Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 4-5

з дисципліни «Числові методи»

Тема: «Інтерполяція та аппроксимація»

Виконав:

студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірила: доцент

Чуб Ольга Ігорівна

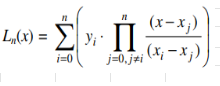
Харків – 2020

# ХОД РАБОТЫ

Основной целью данной работы является исследование алгоритмов интерполяции и аппроксимации функций, заданных таблично.

Входными данными являются функции, заданные в качестве набора пар координат.

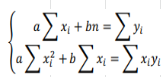
Интерполяция функции выполняется при помощи интерполяционного полинома Лагранжа. Программа спроектирована таким образом, что в состоянии вычислять полиномы любой степени в зависимости от входных данных. Полином Лагранжа для функции, заданной таблично считается по формуле:



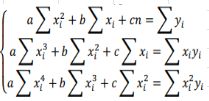
Для аппроксимации заданной функции используется метод наименьших квадратов. Высчитываются полиномы первой и второй степени, соответствующие уравнениям: ; .

Для определения коеффициентов при переменных программа решает следующие системы уравнений относительно переменных a, b, c по методу Гаусса:

Для полинома первой степени:

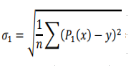


Для полинома второй степени:



После решения системы, найденные коэффициенты используются для вычисления значений полинома.

Для нахождения среднеквадратической ошибки используется следующая формула:



Программа для реализации данного метода была написана на языке Java. Решением задачи интерполяции занимается класс LagrangePolinom, который представляет собой объект, который создается на основе таблицы пар значений переменных. При помощи его метода Ln(x) вычисляется значение полинома в точке.

Для расчета аппроксимации заданной функции используется система классов. Абстрактный класс ApproximationPolinom представляет собой абстракную вычислимую машину и предоставляет методы для расчета среднеквадратической ошибки (так как она считается для любых полиномов одинаково, это реализовано на уровне абстракции) и вычисления всевозможных сумм. Конструктор этого класса принимает в себя таблицу координат, представленную в виде двух массивов, а так же обязывает любых его наследников реализовать абстрактый защищенный метод вычисления коэффициентов, который вызывается при конструировании объекта. Так же класс обязывает реализовать непосредственно метод вычисления значения полинома в точке. Таким образом, любой класс, наследующий ApproximationPolinom должен описать способ вычисления коэффициентов, который будет использоваться сразу же при конструировании этого объекта, и метод вычисления значения полинома.

Для реализации полинома первой степени используется класс FirstOrderPolinom. Класс реализует вычисление коэффициентов из системы, описанной выше при помощи класса Matrix, описанного в лабораторных работах 1-2. Система решается методом Гаусса, и найденные коеффициенты устанавливаются в поля класса при завершении конструирования объекта автоматически. После чего вычислить значение полинома в точке можно при помощи уже реализованного метода P(x).

Полином второй степени аналогичным образом реализован в классе SecondOrderPolinom, с поправкой на нахождение дополнительного коеффициента.

Для удобства пользования был создан графический интерфейс. Главное меню представляет из себя форму с двумя кнопками, при нажатии которых открываются формы для соответствующей задачи – интерполяции или аппроксимации. У формы интерполяции по центру располагается график функции, под ним – кнопки его регуляции, настраивающие диапазон аргумента. Вверху формы расположены поля для ввода порядка уравнения, и значений переменной и функции. По нажатию кнопки Solve в поля “L:” вставляются вычисленные значения полинома от значений переменной X. Так же данные значения помечаются на графике.

Форма аппроксимации реализована похожим образом, только вместо значений полинома в случае нажатия кнопки Solve выводятся уравнения полиномов, а вычисленные точки отображаются на графике.

На все поля для ввода предусмотрена валидация: процесс решения не начнется если введены неверные данные или данные введены не полностью и будет выведено соответствующее сообщение.

В качестве значений по умолчанию в фрейме установлены коэффициенты, предоставленные преподавателем.

Программа скомпилирована в исполняемый файл lab4\_InterpolationApproximation.exe.

# ВЫВОДЫ

Таким образом, в ходе выполнения данной лабораторной работы были рассмотрены алгоритмы вычисления интерполяции функции полиномом Лагранжа и аппроксимации методом наименьших квадратов.

# ЛИСТИНГ ПРОГРАММЫ

Класс LagrangePolinom, инкапсулирующий полином Лагранжа:

**import** java.util.Arrays;

**public** **class** LagrangePolinom {

**private** **double**[] X;

**private** **double**[] Y;

**private** **int** n;

**public** LagrangePolinom(**double**[] X, **double**[] Y) {

**this**.n = X.length;

**this**.X = Arrays.*copyOf*(X, n);

**this**.Y = Arrays.*copyOf*(Y, n);

}

**public** **double** Ln(**double** x) {

**double** result = 0.0;

**for**(**int** i = 0; i < n; i++) {

**double** subResult = 1;

**for** (**int** j = 0; j < n; j++) {

**if**(j != i) {

subResult \*= (x - X[j])/(X[i] - X[j]);

}

}

result += Y[i] \* subResult;

}

**return** result;

}

}

Класс ApproximationPolinom, инкапсулирующий любой абстрактный аппроксимирующий полином.

**import** java.util.Arrays;

**public** **abstract** **class** ApproximationPolinom {

**protected** **double**[] X;

**protected** **double**[] Y;

**protected** **int** n;

**public** ApproximationPolinom(**double**[] X, **double**[] Y) {

**this**.n = X.length;

**this**.X = Arrays.*copyOf*(X, n);

**this**.Y = Arrays.*copyOf*(Y, n);

evaluateCoefficients();

}

**public** **abstract** **double** P(**double** x);

**public** **double** getRootMeanSquareError() {

**double** result = 0.0;

**for** (**int** i = 0; i < n; i++) {

result += Math.*pow*(P(X[i]) - Y[i], 2);

}

**return** Math.*sqrt*(result / n);

}

**protected** **abstract** **void** evaluateCoefficients();

**protected** **double** sum(**double**[] arr) {

**return** sum(arr, 1);

}

**protected** **double** sum(**double**[] arr, **int** power) {

**double** sum = 0.0;

**for** (**int** i = 0; i < n; i++) {

sum += Math.*pow*(arr[i], power);

}

**return** sum;

}

**protected** **double** sum(**int** xPower, **int** yPower) {

**double** sum = 0.0;

**for** (**int** i = 0; i < n; i++) {

sum += Math.*pow*(X[i], xPower) \* Math.*pow*(Y[i], yPower);

}

**return** sum;

}

}

Класс FirstOrderPolinom, инкапсулирующий аппроксимирующий полином первой степени.

**public** **class** FirstOrderPolinom **extends** ApproximationPolinom {

**protected** **double** a;

**protected** **double** b;

**public** FirstOrderPolinom(**double**[] X, **double**[] Y) {

**super**(X, Y);

}

@Override

**public** **double** P(**double** x) {

**return** a\*x + b;

}

@Override

**protected** **void** evaluateCoefficients() {

Row result;

Row[] matrix = **new** Row[] {

**new** Row(sum(X), n, sum(Y)),

**new** Row(sum(X, 2), sum(X), sum(1, 1))

};

**if**(sum(X) == 0) {

result = **new** Matrix(matrix[1], matrix[0]).solveByGauss();

}

**else** {

result = **new** Matrix(matrix[0], matrix[1]).solveByGauss();

}

a = result.elem(0).doubleValue();

b = result.elem(1).doubleValue();

}

@Override

**public** String toString() {

**return** "P1(x) = "+String.*format*("%.3f", a)+"x + "+String.*format*("%.3f", b);

}

}

Класс SecondOrderPolinom, инкапсулирующий аппроксимирующий полином второй степени.

**public** **class** SecondOrderPolinom **extends** FirstOrderPolinom {

**protected** **double** c;

**public** SecondOrderPolinom(**double**[] X, **double**[] Y) {

**super**(X, Y);

}

@Override

**public** **double** P(**double** x) {

**return** a\*x\*x + b\*x + c;

}

@Override

**protected** **void** evaluateCoefficients() {

Row result;

Row[] matrix = **new** Row[] {

**new** Row(sum(X, 2), sum(X), n, sum(Y)),

**new** Row(sum(X, 3), sum(X, 2), sum(X), sum(1, 1)),

**new** Row(sum(X, 4), sum(X, 3), sum(X, 2), sum(2, 1))

};

**if**(sum(X, 2) == 0) {

**if**(sum(X, 3) == 0) {

result = **new** Matrix(matrix[2], matrix[0], matrix[1]).solveByGauss();

}

**else** {

result = **new** Matrix(matrix[1], matrix[0], matrix[2]).solveByGauss();

}

}

**else** {

result = **new** Matrix(matrix[0], matrix[1], matrix[2]).solveByGauss();

}

a = result.elem(0).doubleValue();

b = result.elem(1).doubleValue();

c = result.elem(2).doubleValue();

}

@Override

**public** String toString() {

**return** "P2(x) = "+String.*format*("%.3f", a)+"x^2 + "+String.*format*("%.3f", b)+"x + "+String.*format*("%.3f", c);

}

}

Классы графического интерфейса реализованы на основе библиотеки swing и не обладают функционалом, необходимым к представлению в данной работе, поэтому в листинге не приводятся, но предоставляются вместе со всеми исходными и исполняемыми файлами преподавателю.