

**Wykonał:** Radosław Smoter

**Grupa:** 14

**Nr:** 27

**Numer zadania:** 9

**Przykład:** 62

**Prowadzący:** Prof. dr hab. inż.  
Volodymyr Samotyj

# **Politechnika Krakowska**

**Wydział Inżynierii Elektrycznej i Komputerowej**

**Sprawozdanie: Wstęp do Programowania**

## Spis treści

Polecenie.....	1
Kod programu.....	2
Wyniki.....	7
(1) Dane dla pliku a (x/y):.....	7
(2) Dane z pliku b ()......	7
(3) Wykres.....	8
Opis programu.....	9
Wnioski.....	10

## Polecenie

Na kartce w kratkę narysować ręcznie dowolny wykres. Wizualnie odczytać wartości  $x$  oraz odpowiednie wartości  $y$ . Wartości  $(x, y)$  zapisać do pliku tekstowego a.

Wybrać wzór dla aproksymacji:  $y(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + a_3 \cdot x^3 + a_4 \cdot x^4$  .

Układ rozwiązać metodą Gaussa-Seidela i obliczyć współczynniki aproksymacji  $a_i$  .

Wyniki zapisać do pliku b. Na jednym wykresie narysować  $y(x)$  korzystając z danych z plików a i b.

# Kod programu

```
/**
 * @file main.c
 * @author Radoslaw Smoter (radoslaw.smoter@student.pk.edu.pl)
 * @version 0.1
 * @brief Approximating a function with Gauss-Seidel method
 * @date 2021-12-14
 *
 * @copyright Copyright (c) 2021
 */

/*
 * Draw any plot. Read pairs of values (x, y), save them to a txt file.
 * Choose a formula for the approximation function, e.g.
 *  $y(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 + a_4 * x^4$ 
 * Select 5 pairs (x, y) from the txt file.
 * Create system of linear equations for all 5 (x, y).
 * Solve the system for all  $a_i$ , which are the approximation coefficients, using
 Gauss-Seidel method.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Number of elements to pull out of the file. */
#define N 10

/* loaddata did not return valid data. */
#define ERR_EMPTY_FILECONTENT -10
/* File did not open correctly. */
#define ERR_INVALID_FILE NULL
/* File name is an empty string. */
#define ERR_EMPTY_FILENAME NULL

double *load_data(char *);
void split_data(double *, double *, double *);
void approximate(double A[][N/2], double B[]);

int main(int argc, char const *argv[])
{
    /* filecontent is an array with format XXYX... */
    double *filecontent = load_data("a.txt");

    /* End the program, because it failed: loaddata did not return valid data. */
    if (filecontent == NULL) {
        fprintf(stderr, "Error: Data Did not Load Correctly.\n");
        return ERR_EMPTY_FILECONTENT;
    }
}
```

```

double X[N/2];
double Y[N/2];

split_data(filecontent, X, Y);

/* Free filecontent, because it's no longer used. */
if (filecontent != NULL) free(filecontent);

double approx[N/2][N/2];
for (int i = 0; i < N/2; i++)
    for (int j = 0; j < N/2; j++)
        approx[i][j] = pow(X[i], j);

approximate(approx, Y);

return 0;
}

/* Loads exactly n floating point numbers and returns them as an array. */
double *load_data(char *filename) {
    /* Filename is an empty string. */
    if (strcmp(filename, "") == 0) {
        fprintf(stderr, "Error: File Name is an Empty String.\n");
        return ERR_EMPTY_FILENAME;
    }

    /* Memory for n elements. */
    double *filecontent = malloc(sizeof(double) * N);

    FILE *file = fopen(filename, "r");

    if (file != NULL) {
        /* End of the file indicator. */
        _Bool isEOF = 1;
        /* Buffer for the file output */
        double *fp = malloc(sizeof(double));

        long long i;
        for (i = 0; isEOF == 1 && i < N; i++) {
            isEOF = fscanf(file, "%lf", fp);
            filecontent[i] = *fp;
        }

        free(fp);
        fclose(file);
    }
    /* The File did not open correctly */
    else {
        fprintf(stderr, "Error: The File %s Did Not Open Correctly.\n", filename);
        return ERR_INVALID_FILE;
    }

    /* Everything went correctly */
    return filecontent;
}

```

```

/* Splits one array with length n to two arrays with length n/2. With every odd element
in one array and every even in the other one. */
void split_data(double *A, double *X, double *Y) {
    /* For A */
    int i = 0;
    /* For X */
    int k = 0;
    /* For Y */
    int j = 0;

    while (i < N) {
        if (i % 2 == 0)
            X[k++] = A[i];
        else
            Y[j++] = A[i];
        i++;
    }
}

/* Colors: reset, bold red, bold yellow */
#define C_RESET "\e[0m"
#define C_BRED "\e[1;31m"
#define C_BYEL "\e[1;33m"

/* Print results of the approximation (coefficients) and number of iterations. */
void printV(double v[], unsigned long long iter) {
    /* Coefficients */
    for (int i = 0; i < N/2; i++)
        printf("%11s%d", "a_", i);
    printf("\n");
    for (int i = 0; i < N/2; i++)
        printf("%s%12.5lf%s", C_BYEL, v[i], C_RESET);
    printf("\n");

    /* Number of iteration */
    printf("Current iteration number is: %s%llu%s\n", C_BRED, iter++, C_RESET);
    for (int i = 0; i < 60; i++)
        printf("%s", "-");
    printf("\n");
}

/* Calculate absolute errors between iterations of approximations of the coefficients.
Ends approximation, when all of the errors drop below the level of significance. */
_Bool approx_err(double error[], double v[]) {
    /* Indicates level of significance. */
    const double MAX_ERROR = 1e-14;

    for (int i = 0; i < N/2; i++)
        if (fabs(v[i] - error[i]) > MAX_ERROR)
            return 1;
    return 0;
}

```

```

/* Find coefficients for the approximation polynomial.*/
void approximate(double X[][N/2], double Y[]) {
    /* Iteration number */
    unsigned long long iter = 1;
    /* Coefficients of the approximation equation */
    double coeff[N/2];
    /* Error between each iteration of approximation. Has exact value as coeff in the
previous iteration. Initialised to ones. */
    double error[N/2];

    for (int i = 0; i < N/2; i++) {
        for (int j = 0; j < N/2; j++) {
            /* populate coeff with zeros */
            coeff[i] = 0;
            /* populate error with ones */
            error[i] = 1;
        }
    }

    /* Repeat until error is significant. */
    while (approx_err(error, coeff)) {
        /* Calculate new errors for each coeff */
        for (int i = 0; i < N/2; i++)
            error[i] = coeff[i];

        for (int i = 0; i < N/2; i++) {
            /* Sum of elements (coefficients * powerset) from each row */
            double row_sum = 0;
            for (int j = 0; j < N/2; j++)
                if (i != j)
                    /* Multiply by -1.0, because it's subtraction */
                    row_sum += X[i][j] * coeff[j] * (-1.0);
            /* Add Y's value to row */
            row_sum += Y[i];
            /* Divide all elements from the row by X's value that stands before the
calculated coefficient. */
            coeff[i] = row_sum / X[i][i];
        }

        /* Prevent iter from overflow. */
        if (iter == __INT_MAX__)
            break;
        iter++;
    }

    printV(coeff, iter);

    FILE *file = fopen("b.txt", "w");
    if (file != NULL) {
        for (int i = 0; i < N/2; i++)
            fprintf(file, "%.5lf*x^%d\n", coeff[i], i);
        fclose(file);
    }
    else {
        fprintf(stderr, "Error: File did not open successfully.\n");
        return;
    }
}

```

}  
}



# Wyniki

## (1) Dane dla pliku a (x/y):

1	2.00
2	5.50
3	7.50
4	9.00
5	10.3
6	11.0
7	11.7
8	12.2
9	12.5
10	12.5
11	12.2
12	11.7
13	11.0
14	10.4
15	9.90
16	9.40
17	9.00
18	8.80
19	8.70
20	8.70
21	8.80
22	9.00
23	9.40
24	9.80
25	10.2
26	11.0
27	12.0
28	13.5
29	14.8
30	16.2
31	18.0

## (2) Dane z pliku b ( $a_i \cdot x^i$ ).

-4.70000\*x^0  
9.04167\*x^1  
-2.77083\*x^2  
0.45833\*x^3  
-0.02917\*x^4

### (3) Wykres

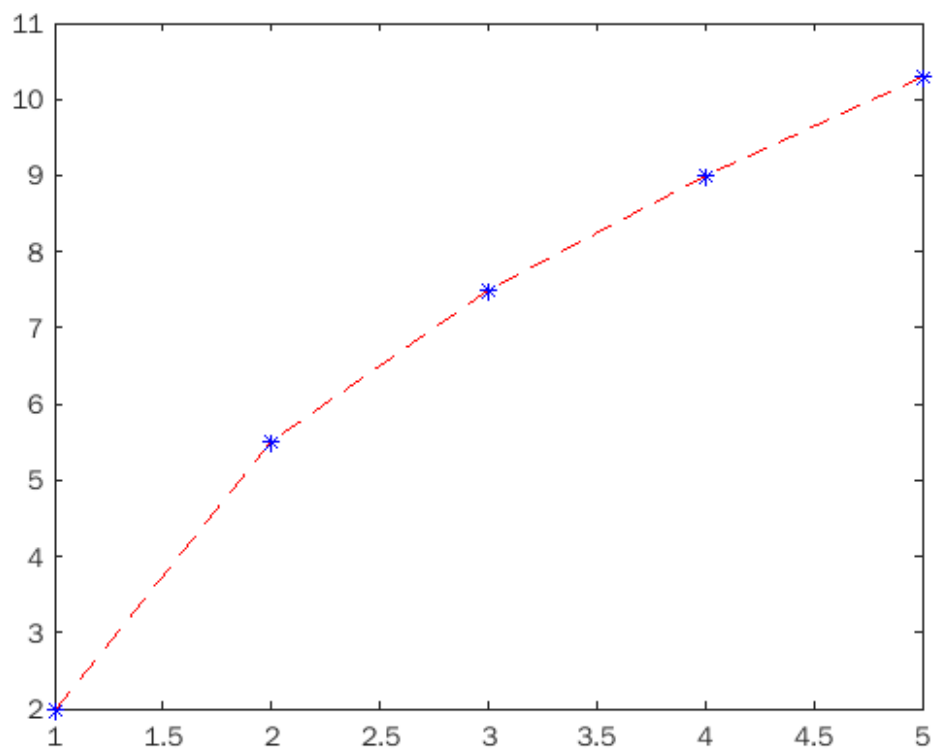


Figure 1: Wykres wielomianu o podanych współczynnikach nałożony na wykres punktowy par  $(x, y)$  pochodzących z warunków zadania.

## Opis programu

Funkcja `main()` wybiera dane z pliku tekstowego „a.txt” za pomocą funkcji `load_data()` i zapisuje je do wskaźnika `filecontent`. W razie niepowodzenia wczytania danych na `stderr` wypisywany jest komunikat o błędzie oraz zakończony program. W przeciwnym razie, `filecontent` jest dzielony funkcją `split_data()` do wartości `x` i `y` do odpowiednio tablic `X[]` i `Y[]`. Wtedy `filecontent` nie jest więcej potrzebny, więc zostaje uwolniony (`free`). Funkcję aproksymującą program przechowuje jako tablicę `approx[][]`, do której wartości przypisuje podwójna pętla wywołująca operację potęgowania na wartościach z `X[]`. Dla takiego zestawu danych `approx[][]` oraz `Y[]` zachodzi aproksymacji współczynników stojących przed kolejnymi potęgami `x`, co zachodzi w funkcji `approximate()`.

Funkcja `load_data()` łączy dane z pliku o nazwie podanej w parametrze. Jeżeli nazwa jest pusta, funkcja kończy swoje działanie i zwracany jest `NULL`. W przeciwnym razie, alokowana jest pamięć na `N` (określone w makrze) elementów. Jeżeli żądany plik udaje się otworzyć, pobierane są wartości aż natrafiono na znak końca pliku lub aż uzyskano żadaną liczbę rekordów. Jeżeli pliku nie udaje się otworzyć, zwracany jest błąd, a funkcja kończy się znakiem `NULL`. W przeciwnym razie, zwracany jest wskaźnik do pobranych danych.

Funkcja `split_data()` „dzieli” dane pochodzące z tablicy `*filecontent` na dwie tablice, w zależności od parzystości indeksu danego rekordu. Wartości o parzystych indeksach są oznaczane jako `x`, a o nieparzystych jako `y`, i w taki sposób są zwracane w tablicach `X[]` oraz `Y[]`.

Funkcja `approximate()` przeprowadza aproksymację zestawu danych złożonego z macierzy wartości potęg kolejnych elementów `x`, zawartych w tablicy `X[][]` oraz wartości `y`, w tablicy `Y[]`. Na początku, funkcja zapełnia tablicę `coeff[]`, przechowującą wartości współczynników równania wielomianowego – zerami, oraz tablicę `error[]`, która przechowuje wartości współczynników z poprzedniej iteracji programu – jedynkami. Pętla `while()` wykonuje się, aż wartości wszystkich różnic współczynników między sąsiednimi iteracjami staną się mniejsze niż błąd określony w funkcji `approx_err()`, której jest to funkcjonalność. Dlatego też wartości `error`, przy każdej iteracji pętli, są ustawiane na wartości `coeff`, z minionej właśnie iteracji. Wewnątrz działa pętla, która oblicza wszystkie współczynniki – sumuje wartości wiersza (wyraz pochodzący z `X[][]` razy współczynnik, który mu odpowiada), pomijając wyraz odpowiadający aktualnie rozpatrywanemu współczynnikowi, następnie odejmuje od tej wartości odpowiadającą wartość `y` oraz dzieli przez wyraz, który stoi przed aktualnie rozpatrywanym współczynnikiem. Każdy nowy współczynnik jest zapisywany do tablicy współczynników – `coeff[]`. Po zakończeniu pętli wyniki są wypisywane na standardowe wyjście (`stdout`) oraz do pliku tekstowego „b.txt”. Jeżeli takiego pliku nie uda się utworzyć, to na `stderr` wypisywany jest błąd i funkcja jest zakańczana.

## Wnioski

Uzyskane wyniki dobrze oddają charakter podanego zestawu danych. Wielomian o podanych współczynnikach ściśle przylega do zadeklarowanych punktów.