Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 1

Тема Аппроксимация таблично заданных функции полиномом Ньютона

Студент Кононенко Сергей

Группа ИУ7-43Б

Оценка (баллы) _____

Преподаватель Градов Владимир Михайлович

Москва — 2020 г.

1. Задание

1. Задана таблица значений функции вида x, $f(x)$. С помощью интерполяции, используя метод полинома Ньютона, найти приближённое значение функции от введённого x_0.
Ввод: значение x_0 и степень полинома.
Вывод: значение функции от x_0.

2. Методом обратной интерполяции найти корень функции $f(x) = 0$. Также найти корень методом половинного деления.
Ввод: ввод не требуется.
Вывод: корень функции $f(x) = 0$, найденный методом половинного деления и методом обратной интерполяции.

2. Описание алгоритма

Для двух точек:

$$y(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j}$$

Для трёх точек:

$$y(x_i, x_j, x_k) = \frac{y(x_i, x_j) - y(x_j, x_k)}{x_i - x_k}$$

Соответственно для n точек можно записать следующее выражение:

$$y(x_i, x_j, ..., x_n) = \frac{y(x_i, x_j, ..., x_{n-1}) - y(x_j, ..., x_n)}{x_i - x_n}$$

Отсюда искомый полином будет равен:

$$\mathcal{P}_n(x) = y_0 + (x - x_0)y(x_0, x_1) + (x - x_0)y(x_0, x_1)y(x_0, x_1, x_3) + ... + (x - x_0)...(x - x_n)y(x_0, x_1, ..., x_n)$$

Что эквивалентно: $\mathcal{P}_n(x) = \sum a_k x^k$

Исходный набор точек сортируется по возрастанию. Для отсортированного набора точек находится позиция, при вставке на которую заданного x_0 функция останется неубывающей на всём промежутке. Для найденной позиции выбирается *интерполяционная база* - поднабор исходного набора точек, который будет использоваться при построении таблицы разделенных разностей. По построенной таблице разделенных разностей находятся коэффициенты *полинома Ньютона*. По полученным коэффициентам строится полином Ньютона и вычисляется значение полинома в заданной точке x_0.

После вычисления значения функции в точке x_0 находится значение x при котором $f(x) = 0$. Данное значение находится методом половинного деления и методом обратной интерполяции.

3. Код программы

Основной пакет приложения.

```
package main

import (
        "fmt"
        "os"
        "sort"

        "./interp"
)

func main() {
        if len(os.Args) <= 1 {
                fmt.Printf("USAGE: lab_01 <datafile>\n")
                os.Exit(1)
        }

        f, err := os.Open(os.Args[1])
        if err != nil {
                fmt.Println("Error:", err)
                os.Exit(1)
        }

        eps := 0.00001
        bisErr := -1.111

        ds := interp.ReadDots(f)
        fmt.Printf("Table loaded from file:\n\n")
        ds.PrintDots()

        fmt.Printf("\nEnter X value and polynom degree: ")
        d, n := interp.ReadFuncData()

        sort.Sort(ds)

        d.Y = interp.Interpolation(ds, d, n)
        fmt.Printf("\nFunction value in %5.2f dot is %5.4f\n\n", d.X, d.Y)

        rootBis := interp.Bisection(ds, n, eps)
        if rootBis-bisErr < eps {
                fmt.Printf("Can't find root by bisection method in given dot set\n\n")
        } else {
                fmt.Printf("Root found by bisection method is %.4f\n", rootBis)
        }

        rootInvInterp := interp.InvInterpolation(ds, n)
```

```
        fmt.Printf("Root found by inverted interpolation method is %.4f\n\n",
                rootInvInterp)
}
```

Пакет обработки интерполяции.

1. Обработка точек на координатной плоскости.

```
package interp

import "fmt"

// Dot type used to represent cartesian dots.
type Dot struct {
        X float64
        Y float64
}

// DotSet type used to represent amount of cartesian dots.
type DotSet []Dot

// PrintDots used to print dots in table form to standart output.
func (ds DotSet) PrintDots() {
        for i := range ds {
                fmt.Printf("%8.2f %8.2f\n", ds[i].X, ds[i].Y)
        }
}

// GetPos used to find place where should insert dot to
// make yet set be sorted.
func (ds DotSet) GetPos(d Dot) int {
        var pos int

        if d.X < ds[0].X {
                pos = 0
        } else if d.X > ds[len(ds)-1].X {
                pos = len(ds) - 1
        } else {
                for i := 1; i < len(ds)-2; i++ {
                        if d.X > ds[i-1].X && d.X < ds[i].X {
                                pos = i
                        }
                }
        }

        return pos
}

func (ds DotSet) Len() int {
```

```
        return len(ds)
}

func (ds DotSet) Less(i, j int) bool {
        return ds[i].X < ds[j].X
}

func (ds DotSet) Swap(i, j int) {
        ds[i], ds[j] = ds[j], ds[i]
}

// AxisSwap used to swap dot's axises: X <-> Y.
func (ds DotSet) AxisSwap() {
        for i := range ds {
                ds[i].X, ds[i].Y = ds[i].Y, ds[i].X
        }
}
```

2. Реализация интерполяции.

```
package interp

import (
        "fmt"
        "io"
        "math"
        "sort"
)

// FTable used to represent table of float numbers.
type FTable [][]float64

// InvInterpolation used to find root of function using inverted interpolation.
func InvInterpolation(ds DotSet, n int) float64 {
        ds.AxisSwap()
        sort.Sort(ds)

        d := Dot{
                X: 0,
        }

        return Interpolation(ds, d, n)
}

// Bisection used to find root of function using bisection method.
func Bisection(ds DotSet, n int, eps float64) float64 {
        if math.Signbit(ds[0].Y) == math.Signbit(ds[len(ds)-1].Y) {
                return -1.111
```

```
        }
        low, high := ds[0].X, ds[len(ds)-1].X
        mid := (low + high) / 2

        for math.Abs(high-low) >= eps*mid+eps {
                lowd := Dot{
                        X: low,
                }
                midd := Dot{
                        X: mid,
                }
                if Interpolation(ds, lowd, n)*Interpolation(ds, midd, n) < 0 {
                        high = mid
                } else {
                        low = mid
                }
                mid = (low + high) / 2
        }

        return mid
}

// Interpolation used to find value of X Dot coordinate
// using Newton polynom.
func Interpolation(ds DotSet, d Dot, n int) float64 {
        tb := MakeTable(ds, d, n)
        p := tb[1][0]
        var c float64

        for i := 2; i < len(tb); i++ {
                c = 1
                for j := 0; j < i-1; j++ {
                        c *= d.X - tb[0][j]
                }
                c *= tb[i][0]
                p += c

        }

        return p
}

// MakeTable used to make table, which contains Newton
// polynom coefficients.
func MakeTable(ds DotSet, d Dot, n int) FTable {
        base := GetBase(ds, d, n)
        baselen := len(base)
        tb := make(FTable, n+2)
```

```
	tblen := len(tb)
	for i := range tb {
		tb[i] = make([]float64, baselen)
	}

	for i := range tb[0] {
		tb[0][i] = base[i].X
		tb[1][i] = base[i].Y
	}

	for i := 2; i < tblen; i++ {
		k := i - 2
		for j := 0; j < baselen-i+1; j++ {
			tb[i][j] = (tb[i-1][j] - tb[i-1][j+1]) /
				(tb[0][j] - tb[0][j+k+1])
		}
	}

	return tb
}

// GetBase used to make DotSet, which contains closest
// dots to find Newton polynom's coefficients.
func GetBase(ds DotSet, d Dot, n int) DotSet {
	base := DotSet{}
	pos := ds.GetPos(d)

	if pos <= n/2 {
		for i := 0; i < n+1; i++ {
			base = append(base, ds[i])
		}
	} else if len(ds)-pos-1 <= n/2 {
		for i := len(ds) - n - 1; i < len(ds); i++ {
			base = append(base, ds[i])
		}
	} else {
		lb := n / 2
		rb := n - lb + 2
		if pos+rb > len(ds)-1 {
			rb--
			lb++
		}
		if pos-lb < 0 {
			rb++
			lb--
		}

		for i := pos - lb; i < pos+rb; i++ {
```

```
                        base = append(base, ds[i])
                }
        }

        return base
}

// ReadFuncData used to read function X coordinate and polynom degree.
func ReadFuncData() (Dot, int) {
        var (
                d Dot
                n int
        )

        fmt.Scan(&d.X, &n)

        return d, n
}

// ReadDots used to read Dot objects to DotSet object from file.
func ReadDots(f io.Reader) DotSet {
        var (
                ds     DotSet
                curDot Dot
        )

        for {
                _, err := fmt.Fscanln(f, &curDot.X, &curDot.Y)
                if err == io.EOF {
                        break
                }
                ds = append(ds, curDot)
        }

        return ds
}
```