Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 3
З дисципліни «Алгоритми та методи обчислень»

На тему «Інтерполяція функцій»

Виконав:
студент 2 курсу ФІОТ
групи ІВ-71
Мазан Я. В.
Залікова – 7109

Перевірив:
ст.вик. Порєв В. М.

Київ – 2019

Мета:
Ознайомлення з інтерполяційними формулами Лагранжа, Ньютона, рекурентним співвідношенням Ейткена, методами оцінки похибки інтерполяції.

Завдання:
Закріплення, поглиблення і розширення знань студентів при вирішенні практичних обчислювальних завдань. Оволодіння обчислювальними методами і практичними методами оцінки похибки обчислень. Придбання умінь і навичок при програмуванні та налагодженні обчислювальних завдань на комп'ютері.

Індивідуальне завдання:

9	$e^{-(x+\sin x)}$	[2, 5]	1.6

Формула інтерполяції 1.6 → інтерполяційний многочлен Ньютона

Код програми:
ErrorBluriness.kt (розмитість похибки та k_Δ):
```
package com.labworks.amc_lab3
import com.jjoe64.graphview.series.DataPoint
import com.jjoe64.graphview.series.LineGraphSeries
class ErrorBluriness(val polynomeP: NewtonInterpolation) : Drawable {
    val polynomePow: Int = polynomeP.array.n
    val polynomePplusOne: NewtonInterpolation =
NewtonInterpolation(InterpolationObject(polynomePow+1,polynomeP.array.minRange,polynomeP.array.maxRa
nge,polynomeP.array.function))
    val polynomePplusTwo: NewtonInterpolation =
NewtonInterpolation(InterpolationObject(polynomePow+2,polynomeP.array.minRange,polynomeP.array.maxRa
nge,polynomeP.array.function))
    val deltaN: List<Double> = List(polynomeP.plotArgs.size, {i -> polynomeP.plotVals[i]-
polynomePplusOne.plotVals[i]})
    val deltaDeltaN: List<Double> = List(polynomeP.plotArgs.size, {i ->
polynomePplusOne.plotVals[i]-polynomePplusTwo.plotVals[i]})
    val interpolationBluriness: List<Double> = List(polynomeP.plotArgs.size, {i -> if (deltaN[i] !=
0.0) Math.abs(deltaDeltaN[i]/deltaN[i]) else 0.0})
    fun getBluriness() : List<Double> {
        return interpolationBluriness
    }
    val deltaExactN: List<Double> = List(polynomeP.plotArgs.size, {i -> polynomeP.plotVals[i] -
polynomeP.array.function(polynomeP.plotArgs[i])})
    val kDelta: List<Double> = List(polynomeP.plotArgs.size, {i -> if (deltaN[i] != 0.0) 1 -
deltaExactN[i]/deltaN[i] else 0.0})
    override fun pointsList(): LineGraphSeries<DataPoint> {
        return LineGraphSeries(Array<DataPoint>(polynomeP.plotArgs.size, {i ->
DataPoint(polynomeP.plotArgs[i],interpolationBluriness[i])}))
    }
}
```

NewtonInterpolation.kt (вміщує в собі дані для інтерполяції):
```
package com.labworks.amc_lab3
import android.widget.Toast
import com.jjoe64.graphview.series.DataPoint
import com.jjoe64.graphview.series.LineGraphSeries
/**
 * Class to collect data necessary to a successful function interpolation
 * @author Yan Mazan
 *
 * @param n         power of our Newton's interpolation polynome
 * @param minRange  starting argument of interpolated function
```

```
 * @param maxRange  ending argument of interpolated function
 * @param function  a function to interpolate
 */
class InterpolationObject (val n: Int, val minRange: Double, val maxRange:Double, val
function: (x:Double) -> Double) : Drawable {
    val delta = (maxRange - minRange)/n
    val keyArgums: MutableList<Double> = (0..n).map({i -> minRange +
delta*i}).toMutableList()
    val keyVals: MutableList<Double> = keyArgums.map(function).toMutableList()
    /**
     * Plot of basic function
     * @override Function pointsList of basic interface Drawab
     * @return Points of raw function onClick within the given range
     */
    override fun pointsList(): LineGraphSeries<DataPoint> {
        val difference = (maxRange-minRange)/1000
        val plotArgs: List<Double> = List(1000, {i -> minRange + difference*i})
        val plotVals: List<Double> = List(1000, {i -> function(plotArgs[i])})
        return LineGraphSeries(Array<DataPoint>(1000, {i ->
DataPoint(plotArgs[i],plotVals[i])}))
    }
}
```

NewtonInterpolation.kt (алгоритм інтерполяції та абсолютна її похибка)

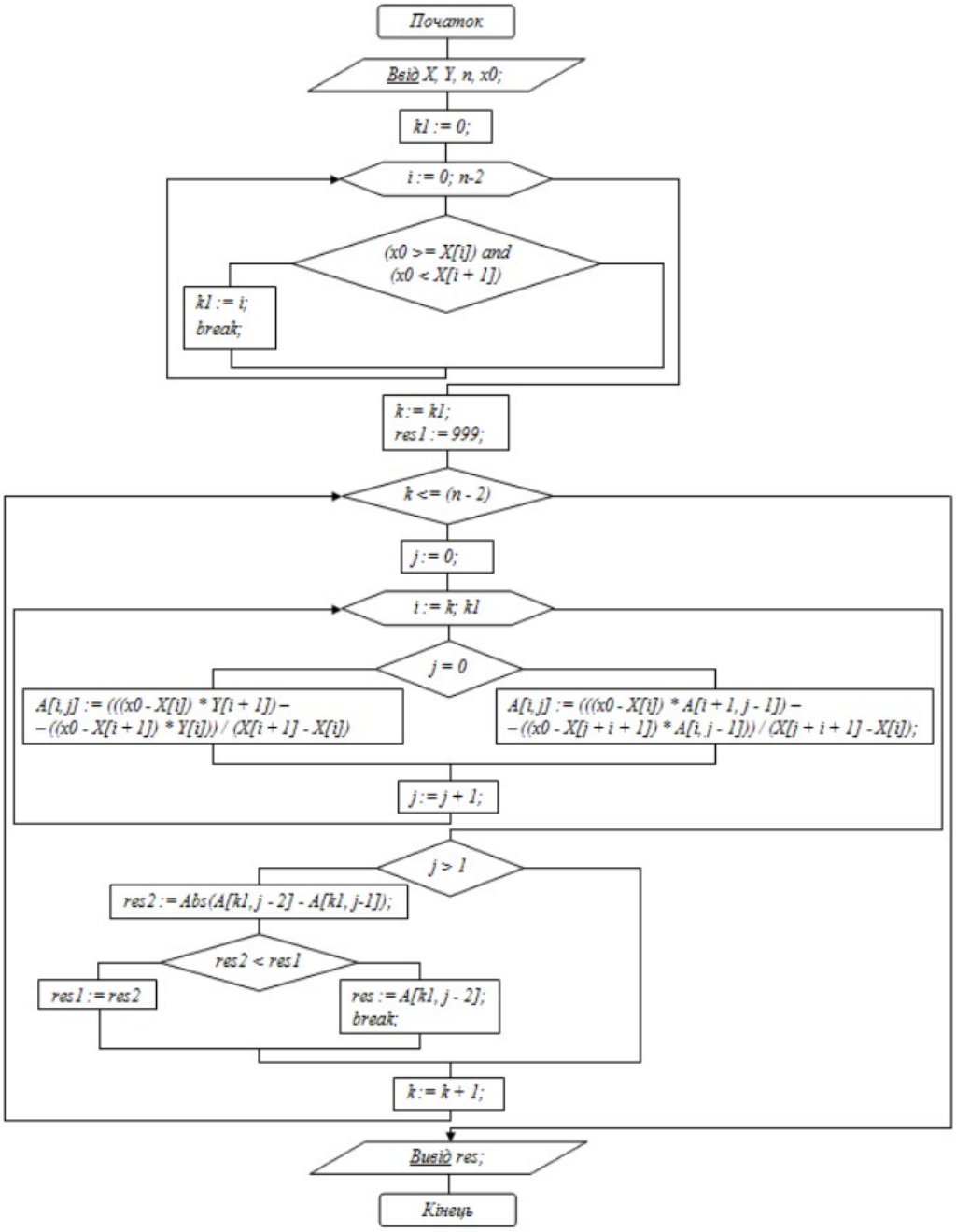
```
package com.labworks.amc_lab3
import com.jjoe64.graphview.series.DataPoint
import com.jjoe64.graphview.series.LineGraphSeries
import java.lang.ArithmeticException
/**
 * Class which performs all interpolation operations
 * @author Yan Mazan
 *
 * @param array InterpolationObject to interpolate
 */
class NewtonInterpolation (val array: InterpolationObject) : Drawable {
    val polynomeCoeff: List<Double> = dividedDifferenceList()
    val plotArgs: List<Double> = calculatePlotArgs()
    val plotVals: List<Double> = calculatePlotVals()
    val infelicityVals: List<Double> = calculateInfelicity()
    /**
     * Calculation of divided differences constant f(x0...xk) that is used to calculate Newton's
interpolation polynome
     * @param k - order of divided difference k in f(x0...xk), k <= n of array
     * @throws ArithmeticException if k > n
     * @return res = sum(i = 0 to k) f(x_i)/(x[i]-x[0])(x[i]-x[1])...(x[i]-x[i-1])(x[i]-x[i+1])...
(x[i]-x[k])
     */
    fun dividedDifference (k: Int) : Double {
        if (k > array.n)
            throw ArithmeticException("k must be smaller than n!")
        var res = 0.0
        for (i in 0..k) {
            var part_sum = 0.0
            val denominator = (0..k)
                .map({j -> if (i!=j) array.keyArgums[i]-array.keyArgums[j] else 1.0})
                .fold(1.0){multipl,el -> multipl*el}
            res += array.keyVals[i]/denominator
        }
        return res
    }
    /**
     * Calculation of divided differences constants list that is used to calculate Newton's
interpolation polynome
     * @return List of divided differences f(x0...xk) where k belongs to range [0,n]
     */
    fun dividedDifferenceList() : List<Double> {
        return List(array.n, {i -> dividedDifference(i)})
    }
    /**
     * @param array: [x0, x1, x2, ..., xn]
     * @input k - number of elements to truncate
     * @returns EG:
```

```
     *          polynomeArgs(0) = []
     *          polynomeArgs(3) = [x0, x1, x2]
     */
    private fun polynomeArgs(k: Int) : List<Double> {
        return array.keyArgums.dropLast(array.n-k+1)
    }
    /**
     * Calculates value of function using interpolation polynome
     * @input x - argument of Newton's polynome
     * @returns N(x) ~= f(x) where f is interpolated function
     */
    fun calculate (x: Double) : Double {
        var res = polynomeCoeff[0]
        for (i in 1..array.n-1) {
            res+=polynomeCoeff[i]*polynomeArgs(i).map({e -> x - e}).fold(1.0){multipl,el ->
multipl*el}
        }
        return res
    }
    /**
     * Privately calculates 1000 arguments in our given class range to onClick then
     * @return List of arguments
     */
    private fun calculatePlotArgs() : List<Double> {
        val delta = (array.maxRange - array.minRange)/400
        return List(400, {i -> array.minRange + delta*i})
    }
    /**
     * Calculates values of interpolation function to onClick them
     * @return List of values of interpolation function depending of argument
     */
    private fun calculatePlotVals() : List<Double> {
        //println(plotArgs)
        return List(400, {i -> calculate(plotArgs[i])})
    }
    /**
     * Measures error value by calculating difference between real and interpolated values
     * @return list of differences (realValue - interpolatedValue)
     */
    private fun calculateInfelicity() : List<Double> {
        return List(400, {i -> plotVals[i] - array.function(plotArgs[i])})
    }
    /**
     * Plot of interpolated function
     * @override Function pointsList of basic interface Drawable
     * @return Points of interpolated function onClick within the given range
     */
    override fun pointsList(): LineGraphSeries<DataPoint> {
        val returnList: Array<DataPoint> = Array(400, {i -> DataPoint(plotArgs[i],plotVals[i])})
        return LineGraphSeries(returnList)
    }
    /**
     * Plot of interpolated function infelicity
     * @return Points of interpolated function onClick within the given range
     */
    fun pointsInfelicityList() : LineGraphSeries<DataPoint> {
        val returnList: Array<DataPoint> = Array(400, {i ->
DataPoint(plotArgs[i],infelicityVals[i])})
        return LineGraphSeries(returnList)
    }
}
```

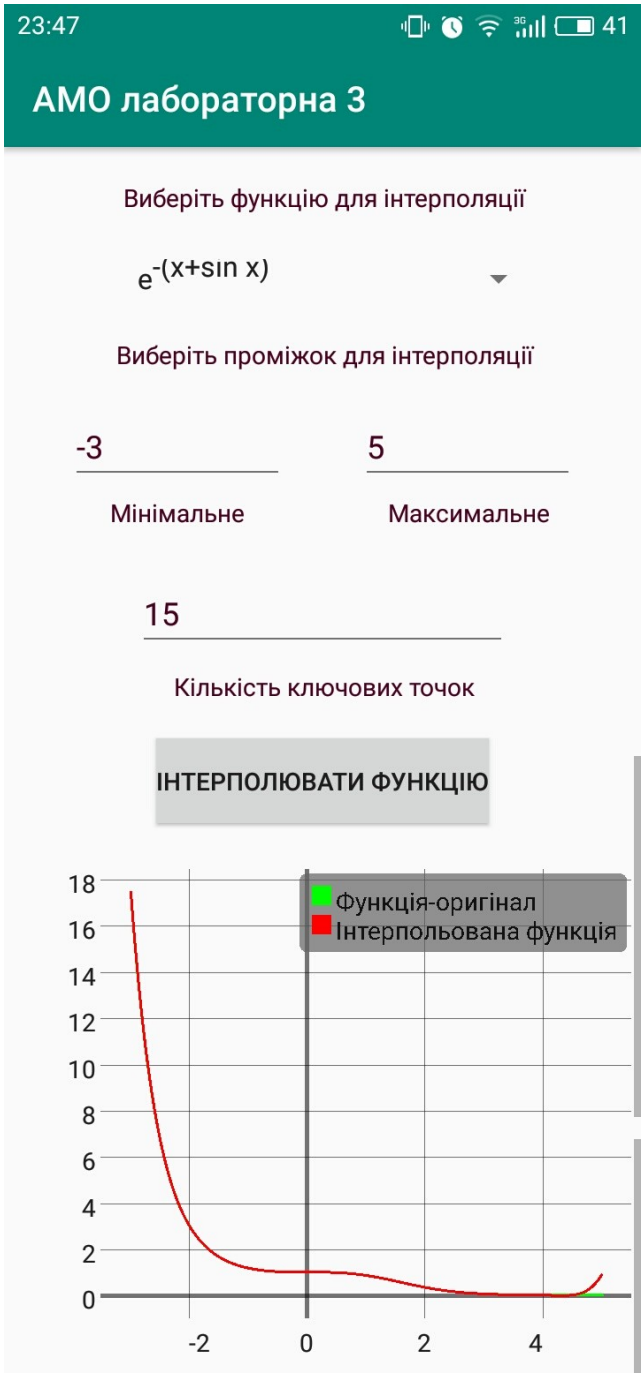
Functions.kt (функції, які треба інтерполювати):

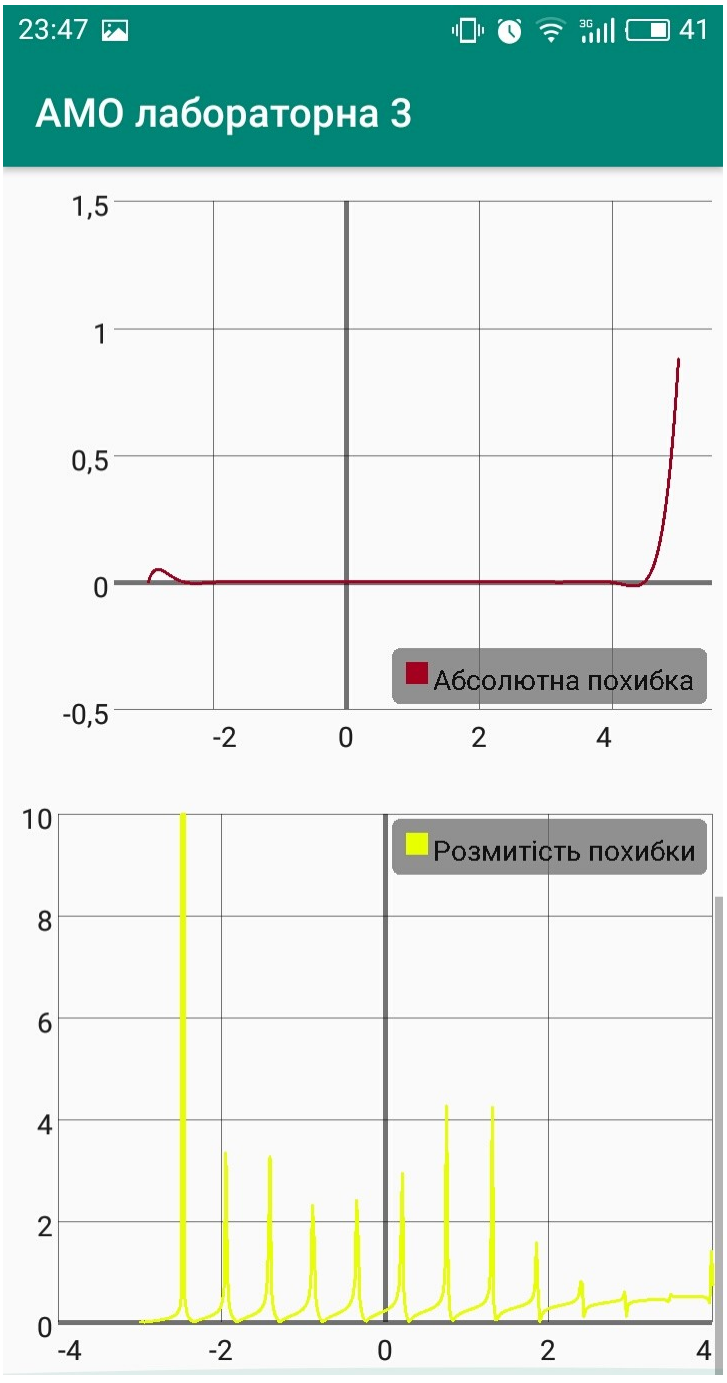
```
package com.labworks.amc_lab3
val ourFunc = {x: Double -> Math.exp(-x+Math.sin(x))}
val sin = {x: Double -> Math.sin(x)}
```

Алгоритм побудови інтерполяційного полінома Ньютона:


Початок
Ввід X, Y, n, x0;
k1 := 0;
i := 0; n-2
(x0 >= X[i]) and
(x0 < X[i + 1])
k1 := i;
break;
k := k1;
res1 := 999;
k <= (n - 2)
j := 0;
i := k; k1
j = 0
A[i, j] := (((x0 - X[i]) * Y[i + 1]) –
– ((x0 - X[i + 1]) * Y[i])) / (X[i + 1] - X[i])
A[i, j] := (((x0 - X[i]) * A[i + 1, j - 1]) –
– ((x0 - X[j + i + 1]) * A[i, j - 1])) / (X[j + i + 1] - X[i]);
j := j + 1;
j > 1
res2 := Abs(A[k1, j - 2] - A[k1, j-1]);
res2 < res1
res1 := res2
res := A[k1, j - 2];
break;
k := k + 1;
Вивід res;
Кінець

Результати виконання програми:





Висновок:

Під час даної лабораторної роботи мною були закріплені знання про інтерполяцію, та методи її реалізації; розроблено відповідну програму на основі алгоритму інтерполяції методом Ньютона; Результати успішної роботи програми наведені вище підтверджують правильність обраних мною рішень.