Algorytmy numeryczne - projekt

Szymon Gosk, Damian Górski, Niuta Godlewska

November 20, 2020

Contents

1 Wstęp

Niniejszy dokument stanowi dokumentacje wszystkich części projektu przedmiotu *Algorytmy numeryczne*. Kolejne części projektu będą opisane w kolejnych sekcjach, a kolejne elementy poszczególnych częsci - w podsekcjach.

Dokument ten jest również podsumowaniem pracy członków zespołu **1.1**, w którego skład wchodzą **Szymon Gosk**, **Damian Górski** i **Niuta Godlewska**. Członkowie zespołu wspólnie oświadczają, iż projekt był realizowany przez każdego z nich w równym stopniu zaangażowania.

Zgodnie z wymaganiami, kod programów został umieszczony w niniejszym dokumencie, jednak dla przejrzystości i wygody czytającego zaleca się zapoznianie się z tym samym kodem na repozytorium projektu: https://github.com/Szymon-Gosk/szymon-gosk.github.io

Zaimplementowany projekt można znaleźć pod adresem: https://szymon-gosk.github.io/

2 Interpolacja wielomianowa : Przyrost naturalny

2.1 Opis

Zadanie: Program, który oszacuje przyrost naturalnyna świecie w przyszłości. Węzły mają przedstawiać przyrost naturalny zmieniający się wczasie.

Metoda numeryczna użyta w tym zadaniu to **metoda newtona**. Początkowo posiadamy zbiór $n + 1$ danych:

$$\{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\}$$

Metoda polega na interpolacji danych w wielomian w postaci:

$$P(x) = \sum_{k=0}^{n} \left(b_k \prod_{i=0}^{k-1} (x - x_i) \right)$$

która pozwala oszacować wartości dla argumentów spoza zbioru danych.

Współczynniki b_k można uzyskać korzystając z poniższego wzoru:

$$b_k = \left(\frac{b_k - \sum\limits_{i=0}^{k-1} \left(b_i \prod\limits_{j=0}^{i-1} (x_k - x_j) \right)}{\prod\limits_{i=0}^{k-1} (x_k - x_i)} \right)$$

Przykład: Dany jest następujący zbiór danych:

$$\{(2005, 6503), (2010, 6894), (2015, 7256)\}$$

Zbiór współczynników dla tego zbioru to:

$b_0 = 6503$
$b_1 = 78.2$
$b_2 = -0.58$
przez co uzyskujemy wielomian w postaci:

$$P(x) = 6503 + 78.2(x - 2005) - 0.58(x - 2005)(x - 2010)$$

lub po przekształceniach:

$$P(x) = -0.58x^2 + 2406.9x - 2487717$$

Korzystając z tego wielomianu można przybliżyć wynik dla dowolnej wartości x. Dla $x = 2020$ podstawiamy wartość:

$$P(2020) = -0.58 \cdot (2020)^2 + 2406.9 \cdot 2020 - 2487717 = 7589$$

Kolejnym krokiem jest metoda ogólna przekształcania wielomianu w postaci Newtona w postać ogólną:

$$\sum_{k=0}^{n} \left(b_k \prod_{i=0}^{k-1} (x - x_i) \right) = \sum_{k=0}^{n} \left(a_k x^k \right)$$

Obecnie, wielomian ma postać:

$$b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + ... + b_n(x - x_0)...(x - x_{n-1})$$

Pierwszym krokiem będzie wymnożenie kolejnych iloczynów $(x - x_i)$ dla danego współczynnika b_k. W tym celu zdefiniujmy macierz $(n + 1) \times (n + 1)$:

$$M = \begin{pmatrix} \lambda_{(0,0)} & \lambda_{(0,1)} & \cdots & \lambda_{(0,j)} & \cdots & \lambda_{(0,n)} \\ \lambda_{(1,0)} & \lambda_{(1,1)} & \cdots & \lambda_{(1,j)} & \cdots & \lambda_{(1,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ \lambda_{(i,0)} & \lambda_{(i,1)} & \cdots & \lambda_{(i,j)} & \cdots & \lambda_{(i,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ \lambda_{(n,0)} & \lambda_{(n,1)} & \cdots & \lambda_{(n,j)} & \cdots & \lambda_{(n,n)} \end{pmatrix}$$

Zamysł elementów $\lambda_{(i,j)}$ jest następujący - zapiszmy wielomian w postaci opisanej wyżej:

$$b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + ... + b_n(x - x_0)...(x - x_{n-1}) = b_0\lambda_{(0,0)} + b_1(\lambda_{(1,1)}x + \lambda_{(0,1)}) + ... + b_n(\lambda_{(n,n)}x^n + \lambda_{(n-1,1)}x^{n-1} + ... + \lambda_{(0,n)}x^0)$$

Innymi słowy, są to iloczyny kolejnych podwielomianów Newtona $\prod_{i=0}^{k-1}(x - x_i)$.

Konsekwencją tego jest fakt, iż:

$$\underset{j<i}{\forall} \lambda_{(i,j)} = 0$$

co pozwala nam zapisać macierz M jako:

$$M = \begin{pmatrix} \lambda_{(0,0)} & \lambda_{(0,1)} & \cdots & \lambda_{(0,j)} & \cdots & \lambda_{(0,n)} \\ 0 & \lambda_{(1,1)} & \cdots & \lambda_{(1,j)} & \cdots & \lambda_{(1,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_{(i,j)} & \cdots & \lambda_{(i,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & \lambda_{(n,n)} \end{pmatrix}$$

Dodatkowo, warto zauważyć, że:

$$\underset{i=j}{\forall} \lambda_{(i,j)} = 1$$

Do uzyskania postaci ogólnej wielomianu, należy dodać do siebie współczynniki λ odpowiadające danym potęgom, pomnożone przez odpowiednie współczynniki b. Wykorzystamy do tego układ liniowy:

$$\begin{pmatrix} \lambda_{(0,0)} & \lambda_{(0,1)} & \cdots & \lambda_{(0,j)} & \cdots & \lambda_{(0,n)} \\ 0 & \lambda_{(1,1)} & \cdots & \lambda_{(1,j)} & \cdots & \lambda_{(1,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_{(i,j)} & \cdots & \lambda_{(i,n)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & \lambda_{(n,n)} \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_i \\ \vdots \\ a_n \end{pmatrix}$$

Rozwiązaniem tego układu dla poszczególnych a_i jest:

$$a_i = \sum_{j=0}^{i} b_j \lambda_{(i,j)}$$

Przy wyznaczaniu współczynników λ posłużymy się zbiorem kombinacji z i po j, oznaczonym jako $\mathbb{C}_i^j$:

$$\lambda_{(i,j)} = (-1)^{(i+j)} \cdot \left(\sum_{K \in \mathbb{C}_i^j} \prod_{k \in K} (x_k) \right)$$

co daje nam wzór ogólny na a_i:

$$a_i = \sum_{j=0}^{i} \left((-1)^{(i+j)} \cdot b_j \left(\sum_{K \in \mathbb{C}_j^i} \prod_{k \in K} (x_k) \right) \right)$$

Teraz możemy zapisać wzór wielomianu w postaci ogólnej:

$$P(x) = \sum_{i=0}^{n} \left(x^i \sum_{j=0}^{i} \left((-1)^{(i+j)} \cdot b_j \left(\sum_{K \in \mathbb{C}_j^i} \prod_{k \in K} (x_k) \right) \right) \right)$$

Przykład: Załóżmy, że mamy wielomian w postaci:

$$P(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + b_3(x - x_0)(x - x_1)(x - x_2)$$

Zapisujemy równanie liniowe z uzupełnionymi, wyliczonymi według wzoru współczynnikami λ:

$$\begin{pmatrix} 1 & -x_0 & x_0x_1 & -x_0x_1x_2 \\ 0 & 1 & -(x_0+x_1) & x_0x_1+x_0x_2+x_1x_2 \\ 0 & 0 & 1 & -(x_0+x_1+x_2) \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Z czego otrzymujemy:

$$a_0 = b_0 - b_1x_0 + b_2x_0x_1 - b_3x_0x_1x_2$$

$$a_1 = b_1 - b_2(x_0 + x_1) + b_3(x_0x_1 + x_0x_2 + x_1x_2)$$

$$a_2 = b_2 - b_3(x_0 + x_1 + x_2)$$

$$a_3 = b_3$$

A więc:

$P(x) = (b_0 - b_1x_0 + b_2x_0x_1 - b_3x_0x_1x_2) + x(b_1 - b_2(x_0 + x_1) + b_3(x_0x_1 + x_0x_2 + x_1x_2)) + x^2(b_2 - b_3(x_0 + x_1 + x_2)) + x^3b_3$

2.2 Opis implementacji algorytmu

Implementacja składowych całościowego algorytmu jest prosta z uwagi na matematyczny zapis obliczeń. Każda suma lub iloczyn, jest równoważny pętli w programie.

Pierwszym elementem algorytmu jest wyliczenie współczynników b. Program oblicza je zgodnie z podanym wzorem, a do liczenia każdej sumy lub iloczynu wykorzystuje pętlę.

Po obliczeniu współczynników b, program tworzy *String*, w którym zawiera dane współczynniki, wartości x_i i zwraca *String* zawierający postać Newtona wielomianu.

Korzystając z powstałego wielomianu program pobiera argument wpisaną przez użytkownika i wylicza wartośc wielomianu dla tego argumentu.

2.3 Struktury danych i struktura programu

Program wykorzystuje proste listy, jeden z podstawowych typów danych w języku *Javascript*.

Dodatkowo przeprowadzane są operacje na liczbach - *float* - a także, w celu wyświetlania rezultatów, na danych typu *String*.

Struktura skryptu obliczeniowego odpowiada warstwie teoretycznej implementacji - tj. kolejnośc wykowywanych zadań w jednym skrypcie jest taka sama jak w sekcji **2.1** oraz **2.2**.

2.4 Program

Poniżej zostały zaprezentowane przykłady działania programu dla różnych przypadków:

TODO TODO TODO TODO

2.5 IO (wejście-wyjście)

3 Projekt nr 2

4 Projekt nr 3

5 Projekt nr 4

6 Tekst kodu programów

6.1 Interpolacja wielomianowa : Przyrost naturalny

```
$(document).ready(function() {

    // przycisk 'Zatwierdz'
    $('#submit-button').click(function() {

        // inicjalizacja zmiennych
        let inputs = [];
```

```
let years = [];
let empty = 0;

// Zebranie wartosci z inputow
$('.input-y').each(function () {
    if($(this).val() !== "") {
        years.push($(this).val());
    }

});

$('.input-v').each(function () {
    if($(this).val() !== "") {
        inputs.push($(this).val());
    }
});

// Sprawdzenie czy sa przynajmniej 3 lata TODO
if(empty > 2) {

}

// zamiana stringosw na inty
let values = [];
inputs = inputs.map(x => +x);
years = years.map(x => +x);

for (let i = 0; i < inputs.length; i++) {
    let temp = [];
    temp[0] = years[i];
    temp[1] = inputs[i];
    values[i] = temp
}

// values[X][0] - X-th year      (x)
// values[X][1] - X-th value     (y)
let N = values.length;
let b = [];

b[0] = values[0][1];
```

```
for(let k = 1; k <= N-1; k++) {

    let sum = values[0][1];
    let denominator = 1;

    for(let i = 1; i <= k-1; i++) {
        let product = 1;
        for(let j = 0; j < i; j++) {
            product *= (values[k][0] - values[j][0]);
        }
        sum += b[i]*product;
    }

    for(let i = 0; i <= k-1; i++) {
        denominator *= (values[k][0] - values[i][0]);
    }

    b[k] = (values[k][1] - sum)/denominator;
}

// Wielomian Newtona

let polynomial = "P(x) = " + b[0];
let polynomial_graph = "" + b[0];

for(let k = 1; k <= N-1; k++) {
    if(b[k] < 0) {
        polynomial = polynomial + " - " + Math.abs(b[k]);
        polynomial_graph = polynomial_graph + " - " + Math.ab
    } else {
        polynomial = polynomial + " + " + b[k];
        polynomial_graph = polynomial_graph + " + " + b[k];
    }
    for(let i = 0; i <= k-1; i++) {
        polynomial = polynomial + "(x - " + values[i][0] + ")
        polynomial_graph = polynomial_graph + "*(x - " + valu
    }
}

$("#polynomial").html(polynomial);
```

```
// Wyliczenie

let P = b[0];
let x = parseInt($('#year-6').val());

for (let k = 1; k <= N - 1; k++) {
    let ingredient = b[k];
    for (let i = 0; i <= k - 1; i++) {
        ingredient *= (x - values[i][0]);
        polynomial = polynomial + "(x - " + values[i][0] + ")
    }
    P += ingredient;
}

P = Math.round(P * 100) / 100;

$('#result').html(P);

// Wykres

// TODO

var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox:[-5

// Macro function plotter
function addCurve(board, func, atts) {
    return board.create('functiongraph', [func], atts);
}

// Simplified plotting of function
function plot(func, atts) {
    if (atts==null) {
        return addCurve(board, func, {strokewidth:2});
    } else {
        return addCurve(board, func, atts);
    }
}

let func = 'function f(x) { ' +
    '    return ' + polynomial_graph + '; ' +
    '} ' +
```

```
            'c = plot(f);';

        // Usage of the macro
        function doIt() {
            console.log(':)');
            eval(func);
        }

        doIt();

    });

});
```

6.2 Projekt nr 2

6.3 Projekt nr 3

6.4 Projekt nr 4