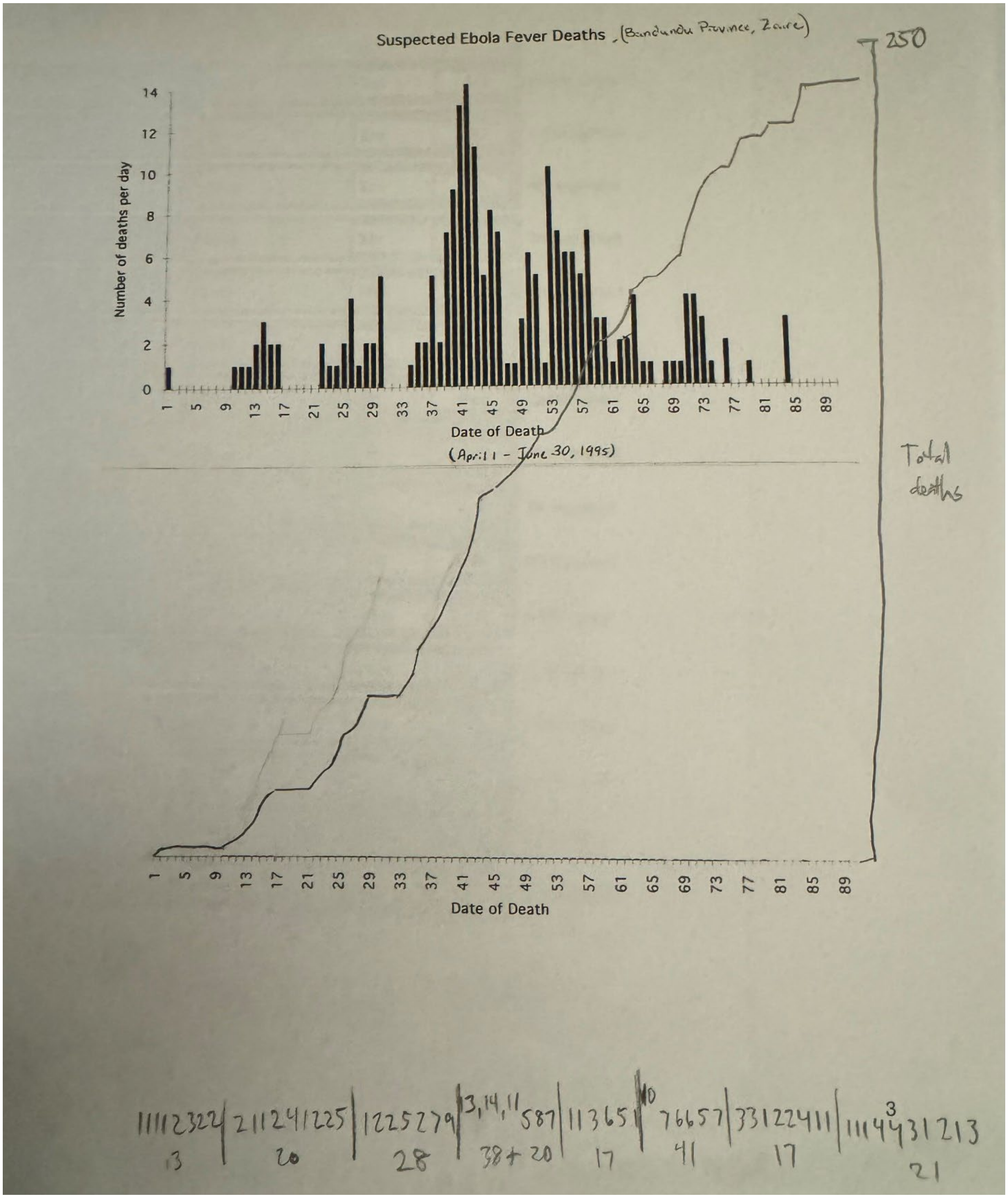
Ebola Project: Christopher Taylor

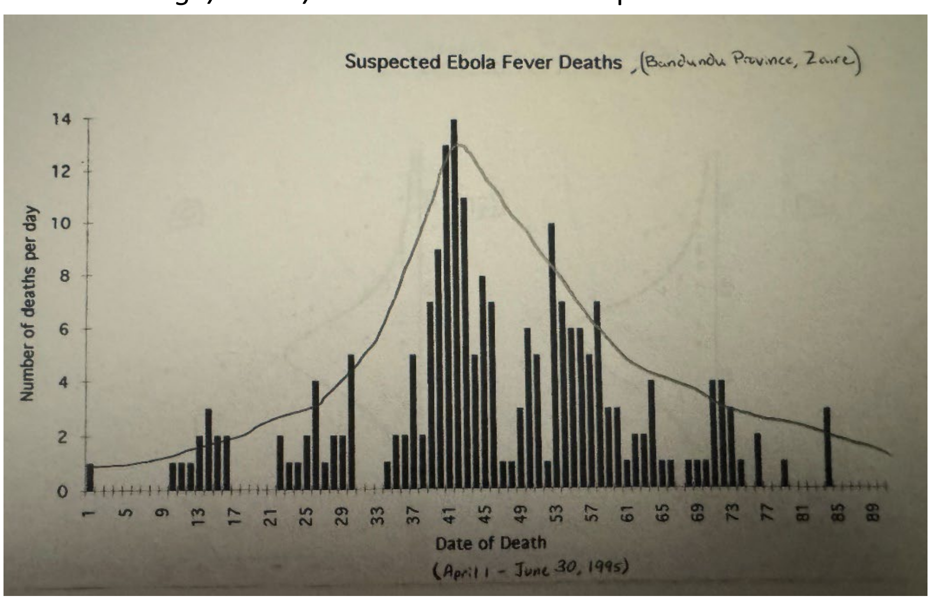
1. Sketch by hand the accumulation function F(x) for the Ebola data provided - draw your sketch underneath the histogram provided. This will be a graph of total deaths versus time. Use a scale of 1 to 250. Add a y-axis. Please carefully count.



2. Using the graph of the accumulation you just created.
   - How many total deaths were there? **215**
   - When was the peak of the illness? **Day 41**
   - How did you decide? **Day 41 had the most deaths per day (14)**
   - What (shape/concept) represents the peak on the original graph versus your graph? **__Original__: the tallest bar of the histogram/barchart. __Mine__: the steepest slope on the line graph**
   - What is the derivative of the accumulation function? **The derivative at that specific day would be dD/dT = 14 where dD is change in deaths and dT is change in time. 14 deaths / 1 day = 14 deaths/day, but the derivative changes depending on which time period (day) we take into account.**
3. Suppose there were 25 deaths before recording started (at time 0). Sketch the graph of total deaths per day for this situation.

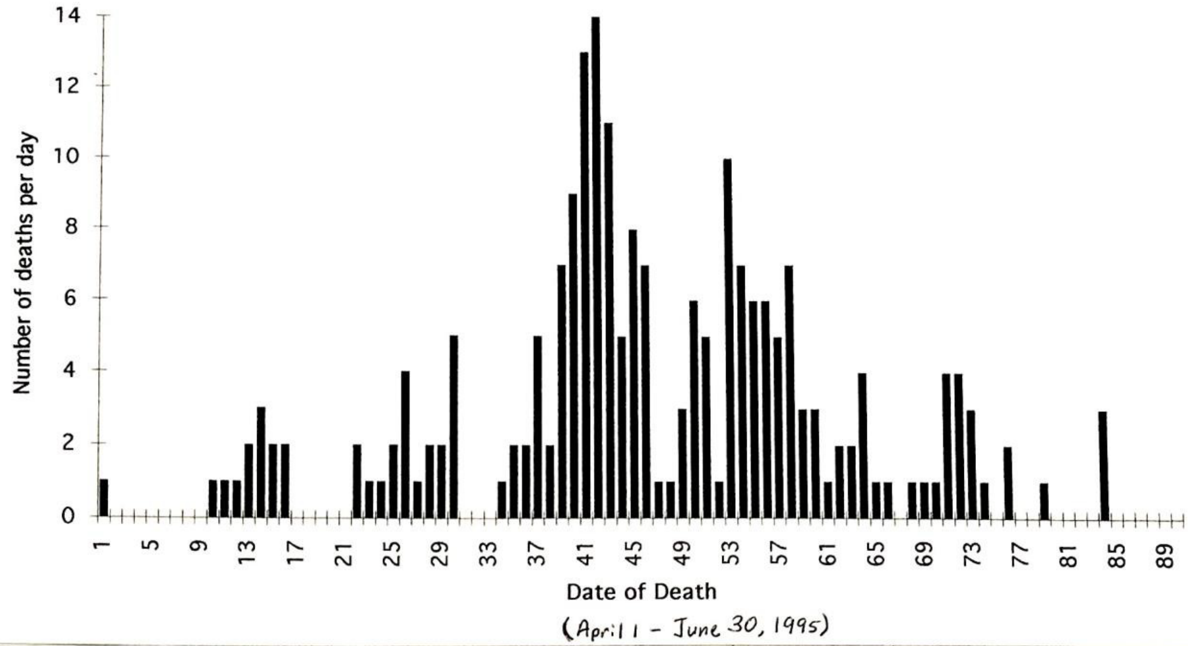
   The graph would be the exact same shape, but instead of starting with the x axis on 0, it would start at 25 and go up from there, with the final total ending at 240.
   - How many total deaths were there? **240**
   - How many deaths were there at time 0? **25**
   - How do we use this information to find the actual total at time 89?

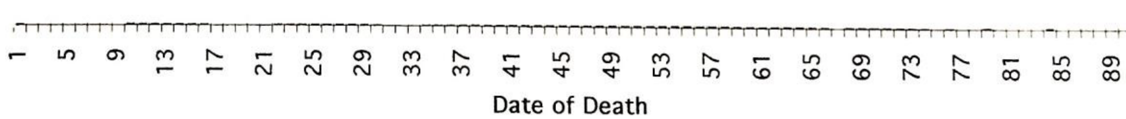
     What I did at the bottom of the picture above is tally up the number of deaths each day and then add them all together, effectively the same thing as summing the accumulation in the For loop of the functions in Activity 2-3 (I actually did those Activities first).
4. If we want to model this data with an algebraic equation we need to smooth the data, that is, make it into a smooth curve.
5. Using a copy of the original graph of the deaths per day on page 2, sketch a “smooth curve” *through/above/below* the stacks that preserves the total number of deaths.



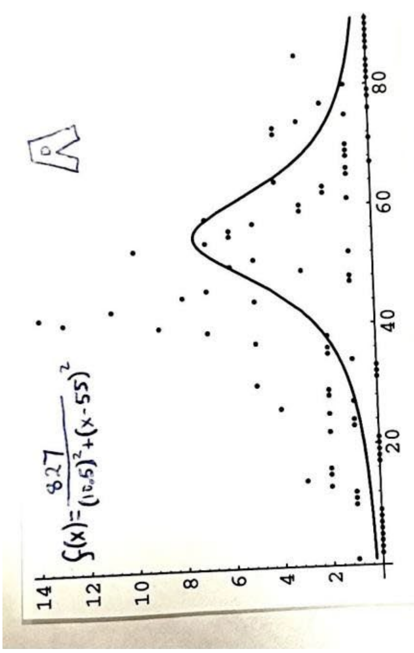
6. Using the curve that you made in #5:
    - Where is the peak of the deaths? **Still Day 41**
    - What will the accumulation function of your new graph look like?
      It would probably be similar to a Logistic function, creating an S curve
    - What is the average number of deaths per day?
      Here, we can do a simple average: 215 deaths / 90 days ≈ 2.4 deaths/day
    - How did you find out? **Took simple average as above**
7. Consider the "smooth" functions we have created on page 3.
    - What is good or bad about each of them?
        1. **Starts off good, but the slope in the middle is a bit too steep, so by the time it levels off again, it's above the actual data plots**
        2. **This one looks pretty darn good. I can't really complain too much about this one at all**
        3. **This one builds a bit too slowly, so by the time of it's steepest slope, it's already under shot the data points and stays a bit low throughout the rest.**
        4. **This one drastically overshoots the data, ending up with far more deaths accumulated than the data actually shows. Up around 350 instead of 215.**
        5. **This one also drastically overshoots the data, although not as much as number 4. We still end up with too many total deaths, up around 300 instead of down around 215.**
        6. **This one probably gets a close second place behind number 2 when it comes to closely matching the data points. Can't complain too much here, although I wonder if the total reaches 215 or ends up just shy of it.**
    - Do they each represent the same number of total deaths?
      Not at all. 1 is up around 250, a bit too high; 2 looks good around 215; 3 looks decent for total, but doesn't match data well; 4 is way off, up around 350 instead of 215; 5 is way above as well with 300 instead of 215; 6 looks good like number 2, right around 215 total.
    - Which is the best fit to the data?
      I would pick number 2, but number 6 is a close second.
8. Match the smooth functions on page with its accumulation function on page 4.
    1. **E**
    2. **D**
    3. **A**
    4. **B**
    5. **C**
    6. **F**

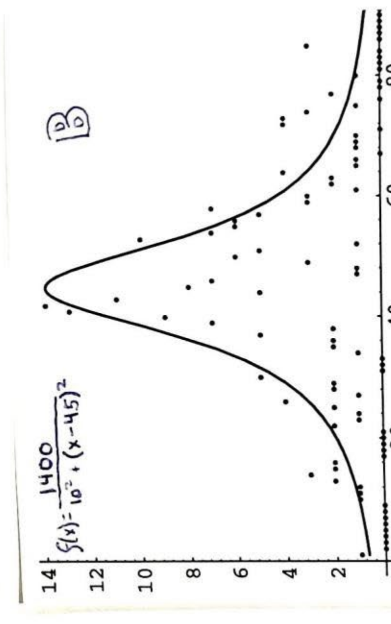
Suspected Ebola Fever Deaths - (Bandundu Province, Zaire)

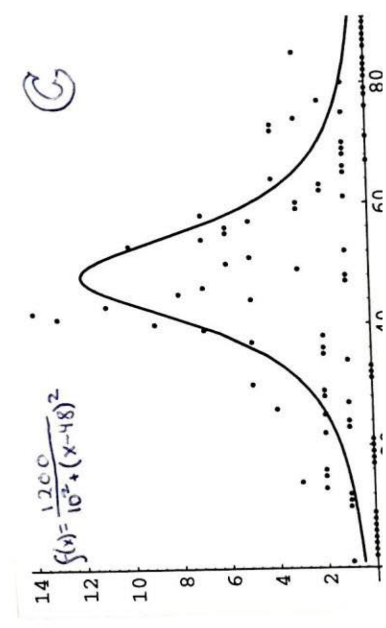


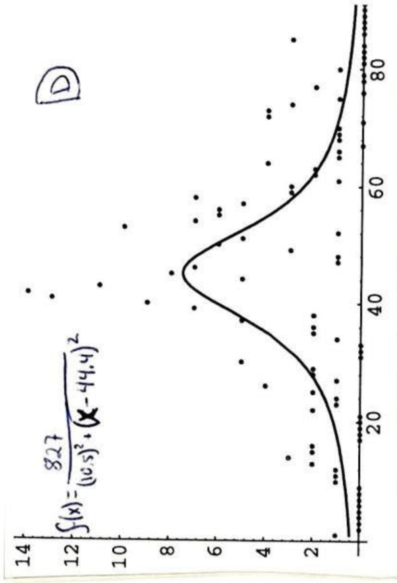


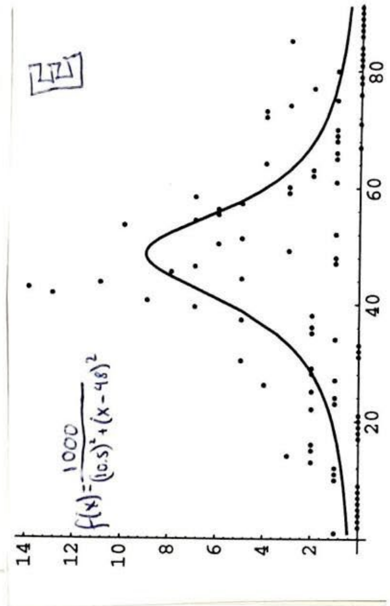
"Smooth" Functions

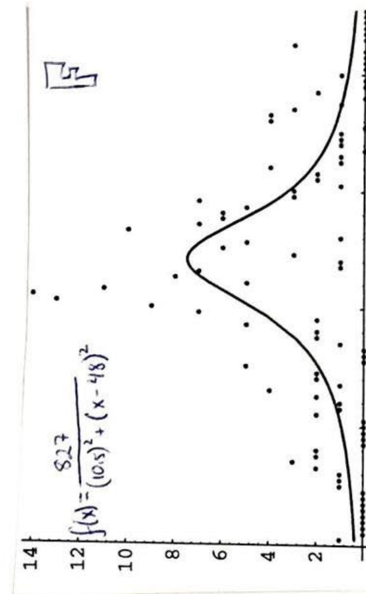




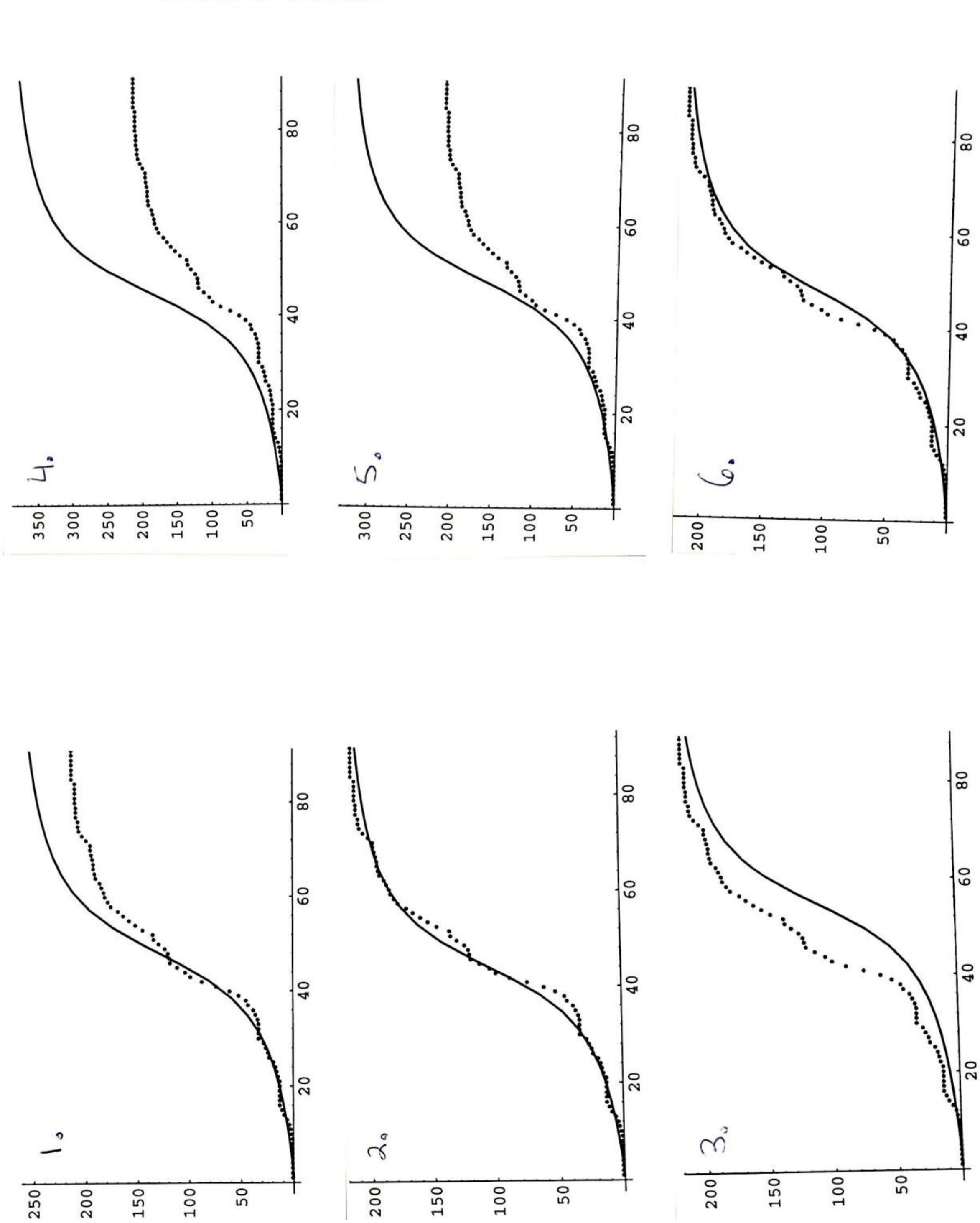








Accumulation functions



HW05 Antiderivative

October 11, 2024

[3]:

```
#Activity 2

import math

#function to find derivative dY/dT
def f(t):
    return math.cos(t**2) #dY/dT = cos(t^2)


#function to create a table of values showing dY, accumulated dY, and dT for␣
 ↪each step in dT
def TABLE(Tinit, Tfinal, steps):
    dT = (Tfinal - Tinit) / steps #definiing dT
    t = Tinit #initializing t with Tinit parameter
    acc = 0 #initializing accumulation variable to keep track of total␣
 ↪accumulation after each step

    #printing the table headers
    print(f"{'Step':<5} {'dY':<15} {'Accumulated dY':<20} {'Ending t':<10}")
    print("=" * 50)

    for k in range(1,steps+1):
        dY = f(t) * dT #finding change in Y per change in time, given function␣
 ↪f (can change function above)
        acc = acc + dY #summing change in Y to get total accumulated
        t = t + dT #updates variable t to current Time step

        #printing each row with formatted values
        print(f"{k:<5} {dY:<15.6f} {acc:<20.6f} {t:<10.6f}") #prints the 3 in a␣
 ↪basic format for tracking
```

[4]:

```
TABLE(1, 14, 100)
```

```
Step  dY              Accumulated dY       Ending t
==================================================
1     0.070239        0.070239             1.130000
```

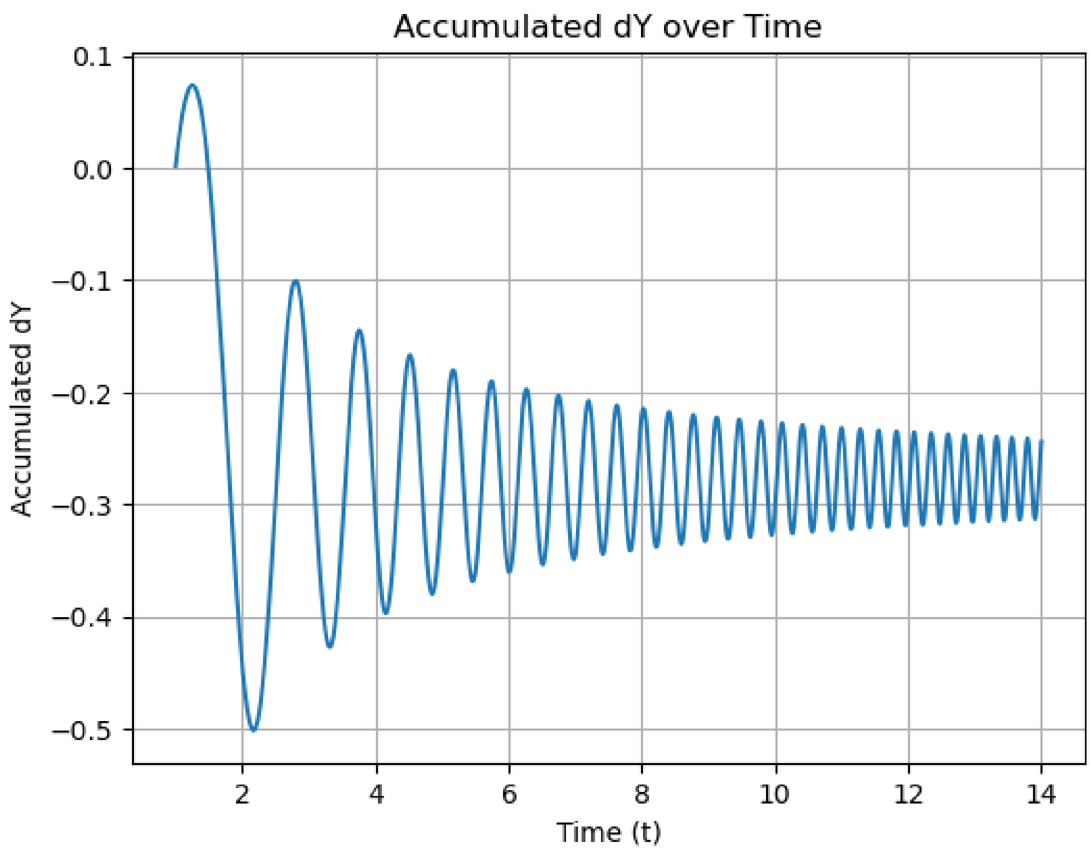
2	0.037659	0.107898	1.260000
3	-0.002184	0.105714	1.390000
4	-0.045954	0.059760	1.520000
5	-0.087619	-0.027860	1.650000
6	-0.118750	-0.146609	1.780000
7	-0.129953	-0.276563	1.910000
8	-0.113678	-0.390240	2.040000
9	-0.068037	-0.458277	2.170000
10	-0.000454	-0.458731	2.300000
11	0.070983	-0.387748	2.430000
12	0.120809	-0.266939	2.560000
13	0.125276	-0.141663	2.690000
14	0.075310	-0.066353	2.820000
15	-0.012774	-0.079126	2.950000
16	-0.097539	-0.176666	3.080000
17	-0.129753	-0.306419	3.210000
18	-0.082898	-0.389316	3.340000
19	0.020715	-0.368602	3.470000
20	0.112461	-0.256140	3.600000
21	0.120058	-0.136082	3.730000
22	0.028911	-0.107171	3.860000
23	-0.089789	-0.196960	3.990000
24	-0.127086	-0.324046	4.120000
25	-0.038959	-0.363005	4.250000
26	0.091771	-0.271233	4.380000
27	0.122780	-0.148453	4.510000
28	0.010422	-0.138032	4.640000
29	-0.116397	-0.254429	4.770000
30	-0.094072	-0.348501	4.900000
31	0.056318	-0.292183	5.030000
32	0.128166	-0.164017	5.160000
33	0.010122	-0.153895	5.290000
34	-0.124563	-0.278458	5.420000
35	-0.058728	-0.337186	5.550000
36	0.106298	-0.230888	5.680000
37	0.086142	-0.144746	5.810000
38	-0.090439	-0.235185	5.940000
39	-0.097212	-0.332397	6.070000
40	0.085386	-0.247011	6.200000
41	0.095923	-0.151088	6.330000
42	-0.093165	-0.244253	6.460000
43	-0.081733	-0.325986	6.590000
44	0.110542	-0.215444	6.720000
45	0.049988	-0.165456	6.850000
46	-0.127373	-0.292829	6.980000
47	0.003342	-0.289486	7.110000
48	0.124697	-0.164789	7.240000
49	-0.071387	-0.236176	7.370000

50	-0.079805	-0.315981	7.500000
51	0.124245	-0.191736	7.630000
52	-0.012647	-0.204383	7.760000
53	-0.112339	-0.316722	7.890000
54	0.108758	-0.207965	8.020000
55	0.010680	-0.197284	8.150000
56	-0.117112	-0.314396	8.280000
57	0.110375	-0.204021	8.410000
58	-0.005493	-0.209514	8.540000
59	-0.101494	-0.311008	8.670000
60	0.126597	-0.184412	8.800000
61	-0.058989	-0.243400	8.930000
62	-0.046490	-0.289890	9.060000
63	0.119627	-0.170264	9.190000
64	-0.121348	-0.291611	9.320000
65	0.058713	-0.232898	9.450000
66	0.030003	-0.202895	9.580000
67	-0.101880	-0.304775	9.710000
68	0.129914	-0.174861	9.840000
69	-0.109882	-0.284743	9.970000
70	0.055458	-0.229284	10.100000
71	0.011912	-0.217372	10.230000
72	-0.072376	-0.289748	10.360000
73	0.113109	-0.176639	10.490000
74	-0.129538	-0.306177	10.620000
75	0.123686	-0.182490	10.750000
76	-0.101376	-0.283866	10.880000
77	0.069568	-0.214298	11.010000
78	-0.034523	-0.248820	11.140000
79	0.000870	-0.247950	11.270000
80	0.028573	-0.219377	11.400000
81	-0.052545	-0.271922	11.530000
82	0.070893	-0.201029	11.660000
83	-0.084114	-0.285143	11.790000
84	0.092964	-0.192179	11.920000
85	-0.098185	-0.290364	12.050000
86	0.100325	-0.190039	12.180000
87	-0.099635	-0.289674	12.310000
88	0.096031	-0.193642	12.440000
89	-0.089110	-0.282752	12.570000
90	0.078211	-0.204542	12.700000
91	-0.062556	-0.267097	12.830000
92	0.041475	-0.225623	12.960000
93	-0.014737	-0.240360	13.090000
94	-0.017020	-0.257380	13.220000
95	0.051820	-0.205560	13.350000
96	-0.085966	-0.291525	13.480000
97	0.113962	-0.177563	13.610000

```
98    -0.129036      -0.306598           13.740000
99    0.124496       -0.182103           13.870000
100   -0.096023      -0.278126           14.000000
```

[13]:
```
#Activity 3

import matplotlib.pyplot as plt

# function to create a table of values showing dY, accumulated dY, and dT for␣
↪each step in dT
def PLOT(Tinit, Tfinal, steps, func):
    dT = (Tfinal - Tinit) / steps  # defining dT
    t = Tinit  # initializing t with Tinit parameter
    acc = 0  # initializing accumulation variable to keep track of total␣
↪accumulation after each step

    #initiating lists to store values for plotting
    Tvals = []  #values for time
    Avals = []  #values for accumulated dY

    #for loop to calculate values at each step
    for k in range(1, steps + 1):
        dY = func(t) * dT  # finding change in Y per change in time, given␣
↪function f
        acc = acc + dY  # summing change in Y to get total accumulated
        t = t + dT  # updates variable t to current Time step

        # Store values for plotting
        Tvals.append(t)
        Avals.append(acc)

    #plotting the accumulation over time
    plt.plot(Tvals, Avals)
    plt.title('Accumulated dY over Time')
    plt.xlabel('Time (t)')
    plt.ylabel('Accumulated dY')
    plt.grid(True)
    plt.show()
```

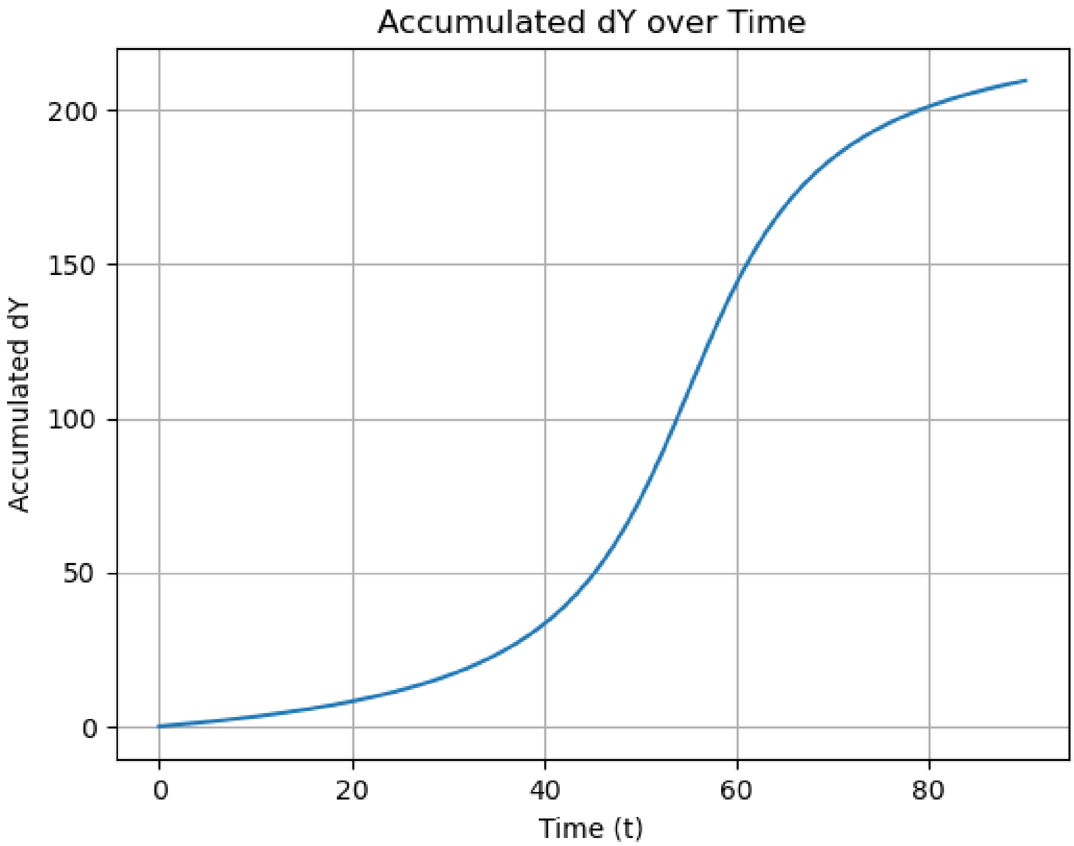
[14]:
```
PLOT(1, 14, 5000, f)
```



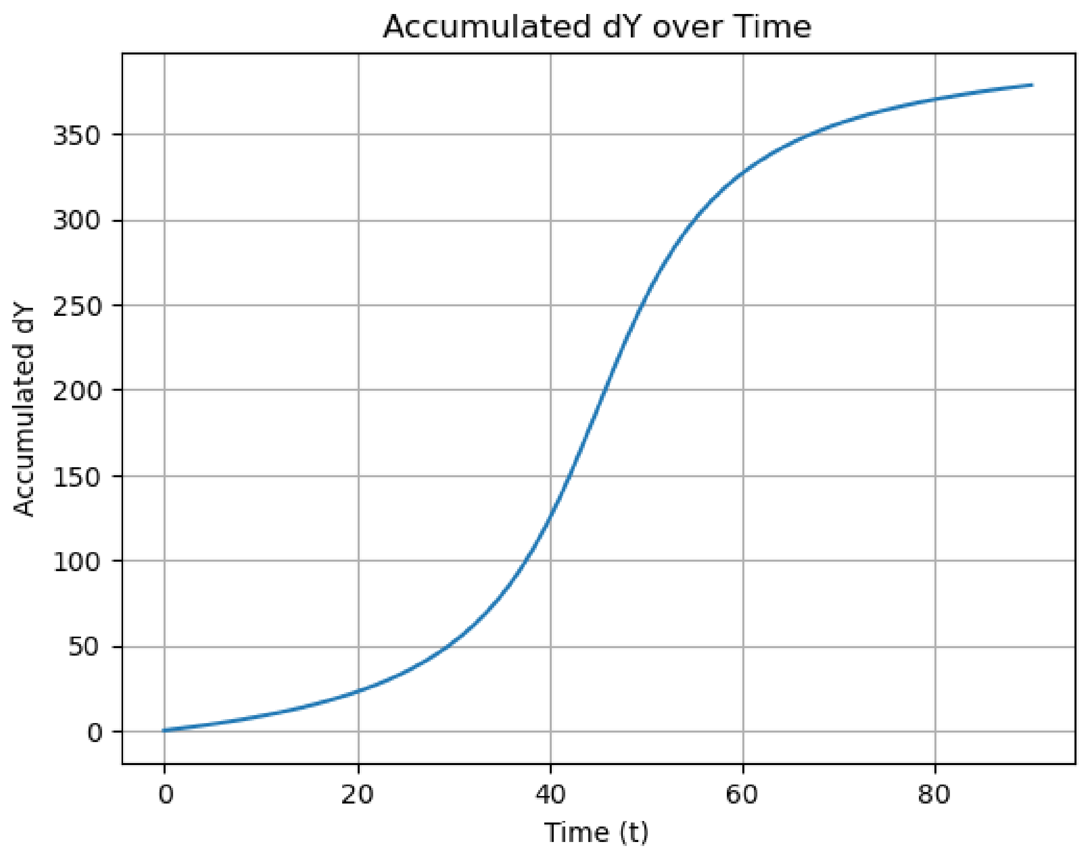
[15]:
```
def A(x):
    r = 827 / ((10.5)**2 + (x - 55)**2)
    return r

def B(x):
    r = 1400 / ((10)**2 + (x - 45)**2)
    return r

def C(x):
    r = 1200 / ((10)**2 + (x - 45)**2)
    return r

def D(x):
    r = 827 / ((10.5)**2 + (x - 44.4)**2)
    return r

def E(x):
    r = 1000 / ((10.5)**2 + (x - 48)**2)
    return r
```

```python
def F(x):
    r = 827 / ((10.5)**2 + (x - 48)**2)
    return r
```

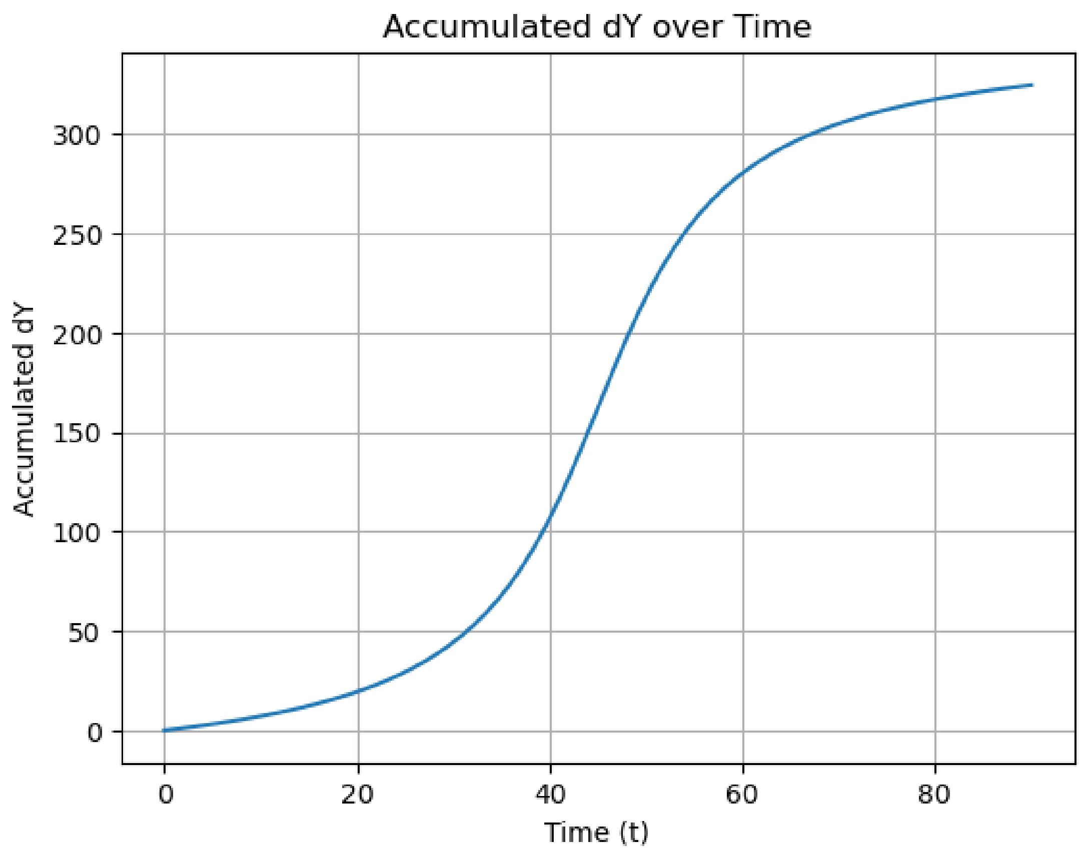
[16]:
```python
PLOT(0, 90, 10000, A)
```



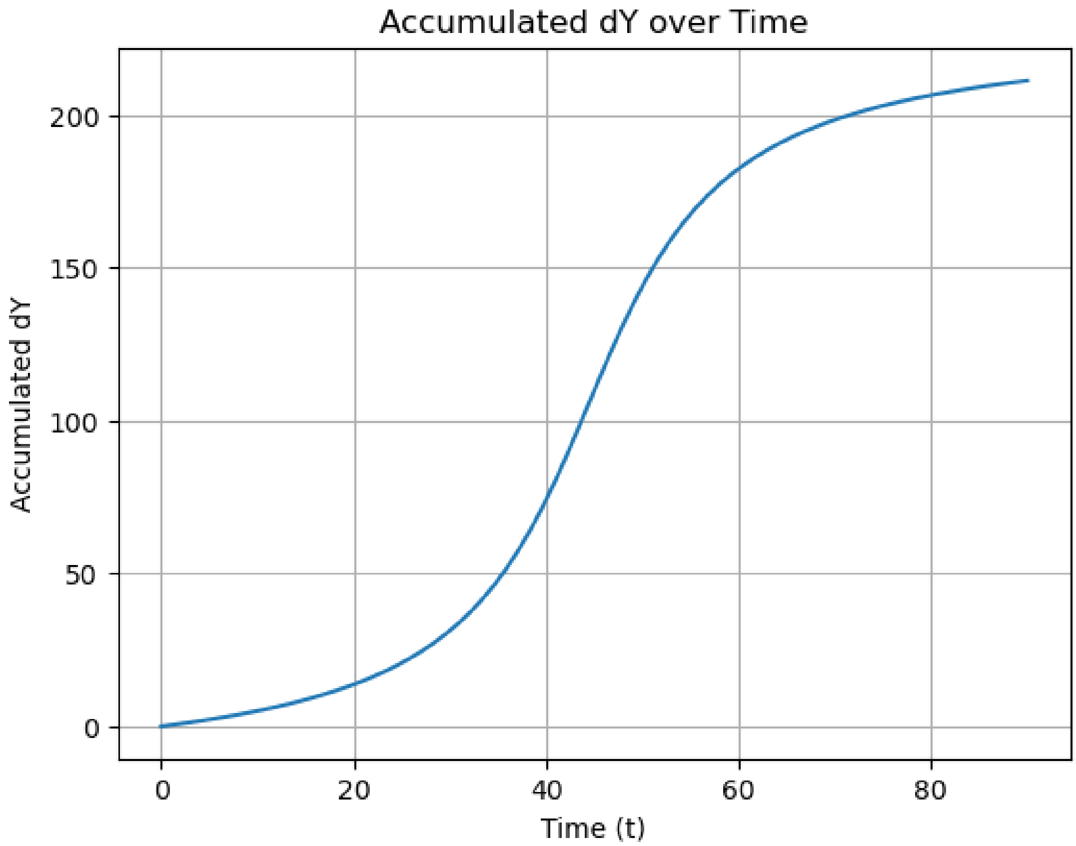
[17]:
```python
PLOT(0, 90, 10000, B)
```



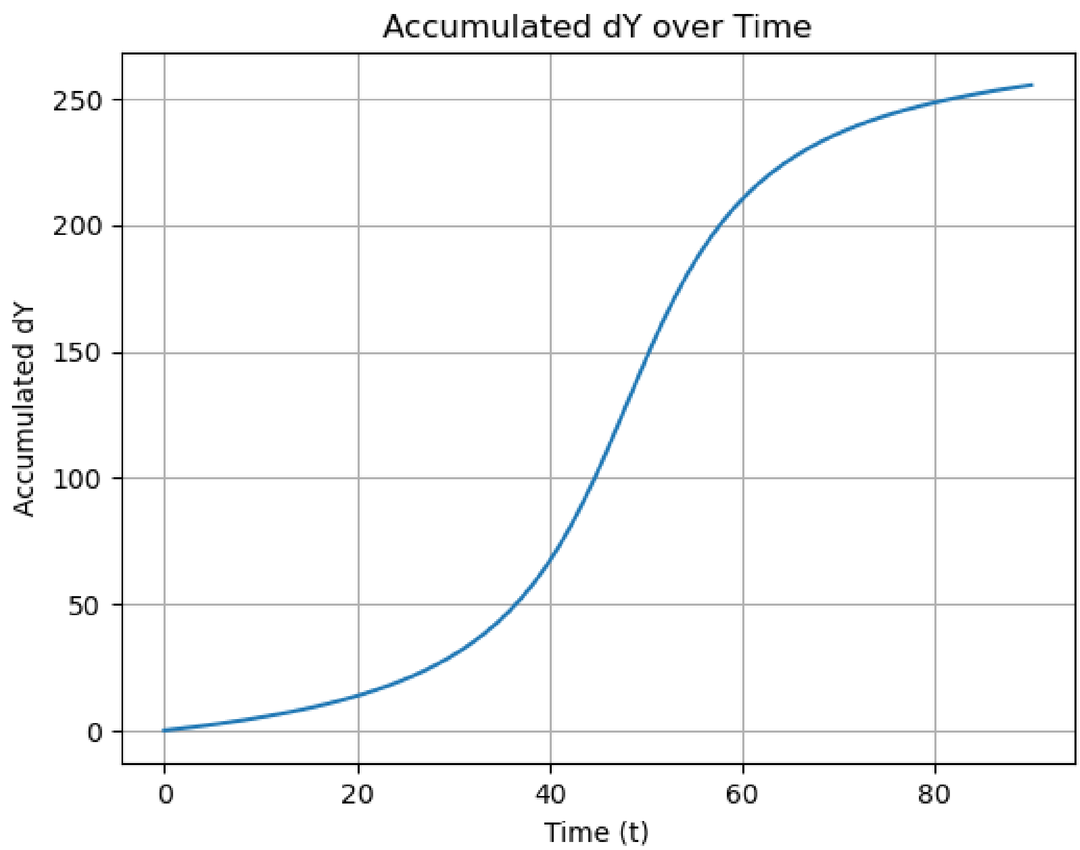
[18]:
```
PLOT(0, 90, 10000, C)
```



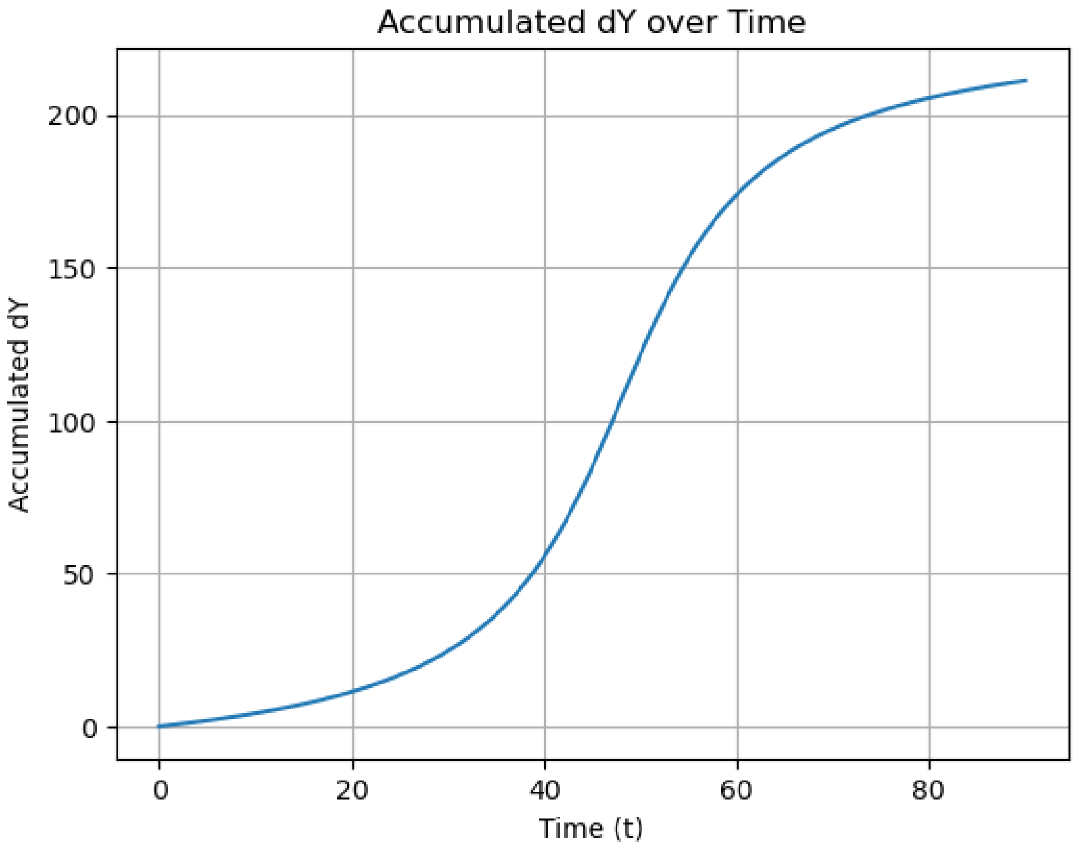
[19]:
```
PLOT(0, 90, 10000, D)
```



[20]:
```
PLOT(0, 90, 10000, E)
```



[21]:
```
PLOT(0, 90, 10000, F)
```



Accumulated dY over Time
Accumulated dY
Time (t)
0
50
100
150
200
0
20
40
60
80


```
[ ]:
```