Παραρτήματα

Κώδικας Υπολογισμών Παρεμβολής-Ελαχίστων Τετραγώνων

interpolation.py

```
import numpy as np
import scipy.integrate as integrate


def df(x, xValues, dValues):
    """Derivative using partial differences
    second order"""
    h = xValues[1] - xValues[0]
    theta = (x - xValues[0]) / h
    return (dValues[1] + 0.5 * (2*theta - 1)*dValues[2]) / h


def d2f(x, xValues, dValues):
    """Second Derivative using partial differences
    second order"""
    h = xValues[1] - xValues[0]
    return dValues[2] / np.power(h, 2)


def differenceTable(fxValues):
    """Calculates difference table and returns its
    top diagonal"""
    n = fxValues.size
    dValues = np.copy(fxValues)
    for i in range(1, n):
        for j in reversed(range(i, n)):
            dValues[j] = dValues[j] - dValues[j-1]
    return dValues


def dividedDifferenceTable(xValues, fxValues):
    """Calculates divided difference table and returs its
    top diagonal"""
    n = fxValues.size
    dValues = np.copy(fxValues)
```

```
    for i in range(1, n):
        for j in reversed(range(i, n)):
            dValues[j] = (dValues[j] - dValues[j-1]) / \
                         (xValues[j] - xValues[j-i])
    return dValues


def addCoefficient(xValues, fxValues, coefficients):
    """Adds extra Newton Polynomial Coefficient"""
    n = coefficients.size
    x = xValues[n]  # Get x of new point to interpolate
    y_x = fxValues[n]  # Get y of new point to interpolate
    p_x = NestedMultiplication(x, xValues, coefficients)  #
    evaluate p(x)
    product = 1.
    for i in range(n):
        product = product * (x - xValues[i])  # (x-x0)...(x-x_n
    -1)
    newCoefficient = (y_x - p_x) / product  # new coefficient
    return np.append(coefficients, newCoefficient)


def NestedMultiplication(x, xValues, coeff):
    """Evaluates Newton Polynomial at x in nested form
    given the interpolating points and its coefficents"""
    n = coeff.size
    y = coeff[n-1]
    for i in reversed(range(n - 1)):
        y = coeff[i] + (x - xValues[i]) * y
    return y


def lSquaresRightHand(f, order, a, b):
    """Calculates right hand vector of least
    squares system of equations"""
    n = order + 1
    y = np.zeros(n)
    for i in range(n):
        y[i] = integrate.quad(lambda x: f(x) * np.power(x, i),
         a, b)[0]
```

```
        y[i] = 0. if np.isclose(y[i], 0., atol=1e-8) else y[i]
    return y


def normalEquations(order, a, b):
    """Calculates normal equations matrix used in least
    squares system of equations using monomials"""
    n = order + 1
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = integrate.quad(lambda x: np.power(x, i+j),
             a, b)[0]
    return A
```

Εκτέλεση Προγραμμάτων