**Инструкция по использованию математической библиотеки для решения систем дифференциальных уравнений, интегрирования и действий с матрицами**

Оглавление

[1. Установка и развертывание. 3](#_Toc451427030)

[1.1. Подключение под C++. 4](#_Toc451427031)

[2. Инструкция программиста 5](#_Toc451427032)

[2.1. Консольное приложение - ConsoleApplication1. 5](#_Toc451427033)

[2.2. Библиотека динамической компоновки - math\_wrapper. 6](#_Toc451427034)

[2.3. Консольное приложение – matr. 8](#_Toc451427035)

[2.4. Форма приложения - matr\_csharp. 9](#_Toc451427036)

[2.5. Юнит тесты. 9](#_Toc451427037)

[3. Функции приложения 10](#_Toc451427038)

[4. Инструкция для программиста - пользователя. 11](#_Toc451427039)

[4.1. Работа с матрицами. 12](#_Toc451427040)

[4.2. Работа с дифференциальными уравнениями*.* 15](#_Toc451427041)

[4.3. Работа с интегрированием уравнений. 21](#_Toc451427042)

[5. Wrapper(ы). 24](#_Toc451427043)

[5.1. Wrapper для дифференцирования уравнений. 24](#_Toc451427044)

[5.2. Wrapper для интегрирования уравнений. 25](#_Toc451427045)

[5.3. Wrapper для работы с матрицами. 26](#_Toc451427046)

[6. Начальные параметры функций. 27](#_Toc451427047)

[6.1. Умножение матрицы на вектор. 27](#_Toc451427048)

[6.2. Умножение матрицы на матрицу. 27](#_Toc451427049)

[6.3. Транспонирование матрицы. 27](#_Toc451427050)

[6.4. Решение определенного интеграла методом Симпсона. 27](#_Toc451427051)

[6.5. Решение определенного интеграла методом Симпсона 3/8. 28](#_Toc451427052)

[6.6. Решение определенного интеграла методом Монте-Карло. 28](#_Toc451427053)

[6.7. Решение системы дифференциальных уравнений методом Эйлера. 28](#_Toc451427054)

[6.8. Решение системы дифференциальных уравнений методом Рунге-Кутта 2. 28](#_Toc451427055)

[6.9. Решение системы дифференциальных уравнений методом Рунге-Кутта 4. 29](#_Toc451427056)

[6.10. Решение системы дифференциальных уравнений методом Эйлера. Перегрузка с tau. 29](#_Toc451427057)

[6.11. Решение системы дифференциальных уравнений методом Рунге-Кутта 2. Перегрузка с tau. 29](#_Toc451427058)

[6.12. Решение системы дифференциальных уравнений методом Рунге-Кутта 4. Перегрузка с tau. 30](#_Toc451427059)

[7. Перспективы развития 30](#_Toc451427060)

# 1. Установка и развертывание.

Программа реализована на языке C-Sharp. Для удобства работы с программой рекомендуется использовать программный продукт Microsoft Visual Studio версии 2008 или более новые.

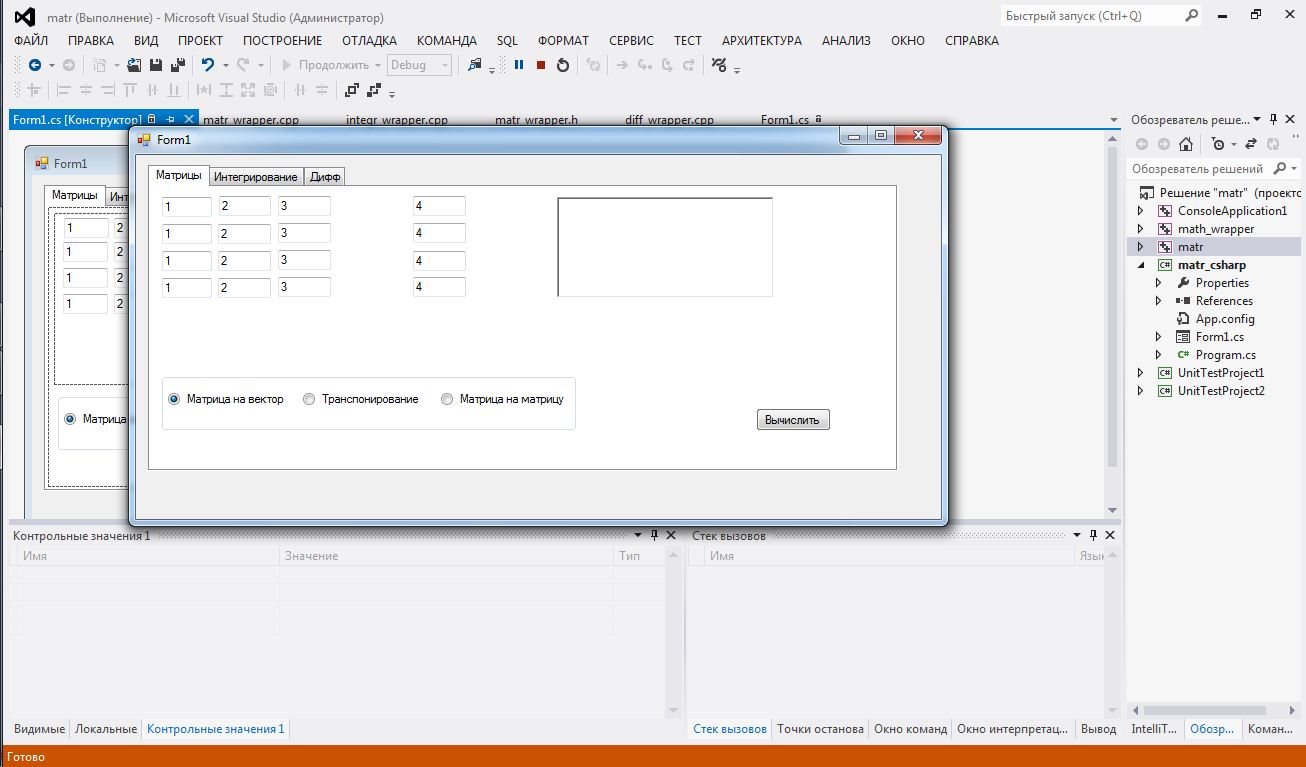


Рис.1. Вид проекта.

Для установки программы необходимо создать новый проект C# и подключить дополнительную библиотеку «библиотека динамической компоновки» math\_wrapper через References. Или подключить к уже имеющемуся проекту.

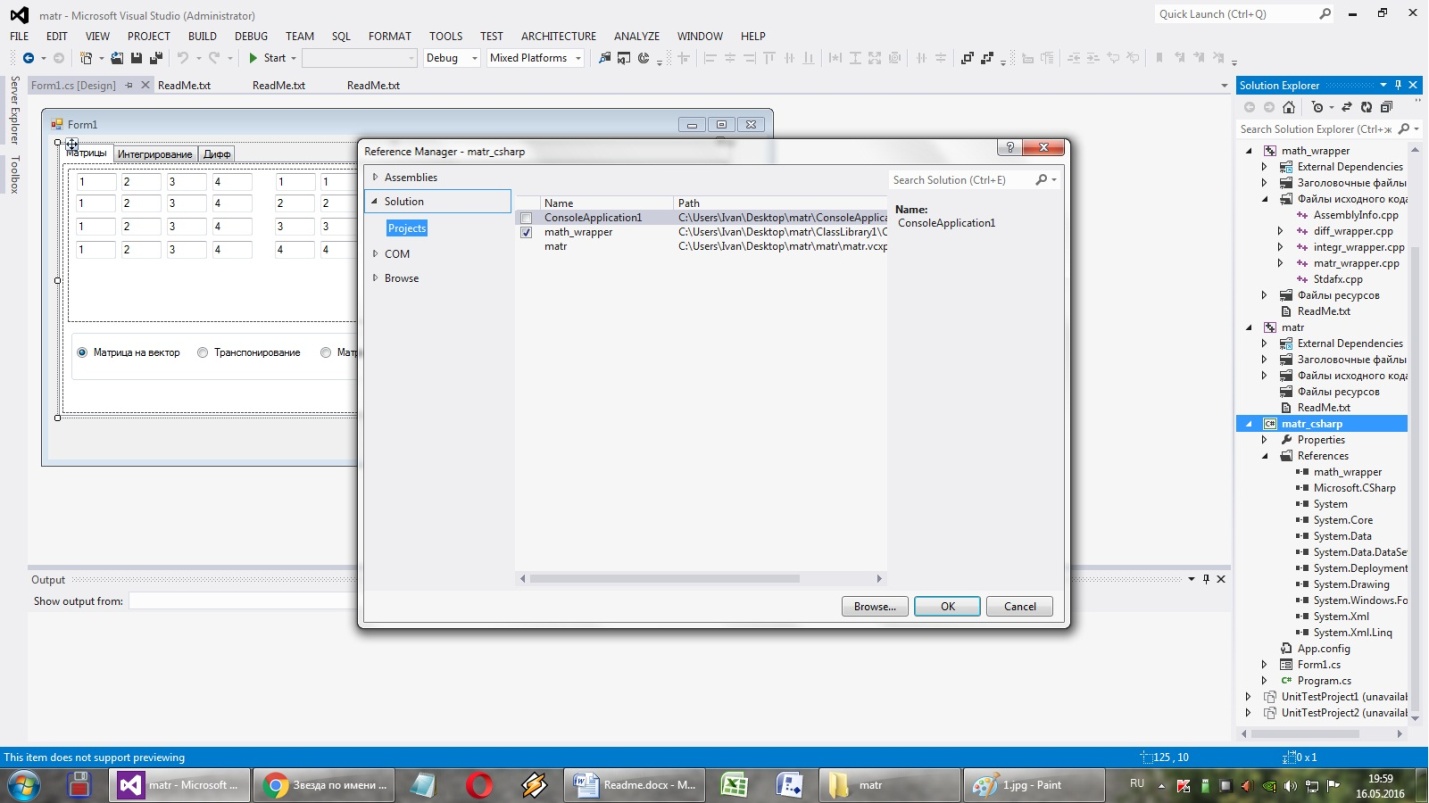


Рис. 2. Меню подключения библиотеки.

Проект имеет следующий Solution Explorer на рис. 3.

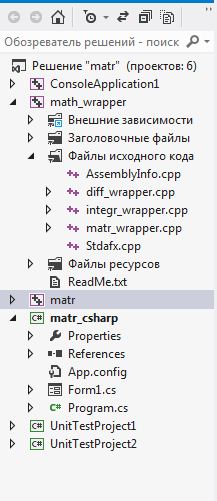


Рис.3.1. Обозреватель решения.

## Подключение под C++.

Подключение библиотеки происходит путём указания полного пути к библиотеке через #include в начале программы.

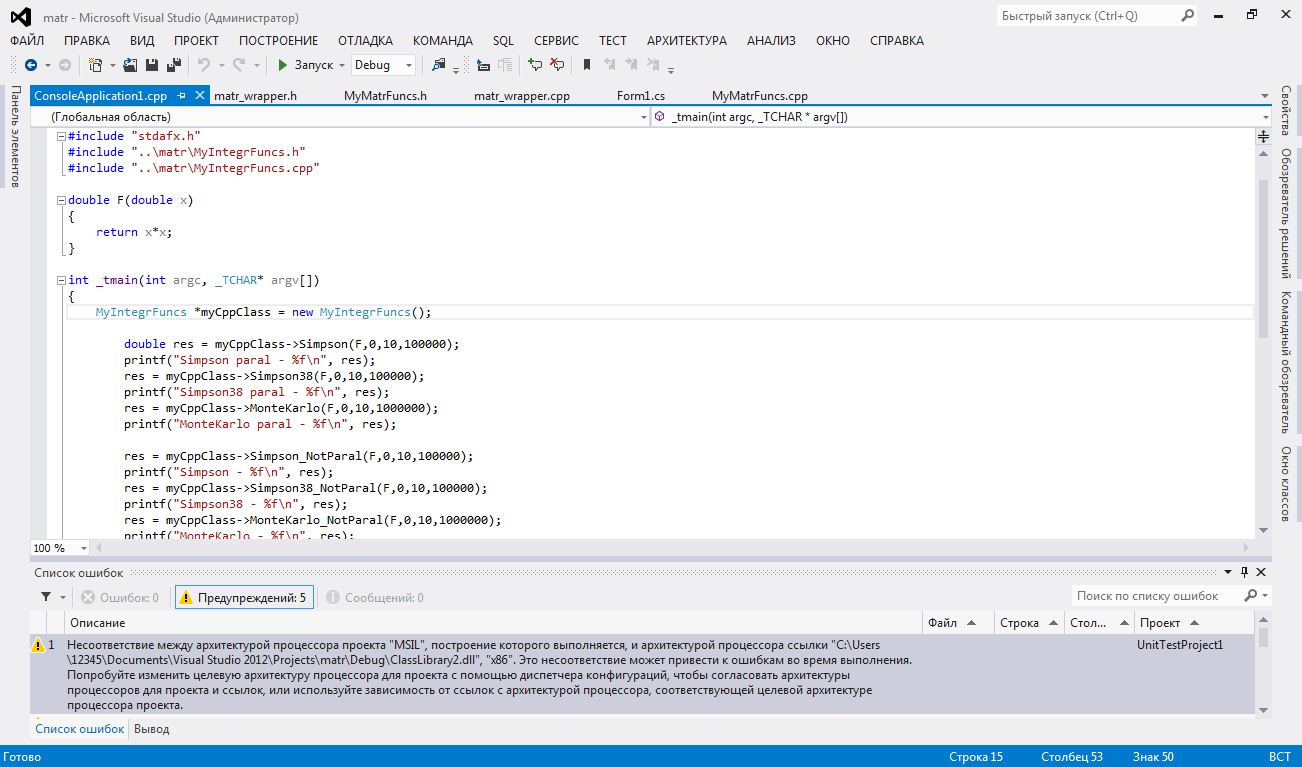


Рис.3.2. Указание пути к библиотеке.

# 2. Инструкция программиста

Проект состоит из следующих компонентов (рис. 3):

1. Консольное приложение - ConsoleApplication1;
2. Библиотека динамической компоновки - math\_wrapper;
3. Консольное приложение – matr;
4. Форма приложения - matr\_csharp;
5. Юнит тесты - UnitTestProject1, UnitTestProject2.

## 2.1. Консольное приложение - ConsoleApplication1.

Это приложение ConsoleApplication1 создано автоматически с помощью мастера приложений. В этом файле представлена сводка содержимого всех файлов, входящих в состав приложения ConsoleApplication1 (рис. 4).

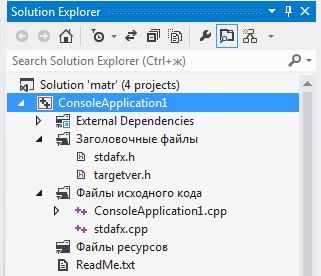


Рис. 4. ConsoleApplication1.

Обзор проекта:

* **ConsoleApplication1.vcxproj**

Это основной файл проекта VC++, создаваемый с помощью мастера приложений. Он содержит данные о версии языка Visual C++, использованной для создания файла, а также сведения о платформах, конфигурациях и функциях проекта, выбранных с помощью мастера приложений.

* **ConsoleApplication1.vcxproj.filters**

Это файл фильтров для проектов VC++, созданный с помощью мастера приложений. Он содержит сведения о сопоставлениях между файлами в вашем проекте и фильтрами. Эти сопоставления используются в среде IDE для группировки файлов с одинаковыми расширениями в одном узле (например CPP-файлы сопоставляются с фильтром "Исходные файлы").

* **ConsoleApplication1.cpp**

Это основной исходный файл приложения.

* **StdAfx.h, StdAfx.cpp**

Эти файлы используются для построения файла предкомпилированного заголовка (PCH) с именем ConsoleApplication1.pch и файла предкомпилированных типов с именем StdAfx.obj.

*С помощью комментариев «TODO:» в мастере приложений обозначаются фрагменты исходного кода, которые необходимо дополнить или изменить.*

## 2.2. Библиотека динамической компоновки - math\_wrapper.

Эта библиотека DLL math\_wrapper создана автоматически с помощью мастера приложений. В этом файле представлена сводка содержимого всех файлов, входящих в состав приложения math\_wrapper (рис. 5).

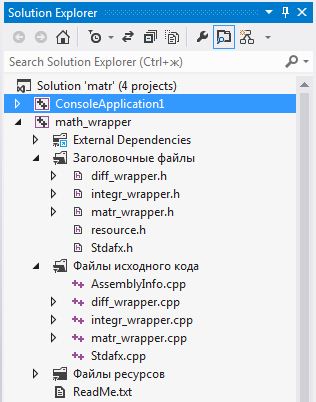


Рис. 5. math\_wrapper.

Обзор библиотеки:

* **math\_wrapper.vcxproj**

Это основной файл проекта VC++, создаваемый с помощью мастера приложений. Он содержит данные о версии языка Visual C++, использованной для создания файла, а также сведения о платформах, конфигурациях и функциях проекта, выбранных с помощью мастера приложений.

* **math\_wrapper.vcxproj.filters**

Это файл фильтров для проектов VC++, созданный с помощью мастера приложений. Он содержит сведения о сопоставлениях между файлами в вашем проекте и фильтрами. Эти сопоставления используются в среде IDE для группировки файлов с одинаковыми расширениями в одном узле (например CPP-файлы сопоставляются с фильтром "Исходные файлы").

* **math\_wrapper.cpp**

Основной исходный файл библиотеки DLL.

* **math\_wrapper.h**

В этом файле содержится объявление класса.

* **AssemblyInfo.cpp**

В него включены настраиваемые атрибуты для изменения метаданных сборки.

*С помощью маркеров задач «TODO:» в мастере приложений обозначаются фрагменты исходного кода, которые необходимо дополнить или изменить.*

## 2.3. Консольное приложение – matr.

Это приложение matr создано автоматически с помощью мастера приложений. В этом файле представлена сводка содержимого всех файлов, входящих в состав приложения matr (рис. 6).

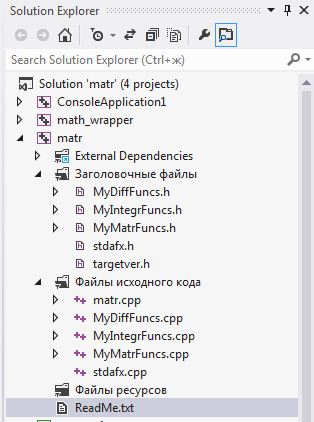


Рис. 6. matr.

Обзор проекта:

* **matr.vcxproj**

Это основной файл проекта VC++, создаваемый с помощью мастера приложений. Он содержит данные о версии языка Visual C++, использованной для создания файла, а также сведения о платформах, конфигурациях и функциях проекта, выбранных с помощью мастера приложений.

* **matr.vcxproj.filters**

Это файл фильтров для проектов VC++, созданный с помощью мастера приложений. Он содержит сведения о сопоставлениях между файлами в вашем проекте и фильтрами. Эти сопоставления используются в среде IDE для группировки файлов с одинаковыми расширениями в одном узле (например CPP-файлы сопоставляются с фильтром "Исходные файлы").

* **matr.cpp**

Это основной исходный файл приложения.

* **StdAfx.h, StdAfx.cpp**

Эти файлы используются для построения файла предкомпилированного заголовка (PCH) с именем matr.pch и файла предкомпилированных типов с именем StdAfx.obj.

*С помощью комментариев «TODO:» в мастере приложений обозначаются фрагменты исходного кода, которые необходимо дополнить или изменить.*

## 2.4. Форма приложения - matr\_csharp.

В этом проекте находится основная форма, её параметры и свойства, подключенные и используемые дополнительные библиотеки и файлы (рис. 7). Этот проект был создан для наглядной демонстрации работы библиотеки. На форме задаются начальные параметры и затем передаются в вызываемые функции. После этого происходит вывод результатов на эту же форму.

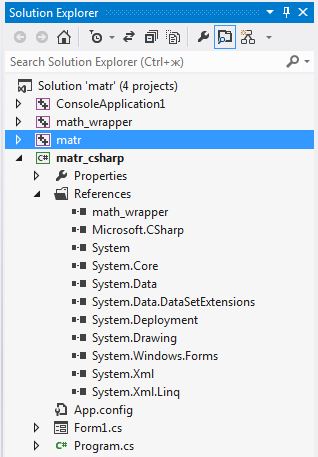


Рис. 7. matr\_csharp.

## 2.5. Юнит тесты.

В программе используются два проекта с юнит тестами:

* UnitTestProject1
* UnitTestProject2

В UnitTestProject1 тестируется функционал для работы с матрицами.

В UnitTestProject2 тестируется интегрирование и дифференцирование систем уравнений.

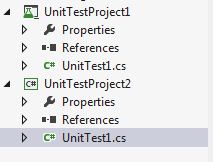


Рис. 8. Юнит тесты.

# 3. Функции приложения

Библиотека способна решать следующие задачи:

1. Последовательная и параллельная работа с матрицами (рис. 9):

* Произведение матрицы на вектор;
* Транспонирование матрицы;
* Произведение двух матриц.

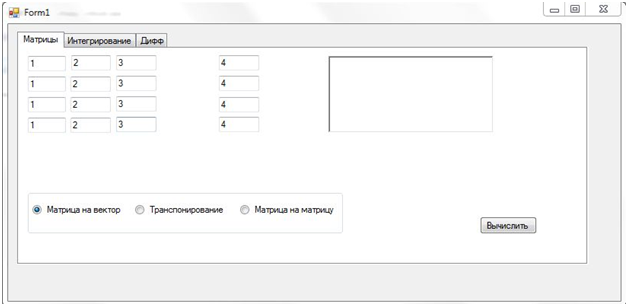


Рис. 9. Работа с матрицами.

1. Последовательное и параллельное интегрирование уравнений (рис. 10):

* Метод Симпсона;
* Метод Симпсон 3/8;
* Метод Монте-Карло.

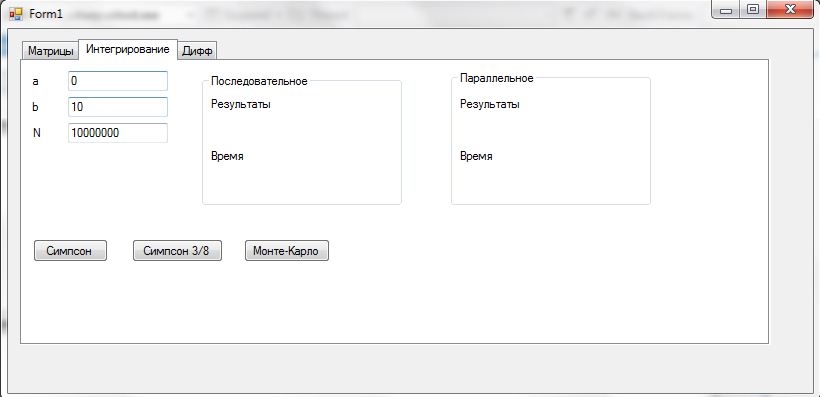


Рис. 10. Интегрирование уравнений.

1. Последовательное и параллельное дифференцирование систем уравнений (рис. 11):

* Метод Эйлера;
* Метод Рунге-Кутта 2;
* Метод Рунге-Кутта 4.

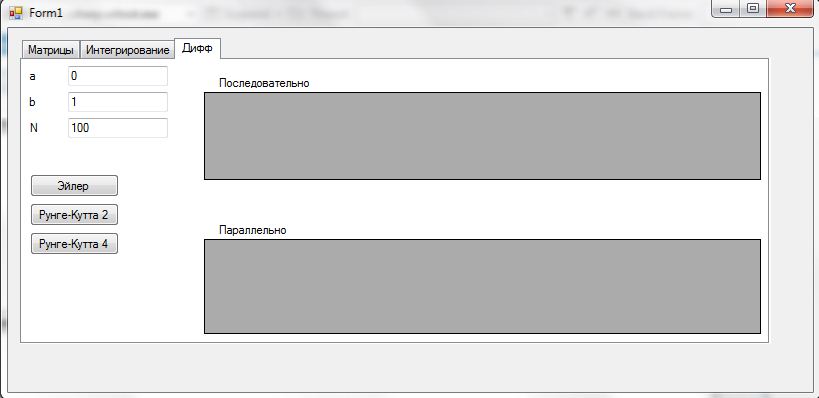


Рис. 11. Дифференцирование систем уравнений.

# 4. Инструкция для программиста - пользователя.

Необходимо подключить math\_wrapper в разделе using.

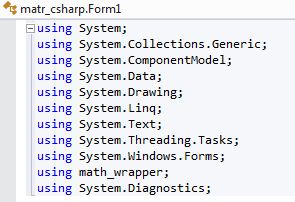


Рис. 12. Using.

## 4.1. Работа с матрицами.

*Файл MyMatrFuncs.cpp*

Необходимо создать экземпляр класса MatrFuncs. Через точку будут доступны методы расчета. Подсказки помогут сориентироваться в передаче параметров.

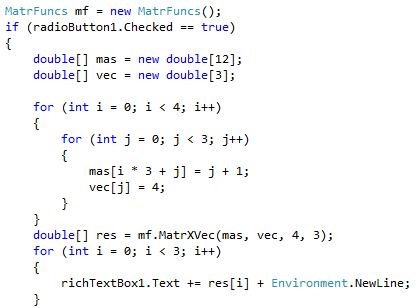


Рис. 4.2. Объявление класса для работы с матрицами

1. Функция произведения матрицы на вектор (параллельная):

double\* MyMatrFuncs::MatrXVect(double\*\* m, double\* v, int n, int k)

{

double\* v\_res;

v\_res = new double[k];

double s = 0;

#pragma omp parallel for reduction(+:s)

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

{

for ( int j = 0; j < k; j++)

{

s = s + m[i][j] \* v[j];

}

v\_res[i] = s;

s = 0;

}

return v\_res;

}

1. Функция транспонирования матрицы (параллельная):

double\*\* MyMatrFuncs::MatrTransp(double\*\* m, int n)

{

double\*\* m\_res = new double\*[n];

#pragma omp parallel for

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

{

m\_res[i]=new double[n];

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

{

m\_res[i][j] = m [j][i];

}

}

return m\_res;

}

1. Функция произведения матриц (параллельная):

double\*\* MyMatrFuncs::MatrXMatr(double\*\* m1, double\*\* m2, int h1, int w1, int h2, int w2)

{

if (w1 != h2)

{

return 0;

}

double\*\* m\_res;

m\_res = new double\*[h1];

for (int i = 0; i < h1; i++)

{

m\_res[i] = new double[h2];

}

for (int i = 0; i < h1; i++)

{

for (int j = 0; j < w2; j++)

{

m\_res[i][j] = 0;

}

}

double sum = 0;

#pragma omp parallel for reduction(+:sum)

for (int i = 0; i < h1; i++)

{

for (int k = 0; k < w2; k++)

{

for (int j = 0; j < w1; j++)

{

sum += m1[i][j] \* m2[j][k];

}

m\_res[i][k] = sum;

sum = 0;

}

}

return m\_res;

}

1. Функция произведения матрицы на вектор (последовательная):

double\* MyMatrFuncs::MatrXVect\_NotParal(double\*\* m, double\* v, int n, int k)

{

double\* v\_res;

v\_res = new double[k];

double s = 0;

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

{

for ( int j = 0; j < k; j++)

{

s = s + m[i][j] \* v[j];

}

v\_res[i] = s;

s = 0;

}

return v\_res;

}

1. Функция транспонирования матрицы (последовательная):

double\*\* MyMatrFuncs::MatrTransp\_NotParal(double\*\* m, int n)

{

double\*\* m\_res = new double\*[n];

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

{

m\_res[i]=new double[n];

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

{

m\_res[i][j] = m [j][i];

}

}

return m\_res;

}

1. Функция произведения матриц (последовательная):

double\*\* MyMatrFuncs::MatrXMatr\_NotParal(double\*\* m1, double\*\* m2, int h1, int w1, int h2, int w2)

{

if (w1 != h2)

{

return 0;

}

double\*\* m\_res;

m\_res = new double\*[h1];

for (int i = 0; i < h1; i++)

{

m\_res[i] = new double[h2];

}

for (int i = 0; i < h1; i++)

{

for (int j = 0; j < w2; j++)

{

m\_res[i][j] = 0;

}

}

double sum = 0;

for (int i = 0; i < h1; i++)

{

for (int k = 0; k < w2; k++)

{

for (int j = 0; j < w1; j++)

{

sum += m1[i][j] \* m2[j][k];

}

m\_res[i][k] = sum;

sum = 0;

}

}

return m\_res;

}

## 4.2. Работа с дифференциальными уравнениями*.*

*Файл MyDiffFuncs.cpp*

Необходимо создать экземпляр класса DiffFuncs. Через точку будут доступны методы расчета. Подсказки помогут сориентироваться в передаче параметров.

Также следует обратить внимание на то, что при передаче функций в метод необходимо использовать небезопасный контекст. Объявление массива функций следует производить, используя делегат из враппера – DiffFuncs.FDelegate.

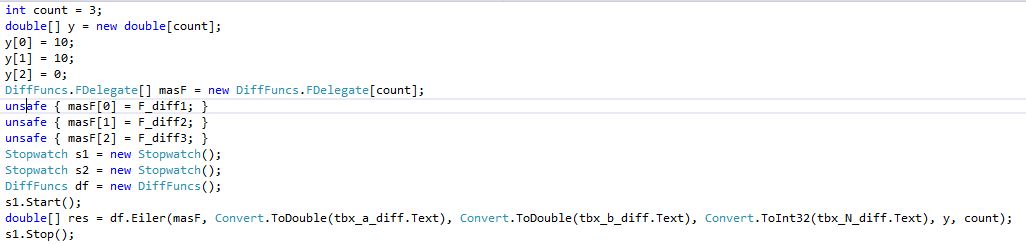


Рис. 13. Пример работы с методом Эйлера.

1. Функция - метод Эйлера (параллельный):

double\* MyDiffFuncs::Eiler\_Paral(void\*\* Function, double a, double b, int n, double\* y, int count)

{

double h = (b - a) / n;

FType\* F = (FType\* )(Function);

for (int j = 0; j < n; j++)//цикл по веремни

{

#pragma omp parallel for

for (int i = 0; i < count; i++)

{

y[i] = y[i] + h \* F[i](j \* h, y);

}

}

return y;

}

1. Функция – Рунге-Кутта 2 (параллельная):

double\* MyDiffFuncs::RungeKutta2\_Paral(void\*\* Function, double a, double b, int n, double\* y, int count)

{

double h = (b - a) / n;

FType\* F = (FType\* )(Function);

double\* z = new double[count];

for (int i = 0; i < count; i++)

z[i] = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

#pragma omp parallel for

for (int i = 0; i < count; i++)

{

z[i] = y[i] + h/2 \* F[i](j \* h, y);

y[i] = y[i] + h \* F[i](j \* h, z);

}

}

return y;

}

1. Функция – Рунге-Кутта 4 (параллельная):

double\* MyDiffFuncs::RungeKutta4\_Paral(void\*\* Function, double a, double b, int n, double\* y, int count)

{

FType\* F = (FType\* )(Function);

// значения функций

double\* xx = new double[count];// при t=0.

double\* R1 = new double[count]; // первое слагаемое из формулы для трёх функций

double\* R2 = new double[count]; // второе слагаемое из формулы для трёх функций

double\* R3 = new double[count]; // третье слагаемое из формулы для трёх функций

double\* R4 = new double[count]; // четвертое слагаемое из формулы для трёх функций

double h = (b - a) / n;

double t;

for (int q = 0; q < n - 1; q++)

{

t = q \* h;

// первое слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

R1[i] = h \* F[i](t, y);

// второе слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R1[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R2[i] = h \* F[i](t + 0.5 \* h, xx);

// третье слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R2[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R3[i] = h \* F[i](t + h, xx);

// четвертое слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + R3[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R4[i] = h \* F[i](t + h, xx);

// новые значения функций

#pragma omp parallel for

for (int i = 0; i < count; i++)

y[i] = y[i] + (R1[i] + 2 \* R2[i] + 2 \* R3[i] + R4[i]) / 6;

}

return y;

}

1. Функция – метод Эйлера (последовательная):

double\* MyDiffFuncs::Eiler(void\*\* Function, double a, double b, int n, double\* y, int count)

{

double h = (b - a) / n;

FType\* F = (FType\* )(Function);

double t = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

for (int i = 0; i < count; i++)

{

y[i] = y[i] + h \* F[i](j \* h, y);

}

}

return y;

}

1. Функция – Рунге-Кутта 2 (последовательная):

double\* MyDiffFuncs::RungeKutta2(void\*\* Function, double a, double b, int n, double\* y, int count)

{

double h = (b - a) / n;

FType\* F = (FType\* )(Function);

double\* z = new double[count];

for (int i = 0; i < count; i++)

z[i] = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

for (int i = 0; i < count; i++)

{

z[i] = y[i] + h/2 \* F[i](j \* h, y);

y[i] = y[i] + h \* F[i](j \* h, z);

}

}

return y;

}

1. Функция – Рунге-Кутта 4 (последовательная):

double\* MyDiffFuncs::RungeKutta4(void\*\* Function, double a, double b, int n, double\* y, int count)

{

FType\* F = (FType\* )(Function);

// значения функций

double\* xx = new double[count];// при t=0.

double\* R1 = new double[count]; // первое слагаемое из формулы для трёх функций

double\* R2 = new double[count]; // второе слагаемое из формулы для трёх функций

double\* R3 = new double[count]; // третье слагаемое из формулы для трёх функций

double\* R4 = new double[count]; // четвертое слагаемое из формулы для трёх функций

double h = (b - a) / n;

double t;

for (int q = 0; q < n - 1; q++)

{

t = q \* h;

// первое слагаемое

for (int i = 0; i < count; i++)

R1[i] = h \* F[i](t, y);

// второе слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R1[i];

for (int i = 0; i < count; i++)

R2[i] = h \* F[i](t + 0.5 \* h, xx);

// третье слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R2[i];

for (int i = 0; i < count; i++)

R3[i] = h \* F[i](t + h, xx);

// четвертое слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + R3[i];

for (int i = 0; i < count; i++)

R4[i] = h \* F[i](t + h, xx);

// новые значения функций

for (int i = 0; i < count; i++)

y[i] = y[i] + (R1[i] + 2 \* R2[i] + 2 \* R3[i] + R4[i]) / 6;

}

return y;

}

**Реализация перегрузки с использованием tau (**вместо промежутка [a, b] передается заранее известный шаг tau**):**

1. Функция - метод Эйлера (параллельный):

double\* MyDiffFuncs::Eiler\_Paral(void\*\* Function, double tau, int n, double\* y, int count)

{

double h = tau;

FType\* F = (FType\* )(Function);

for (int j = 0; j < n; j++)//цикл по веремни

{

#pragma omp parallel for

for (int i = 0; i < count; i++)

{

y[i] = y[i] + h \* F[i](j \* h, y);

}

}

return y;

}

1. Функция – Рунге-Кутта 2 (параллельная):

double\* MyDiffFuncs::RungeKutta2\_Paral(void\*\* Function, double tau, int n, double\* y, int count)

{

double h = tau;

FType\* F = (FType\* )(Function);

double\* z = new double[count];

for (int i = 0; i < count; i++)

z[i] = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

#pragma omp parallel for

for (int i = 0; i < count; i++)

{

z[i] = y[i] + h/2 \* F[i](j \* h, y);

y[i] = y[i] + h \* F[i](j \* h, z);

}

}

return y;

}

1. Функция – Рунге-Кутта 4 (параллельная):

double\* MyDiffFuncs::RungeKutta4\_Paral(void\*\* Function, double tau, int n, double\* y, int count)

{

FType\* F = (FType\* )(Function);

// значения функций

double\* xx = new double[count];// при t=0.

double\* R1 = new double[count]; // первое слагаемое из формулы для трёх функций

double\* R2 = new double[count]; // второе слагаемое из формулы для трёх функций

double\* R3 = new double[count]; // третье слагаемое из формулы для трёх функций

double\* R4 = new double[count]; // четвертое слагаемое из формулы для трёх функций

double h = tau;

double t;

for (int q = 0; q < n - 1; q++)

{

t = q \* h;

// первое слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

R1[i] = h \* F[i](t, y);

// второе слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R1[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R2[i] = h \* F[i](t + 0.5 \* h, xx);

// третье слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R2[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R3[i] = h \* F[i](t + h, xx);

// четвертое слагаемое

#pragma omp parallel for

for (int i = 0; i < count; i++)

xx[i] = y[i] + R3[i];

#pragma omp parallel for

for (int i = 0; i < count; i++)

R4[i] = h \* F[i](t + h, xx);

// новые значения функций

#pragma omp parallel for

for (int i = 0; i < count; i++)

y[i] = y[i] + (R1[i] + 2 \* R2[i] + 2 \* R3[i] + R4[i]) / 6;

}

return y;

}

1. Функция – метод Эйлера (последовательная):

double\* MyDiffFuncs::Eiler(void\*\* Function, double tau, int n, double\* y, int count)

{

double h = tau;

FType\* F = (FType\* )(Function);

double t = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

for (int i = 0; i < count; i++)

{

y[i] = y[i] + h \* F[i](j \* h, y);

}

}

return y;

}

1. Функция – Рунге-Кутта 2 (последовательная):

double\* MyDiffFuncs::RungeKutta2(void\*\* Function, double tau, int n, double\* y, int count)

{

double h = tau;

FType\* F = (FType\* )(Function);

double\* z = new double[count];

for (int i = 0; i < count; i++)

z[i] = 0;

for (int j = 0; j < n; j++)//цикл по веремни

{

for (int i = 0; i < count; i++)

{

z[i] = y[i] + h/2 \* F[i](j \* h, y);

y[i] = y[i] + h \* F[i](j \* h, z);

}

}

return y;

}

1. Функция – Рунге-Кутта 4 (последовательная):

double\* MyDiffFuncs::RungeKutta4(void\*\* Function, double tau, int n, double\* y, int count)

{

FType\* F = (FType\* )(Function);

// значения функций

double\* xx = new double[count];// при t=0.

double\* R1 = new double[count]; // первое слагаемое из формулы для трёх функций

double\* R2 = new double[count]; // второе слагаемое из формулы для трёх функций

double\* R3 = new double[count]; // третье слагаемое из формулы для трёх функций

double\* R4 = new double[count]; // четвертое слагаемое из формулы для трёх функций

double h = tau;

double t;

for (int q = 0; q < n - 1; q++)

{

t = q \* h;

// первое слагаемое

for (int i = 0; i < count; i++)

R1[i] = h \* F[i](t, y);

// второе слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R1[i];

for (int i = 0; i < count; i++)

R2[i] = h \* F[i](t + 0.5 \* h, xx);

// третье слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + 0.5 \* R2[i];

for (int i = 0; i < count; i++)

R3[i] = h \* F[i](t + h, xx);

// четвертое слагаемое

for (int i = 0; i < count; i++)

xx[i] = y[i] + R3[i];

for (int i = 0; i < count; i++)

R4[i] = h \* F[i](t + h, xx);

// новые значения функций

for (int i = 0; i < count; i++)

y[i] = y[i] + (R1[i] + 2 \* R2[i] + 2 \* R3[i] + R4[i]) / 6;

}

return y;

}

## 4.3. Работа с интегрированием уравнений.

*Файл MyIntegrFuncs.cpp*

Необходимо создать экземпляр класса IntegrFuncs. Через точку будут доступны методы расчета. Подсказки помогут сориентироваться в передаче параметров.

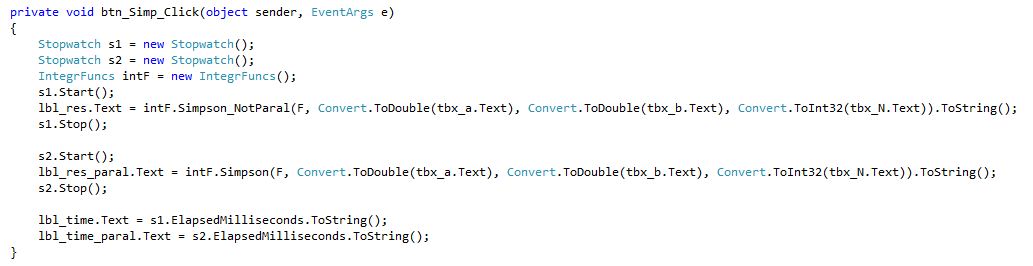


Рис. 14. Пример работы с методом Симпсона.

1. Функция – метод Симпсона (параллельная):

double MyIntegrFuncs::Simpson(void\* Function, double a, double b, int N)

{

FType F = (FType )(Function);//приведение указателя на функцию к нужному типу

double h = (b - a) / N;

double x = a + h;

double s = 0;

double x1;

int num = (int)((b - x) / (2 \* h) + 1);

#pragma omp parallel for reduction(+:s)

for (int i = 0; i < num; i++)

{

s = s + 4 \* F(a + h + 2 \* h \* i);

}

#pragma omp parallel for reduction(+:s)

for (int i = 0; i < num; i++)

{

s = s + 2 \* F(a + h + h + 2 \* h \* i);

}

return h / 3 \* (s + F(a) - F(b));

}

1. Функция – метод Симпсона 3/8 (параллельная):

double MyIntegrFuncs::Simpson38(void\* Function, double a, double b, int N)

{

FType F = (FType )(Function);

double h = (b - a) / (3 \* N);

double s1 = F(a) + F(b);

int m = 3 \* N - 1;

#pragma omp parallel for reduction(+:s1)

for (int i = 1; i <= m; i++)

{

if (i % 3 == 0)

s1 = s1 + 2 \* F(a + h \* i);

else

s1 = s1 + 3 \* F(a + h \* i);

}

return s1 \* 3 / 8 \* h;

}

1. Функция – метод Монте-Карло (параллельная):

double MyIntegrFuncs::MonteKarlo(void\* Function, double a, double b, int N)

{

FType F = (FType )(Function);

double c, d, h;

// Находим максимум функции

srand(time(0));

int n = N;

int col = 0;

h = (b - a) / n; // Шаг интегрирования

double S;

double a1 = a, b1 = b;

double x1 = 0, x2 = 0;

double x = 0;

double e = 0.0001;

double q = (1 + sqrt(5.0)) / 2;

x1 = b1 - (b1 - a1) / q;

x2 = a1 + (b1 - a1) / q;

while(abs(b1 - a1) > e)

{

if (F(x1) <= F(x2))

{

a1 = x1;

x1 = x2;

x2 = a1 + (b1 - a1) / q;

}

else

{

b1 = x2;

x2 = x1;

x1 = b1 - (b1 - a1) / q;

}

}

c = F((a1 + b1) / 2);

S = c \* (b - a);

double m;

double r3, r4, r5, r6;

int r1, r2;

#pragma omp parallel for private (r1, r2, r3, r4, r5, r6) reduction(+:col)

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

{

r1 = rand()%100;

r2 = rand()%100;

r3 = r1 \* 0.01; // генерируем число от 0 до 1

r4 = r2 \* 0.01;

r5 = r3 \* (b - a) + a;

r6 = r4 \* c;

m = F(r5);

if (r6 < m)

{

col = col + 1;

}

}

return S \* ((double)col / n);

}

1. Функция – метод Симпсона (последовательная):

double MyIntegrFuncs::Simpson\_NotParal(void\* Function, double a, double b, int N)

{

FType F = (FType )(Function);

double h = (b - a) / N;

double x = a + h;

double s = 0;

int num = (int)((b - x) / (2 \* h) + 1);

for (int i = 0; i < num; i++)

{

s = s + 4 \* F(x);

x = x + h;

s = s + 2 \* F(x);

x = x + h;

}

return h / 3 \* (s + F(a) - F(b));

}

1. Функция – метод Симпсона 3/8 (последовательная):

double MyIntegrFuncs::Simpson38\_NotParal(void\* Function, double a, double b, int N)

{

FType F = (FType )(Function);

double h = (b - a) / (3 \* N);

double s1 = F(a) + F(b);

double m = 3 \* N - 1;

double x = 0;

for (int i = 1; i <= m; i++)

{

x = a + h \* i;

if (i % 3 == 0)

s1 = s1 + 2 \* F(x);

else

s1 = s1 + 3 \* F(x);

}

s1 = s1 \* 3 / 8 \* h;

return s1;

}

1. Функция – метод Монте-Карло (последовательная):

double MyIntegrFuncs::MonteKarlo\_NotParal(void\* Function, double a, double b, int N)

{

FType F= (FType )(Function);

double c, d, h;

// Находим максимум функции

srand(time(0));

int n = N;

int col = 0;

h = (b - a) / n; // Шаг интегрирования

double S;

double a1 = a, b1 = b;

double x1 = 0, x2 = 0;

double x = 0;

double e = 0.0001;

double q = (1 + sqrt(5.0)) / 2;

x1 = b1 - (b1 - a1) / q;

x2 = a1 + (b1 - a1) / q;

while(abs(b1 - a1) > e)

{

if (F(x1) <= F(x2))

{

a1 = x1;

x1 = x2;

x2 = a1 + (b1 - a1) / q;

}

else

{

b1 = x2;

x2 = x1;

x1 = b1 - (b1 - a1) / q;

}

}

c = F((a1 + b1) / 2);

S = c \* (b - a);

double m;

double r3, r4, r5, r6;

int r1, r2;

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

{

r1 = rand()%100;

r2 = rand()%100;

r3 = r1 \* 0.01; // генерируем число от 0 до 1

r4 = r2 \* 0.01;

r5 = r3 \* (b - a) + a;

r6 = r4 \* c;

m = F(r5);

if (r6 < m)

{

col = col + 1;

}

}

return S \* ((double)col / n);

}

# 5. Wrapper(ы).

Находятся в проекте math\_wrapper. Используется для взаимодействия управляемого кода с неуправляемым, если мы хотим в C# использовать более быструю реализацию функций на C++.

## 5.1. Wrapper для дифференцирования уравнений.

*Файл diff\_wrapper.h*

#pragma once

#include "..\matr\MyDiffFuncs.h"

#include "..\matr\MyDiffFuncs.cpp"

using namespace System;

using namespace System::Runtime::InteropServices;

namespace math\_wrapper {

public ref class DiffFuncs

{

public: DiffFuncs();

private:

MyDiffFuncs \*myCppClass;

void\*\* delegatePointer;

public:

[UnmanagedFunctionPointerAttribute(CallingConvention::Cdecl)]

delegate double FDelegate(double x, double\* y);

public:

array<double>^ Eiler(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta2(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta4(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

public:

array<double>^ Eiler\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta2\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta4\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count);

**Перегрузки с использованием tau:**

public:

array<double>^ Eiler(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta2(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta4(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

public:

array<double>^ Eiler\_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta2\_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

public:

array<double>^ RungeKutta4\_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count);

};

}

## 5.2. Wrapper для интегрирования уравнений.

*Файл integr\_wrapper.h*

#pragma once

#include "..\matr\MyIntegrFuncs.h"

#include "..\matr\MyIntegrFuncs.cpp"

using namespace System;

using namespace System::Runtime::InteropServices;

namespace math\_wrapper {

public ref class IntegrFuncs

{

public: IntegrFuncs();

private:

MyIntegrFuncs \*myCppClass;

void\* delegatePointer;

public:

[UnmanagedFunctionPointerAttribute(CallingConvention::Cdecl)]

delegate double FDelegate(double x);

public:

double Simpson(FDelegate ^ fdelegate, double a, double b, int N);

public:

double Simpson38(FDelegate ^ fdelegate, double a, double b, int N);

public:

double MonteKarlo(FDelegate ^ fdelegate, double a, double b, int N);

public:

double Simpson\_NotParal(FDelegate ^ fdelegate, double a, double b, int N);

public:

double Simpson38\_NotParal(FDelegate ^ fdelegate, double a, double b, int N);

public:

double MonteKarlo\_NotParal(FDelegate ^ fdelegate, double a, double b, int N);

};

}

## 5.3. Wrapper для работы с матрицами.

*Файл matr\_wrapper.h*

#pragma once

#include "..\matr\MyMatrFuncs.h"

#include "..\matr\MyMatrFuncs.cpp"

using namespace System;

namespace math\_wrapper {

public ref class MatrFuncs

{

public: MatrFuncs();

private:

MyMatrFuncs \*myCppClass;

// TODO: здесь следует добавить свои методы для этого класса.

public:

array<double>^ MatrXVec(array<array<double>^>^ m, array<double>^ v, int n, int k);

public:

array<array<double>^>^ MatrTranspon(array<array<double>^>^ m, int n);

public:

array<array<double>^>^ MatrXMatr(array<array<double>^>^ m1, array<array<double>^>^ m2, int h1, int w1, int h2, int w2);

public:

array<double>^ MatrXVec\_NotParal(array<array<double>^>^ m, array<double>^ v, int n, int k);

public:

array<array<double>^>^ MatrTranspon\_NotParal(array<array<double>^>^ m, int n);

public:

array<array<double>^>^ MatrXMatr\_NotParal(array<array<double>^>^ m1, array<array<double>^>^ m2, int h1, int w1, int h2, int w2);

public:

array<double>^ MatrXVec(array<double>^ m, array<double>^ v, int n, int k);

public:

array<double>^ MatrTranspon(array<double>^ m, int n);

public:

array<double>^ MatrXMatr(array<double>^ m1, array<double>^ m2, int h1, int w1, int h2, int w2);

public:

array<double>^ MatrXVec\_NotParal(array<double>^ m, array<double>^ v, int n, int k);

public:

array<double>^ MatrTranspon\_NotParal(array<double>^ m, int n);

public:

array<double>^ MatrXMatr\_NotParal(array<double>^ m1, array<double>^ m2, int h1, int w1, int h2, int w2);

};

}

# 6. Начальные параметры функций.

## 6.1. Умножение матрицы на вектор.

* array<double>^ math\_wrapper::MatrFuncs::MatrXVec(array<array<double>^>^ m, array<double>^ v, int n, int k),
* array<double>^ math\_wrapper::MatrFuncs::MatrXVec(array<double>^ m, array<double>^ v, int n, int k),

Где:

1. m - матрица.
2. v – вектор.
3. n - высота матрицы.
4. k - ширина матрицы.
5. Возвращает – вектор.

## 6.2. Умножение матрицы на матрицу.

* array<array<double>^>^ math\_wrapper::MatrFuncs::MatrXMatr(array<array<double>^>^ m1, array<array<double>^>^ m2, int n, int k),
* array<double>^ math\_wrapper::MatrFuncs::MatrXMatr(array<double>^ m1, array<double>^ m2, int h1, int w1, int h2, int w2),

Где:

1. m1 - матрица.
2. m2 – вектор.
3. n - высота матрицы.
4. k - ширина матрицы.
5. Возвращает – матрицу.

## 6.3. Транспонирование матрицы.

* array<array<double>^>^ math\_wrapper::MatrFuncs::MatrTranspon(array<array<double>^>^ m, int n),
* array<double>^ math\_wrapper::MatrFuncs::MatrTranspon(array<double>^ m, int n),

Где:

1. m - матрица.
2. n - размер матрицы.
3. Возвращает – матрицу.

## 6.4. Решение определенного интеграла методом Симпсона.

* double math\_wrapper::IntegrFuncs::Simpson(FDelegate ^ fdelegate, double a, double b, int N),
* double math\_wrapper::IntegrFuncs::Simpson\_NotParal(FDelegate ^ fdelegate, double a, double b, int N),

Где:

1. fdelegate - функция.
2. a - нижняя граница.
3. b - нижняя граница.
4. N – количество шагов интегрирования.
5. Возвращает – значение интеграла.

## 6.5. Решение определенного интеграла методом Симпсона 3/8.

* double math\_wrapper::IntegrFuncs::Simpson38(FDelegate ^ fdelegate, double a, double b, int N),
* double math\_wrapper::IntegrFuncs::Simpson38\_NotParal(FDelegate ^ fdelegate, double a, double b, int N),

Где:

1. fdelegate - функция.
2. a - нижняя граница.
3. b - нижняя граница.
4. N – количество шагов интегрирования.
5. Возвращает – значение интеграла.

## 6.6. Решение определенного интеграла методом Монте-Карло.

* double math\_wrapper::IntegrFuncs::MonteKarlo(FDelegate ^ fdelegate, double a, double b, int N),
* double math\_wrapper::IntegrFuncs:: MonteKarlo \_NotParal(FDelegate ^ fdelegate, double a, double b, int N),

Где:

1. fdelegate - функция.
2. a - нижняя граница.
3. b - нижняя граница.
4. N – количество шагов интегрирования.
5. Возвращает – значение интеграла.

## 6.7. Решение системы дифференциальных уравнений методом Эйлера.

* array<double>^ math\_wrapper::DiffFuncs::Eiler(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs::Eiler\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. a – начальное значение интервала.
3. b - конечное значение интервала.
4. N – количество шагов.
5. arr\_y – массив с начальными значениями.
6. count – количество уравнений.
7. Возвращает – массив с найденными значениями.

## 6.8. Решение системы дифференциальных уравнений методом Рунге-Кутта 2.

* array<double>^ math\_wrapper::DiffFuncs::RungeKutta2(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs:: RungeKutta2\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. a – начальное значение интервала.
3. b - конечное значение интервала.
4. N – количество шагов.
5. arr\_y – массив с начальными значениями.
6. count – количество уравнений.
7. Возвращает – массив с найденными значениями.

## 6.9. Решение системы дифференциальных уравнений методом Рунге-Кутта 4.

* array<double>^ math\_wrapper::DiffFuncs::RungeKutta4(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs:: RungeKutta4\_Paral(array<FDelegate ^>^ fdelegate, double a, double b, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. a – начальное значение интервала.
3. b - конечное значение интервала.
4. N – количество шагов.
5. arr\_y – массив с начальными значениями.
6. count – количество уравнений.
7. Возвращает – массив с найденными значениями.

## 6.10. Решение системы дифференциальных уравнений методом Эйлера. Перегрузка с tau.

* array<double>^ math\_wrapper::DiffFuncs::Eiler(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs:: Eiler \_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. tau – шаг.
3. N – количество шагов.
4. arr\_y – массив с начальными значениями.
5. count – количество уравнений.
6. Возвращает – массив с найденными значениями.

## 6.11. Решение системы дифференциальных уравнений методом Рунге-Кутта 2. Перегрузка с tau.

* array<double>^ math\_wrapper::DiffFuncs::RungeKutta2(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs:: RungeKutta2\_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. tau – шаг.
3. N – количество шагов.
4. arr\_y – массив с начальными значениями.
5. count – количество уравнений.
6. Возвращает – массив с найденными значениями.

## 6.12. Решение системы дифференциальных уравнений методом Рунге-Кутта 4. Перегрузка с tau.

* array<double>^ math\_wrapper::DiffFuncs::RungeKutta4(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),
* array<double>^ math\_wrapper::DiffFuncs:: RungeKutta4\_Paral(array<FDelegate ^>^ fdelegate, double tau, int N, array<double>^ arr\_y, int count),

Где:

1. fdelegate - массив функций.
2. tau – шаг.
3. N – количество шагов.
4. arr\_y – массив с начальными значениями.
5. count – количество уравнений.
6. Возвращает – массив с найденными значениями.

# 7. Перспективы развития

Добавление новых математических функций.