Week 7

October 15, 2024

1 Week 7 Coding activities

10/17/2024 Brooke Neupert

1.1 Code Program: TABLE from page 266 in Python and add useful code comments, including a summary explaining what it does to someone who does not know. Include a simple demo.

[14]:
```
import numpy as np
import matplotlib.pyplot as plt

def func(t):
    return np.cos(t**2)
'''
Table(fx, ti, tf, n)

inputs: fx (function), ti (initial time, int), tf (final time, int), n(number␣
 ↪of steps, int)
output: none, print statement.

This function takes in a function fx and time parameters to use Euler's method␣
 ↪to solve the differential equation dy/dt=fx(t).
the time step is set by dividing the number of time units (tf-ti) by the number␣
 ↪of desired steps (n). For each timestep,
the equation finds dy/dt for that timestep, giving us the change in y and the␣
 ↪change in t. For each timestep, the change in y,
the accumulated changes in y (sum of all obtained y values) and the ending␣
 ↪timestamp t.

The function prints out a table describing the change in y (dy), accumulated y␣
 ↪(acc.y) and ending time (t).
'''

def Table(fx, ti, tf, n):
    deltat = (tf - ti)/n #divides # of time units by # of desired steps n.␣
 ↪Increment by dt each loop
    t = ti
```

```
    acc = 0 #initialize accumulator to 0

    #I'm adding in a housekeeping line to distinguish columns
    print('dy        | acc. y   |  t        |')

    #this loop runs for n steps of length deltat, and calculates dy and acc.y␣
↪at each t before printing them.
    for k in range(n):
        dy = fx(t) * deltat #make sure fx only has 1 input variable
        acc = acc + dy #accumulator changes by dy each timestep
        t += deltat
        print(f'  {dy:.2f}   |   {acc:.2f}   |   {t:.2f}   |')
```

An example of the scenario described on page 266 is shown below. The function used is

$$\frac{dy}{dt} = \cos t^2$$

. As seen in the table, 4 time units divided by 8 steps calculates the value of dy and accumulated y every 0.5 time units from 0 to 4.

[18]:
```
Table(func, 0, 4, 8)
```

```
dy        | acc. y   |  t        |
  0.50    |   0.50   |   0.50    |
  0.48    |   0.98   |   1.00    |
  0.27    |   1.25   |   1.50    |
  -0.31    |   0.94   |   2.00    |
  -0.33    |   0.61   |   2.50    |
  0.50    |   1.11   |   3.00    |
  -0.46    |   0.66   |   3.50    |
  0.48    |   1.13   |   4.00    |
```

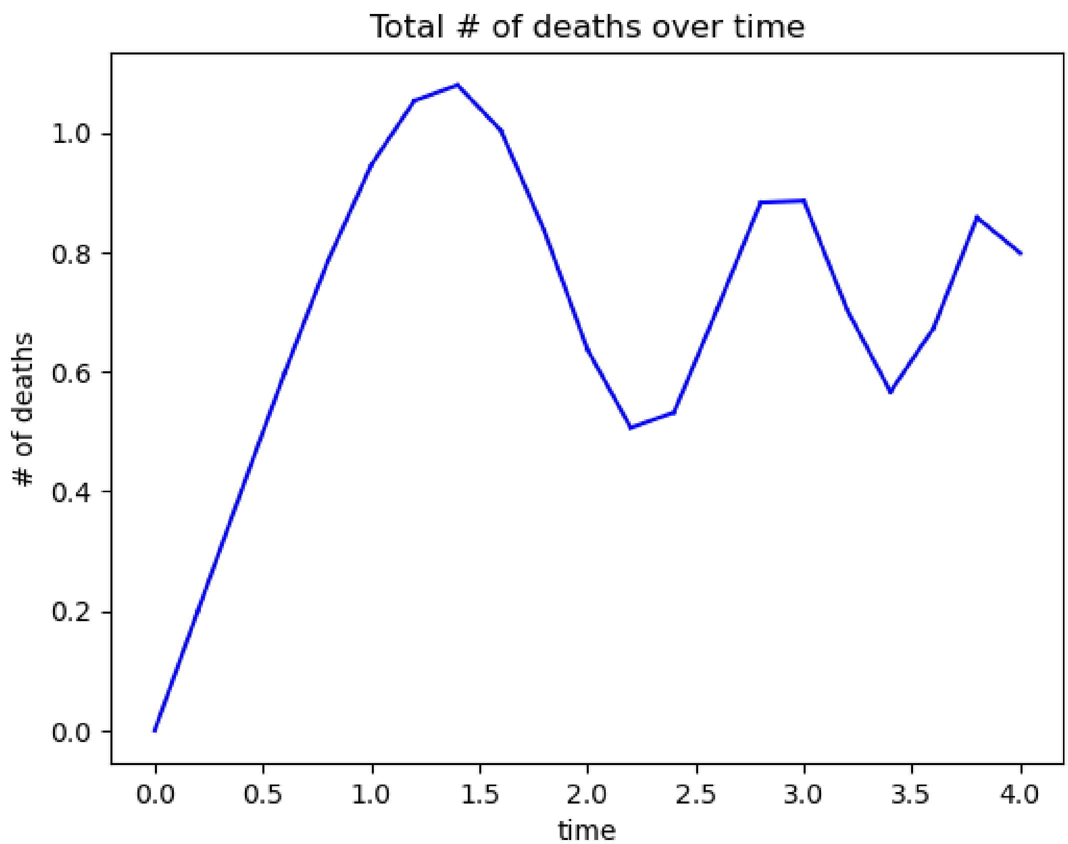
1.2 Activity 3: Code Program: PLOT from page 267 in Python and add useful code comments, including a summary explaining what it does to someone who does not know. Use PLOT to graph all six of the accumulation functions given on page 4 of the Ebola Activity.

[45]:
```
'''
PlotTable(fx, ti, tf, n)

inputs: fx (function), ti (initial time, int), tf (final time, int), n(number␣
↪of steps, int)
output: no return value, plots segments.

PlotTable is an extension of the Table function above, which uses Euler's␣
↪method to calculate the value of dy/dt = f(t) for
```

```
a range of values (t). The number of timesteps is determined by the parameters␣
 ↪ti, tf, and n, with the length of a single
t interval being (tf-ti)/n. Rather than printing the values found by␣
 ↪calculating dy/dt, this function plots the accumulated y
as the loop iterates over each time segment. A higher n value will give a␣
 ↪smoother curve.
'''

def PlotTable(fx, ti, tf, n):
    deltat = (tf - ti)/n #divides # of time units by # of desired steps n.␣
 ↪Increment by dt each loop
    t = ti
    acc = 0 #initialize accumulator to 0

    fig, ax = plt.subplots() #initialize axes

    #this loop runs for n steps of length deltat, and calculates dy and acc.y␣
 ↪at each t before printing them.
    for k in range(n):
        dy = fx(t) * deltat #make sure fx only has 1 input variable
        ax.plot([t, t+deltat], [acc, acc+dy], color = 'blue') #plot a single␣
 ↪line segment (t, acc) to (t+dt, acc+dy)
        acc = acc + dy #accumulator changes by dy each timestep
        t = t + deltat

    #finally, format plot (title and axis labels are for part 2
    ax.set_xlabel("time")
    ax.set_ylabel("# of deaths")
    ax.set_title("Total # of deaths over time")
    plt.show()
```

[46]:

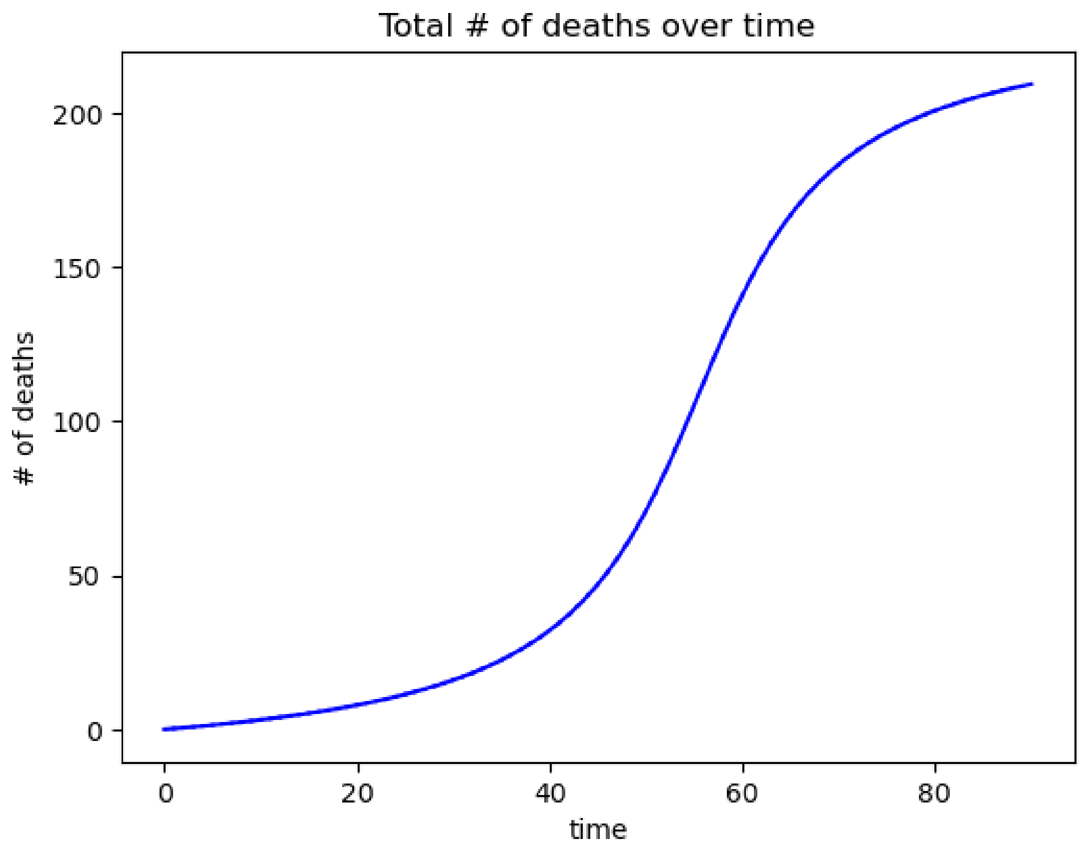
```
PlotTable(func, 0, 4, 20) #demo using same values as the demo for Table()
```

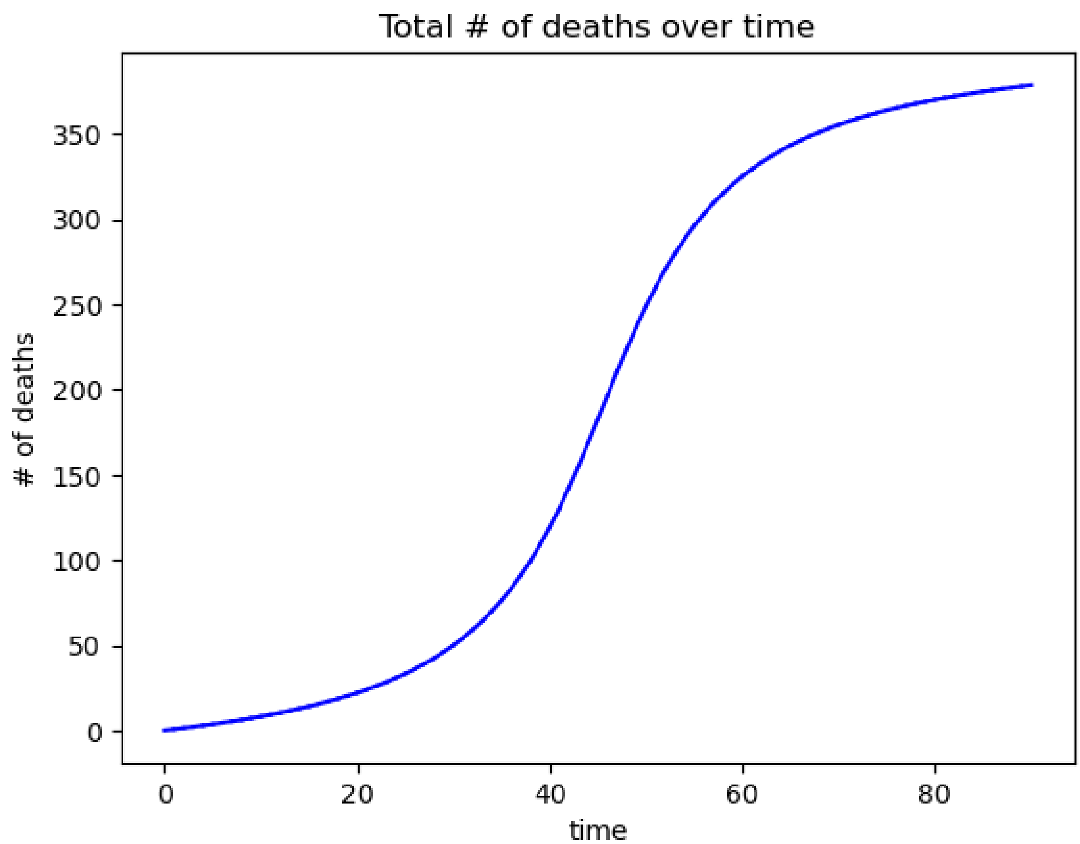


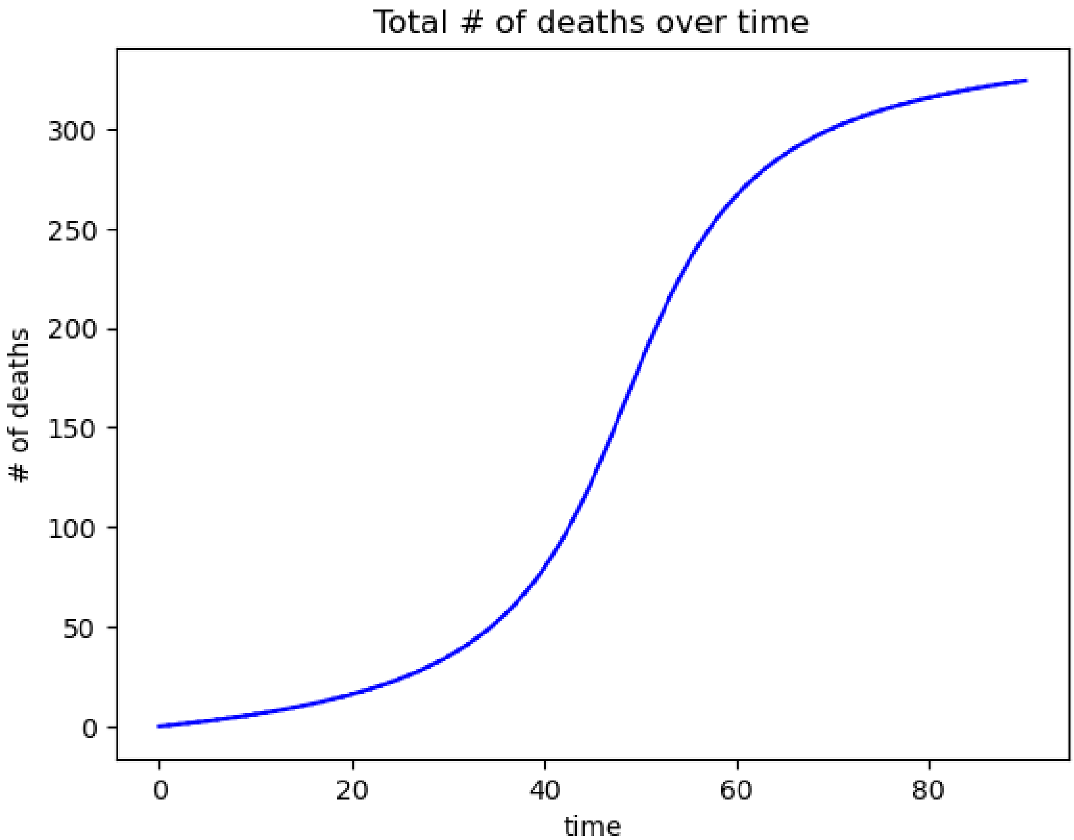
Next, plotting the accumulation functions:

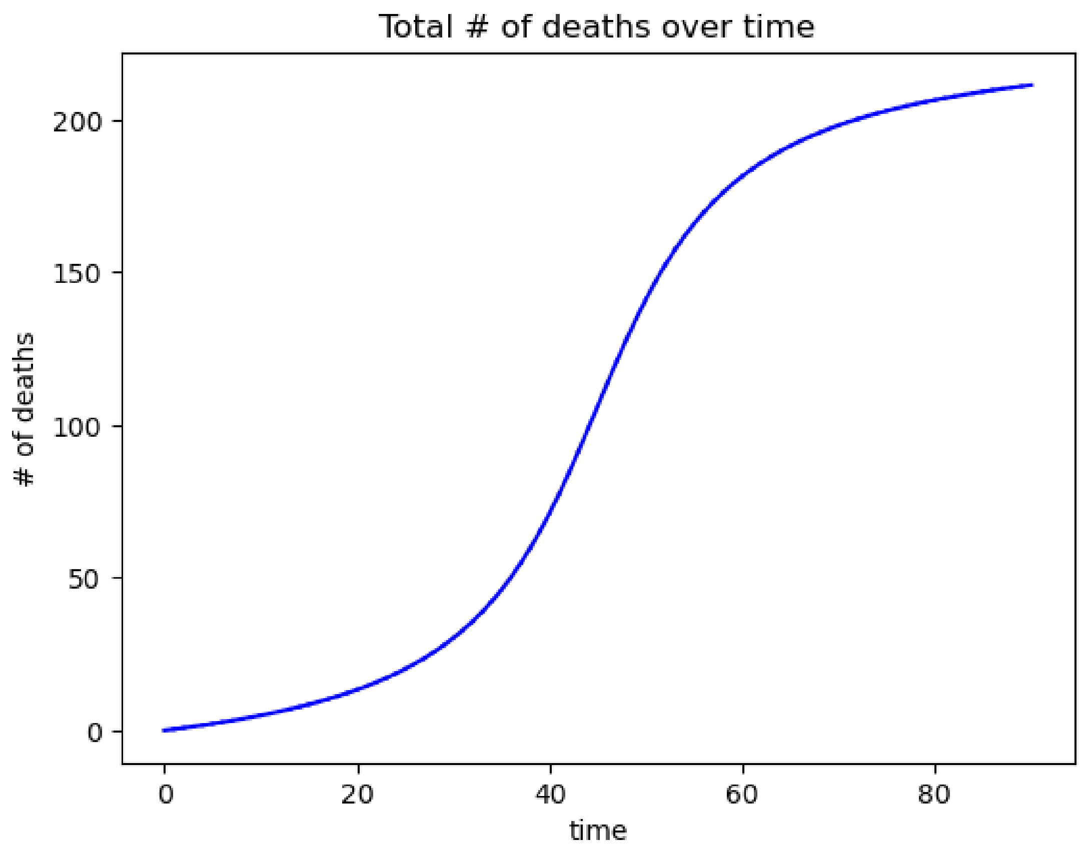
[51]:
```
def ebola_fa(t, a=827, b=10.5, c=55):
    return a/((b**2) + (t-c)**2)
def ebola_fb(t, a=1400, b=10, c=45):
    return a/((b**2) + (t-c)**2)
def ebola_fc(t, a=1200, b=10, c=48):
    return a/((b**2) + (t-c)**2)
def ebola_fd(t, a=827, b=10.5, c=44.4):
    return a/((b**2) + (t-c)**2)
def ebola_fe(t, a=1000, b=10.5, c=48):
    return a/((b**2) + (t-c)**2)
def ebola_ff(t, a=827, b=10.5, c=48):
    return a/((b**2) + (t-c)**2)

PlotTable(ebola_fa, 0, 90, 90)
PlotTable(ebola_fb, 0, 90, 90)
PlotTable(ebola_fc, 0, 90, 90)
PlotTable(ebola_fd, 0, 90, 90)
PlotTable(ebola_fe, 0, 90, 90)
```

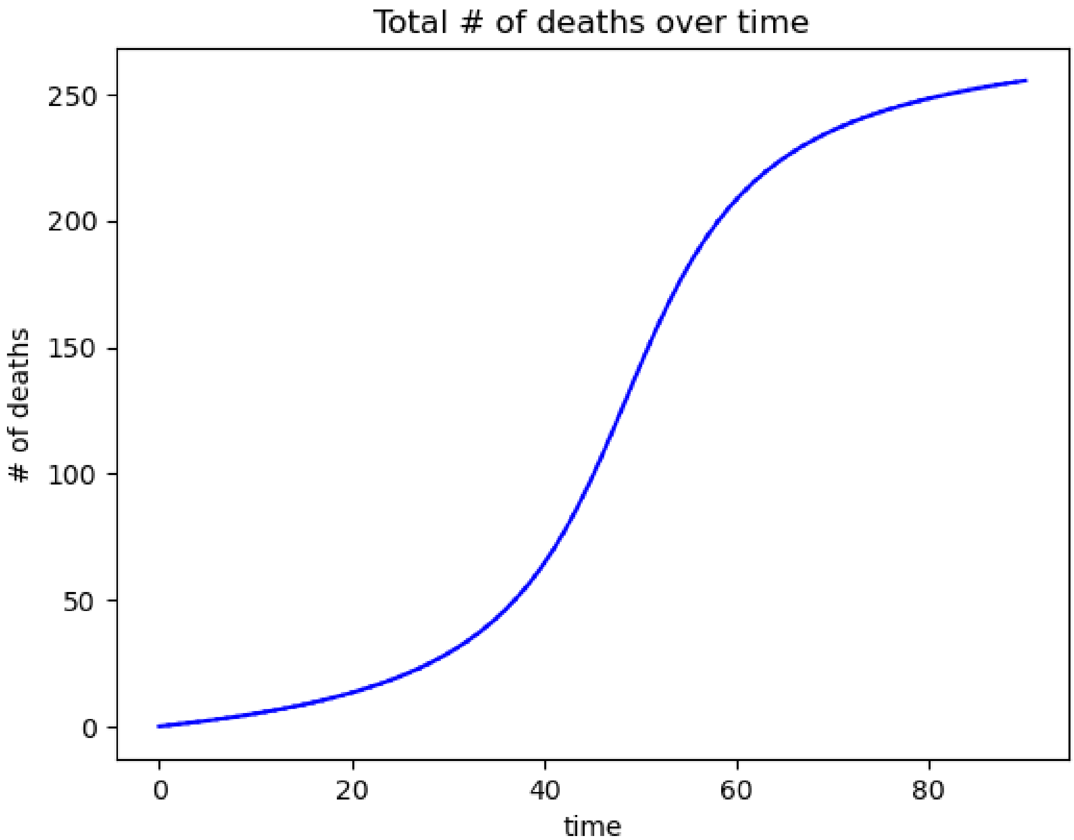
```
PlotTable(ebola_ff, 0, 90, 90)
```

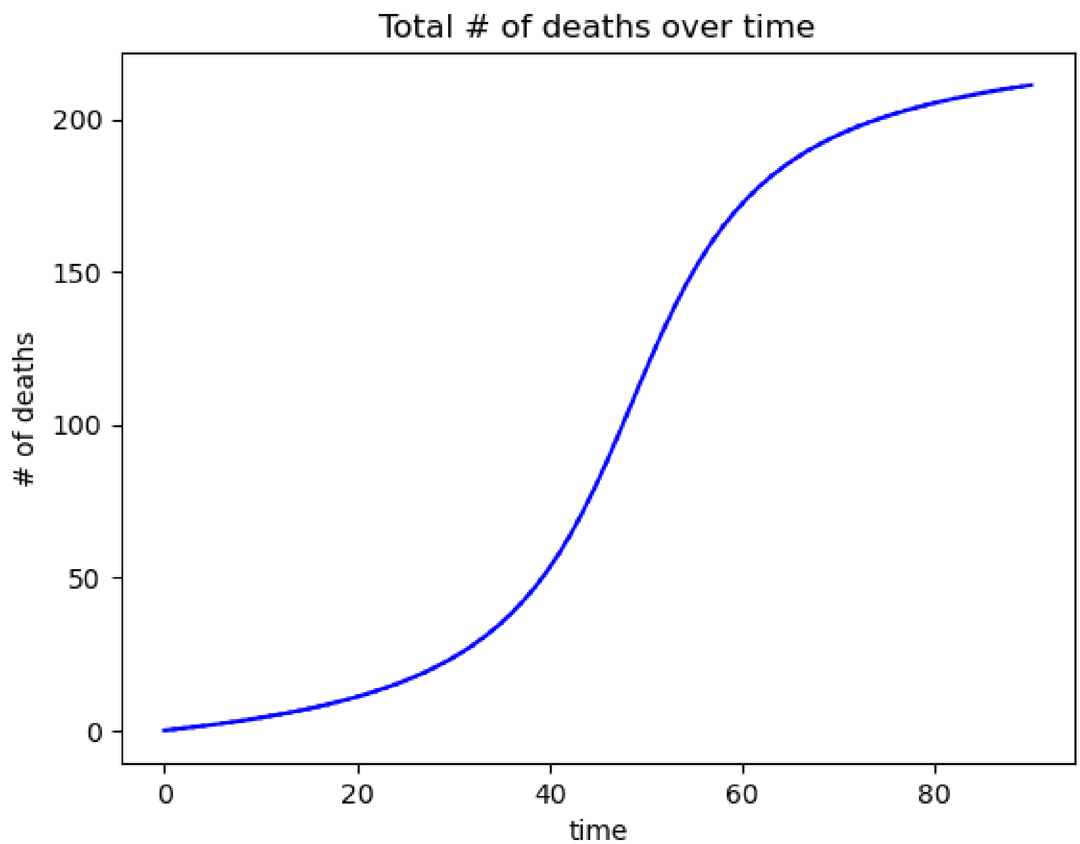



Total # of deaths over time
# of deaths
time
0
50
100
150
200
250
300
350
0
20
40
60
80


Total # of deaths over time
# of deaths
0
50
100
150
200
250
300
0
20
40
60
80
time


Total # of deaths over time
# of deaths
time
0
50
100
150
200
0
20
40
60
80


Total # of deaths over time
# of deaths
time
250
200
150
100
50
0
0
20
40
60
80



Total # of deaths over time
# of deaths
200
150
100
50
0
0
20
40
60
80
time


```
[ ]:
```