Chandler Justice - A02313187

**MATH5620** Homework #5
**Due**: April 30, 2024 @ 23:59

**Question 1**: For the Logisitic Equation, defined in Question 2, write out the details for finding the carrying capacity for the system. You should be able to do this without solving the ODE. Build a code that will return the carrying capacity of the population being modeled given appropriate input(s).

**Solution 1**: I wrote the following code to determine the carrying capacity:

```
def determine_carrying_capacity(a, b):
    return a/b
```

This equation first takes the growth rate divided by the decay rate. This gives us the carrying capacity of the ODE in question.

**Question 2**: Write a code that will implement the Explicit Euler Method on the Logisitic equation,

$$\frac{dP}{dt} = \alpha\ P - \beta\ P^2$$

with $P(0) = 10$. Test your code using the following cases:

- $\alpha = 1.5, \beta = 0.001$
- $\alpha = 1.5, \beta = 0.01$
- $\alpha = .15, \beta = 0.001$

Make sure $\Delta t$ is small enough to be able to resolve the solution and see that the carrying capacity is acheived.

**Solution 2**: I wrote the following code that utilizes the explicit Euler method:

```
def explicit_euler(a, b, h, p0, function):

    time_step = np.arange(0, 100, h)
    P = np.zeros(len(time_step)) # stores state at a given time step
    P[0] = p0

    for i in range(0, len(P) -1):
        P[i + 1] = P[i] + h*function(a, b, P[i])

    return P
```

which returns the following results given the requested inputs:

```
a = 1.5, b = 0.001: [  10.           10.149        10.30020498 ... 1500.         1500.  1500.        ]
a = 1.5, b = 0.01: [ 10.          10.14        10.28181804 ... 150.         150. 150.        ]
a = 0.15, b = 0.001: [ 10.          10.014       10.0280182  ... 149.99935933 149.9993603 149.99936126]
```

These results line up with the expected carrying capacity.

**Question 3**: Repeat the work in Question 1 using the Implicit Euler Method. Use some algebra to figure out how to write out the algorithm. As discussed in class you will need to determine an appropriate for for the left hand side of the equation.
**Solution 3**: I wrote the following code to implement the Implicit Euler Method:

```
def implicit_euler(a, b, h, p0, function):

    time_step = np.arange(0, 100, h)
    P = np.zeros(len(time_step))
    P[0] = p0
    explicit_results = explicit_euler(a, b, h, p0, function)

    for i in range(0, len(P) - 1):
        P[i + 1] = P[i] + (h * function(a, b, explicit_results[i + 1]))

    return P
```

The results between this method and the explicit method are basically identical.

**Question 4**: Repeat the work in Question 1. using a predictor-corrector method defined by explicit Euler for the prediction and implicit Euler for the correction step.
**Solution 4**: Here is my implementation of the predictor-corrector method:

```
def trapezoid_euler(a, b, h, p0, function):

    time_step = np.arange(0, 100, h)
    P = np.zeros(len(time_step))

    P[0] = p0

    for i in range(0, len(P) - 1):
        predicate = P[i] + h * function(a, b, P[i])
        corrector = P[i] + h/2 * (function(a, b, P[i]) + function(a, b, predicate))
        P[i + 1] = corrector
    return P
```

Which further verifies the results obtained in question 2.

---