

ECE60146 Homework 1

Leonard Fan

January 2025

1 Tasks

1.1 1

The task asks us to create a BioModel class with a sequence instance variable. The code given to us in the assignment is as follows:

```
1 # Create class BioModel
2 class BioModel(object):
3     def __init__(self, sequence):
4         self.sequence = sequence # Class has input sequence
```

1.2 2

The task asks us to extend the BioModel class into an ExponentialGrowthModel subclass. The `__init__` method is defined with two parameters: `start` and `rate`, which serve as the start and the rate of the growth model.

```
1 # Create class ExponentialGrowthModel
2 class ExponentialGrowthModel(BioModel):
3     def __init__(self, start, rate):
4         super().__init__(sequence=[])
5         self.start = start # Class has input start
6         self.rate = rate   # Class has input rate
```

To make ExponentialGrowthModel a subclass of BioModel, we instead use BioModel in the class definition. In the `__init__` method, we define the two input parameters `start` and `rate`. The values are stored in `self.start` and `self.rate`.

1.3 3

The task is to expand `ExponentialGrowthModel` further to make its instances callable. It will include a method with a parameter `length`. A sequence of values is calculated by:

$$\text{next value} = \text{current value} \times (1 + \text{rate})$$

The instance should also print the computed sequence.

```
1 # Create class ExponentialGrowthModel
2 class ExponentialGrowthModel(BioModel):
3     def __init__(self, start, rate):
4         super().__init__(sequence=[])
5         self.start = start # Class has input start
6         self.rate = rate   # Class has input rate
7
8     # Task 3
9     # Callable method
10    def __call__(self, length):
11        self.sequence = [self.start] # Sequence begins with
12                                     # start parameter
13        curr_val = self.start        # Calculation begins
14                                     # with start parameter
15
16        # Loop through and calculate sequence
17        for _ in range(1, length):
18            next_val = curr_val * (1 + self.rate) #
19            # Calculate next value
20            self.sequence.append(next_val)        # Add next
21            # value to sequence
22            curr_val = next_val                    # Update
23            # current value to calculated value
24
25        print(self.sequence) # Print final sequence
26
27    # Define __len__ method to print length when called
28    def __len__(self):
29        return len(self.sequence)
```

The `__call__` method will run a program when the instance is called. We start with the sequence with the given `start` parameter to calculate the sequence. The variable `curr_val` is iterated through and updated using the equation defined previously. Each new calculated value is appended to the parameter `self.sequence`. Lastly, the final sequence is printed.

The example shows the `len()` function called on the instance. We must add the

`__len__` method. This method returns the length of the sequence when called.

The output is tested with the example:

```
1 GM = ExponentialGrowthModel(start=100, rate=0.1)
2 GM(length=5) # Output: [100, 110.0, 121.0, 133.1, 146.41]
3 print(len(GM)) # Output: 5
```

The output is as follows:

```
1 [100, 110.00, 121.00, 133.10, 146.41]
2 5
```

Which matches the expected output in the example.

1.4 4

The task is to modify `BioModel` to be used as an integrator.

```
1 # Define iter method to start iterable
2 def __iter__(self):
3     self.index = 0 # Start index at 0
4     return self
5
6 # Define next method to continue iterable
7 def __next__(self):
8     # Logic to determine when to stop iteration
9     if self.index < len(self.sequence):
10         self.index += 1
11         return self.sequence[self.index - 1] # Subtract one
12         # from index for accurate output
13     else:
14         raise StopIteration # Stop iteration if index is not
15         # less than length
```

To make an instance iterable, we need to define the `__iter__` and `__next__` methods. `__iter__` method defines the start of the iterable. We start with the zero index to begin the sequence. `__next__` method defines the iteration pattern. If the index is less than the length of the sequence, we increment the index. Otherwise, we stop the iterable and end iteration.

The output is tested with the example:

```
1 print([n for n in GM]) # Output: [100, 110.0, 121.0, 133.1,
146.41]
```

The output is as follows:

```
1 [100, 110.00, 121.00, 133.10, 146.41]
```

Which follows the expected output.

1.5 5

The task is to simulate a decay model. The class is identical except for the decay model. The formula for the decay model is:

$$\text{next value} = \text{current value} \times (1 - \text{rate})$$

```
1 # Create class ExponentialDecayModel
2 class ExponentialDecayModel(BioModel):
3     def __init__(self, start, rate):
4         super().__init__(sequence=[])
5         self.start = start # Class has input start
6         self.rate = rate # Class has input rate
7
8     # Callable method
9     def __call__(self, length):
10        self.sequence = [self.start] # Sequences begins with
11        start parameter
12        curr_val = self.start # Calculation begins with
13        start parameter
14
15        # Loop through and calculate sequence
16        for _ in range(1, length):
17            next_val = curr_val * (1 - self.rate) #
18            Calculate next value
19            self.sequence.append(next_val) # Add next value
20            to sequence
21            curr_val = next_val # Update current value to
22            calculated value
23
24        print(self.sequence) # Print final sequence
25
26    # Define __len__ method to print length when called
27    def __len__(self):
28        return len(self.sequence)
```

Most of the `ExponentialDecayModel` subclass is the same as `ExponentialGrowthModel`. The only difference is the calculation in the `__call__` method. The formula is adjusted with the previously mentioned equation.

The output is tested with the example:

```

1 DM = ExponentialDecayModel(start=100, rate=0.2)
2 DM(length=5) # Output: [100, 80.0, 64.0, 51.2, 40.96]
3 print(len(DM)) # Output: 5
4 print([n for n in DM]) # Output: [100, 80.0, 64.0, 51.2,
    40.96]

```

The output is as follows:

```

1 [100, 80.0, 64.0, 51.2, 40.96]
2 5
3 [100, 80.0, 64.0, 51.2, 40.96]

```

Which follows the expected output.

1.6 6

The task is to modify `BioModel` to allow for comparison between two instances using the `==` operator. If the sequences are the same length, they should be compared element-wise and return the count of matching elements. Otherwise raise a `ValueError`

```

1     def __eq__(self, other):
2         if not isinstance(other, BioModel):
3             return NotImplemented
4         if len(self.sequence) != len(other.sequence):
5             raise ValueError("Two arrays are not equal in
                length!")
6
7         count_match = 0
8         for x, y in zip(self.sequence, other.sequence):
9             if x == y:
10                 count_match += 1
11
12         return count_match

```

The method `__eq__` defines what an instance should do when compared. When called, it compares to the "other" instance. We cannot compare the two if the other instance is not of type `BioModel`. Then, it compares the length for both sequences. If they are not equal, then an error `ValueError` is raised. If they are equal, each element will be compared pairwise. A counter `count_match` keeps track of the number of matching elements. Lastly, the value is returned once all values are compared.

The output is tested with the example:

```

1 print("Task_6_result")
2 GM = ExponentialGrowthModel(start=100, rate=0.1)

```

```

3 GM(length=5) # [100, 110.0, 121.0, 133.1, 146.41]
4
5 GM2 = ExponentialGrowthModel(start=100, rate=0.2)
6 GM2(length=5) # [100, 120.0, 144.0, 172.8, 207.36]
7
8 print(GM == GM2) # Output: 1 (only the first element
   matches)
9
10 GM3 = ExponentialGrowthModel(start=100, rate=0.2)
11 GM3(length=3) # [100, 120.0, 144.0]
12
13 print(GM == GM3) # Raises ValueError: Two arrays are not
   equal in length!

```

The output is as follows:

```

1 [100, 110.0, 121.0, 133.10, 146.41]
2 [100, 120.0, 144.0, 172.8, 207.36]
3 1
4 [100, 120.0, 144.0]
5 Traceback (most recent call last):
6   File "c:\Users\Fanle\OneDrive\Desktop\Code\ECE60146\
   hw1_tasks.py", line 134, in <module>
7     print(GM == GM3) # Raises ValueError: Two arrays are
   not equal in length!
   ~~~~~
8
9   File "c:\Users\Fanle\OneDrive\Desktop\Code\ECE60146\
   hw1_tasks.py", line 29, in __eq__
10     raise ValueError("Two arrays are not equal in length!")
11 ValueError: Two arrays are not equal in length!

```

Which follows the expected output.

1.6.1 7

This task asks us to choose our **start** and **rate** parameter values and then show the results.

The values were chosen randomly by a random number generator. The **start** value for all tasks ranges from 100 to 999. The generator chose 285. Each **rate** is randomly selected from 0.1 to 0.9 except for the **GM** and **GM2** instances in example 6. They will be set to the same to show the count matches correctly.

The output is tested with the example:

```

1 # Task 3 result
2 print("Task_3_result")
3 GM = ExponentialGrowthModel(start=285, rate=0.9)

```

```

4 GM(length=5)
5 print(len(GM)) # Output: 5
6
7 # Task 4 Result
8 print("Task_4_result")
9 print([n for n in GM])
10
11 # Task 5 Result
12 DM = ExponentialDecayModel(start=285, rate=0.4)
13 DM(length=5)
14 print(len(DM)) # Output: 5
15 print([n for n in DM])
16
17 # Task 6 Result
18 GM = ExponentialGrowthModel(start=285, rate=0.3)
19 GM(length=5)
20
21 GM2 = ExponentialGrowthModel(start=285, rate=0.3)
22 GM2(length=5)
23
24 print(GM == GM2) # Output: 5 (All output matches)
25
26 GM3 = ExponentialGrowthModel(start=285, rate=0.1)
27 GM3(length=3) # [100, 120.0, 144.0]
28
29 print(GM == GM3) # Raises ValueError: Two arrays are not
    equal in length!

```

The output is as follows:

```

1 Task 3 result
2 [285, 541.5, 1028.85, 1954.8149999999998,
   3714.1484999999993]
3 5
4 Task 4 result
5 [285, 541.5, 1028.85, 1954.8149999999998,
   3714.1484999999993]
6 Task 5 result
7 [285, 171.0, 102.6, 61.559999999999995, 36.935999999999999]
8 5
9 [285, 171.0, 102.6, 61.559999999999995, 36.935999999999999]
10 Task 6 result
11 [285, 370.5, 481.65000000000003, 626.1450000000001,
   813.9885000000002]
12 [285, 370.5, 481.65000000000003, 626.1450000000001,
   813.9885000000002]
13 5
14 [285, 313.5, 344.85]
15 Traceback (most recent call last):
16   File "c:\Users\Fanle\OneDrive\Desktop\Code\ECE60146\

```

```

    hw1_choice_parameters.py", line 115, in <module>
17     print(GM == GM3) # Raises ValueError: Two arrays are
        not equal in length!
18         ~~~~~
19     File "c:\Users\Fanle\OneDrive\Desktop\Code\ECE60146\
        hw1_choice_parameters.py", line 29, in __eq__
20         raise ValueError("Two arrays are not equal in length!")
21 ValueError: Two arrays are not equal in length!

```

Which follows the expected output.

2 Bonus

2.1 1

This task asks us to create a new subclass called `CombinedBioModel`. It has the following requirements:

1. Generate a growth sequence using `ExponentialGrowthModel`
2. Generate a decay sequence using `ExponentialDecayModel`
3. Combine these sequences by multiplying the corresponding values from both sequences element-wise.

```

1 # Create class CombinedBioModel
2 class CombinedBioModel(BioModel):
3     def __init__(self, growth_start, growth_rate,
4                 decay_start, decay_rate):
5         super().__init__(sequence = [])
6         self.growth_model = ExponentialGrowthModel(start=
7             growth_start, rate=growth_rate) # Class has
8             growth model
9         self.decay_model = ExponentialDecayModel(start=
10            decay_start, rate=decay_rate) # Class has decay
11            model
12
13 # Callable method
14 def __call__(self, length):
15     # Initialize growth and decay model with length
16     self.growth_model(length)
17     self.decay_model(length)
18
19     # Multiply elements pairwise from growth and decay
20     model

```

```

15         self.sequence = [g * d for g, d in zip(self.
            growth_model.sequence, self.decay_model.sequence)
16     ]
17     # Print calculated sequence
18     print(self.sequence)

```

The class `CombinedBioModel` initializes a growth model and decay model instances using the input parameters. The `__call__` method receives an input parameter `length`, initializing the two instances. The pairwise product is then calculated using list comprehension.

The output is tested with the example:

```

1 print("Bonus_1_1_result")
2 CBM = CombinedBioModel(growth_start=100, growth_rate=0.1,
    decay_start=1.0, decay_rate=0.05)
3 CBM(length=5) # Output: [100.0, 104.5, 109.2, 114.12,
    119.25]

```

The output is as follows:

```

1 [100, 110.00, 121.00, 133.10, 146.41]
2 [1.0, 0.95, 0.9025, 0.86, 0.81]
3 [100.0, 104.50, 109.2, 114.12, 119.25]

```

Which follows the expected output.

2.2 2

This task asks us to visualize the comparison between two growth and decay rates.

```

1 # Two growth and decay rates
2 # Length is set to 10 for visualization
3 # Growth and decay set at same start point to visualize
    difference
4 growth_rate_1 = 0.1
5 growth_rate_2 = 0.5
6 decay_rate_1 = 0.02
7 decay_rate_2 = 0.1
8 length = 10
9 growth_start = 10
10 decay_start = 10
11
12 # Combination of Growth Rates
13 print("Growth_rate_1, Decay_rate_1")

```

```

14 combined_model_1 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_1, decay_start=
    decay_start, decay_rate=decay_rate_1)
15 combined_model_1(length)
16 growth_model_1 = combined_model_1.growth_model
17 decay_model_1 = combined_model_1.decay_model
18
19 print("Growth_rate_2, Decay_rate_2")
20 combined_model_2 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_2, decay_start=
    decay_start, decay_rate=decay_rate_2)
21 combined_model_2(length)
22 growth_model_2 = combined_model_2.growth_model
23 decay_model_2 = combined_model_2.decay_model
24
25 print("Growth_rate_2, Decay_rate_1")
26 combined_model_3 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_2, decay_start=
    decay_start, decay_rate=decay_rate_1)
27 combined_model_3(length)
28
29 print("Growth_rate_1, Decay_rate_2")
30 combined_model_4 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_1, decay_start=
    decay_start, decay_rate=decay_rate_2)
31 combined_model_4(length)
32
33 fig, axs = plt.subplots(2, 2, figsize=(12, 10))
34 fig.suptitle("Growth, Decay, and Combined Dynamics",
    fontsize=16)
35
36 # Plot subplots
37 axs[0, 0].plot(range(length), growth_model_1.sequence, label
    ="Growth", color="blue", linestyle="solid")
38 axs[0, 0].plot(range(length), decay_model_1.sequence, label=
    "Decay", color="red", linestyle="dashed")
39 axs[0, 0].plot(range(length), combined_model_1.sequence,
    label="Combined", color="green", linestyle="dotted")
40 axs[0, 0].set_title(f"Growth_Rate: {growth_rate_1}, Decay
    Rate: {decay_rate_1}")
41 axs[0, 0].set_xlabel("Iteration")
42 axs[0, 0].set_ylabel("Value")
43 axs[0, 0].legend()
44
45 axs[0, 1].plot(range(length), growth_model_2.sequence, label
    ="Growth", color="blue", linestyle="solid")
46 axs[0, 1].plot(range(length), decay_model_1.sequence, label=
    "Decay", color="red", linestyle="dashed")
47 axs[0, 1].plot(range(length), combined_model_3.sequence,
    label="Combined", color="green", linestyle="dotted")

```

```

48  axs[0, 1].set_title(f"Growth_Rate:_{growth_rate_2},_Decay_
    Rate:_{decay_rate_1}")
49  axs[0, 1].set_xlabel("Iteration")
50  axs[0, 1].set_ylabel("Value")
51  axs[0, 1].legend()
52
53  axs[1, 0].plot(range(length), growth_model_1.sequence, label
    ="Growth", color="blue", linestyle="solid")
54  axs[1, 0].plot(range(length), decay_model_2.sequence, label=
    "Decay", color="red", linestyle="dashed")
55  axs[1, 0].plot(range(length), combined_model_4.sequence,
    label="Combined", color="green", linestyle="dotted")
56  axs[1, 0].set_title(f"Growth_Rate:_{growth_rate_1},_Decay_
    Rate:_{decay_rate_2}")
57  axs[1, 0].set_xlabel("Iteration")
58  axs[1, 0].set_ylabel("Value")
59  axs[1, 0].legend()
60
61  axs[1, 1].plot(range(length), growth_model_2.sequence, label
    ="Growth", color="blue", linestyle="solid")
62  axs[1, 1].plot(range(length), decay_model_2.sequence, label=
    "Decay", color="red", linestyle="dashed")
63  axs[1, 1].plot(range(length), combined_model_2.sequence,
    label="Combined", color="green", linestyle="dotted")
64  axs[1, 1].set_title(f"Growth_Rate:_{growth_rate_2},_Decay_
    Rate:_{decay_rate_2}")
65  axs[1, 1].set_xlabel("Iteration")
66  axs[1, 1].set_ylabel("Value")
67  axs[1, 1].legend()
68
69  # Set all y axis equal
70  for ax in axs.flat:
71      ax.set_ylim(0, 200)
72
73  plt.show()

```

Values were chosen at random. The plots are shown in Figure 1.

3 Source Code

The code for all the tasks and bonus is presented below

```

1  hw1_tasks.py
2
3  import matplotlib.pyplot as plt
4
5  # Task 1

```

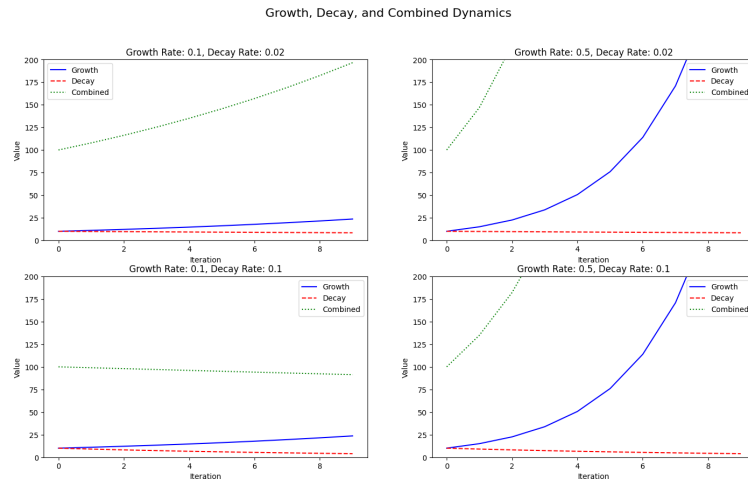


Figure 1: Generated growth, decay and combined sequences for two growth and two decay rates

```

6  # Create class BioModel
7  class BioModel(object):
8      def __init__(self, sequence):
9          self.sequence = sequence # Class has input sequence
10
11     #Task 4
12     # Define iter method to start iterable
13     def __iter__(self):
14         self.index = 0 # Start index at 0
15         return self
16
17     # Define next method to continue iterable
18     def __next__(self):
19         # Logic to determine when to stop iteration
20         if self.index < len(self.sequence):
21             self.index += 1
22             return self.sequence[self.index - 1] # Too lazy
23                                                     to add more lines. Subtract one from index
24                                                     for accurate output
25         else:
26             raise StopIteration # Stop iteration if index is
27                                     not less than length
28
29     # Task 6
30     def __eq__(self, other):
31         if not isinstance(other, BioModel):
32             return NotImplemented

```

```

30         if len(self.sequence) != len(other.sequence):
31             raise ValueError("Two arrays are not equal in
32                                 length!")
33
34         count_match = 0
35         for x, y in zip(self.sequence, other.sequence):
36             if x == y:
37                 count_match += 1
38
39         return count_match
40
41 # Task 2
42 # Create class ExponentialGrowthModel
43 class ExponentialGrowthModel(BioModel):
44     def __init__(self, start, rate):
45         super().__init__(sequence=[])
46         self.start = start # Class has input start
47         self.rate = rate # Class has input rate
48
49 # Task 3
50 # Callable method
51 def __call__(self, length):
52     self.sequence = [self.start] # Sequences begins with
53                                     start parameter
54     curr_val = self.start # Calculation begins with
55                                     start parameter
56
57     # Loop through and calculate sequence
58     for _ in range(1, length):
59         next_val = curr_val * (1 + self.rate) #
60                                     Calculate next value
61         self.sequence.append(next_val) # Add next value
62                                     to sequence
63         curr_val = next_val # Update current value to
64                                     calculated value
65
66     print(self.sequence) # Print final sequence
67
68 # Define __len__ method to print length when called
69 def __len__(self):
70     return len(self.sequence)
71
72 # Task 5
73 # Create class ExponentialDecayModel
74 class ExponentialDecayModel(BioModel):
75     def __init__(self, start, rate):
76         super().__init__(sequence=[])
77         self.start = start # Class has input start
78         self.rate = rate # Class has input rate

```

```

74     # Callable method
75     def __call__(self, length):
76         self.sequence = [self.start] # Sequences begins with
            start parameter
77         curr_val = self.start # Calculation begins with
            start parameter
78
79         # Loop through and calculate sequence
80         for _ in range(1, length):
81             next_val = curr_val * (1 - self.rate) #
                Calculate next value
82             self.sequence.append(next_val) # Add next value
                to sequence
83             curr_val = next_val # Update current value to
                calculated value
84
85         print(self.sequence) # Print final sequence
86
87         # Define __len__ method to print length when called
88         def __len__(self):
89             return len(self.sequence)
90
91     # Bonus 1
92     # Create class CombinedBioModel
93     class CombinedBioModel(BioModel):
94         def __init__(self, growth_start, growth_rate,
            decay_start, decay_rate):
95             super().__init__(sequence = [])
96             self.growth_model = ExponentialGrowthModel(start=
                growth_start, rate=growth_rate) # Class has
                growth model
97             self.decay_model = ExponentialDecayModel(start=
                decay_start, rate=decay_rate) # Class has decay
                model
98
99         # Callable method
100        def __call__(self, length):
101            # Initialize growth and decay model with length
102            self.growth_model(length)
103            self.decay_model(length)
104
105            # Multiply elements pairwise from growth and decay
                model
106            self.sequence = [g * d for g, d in zip(self.
                growth_model.sequence, self.decay_model.sequence)
                ]
107
108            # Print calculated sequence
109            print(self.sequence)
110

```

```

111 # Task 3 result
112 print("Task_3_result")
113 GM = ExponentialGrowthModel(start=100, rate=0.1)
114 GM(length=5) # Output: [100, 110.0, 121.0, 133.1, 146.41]
115 print(len(GM)) # Output: 5
116
117 # Task 4 Result
118 print("Task_4_result")
119 print([n for n in GM]) # Output: [100, 110.0, 121.0, 133.1,
120                               146.41]
121
122 # Task 5 Result
123 print("Task_5_result")
124 DM = ExponentialDecayModel(start=100, rate=0.2)
125 DM(length=5) # Output: [100, 80.0, 64.0, 51.2, 40.96]
126 print(len(DM)) # Output: 5
127 print([n for n in DM]) # Output: [100, 80.0, 64.0, 51.2,
128                               40.96]
129
130 # Task 6 Result
131 print("Task_6_result")
132 GM = ExponentialGrowthModel(start=100, rate=0.1)
133 GM(length=5) # [100, 110.0, 121.0, 133.1, 146.41]
134
135 GM2 = ExponentialGrowthModel(start=100, rate=0.2)
136 GM2(length=5) # [100, 120.0, 144.0, 172.8, 207.36]
137
138 print(GM == GM2) # Output: 1 (only the first element
139                 matches)
140
141 GM3 = ExponentialGrowthModel(start=100, rate=0.2)
142 GM3(length=3) # [100, 120.0, 144.0]
143
144 # Commented out for running bonus outputs
145 # print(GM == GM3) # Raises ValueError: Two arrays are not
146 # equal in length!
147
148 # Bonus 1
149 print("Bonus_1_result")
150 CBM = CombinedBioModel(growth_start=100, growth_rate=0.1,
151                        decay_start=1.0, decay_rate=0.05)
152 CBM(length=5) # Output: [100.0, 104.5, 109.2, 114.12,
153                        119.25]
154
155 # Bonus 2
156 print("Bonus_2_result")
157
158 # Two growth and decay rates
159 # Length is set to 10 for visualization

```

```

154 # Growth and decay set at same start point to visualize
    difference
155 growth_rate_1 = 0.1
156 growth_rate_2 = 0.5
157 decay_rate_1 = 0.02
158 decay_rate_2 = 0.1
159 length = 10
160 growth_start = 10
161 decay_start = 10
162
163 # Combination of Growth Rates
164 print("Growth_rate_1, Decay_rate_1")
165 combined_model_1 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_1, decay_start=
    decay_start, decay_rate=decay_rate_1)
166 combined_model_1(length)
167 growth_model_1 = combined_model_1.growth_model
168 decay_model_1 = combined_model_1.decay_model
169
170 print("Growth_rate_2, Decay_rate_2")
171 combined_model_2 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_2, decay_start=
    decay_start, decay_rate=decay_rate_2)
172 combined_model_2(length)
173 growth_model_2 = combined_model_2.growth_model
174 decay_model_2 = combined_model_2.decay_model
175
176 print("Growth_rate_2, Decay_rate_1")
177 combined_model_3 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_2, decay_start=
    decay_start, decay_rate=decay_rate_1)
178 combined_model_3(length)
179
180 print("Growth_rate_1, Decay_rate_2")
181 combined_model_4 = CombinedBioModel(growth_start=
    growth_start, growth_rate=growth_rate_1, decay_start=
    decay_start, decay_rate=decay_rate_2)
182 combined_model_4(length)
183
184 fig, axs = plt.subplots(2, 2, figsize=(12, 10))
185 fig.suptitle("Growth, Decay, and Combined Dynamics",
    fontsize=16)
186
187 # Plot subplots
188 axs[0, 0].plot(range(length), growth_model_1.sequence, label
    ="Growth", color="blue", linestyle="solid")
189 axs[0, 0].plot(range(length), decay_model_1.sequence, label=
    "Decay", color="red", linestyle="dashed")
190 axs[0, 0].plot(range(length), combined_model_1.sequence,
    label="Combined", color="green", linestyle="dotted")

```

```

191  axs[0, 0].set_title(f"Growth_Rate:_{growth_rate_1},_Decay_
      Rate:_{decay_rate_1}")
192  axs[0, 0].set_xlabel("Iteration")
193  axs[0, 0].set_ylabel("Value")
194  axs[0, 0].legend()
195
196  axs[0, 1].plot(range(length), growth_model_2.sequence, label
      ="Growth", color="blue", linestyle="solid")
197  axs[0, 1].plot(range(length), decay_model_1.sequence, label=
      "Decay", color="red", linestyle="dashed")
198  axs[0, 1].plot(range(length), combined_model_3.sequence,
      label="Combined", color="green", linestyle="dotted")
199  axs[0, 1].set_title(f"Growth_Rate:_{growth_rate_2},_Decay_
      Rate:_{decay_rate_1}")
200  axs[0, 1].set_xlabel("Iteration")
201  axs[0, 1].set_ylabel("Value")
202  axs[0, 1].legend()
203
204  axs[1, 0].plot(range(length), growth_model_1.sequence, label
      ="Growth", color="blue", linestyle="solid")
205  axs[1, 0].plot(range(length), decay_model_2.sequence, label=
      "Decay", color="red", linestyle="dashed")
206  axs[1, 0].plot(range(length), combined_model_4.sequence,
      label="Combined", color="green", linestyle="dotted")
207  axs[1, 0].set_title(f"Growth_Rate:_{growth_rate_1},_Decay_
      Rate:_{decay_rate_2}")
208  axs[1, 0].set_xlabel("Iteration")
209  axs[1, 0].set_ylabel("Value")
210  axs[1, 0].legend()
211
212  axs[1, 1].plot(range(length), growth_model_2.sequence, label
      ="Growth", color="blue", linestyle="solid")
213  axs[1, 1].plot(range(length), decay_model_2.sequence, label=
      "Decay", color="red", linestyle="dashed")
214  axs[1, 1].plot(range(length), combined_model_2.sequence,
      label="Combined", color="green", linestyle="dotted")
215  axs[1, 1].set_title(f"Growth_Rate:_{growth_rate_2},_Decay_
      Rate:_{decay_rate_2}")
216  axs[1, 1].set_xlabel("Iteration")
217  axs[1, 1].set_ylabel("Value")
218  axs[1, 1].legend()
219
220  # Set all y axis equal
221  for ax in axs.flat:
222      ax.set_ylim(0, 200)
223
224  plt.show()

1  hw1_choice_parameters.py
2

```

```

3 import matplotlib.pyplot as plt
4
5 # Task 1
6 # Create class BioModel
7 class BioModel(object):
8     def __init__(self, sequence):
9         self.sequence = sequence # Class has input sequence
10
11 #Task 4
12 # Define iter method to start iterable
13 def __iter__(self):
14     self.index = 0 # Start index at 0
15     return self
16
17 # Define next method to continue iterable
18 def __next__(self):
19     # Logic to determine when to stop iteration
20     if self.index < len(self.sequence):
21         self.index += 1
22         return self.sequence[self.index - 1] # Too lazy
23         # to add more lines. Subtract one from index
24         # for accurate output
25     else:
26         raise StopIteration # Stop iteration if index is
27         # not less than length
28
29 # Task 6
30 def __eq__(self, other):
31     if not isinstance(other, BioModel):
32         return NotImplemented
33     if len(self.sequence) != len(other.sequence):
34         raise ValueError("Two arrays are not equal in
35         length!")
36
37     count_match = 0
38     for x, y in zip(self.sequence, other.sequence):
39         if x == y:
40             count_match += 1
41
42     return count_match
43
44 # Task 2
45 # Create class ExponentialGrowthModel
46 class ExponentialGrowthModel(BioModel):
47     def __init__(self, start, rate):
48         super().__init__(sequence=[])
49         self.start = start # Class has input start
50         self.rate = rate # Class has input rate
51
52 # Task 3

```

```

49     # Callable method
50     def __call__(self, length):
51         self.sequence = [self.start] # Sequences begins with
            start parameter
52         curr_val = self.start # Calculation begins with
            start parameter
53
54         # Loop through and calculate sequence
55         for _ in range(1, length):
56             next_val = curr_val * (1 + self.rate) #
                Calculate next value
57             self.sequence.append(next_val) # Add next value
                to sequence
58             curr_val = next_val # Update current value to
                calculated value
59
60         print(self.sequence) # Print final sequence
61
62         # Define __len__ method to print length when called
63         def __len__(self):
64             return len(self.sequence)
65
66     # Task 5
67     class ExponentialDecayModel(BioModel):
68         def __init__(self, start, rate):
69             super().__init__(sequence=[])
70             self.start = start
71             self.rate = rate
72
73         def __call__(self, length):
74             self.sequence = [self.start]
75             curr_val = self.start
76
77             for _ in range(1, length):
78                 next_val = curr_val * (1 - self.rate)
79                 self.sequence.append(next_val)
80                 curr_val = next_val
81
82             print(self.sequence)
83
84         def __len__(self):
85             return len(self.sequence)
86
87     # Task 3 result
88     print("Task_3_result")
89     GM = ExponentialGrowthModel(start=285, rate=0.9)
90     GM(length=5)
91     print(len(GM)) # Output: 5
92
93     # Task 4 Result

```

```

94 print("Task_4_result")
95 print([n for n in GM])
96
97 # Task 5 Result
98 print("Task_5_result")
99 DM = ExponentialDecayModel(start=285, rate=0.4)
100 DM(length=5)
101 print(len(DM)) # Output: 5
102 print([n for n in DM])
103
104 # Task 6 Result
105 print("Task_6_result")
106 GM = ExponentialGrowthModel(start=285, rate=0.3)
107 GM(length=5)
108
109 GM2 = ExponentialGrowthModel(start=285, rate=0.3)
110 GM2(length=5)
111
112 print(GM == GM2) # Output: 5 (All output matches)
113
114 GM3 = ExponentialGrowthModel(start=285, rate=0.1)
115 GM3(length=3) # [100, 120.0, 144.0]
116
117 print(GM == GM3) # Raises ValueError: Two arrays are not
    equal in length!

```