Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт

До лабораторної роботи №8

На тему: «Наближення дискретних (таблично заданих) функцій»

З дисципліни: “Чисельні методи”

Лектор:
доц. каф. ПЗ
Мельник Н.Б.
Виконав:
ст. гр. ПЗ-18
Лук’янов Н.О.
Прийняв:
проф. каф. ПЗ
Гавриш В.І.
« … » … 2023 р.

∑ = ____________

Львів – 2023

Тема: наближення дискретних (таблично заданих) функцій**.**

Мета: ознайомлення із методом інтерполяції таблично заданих функцій.

Завдання

Використовуючи інтерполяційні поліноми Лангранжа та Ньютона, обчислити значення таблично зданої функції у точці $x_0 = 0{,}22$.

Таблиця 1. Таблично задана функція згідно з варіантом

x	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
f(x)	1,640390	1,680188	1,715637	1,746772	1,773649	1,796348	1,814967	1,829628	1,840467	1,847642

Інтерполяційний поліном Лагранжа

Один з методів знаходження інтерполяційного полінома запропонував Лагранж. Основна ідея цього методу полягає в пошуку полінома, який в одному довільному вузлі інтерполяції приймає значення одиниця, а в усіх інших вузлах - нуль.

Наближену функцію y = F(x) розглянемо у вигляді

$$F(x) = L_n(x) = \sum_{i=0}^{n} P_i(x) f(x_i),$$

де $P_i(x)$-такий многочлен, що

$$P_i(x_j) = \begin{cases} 0, & i \neq j, \\ 1, & i = j, \end{cases} \qquad i, j = \overline{0, n}.$$

Оскільки точки x_0, x_1,..., x_n є коренями полінома, то його можна записати у такому вигляді

$$P_i(x) = \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)},$$

а наближена функція F(x) , яку називають інтерполяційним многочленом Лагранжа, матиме вигляд

$$F(x) = L_n(x) = \sum_{i=0}^{n} \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} f(x_i)$$

або

$$L_n(x) = \sum_{i=0}^{n} L_i(x) y_i,$$

де коефіцієнти Лагранжа можна записати формулою

$$L_i(x) = \frac{(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}.$$

Основні етапи обчислювального алгоритму, реалізованого у програмному продукті мовою С :

1) введення даних користувачем для ініціалізації масиву, що відповідає за зберігання таблично заданої функції;
2) вивід, введеної користувачем таблично заданої функції, на екран за допомогою функції PrintFunction() (рис. 1);
3) введення даних користувачем для ініціалізації змінної, що відповідає за зберігання точки, у якій потрібно обчислити значення;
4) перевірка, чи вузли рівновіддалені, за допомогою функції IsEquidistant() (рис. 2);
5) обчислення значення таблично заданої функції у точці, заданої користувачем, за допомогою функції LagrangePolynomial() (рис. 3);
6) результат виконання програми (рис. 4).

```
void PrintFunction(double* Function, const int SIZE)
{
    printf("   x          y\n");
    for (int i = 0; i < SIZE; i++)
    {
        printf("| %.1f |  | %.5f |\n", *(Function + i), *(Function + SIZE + i));
    }
}
```

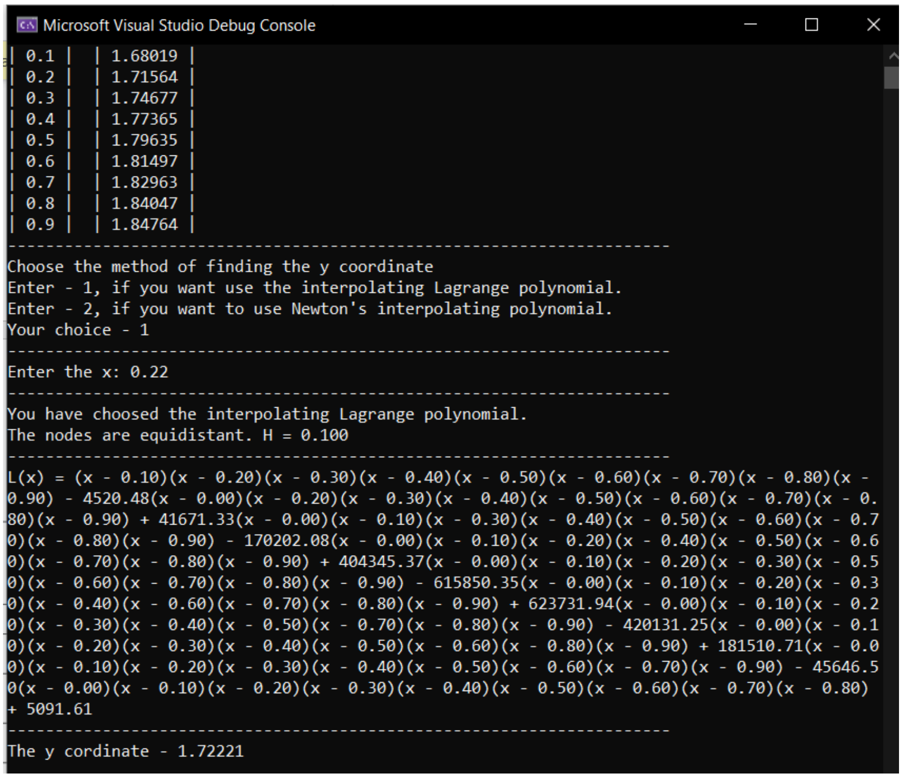
Рис. 1. Функція PrintFunction

```cpp
bool IsEquidistant(double* fArray, double* h, const int SIZE)
{

    if (SIZE >= 2)
    {
        *h = (*(fArray + 0) - *(fArray + 1));
    }
    else
    {
        return false;
    }

    *h = fabs(*h);

    for (int i = 0; i < SIZE-1; i++)
    {
        if ((fabs(*(fArray + i) - *(fArray + i + 1)) + +0.01) < *h)
        {
            return false;
        }
    }

    return true;
}
```

Рис. 2. Функція IsEquidistant

```cpp
double LagrangePolynomial(double x, double* Coefficients, int SIZE)
{
    double dResult = 0.0;

    for (int i = 0; i < SIZE; i++)
    {
        double D = 1.0;

        for (int j = 0; j < SIZE; j++)
        {
            if (j == i)
            {
                continue;
            }
            else
            {
                D *= (x - *(Coefficients + j)) / (*(Coefficients + i) - *(Coefficients + j));
            }
        }

        dResult += *(Coefficients + SIZE + i) * D;
    }

    return dResult;
}
```

Рис. 3. Функція LagrangePolynomial



```
Microsoft Visual Studio Debug Console
| 0.1 |   | 1.68019 |
| 0.2 |   | 1.71564 |
| 0.3 |   | 1.74677 |
| 0.4 |   | 1.77365 |
| 0.5 |   | 1.79635 |
| 0.6 |   | 1.81497 |
| 0.7 |   | 1.82963 |
| 0.8 |   | 1.84047 |
| 0.9 |   | 1.84764 |
----------------------------------------------------------------------
Choose the method of finding the y coordinate
Enter - 1, if you want use the interpolating Lagrange polynomial.
Enter - 2, if you want to use Newton's interpolating polynomial.
Your choice - 1
----------------------------------------------------------------------
Enter the x: 0.22
----------------------------------------------------------------------
You have choosed the interpolating Lagrange polynomial.
The nodes are equidistant. H = 0.100
----------------------------------------------------------------------
L(x) = (x - 0.10)(x - 0.20)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.70)(x - 0.80)(x -
0.90) - 4520.48(x - 0.00)(x - 0.20)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.70)(x - 0.
80)(x - 0.90) + 41671.33(x - 0.00)(x - 0.10)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.7
0)(x - 0.80)(x - 0.90) - 170202.08(x - 0.00)(x - 0.10)(x - 0.20)(x - 0.40)(x - 0.50)(x - 0.6
0)(x - 0.70)(x - 0.80)(x - 0.90) + 404345.37(x - 0.00)(x - 0.10)(x - 0.20)(x - 0.30)(x - 0.5
0)(x - 0.60)(x - 0.70)(x - 0.80)(x - 0.90) - 615850.35(x - 0.00)(x - 0.10)(x - 0.20)(x - 0.3
0)(x - 0.40)(x - 0.60)(x - 0.70)(x - 0.80)(x - 0.90) + 623731.94(x - 0.00)(x - 0.10)(x - 0.2
0)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.70)(x - 0.80)(x - 0.90) - 420131.25(x - 0.00)(x - 0.1
0)(x - 0.20)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.80)(x - 0.90) + 181510.71(x - 0.0
0)(x - 0.10)(x - 0.20)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.70)(x - 0.90) - 45646.5
0(x - 0.00)(x - 0.10)(x - 0.20)(x - 0.30)(x - 0.40)(x - 0.50)(x - 0.60)(x - 0.70)(x - 0.80)
+ 5091.61
----------------------------------------------------------------------
The y cordinate - 1.72221
```

Рис. 4. Результат виконання програми

Інтерполяційний поліном Ньютона

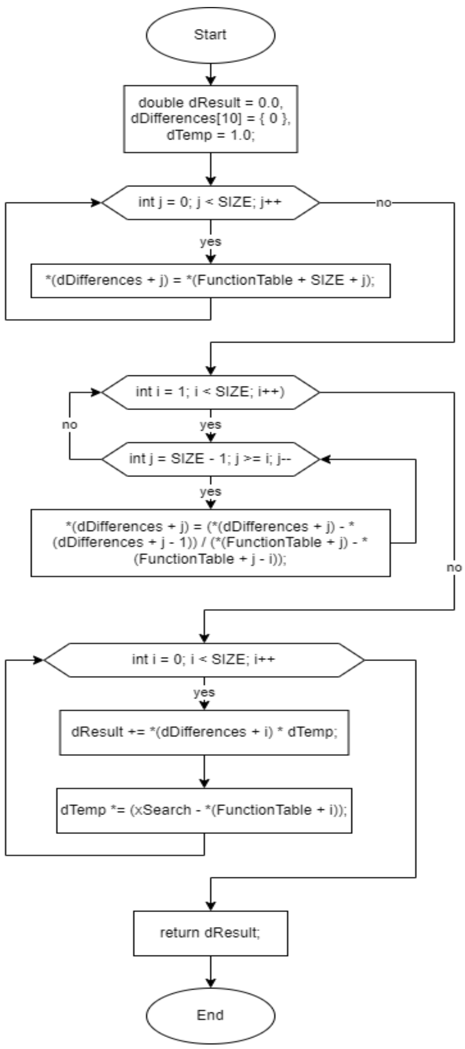


Рис. 5. Блок-схема обчислювального алгоритму

Основні етапи обчислювального алгоритму, реалізованого у програмному продукті мовою С :

1) введення даних користувачем для ініціалізації масиву, що відповідає за зберігання таблично заданої функції;
2) вивід, введеної користувачем таблично заданої функції, на екран за допомогою функції PrintFunction();
3) введення даних користувачем для ініціалізації змінної, що відповідає за зберігання точки, у якій потрібно обчислити значення;
4) перевірка, чи вузли рівновіддалені, за допомогою функції IsEquidistant();
5) обчислення значення таблично заданої функції у точці, заданої користувачем, за допомогою функції DoNewtotInterpolation() (рис. 6);
6) результат виконання програми (рис. 7).

```c
double DoForwardNewtoтInterpolation(double *FunctionTable, int SIZE, double xSearch)
{
    double dResult = 0.0, dDifferences[10] = { 0 }, dTemp = 1.0;

    for (int j = 0; j < SIZE; j++)
    {
        *(dDifferences + j) = *(FunctionTable + SIZE + j);
    }

    for (int i = 1; i < SIZE; i++)
    {
        for (int j = SIZE - 1; j >= i; j--)
        {
            *(dDifferences + j) = (*(dDifferences + j) - *(dDifferences + j - 1)) / (*(FunctionTable + j) - *(FunctionTable + j - i));
        }
    }

    for (int i = 0; i < SIZE; i++)
    {
        dResult += *(dDifferences + i) * dTemp;
        dTemp *= (xSearch - *(FunctionTable + i));
    }

    return dResult;
}
```

Рис. 6. Функція DoNewtoтInterpolation

```
Microsoft Visual Studio Debug Console
| 0.1 |  | 1.68019 |
| 0.2 |  | 1.71564 |
| 0.3 |  | 1.74677 |
| 0.4 |  | 1.77365 |
| 0.5 |  | 1.79635 |
| 0.6 |  | 1.81497 |
| 0.7 |  | 1.82963 |
| 0.8 |  | 1.84047 |
| 0.9 |  | 1.84764 |
-----------------------------------------------------------------------
Choose the method of finding the y coordinate
Enter - 1, if you want use the interpolating Lagrange polynomial.
Enter - 2, if you want to use Newton's interpolating polynomial.
Your choice - 2
-----------------------------------------------------------------------
Enter the x: 0.22
-----------------------------------------------------------------------
You have choosed the Newton's interpolating polynomial.
The nodes are equidistant. H = 0.100
Forward interpolation is used!
-----------------------------------------------------------------------
Newton Polynomial:
0.00 + 0.10*(x - 0.00) + 0.20*(x - 0.00)*(x - 0.10) + 0.30*(x - 0.00)*(x - 0.10)*(x - 0.20)
+ 0.40*(x - 0.00)*(x - 0.10)*(x - 0.20)*(x - 0.30) + 0.50*(x - 0.00)*(x - 0.10)*(x - 0.20)*(
x - 0.30)*(x - 0.40) + 0.60*(x - 0.00)*(x - 0.10)*(x - 0.20)*(x - 0.30)*(x - 0.40)*(x - 0.50
) + 0.70*(x - 0.00)*(x - 0.10)*(x - 0.20)*(x - 0.30)*(x - 0.40)*(x - 0.50)*(x - 0.60) + 0.80
*(x - 0.00)*(x - 0.10)*(x - 0.20)*(x - 0.30)*(x - 0.40)*(x - 0.50)*(x - 0.60)*(x - 0.70) + 0
.90*(x - 0.00)*(x - 0.10)*(x - 0.20)*(x - 0.30)*(x - 0.40)*(x - 0.50)*(x - 0.60)*(x - 0.70)*
(x - 0.80)
-----------------------------------------------------------------------
The y cordinate - 1.72221
```

Рис. 7. Результат виконання програми

Висновки

У результаті виконання лабораторної роботи, складено програму обчислення таблично заданої функції в точці $x_0 = 0{,}22$, використовуючи інтерполяційні поліноми Лагранжа та Ньютона. Програмний продукт розроблений у середовищі Microsoft Visual Studio мовою програмування С.

Інтерполяційний поліном Лагранжа:

$$L(x) = 0{,}30588x^9 - 1{,}22271x^8 + 2{,}05208x^7 - 1{,}87881x^6 + 1{,}01904x^5 - 0{,}32307x^4 + 0{,}06312x^3 - 0{,}22438x^2 + 0{,}42002x + 1{,}64039$$

Інтерполяційний поліном Ньютона:

$$P(x) = 0{,}30588x^9 - 1{,}22271x^8 + 2{,}05208x^7 - 1{,}87881x^6 + 1{,}01904x^5 - 0{,}32307x^4 + 0{,}06312x^3 - 0{,}22438x^2 + 0{,}42002x + 1{,}64039$$

Додаток

Назва файлу: **Lab8.c**

```
#include <stdio.h>
#include <math.h>
#include <Windows.h>
#include <stdbool.h>

void PrintLine(int, char);
void PrintFunction(double*, const int);
bool IsEquidistant(double*, double*, const int);
double LagrangePolynomial(double, double*, int);
void LimiterX(double*);
double DoNewtonInterpolation(double*, int, double);
void PrintLagrangePolynomial(double*, int);
void PrintNewtonPolynomial(double*, int n);

int main()
{
	SetConsoleOutputCP(1251);

	double Function[2][10] = {
		{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9},
		{1.640390, 1.680188, 1.715637, 1.746772, 1.773649, 1.796348, 1.814967,
1.829628, 1.840467, 1.847642}
	};

	double x = 0.0, y = 0.0, h = 0.0;
	int Choice = 0;


	printf("Tabulated function:\n");
	PrintLine(20, '-');
	PrintFunction(Function, 10);
	PrintLine(70, '-');

	printf("Choose the method of finding the y coordinate\n");
	printf("Enter - 1, if you want use the interpolating Lagrange polynomial.\nEnter - 2,
if you want to use Newton's interpolating polynomial.\nYour choice - ");
	scanf_s("%d", &Choice);
	PrintLine(70, '-');

	printf("Enter the x: ");
	scanf_s("%lf", &x);
	PrintLine(70, '-');

	LimiterX(&x);

	switch (Choice)
	{
	case 1:
		printf("You have choosed the interpolating Lagrange polynomial.\n");
		if (IsEquidistant(Function, &h, 10))
		{
			printf("The nodes are equidistant. H = %.3f\n", h);
			y = LagrangePolynomial(x, Function, 10);
			printf("The y cordinate - %.5f\n", y);
		}
		else
		{
			printf("The nodes are not equidistant...\n");
		}
		break;
	case 2:
		printf("You have choosed the Newton's interpolating polynomial.\n");

		if (IsEquidistant(Function, &h, 10))
```

```c
		{
			printf("The nodes are equidistant. H = %.3f\n", h);
			printf("Forward interpolation is used!\n");
			y = DoNewtonInterpolation(Function, 10, x);
			printf("The y cordinate - %.5f\n", y);
		}
		else
		{
			printf("The nodes are not equidistant...\n");
		}
		break;
	default:
		break;
	}
}

void PrintLine(int number, char symbol)
{
	while (number > 0)
	{
		printf("%c", symbol);
		number--;
	}
	printf("\n");
}

void PrintFunction(double* Function, const int SIZE)
{
	printf("   x          y\n");
	for (int i = 0; i < SIZE; i++)
	{
		printf("| %.1f |  | %.5f |\n", *(Function + i), *(Function + SIZE + i));
	}
}

bool IsEquidistant(double* fArray, double* h, const int SIZE)
{

	if (SIZE >= 2)
	{
		*h = (*(fArray + 0) - *(fArray + 1));
	}
	else
	{
		return false;
	}

	*h = fabs(*h);

	for (int i = 0; i < SIZE-1; i++)
	{
		if ((fabs(*(fArray + i) - *(fArray + i + 1)) + +0.01) < *h)
		{
			return false;
		}
	}

	return true;
}

double LagrangePolynomial(double x, double* Coefficients, int SIZE)
{
	double dResult = 0.0;

	for (int i = 0; i < SIZE; i++)
	{
		double D = 1.0;
```

```
			for (int j = 0; j < SIZE; j++)
			{
				if (j == i)
				{
					continue;
				}
				else
				{
					D *= (x - *(Coefficients + j)) / (*(Coefficients + i) -
*(Coefficients + j));
				}
			}

			dResult += *(Coefficients + SIZE + i) * D;
		}

		PrintLine(70, '-');
		PrintLagrangePolynomial(Coefficients, 10);
		PrintLine(70, '-');

		return dResult;
}

void LimiterX(double* x)
{
		if (*x < 0.0)
		{
			*x = 0.0;
		}
		else if (*x > 0.9)
		{
			*x = 0.9;
		}
		else
		{
			return;
		}
}

double DoNewtonInterpolation(double* FunctionTable, int SIZE, double xSearch)
{
		double dResult = 0.0;

		double dividedDiff[10];
		for (int i = 0; i < SIZE; i++) {
			dividedDiff[i] = *(FunctionTable + SIZE + i);
		}
		for (int j = 1; j < SIZE; j++) {
			for (int i = SIZE - 1; i >= j; i--) {
				dividedDiff[i] = (dividedDiff[i] - dividedDiff[i - 1]) /
(*(FunctionTable + i) - *(FunctionTable + i - j));
			}
		}

		double term = 1.0;
		for (int i = 0; i < SIZE; i++) {
			dResult += dividedDiff[i] * term;
			term *= (xSearch - *(FunctionTable + i));
		}

		PrintLine(70, '-');
		PrintNewtonPolynomial(FunctionTable, 10);
		PrintLine(70, '-');

		return dResult;
}

int IsForwardInterpolation(double *FunctionTable, int SIZE, double xSearch)
```

```
{
	bool IsForward = false;

	if (xSearch >= *(FunctionTable + 0) && xSearch <= *(FunctionTable + SIZE - 1))
	{
		IsForward = true;
	}

	return IsForward;
}

void PrintLagrangePolynomial(double* Function, int n)
{
	printf("L(x) = ");

	for (int i = 0; i < n; i++) {
		double term = 1.0;

		for (int j = 0; j < n; j++) {
			if (j != i) {
				term *= (*(Function + i) - *(Function + j));
				printf("(x - %.2f)", *(Function + j));
			}
		}

		term = *(Function + 10 + i) / term;

		if (term >= 0) {
			printf(" + %.2f", term);
		}
		else {
			printf(" - %.2f", -term);
		}
	}

	printf("\n");
}

void PrintNewtonPolynomial(double* Function, int n)
{
	printf("Newton Polynomial:\n");

	for (int i = 0; i < n; i++) {
		printf("%.2f", *(Function + i));

		for (int j = 0; j < i; j++) {
			printf("*(x - %.2f)", *(Function + j));
		}

		if (i < n - 1) {
			printf(" + ");
		}
	}

	printf("\n");
}
```