7.5

Mathematical Methods

Padé Approximants

1 The edit distance

Question 1

edit_distance.py for question 2

```python
import numpy as np


class edit_distance:
    def __init__(self, string1, string2):
        self.string1 = string1
        self.string2 = string2
        self.distance_matrix = \
            np.zeros( (len(self.string1) + 1,
                    len(self.string2) + 1) )


    def min_distance(self):
        # Boundary conditions
        for i in range(len(self.string1) + 1):
            self.distance_matrix[i][0] = i
        for j in range(len(self.string2) + 1):
            self.distance_matrix[0][j] = j

        # Find D(i,j) for each i,j
        for i in range(len(self.string1)):
            for j in range(len(self.string2)):
                char_match = 1  # for identifying S_i == T_j
                if self.string1[i] == self.string2[j]:
                    char_match = 0

                self.distance_matrix[i + 1][j + 1] = \
                min(self.distance_matrix[i + 1][j] + 1,
                    self.distance_matrix[i][j + 1] + 1,
                    self.distance_matrix[i][j] + char_match)

        return int(self.distance_matrix[len(self.string1)] \
                                [len(self.string2)])

if __name__ == "__main__":
    test_case = edit_distance('shesells', 'seashells')
    print(test_case.min_distance())
    print(test_case.distance_matrix)
```

edit_distance.py for question 2

```
import numpy as np


class edit_distance:
    def __init__(self, string1, string2):
        self.string1 = string1
        self.string2 = string2
        self.distance_matrix = \
            np.zeros( (len(self.string1) + 1,
                    len(self.string2) + 1) )


    def min_distance(self):
        # Boundary conditions
        for i in range(len(self.string1) + 1):
            self.distance_matrix[i][0] = i
        for j in range(len(self.string2) + 1):
            self.distance_matrix[0][j] = j

        # Find D(i,j) for each i,j
        for i in range(len(self.string1)):
            for j in range(len(self.string2)):
                char_match = 1  # for identifying S_i == T_j
                if self.string1[i] == self.string2[j]:
                    char_match = 0

                self.distance_matrix[i + 1][j + 1] = \
                min(self.distance_matrix[i + 1][j] + 1,
                    self.distance_matrix[i][j + 1] + 1,
                    self.distance_matrix[i][j] + char_match)

        return int(self.distance_matrix[len(self.string1)] \
                                [len(self.string2)])

if __name__ == "__main__":
    test_case = edit_distance('shesells', 'seashells')
    print(test_case.min_distance())
    print(test_case.distance_matrix)
```