Research Project

Mark Saleh 103994313

# Research Question

How does the choice of programming paradigm (procedural or object oriented) affect the implementation of complex and/or repetitive mathematical functions?

# Research Method

I intend to write implementations of mathematical functions that involve thousands of repetitions. Some examples of these include Gaussian elimination, Newton’s method, integration, and sorting. Then I intend to compare how ‘good’ procedural and object-oriented methods are at solving them.

For the coding language, I need one that is good at both procedural and object-oriented programming to make the comparison fair. Python is “multi-paradigm”, meaning that it can be used for both procedural and object-oriented programming. Additionally, I found it very simple to work with while doing the “clock in another language” task. So I choose Python.

There are several comparisons I can make between the two implementations. One is the speed at which it solves the functions. Another is the length of the code, which can be measures in lines, characters, or bytes. I can also compare their readability, although this would have to be a subjective comparison. Finally, I will award points for any other differences that stick out, such as code elegance.

At the end, I will also try to explain why certain differences arise between the two codes. This will require me to use my knowledge of the object-oriented programming concepts in Unit Learning Outcomes to compare it with the more familiar procedural paradigm.

As an example, here is some code to compare Gaussian elimination in both paradigms. This is (so far) part of my research project:

import time

import random

def generate\_random\_matrix(rows, columns):

    matrix = []

    for \_ in range(rows):

        row = [random.randint(1, 100) for \_ in range(columns)]

        matrix.append(row)

    return matrix

rows = 500

columns = 501

random\_matrix = generate\_random\_matrix(rows, columns)

class GaussianElimination:

    def \_\_init\_\_(self, matrix):

        self.matrix = matrix

    def eliminate(self):

        n = len(self.matrix)

        for i in range(n):

            pivot = self.matrix[i][i]

            for j in range(i + 1, n):

                factor = self.matrix[j][i] / pivot

                for k in range(i, n + 1):

                    self.matrix[j][k] -= self.matrix[i][k] \* factor

    def solve(self):

        self.eliminate()

        n = len(self.matrix)

        solution = [0] \* n

        for i in range(n - 1, -1, -1):

            solution[i] = self.matrix[i][n] / self.matrix[i][i]

            for j in range(i - 1, -1, -1):

                self.matrix[j][n] -= self.matrix[j][i] \* solution[i]

        return solution

def gaussian\_elimination(matrix):

    n = len(matrix)

    for i in range(n):

        pivot = matrix[i][i]

        for j in range(i + 1, n):

            factor = matrix[j][i] / pivot

            for k in range(i, n + 1):

                matrix[j][k] -= matrix[i][k] \* factor

    solution = [0] \* n

    for i in range(n - 1, -1, -1):

        solution[i] = matrix[i][n]

        for j in range(i + 1, n):

            solution[i] -= matrix[i][j] \* solution[j]

        solution[i] /= matrix[i][i]

    return solution

start\_time = time.time()

procedural = gaussian\_elimination(random\_matrix)

procedural\_time = time.time() - start\_time

start\_time = time.time()

gaussian = GaussianElimination(random\_matrix)

object = gaussian.solve()

object\_time = time.time() - start\_time

print("Procedural Gaussian Elimination Solution:", procedural)

print("Object-Oriented Gaussian Elimination Solution:", object)

print("Procedural Execution Time:", procedural\_time)

print("Object-Oriented Execution Time:", object\_time)