Zadanie 20

20 O. Dopasuj wielomiany niskich stopni do danych zawartych w pliku `http://th-www.if.uj.edu.pl/zfs/gora/metnum16/w.txt`, zakładając, że pomiary są nieskorelowane i obarczone takim samym błędem. Ustal za pomocą kryterium Akaike, jaki stopień wielomianu wybrać. Przyjmując

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - w(x_i))^2\,, \tag{17}$$

gdzie (x_i, y_i) oznaczają punkty pomiarowe, N jest liczbą pomiarów, $w(x)$ dopasowanym wielomianem, znajdź macierz kowariancji estymatorów (czyli współczynników dopasowanego wielomianu).
Jest to jeden z niewielu przypadków, w których trzeba explicite znaleźć odwrotność jakiejś macierzy.

Kod w języku Python:

```python
import numpy as np


def loadData():
    plik = open("dane.txt")
    tmp_points = []
    tmp_values = []
    for linia in plik:
        tmp = linia.strip().split()
        tmp_points.append(float(tmp[0]))
        tmp_values.append(float(tmp[1]))
    return tmp_points, tmp_values


def Akaike(stopien, points, values):
    tmpArray = [[0] * (stopien + 1) for i in range(stopien + 1)]
    for i in range(stopien + 1):
        for j in range(stopien + 1):
            tmpArray[i][j] = pow(points[i], stopien - j)

    A = np.array(tmpArray)
    b = [0] * (stopien + 1)
    for i in range(len(b)):
        b[i] = values[i]

    a = np.linalg.solve(A, b)
    Q = 0

    for i in range(stopien):
        tmp = 0
        for j in range(stopien + 1):
            tmp += (a[stopien - j] * pow(points[i], stopien - 1))
        tmp -= values[i]
        Q += tmp

    if Q > 0:
        Q = np.log(Q) + float(2 * s / len(values))
        return Q
    else:
```

```python
        return 0


def calculateW(n, points):
    sigma2 = 0
    sigmaX2 = 0
    for i in range(len(points)):
        sigmaX2 += pow(points[i], i + 1)
        sigma2 += points[i]
    sigmaX2 *= n
    W = sigmaX2 - sigma2
    return W


def calculateWa(n, points, values):
    sigmaXY = 0
    sigmaX = 0
    sigmaY = 0
    for i in range(len(points)):
        sigmaXY += points[i] * values[i]
        sigmaX += points[i]
        sigmaY += values[i]
    sigmaXY *= n
    Wa = sigmaXY - sigmaX * sigmaY
    return Wa


def calculateWb(points, values):
    sigmaX2 = 0
    sigmaY = 0
    sigmaX = 0
    sigmaXY = 0
    for i in range(len(values)):
        sigmaX2 += pow(points[i], 2)
        sigmaY += values[i]
        sigmaX += points[i]
        sigmaXY += points[i] * values[i]

    Wb = sigmaX2 * sigmaY - sigmaX * sigmaXY
    return Wb


def function(a, x):
    result = 0
    for i in range(len(a)):
        result += a[i] * pow(x, len(a) - 1)
    return result


def var(points, values, a):
    result = 0
    for i in range(len(points)):
        tmp = values[i] - function(a, points[i])
        result += pow(tmp, 2)
    result /= len(points)
    return result;


def createMatrix(points):
```

```python
    A = [[0] * 2 for i in range(len(points))]
    for i in range(len(points)):
        A[i][0] = points[i]
        A[i][1] = 1
    return A


# Wczytywanie danych z pliku
tmp_points, tmp_values = loadData()
points = np.array(tmp_points)
values = np.array(tmp_values)

s = 10  # Stopien wielomianu dla ktorgo szukamy AIC

print("\nWartosc kryterium Akaike'a dla stopnia:")
akaikeValues = [0] * s
for i in range(s):
    akaikeValues[i] = Akaike(i + 1, points, values)  # 0 gdy wynik bylby ujemny
    print(i + 1, ") Q = ", akaikeValues[i])

n = 1  # Wybieramy stopien wielomianu
for i in range(s - 1):
    if akaikeValues[i + 1] < akaikeValues[i]:
        if akaikeValues[i + 1] > 0:
            n = i + 2

print("Wybieramy wielomian ", n, " stopnia\n")  # Bedzie to 1 stopien

a = [0] * (n + 1)  # Tablica wspolczynnikow a i b

# Obliczanie wspolczynnikow a i b
W = calculateW(n, points)
Wa = calculateWa(n, points, values)
Wb = calculateWb(points, values)

a[0] = float(Wa / W)
a[1] = float(Wb / W)

print("Obliczone wspolczynniki wielomianu:", "\na = ", a[0], "\nb = ", a[1])

# Obliczanie wariancji
Var = var(points, values, a)
print("\nObliczona wariancja: ", Var)

# Obliczanie macierzy kowariancji
A = np.array(createMatrix(points))
Cp = np.dot(A.transpose(), A)
Cp = np.linalg.inv(Cp)
Cp = Cp * Var

print("\nMacierz kowariancji:\n", Cp)
```

Wyniki:

```
Wartosc kryterium Akaike'a dla stopnia:
1 ) Q =  3.2757912761029937
2 ) Q =  9.169591277380185
3 ) Q =  13.451122098439544
4 ) Q =  16.072126064596397
5 ) Q =  0
6 ) Q =  0
7 ) Q =  0
8 ) Q =  0
9 ) Q =  0
10 ) Q =  0
Wybieramy wielomian  1  stopnia

Obliczone wspolczynniki wielomianu:
a =  -7.564287580515825
b =  -20.33217858958501

Obliczona wariancja:  260.6896717969843

Macierz kowariancji:
 [[ 6.11028713 -0.04773662]
 [-0.04773662  2.037011  ]]
```

Pomiary są nieskorelowane i obarczone takim samym błędem. Wykorzystując liniowe zagadnienie najmniejszych kwadratów minimalizowałem formę kwadratową po a_s, a_{s-1},… a_0. Obliczając kolejne pochodne po a_s, a_{s-1},… a_0, otrzymujemy s+1 równań z s+1 niewiadomymi. Do rozwiązania zagadnień numerycznych i algebraicznych użyłem biblioteki NumPy. Znając wektor współczynników jesteśmy w stanie obliczyć AIC.

$$AIC = \ln Q + \frac{2s}{N},$$

Wartość kryterium Akaike była najmniejsza dla wielomianu pierwszego stopnia. Do obliczenia wartośći współczynników a i b wykorzystałem regułę Cramer'a:

$$W = \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{pmatrix}$$

$$W_a = \begin{pmatrix} \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} y_i & n \end{pmatrix}$$

$$W_b = \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i \end{pmatrix}$$

Aby znaleźć macierz kowariancji estymatorów należy wyliczyć wariancje ze wzoru podanego w treści zadania oraz macierz $A^{n \times 2}$ równań liniowych dla n punktów, w których wykonano pomiary. Ponieważ pomiary były nieskorelowane i obarczone takim samym błędem macierz kowariancji upraszcza się do wzoru:

$$\mathbf{C_p} = \sigma^2 \left(\mathbf{A}^T \mathbf{A}\right)^{-1}$$