Содержание

ВВЕДЕНИЕ

Численные методы являются одним из мощных математических средств решения задачи. Простейшие численные методы используются повсеместно, однако существуют задачи, где без достаточно сложных численных методов не удалось бы получить ответа.

С развитием технологий большое распространение получили быстродействующие вычислительные машины (ЭВМ). В связи с этим появилась возможность осуществлять вычисления с заданной точностью за приемлемое время. Современные численные методы и мощные ЭВМ, выполняющие миллионы операций в секунду позволяют решать такие задачи, о которых до появления ЭВМ могли только мечтать.

Одной из важнейших задач численного анализа является задача интерполяции функции: требуется восстановить функцию f(x) для всех значений $x \in [a, b]$ если известны её значения в некотором конечном числе точек этого отрезка. Эти известные значения, как правило, находятся в результате наблюдений или измерений в каком – то эксперименте либо в результате каких – то вычислений.

Интерполяция применяется во многих задачах, связанных с вычислениями. Укажем некоторые из этих задач. Обработка физического эксперимента – построение приближенных формул по данным вычислительного эксперимента. Здесь возникают нестандартные задачи интерполяции, так как обычно пишутся формулы, возможно, более простой структуры.

Основная цель интерполяции – получить быстрый (экономичный) алгоритм вычисления значений f (x) для значений x, не содержащихся в таблице данных.

Целью данной работы является изучение интерполирования функций и её реализация на одном из языков программирования.

1 Сущность метода интерполяции

Сущность метода интерполяции заключается в нахождении прогнозных значений функций объекта $Y_i=f(x_j)$, где j=0,...n, в некоторых точках внутри отрезка $x_0,...x_n$ по известным значениям параметров в точках $x_o<x<x_n$.

Основные условия, предъявляемые к функциям при интерполяции:

- функция должна быть непрерывна и аналитична;
- для конкретного вида функций или их производных указаны такие неравенства, которые должны определить применимость интерполяции к данной функции;
- функция должна быть в достаточной степени гладкой, т.е. чтобы она обладала достаточным числом не слишком быстро возрастающих производных[1].

1.1 Определение термина интерполяции

Пусть для функции f(x), определенной на какой - либо части R, известны её значения на некотором конечном множестве точек $x_1, x_2, \ldots, x_n$ □ [a,b], и в этих точках функция f(x) определена как:

$$f(x_i) \equiv f_i, \quad i = \overline{0,n}$$

Требуется вычислить, хотя бы приближенно, значения при всех x.

Такая задача может возникнуть при проведении различных экспериментов, когда значения искомой функции определяются в дискретные моменты времени, либо в теории приближения, когда сложная функция сравнительно просто вычисляется при некоторых значениях аргумента, для функций заданных таблицей или графически и т. п. [2].

Обычно функцию $g(x_i)$, $x_i \in [a,b]$, $i = \overline{0,n}$, с помощью которой осуществляется приближение, находят так, чтобы:

$$g(x_i) = f_i \ , (i = \overline{0,n})$$

Такой способ приближения называют интерполяцией или

интерполированием. Точки x_1, x_2, …, x_n называют узлами интерполяции. Функцию $g(x_i)$, $i = \overline{0,n}$, называют интерполянтом.

Такие функции строятся на основе комбинаций из элементарных функций.

$$g(x) = \sum_{j=0}^{n} a_j \varphi_j(x), \quad i = \overline{0,n}$$

$\{\varphi_j(x)\}_0^n$ - фиксированная линейно- независимая система

$a_j \in R$ ($\{\varphi_j(x)\}_0^n$ $i = \overline{0,n}$) - пока неизвестные параметры

Математическая постановка задачи интерполирования заключается в следующем. Пусть **R** - пространство действительных функций, определенных на отрезке [a,b], и $\{\varphi_j(x)\}_0^n$ - заданная конечная или счетная система функций из **R**, такая, что их любая конечная подсистема является линейно-независимой. Для данной конечной совокупности точек x_1, x_2, …, x_n ($x_i \neq x_j$ при $i \neq j$), принадлежащих отрезку [a,b], и данной функции f(x) из **R** найти функцию φ, являющуюся линейной комбинацией функций $\{\varphi_j(x)\}_0^n$ так, чтобы в заданных точках значения f и φ совпадали [1]. Другими словами, определить константы a_1, a_2, …, a_n так, чтобы

$$\sum_{j=0}^{n} a_j \varphi_j(x_i) = f_i \ (i = \overline{0,n})$$

Совершенно ясно, почему число коэффициентов $\{a_i\}_0^n$ должно совпадать с числом узлов интерполяции x_i. Это нужно для того, чтобы матрица системы была квадратной (т.е. число неизвестных совпадало бы с числом условий, из которых находятся эти неизвестные). Кроме того, для однозначной разрешимости данной системы (при произвольной правой части) необходимо и достаточно, чтобы ее определитель был отличен от нуля, т. е. [1]:

$\forall \{x_i\}_0^n \subset [a,b] \ (x_k \neq x_j, k \approx j)$:

$$D_{n+1}=\begin{vmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \dots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \dots & \varphi_n(x_1) \\ \dots & \dots & \dots & \dots \\ \varphi_0(x_n) & \varphi_1(x_n) & \dots & \varphi_n(x_n) \end{vmatrix} \neq 0$$

Естественно, интерполянт необходимо построить в виде более легкой учетной функции, поэтому за часто берут такие системы как:

{1, x, x^2, …, x^n}, {1, sinx, cosx, sin2x, cos2x, …, sin(nx), cos(nx)} ,

{1, $e^{x\square 1}$, $e^{x\square 2}$, …, $e^{x\square n}$} (□i □R, □i≠□j (i≠j), n□N).

1.2 Алгебраический многочлен

Выберем в качестве базисных функций систему всех степеней

$\varphi_j(x)=x^j, j=0,1,2,\dots$

Будем считать все узлы x_n попарно различными. Докажем, что интерполяционный алгебраический многочлен $P_N(x)$ в этом случае является единственным. Предположим, что существуют два интерполяционных многочлена $P_N(x)$ и $\widetilde{P}_N(x)$. Они совпадают в N+1 точках x_n. Тогда их разность $P_N(x)$ - $\widetilde{P}_N(x)$ также является многочленом степени не выше N и обращается в нуль в (N+1)-й точке, т. е. имеет N+1 корень. Последнее невозможно, поскольку у такого многочлена имеется не более N корней. Это доказывает единственность алгебраического интерполяционного многочлена.

Помимо алгебраического многочлена, единственность обеспечивают и другие системы функций. В радиотехнических расчетах нередко используют интерполяцию тригонометрическими многочленами (так называемые формулы Бесселя). В задачах квантово-механического расчета молекул используют систему экспонент $\varphi_j(x)=\exp(a_j(x)), j=0,1,2,\dots$, но соответствующий алгоритм является специфическим [4].

1.3 Многочлен Ньютона

Алгебраический интерполяционный многочлен единственный, но существует много форм его записи. Они были предложены потому, что давали

некоторые преимущества в какихто частных случаях: например, если сетка равномерная или заранее известно число узлов, которое будет взято для построения многочлена, и т. п. Далее будет приведена форма Ньютона; она удобна тем, что пригодна на произвольной неравномерной сетке, позволяет увеличивать или уменьшать число узлов в ходе расчета, а также апостериорно оценивать достигнутую точность [4].

1.3.1 Разделение разности

Определим разделенные разности первого, второго и так далее порядков функции и(x) следующим образом:

$$u(x_0,x_1)=\frac{u(x_0)-u(x_1)}{x_0-x_1}$$

$$u(x_0,x_1,x_2)=\frac{u(x_0,x_1)-u(x_1,x_2)}{x_0-x_2}$$

и т. д. Общий закон легко виден: справа в числителе стоят дваразделенные разности предыдущего порядка со сдвинутыми на 1 индексами, а в знаменателе стоит разность крайних значений аргумента. Процесс построения разделенных разностей можно продолжать, пока у нас хватает точек.

Возьмем многочлен N-й степени $P_n(x)$ и N+1 точку x_n, 0 $0\leqslant n\leqslant N$. Добавим к этим точкам искомую точку x, поместив её в таблице впереди точки x_0. Построим разделенные разности этого многочлена по расширенной системе точек. Первая разделенная разность равна

$$P(x,x_0)=\frac{P(x)-P(x_0)}{x-x_0}$$

Здесь числитель обращается в нуль при $x=x_0$, поэтому многочлен $P(x)-P(x_0)$ нацело делится на $x-x_0$. Следовательно первая разделенная разность многочлена N-й степени является многочленом (N-1)й степени от x.

Вторая разделенная разность равна

$$P(x,x_0,x_1)=\frac{P(x,x_0)-P(x_0,x_1)}{x-x_1}$$

Если $x=x_1$, то этот числитель также обращается в нуль, значит, он нацело

делится на x-x_1. При этом образуется многочлен степени N-2 по каждому из трех аргументов.

Продолжим этот процесс. Когда подключается (N-1)я точка, получается (N-1) разделенная разность, являющаяся многочленом нулевой степени, т. е. константой. Поэтому при подключении N-й точки правая часть обратится в нуль:

$$P(x, x_0, \ldots, x_N) = 0$$

Таким образом, процесс построения разделенных разностей для многочлена исчерпывается: он не может использовать дальнейшие точки таблицы [4].

1.3.2 Форма Ньютона

Восстановим интерполяционный многочлен по его разделенным разностям. Для этого перепишем формулы:

$P(x, x_0) = \frac{P(x) - P(x_0)}{x - x_0}$ - $P(x, x_0, \ldots, x_N) = 0$ в следующем виде:

$$P(x) = P(x_0) + (x - x_0) P(x, x_0);$$

$$P(x, x_0) = P(x_0, x_1) + (x - x_1) P(x, x_0, x_1);$$

$$P(x, x_0, x_1) = P(x_0, x_1, x_2) + (x - x_2) P(x, x_0, x_1, x_2);$$

$$P(x, x_0, x_1, \ldots, x_N) = 0.$$

Подставив здесь каждую последующую формулу в предыдущую, получим выражение многочлена через его разделенные разности:

$$P(x) = P(x_0) + \sum_{n=1}^{N} (x - x_0)(x - x_1) \ldots (x - x_{n-1} P(x_0, x_1, \ldots, x_n))$$

Здесь разделенные разности в правой части содержат только табулированные узлы; узел x в них не входит. Поскольку эти разделенные разности выражаются через табулированные выражения многочлена, это позволяет восстановить многочлен в (N + 1)-м узле.

Интерполяционный многочлен по определению совпадает с функцией и(x) в (N + 1)-м выбранном узле.

Это дает

$$\omega_x = \prod_{k=0}^{n-1} (x - x_k)$$

Видно,что формула применима на произвольной неравномерной сетке, а число членов можно увеличивать или уменьшать, если исходные таблицы содержат достаточно много точек.

Для вычислений *б*ерется исходная таблица из N+1 узла x_n, $0 \leqslant n \leqslant N$. По каждой паре соседних узлов вычисляется разделенная разность первого порядка. Таких разностей будет N. По каждой паре соседних разделенных разностей первого порядка вычисляется разделенная разность второго порядка; их будет N -1. Продолжив этот процесс, доходим до единственной разделенной разности N-го порядка. Таким образом, таблица разделенных разностей будет треугольной. Но для окончательных вычислений достаточно хранить лишь одну верхнюю строку разделенных разностей всех порядков. Остальная часть таблицы является промежуточным результатом и служит только для вычисления этой строки [4].

1.4 Обратная интерполяция

В практике часто требуются обратные функции. Но даже если известно явное выражение функции u(x), найти явное выражение для обратной функции x(u) далеко не всегда возможно. Однако численно найти обратную функцию очень легко. Для этого выберем достаточно подробную сетку узлов $\{x_n\}$, покрывающую весь необходимый диапазон изменения аргумента. Вычисляя прямую функцию, составляем таблицу $u_n = u(x_n)$. Меняем столбцы этой таблицы местами и получаем таблицу (u_n,x_n). Если рассматривать u как аргумент, а x как функцию, то это будет таблица значений обратной функции.

Тем самым обратная функция x(u) табулирована. В узлах u_n она известна. Для проиежуточных значений u её можно найти, строя по обратной таблице интерполяционный многочлен Ньютона.

Этот прием называется обратной интерполяцией [4].

1.5 Сплайн нтерполяция

Для построения графиков функции по точкам с помощью лекал стараются выбрать лекало, на которое попадает как можно больше точек графика. Еще лучший результат достигается при использовании гибкого бруска (металлической линейки, поставленной на ребро), который прикладывают к точкам графика.

Уравнение гибкого бруска было написано еще Бернулли. Шонберг приближенно заменил решение этого уравнения полиномом третьей степени. Между каждой парой соседних узлов строился свой полином, а в узлах соседние полиномы «склеивались» так, чтобы обеспечить максимально возможную гладкость интерполяции.

Описанный прием быстро распространился и был обобщен на случай полинома большей степени и даже неполимиальной интерполяции. Этот прием — интерполирование функции кусочно-полиномиальной функциеё получил название сплайн интерполяции (от англ. Spline – гибкий брусок).

Пусть задана сетка $\{x_n, 0 \leqslant n \leqslant N\}$; точки x_n называют узлами сплайна. Полиномиальным сплайном $S_p(x)$ дефекта q называется функция, удовлетворяющая следующим требованиям [4]:

1. $S_p(x)$ на каждом интервале $[x_{n-1}, x_n]$ является полиномом степени p
2. Эти полиномы «склеены» во внутренних узлах так, что сплайн остается непрерывным вместе со своими p – q производными:

$$S_p^{(k)}(x_n - 0) = S_p^{(k)}(x_n - 0), 0 \leqslant k \leqslant p - q, 1 \leqslant n \leqslant N - 1$$

Чаще всего ограничиваются сплайнами дефекта q = 1, когда разрывна лишь старшая (p-я) производная, а все младшие производные непрерывны. В этом случае говорят просто о сплайне степени p, опуская упоминания о дефекте.

Если сплайн в заданных точках совпадает с табулированной функцией, то такой сплайн называется интерполяционным.

На практике наибольшее распространение получили кубические сплайны

$p=3$ $s(x)=s_j(x)=a_j+b_j(x-x_j)+c_j(x-x_j)^2/2+d_j(x-x_j)^3/6,\ \ j=\overline{1,n}$.

Построить сплайн – найти коэффициенты a_j,b_j,c_j,d_j. При этом задаются различные условия гладкости, а значит и дефекты. Пусть $s(x)\in C^2[a,b]$ т.е. дефект сплайна $r=1$. Тогда для определения коэффициентов a_j,b_j,c_j,d_j имеем условия, интерполяции $s(x_j)=f_j,\ j=\overline{0,n}$, непрерывности функций $s(x),\ s'(x),\ s''(x),$ в точках $x_j,\ j=\overline{1,n-1}$, ограничения в крайних точках (зависят от конкретных условий приближения функции), например, $s''(a)=s''(b)=0$.

Эти условия приводят к замкнутой системе уравнений для определения a_j,b_j,c_j,d_j:

$$\begin{cases} a_j=f_j, \\ hb_j-h^2c_j/2+h^3d_j/6=f_j-f_{j-1},\ j=\overline{1,n}, \\ hc_j-h^2d_j/2=b_j-b_{j-1},\ \ hd_j=c_j-c_{j-1},\ \ j=\overline{2,n}, \\ c_n=0,\ c_1-hd_1=0. \end{cases} \qquad (1)$$

Из (1), исключая коэффициенты a_j,b_j,d_j получаем систему линейных алгебраических уравнений (СЛАУ) с неизвестными c_j

$$\begin{cases} c_{j+1}+4c_j+c_{j-1}=\dfrac{6(f_{j+1}-2f_j+f_{j-1})}{h^2},\quad j=\overline{1,n-1} \\ c_0=c_n=0. \end{cases} \qquad (2)$$

Введя обозначения $v_j=\dfrac{6(f_{j+1}-2f_j+f_{j-1})}{h^2},\quad j=\overline{1,n-1}$, представим СЛАУ в векторном виде

$$\begin{bmatrix} 4 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & 0 & 0 & \cdots & 1 & 4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \cdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \cdots \\ v_{n-1} \end{bmatrix} \qquad (3)$$

Решаем (2) и получаем коэффициенты

$c_j,\ j=\overline{1,n-1},\ c_0=c_n=0.$

Затем из (1) находим коэффициенты a_j, b_j, d_j по явным формулам

$$a_j=f_j, \qquad d_j=\frac{c_j-c_{j-1}}{h}, \qquad b_j=\frac{h}{2}c_j-\frac{h^2}{6}d_j+\frac{f_j-f_{j-1}}{h}, \quad j=\overline{1,n}.$$

Таким образом получаем матрицу $\mathbf{S}$(двумерный массив)размера $n\times 4$ для всех коэффициентов $a_j,b_j,c_j,d_j,\ j=\overline{1,n}$.

Для каждого заданного промежуточного значения $x\in[a,b]$ находим $j-$ отрезок $x_{j-1}\le x\le x_j$ и вычисляем значение $s(x)\approx f(x)$ с использованием $j-$ой строки матрицы $\mathbf{S}$, в виде

$$s_j(x)=((\frac{d_j}{6}(x-x_j)+\frac{c_j}{2})(x-x_j)+b_j)(x-x_j)+a_j,\ \ x_{j-1}\le x\le x_j,\ \ j=\overline{1,n}$$

2 Интерполяция кубическим сплайном

Реализация алгоритма построения спплайна, а также графического интерфейса пользователя осущетвлялась при помощи языка программирования Managed C++ с общеязыковой исполняющей средой Common Language Runtime (CLR). CLR является одним из основных компонентов пакета Microsoft.

Для хранения вводных значений для задачи интерполяции был выбран следующий формат файла:

1. В заголовке файла указывается количество значений, которое необходимо считать.
2. Ниже записывается количество пар чисел указанных в заголовке файла. Первое число содержит значение аргумента неизвестной функции, второе значение самой функции от этого аргумента.

Для хранения значений считанных из класса был создан класс описывающий точки из входного файла. Его контракт представлен ниже:

```
class knot
{
public:
	knot();
	double x, f;
	void Add(double arg, double func);
};
```

Загрузка вводных значений осуществляется при помощи метода `Load_Data()`

```
int Load_Data()
{
	OpenFileDialog^ of = gcnew OpenFileDialog();
	isFileOpened = false;
	if (of->ShowDialog() != System::Windows::Forms::DialogResult::OK){
		return -1;
	}
	char* FileName;
	int i = 0;
	double ii;
	FILE *File;
```

```
	IntPtr ptrToNativeString = Marshal::StringToHGlobalAnsi(of->FileName);
	FileName = static_cast<char*>(ptrToNativeString.ToPointer());
	if (!fopen_s(&File, FileName, "r"))
	{
		isFileOpened = true;
	}
	else
	{
		MessageBox::Show("File don't exists");
		return -1;
	}
	double x, f;
	fscanf(File, "%d", &n);
	KnotArray = new knot[n];
	while (!feof(File))
	{
		fscanf(File, "%lf%lf", &x, &f);
		KnotArray[i].Add(x, f);
		i++;
		if (i == n + 1)
			return -1;
	}
	fclose(File);
	return 1;
}
```

Вычисление значений коэффициентов сплайна производится с помощью метода прогонки для трехдиагональной матрицы. Для начала необходимо вычислить прогоночные коэффциенты, это прямой ход прогонки. Далее по подготовленным прогоночным коэффициентам выполняется обратный ход прогонки и вычисление недостающих коэффициентов. Обратный ход прогонки — нахождение решения реализован в методе `SolveTriDiag`

```
void SolveTriDiag(double** TDM, double* F)
{
	double* alph = new double[n - 1];
	double* beta = new double[n - 1];
	int i;

	alph[0] = -TDM[2][0] / TDM[1][0];
	beta[0] = F[0] / TDM[1][0];

	for (i = 1; i < n - 1; i++)
	{
```

```
            alph[i] = -TDM[2][i] / (TDM[1][i] + TDM[0][i] * alph[i - 1]);
            beta[i] = (F[i] - TDM[0][i] * beta[i - 1]) / (TDM[1][i] + TDM[0][i] * alph[i -
1]);
      }
      b[n - 1] = (F[n - 1] - TDM[0][n - 1] * beta[n - 2]) / (TDM[1][n - 1] + TDM[0][n - 1]
* alph[n - 2]);

      for (i = n - 2; i > -1; i--)
      {
            b[i] = b[i + 1] * alph[i] + beta[i];
      }
}
```

Все необходимые шаги реализованы в методе `BuildSpline`.

```
Int BuildSpline(){
      for (i = 0; i < n - 1; i++)
      {
            a[i] = KnotArray[i].f;
            h[i] = KnotArray[i + 1].x - KnotArray[i].x;
            delta[i] = (KnotArray[i + 1].f - KnotArray[i].f) / h[i];
            TriDiagMatrix[0][i] = i > 0 ? h[i] : x3;
            f[i] = i > 0 ? 3 * (h[i] * delta[i - 1] + h[i - 1] * delta[i]) : 0;
      }
      TriDiagMatrix[1][0] = h[0];
      TriDiagMatrix[2][0] = h[0];
      for (int i = 1; i < n - 1; i++)
      {
            TriDiagMatrix[1][i] = 2 * (h[i] + h[i - 1]);
            TriDiagMatrix[2][i] = h[i];
      }
      TriDiagMatrix[1][n - 1] = h[n - 2];
      TriDiagMatrix[2][n - 1] = xn;
      TriDiagMatrix[0][n - 1] = h[n - 2];

      i = n - 1;
      f[0] = ((h[0] + 2 * x3)*h[1] * delta[0] + powf(h[0], 2)*delta[1]) / x3;
      f[n - 1] = (powf(h[i - 1], 2)*delta[i - 2] + (2 * xn + h[i - 1])*h[i - 2] * delta[i -
      1]) / xn;

      SolveTriDiag(TriDiagMatrix, f);

      Coef = new double*[n];

      for (int iter = 0; iter < n; iter++)
      {
            Coef[iter] = new double[n - 1];
      }
      int j;
      for (j = 0; j < n - 1; j++)
      {
            d[j] = (b[j + 1] + b[j] - 2 * delta[j]) / (h[j] * h[j]);
            c[j] = 2 * (delta[j] - b[j]) / h[j] - (b[j + 1] - delta[j]) / h[j];

            Coef[j][0] = a[j];
            Coef[j][1] = b[j];
            Coef[j][2] = c[j];
            Coef[j][3] = d[j];
      }

}
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А.А.Самарский. Введение в численные методы. М.: «Наука», 1987.
2. Бахвалов Н.С. Численные методы. Главная редакция физико-математической литературы изд-ва «Наука», М., 1975 г
3. Калиткин Н.Н. Численные методы.
4. Калиткин Н.Н. Альшина Е.А. Численные методы в двух книгах. Москва Издательский центр Академия, 2013

ПРИЛОЖЕНИЕ А

Листинг программы

```
#pragma once
#include <math.h>
#include <stdio.h>
#include "knot.h"
#include "vector.h"

namespace PrimerMeltingT {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::Runtime::InteropServices;

	public ref class Form1 : public Form
	{
	private:

		knot* KnotArray;
		int n;
		double** Coef;
		private: System::Windows::Forms::ToolStripMenuItem^  открытьToolStripMenuItem;
		private: System::Windows::Forms::ToolStripMenuItem^  выходToolStripMenuItem;
		private: System::Windows::Forms::ToolStripMenuItem^
оПрограммеToolStripMenuItem1;
		double* b;
	private: System::Windows::Forms::DataVisualization::Charting::Chart^  chart1;
	private: System::Windows::Forms::ToolStripMenuItem^
нарисоватьСинусToolStripMenuItem;

			 bool isFileOpened = false;

		void SolveTriDiag(double** TDM, double* F)
		{
			double* alph = new double[n - 1];
			double* beta = new double[n - 1];

			int i;

			alph[0] = -TDM[2][0] / TDM[1][0];
			beta[0] = F[0] / TDM[1][0];

			for (i = 1; i < n - 1; i++)
			{
				alph[i] = -TDM[2][i] / (TDM[1][i] + TDM[0][i] * alph[i - 1]);
				beta[i] = (F[i] - TDM[0][i] * beta[i - 1]) / (TDM[1][i] + TDM[0]
[i] * alph[i - 1]);
			}
			b[n - 1] = (F[n - 1] - TDM[0][n - 1] * beta[n - 2]) / (TDM[1][n - 1] +
TDM[0][n - 1] * alph[n - 2]);

			for (i = n - 2; i > -1; i--)
```

```
		{
			b[i] = b[i + 1] * alph[i] + beta[i];
		}
	}

	int BuildSpline()
	{
		double* a = new double[n - 1];
		double* c = new double[n - 1];
		double* d = new double[n - 1];
		double* delta = new double[n - 1];
		double* h = new double[n - 1];
		double** TriDiagMatrix = new double*[3];

		b = new double[n];

		TriDiagMatrix[0] = new double[n];
		TriDiagMatrix[1] = new double[n];
		TriDiagMatrix[2] = new double[n];

		double* f = new double[n];
		double x3, xn;
		int i;

		if (n < 3)
			return -1;

		x3 = KnotArray[2].x - KnotArray[0].x;
		xn = KnotArray[n - 1].x - KnotArray[n - 3].x;

		for (i = 0; i < n - 1; i++)
		{
			a[i] = KnotArray[i].f;
			h[i] = KnotArray[i + 1].x - KnotArray[i].x;
			delta[i] = (KnotArray[i + 1].f - KnotArray[i].f) / h[i];
			TriDiagMatrix[0][i] = i > 0 ? h[i] : x3;
			f[i] = i > 0 ? 3 * (h[i] * delta[i - 1] + h[i - 1] * delta[i]) :
0;
		}
		TriDiagMatrix[1][0] = h[0];
		TriDiagMatrix[2][0] = h[0];
		for (int i = 1; i < n - 1; i++)
		{
			TriDiagMatrix[1][i] = 2 * (h[i] + h[i - 1]);
			TriDiagMatrix[2][i] = h[i];
		}
		TriDiagMatrix[1][n - 1] = h[n - 2];
		TriDiagMatrix[2][n - 1] = xn;
		TriDiagMatrix[0][n - 1] = h[n - 2];


		i = n - 1;
		f[0] = ((h[0] + 2 * x3)*h[1] * delta[0] + powf(h[0], 2)*delta[1]) / x3;
		f[n - 1] = (powf(h[i - 1], 2)*delta[i - 2] + (2 * xn + h[i - 1])*h[i -
2] * delta[i - 1]) / xn;

		SolveTriDiag(TriDiagMatrix, f);

		Coef = new double*[n];

		for (int iter = 0; iter < n; iter++)
		{
```

```
                        Coef[iter] = new double[n - 1];
                }
                int j;
                for (j = 0; j < n - 1; j++)
                {
                        d[j] = (b[j + 1] + b[j] - 2 * delta[j]) / (h[j] * h[j]);
                        c[j] = 2 * (delta[j] - b[j]) / h[j] - (b[j + 1] - delta[j]) /
h[j];

                        Coef[j][0] = a[j];
                        Coef[j][1] = b[j];
                        Coef[j][2] = c[j];
                        Coef[j][3] = d[j];
                }
        }

        double Interpolate(double x)
        {
                //double result;
                int i = 0;

                while (KnotArray[i].x <= x)
                        i++;
                i--;
                return Coef[i][0] + Coef[i][1] * (x - KnotArray[i].x) + Coef[i][2] *
powf((x - KnotArray[i].x), 2) + Coef[i][3] * powf((x - KnotArray[i].x), 3);

        }

        int Load_Data()
        {
                OpenFileDialog^ of = gcnew OpenFileDialog();

                isFileOpened = false;

                if (of->ShowDialog() != System::Windows::Forms::DialogResult::OK){
                        return -1;
                }

                char* FileName;
                int i = 0;
                double ii;
                FILE *File;
                IntPtr ptrToNativeString = Marshal::StringToHGlobalAnsi(of->FileName);
                FileName = static_cast<char*>(ptrToNativeString.ToPointer());
                if (!fopen_s(&File, FileName, "r"))
                {
                        isFileOpened = true;
                }
                else
                {
                        MessageBox::Show("File don't exists");
                        return -1;
                }
                double x, f;
                fscanf(File, "%d", &n);
                KnotArray = new knot[n];
                while (!feof(File))
                {
                        fscanf(File, "%lf%lf", &x, &f);
                        KnotArray[i].Add(x, f);
                        i++;
```

```
                        if (i == n + 1)
                                return -1;
                }
                fclose(File);
                return 1;
        }

    public:
        Form1(void)
        {
                InitializeComponent();
                /*
                if (Load_Data() != -1)
                {

                scanf("%lf", &x);
                printf("x:  %lf\nans:   %lf\n", x, Interpolate(x));
                scanf("%lf", &x);

                }*/
        }

    protected:
        ~Form1()
        {
                if (components)
                {
                        delete components;
                }
        }
    private: System::Windows::Forms::MenuStrip^  menuStrip1;
    protected:
    private: System::Windows::Forms::ToolStripMenuItem^  файлToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^  оПрограммеToolStripMenuItem;
    private: System::Windows::Forms::ContextMenuStrip^  contextMenuStrip1;
    private: System::ComponentModel::IContainer^  components;
    private: Boolean isDrawSin = false;
    protected:

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>


#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {
                this->components = (gcnew System::ComponentModel::Container());
                System::Windows::Forms::DataVisualization::Charting::ChartArea^
chartArea2 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());
                System::Windows::Forms::DataVisualization::Charting::Legend^  legend2 =
(gcnew System::Windows::Forms::DataVisualization::Charting::Legend());
                this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
                this->файлToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                this->открытьToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
```

```
                    this->выходToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->оПрограммеToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->оПрограммеToolStripMenuItem1 = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->contextMenuStrip1 = (gcnew
System::Windows::Forms::ContextMenuStrip(this->components));
                    this->chart1 = (gcnew
System::Windows::Forms::DataVisualization::Charting::Chart());
                    this->нарисоватьСинусToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                    this->menuStrip1->SuspendLayout();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>chart1))->BeginInit();
                    this->SuspendLayout();
                    //
                    // menuStrip1
                    //
                    this->menuStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(4) {
                            this->файлToolStripMenuItem,
                                    this->оПрограммеToolStripMenuItem, this-
>нарисоватьСинусToolStripMenuItem, this->оПрограммеToolStripMenuItem1
                    });
                    this->menuStrip1->Location = System::Drawing::Point(0, 0);
                    this->menuStrip1->Name = L"menuStrip1";
                    this->menuStrip1->Size = System::Drawing::Size(552, 24);
                    this->menuStrip1->TabIndex = 0;
                    this->menuStrip1->Text = L"menuStrip1";
                    //
                    // файлToolStripMenuItem
                    //
                    this->файлToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {
                            this->открытьToolStripMenuItem,
                                    this->выходToolStripMenuItem
                    });
                    this->файлToolStripMenuItem->Name = L"файлToolStripMenuItem";
                    this->файлToolStripMenuItem->Size = System::Drawing::Size(48, 20);
                    this->файлToolStripMenuItem->Text = L"Файл";
                    //
                    // открытьToolStripMenuItem
                    //
                    this->открытьToolStripMenuItem->Name = L"открытьToolStripMenuItem";
                    this->открытьToolStripMenuItem->Size = System::Drawing::Size(121, 22);
                    this->открытьToolStripMenuItem->Text = L"Открыть";
                    this->открытьToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::открытьToolStripMenuItem_Click);
                    //
                    // выходToolStripMenuItem
                    //
                    this->выходToolStripMenuItem->Name = L"выходToolStripMenuItem";
                    this->выходToolStripMenuItem->Size = System::Drawing::Size(121, 22);
                    this->выходToolStripMenuItem->Text = L"Выход";
                    this->выходToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::выходToolStripMenuItem_Click);
                    //
                    // оПрограммеToolStripMenuItem
                    //
                    this->оПрограммеToolStripMenuItem->Name =
L"оПрограммеToolStripMenuItem";
```

```
                    this->оПрограммеToolStripMenuItem->Size = System::Drawing::Size(118,
20);
                    this->оПрограммеToolStripMenuItem->Text = L"Интерполировать";
                    this->оПрограммеToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::оПрограммеToolStripMenuItem_Click);
                    //
                    // оПрограммеToolStripMenuItem1
                    //
                    this->оПрограммеToolStripMenuItem1->Name =
L"оПрограммеToolStripMenuItem1";
                    this->оПрограммеToolStripMenuItem1->Size = System::Drawing::Size(94,
20);
                    this->оПрограммеToolStripMenuItem1->Text = L"О программе";
                    this->оПрограммеToolStripMenuItem1->Click += gcnew
System::EventHandler(this, &Form1::оПрограммеToolStripMenuItem1_Click);
                    //
                    // contextMenuStrip1
                    //
                    this->contextMenuStrip1->Name = L"contextMenuStrip1";
                    this->contextMenuStrip1->Size = System::Drawing::Size(61, 4);
                    //
                    // chart1
                    //
                    chartArea2->Name = L"ChartArea1";
                    this->chart1->ChartAreas->Add(chartArea2);
                    this->chart1->Dock = System::Windows::Forms::DockStyle::Fill;
                    legend2->Name = L"Legend1";
                    this->chart1->Legends->Add(legend2);
                    this->chart1->Location = System::Drawing::Point(0, 24);
                    this->chart1->Name = L"chart1";
                    this->chart1->Size = System::Drawing::Size(552, 295);
                    this->chart1->TabIndex = 1;
                    this->chart1->Text = L"chart1";
                    //
                    // нарисоватьСинусToolStripMenuItem
                    //
                    this->нарисоватьСинусToolStripMenuItem->Name =
L"нарисоватьСинусToolStripMenuItem";
                    this->нарисоватьСинусToolStripMenuItem->Size =
System::Drawing::Size(119, 20);
                    this->нарисоватьСинусToolStripMenuItem->Text = L"Нарисовать синус";
                    this->нарисоватьСинусToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::нарисоватьСинусToolStripMenuItem_Click_1);
                    //
                    // Form1
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->BackColor = System::Drawing::Color::White;
                    this->ClientSize = System::Drawing::Size(552, 319);
                    this->Controls->Add(this->chart1);
                    this->Controls->Add(this->menuStrip1);
                    this->MainMenuStrip = this->menuStrip1;
                    this->Name = L"Form1";
                    this->menuStrip1->ResumeLayout(false);
                    this->menuStrip1->PerformLayout();
                    (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>chart1))->EndInit();
                    this->ResumeLayout(false);
                    this->PerformLayout();

                }
```

```
#pragma endregion

	private: System::Void открытьToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
		Load_Data();
	}
	private: System::Void оПрограммеToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
		if (isFileOpened){
			double min = 0;
			double max = 0;

			min = KnotArray[0].x;
			max = KnotArray[n - 1].x;

			this->Text = fabs(KnotArray[0].x - KnotArray[1].x).ToString();

			BuildSpline();
			chart1->Series->Clear();
			chart1->Series->Add("Interpolation");
			chart1->Series["Interpolation"]->ChartType =
DataVisualization::Charting::SeriesChartType::Line;
			for (double i = min; i < max; i+=0.1){
				chart1->Series["Interpolation"]->Points->AddXY(i,
Interpolate(i));
			}

			chart1->Series->Add("Dots");
			chart1->Series["Dots"]->ChartType =
DataVisualization::Charting::SeriesChartType::Point;
			for (int i = 0; i < n; i++){
				chart1->Series["Dots"]->Points->AddXY(KnotArray[i].x,
KnotArray[i].f);
			}

			if (isDrawSin)
			{
				chart1->Series->Add("Graphic");
				chart1->Series["Graphic"]->ChartType =
DataVisualization::Charting::SeriesChartType::Line;
				for (double i = min; i < max; i += 0.1){
					chart1->Series["Graphic"]->Points->AddXY(i, sinf(i));
				}
			}
		}
	}
	private: System::Void выходToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
		Application::Exit();
	}
	private: System::Void оПрограммеToolStripMenuItem1_Click(System::Object^  sender,
System::EventArgs^  e) {
		MessageBox::Show("Интерполирование данных методом кубического сплайна\nСоздал
Смирнов Сергей\n2015 (C)");
}

	private: System::Void нарисоватьСинусToolStripMenuItem_Click_1(System::Object^
sender, System::EventArgs^  e) {
		isDrawSin ^= 1;
	}
};
};
```

```
#pragma once
class knot
{
public:
	knot();
	double x, f;
	void Add(double arg, double func);
};

#pragma once
class vector
{
public:
	vector();
	double* x;
	void Add(int m);
};

#include "stdafx.h"
#include "knot.h"


knot::knot()
{
}

void knot::Add(double arg, double func){
	x = arg;
	f = func;
}

#include "stdafx.h"
#include "vector.h"


vector::vector()
{
}

void vector::Add(int m){
	x = new double[m];
}

// PrimerMeltingT.cpp : main project file.

#include "stdafx.h"
#include "Form1.h"

using namespace PrimerMeltingT;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	// Enabling Windows XP visual effects before any controls are created
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	// Create the main window and run it
	Application::Run(gcnew Form1());
	return 0;
}
```

ПРИЛОЖЕНИЕ Б

Скриншоты программы

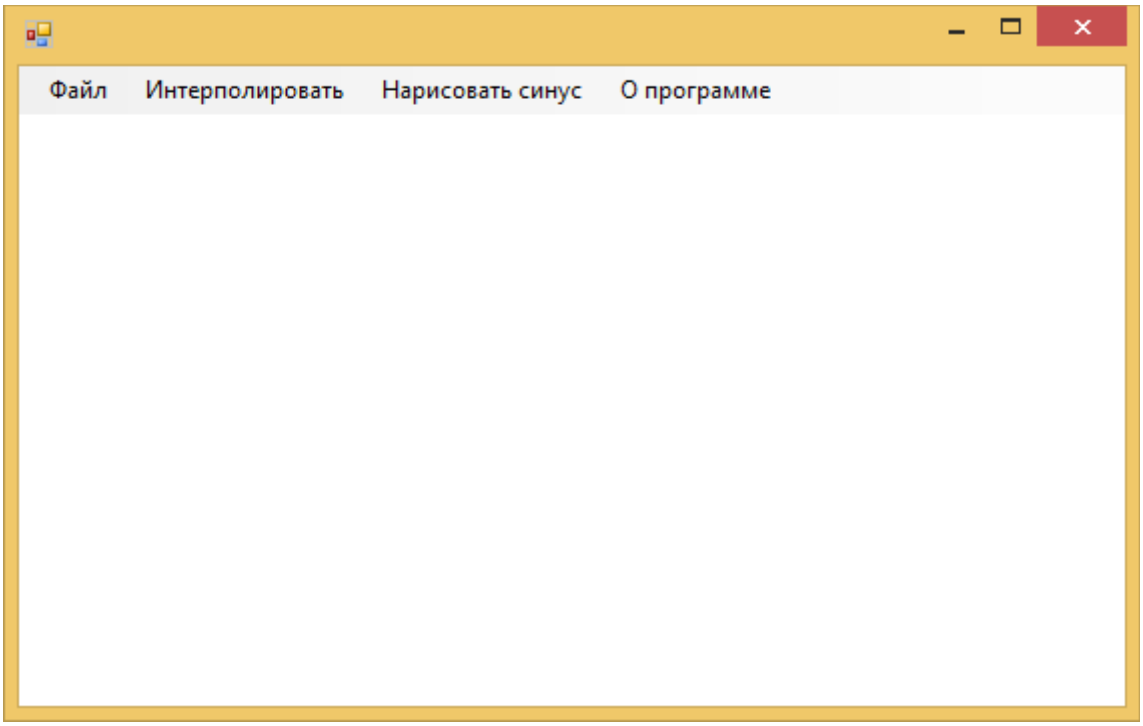

Рисунок 1. Главное окно программы.

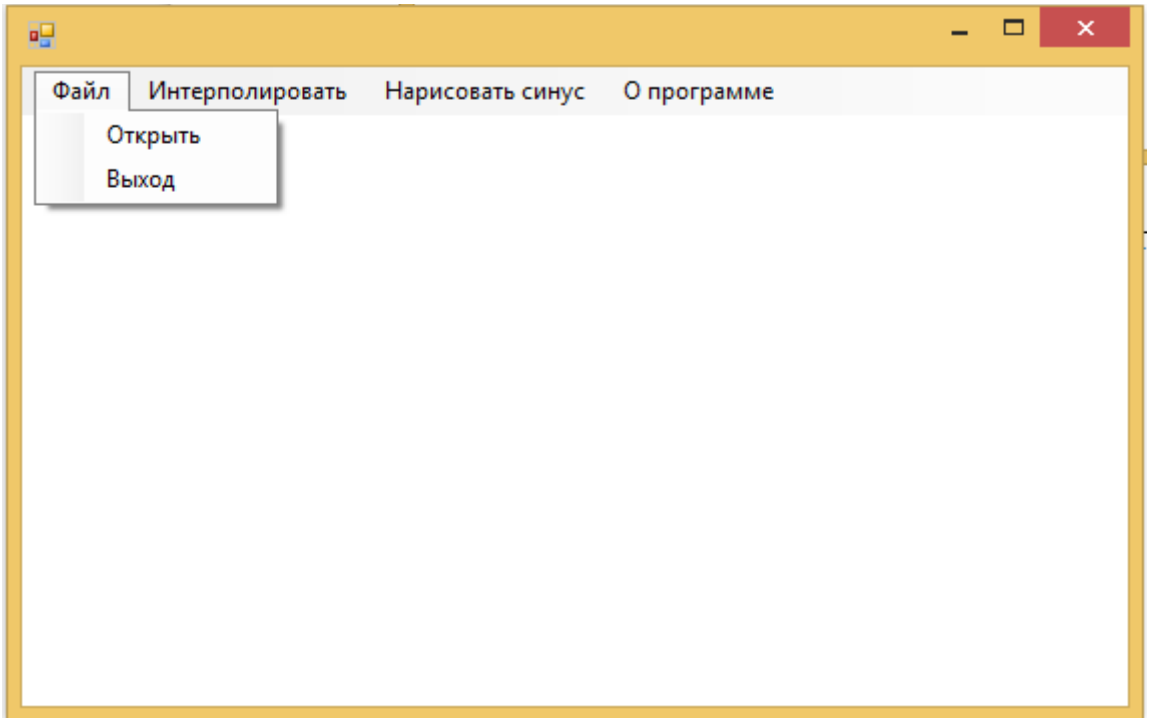

Рисунок 2. Меню выбора файла.

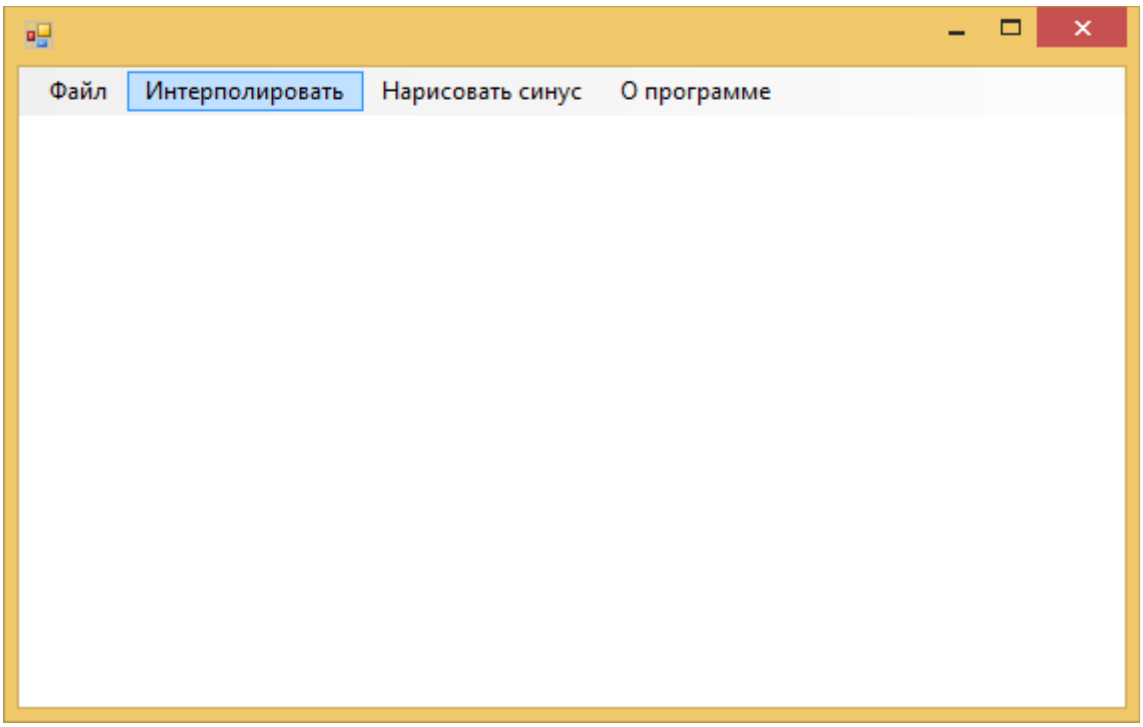

Рисунок 3. Запуск обработки выбранного файла с данными.

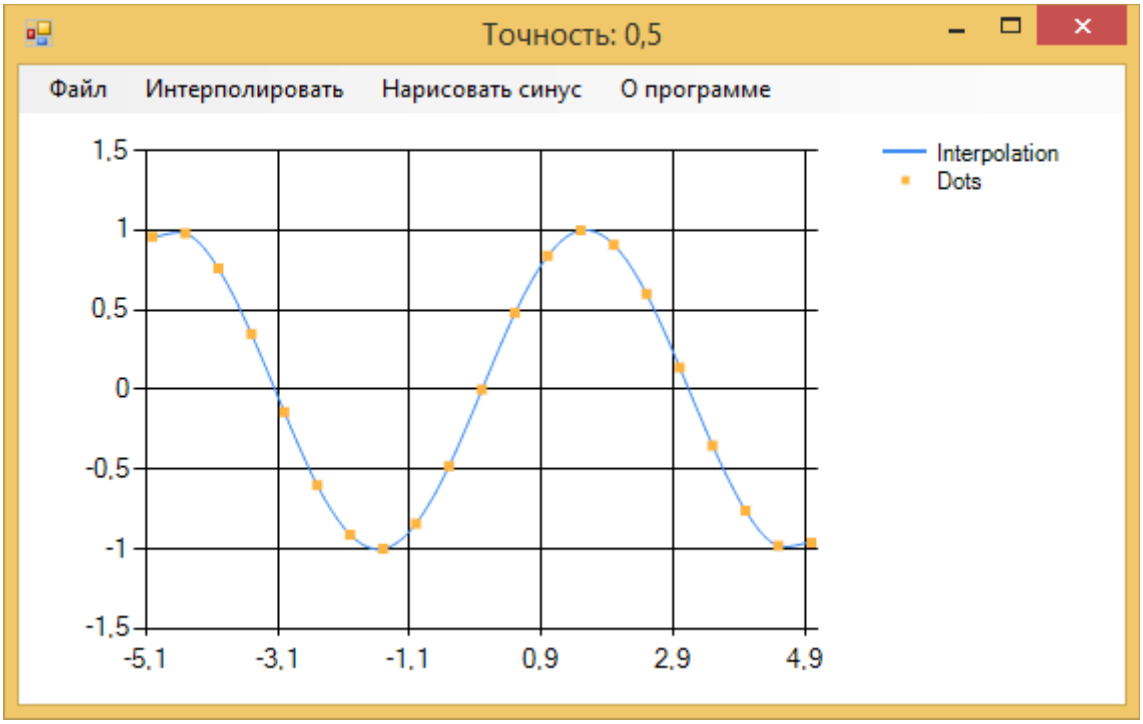

Рисунок 4. Результат интерполирования sin(x) с шагом 0.5.

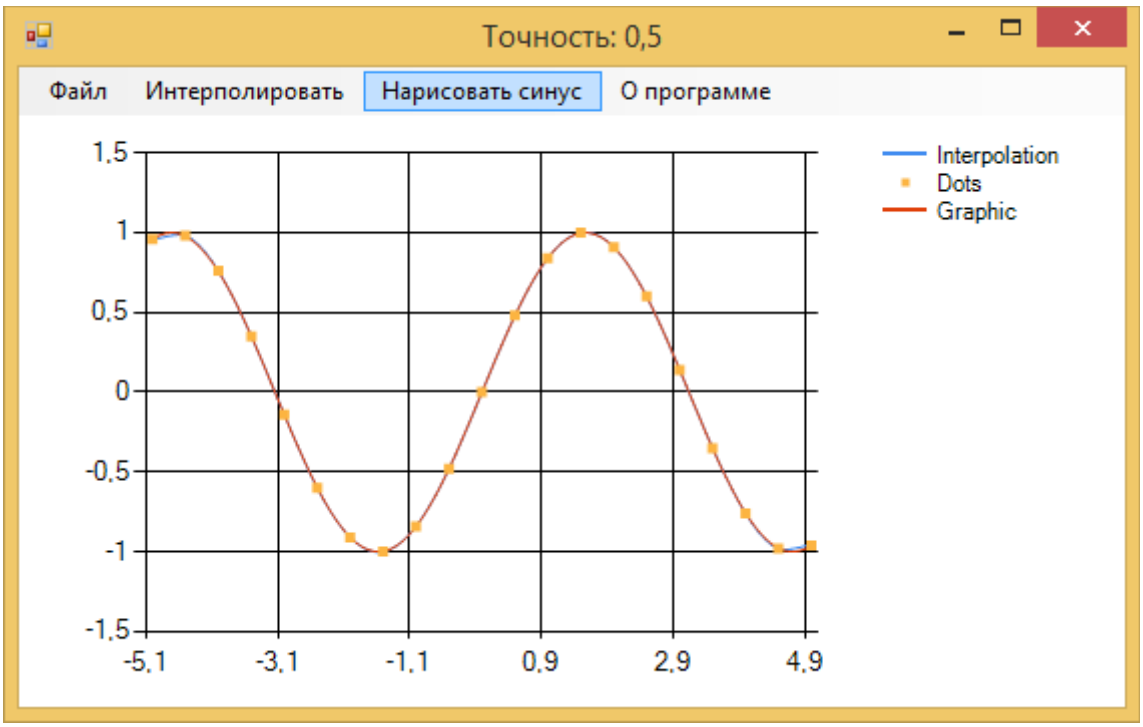

Рисунок 5. Обрисовка графика sin(x) поверх полученных результатов.

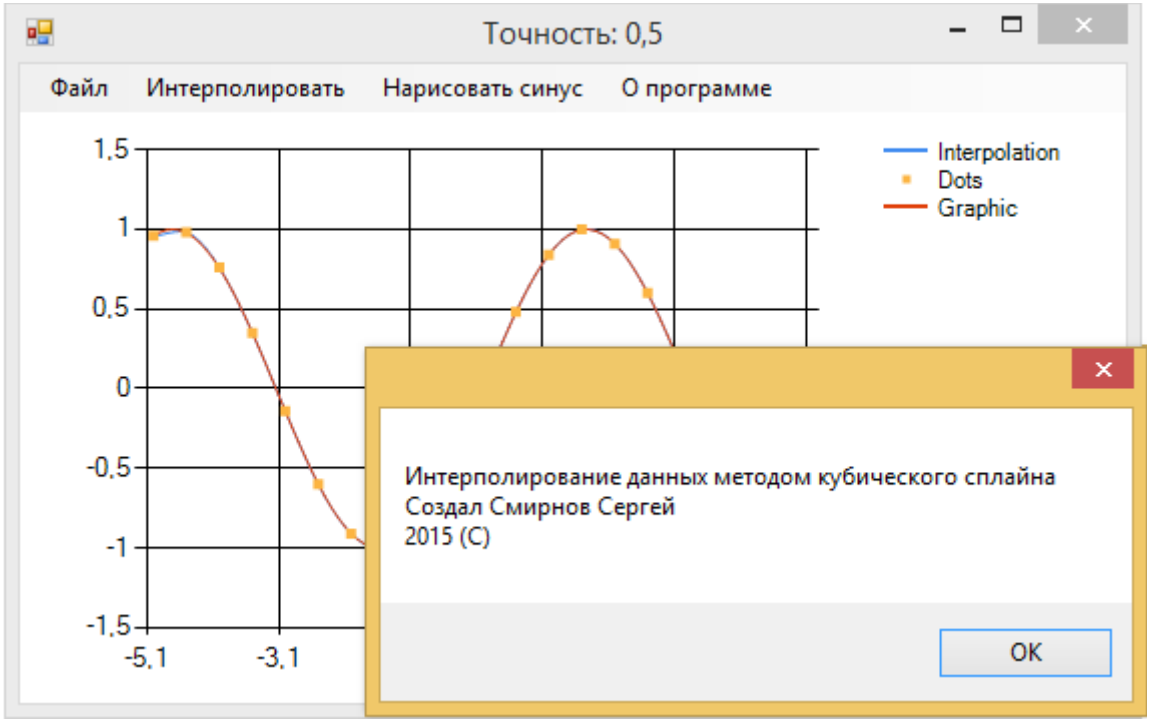

Рисунок 6. Информация о программе.