**Лабораторная работа № 4**

**«Интерполяция алгебраическими многочленами»**

**Чеботаревский Никита, 2 курс, 7 группа**

**Постановка задачи**

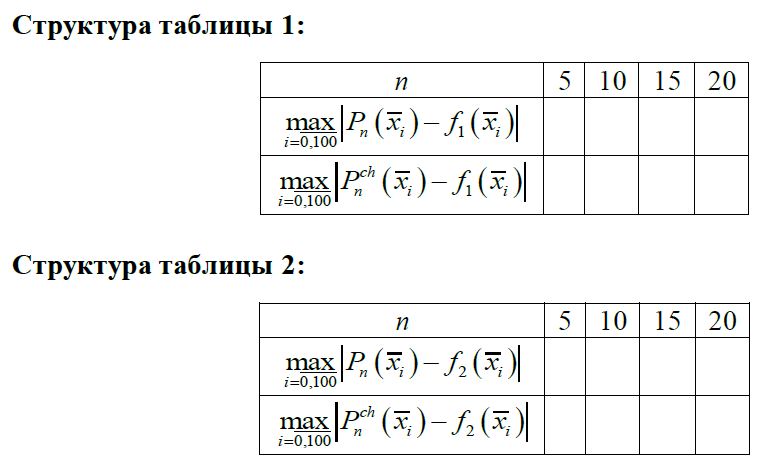
На отрезке [a, b] заданы функции . Построить многочлены степени n = 5, 10, 15, 20, интерполирующие каждую из них:

1. на сетке равноотстоящих узлов
2. на сетке чебышёвских узлов

Построить графики функций и интерполяционных многочленов для каждого n. Оценить погрешность интерполирования в узлах сетки , i = . Сравнить полученные результаты.

В отчёт включить следующую информацию:

1. Способ выбора узлов
2. Представление, использованное при построении интерполяционных многочленов
3. Графики функций и интерполяционных многочленов по равноотстоящим узлам. Графики функций и интерполяционных многочленов по чебышёвским узлам.
4. Графики функций и интерполяционных многочленов по равноотстоящим узлам. Графики функций и интерполяционных многочленов по чебышёвским узлам.
5. Оценка погрешности интерполирования функции , оформить в виде таблицы 1
6. Оценка погрешности интерполирования функции , оформить в виде таблицы 1
7. Листинг программы с комментариями



**Теоретические сведения**

Функции имеют следующий вид:



1. Построение чебышёвских узлов на отрезке [a, b]:

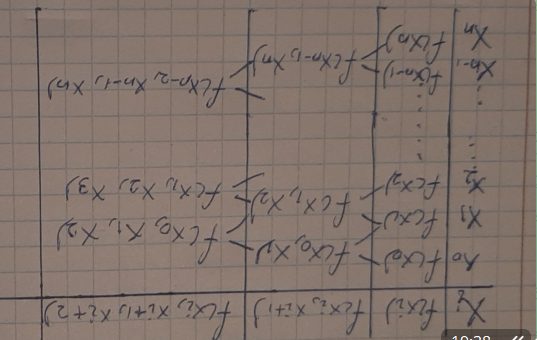
n – степень многочлена

1. Построение равноотстоящих узлов на отрезке [a, b]:

Построение интерполяционного многочлена:

Разделённой разностью k-ого порядка называют:

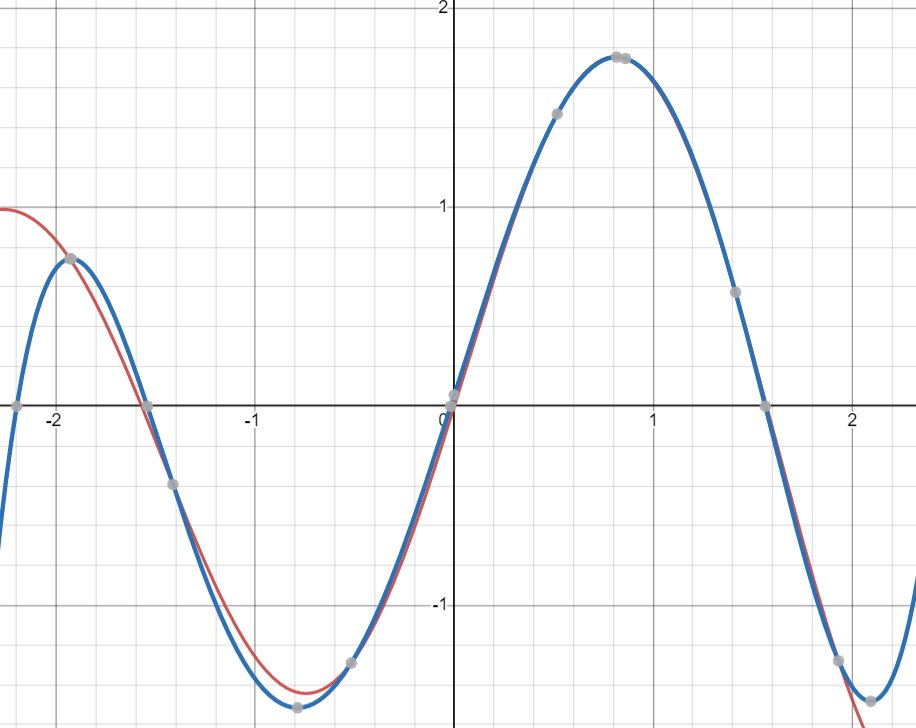
Таблица разделённых разностей:



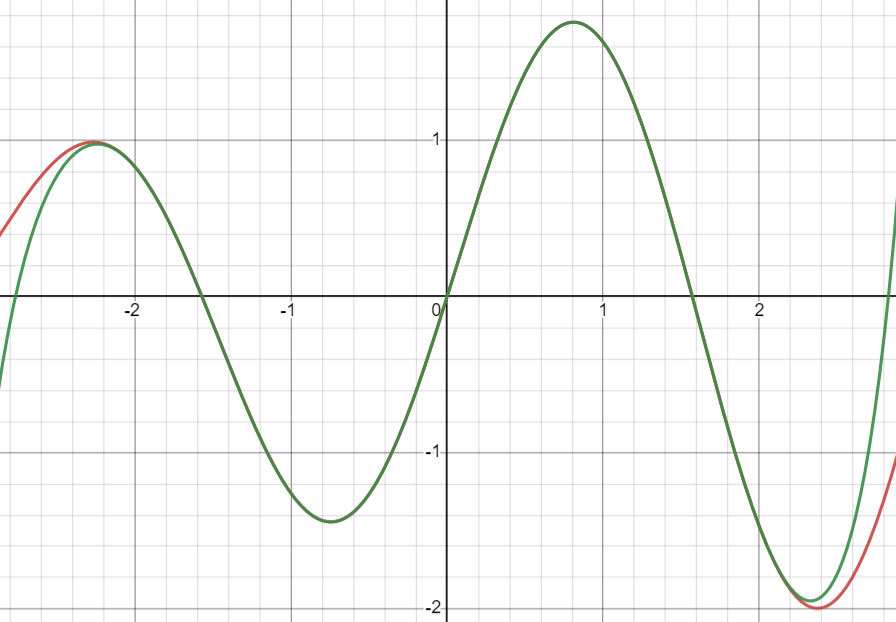
И тогда интерполяционный многочлен имеет вид:

**Построение графика функций и её интерполяционных многочленов.**

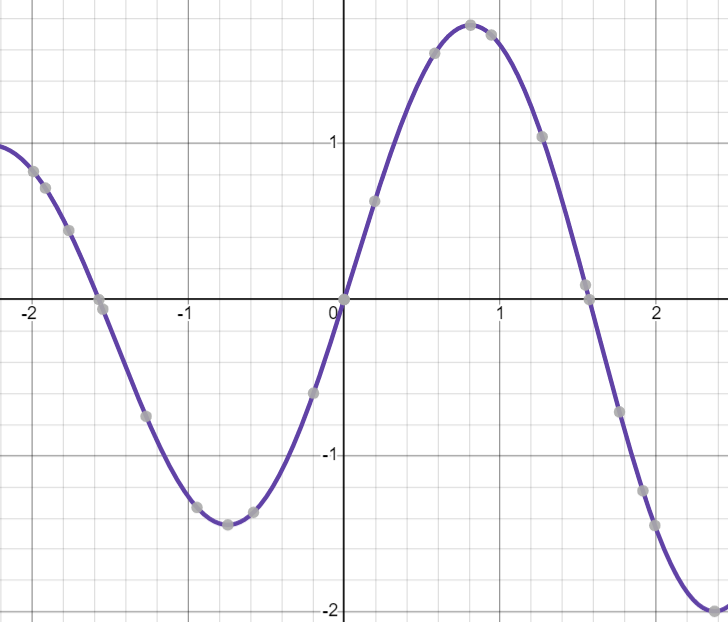
1. **Для чебышёвских узлов**
2. n = 5



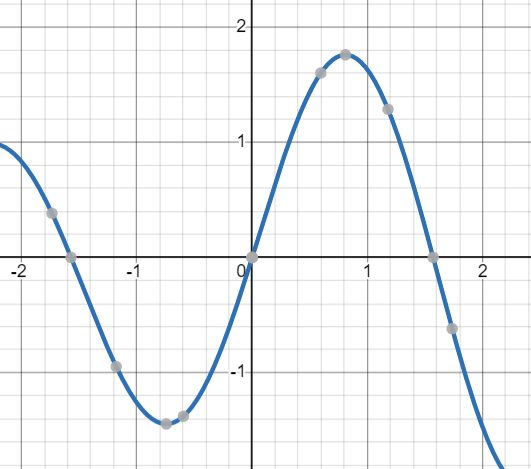
1. n = 10



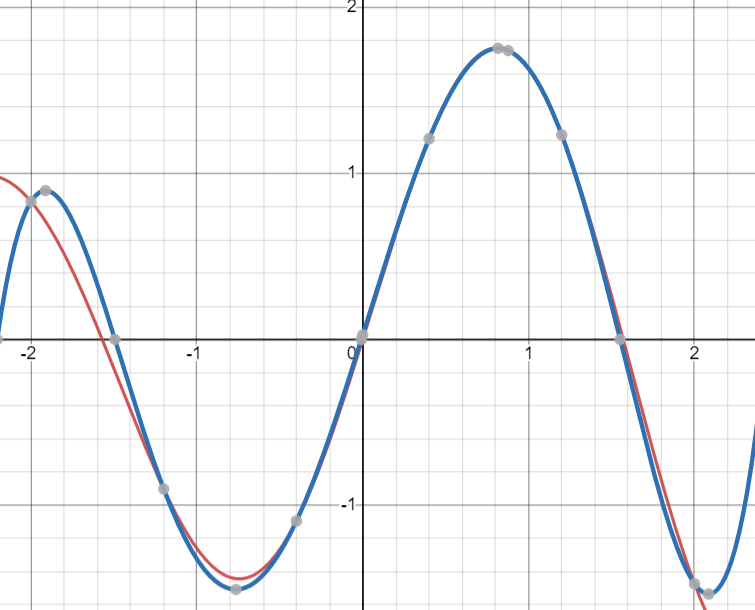
1. n = 15



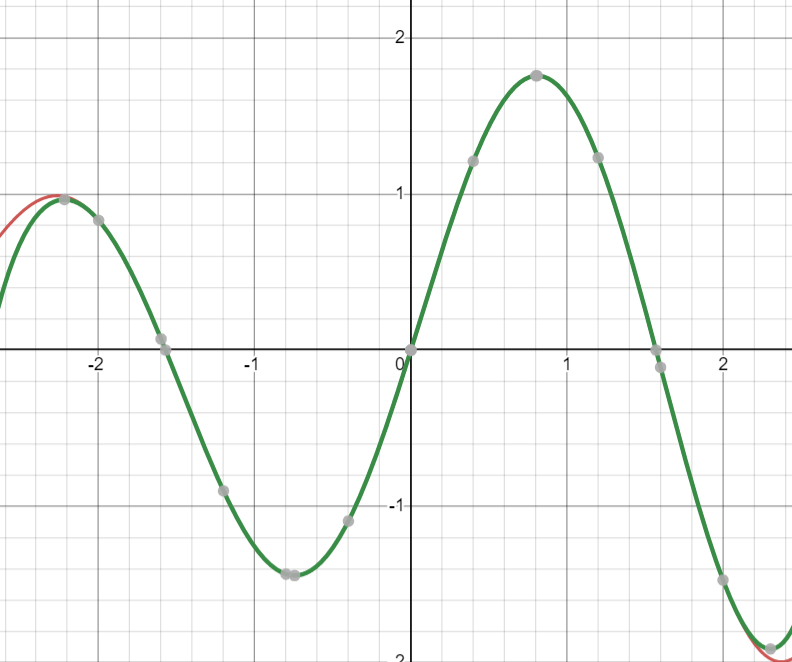
1. n = 20



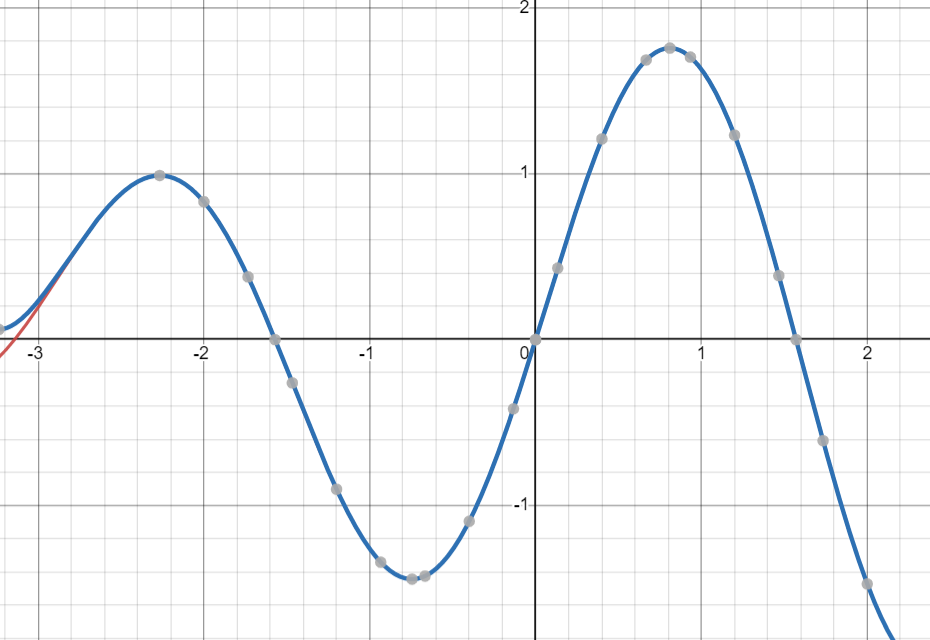
1. **Для равноотстоящих узлов:**
2. n = 5



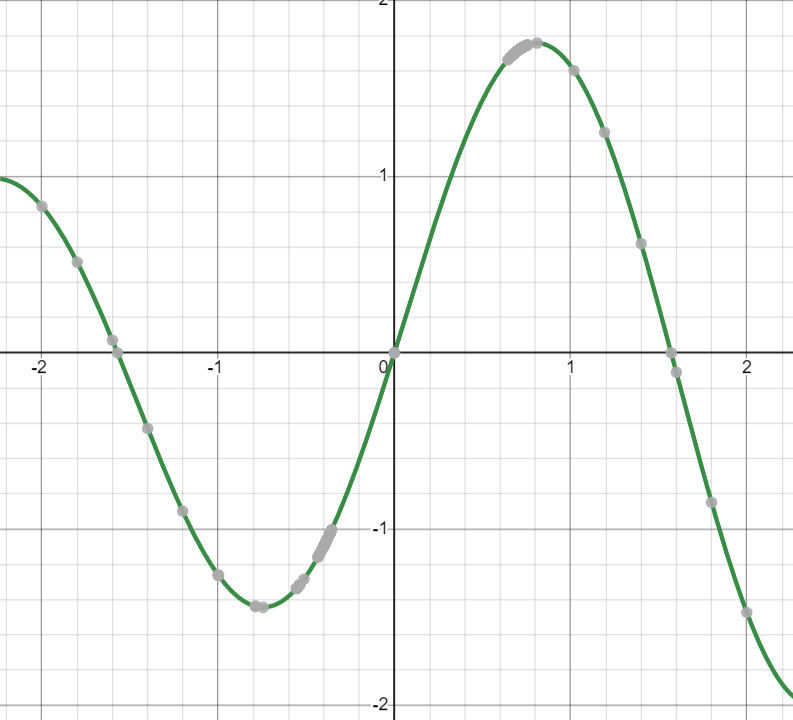
1. n = 10



1. n = 15



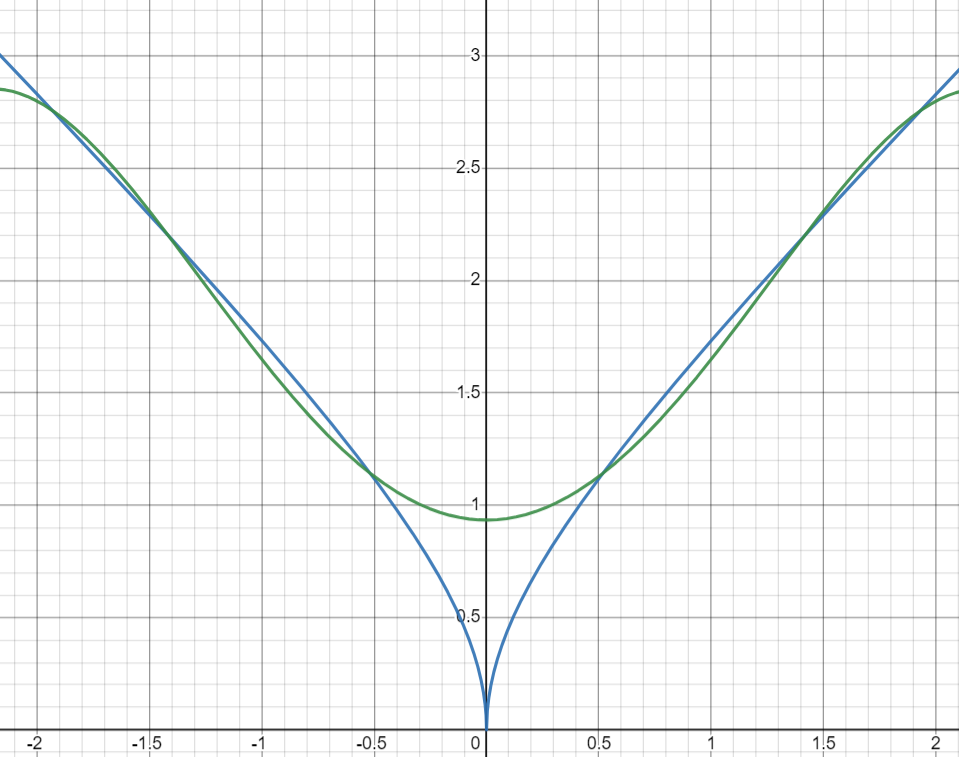
1. n = 20



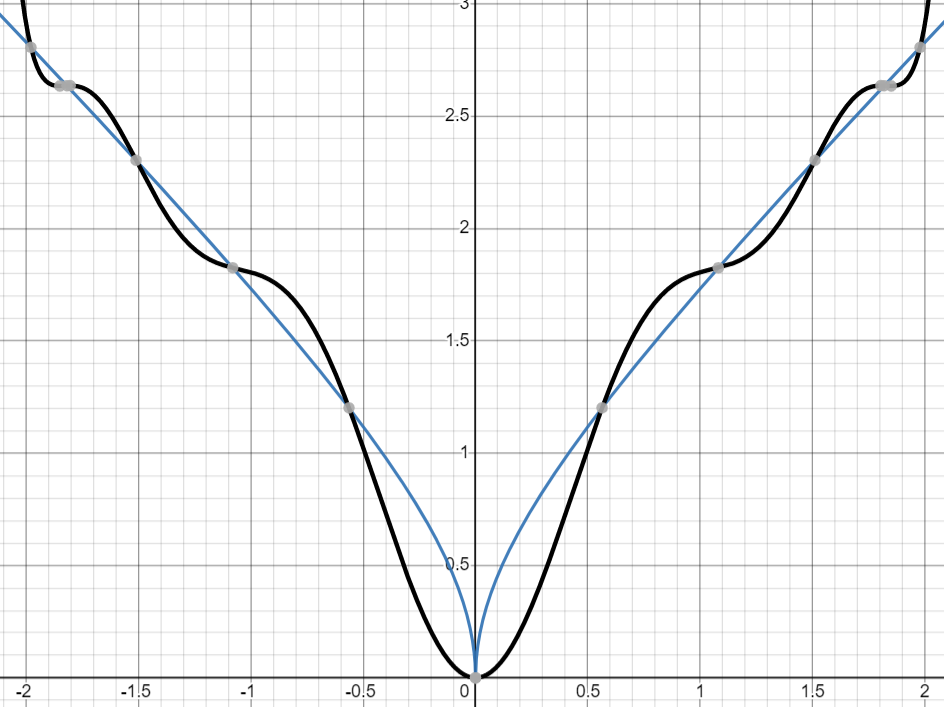
**Построение графика функций и её интерполяционных многочленов.**

**Для чебышёвских узлов:**

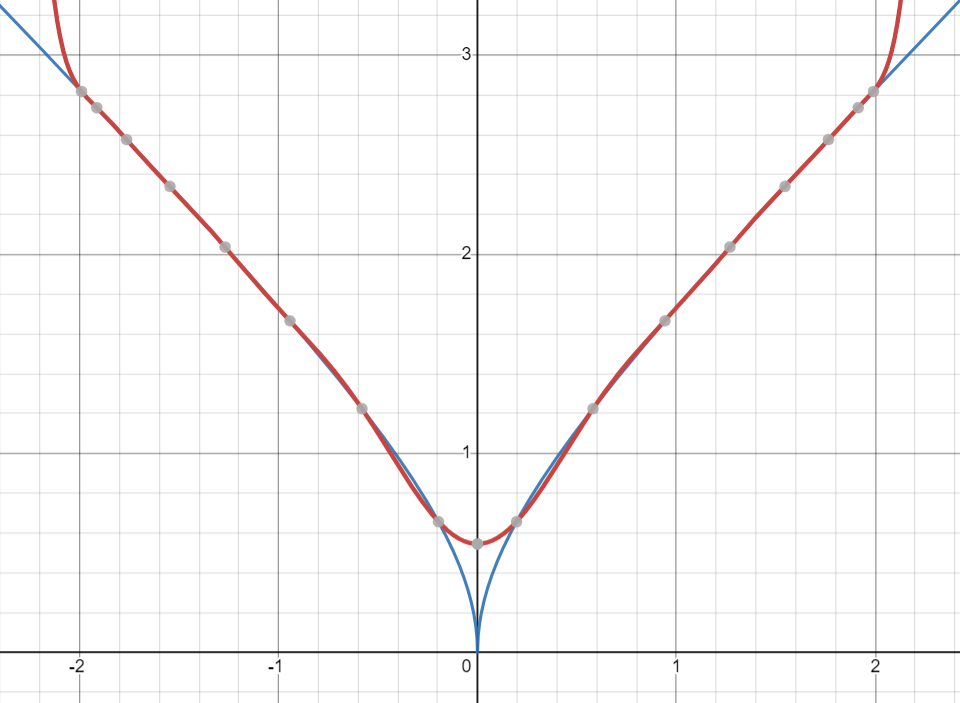
1. n = 5



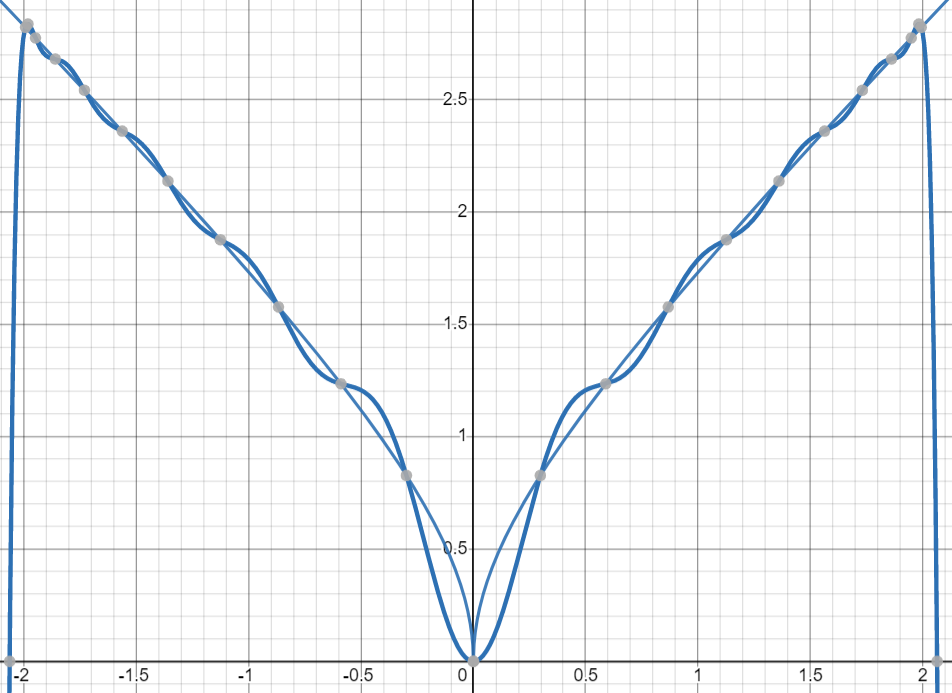
1. n = 10



1. n = 15

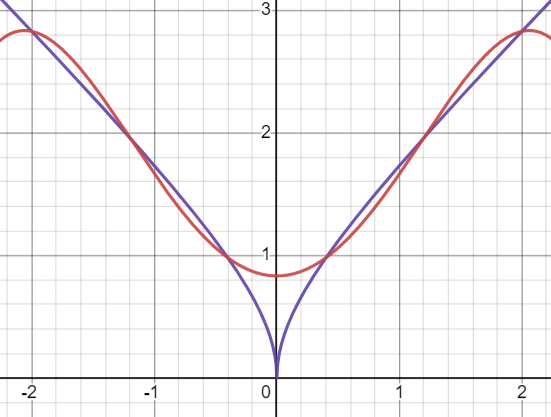


1. n = 20

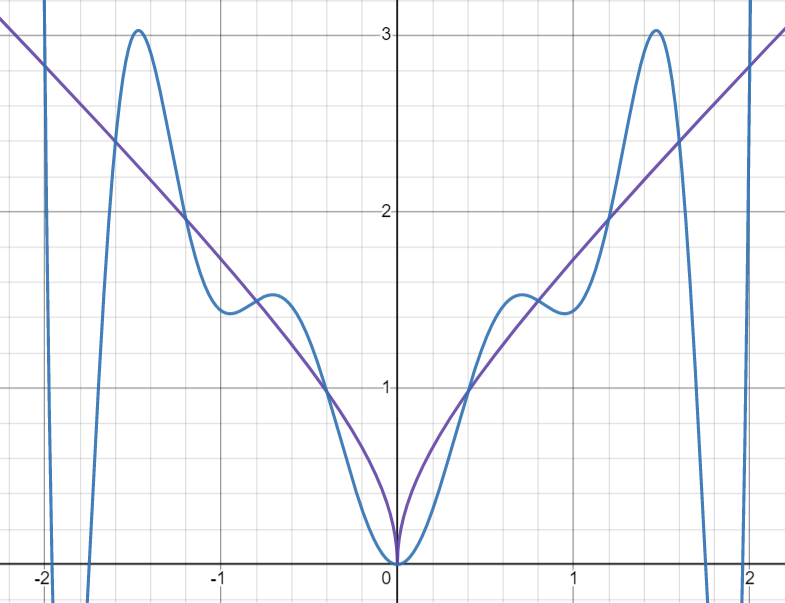


**Для равноотстоящих узлов:**

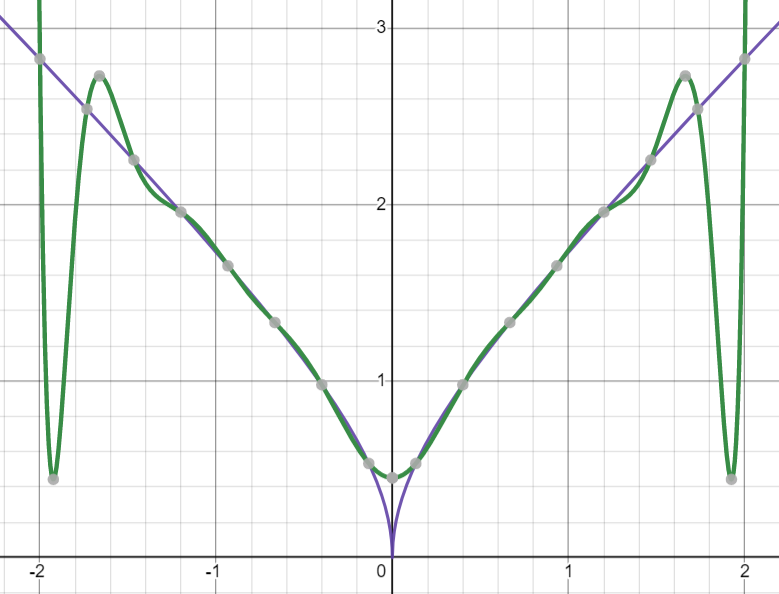
1. n = 5



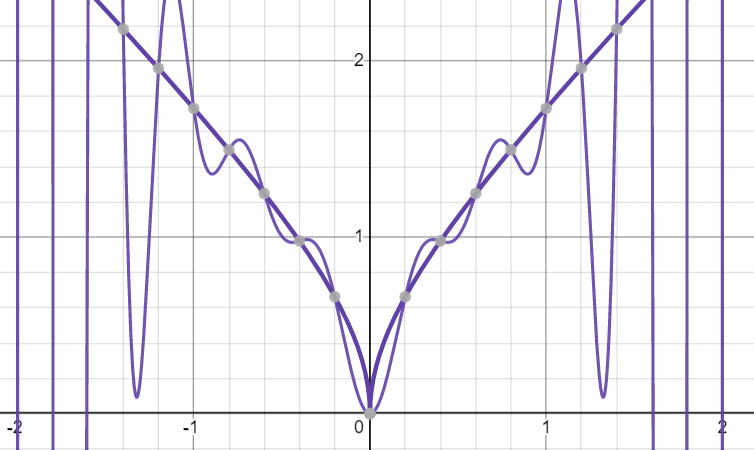
1. n = 10



1. n = 15



1. n = 20



**Оценка погрешности интерполирования функции**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 5 | 10 | 15 | 20 |
|  | 0.3027348 | 0.0014721 | 3.43724e-07 | 4.40339e-11 |
|  | 0.1355323 | 0.0001598 | 7.5625e-09 | 2.3459e-13 |

**Оценка погрешности интерполирования функции**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 5 | 10 | 15 | 20 |
|  | 0.8317568 | 4.4806121 | 2.2965362 | 884.3746 |
|  | 0.9338594 | 0.4533896 | 0.5448068 | 0.3212382 |

**Листинг программы**

*import* math *as* m  
*import* numpy *as* np  
  
  
*def* f1(x):  
 *return* m.sin(2 \* x) \* m.log(x + 5, m.e)  
  
  
*def* f2(x):  
 *return* m.sqrt(2 \* abs(x) + x \*\* 2)  
  
  
*def* count\_optimal\_knots(n, section):  
 x\_k = []  
 *for* i *in* range(n + 1):  
 coefficient = m.cos(((2 \* i + 1) / (2 \* n + 2)) \* m.pi)  
 x\_k.append(sum(section) / 2 + ((section[1] - section[0]) / 2) \* coefficient)  
  
 *return* x\_k  
  
  
*def* count\_equidistant\_nodes(n, section):  
 x\_k = [section[0]]  
 *for* i *in* range(1, n + 1):  
 x\_k.append(section[0] + i \* (section[1] - section[0]) / n)  
  
 *return* x\_k  
  
  
*def* pnx(x, numbers, knots):  
 value\_of\_polynomial = numbers[0]  
 array = []  
 *for* knot *in* knots:  
 array.append(x - knot)  
  
 *for* i *in* range(1, len(numbers)):  
 temp = numbers[i]  
 *for* j *in* range(i):  
 temp \*= array[j]  
 value\_of\_polynomial += temp  
  
 *return* value\_of\_polynomial  
  
  
*def* create\_x\_(section):  
 x\_ = []  
 *for* i *in* range(101):  
 x\_.append(section[0] + (i \* (section[1] - section[0])) / 100)  
  
 *return* x\_  
  
  
*def* create\_polynomial(n, knots, fun, index\_id):  
 matrix = [[0] \* (n + 1) *for* \_ *in* range(n + 1)]  
 *for* i *in* range(n + 1):  
 matrix[i][0] = fun[index\_id](knots[i])  
  
 counter = 0  
 *for* j *in* range(1, n + 1):  
 *for* k *in* range(n + 1 - j):  
 matrix[k][j] = (matrix[k + 1][j - 1] - matrix[k][j - 1]) / (knots[k + 1 + counter] - knots[k])  
 counter += 1  
 *return* matrix[0]  
  
  
*def* view\_of\_polynomial(numbers, knots):  
 res = f"{numbers[0]}"  
 res += " + "  
 *for* i *in* range(1, len(numbers)):  
 temp = f"{numbers[i]}"  
 *for* j *in* range(i):  
 temp += f"(x - {knots[j]})"  
 res += temp  
 res += " + "  
 *return* res  
  
  
*def* count\_accuracy(section, numbers, knots, func, index\_id):  
 new\_x\_ = create\_x\_(section)  
 result\_r = [0.0] \* 101  
 *for* k *in* range(101):  
 result\_r[k] = abs(pnx(new\_x\_[k], numbers, knots) - func[index\_id](new\_x\_[k]))  
  
 *return* max(result\_r)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 a\_b = np.array([-2, 2])  
 functions = [f1, f2]  
  
 *for* index *in* range(5, 21, 5):  
 print(f"Polynomial order for f1 {index} with optimal knots")  
 coefficients1\_1 = create\_polynomial(index, count\_optimal\_knots(index, a\_b), functions, 0)  
 result1\_1 = view\_of\_polynomial(coefficients1\_1, count\_optimal\_knots(index, a\_b))  
 print(result1\_1)  
 print(count\_accuracy(a\_b, coefficients1\_1, count\_optimal\_knots(index, a\_b), functions, 0))  
 print()  
  
 *for* index *in* range(5, 21, 5):  
 print(f"Polynomial order for f1 {index} with equal distance knots")  
 coefficients1\_2 = create\_polynomial(index, count\_equidistant\_nodes(index, a\_b), functions, 0)  
 result1\_2 = view\_of\_polynomial(coefficients1\_2, count\_equidistant\_nodes(index, a\_b))  
 print(result1\_2)  
 print(count\_accuracy(a\_b, coefficients1\_2, count\_equidistant\_nodes(index, a\_b), functions, 0))  
 print()  
  
 *for* index *in* range(5, 21, 5):  
 print(f"Polynomial order for f2 {index} with optimal knots")  
 coefficients2\_1 = create\_polynomial(index, count\_optimal\_knots(index, a\_b), functions, 1)  
 result2\_1 = view\_of\_polynomial(coefficients2\_1, count\_optimal\_knots(index, a\_b))  
 print(result2\_1)  
 print(count\_accuracy(a\_b, coefficients2\_1, count\_optimal\_knots(index, a\_b), functions, 1))  
 print()  
  
 *for* index *in* range(5, 21, 5):  
 print(f"Polynomial order for f2 {index} with equal distance knots")  
 coefficients2\_2 = create\_polynomial(index, count\_equidistant\_nodes(index, a\_b), functions, 1)  
 result2\_2 = view\_of\_polynomial(coefficients2\_2, count\_equidistant\_nodes(index, a\_b))  
 print(result2\_2)  
 print(count\_accuracy(a\_b, coefficients2\_2, count\_equidistant\_nodes(index, a\_b), functions, 1))  
 print()

**Вывод**

Таким образом, интерполяционный многочлен Ньютона позволяет найти оптимальное приближение некоторой функции степени на отрезке . Абсолютная погрешность зависит от степени интерполяционного многочлена, а также от выбранных узлов интерполирования. Наиболее точный результат дают оптимальные узлы (чебышёвские узлы).