Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 6

з дисципліни «Числові методи»

Тема: «Чисельне інтегрування»

Виконав:

студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірила: доцент

Чуб Ольга Ігорівна

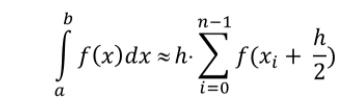
Харків – 2021

# ХОД РАБОТЫ

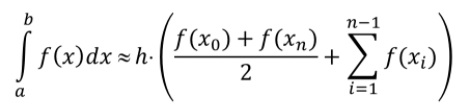
Основной целью данной работы является исследование алгоритмов численного интегрирования при помощи методов средних квадратов, трапеций и метода Симпсона.

Входными данными являются любые интегрируемые функции одной переменной и пределы интегрирования, на которых заданные функции определены, непрерывны и интегрируемы.

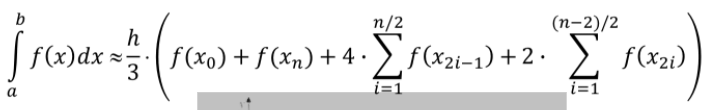
Метод средних прямоугольников численного дифференцирования заключается в разбиении отрезка интегрирования на определенное количество промежутков (в случае таблично заданной функции, на количество точек). После этого из этих промежутков формируются прямоугольники со стороной, равной шагу и высотой, равной значению функции в точке, и вычисляется их суммарная площадь. Ввиду того, что в результате получается многоугольник, приближенный к кривой, метод имеет погрешности в вычислении. Общая формула вычисления интеграла по методу средних квадратов:



Метод трапеций тоже является приближенным. Он отличается от метода прямоугольников тем, что значения функции соединяются линейными отрезками, формирующими условные трапеции на интегрируемой области. Общая формула вычисления интеграла по методу трапеций:



Метод Симпсона является еще более точным, чем два предыдущих, так как для вычисления площади каждого разбиения использует не линейные функции, а фрагменты параболы, что лучше приближает функцию к форме кривой. Метод использует отдельно суммы четных и нечетных элементов разбиения. Общая формула вычисления интеграла по методу Симпсона:



Программа для реализации данного метода была написана на языке Java. Для начала был создан интерфейс Function с единственным методом f(x), который нужно было определить любому классу реализующему интерфейс функции. В программе используется несколько реализаций данного интерфейса: FirstVariantFunction (функция по номеру варианта в списке группы), ExampleFunction (фукнция, аналогичная функции, предоставленной преподавателем в примере) и AnalyticFunction, функция, которую можно задавать в текстовом виде.

Основными расчётами занимается класс Integral. Его конструктор принимает в себя объект функции, для которой нужно вычислять интеграл, а его методы solveBySquresMethod(a, b), solveByTrapezeMethod(a, b) и solveBySimpsonMethod(a, b) решают интеграл указанным методом в заданных пределах интегрирования.

Для удобства пользования был создан графический интерфейс. Предусмотрены поля для ввода подынтегральной функции и пределов интегрирования. Вычисление запускается после нажатия кнопки Solve или клавиши Enter на клавиатуре в фокусе поля функции.

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 числа и функция, предоставленные преподавателем.

Программа скомпилирована в исполняемый файл lab6\_NumericIntergation.exe.

# ВЫВОДЫ

Таким образом, в ходе выполнения данной лабораторной работы были рассмотрены алгоритмы численного интегрирования заданной функции методами средних квадратов, трапеций и Симпсона. Рабочая версия программы скомпилирована и предоставлена преподавателю.

# ЛИСТИНГ ПРОГРАММЫ

Интерфейс Function, задающий обязанность реализации вычисления метода функции:

**public** **interface** Function {

**double** f(**double** x);

}

Класс FirstVariantFunction, инкапсулирующий функцию, заданную в условии лабораторный работы (вариант - 1):

**public** **class** FirstVariantFunction **implements** Function {

@Override

**public** **double** f(**double** x) {

**return** (1.5\*x\*x + x)/(Math.*pow*(x, 5) + 1);

}

@Override

**public** String toString() {

**return** "(1.5x^2 + x) / (x^5 + 1)";

}

}

Класс ExampleFunction, инкапсулирующий функцию, приведенную преподавателем в качестве примера:

**public** **class** ExampleFunction **implements** Function {

@Override

**public** **double** f(**double** x) {

**return** (x+1)\*Math.*sin*(x);

}

@Override

**public** String toString() {

**return** "(x + 1) \* sin(x)";

}

}

Класс AnalyticFunction, инкапсулирующий функцию, которую можно получить из текста:

**import** edu.hws.jcm.data.Expression;

**import** edu.hws.jcm.data.Parser;

**import** edu.hws.jcm.data.Variable;

**public** **class** AnalyticFunction **implements** Function {

**private** Expression fun;

**private** Parser parser;

**private** Variable var;

**private** Variable par;

**public** AnalyticFunction() {

parser = **new** Parser(Parser.***STANDARD\_FUNCTIONS***);

var = **new** Variable("x");

parser.add(var);

fun = parser.parse("x");

}

/\*\*

\* Creates new function of variable var

\* **@param** expr - new function expression of variable var

\* **@param** var - variable that the function depends on

\*/

**public** AnalyticFunction(String expr, String var) **throws** edu.hws.jcm.data.ParseError {

parser = **new** Parser(Parser.***STANDARD\_FUNCTIONS***);

**this**.var = **new** Variable(var);

parser.add(**this**.var);

fun = parser.parse(expr);

}

/\*\*

\* Creates new function of variable var with parameter par

\* **@param** expr - new function expression of variable var

\* **@param** var - variable that the function depends on

\* **@param** par - parameter specified in function

\*/

**public** AnalyticFunction(String expr, String var, String par) {

parser = **new** Parser(Parser.***STANDARD\_FUNCTIONS***);

**this**.var = **new** Variable(var);

**this**.par = **new** Variable(par);

parser.add(**this**.var);

parser.add(**this**.par);

fun = parser.parse(expr);

}

/\*\*

\* Sets the new function expression of current variable

\* **@param** fun - new function expression

\*/

**public** **void** setFunction(String fun) {

**this**.fun = parser.parse(fun);

}

/\*\*

\* Sets the new function expression of new variable

\* **@param** fun - new function expression

\* **@param** var - new variable

\*/

**public** **void** setFunction(String fun, String var) {

parser.remove(**this**.var.toString());

**this**.var = **new** Variable(var);

parser.add(**this**.var);

**this**.fun = parser.parse(fun);

}

/\*\*

\* Sets the new function expression of new variable with new parameter

\* **@param** fun - new function expression

\* **@param** var - new variable

\* **@param** par - new parameter

\*/

**public** **void** setFunction(String fun, String var, String par) {

parser.remove(**this**.var.toString());

**if**(**this**.par != **null**)

parser.remove(**this**.par.toString());

**this**.var = **new** Variable(var);

**this**.par = **new** Variable(par);

parser.add(**this**.var);

parser.add(**this**.par);

**this**.fun = parser.parse(fun);

}

/\*\*

\* **@return** current variable symbol

\*/

**public** String getVariable() {

**return** var.toString();

}

/\*\*

\* **@return** the Variable object of current variable

\*/

**public** Variable getVariableObject() {

**return** var;

}

/\*\*

\* sets the new variable of function

\* **@param** var - new variable symbol

\*/

**public** **void** setVariable(String var) {

**this**.var = **new** Variable(var);

}

/\*\*

\* **@return** current parameter symbol or null

\*/

**public** String getParameter() {

**if**(par == **null**)

**return** **null**;

**return** par.toString();

}

/\*\*

\* **@return** the Variable object of current parameter

\*/

**public** Variable getParameterObject() {

**return** par;

}

/\*\*

\* sets the new parameter of function

\* **@param** par - new parameter symbol

\*/

**public** **void** setParameter(String par) {

**this**.par = **new** Variable(par);

}

/\*\*

\* gets current parameter value of function

\* **@return** - paramater value

\*/

**public** **double** getParameterValue() {

**return** par.getVal();

}

/\*\*

\* sets new parameter value to current parameter

\* **@param** param - new value of current parameter

\*/

**public** **void** setParameterValue(**double** param) {

par.setVal(param);

}

/\*\*

\* turn the variable to the parameter and parameter to variable

\*/

**public** **void** swapVariableAndParameter(){

String p = getParameter();

String v = getVariable();

parser.remove(p);

parser.remove(v);

var = **new** Variable(p);

par = **new** Variable(v);

parser.add(var);

parser.add(par);

fun = parser.parse(fun.toString());

}

@Override

**public** String toString() {

**return** fun.toString();

}

@Override

**public** **double** f(**double** x) {

var.setVal(x);

**return** fun.getVal();

}

**public** AnalyticFunction deritative() {

**if**(par == **null**) **return** **new** AnalyticFunction(fun.derivative(var).toString(), var.toString());

**else** **return** **new** AnalyticFunction(fun.derivative(var).toString(), var.toString(), par.toString());

}

}

Класс Integral, инкапсулирующий интеграл для заданного подынтегрального выражения:

**public** **class** Integral {

**private** Function function;

**public** Integral(Function function) {

**this**.function = function;

}

**public** **double** solveBySquaresMethod(**double** a, **double** b) {

**int** n = 10;

**double** h = (b-a)/n;

**double** result = 0.0;

**for** (**double** x = a; x <= b; x += h) {

result += function.f(x + h/2);

}

**return** h \* result;

}

**public** **double** solveByTrapezeMethod(**double** a, **double** b) {

**int** n = 10;

**double** h = (b-a)/n;

**double** coeff = (function.f(a) + function.f(b)) / 2;

**double** result = 0.0;

**for** (**double** x = a + h; x < b; x += h) {

result += function.f(x);

}

**return** h \* (coeff + result);

}

**public** **double** solveBySimpsonMethod(**double** a, **double** b) {

**int** n = 10;

**double** h = (b-a)/n;

**double** x\_0 = function.f(a);

**double** x\_n = function.f(b);

**double** x\_odd = 0.0;

**for** (**double** x = a + h; x < b; x += 2 \* h) {

x\_odd += function.f(x);

}

**double** x\_even = 0.0;

**for** (**double** x = a + 2 \* h; x < b; x += 2 \* h) {

x\_even += function.f(x);

}

**return** h/3 \* (x\_0 + x\_n + 4 \* x\_odd + 2 \* x\_even);

}

}

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.