

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики
Кафедра прикладной механики и информатики

Отчет по лабораторной работе №2

на тему:

**«Численные исследования метода сходимости многочлена Ньютона к
приближаемой функции при увеличении степени многочлена»**

Выполнил: студент 3 к. 1 гр. ПМИ в.о.

Бедарев Анатолий Андреевич

Проверил: к.ф-м.н, доц.

Гудович Николай Николаевич

Воронеж – 2017

Содержание

Постановка задачи	3
Указания к выполнению лабораторной работы	4
Ход выполнения работы	7
Выводы по работе	21
Список литературы	22
Приложение (листинг)	23

Постановка задачи

1. Составить и отладить программу приближенного нахождения значения функции с использованием интерполяционного многочлена Ньютона. Запрограммировать вычисление многочлена Ньютона в произвольной точке x^* отрезка $[a, b]$ следующими тремя способами:

- способом решения системы линейных алгебраических уравнений;
- способом составления таблицы разделенных разностей;
- способом, основанным на использовании формулы

$$f(x_0, x_1, \dots, x_n) = \sum_{i=0}^m f(x_i) \frac{1}{\prod_{j=0, j \neq i}^m (x_i - x_j)}, \quad m \geq 1 \quad (1)$$

Входные данные:

- отрезок $[a, b]$;
- функция $F(x)$, по которой производится расчет значений в узлах интерполяции (значения которой приближаются интерполяционным многочленом)

$$F(x) = \frac{1-x}{1+x^2}, \quad x \in [-1; 2]; \quad (2)$$

- произвольная точка x^* отрезка $[a, b]$, для которой считается значение интерполяционного многочлена.

В программе предусмотреть вычисление набора узлов интерполяции x_0, x_1, \dots, x_n (считать равноотстоящими друг от друга на отрезке $[a, b]$).

Выходные данные:

- значение многочлена в точке x^* (приближенное значение функции, $P_n(x^*)$).

2. Составить и отладить программу построения графиков исходной функции $F(x)$ и ее интерполяционного многочлена $P_n(x)$, построенного по равноотстоящим узлам интерполяции на отрезке $[a, b]$ тремя указанными способами.

3. Провести численный эксперимент для выяснению вопроса о сходимости графика интерполяционного многочлена к графику исходной функции и влиянии степени интерполяционного полинома на точность интерполяции. Выяснить, какой из способов расчета коэффициентов многочлена Ньютона более устойчив к влиянию ошибок округления.

Указания к выполнению лабораторной работы

Интерполяцией называется такой вид точечной аппроксимации, когда аппроксимирующая функция представляет собой алгебраический многочлен (полином) $\varphi(x)$ степени n , который в $n+1$ точке (узле) x_i ($i=0, 1, \dots, n$), заданных на отрезке $[a, b]$, совпадает со значением аппроксимируемой функции $f(x)$ в этих узлах. Такой многочлен называется интерполяционным многочленом.

Представление интерполяционного многочлена $P_n(x)$ в виде разложения по базису в пространстве многочленов степени не выше n , а именно по базису, состоящему из многочленов

$$\begin{aligned}\psi_0(x) &= 1, \quad \psi_1(x) = (x - x_0), \quad \psi_2(x) = (x - x_0)(x - x_1), \dots, \\ \psi_n(x) &= (x - x_0)(x - x_1)\cdots(x - x_{n-1}), \quad (3)\end{aligned}$$

дает многочлен Ньютона вида:

$$\begin{aligned}P_n(x) &= d_0 + d_0(x - x_0) + d_0(x - x_0)(x - x_1) + \dots + \\ &+ d_0(x - x_0)(x - x_1)\dots(x - x_{m-1}) + \dots \quad (4) \\ &+ d_0(x - x_0)(x - x_1)\dots(x - x_{n-1})\end{aligned}$$

Воспользовавшись условиями интерполяционности, а именно равенством значений функции $f(x)$ и полинома $P_n(x)$ в узлах интерполяции, можно получить систему линейных алгебраических уравнений с верхнетреугольной матрицей:

$$\begin{cases} d_0 = f(x_0), \\ d_0 + d_1(x_1 - x_0) = f(x_1), \\ d_0 + d_1(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) = f(x_2), \\ d_0 + d_1(x_3 - x_0) + d_2(x_3 - x_0)(x_3 - x_1) + d_3(x_3 - x_0)(x_3 - x_1)(x_3 - x_2) = f(x_3), \\ \dots \end{cases} \quad (5)$$

причем данная система имеет $(n+1)$ уравнение и $(n+1)$ неизвестное, то есть разрешается однозначно следующим образом:

$$\begin{cases} d_0 = f(x_0), \\ d_1 = \frac{f(x_1) - d_0}{x_1 - x_0}, \\ d_2 = \frac{f(x_2) - d_0 - d_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}, \\ d_3 = \frac{f(x_3) - d_0 + d_1(x_3 - x_0) + d_2(x_3 - x_0)(x_3 - x_1)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}, \\ \dots \end{cases} \quad (6)$$

что позволяет легко запрограммировать вычисление коэффициентов d_i путем реализации итерационного индукционного алгоритма. Данный алгоритм на основе данных о младших коэффициентах d_i позволяет рассчитывать коэффициенты последующих порядков. Такой подход в данной работе носит название **метода вычисления многочлена Ньютона путем решения системы линейных алгебраических уравнений**.

Кроме отмеченного, на практике часто используют другой подход, который основан **на составлении таблицы разделенных разностей от функции $f(x)$** . В этом случае многочлен Ньютона может быть записан в виде:

$$\begin{aligned} P_n(x) = & f(x_0) + f(x_0, x_1)(x-x_0) + f(x_0, x_1, x_2)(x-x_0)(x-x_1) + \dots + \\ & + f(x_0, x_1, \dots, x_m)(x-x_0)(x-x_1)\dots(x-x_{m-1}) + \dots \\ & + f(x_0, x_1, \dots, x_n)(x-x_0)(x-x_1)\dots(x-x_{n-1}) \end{aligned} \quad (7)$$

Здесь числовой коэффициент при m -ой ($m = 0, 1, \dots, n$) базисной функции

$$(x-x_0)(x-x_1)\dots(x-x_{m-1})$$

обозначается символом

$$f(x_0, x_1, \dots, x_m), \quad (8)$$

который называется **разделенной разностью** m -го порядка в узле x_0 .

Достоинство разделенных разностей заключается в том, что они определяются индуктивно, т.е. разделенная разность m -го порядка выражается через разделенные разности предшествующего $(m - 1)$ -го порядка:

- разделенные разности нулевого порядка:

$$f(x_0), f(x_1), \dots, f(x_n); \quad (9)$$

- разделенные разности первого порядка:

$$f(x_0, x_1) = \frac{f(x_0) - f(x_1)}{x_0 - x_1}, \quad f(x_1, x_2) = \frac{f(x_1) - f(x_2)}{x_1 - x_2}, \quad \dots, \quad f(x_{n-1}, x_n) = \frac{f(x_{n-1}) - f(x_n)}{x_{n-1} - x_n}; \quad (10)$$

- разделенные разности второго порядка:

$$f(x_0, x_1, x_2) = \frac{f(x_0, x_1) - f(x_1, x_2)}{x_0 - x_2}, \quad f(x_1, x_2, x_3) = \frac{f(x_1, x_2) - f(x_2, x_3)}{x_1 - x_3}, \dots,$$

$$f(x_{n-2}, x_{n-1}, x_n) = \frac{f(x_{n-2}, x_{n-1}) - f(x_{n-1}, x_n)}{x_{n-2} - x_n}; \quad (11)$$

...

- разделенная разность n-го порядка:

$$f(x_0, x_1, \dots, x_{n-1}, x_n) = \frac{f(x_0, x_1, \dots, x_{n-1}) - f(x_1, \dots, x_{n-1}, x_n)}{x_0 - x_n} \quad (12)$$

Коэффициенты d_i являются разделенными разностями из первой строки треугольной таблицы вида:

Таблица 1 – Элементы таблицы разделенных разностей

x_0	$f(x_0)$	$f(x_0, x_1)$	$f(x_0, x_1, x_2)$...	$f(x_0, \dots, x_{n-1})$	$f(x_0, \dots, x_n)$
x_1	$f(x_1)$	$f(x_1, x_2)$	$f(x_1, x_2, x_3)$...	$f(x_1, \dots, x_n)$	-
x_2	$f(x_2)$	$f(x_2, x_3)$	$f(x_2, x_3, x_4)$...	-	-
x_3	$f(x_3)$	$f(x_3, x_4)$	$f(x_3, x_4, x_5)$...	-	-
...
x_{n-2}	$f(x_{n-2})$	$f(x_{n-2}, x_{n-1})$	$f(x_{n-2}, x_{n-1}, x_n)$...	-	-
x_{n-1}	$f(x_{n-1})$	$f(x_{n-1}, x_n)$	-	...	-	-
x_n	$f(x_n)$	-	-	...	-	-

То есть $d_0 = f(x_0)$, $d_1 = f(x_0, x_1)$, $d_2 = f(x_0, x_1, x_2)$ и т.д.

Все это позволяют отказаться от необходимости решать систему уравнений (5), а вместо этого на основе построения таблицы разделенных разностей реализовать итеративный алгоритм с индуктивным определением коэффициентов полинома Ньютона как разделенных разностей на основе разделенных разностей меньших порядков.

Существует и *третий способ нахождения коэффициентов полинома Ньютона*, который позволяет вычислять элементы первой верхней строки треугольной таблицы разделенных разностей, не вычисляя элементы в нижних строках таблицы. Данный способ основан на свойстве разделенных разностей соответствовать формуле вида:

$$f(x_0, x_1, \dots, x_n) = \sum_{i=0}^m f(x_i) \frac{1}{\prod_{j=0, j \neq i}^m (x_i - x_j)}, \quad m \geq 1$$

Ход выполнения работы

Выполнение лабораторной работы велось параллельно на языке технического моделирования MatLab 2017 и на языке C# в среде IDE Microsoft Visual Studio 2017. Для выполнения поставленных задач было разработано несколько программных модулей (см. Приложение).

MatLab использовался для быстрого прототипирования и отладки, вывода графиков и поведения исследований полиномов различной степени. После этого отлаженные алгоритмы реализовывались средствами языка C#, а затем осуществлялся анализ их эффективности с точки зрения традиционных средств программирования.

На первом этапе в среде MatLab реализованы три описанных выше алгоритма нахождения коэффициентов полинома Ньютона; алгоритмы оформлены отдельными программными модулями в следующих файлах:

- *coefPolyNewton1.m* – модуль, реализующий вычисление коэффициентов способом решения системы линейных алгебраических уравнений;
- *coefPolyNewton2.m* – модуль, реализующий вычисление коэффициентов способом составления таблицы разделенных разностей;
- *coefPolyNewton3.m* – модуль, реализующий вычисление коэффициентов способом, основанным на использовании формулы (1).

Для вычисления значений функции $F(x)$ для узла x использовался реализованный ранее в первой лабораторной работе модуль *f.m*.

Модули *coefPolyNewton* получают на вход в качестве параметров массивы координат узлов, например, следующего вида:

$$X = [1; 2; 3];$$

$$Y = [-2; 1; 6].$$

Вызов модулей осуществляется командами:

$$\text{coefPolyNewton1}(X, Y)$$

$$\text{coefPolyNewton2}(X, Y)$$

$$\text{coefPolyNewton3}(X, Y)$$

В свою очередь, чтобы избежать повторных расчетов значений узлов и однозначно определить интерфейс, обеспечивающий доступ ко всем трем модулям, реализован промежуточный объединяющий модуль *coefPolyNewtonBase.m* с командой для его вызова следующего вида:

$$[D, X] = \text{coefPolyNewtonBase}(a, b, n, \text{number})$$

где передаваемые параметры a , b , n – соответственно границы отрезка $[a, b]$ и степень интерполяционного полинома, а $number$ – номер способа, которым рассчитываются коэффициенты полинома Ньютона (1, 2 или 3).

Модуль возвращает матрицу коэффициентов D и матрицу узловых точек X , чтобы затем на их основе можно было рассчитать значение полинома Ньютона в любой точке по формуле через разности $(x-x_0)$ и т.д.

На втором этапе в среде MatLab разработан модуль *pointNewton.m*, реализующий вычисление значения интерполяционного многочлена в точке x , а также дополнительные модули для формирования графиков.

Первый модуль (*pointNewton.m*) получает в качестве передаваемых параметров матрицу коэффициентов D , матрицу узловых точек X и значение x , для которого надо посчитать $P(x)$:

$$Px = pointNewton(D, X, xx)$$

Построение графиков выполняют следующие модули:

- *plotNewton.m* - выводит одиночный график полинома, совмещенный с графиком функции и узловыми точками;
- *plotNewtonFull.m* - выводит совмещенные графики полиномов, коэффициенты которых посчитаны всеми тремя способами (для сравнения).

Вызываются эти модули следующим образом:

$$plotNewton(a, b, n, number)$$

$$plotNewtonFull(a, b, n)$$

где a и b - границы отрезка, n - степень полинома, $number$ - номер способа расчета коэффициентов полинома Ньютона (см. выше *coefPolyNewtonBase*).

В результате работы модуля *plotNewtonFull.m* формируется новое окно с совмещенными графиками (пример на рис. 1).

При построении графиков равноотстоящие узлы интерполяции дополняются промежуточными точками, поскольку узловых точек недостаточно для демонстрации расхождения графиков (в узловых точках они совпадают).

На третьем этапе в среде Visual Studio 2017 было реализовано приложение на языке C# на основе WinForms и свободной библиотеки ZedGraph с аналогичным функционалом. Основное окно программы с отрисованным графиком приведено на рис. 2.

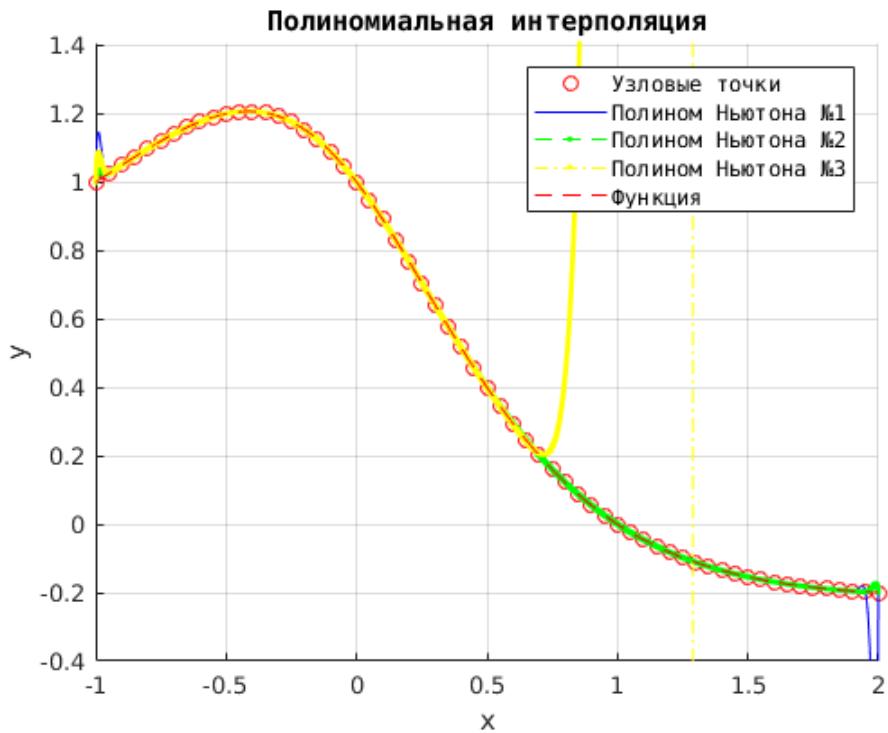


Рисунок 1 – (пример) Результат работы модуля *plotNewtonFull.m*
Графики интерполяционного многочлена Ньютона, посчитанного тремя разными способами (с нанесением узлов интерполяции и графика функции $F(x)$)

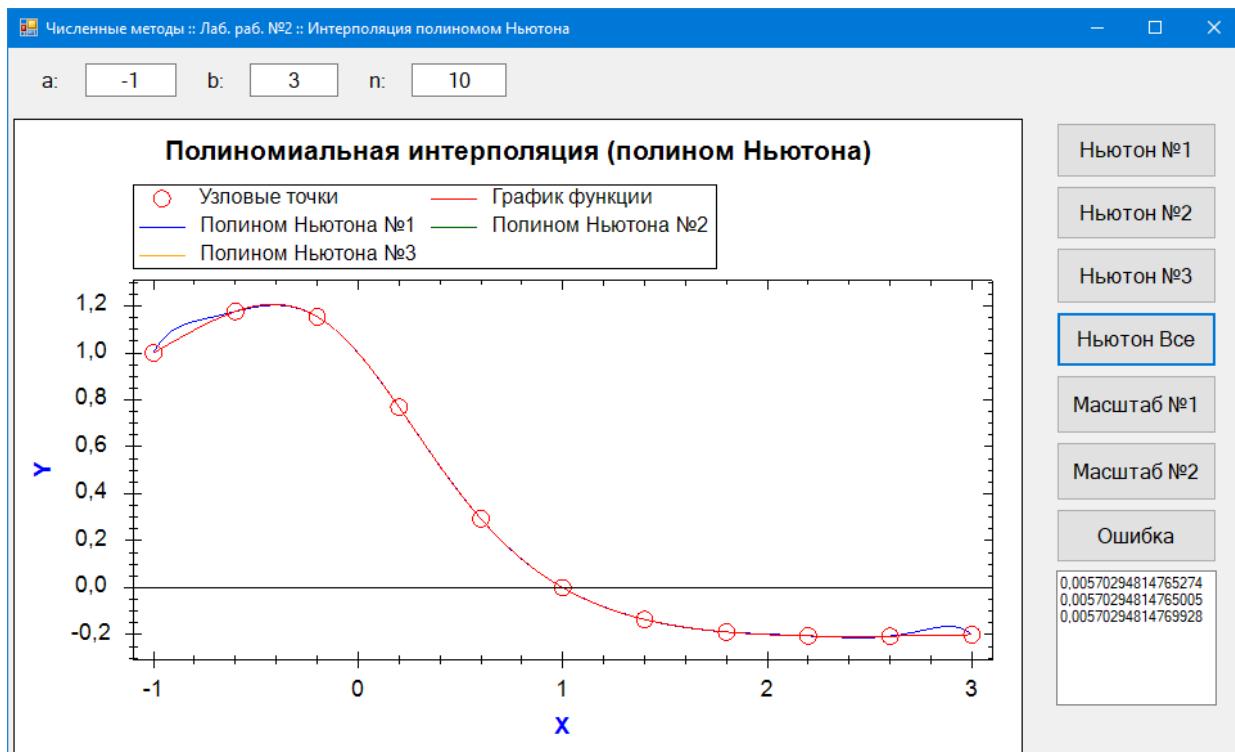


Рисунок 2 – Окно программы на C#: сравнение графиков интерполяционного многочлена Ньютона и функции $F(x)$

На четвертом этапе для выяснения вопроса о сходимости графика интерполяционного многочлена к графику исходной функции, влиянии степени интерполяционного полинома и способа расчета коэффициентов многочлена на точность интерполяции выполнен численный эксперимент. В качестве основного инструмента использовались разработанные ранее модули MatLab и программа на языке C#.

Изменяя степень полинома Лагранжа от 2 до 100, были получены сравнительные графики (рис. 5 - 10). Их анализ показал, прежде всего, что **интерполяционный многочлен Ньютона для нашей функции не обладает сходимостью**.

Все три метода расчета коэффициентов полинома Ньютона проявляют **схожие свойства**: при небольших значениях степени полинома с увеличением степени наблюдается постепенное улучшение приближения к функции. Затем этот рост сходимости замедляется, и, начиная с некоторого номера, увеличение степени полинома уже приводит к все большему проявлению расходимости.

Следует заметить, что в силу различия алгоритмов расчета максимальная степень полинома Ньютона, при котором еще наблюдается рост сходимости, у данных трех способов различен. Очевидно, что это связано с ошибками округления, которые максимальны в алгоритмах с большим количеством промежуточных вычислений. Если обратится к рис. 11, на котором показано сравнение поведения ошибки интерполяции от степени полинома для трех рассматриваемых способов расчета коэффициентов, то можно заметить следующее. До отметки степени полинома $N \approx 53$ характер роста сходимости всех трех методов практически полностью совпадает. Однако затем из-за различий в алгоритмах расчета влияние ошибки округления начинает существенное сказываться на сходимости, причем для каждого из способов характер зависимости приобретает отличительные черты, достигая максимального уровня точности при разных значениях N .

Для метода, основанного на решении системы линейных уравнений, характерно более быстрое достижение точности приближения (при $N = 59$) с последующим быстрым накоплением влияния ошибки округления при более высоких значениях N и резким скачком потери точности приближения.

Метод, основанный на применении формулы (1), для исследуемой функции оказался более устойчив к влиянию ошибок округления. Для него характерно продолжение увеличения точности приближения до отметки $N = 62$ с последующим более плавным накоплением влияния ошибок округления, чем в предыдущем случае.

Самым устойчивым оказался метод расчета коэффициентов полинома Ньютона на основе таблиц разделенных разностей. Он показал продолжение роста точности приближения вплоть до отметки $N = 68$ с последующим колебательным ростом влияния ошибки вдоль линии тренда.

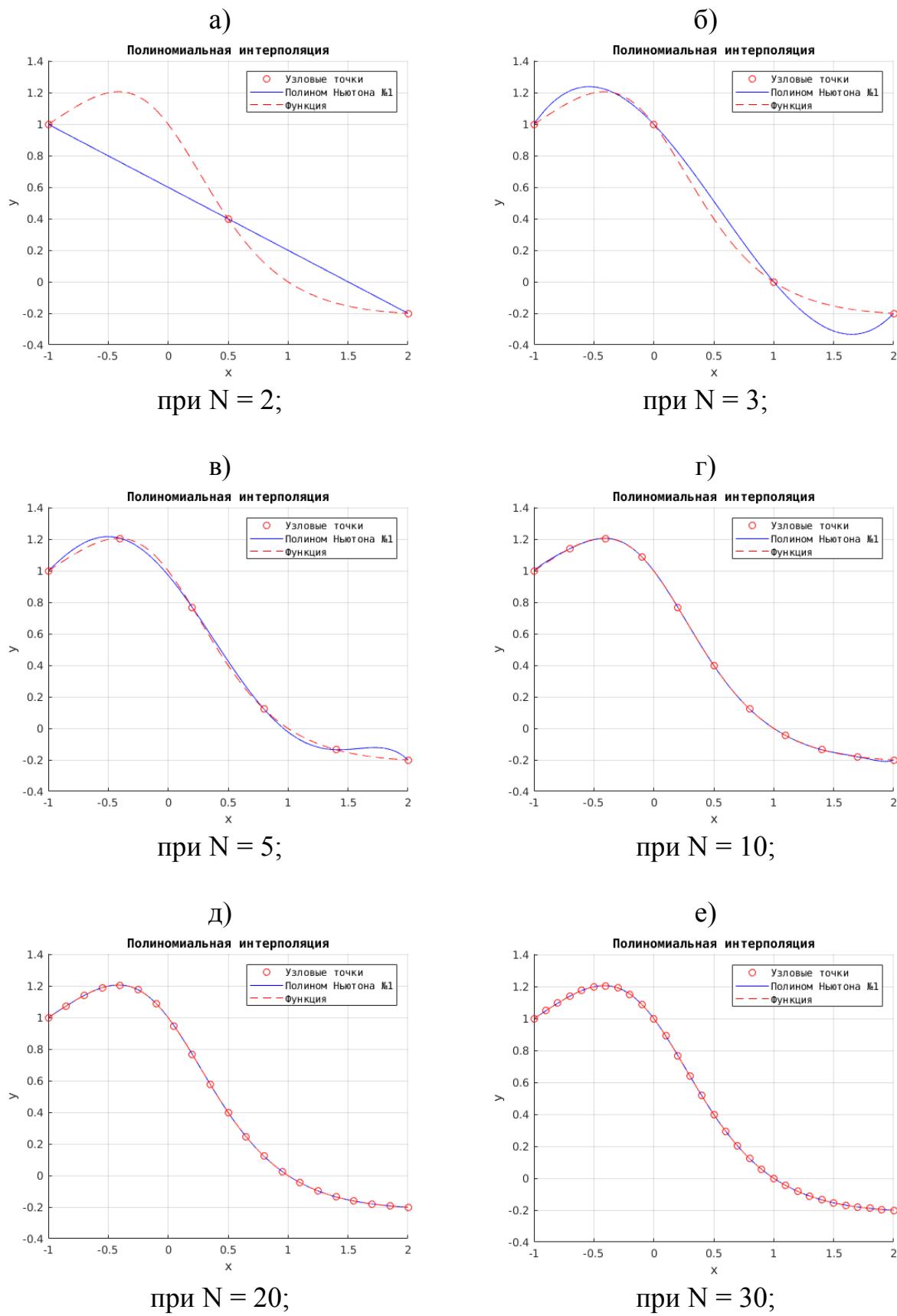


Рисунок 5 – Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены способом решения системы линейных алгебраических уравнений (способ №1)

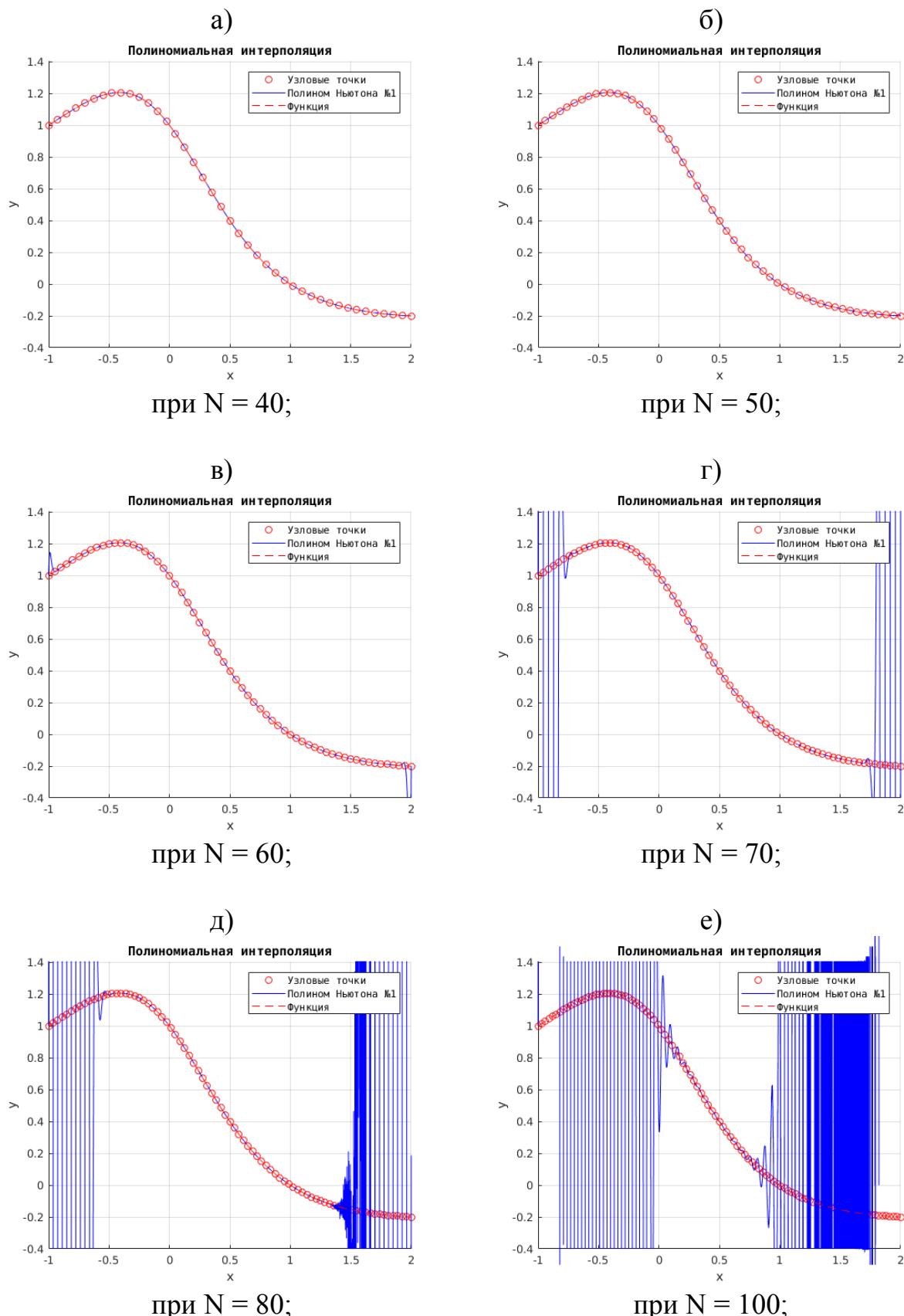


Рисунок 6 – (продолжение) Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены способом решения системы линейных алгебраических уравнений (способ №1)

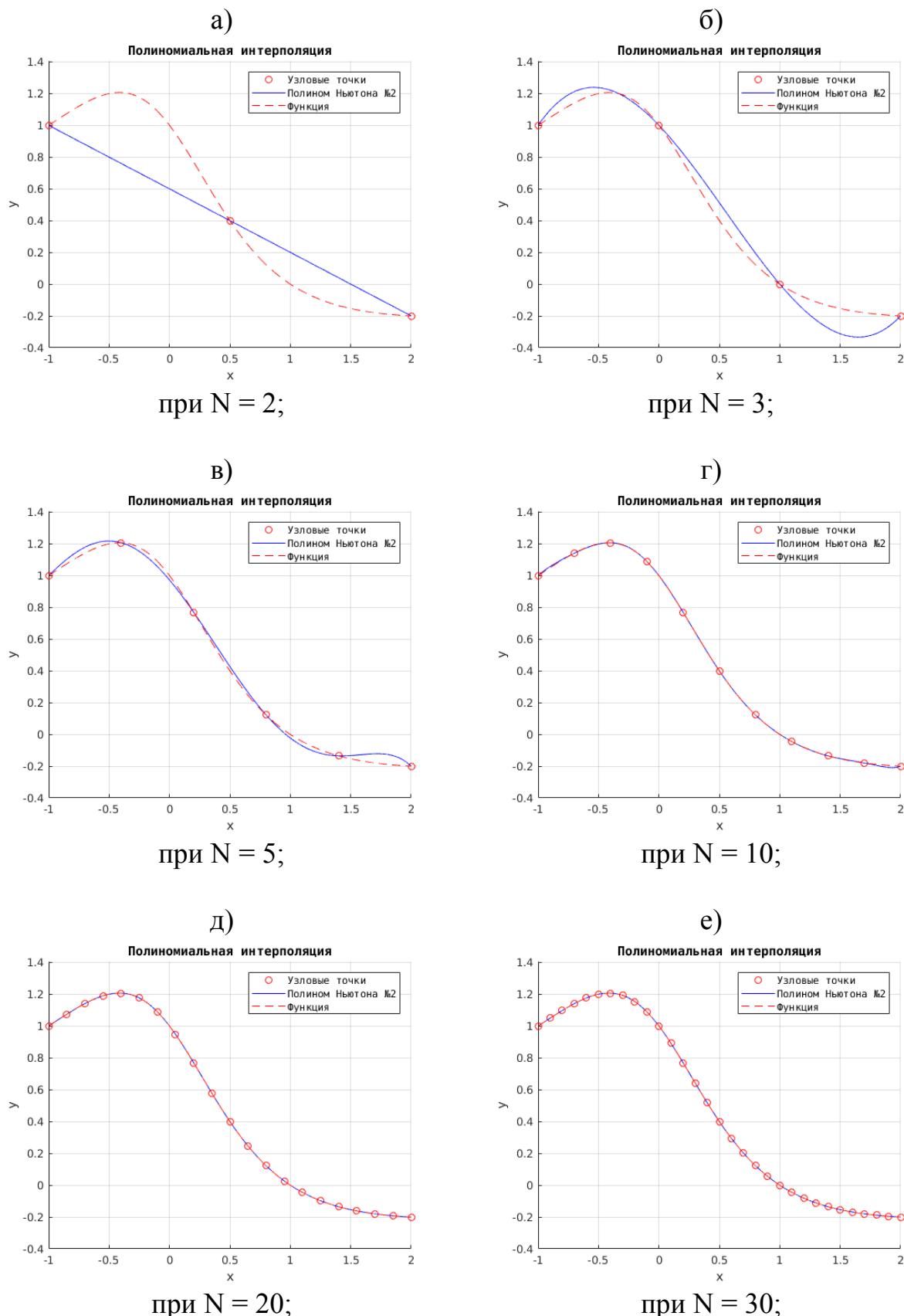
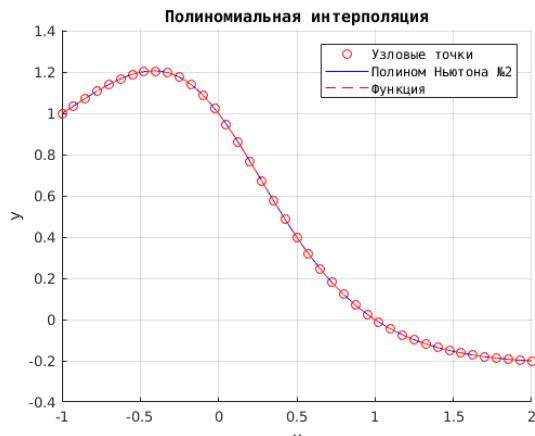
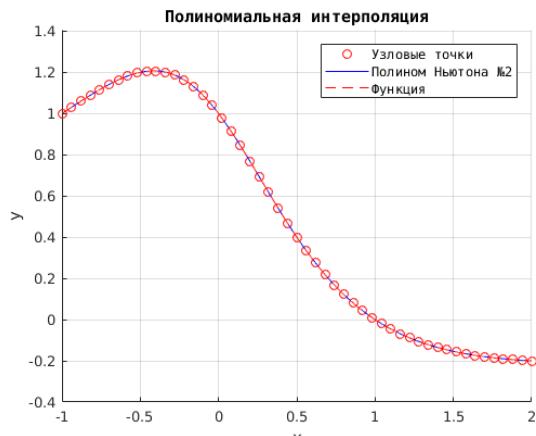


Рисунок 7 – Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены способом составления таблицы разделенных разностей (способ №2)

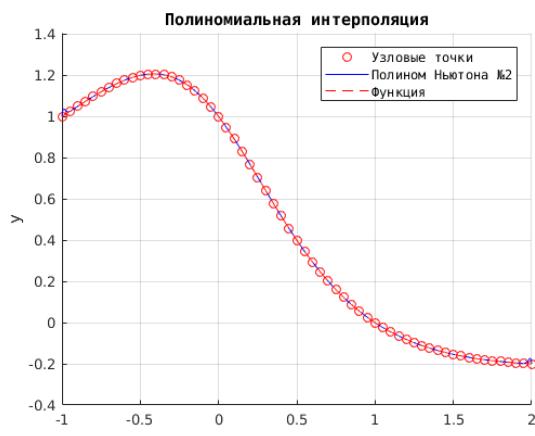
а)

при $N = 40$;

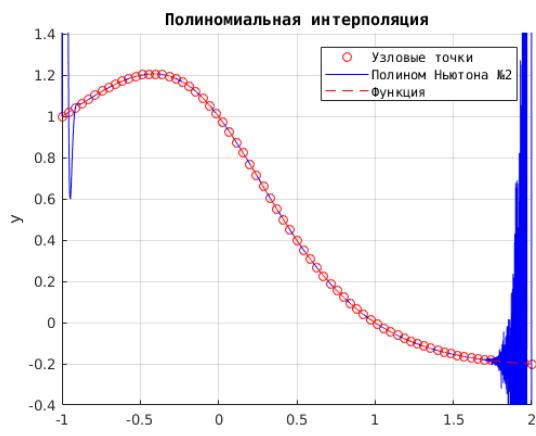
б)

при $N = 50$;

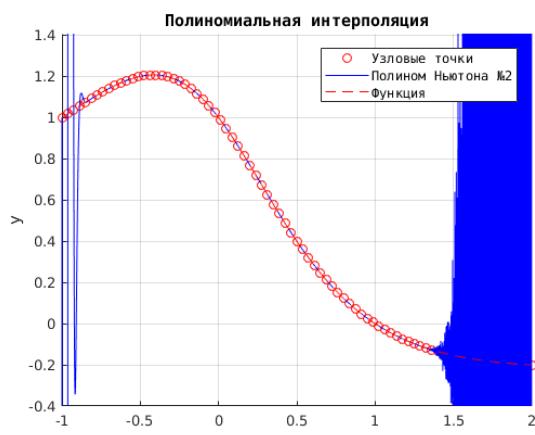
в)

при $N = 60$;

г)

при $N = 70$;

д)

при $N = 80$;

е)

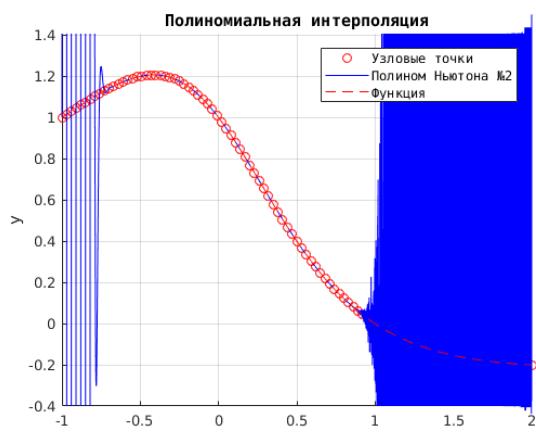
при $N = 100$;

Рисунок 8 – (продолжение) Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены способом составления таблицы разделенных разностей (способ №2)

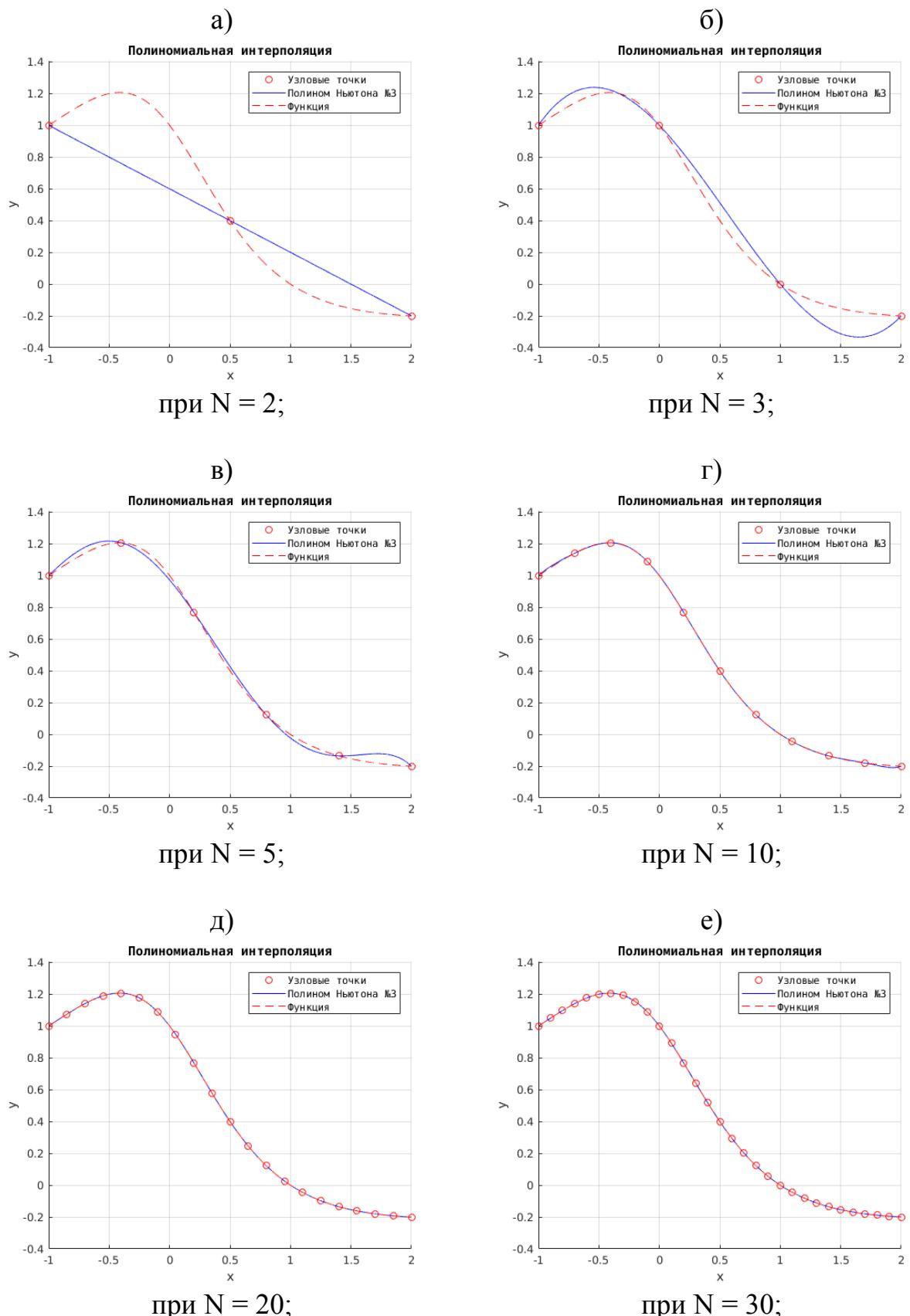


Рисунок 9 – Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены по формуле (1) (способ №3)

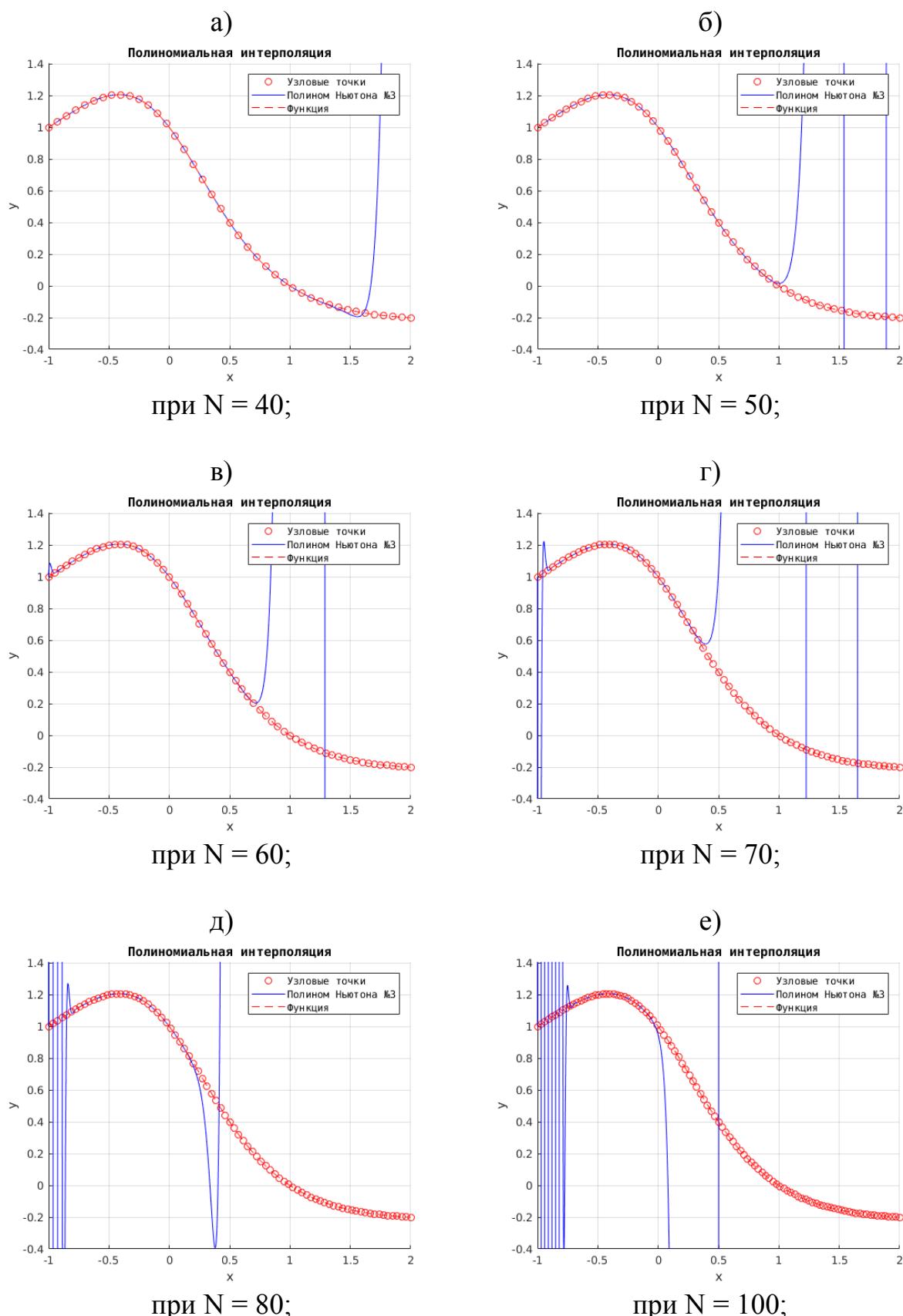


Рисунок 10 – (продолжение) Сравнение точности приближения при различных значениях степени полинома Ньютона, коэффициенты вычислены по формуле (1) (способ №3)

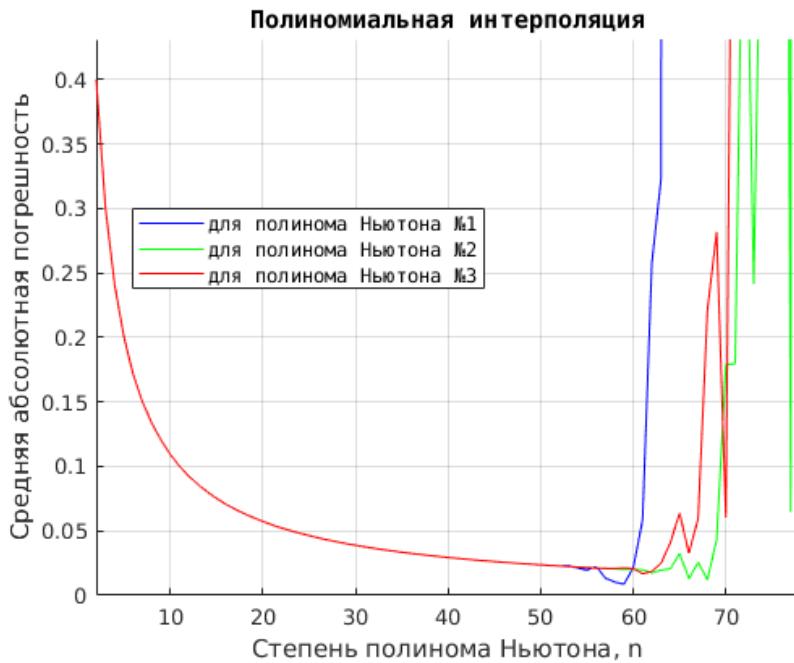


Рисунок 11 – Зависимость средней ошибки интерполяции от степени полинома Ньютона: из-за различий в алгоритмах расчета влияние ошибки округления начинает сказываться при разных значениях N

Следует заметить, что для последних двух методов характерно колебательное поведение вблизи и после прохождения критической отметки своего максимального значения степени N , при котором достигается максимальная сходимость. С целью изучения этого поведения были поставлены дополнительные численные эксперименты с последующим сравнением графиков полиномов.

Установлено, что изучаемые три способа расчета отличаются и уровнем осцилляций, и скоростью их проявления. Так, например, начиная с отметки $N = 35$ (рис. 12) для метода расчета коэффициентов №3 (с использованием ф. (1)) начинается проявляться расходимость на правом конце отрезка $[a, b]$, которые увеличиваются с ростом N и выливаются в характерные осцилляции (рис. 13). При этом другие два метода остаются устойчивыми к осцилляциям вплоть до отметки $N = 50$. Затем наблюдается появление осцилляций для метода №1 (на основе решения системы линейных уравнений), характерных для обоих концов отрезка (рис. 14). И только на отметке $N = 60$ начинают проявляться первые заметные осцилляции для метода №2 (вычисление таблицы разделенных разностей) (рис. 15), которые уже достаточно заметны к $N = 65$ (рис. 16). В то же время для методов №1 и №2 эти осцилляции достигают весьма внушительного уровня (рис. 16, 17).

Отдельно следует отметить метод №3, для которого при росте N выше 50 сходимость практически полностью теряется на правой части отрезка $[a, b]$. Очевидно, что этот способ при высоких значениях N неспособен обеспечить достаточный уровень точности, а сама точность интерполяции сильно зависит от расположения интерполируемой точки внутри отрезка интерполяции.

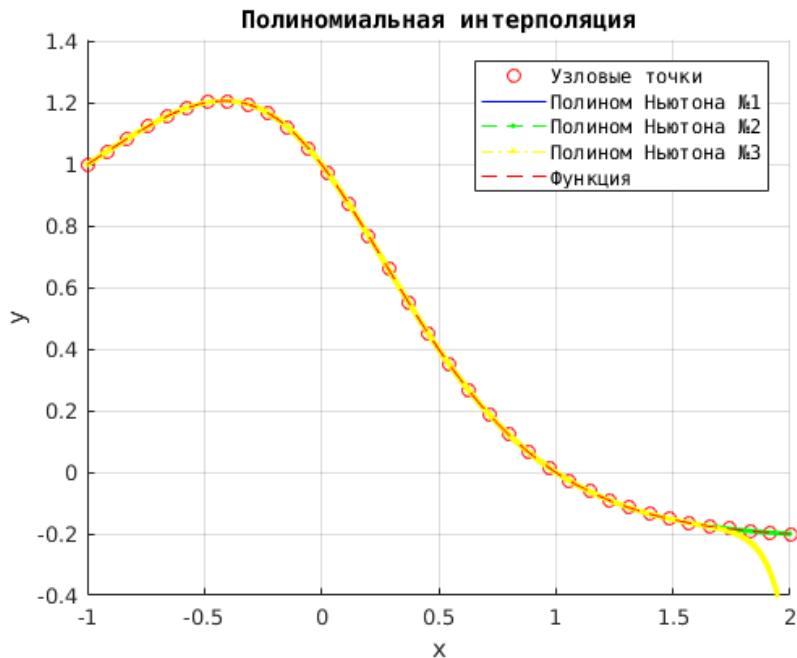


Рисунок 12 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов: при $N = 35$ заметна расходимость на правом конце отрезка $[a, b]$ – начало проявления осцилляций для метода расчета №3

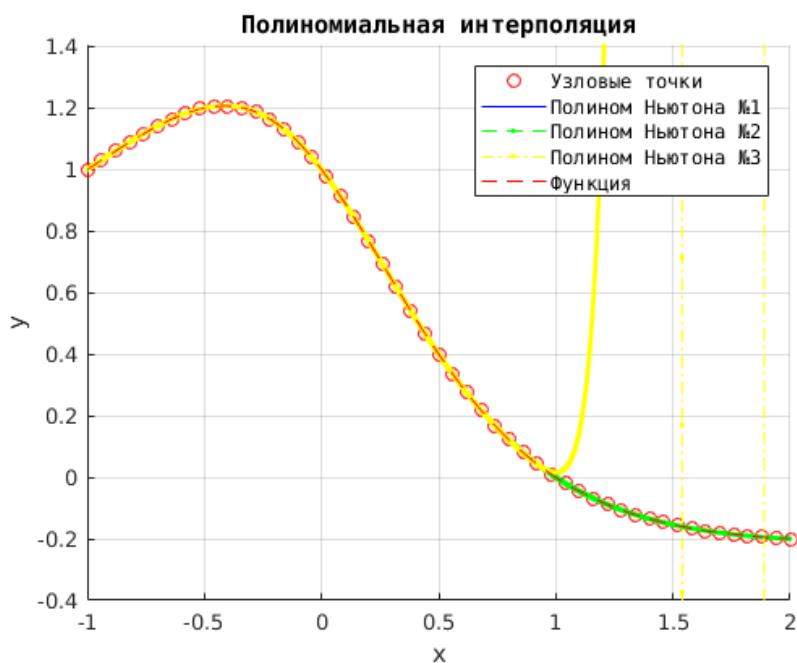


Рисунок 13 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов: $N = 50$, заметно существенное увеличение осцилляций для метода расчета №3 на правом конце отрезка $[a, b]$

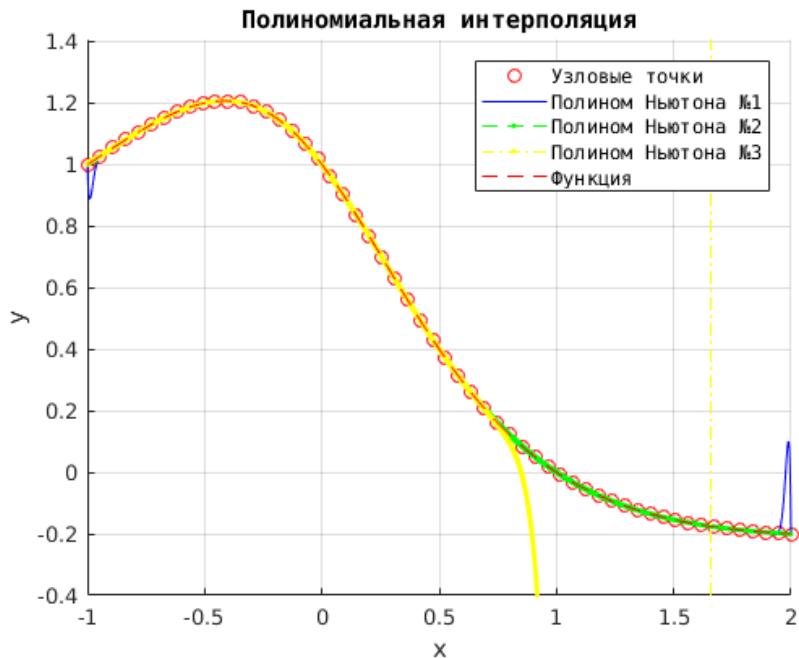


Рисунок 14 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов: $N = 55$, появление осцилляций для метода расчета №1 на обоих концах отрезка $[a, b]$

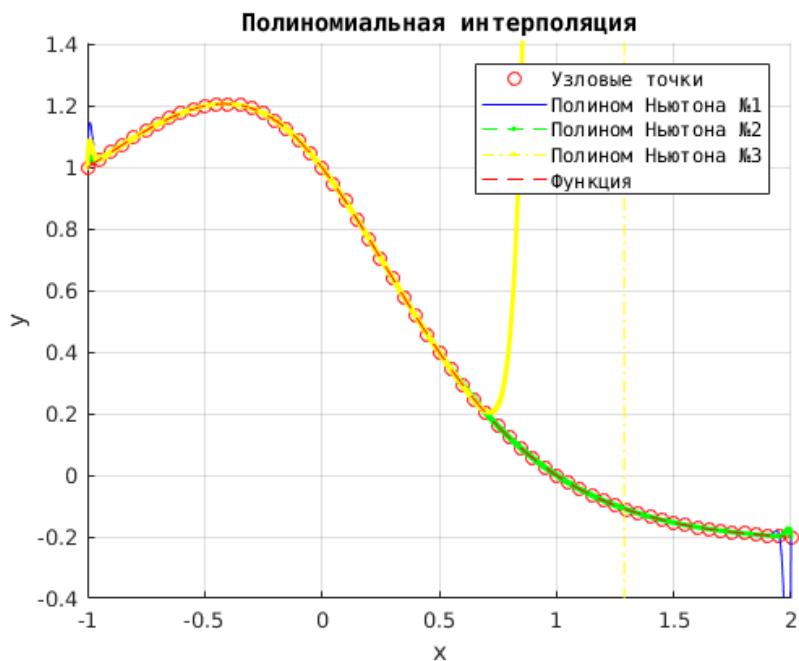


Рисунок 15 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов: $N = 60$, появление небольших осцилляций для метода расчета №2 на обоих концах отрезка $[a, b]$

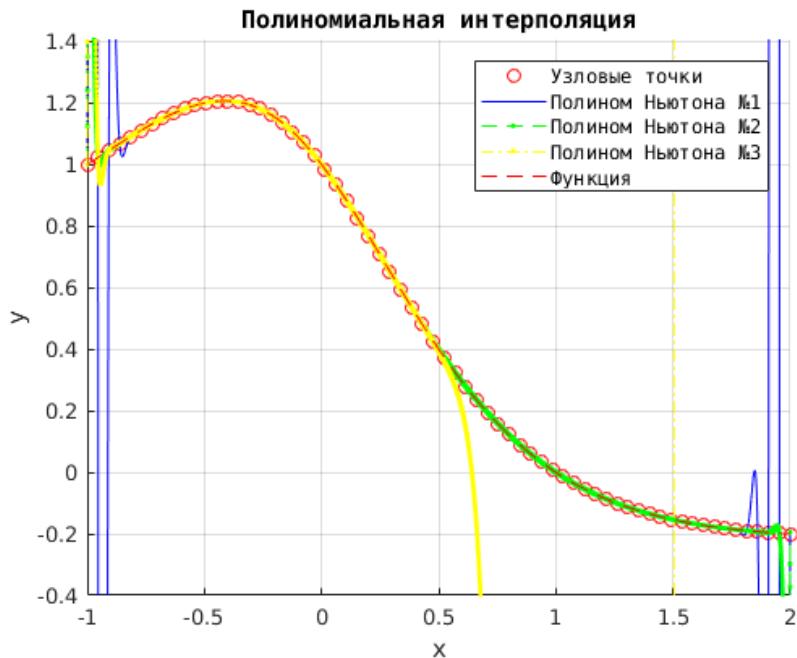


Рисунок 16 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов ($N = 65$)

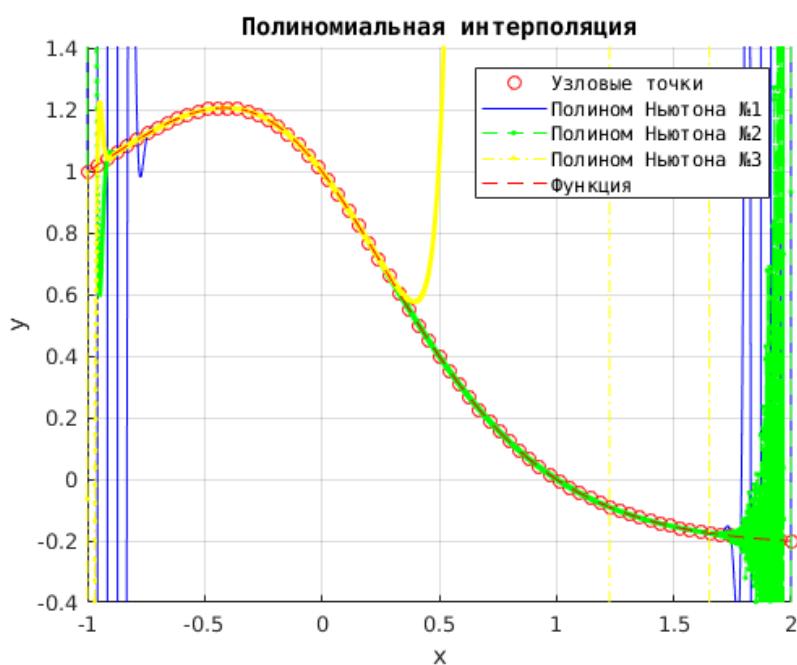


Рисунок 17 – Сравнение точности приближения полинома Ньютона для трех методов расчета коэффициентов ($N = 70$)

Выводы по работе

1. В ходе лабораторной работе рассмотрен способ интерполяции функций алгебраическими многочленами методом Ньютона, разработаны функциональные модули для MatLab и программа на C#, с помощью которых можно исследовать интерполяцию функций многочленами.
2. В результате численного эксперимента установлено, что интерполяционный многочлен Ньютона для заданной функции не обладает сходимостью: при небольших значениях степени полинома с ее увеличением наблюдается постепенное улучшение приближения полинома к функции. Затем этот рост сходимости замедляется, и, начиная с некоторого номера, увеличение степени полинома уже приводит к все большему проявлению расходимости.
3. Различия алгоритмов расчета являются причиной несовпадения максимальной степени полинома Ньютона, при котором еще наблюдается рост сходимости, у данных трех способов. Это связано с ошибками округления, которые максимальны в алгоритмах с большим количеством промежуточных вычислений.
4. Уровень ошибок округления зависит от выбранного способа расчета значений интерполяционного многочлена. Найлучшим в плане устойчивости к ошибкам округления является метод расчета коэффициентов полинома Ньютона на основе таблиц разделенных разностей.
5. Метод расчета коэффициентов полинома Ньютона на основе решения системы линейных уравнений позволяет добиться большей точности при меньших значениях степени полинома, но является самым неустойчивым к ошибкам округления при высоких степенях полинома.
6. При увеличении степени многочлена выше критического значения N , при котором еще сдерживается рост ошибок округления, существенный вклад в поведение полинома начинают оказывать регулярные осцилляции. Наиболее устойчивым к их влиянию является метод расчета на основе вычисления таблицы разделенных разностей.

Список литературы

1. Гудович Н.Н. Избранные вопросы курса численных методов. Выпуск 2. Многочлен Ньютона: учебно-методическое пособие для вузов. Воронеж: Издательско-полиграфический центр Воронежского государственного университета, 2002 г. – 28 с.
2. Кетков, Ю.Л. и др. Matlab 7: программирование, численные методы / Ю.Л. Кетков, А.Ю. Кетков, М.М. Шульц, - СпБ.: БХВ-Петербург, 2005 г. – 752 с.

Приложение (листинг)

Код программы на C#, файл Form1.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using ZedGraph;
using System.Collections.Generic;
using System.Drawing.Drawing2D;
namespace Lab02
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent(); // Инициализация компонентов
            GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования
            pane.XAxis.Title.Text = "X"; // Изменяем текст надписи по оси X
            pane.XAxis.Title.FontSpec.FontColor = Color.Blue; // и цвет на синий
            pane.YAxis.Title.Text = "Y"; // Изменим текст по оси Y
            pane.YAxis.Title.FontSpec.FontColor = Color.Blue; // и цвет на синий
            pane.Title.Text = "Полиномиальная интерполяция (полином Ньютона)"; // Изменяем заголовок холста
        }
        // [&] Исходная функция, которую аппроксимируем многочленом (можно написать любую)
        private double f(double x)
        {
            return (1 - x) / (1 + x * x);
        }
        // [&] Функция вычисления ошибки:
        double delta(List<double> Y1, List<double> Y2)
        {

```

```

double sum = 0.0;

int n = Y1.Count;

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

{

    sum = sum + Math.Abs(Y1[i] -Y2[i]);

}

return sum / n;

}

// [c] Только для отладки:

// Метод вывода в консоль значений коэффициентов полинома Ньютона

void print(List<double> D)

{

foreach(double d in D)

{

System.Console.Write(d);

System.Console.Write("; ");

}

System.Console.WriteLine();

}

// [1] % Метод для вычисления коэффициентов полинома Ньютона путем решения n+1 линейных

алгебраических уравнений

List<double> coefPolyNewton1(List<double> X, List<double> Y)

{

List<double> D = new List<double>(); // Создаем список коэффициентов

int n = X.Count; // Количество узловых точек

for (int i = 0; i < n; i++) // Внешний цикл по D "слева"

{

double chislitel = Y[i];

for (int j = 0; j < i; j++) // Вложенный цикл по D "справа"

{

double pp = D[j];

for (int k = 0; k < j; k++) // Внутренний цикл по X

{

```

```

pp = pp * (X[i] - X[k]);
}

chislitel = chislitel - pp;

}

// Посчитали числитель, находим знаменатель:

double znamenatel = 1.0;

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

{
znamenatel = znamenatel * (X[i] - X[j]);

}

// Делим числитель на знаменатель и сохраняем полученный результат как коэффициент в список
коэффициентов D:

D.Add(chislitel / znamenatel);

}

// print(D);

return D;
}

// [2] Метод для вычисления коэффициентов полинома Ньютона путем вычисления таблицы разделенных
разностей RR

List<double> coefPolyNewton2(List<double> X, List<double> Y)

{
List<double> D = new List<double>(); // Создаем список коэффициентов

int n = X.Count; // Количество узловых точек

// Создаем и инициализируем двумерный массив для таблицы разделенных разностей

double[,] RR = new double[n, n];

// Сначала заполняем нулевой столбец таблицы разделенных разностей RR:

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

{
RR[i, 0] = Y[i];

}

// Затем на основе нулевого заполненного столбца заполняем оставшиеся:

for (int j = 1; j < n; j++) // Внешний цикл по столбцам RR

{

```

```

int m = n - j; // Каждый переход на столбец вправо сокращает количество разностей
for (int i = 0; i < m; i++) // Внутренний цикл по строкам RR
{
    RR[i, j] = (RR[i, j - 1] - RR[i + 1, j - 1]) / (X[i] - X[i + j]);
}
}

// И в конце переписываем из нулевой строки RR в матрицу коэффициентов D:
for (int j = 0; j < n; j++)
{
    D.Add(RR[0, j]);
}

// print(D);

return D;
}

// [3] Метод для вычисления коэффициентов полинома Ньютона путем вычисления суммы произведений
// без вычисления таблицы разделенных разностей
List<double> coefPolyNewton3(List<double> X, List<double> Y)
{
    List<double> D = new List<double>(); // Создаем список коэффициентов
    int n = X.Count; // Количество узловых точек
    for (int m = 0; m < n; m++) // Внешний цикл по D(m)
    {
        double sum = 0.0; // Для каждого D(m) в первую очередь обнуляем сумму,
        // а затем считаем произведение и суммируем с суммой:
        for (int i = 0; i <= m; i++) // Вложенный цикл по Сумме
        {
            double pp = Y[i]; // Числитель одного из слагаемых
            double dd = 1.0; // Начальное значение знаменателя
            // и вычисляем знаменатель:
            for (int j = 0; j <= m; j++) // Внутренний цикл по Произведению
            {
                if (i != j) // Должны быть не равны, иначе знаменатель обнулится

```

```

{
dd = dd * (X[i] - X[j]);
}
}

sum = sum + pp / dd; // Добавляем к сумме
}

D.Add(sum);
}

//print(D);

return D;
}

// [*] Метод рассчета списка координат X узловых точек для интервала [a, b] и степени многочлена n:

List<double> createXList(double a, double b, int n)
{
List<double> X = new List<double>(); // Создаем список координат x для узловых точек
double delta = Math.Abs(a - b) / n; // Находим интервал разбиения
X.Add(a); // Начало отрезка
for (int i = 1; i < n; i++)
{
// Заполняем вектор X
X.Add(X[i - 1] + delta);
}
X.Add(b); // конец отрезка
return X;
}

// [*] Метод рассчета списка координат Y узловых точек для интервала [a, b] и степени многочлена n:

List<double> createYList(List<double> X)
{
List<double> Y = new List<double>(); // Создаем список координат x для узловых точек
for (int i = 0; i < X.Count; i++)
{
// Заполняем вектор Y
Y.Add(f(X[i]));
}
}

```

```

}

return Y;
}

// [!] Основной метод рассчета коэффициентов полинома Ньютона тремя разными способами,
// номер способа передается последним параметром(1, 2 или 3),
// первые три параметра(a, b, n) - границы отрезка и степень полинома

List<double> coefPolyNewtonBase(double a, double b, int n, int number)

{
    // Считаем значения X и Y в равноотстоящих точках на отрезке[a, b]:
    List<double> X = createXList(a, b, n);
    List<double> Y = createYList(X);

    // В зависимости от выбранного варианта расчета запускаем нужную функцию:
    switch (number)
    {
        case 1:
            return coefPolyNewton1(X, Y);
        case 2:
            return coefPolyNewton2(X, Y);
        case 3:
            return coefPolyNewton3(X, Y);
        default: // Если номер метода некорректен
            X = createXList(0, 0, n);
            Y = createYList(X);
            return coefPolyNewton1(X, Y);
    }
}

// [!] Метод рассчета значения полинома Ньютона в точке xx. На вход получает матрицу
коэффициентов D,
// матрицу узловых точек X и значение xx, для которого надо посчитать P(xx)

double pointNewton(List<double> D, List<double> X, double xx)

{
    int n = X.Count; // Количество узловых точек
    double p = 0; // Начальное значение полинома в точке
}

```

```

for (int i = 0; i < n; i++) // Внешний цикл по коэффициентам D(i)

{
    double pp = D[i];

    for (int j = 0; j < i; j++) // Внутренний цикл по произведениям на разности (по X(j))

    {
        pp = pp * (xx - X[j]);
    }

    p = p + pp;
}

return p;
}

// [V] Метод очистки холста от старых графиков:

void clearPlot()

{
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования

    pane.CurveList.Clear(); // Очищаем список кривых
}

// [V] Метод нанесения на холст узловых точек:

void plotBasePoints(List<double> X, List<double> Y)

{
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования:

    PointPairList list = new PointPairList(); // Создаем список точек

    for (int i = 0; i < X.Count; i++)

    {
        list.Add(X[i], Y[i]); // Заполняем список точек:
    }

    // Создаем кривую с названием "Узловые точки". Обводка кружков будут рисоваться красным цветом
    // (Color.Red),

    // Опорные точки - кружки (SymbolType.Circle)

    LineItem myCurve = pane.AddCurve("Узловые точки", list, Color.Red, SymbolType.Circle);

    myCurve.Line.Visible = false; // У кривой линия будет невидимой

    myCurve.Symbol.Size = 10; // Размер кружков 10

    zedGraphControl1.AxisChange(); // Обновляем данные об осях.
}

```

```

zedGraphControl1.Invalidate(); // Обновляем график
}

// [V] перегруженный метод нанесения на холст узловых точек:
void plotBasePoints(double a, double b, int n)
{
    List<double> X = createXList(a, b, n); // Формируем вектор X
    List<double> Y = createYList(X); // Формируем вектор Y
    plotBasePoints(X, Y);
}

// [V] Метод нанесения на холст графика функции:
void plotFunction(double a, double b, int countOfPoints)
{
    // Передаем в функцию границы отрезка a и b, число разбиений (тут уже побольше, чем узлов, раз в 10 побольше)
    List<double> X = createXList(a, b, countOfPoints); // Формируем вектор X
    List<double> Y = createYList(X); // Формируем вектор Y
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования
    PointPairList list = new PointPairList(); // Создаем список точек
    for (int i = 0; i < X.Count; i++)
    {
        list.Add(X[i], Y[i]); // Заполняем список точек:
    }
    LineItem myCurve = pane.AddCurve("График функции", list, Color.Red, SymbolType.None); // Создаем кривую
    zedGraphControl1.AxisChange(); // Обновляем данные об осях.
    zedGraphControl1.Invalidate(); // Обновляем график
}

// [V] Метод нанесения на холст графика полинома:
// a,b - границы отрезка, n - степень полинома, countOfPoints - число точек на графике (больше n hfp d 10),
// number - номер способа расчета коэффициентов полинома
void plotNewton(double a, double b, int n, int countOfPoints, int number)
{

```

```

List<double> D = coefPolyNewtonBase(a, b, n, number); // Формируем список коэффициентов
способом number !!!!

List<double> basePointsX = createXList(a, b, n); // Формируем узловые точки

List<double> X = createXList(a, b, countOfPoints); // Формируем вектор X, точки уже для
графика, не для узлов

List<double> Y = new List<double>(); // Формируем вектор Y, для графика

foreach (double xx in X)

{
    Y.Add(pointNewton(D, basePointsX, xx));
}

GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования

PointPairList list = new PointPairList(); // Создаем список точек (координат x-y)

for (int i = 0; i < X.Count; i++)

{
    list.Add(X[i], Y[i]); // Заполняем список точек:
}

LineItem myCurve;

switch (number) // Создаем кривую с параметрами в зависимости от выбранного способа

{
    case 1:
        myCurve = pane.AddCurve("Полином Ньютона №1", list, Color.Blue, SymbolType.None);
        break;
    case 2:
        myCurve = pane.AddCurve("Полином Ньютона №2", list, Color.DarkGreen, SymbolType.None);
        break;
    case 3:
        myCurve = pane.AddCurve("Полином Ньютона №3", list, Color.Orange, SymbolType.None);
        break;
}

zedGraphControl1.AxisChange(); // Обновляем данные об осях

zedGraphControl1.Invalidate(); // Обновляем график

}

// [^] Метод автомасштаба (расчетный, работает лучше)

```

```

private void plotMashtab(double a, double b, int n)
{
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования

    // Устанавливаем интересующий нас интервал по оси X
    pane.XAxis.Scale.Min = a-0.1; // xmin_limit;
    pane.XAxis.Scale.Max = b+0.1; // xmax_limit;

    // Устанавливаем интересующий нас интервал по оси Y
    // Для этого нам надо получить минимум и максимум Y
    List<double> X = createXList(a, b, 1000); // Формируем вектор X
    List<double> Y = createYList(X); // Формируем вектор Y
    Y.Sort(); // Сортируем по возрастанию

    // Устанавливаем интересующий нас интервал по оси Y
    pane.YAxis.Scale.Min = Y[0] - 0.1; // ymin_limit;
    pane.YAxis.Scale.Max = Y[Y.Count-1] + 0.1; // ymax_limit;
    zedGraphControl1.AxisChange(); // Обновляем данные об осях
    zedGraphControl1.Invalidate(); // Обновляем график
}

// [^] Метод автомасштаба (встроенный, похоже)

private void autoMashtab()
{
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования

    // Установим масштаб по умолчанию для оси X
    pane.XAxis.Scale.MinAuto = true;
    pane.XAxis.Scale.MaxAuto = true;

    // Установим масштаб по умолчанию для оси Y
    pane.YAxis.Scale.MinAuto = true;
    pane.YAxis.Scale.MaxAuto = true;
    zedGraphControl1.AxisChange(); // Обновим данные об осях
    zedGraphControl1.Invalidate(); // Обновляем график
}

// [**] Аккумулирующий метод отстройки графиков

void plotButtonCommand(int number)

```

```

{
    // Читаем значения полей в соответствующие переменные:
    double a = 0.0;
    Double.TryParse(txtA.Text, out a);

    double b = 0.0;
    Double.TryParse(txtB.Text, out b);

    int n = 0;
    int.TryParse(txtN.Text, out n);

    // Работаем с графиками:
    clearPlot(); // Очищаем холст
    plotBasePoints(a, b, n); // Печатаем узловые точки
    plotFunction(a, b, 10000); // Печатаем график функции (1000 точек)
    if (number < 4) // Если получили номер от 1 до 3, то выводим один график
    {
        plotNewton(a, b, n, 10000, number); // Печатаем график полинома, посчитанного number
        способом, выводим 1000 точек
    }
    else // иначе выводим все графики
    {
        for (int i = 1; i < 4; i++)
        {
            plotNewton(a, b, n, 10000, i);
        }
    }
    //plotMashtab(a, b, n); // Выставляем автомасштаб (рассчетным способом)
}

// [^] Метод печати графика ошибки для текущего номера способа расчета number
void plotErrors(double a, double b, int n, int number)
{
    GraphPane pane = zedGraphControl1.GraphPane; // Получаем панель для рисования
    PointPairList Errors = new PointPairList(); // Создаем список точек (координат x-y)
    for (int i = 2; i <= 40; i++) // Пробегаем по всем значениям n от 2 до 40
    {
}

```

```

List<double> pointsX = createXList(a, b, i); // Узлы

List<double> D = coefPolyNewtonBase(a, b, i, number); // Коэффициенты

List<double> X = createXList(a, b, 100); // Точки для расчетов

List<double> Y1 = createYList(X); // Значения исходной функции

List<double> Y2 = new List<double>(); // Значения полинома

foreach (double x in X)

{

    Y2.Add(pointNewton(D, pointsX, x));

}

double error = delta(Y1, Y2);

Errors.Add(i, error); // заносим точку ошибки в список

if (i == n)

{

    txt_Error.Text += error + Environment.NewLine; // Если совпал номер, выведем в поле

}

}

// И тут надо напечатать:

LineItem myCurve;

switch (number) // Создаем кривую с параметрами в зависимости от выбранного способа

{

    case 1:

        myCurve = pane.AddCurve("Погрешность метода №1", Errors, Color.Blue, SymbolType.None);

        break;

    case 2:

        myCurve = pane.AddCurve("Погрешность метода №2", Errors, Color.DarkGreen, SymbolType.None);

        break;

    case 3:

        myCurve = pane.AddCurve("Погрешность метода №3", Errors, Color.Orange, SymbolType.None);

        break;

}

zedGraphControl1.AxisChange(); // Обновляем данные об осях

zedGraphControl1.Invalidate(); // Обновляем график

```

```

}

// [btn] Вывод графика полинома Ньютона по первому методу

private void btn_Newton1_Click(object sender, EventArgs e)
{
    plotButtonCommand(1);

}

// [btn] Вывод графика полинома Ньютона по второму методу

private void btn_Newton2_Click(object sender, EventArgs e)
{
    plotButtonCommand(2);

}

// [btn] Вывод графика полинома Ньютона по третьему методу

private void btn_Newton3_Click(object sender, EventArgs e)
{
    plotButtonCommand(3);

}

// [btn] Вывод графика полинома Ньютона по всем методам

private void btn_NewtonFull_Click(object sender, EventArgs e)
{
    plotButtonCommand(4);

}

// [btn] Вызов Метода автомасштаба №1

private void btn_Mashtab1_Click(object sender, EventArgs e)
{
    // Читаем значения полей в соответствующие переменные:

    double a = 0.0;
    Double.TryParse(txtA.Text, out a);

    double b = 0.0;
    Double.TryParse(txtB.Text, out b);

    int n = 0;
    int.TryParse(txtN.Text, out n);

    plotMashtab(a, b, n); // Выставляем автомасштаб (рассчетным способом)
}

```

```

}

// [btn] Вызов Метода автомасштаба №2

private void btn_Mashtab2_Click(object sender, EventArgs e)

{

// Читаем значения полей в соответствующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

autoMashtab(); // Выставляем автомасштаб (встроенным способом)

}

// [btn] Метод вызова печати графика ошибки

private void btn_Error_Click(object sender, EventArgs e)

{

// Читаем значения полей в соответствующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

txt_Error.Text = ""; // Очищаем

clearPlot(); // Очищаем

for (int j = 1; j < 4; j++) // Пробегаем по всем способам расчета коэффициентов

{



plotErrors(a, b, n, j);

}

}

}

}

```

Модуль MatLab f.m

```
function Y=f(X)
Y = X.^2-3;
end
```

Модуль MatLab coefPolyNewton1.m

```
function D = coefPolyNewton1(X, Y)

% Функция для вычисления коэффициентов полинома Ньютона методом решения n+1
% линейных алгебраических уравнений

D = X * 0;

n = length(X); % Количество узловых точек

for i = 1 : n % внешний цикл по D "слева"

chislitel = Y(i);

for j = 1 : i-1 % Вложенный цикл по D "справа"

pp = D(j);

for k = 1 : j-1 % Внутренний цикл по X

pp = pp * (X(i) - X(k));

end

chislitel = chislitel - pp;

end

% К этому моменту мы посчитали числитель, пора и за знаменатель взяться

znamenatel = 1;

for j = 1 : i-1

znamenatel = znamenatel * (X(i) - X(j));

end

% Посчитали знаменатель, делим одно на другое и сохраняем коэффициент в
% матрицу коэффициентов D:

D(i) = chislitel / znamenatel;

end

end
```

Модуль MatLab coefPolyNewton2.m

```
function D = coefPolyNewton2(X, Y)

% Функция для вычисления коэффициентов полинома Ньютона методом вычисления
% таблицы разделенных разностей RR

D = X * 0;

n = length(X); % Количество узловых точек

RR = zeros(n, n); % Заготовка - нулевой двумерный массив n x n

% Сначала заполняем первый столбец таблицы разделенных разностей RR:

j = 1;
for i = 1 : n
    RR(i,j) = Y(i);
end

% Затем на основе заполненного столбца заполняем оставшиеся:

for j = 2 : n % Внешний цикл по столбцам RR

m = n-j+1; % Каждый переход на столбец вправо сокращает количество разностей

for i = 1 : m % Внутренний цикл по строкам RR

    RR(i,j) = (RR(i, j-1) - RR(i+1, j-1)) / (X(i) - X(i+j-1)); % - X(i+j)

end

end

% И в конце переписываем из первой строки RR в матрицу коэффициентов D:

i = 1;
for j = 1 : n
    D(j) = RR(i,j);
end
end
```

Модуль MatLab coefPolyNewton3.m

```
function D = coefPolyNewton3(X, Y)

% Функция для вычисления коэффициентов полинома Ньютона методом вычисления
% суммы произведений без вычисления таблицы разделенных разностей

D = X * 0;
```

```

n = length(X); % Количество узловых точек

for m = 1 : n % Внешний цикл по D(m)

sum = 0; % Для каждого D(m) в первую очередь обнуляем сумму,
% а затем считаем произведение и суммируем с суммой:

for i = 1 : m % Вложенный цикл по Сумме

pp = Y(i); % Числитель одного из слагаемых

dd = 1; % Начальное значение знаменателя

% и вычисляем знаменатель:

for j = 1 : m % Внутренний цикл по Произведению

if (i ~= j) % Если не равны, иначе знаменатель обнуляется

dd = dd * (X(i) - X(j));

end

end

sum = sum + pp / dd; % Добавляем к сумме

end

D(m) = sum;

end

end

```

Модуль MatLab coefPolyNewtonBase.m

```

function [D, X] = coefPolyNewtonBase(a, b, n, number)

% Основная функция расчета коэффициентов полинома Ньютона тремя разными
% методами, номер метода передается последним параметром (1, 2 или 3),
% первые три параметра (a, b, n) - границы отрезка и степень полинома
%
% Функция возвращает матрицу коэффициентов D и матрицу узловых точек X,
% чтобы потом можно было рассчитать значение полинома Ньютона в любой точке
% по формуле через разности (x-x0) и т.д.
%
% Считаем значения X и Y в равноотстоящих точках на отрезке [a, b]
X = linspace(a, b, n+1); % n+1 узловых точек
Y = f(X);

```

% В зависимости от выбранного варианта расчета запускаем нужную функцию:

```
if (number == 1)
D = coefPolyNewton1(X, Y);
else
if (number == 2)
D = coefPolyNewton2(X, Y);
else
if (number == 3)
D = coefPolyNewton3(X, Y);
else
D = X * 0; % Если номер метода некорректен
end
end
end
end
```

Модуль MatLab pointNewton.m

```
function p = pointNewton(D, X, xx)

% Функция рассчета значения полинома Ньютона в точке xx

% На вход получает матрицу коэффициентов D, матрицу узловых точек X и

% значение xx, для которого надо посчитать P(xx)

n = length(X); % Количество узловых точек

p = 0; % Начальное значение полинома в точке

for i = 1 : n % Внешний цикл по коэффициентам D(i)

pp = D(i);

for j = 2 : i % Внутренний цикл по произведениям на разности (по X(j))

pp = pp * (xx - X(j-1));

end

p = p + pp;

end
```

Модуль MatLab **plotNewton.m**

```
function plotNewton(a, b, n, number)

% Основная функция, вычисляет коэффициенты полинома Ньютона по одному из
% методов расчета и строит сравнительные графики для исходной функции и
% полинома Ньютона, с нанесением узловых точек

% 1 Создаем новое окно для графика и подписываем оси

figure;

xlabel('x');

ylabel('y');

hold on;

grid on;

% 2 Вычисляем коэффициенты полинома Ньютона и узловые точки через базовую
% функцию:

[D, X] = coefPolyNewtonBase(a, b, n, number);

% 3 Печатаем узловые точки:

Y = f(X);

plot (X, Y, 'ro');

% 3 Печатаем график полинома Ньютона:

XX = linspace(a, b, 10000);

LX = XX * 0;

for i = 1:1:length(XX)

LX(i) = pointNewton(D, X, XX(i));

end

plot(XX, LX, 'b');

% 4 Печатаем график функции:

X = XX;

Y = f(X);

plot(X, Y, 'r--');

% 5 Подписываем легенду

title('Полиномиальная интерполяция', 'FontName', 'Courier');

name = strcat('Полином Ньютона №', int2str(number)); % Конкatenируем строки
```

```

h1 = legend( 'узловые точки', name, 'Функция');

set(h1, 'FontName', 'Courier');

% 6 Выставляем более-менее приемлемый масштаб:

axis([a b min(Y)-0.2 max(Y)+0.2])

end

```

Модуль MatLab plotNewtonFull.m

```

function plotNewtonFull(a, b, n)

% Основная функция, вычисляет коэффициенты полинома Ньютона по каждому из
% методов расчета и строит сравнительные графики для исходной функции и
% трех полиномов Ньютона, с нанесением узловых точек

% 1 Создаем новое окно для графика и подписываем оси

figure;
xlabel('x');
ylabel('y');
hold on;
grid on;

% 2 Вычисляем коэффициенты полинома №1 и узловые точки через базовую
% функцию:

[D1, X] = coefPolyNewtonBase(a, b, n, 1);

%D1

% 3 Печатаем узловые точки:

Y = f(X);
plot (X, Y, 'ro');

% 4 Вычисляем коэффициенты полинома №2 через
% вспомогательную функцию:

D2 = coefPolyNewton2(X, Y);

%D2

% 5 Вычисляем коэффициенты полинома №3 через
% вспомогательную функцию:

D3 = coefPolyNewton3(X, Y);

%D3

```

```

% 6 Печатаем график полинома Ньютона №1:
XX = linspace(a, b, 10000);
LX = XX * 0;
for i = 1:1:length(XX)
    LX(i) = pointNewton(D1, X, XX(i));
end
plot(XX, LX, 'b');

% 7 Печатаем график полинома Ньютона №2:
for i = 1:1:length(XX)
    LX(i) = pointNewton(D2, X, XX(i));
end
plot(XX, LX, 'g--.');

% 8 Печатаем график полинома Ньютона №3:
for i = 1:1:length(XX)
    LX(i) = pointNewton(D3, X, XX(i));
end
plot(XX, LX, 'y--');

% 9 Печатаем график функции:
Y = f(XX);
plot(XX, Y, 'r--');

% 10 Подписываем легенду
title('Полиномиальная интерполяция', 'FontName', 'Courier');
h1 = legend('Узловые точки', 'Полином Ньютона №1', 'Полином Ньютона №2', 'Полином Ньютона №3', 'Функция');
set(h1, 'FontName', 'Courier');

% 11 Выставляем более-менее приемлемый масштаб:
axis([a b min(Y)-0.2 max(Y)+0.2])
end

```

Модуль MatLab **deltaNewton.m**

```

function d = deltaNewton(Y1, Y2)

% Функция вычисления погрешности между значениями функции и значениями
% интерполяционного многочлена

```

```

n = length(Y1); % Количество узловых точек

D = Y1 * 0; % Матрица разности

for i = 1 : n

D(i) = Y1(i) - Y2(i);

end

d = 0;

for i = 1 : n

d = abs(D(i));

end

d = d / n;

end

```

Модуль MatLab **plotDeltaNewton.m**

```

function plotDeltaNewton(a, b, size1, size2)

% Функция построения графика зависимости погрешности от степени полинома для всех трех

% способов вычисления коэффициентов интерполяционного многочлена

color = ['b'; 'g'; 'r']; % Матрица цветов

% Создаем новое окно для графика и подписываем оси

figure;

xlabel('Степень полинома Ньютона, n');

ylabel('Средняя абсолютная погрешность');

grid on; hold on;

for number = 1:3 % Для все трех способов расчета

N = size1 : size2;

E = N * 0; % Создаем матрицу ошибок

k = 1;

for n = size1 : size2

[D, X] = coefPolyNewtonBase(a, b, n, number);

% Считаем значения X и Y в равноотстоящих точках на отрезке [a, b]

X1 = linspace(a, b, 10000); % 10000 узловых точек

Y1 = f(X);

% Считаем значения полинома в этих 10000 точках:

```

```

Y2 = Y1 * 0;

for i = 1:length(X1)
    Y2(i) = pointNewton(D, X, X1(i));
end

% Считаем значения ошибок

E(k) = deltaNewton(Y1, Y2);

k = k+1;

end

% Печатаем график:

plot (N, E, color(number));

end

% Подписываем легенду

title('Полиномиальная интерполяция', 'FontName', 'Courier');

h1 = legend('для полинома Ньютона №1', 'для полинома Ньютона №2', 'для полинома Ньютона №3');

set(h1, 'FontName', 'Courier');

% Выставляем более-менее приемлемый масштаб:

axis([size1 size2 0 3]) % axis([size1 size2 0-2 3+2])

end

```
