

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева»**

ИИТК/09.03.01/Информатика и вычислительная техника/Автоматизированные системы обработки информации и управления
институт/ факультет/ подразделение
Информатики и вычислительной техники
кафедра/ цикловая комиссия

ОПРЕДЕЛЕННЫЕ ИНТЕГРАЛЫ

Преподаватель

Обучающиеся группы БИА21-01

подпись, дата

Семенкина М. Е.

Путинцев А.Ю.
Юрченко М.А.
Турсунов Д.М
Сухарева А.С.

Красноярск 2022

ВВЕДЕНИЕ

Цель работы: Изучение возможных путей для вычисления определенных интегралов с помощью компьютера.

Задачи:

1. Реализовать численное нахождение определенных интегралов от произвольной функции (квадратурные формулы треугольников, трапеций, Симпсона).
2. Выполнить тестирование на 8 различных тестовых интегралах. Сравнить погрешности полученных результатов. (Интегралы для точной оценки можно вычислить вручную)
3. Построить графики функций и квадратур. Сравнить погрешности полученных для различного числа точек, разбивающих отрезок.

ХОД РАБОТЫ

1. Сначала объявляем меню для выбора функций-формул.

исполняемый файл integral.cpp

```
double function(double x, int s) {  
    switch (s)  
    {  
        case 1:  
            return x;  
        case 2:  
            return x+1;  
        case 3:  
            return sqrt(x);  
        case 4:  
            return pow(x, 2);  
        case 5:  
            return pow(x,3);  
        case 6:  
            return sin(x);  
        case 7:  
            return cos(x);  
        case 8:  
            return pow(sin(x)-cos(x),2);  
        default:  
            return 0.0;  
    }  
}
```

2. Пишем код для нахождения определенных интегралов от произвольной функции квадратурной формулы прямоугольников. Код приведен ниже:

```
double sumofInt(double lbound, double ubound, int n, double dx, int  
s) {  
    double recarea;  
    double funcvalue;  
    double xi;  
    double sum = 0;  
    if (s == 3) {  
        for (int i = 1; i < n; i++) {  
            xi = lbound + i * dx;
```

```

        if (xi == 0) {
            recarea = 0;
        }
        else {
            funcvalue = function(xi, s);
            recarea = funcvalue * dx;
        }
        sum += recarea;
    }
}
else {
    for (int i = 1; i < n; i++) {
        double xi = lbound + i * dx;
        double funcvalue = function(xi, s);
        double recarea = funcvalue * dx;
        sum += recarea;
    }
}
sum = round(sum * 1000000000000) / 1000000000000;
return sum;
}

```

3. Пишем код для нахождения определенных интегралов от произвольной функции квадратурной формулы трапеций. Код приведен ниже:

```

double sumofIntTrap(double lbound, double ubound, int n,
double dx, int s) {
    double sum = 0;
    double f1 = function(lbound,s);
    double f2 = function(ubound,s);
    double tf;
    double xi;
    if (s == 3) {
        for (int i = 1; i < n; i++) {
            xi = lbound + (i * dx);
            if (xi == 0) {
                tf = 0;
            }
            else {
                tf = function(xi, s);
            }
        }
    }
}

```

```

        tf = tf * 2;
    }
    sum = sum + tf;
}
}
else {
for (int i = 1; i < n; i++) {
    xi = lbound + (i * dx);
    tf = function(xi, s);
    tf = tf * 2;
    sum = sum + tf;
}
}
sum = sum + f1 + f2;
sum = (dx * sum) / 2;
sum = round(sum * 100000000000) / 100000000000;
return sum;
}

```

4. Пишем код для нахождения определенных интегралов от произвольной функции Симпсона. Код приведен ниже:

```

double sumofIntSimp(double lbound, double ubound, int n,
double dx, int s) {
    double sum = 0;
    double f1 = function(lbound, s);
    double f2 = function(ubound, s);
    bool flag = 0;
    double tf;
    double xi;

    if (s == 3) {
        for (int i = 1; i < n; i++) {
            xi = lbound + (i * dx);
            if (flag == 0) {
                if (xi == 0) {
                    tf = 0;

```

```

    }
    else {
        tf = function(xi, s);
        tf = tf * 4;
    }
    sum = sum + tf;
    flag = 1;
}
else {
    if (xi == 0) {
        tf = 0;
    }
    else {
        tf = function(xi, s);
        tf = tf * 2;
    }
    sum = sum + tf;
    flag = 0;
}
}
}
else {
    for (int i = 1; i < n; i++) {
        xi = lbound + (i * dx);
        if (flag == 0) {
            tf = function(xi, s);
            tf = tf * 4;
            sum = sum + tf;
            flag = 1;
        }
        else {
            tf = function(xi, s);
            tf = tf * 2;
            sum = sum + tf;
            flag = 0;
        }
    }
}
}

```

```

    sum = sum + f1 + f2;
    sum = (dx * sum) / 3;
    sum = round(sum * 100000000000) / 100000000000;
    return sum;
}

```

исполняемый файл integralsUI.h

// Получение вводимых данных

```

n = Double::Parse(textBox6->Text);
ubound = Double::Parse(textBox1->Text);
lbound = Double::Parse(textBox2->Text);

```

// Визуализация графика функции

```

dx = (double)(ubound - lbound) / n;
double step = 0.01;
a = lbound - 25;
b = ubound + 25;
x = a;
if (s == 3) {
    while (x < 0) {
        y = function(x, s);
        this->chart1->Series[0]->Points->AddXY(x, y);
        x += step;
    }
    x = x + step;
    while (x < b) {
        y = function(x, s);
        this->chart1->Series[1]->Points->AddXY(x, y);
        x += step;
    }
}
else {
    while (x <= b) {
        y = function(x, s);
        this->chart1->Series[0]->Points->AddXY(x, y);
        x += step;
    }
}

```

//Визуализация квадратуры Симпсона

// Метод Крамера (Метод крамера используется в данной

программе чтобы найти коэффициенты А, В и С для параболы по трем точкам)

```
double x0, y0, x1, y1, x2, y2;
double A, B, C;
for (int j = 0; j < n / 2; j++) {
    x0 = lbound + j * dx * 2;
    y0 = function(x0, s);
    x1 = x0 + dx;
    y1 = function(x1, s);
    x2 = x0 + (2 * dx);
    y2 = function(x2, s);

    double mx[3][3];
    double d1[3][3];
    double d2[3][3];
    double d3[3][3];
    mx[0][0] = pow(x0, 2);
    mx[0][1] = x0;
    mx[0][2] = 1;
    mx[1][0] = pow(x1, 2);
    mx[1][1] = x1;
    mx[1][2] = 1;
    mx[2][0] = pow(x2, 2);
    mx[2][1] = x2;
    mx[2][2] = 1;

    d1[0][0] = y0;
    d1[0][1] = x0;
    d1[0][2] = 1;
    d1[1][0] = y1;
    d1[1][1] = x1;
    d1[1][2] = 1;
    d1[2][0] = y2;
    d1[2][1] = x2;
    d1[2][2] = 1;

    d2[0][0] = pow(x0, 2);
    d2[0][1] = y0;
    d2[0][2] = 1;
    d2[1][0] = pow(x1, 2);
    d2[1][1] = y1;
    d2[1][2] = 1;
    d2[2][0] = pow(x2, 2);
    d2[2][1] = y2;
    d2[2][2] = 1;

    d3[0][0] = pow(x0, 2);
    d3[0][1] = x0;
    d3[0][2] = y0;
    d3[1][0] = pow(x1, 2);
```



```

d3[1][1] = x1;
d3[1][2] = y1;
d3[2][0] = pow(x2, 2);
d3[2][1] = x2;
d3[2][2] = y2;

double opmx, opd1, opd2, opd3;
opmx = mx[0][0] * ((mx[1][1] * mx[2][2]) - (mx[1][2] * mx[2][1])) -
mx[0][1] * ((mx[1][0] * mx[2][2]) - (mx[1][2] * mx[2][0])) + mx[0][2] *
((mx[1][0] * mx[2][1]) - (mx[1][1] * mx[2][0]));
opd1 = d1[0][0] * ((d1[1][1] * d1[2][2]) - (d1[1][2] * d1[2][1])) -
d1[0][1] * ((d1[1][0] * d1[2][2]) - (d1[1][2] * d1[2][0])) + d1[0][2] *
((d1[1][0] * d1[2][1]) - (d1[1][1] * d1[2][0]));
opd2 = d2[0][0] * ((d2[1][1] * d2[2][2]) - (d2[1][2] * d2[2][1])) -
d2[0][1] * ((d2[1][0] * d2[2][2]) - (d2[1][2] * d2[2][0])) + d2[0][2] *
((d2[1][0] * d2[2][1]) - (d2[1][1] * d2[2][0]));
opd3 = d3[0][0] * ((d3[1][1] * d3[2][2]) - (d3[1][2] * d3[2][1])) -
d3[0][1] * ((d3[1][0] * d3[2][2]) - (d3[1][2] * d3[2][0])) + d3[0][2] *
((d3[1][0] * d3[2][1]) - (d3[1][1] * d3[2][0]));

A = opd1 / opmx;
B = opd2 / opmx;
C = opd3 / opmx;

rabotka1515::MyForm::chart1->Series->Add(j.ToString());
this->chart1->Series[j.ToString()]->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Spline;
this->chart1->Series[j.ToString()]->BorderWidth = 2;
this->chart1->Series[j.ToString()]->BorderDashStyle =
System::Windows::Forms::DataVisualization::Charting::ChartDashStyle::Dot;

for (x = x0; x <= x2; x = x + (dx * 0.1)) {
    y = (A * pow(x, 2) + B * x + C);
    this->chart1->Series[j.ToString()]->Points->AddXY(x, y);
}

}
sernum = n / 2;
seriesflag = 1;
for (x = lbound; x <= ubound - dx; x = x + dx) {
    y = 0;
    this->chart1->Series[7]->Points->AddXY(x, y);
    y = function(x, s);
    this->chart1->Series[7]->Points->AddXY(x, y);
    y = function(x + dx, s);
    x = x + dx;
    this->chart1->Series[7]->Points->AddXY(x, y);
    x = x - dx;
}

```

// Визуализация квадратуры прямоугольников

```
for (x = lbound; x <= ubound - dx; x = x + dx) {
    y = 0;
    this->chart1->Series[2]->Points->AddXY(x, y);
    if (function(x,s)<0) {
        // y<0 y>
        if (function(x + dx, s) > function(x, s)) {
            if (function(x + dx, s) >= 0) {
                y = 0;
                this->chart1->Series[2]->Points->AddXY(x, y);
                x = x + dx;
                this->chart1->Series[2]->Points->AddXY(x, y);
                x = x - dx;
            }
            else {
                y = function(x + dx, s);
                this->chart1->Series[2]->Points->AddXY(x, y);
                x = x + dx;
                this->chart1->Series[2]->Points->AddXY(x, y);
                x = x - dx;
            }
        }
        // y<0 y<
        else if (function(x + dx, s) <= function(x, s)) {
            y = function(x, s);
            this->chart1->Series[2]->Points->AddXY(x, y);
            x = x + dx;
            this->chart1->Series[2]->Points->AddXY(x, y);
            x = x - dx;
        }
    }
    else {
        //y>0 y>
        if (function(x + dx, s) > function(x, s)) {
            y = function(x, s);
            this->chart1->Series[2]->Points->AddXY(x, y);
            x = x + dx;
            this->chart1->Series[2]->Points->AddXY(x, y);
            x = x - dx;
        }
        //y>0 y<
        else if (function(x + dx, s) <= function(x, s)) {
            if (function(x + dx, s) <= 0) {
                y = 0;
                this->chart1->Series[2]->Points->AddXY(x, y);
                x = x + dx;
                this->chart1->Series[2]->Points->AddXY(x, y);
            }
        }
    }
}
```

```

        x = x - dx;
    }
    else {
        y = function(x + dx, s);
        this->chart1->Series[2]->Points->AddXY(x, y);
        x = x + dx;
        this->chart1->Series[2]->Points->AddXY(x, y);
        x = x - dx;
    }
}

}

}

```

// Визуализация границ

```

for (int i = -50; i < 50; i++) {
    x = lbound;
    y = i;
    this->chart1->Series[3]->Points->AddXY(x, y);
    x = ubound;
    this->chart1->Series[4]->Points->AddXY(x, y);
}

```

// Вывод вычислений из integral.cpp

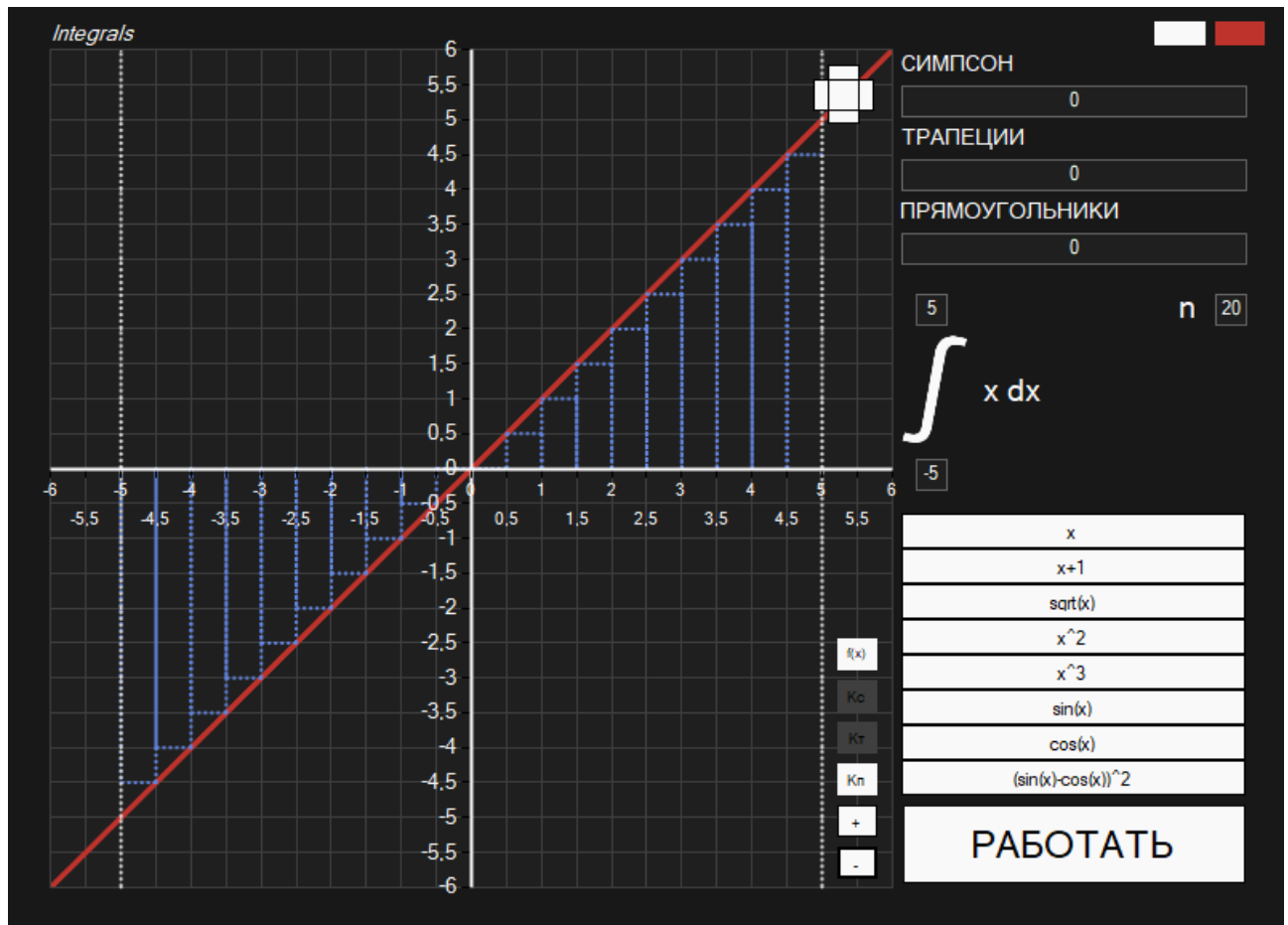
```

this->textBox5->Text = sumofIntSimp(lbound, ubound, n, dx, s).ToString();
this->textBox4->Text = sumofIntTrap(lbound, ubound, n, dx, s).ToString();
this->textBox3->Text = sumofInt(lbound, ubound, n, dx, s).ToString();

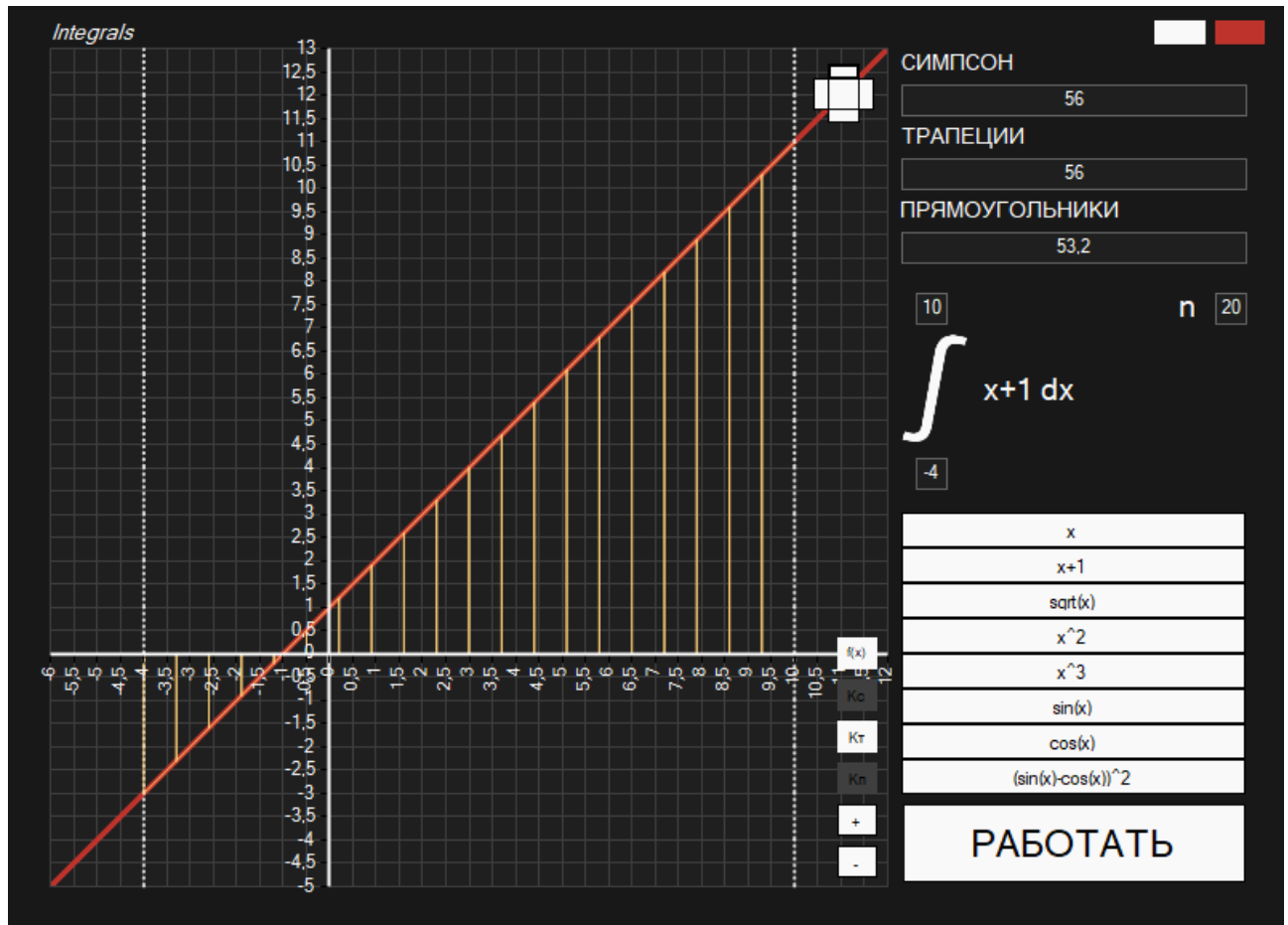
```

Графики:

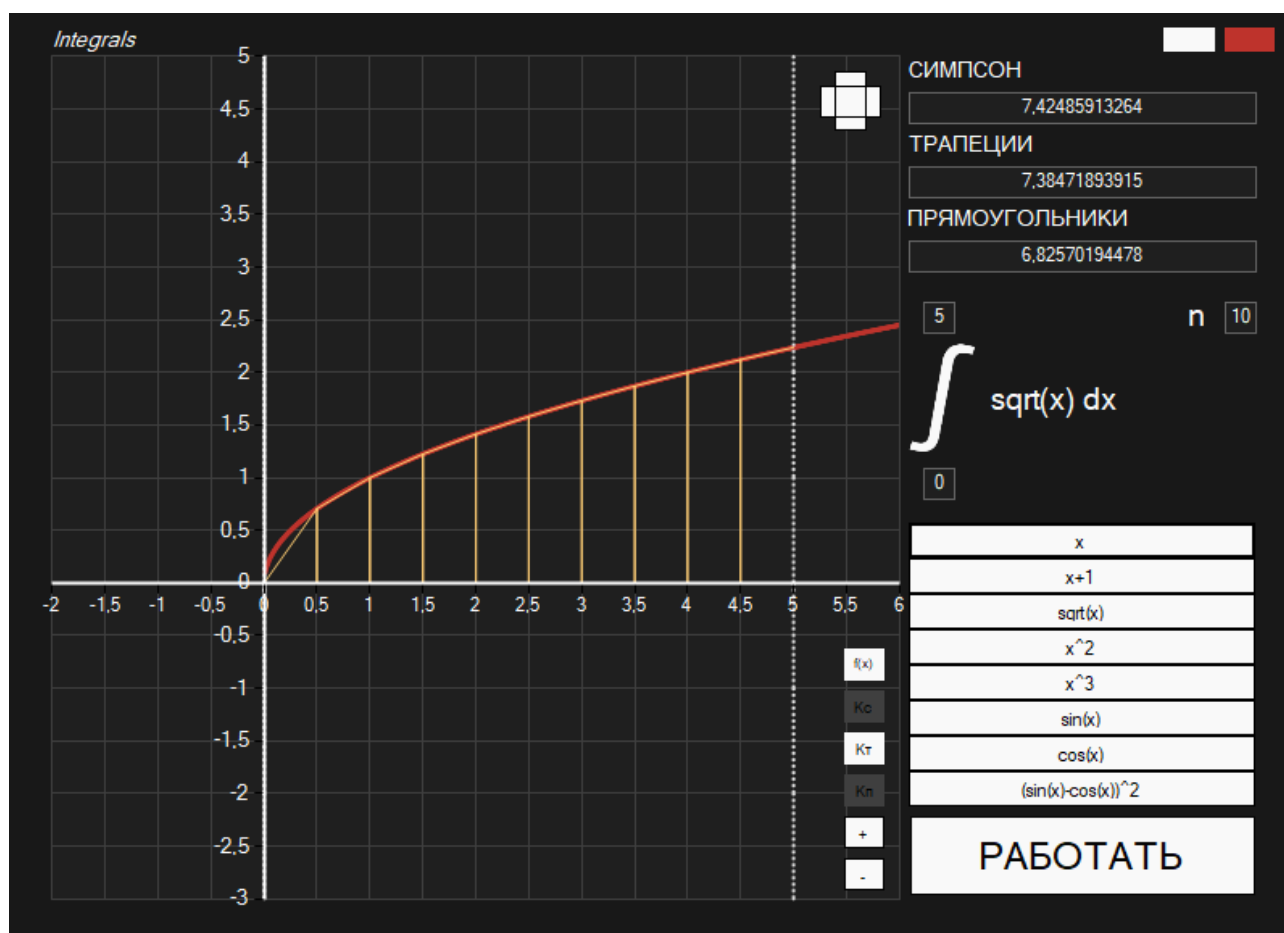
1. Пример работы интеграла от функции x



2. Пример работы интеграла от функции $x+1$

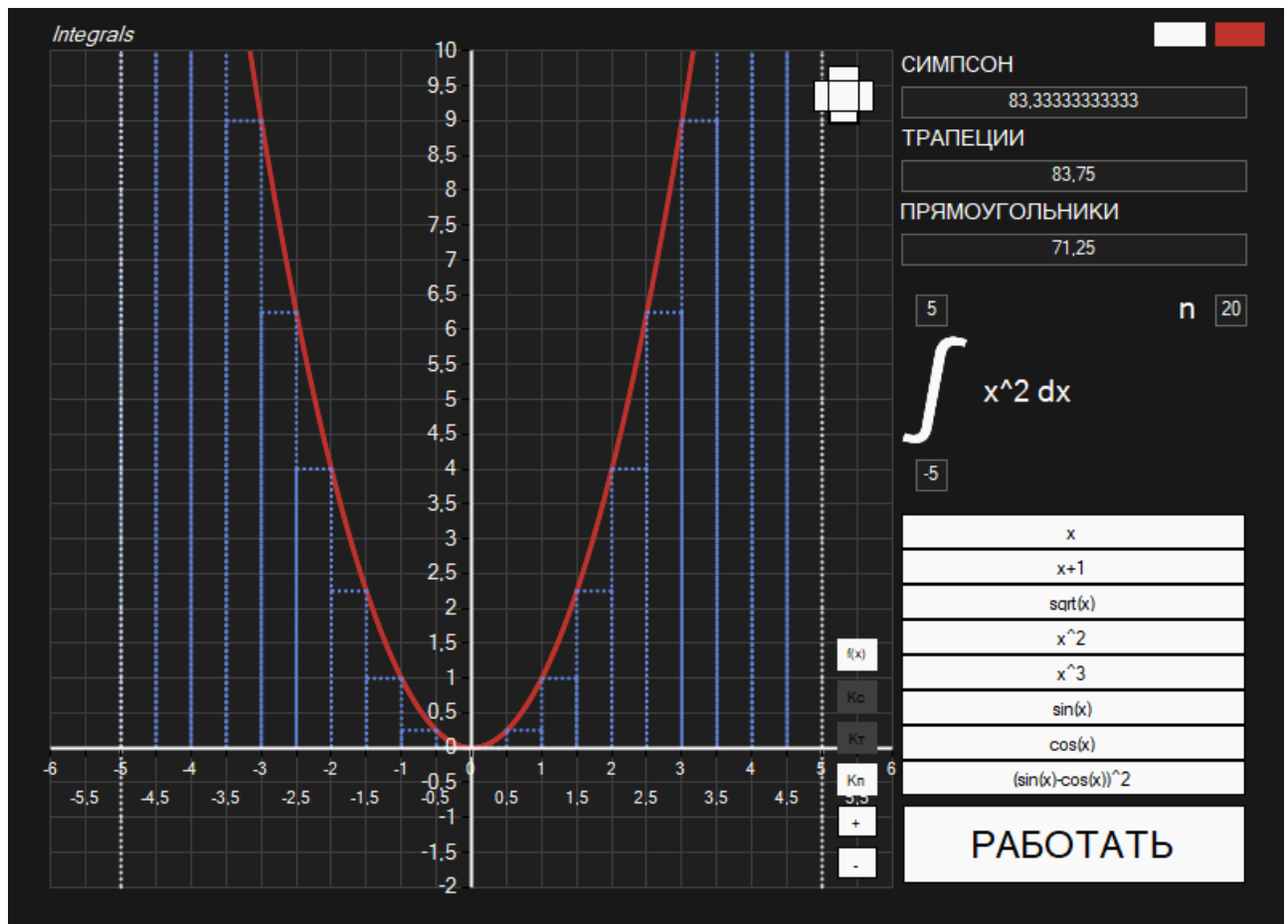


3. Пример работы интеграла от функции \sqrt{x}

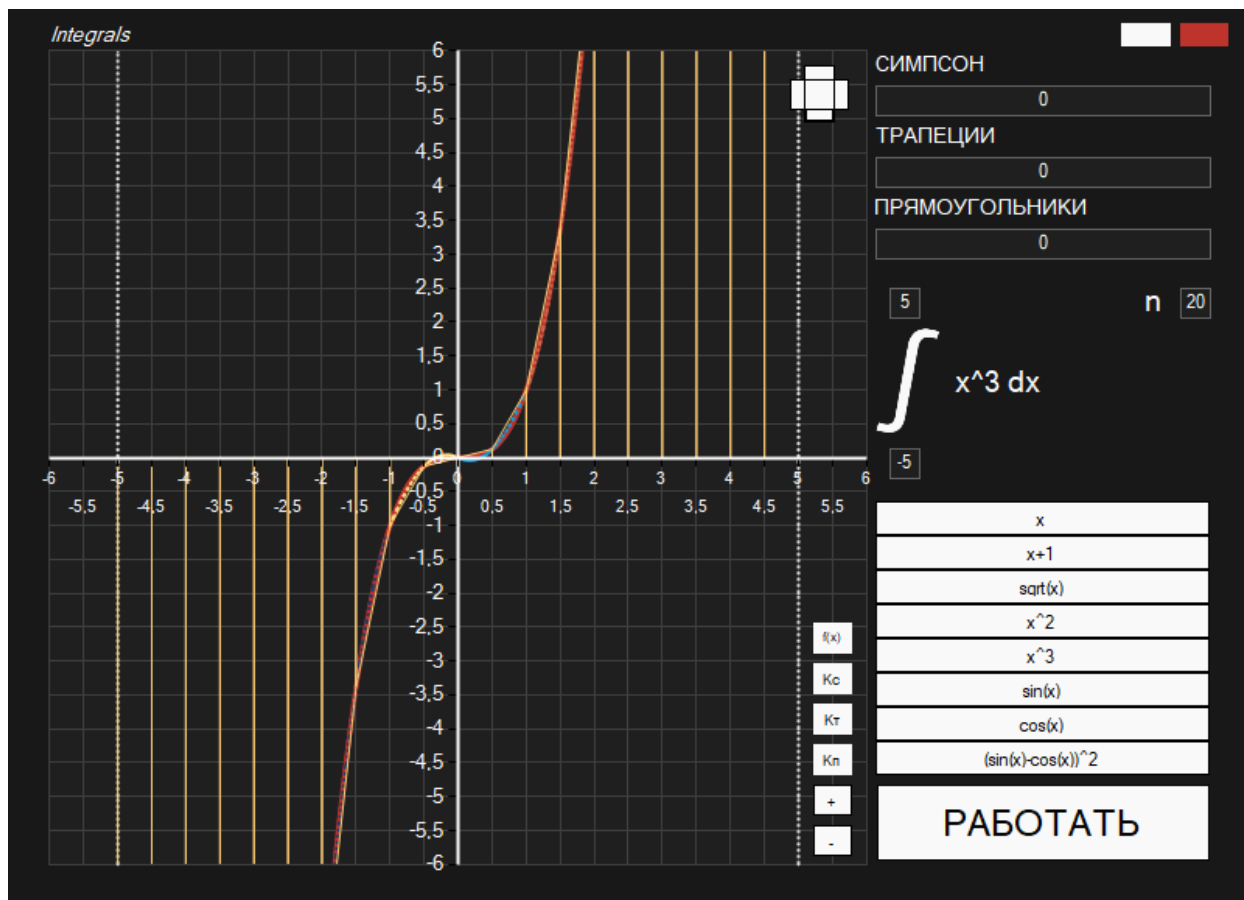


4. Пример работы интеграла от функции

$$x^2$$

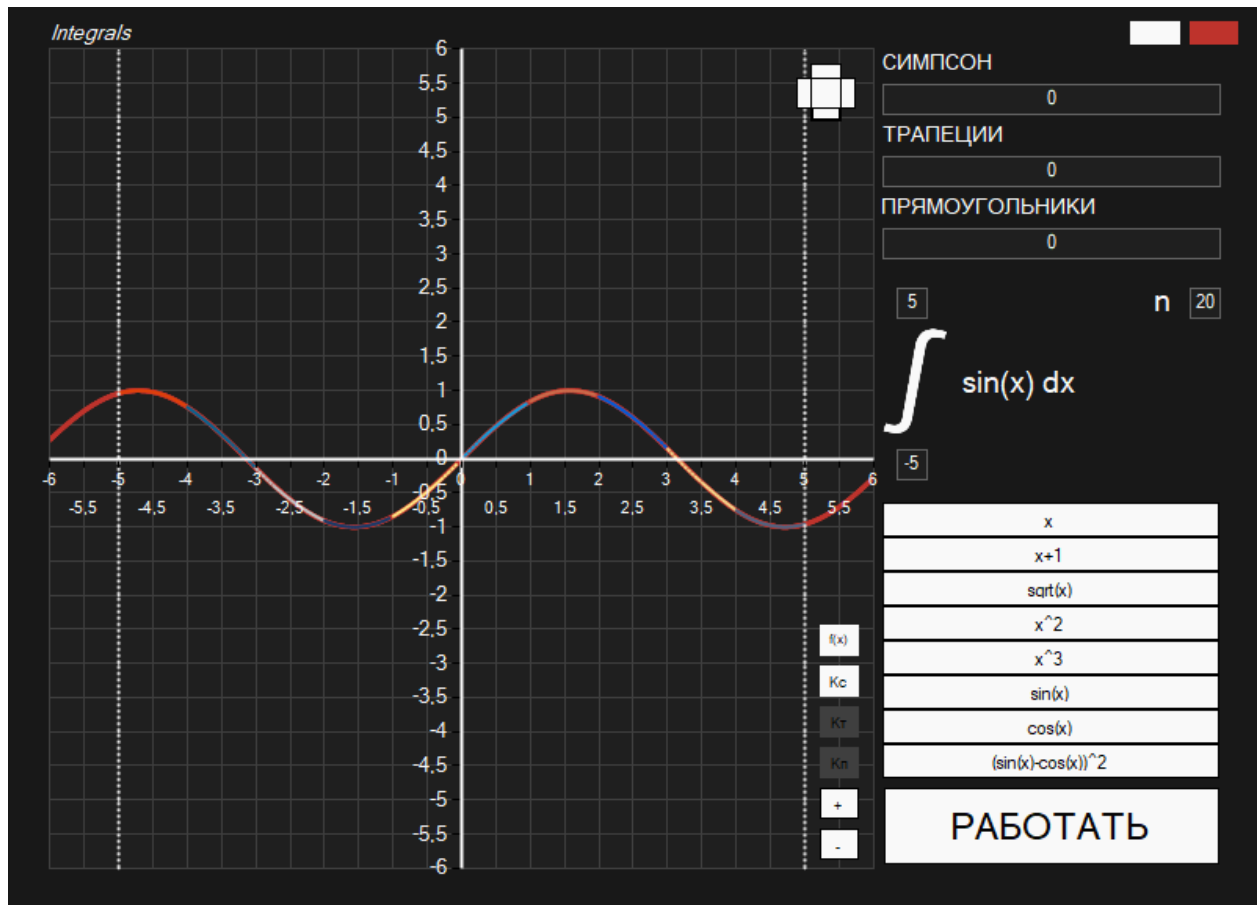


5. Пример работы интеграла от функции x^3



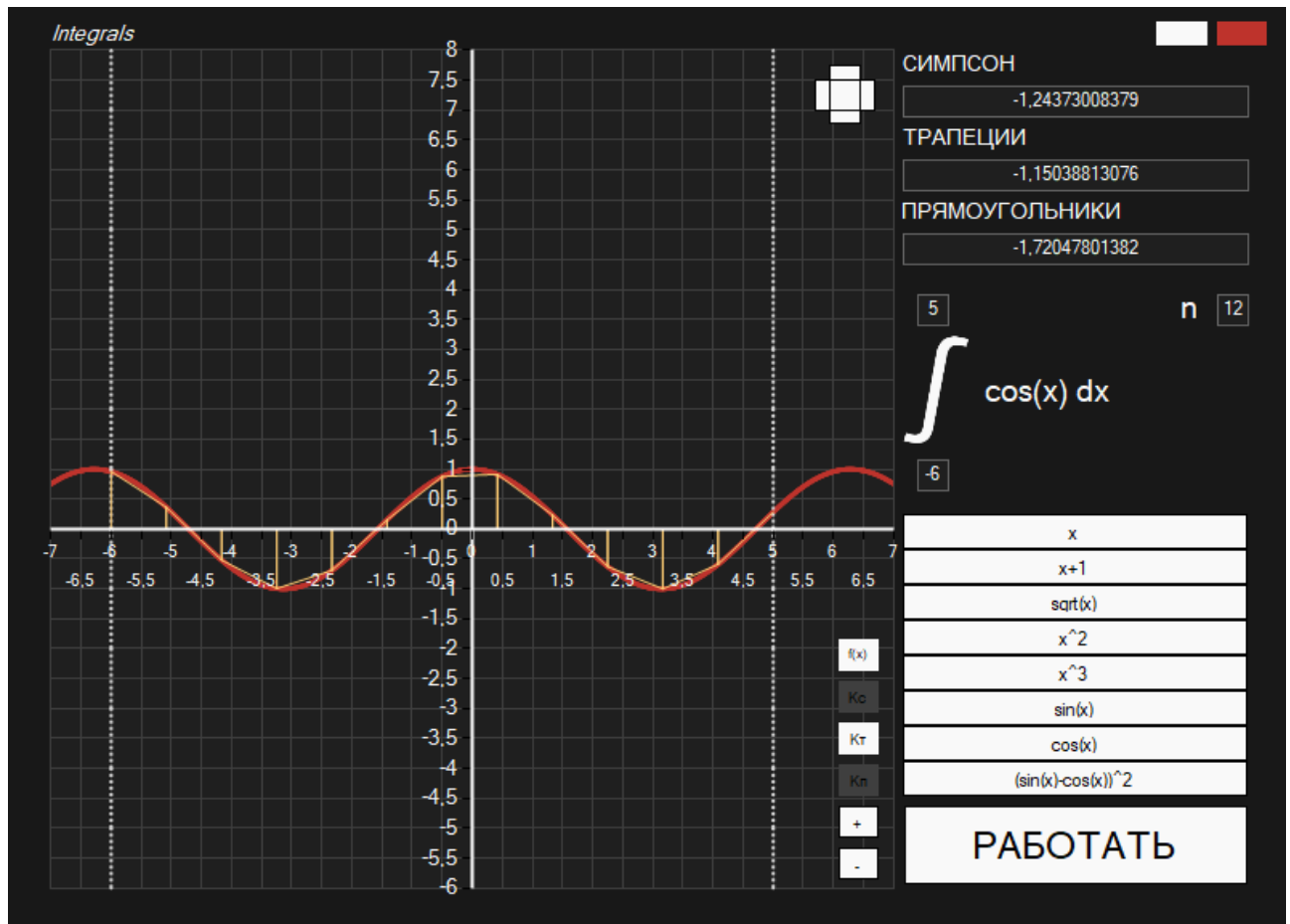
6. Пример работы интеграла от функции

$\sin x$



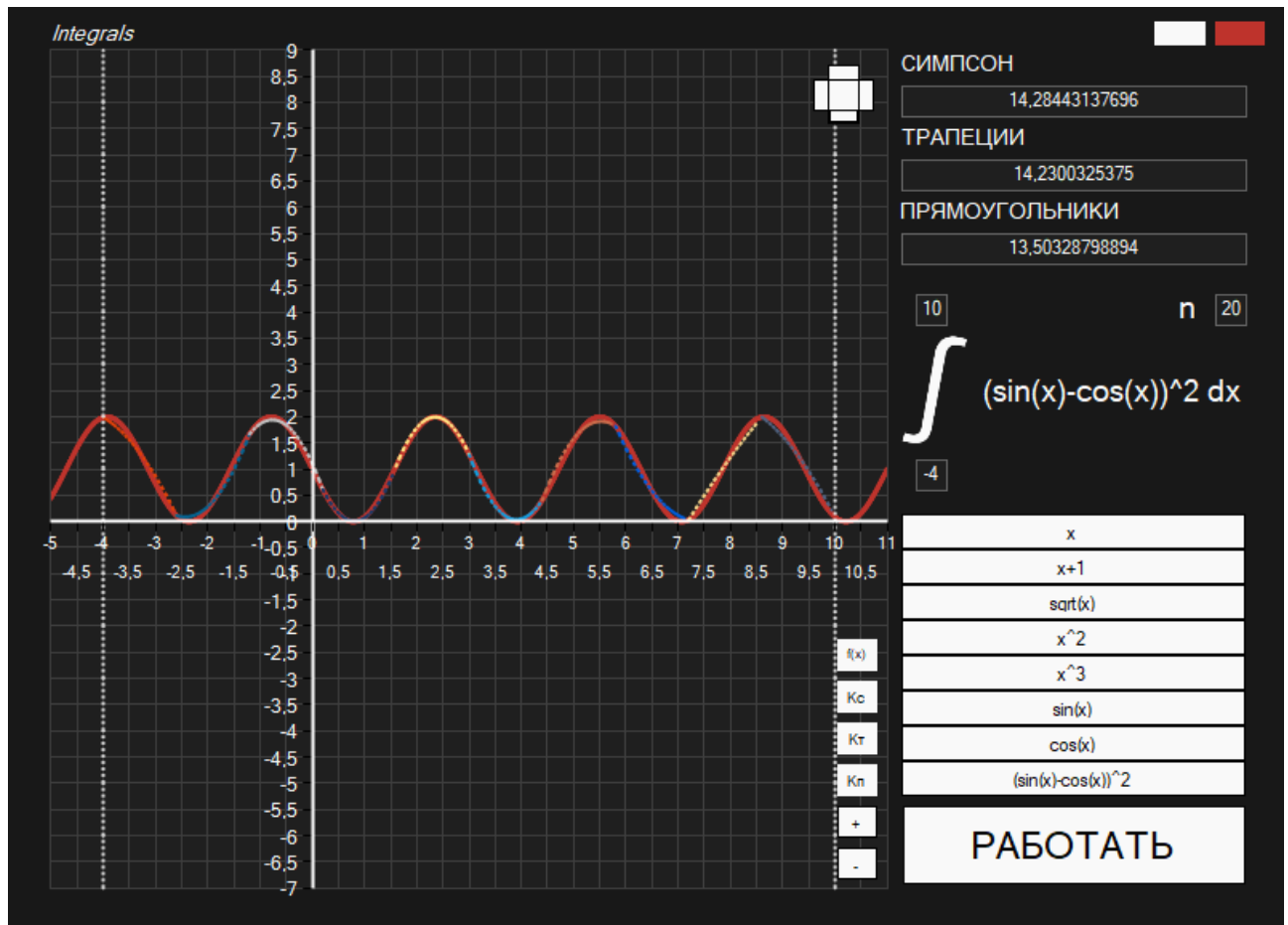
7. Пример работы интеграла от функции

$$\cos(x)$$



8. Пример работы интеграла от функции

$$(\sin x - \cos x)^2$$



Выводы

Заключением нашей работы стало создание программы, которая позволяет пользователю:

- 1) вычислить определённый интеграл от любой произвольной функции (что мы и проверили на 8 различных тестовых интегралах);
- 2) выводить пользователю график функции с квадратурами и настраиваемым количеством шагов (настройка точности подсчетов);