Sprawozdanie z przedmiotu Metody obliczeniowe

Dział: Interpolacja

Wykonał:

Imię i nazwisko Bartłomiej Kleszczewski

Nr indeksu 87058

Grupa 3

Prowadzący dr.hab Agnieszka Bołtuć

1. Przykład nr 13

(-4,1127)(-2,81)(0,3)(2,77)(4,1023) x = 3

2. Rozwiązanie rachunkowe przykładu wybraną metodą

Interpolacja Metoda Lagrange'a

Przykład 13)

$(-4,1127)(-2,81)(0,3)(2,77)(4,1023) \qquad x=3$

$$L_0 = \frac{(3+2)(3-0)(3-2)(3-4)}{(-4+2)(-4-0)(-4-2)(-4-4)} = -\frac{15}{384}$$

$$L_1 = \frac{(3+4)(3-0)(3-2)(3-4)}{(-2+4)(-2-0)(-2-2)(-2-4)} = \frac{-21}{-96} = \frac{21}{96}$$

$$L_2 = \frac{(3+4)(3+2)(3-2)(3-4)}{(0+4)(0+2)(0-2)(0-4)} = -\frac{35}{64} = L_2$$

$$L_3 = \frac{(3+4)(3+2)(3-0)(3-4)}{(2+4)(2+2)(2-0)(2-4)} = -\frac{105}{-96} = \frac{105}{96}$$

$$L_4 = \frac{(3+4)(3+2)(3-0)(3-2)}{(4+4)(4+2)(4-0)(4-2)} = \frac{105}{384}$$

$$R_0 = 1127 \cdot -\frac{15}{384} = \frac{-16905}{384}$$

$$R_1 = 81 \cdot \frac{21}{96} = \frac{1701}{96} = \frac{6804}{384}$$

$$R_2 = 3 \cdot -\frac{35}{64} = \frac{-105}{64} = -\frac{630}{384}$$

$$R_3 = 77 \cdot \frac{105}{96} = \frac{8085}{96} = \frac{32340}{384}$$

$$R_4 = 1023 \cdot \frac{105}{384} = \frac{107415}{384}$$

$$W(3) = \frac{-16905+6804-630+32340+107415}{384} = \frac{129024}{384} = 336$$

3. Rozwiązania przykładu poszczególnymi programami

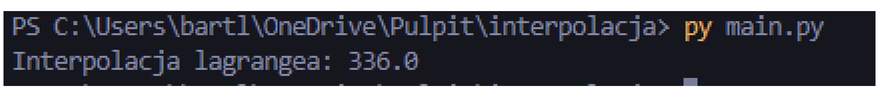
Stróktóra klasy

```
class Interpolation:

    def __init__(self, points, derX, derY):
        self.points = points
        self.diffQuotients = self.computeDifferenceQuotients()
        self.h = points[1][0] - points[0][0]
        self.progressiveDiffs = self.computeProgressiveDifferences()
        self.derX = derX
        self.derY = derY ##pochodne
```

a) metoda Lagrange'a

```
    def langreneInterpolation(self, x):
        total = 0
        n = len(self.points)
        for i in range(n):
            xi, yi = self.points[i]
            Li = 1
            for j in range(n):
                if i != j:
                    xj, _ = self.points[j]
                    Li *= (x - xj) / (xi - xj)
            total += yi * Li
        return total
```



b) metoda Newtona z ilorazami różnicowymi

```
    def computeDifferenceQuotients(self):
        n = len(self.points)
        table = [[0] * n for _ in range(n)]
        for i in range(n):
            table[i][0] = self.points[i][1]

        for j in range(1, n):
            for i in range(n - j):
                numerator = table[i + 1][j - 1] - table[i][j - 1]
                denominator = self.points[i + j][0] - self.points[i][0]
                table[i][j] = numerator / denominator
```

```
        return [table[0][j] for j in range(n)]

    def newtonInterpolation(self, x):
        n = len(self.points)
        result = self.diffQuotients[0]
        productTerm = 1

        for i in range(1, n):
            productTerm *= x - self.points[i - 1][0]
            result += self.diffQuotients[i] * productTerm

        return result
```

```
PS C:\Users\bartl\OneDrive\Pulpit\interpolacja> py main.py
336.0
```

c) metoda Newtona z różnicami progresywnymi

```
    def computeProgressiveDifferences(self):
        n = len(self.points)
        diffs = [y for _, y in self.points]
        progressiveDiffs = [diffs]

        for i in range(1, n):
            currentDiffs = []
            for j in range(n - i):
                diff = progressiveDiffs[i - 1][j + 1] - progressiveDiffs[i - 1][j]
                currentDiffs.append(diff)
            progressiveDiffs.append(currentDiffs)

        return progressiveDiffs

    def newtonProgressiveInterpolation(self, x):
        n = len(self.points)
        result = self.progressiveDiffs[0][0]
        productTerm = 1

        for i in range(1, n):
            productTerm *= x - self.points[i - 1][0]
            result += (
                (self.progressiveDiffs[i][0] / (self.h**i))
                / self.factorial(i)
                * productTerm
            )

        return result
```

```
PS C:\Users\bartl\OneDrive\Pulpit\interpolacja> py main.py
336.0
```

d) metoda funkcji sklejanych

```
def cubicSplineInterpolation(self, x):
    xPkt = [p[0] for p in self.points]
    yPkt = [p[1] for p in self.points]
    n = len(xPkt)
    A = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
    b = yPkt + self.derY
    for i in range(n):
        xi = xPkt[i]
        A[i][:4] = [1, xi, xi**2, xi**3]
        for j in range(1, i):
            A[i][j + 3] = (xi - xPkt[j]) ** 3
    for i in range(2):
        xi = self.derX[i]
        A[n + i][1:4] = [1, 2 * xi, 3 * xi**2]
        if i == 1:
            for j in range(1, n - 1):
                A[n + i][j + 3] = 3 * (xi - xPkt[j]) ** 2

    xWyniki = GaussElimination(A, b)

    hi = 0
    for i in range(1, n):
        if x < xPkt[i]:
            hi = i
            break
    wartoscWPunkcie = (
        xWyniki[0] + xWyniki[1] * x + xWyniki[2] * x**2 + xWyniki[3] * x**3
    )
    for i in range(1, hi):
        wartoscWPunkcie += xWyniki[i + 3] * (x - xPkt[i]) ** 3
    return wartoscWPunkcie
```

```
PS C:\Users\bartl\OneDrive\Pulpit\interpolacja> py main.py
331.9999999999998
```

```
xPochodne = [-4, 4]
yPochodne = [-1093,1003]
```