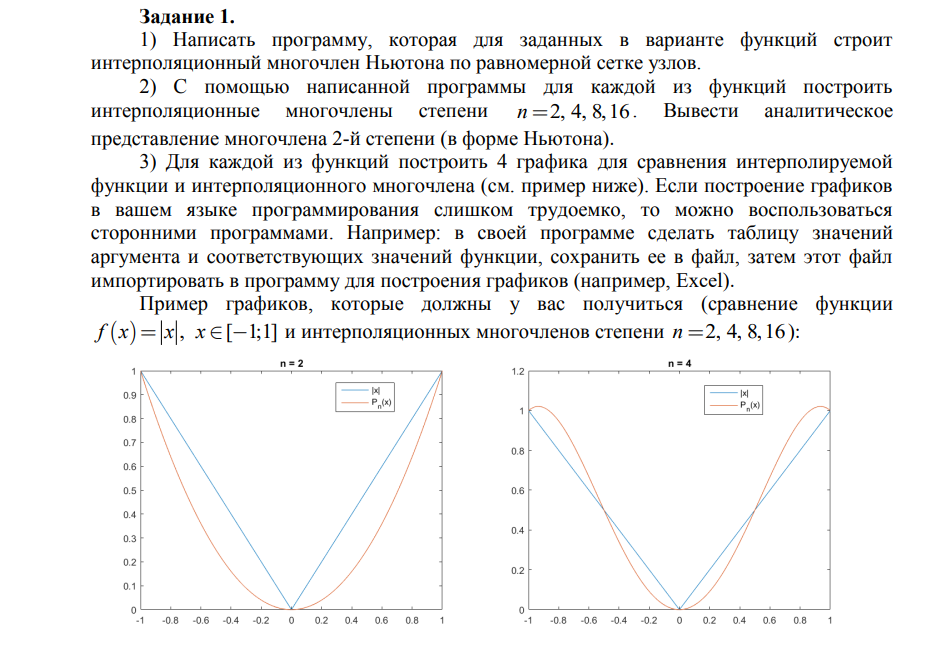
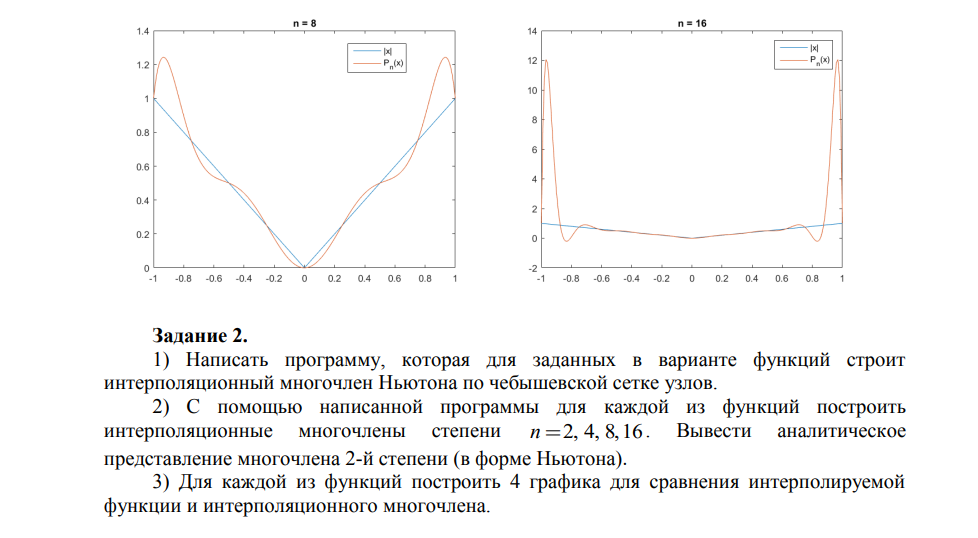
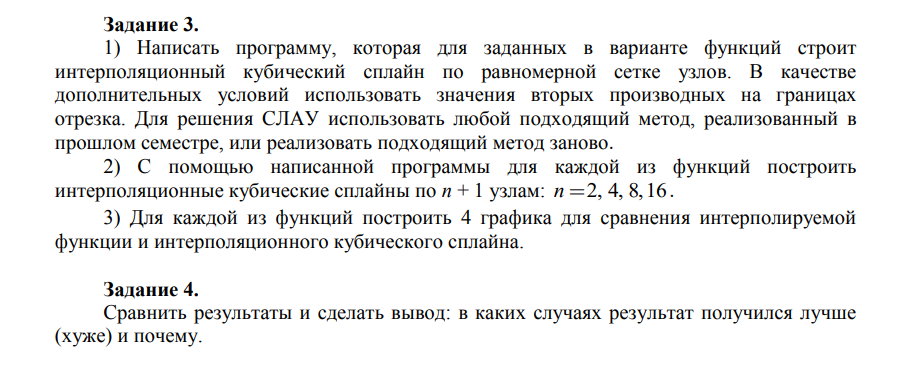
**Лабораторная работа № 1 «Приближение функций»**

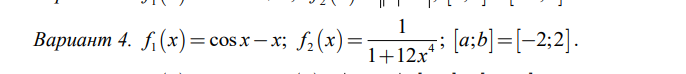
Выполнил студент 3 курса 4 группы ФПМИ БГУ Видевич Александр.

**Задача.**

****

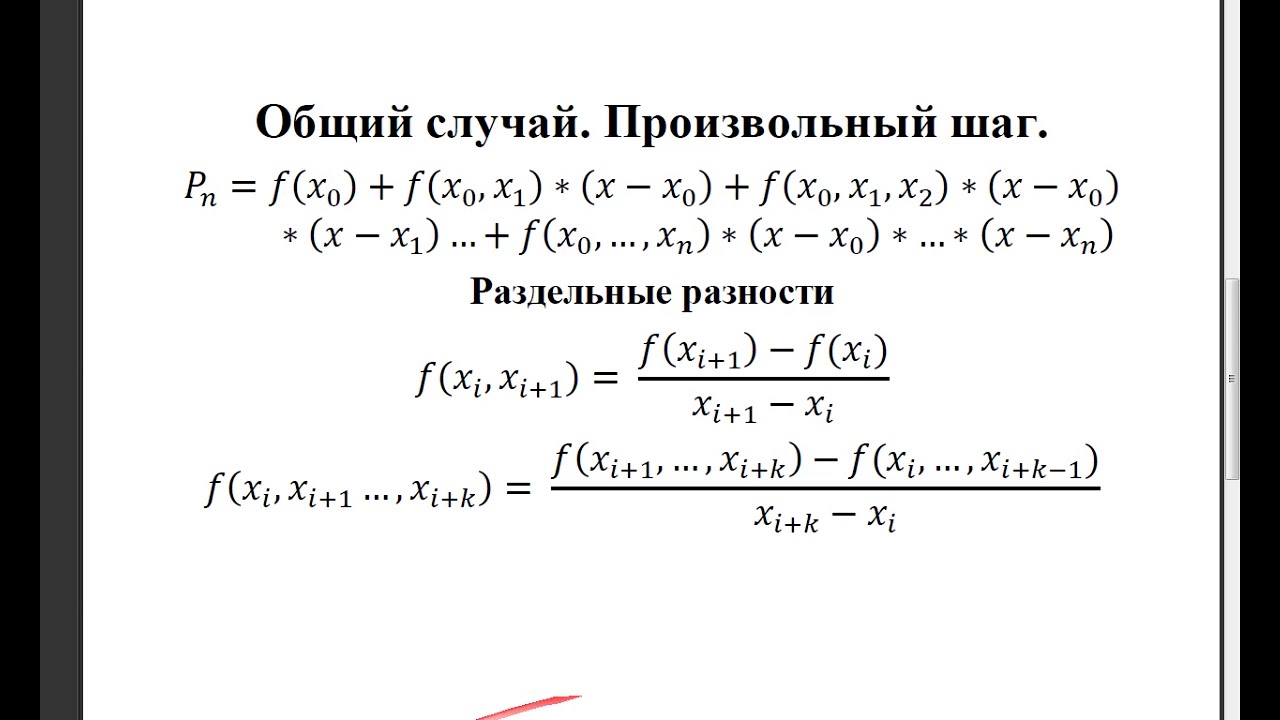
****

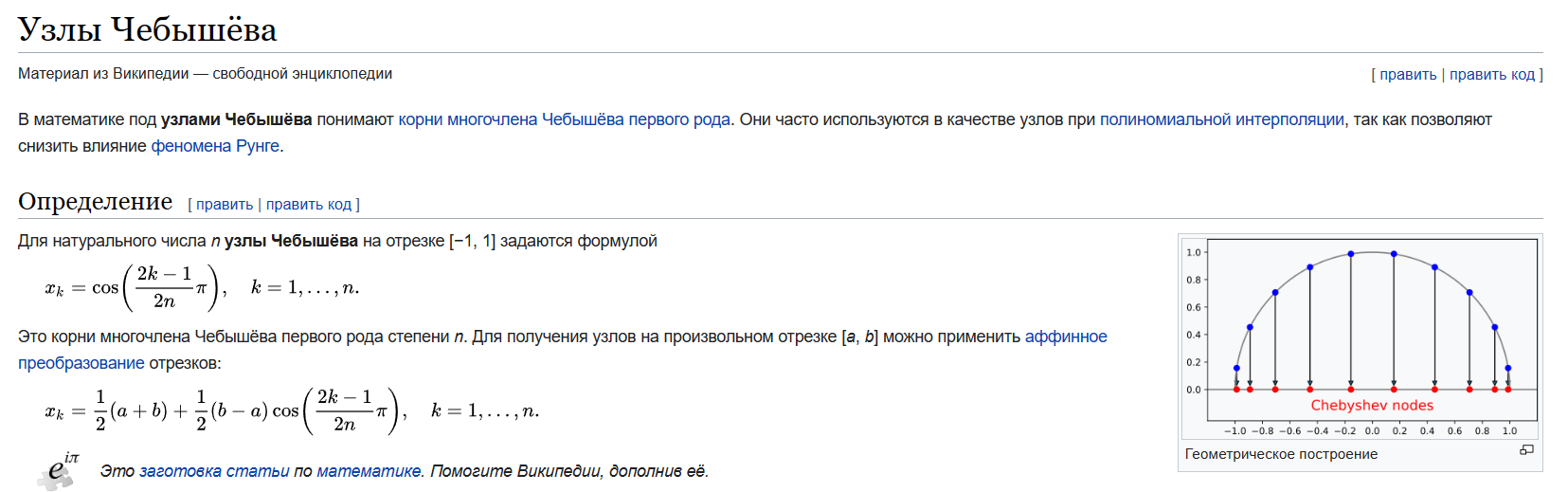
****

****

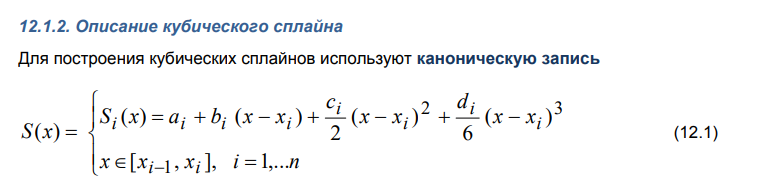
**Использованная теория.**

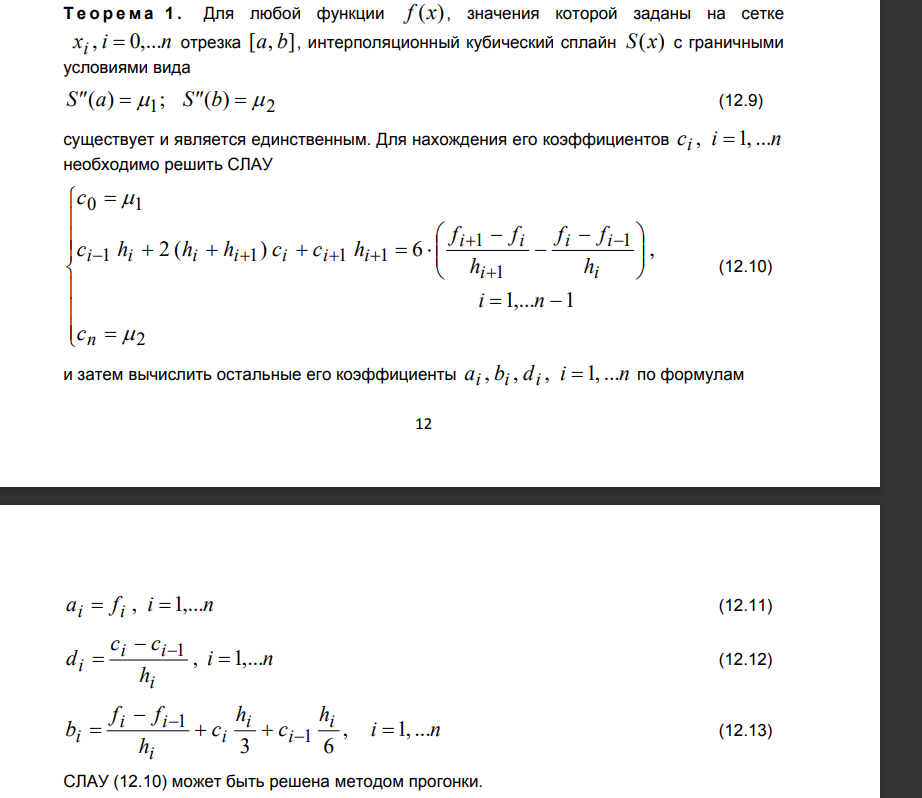
интерполяционный многочлен ньютона











**Листинг программы**

*#include* <cmath>

*#include* <functional>

*#include* <iostream>

*#include* <vector>

*#include* <fstream>

*#include* <filesystem>

*#include* <windows.h>

*const* double a = -*2*;

*const* double b = *2*;

*const* std::*vector*<int> degrees = {*2*, *4*, *8*, *16*};

double f1(double x)

{

*return* std::cos(x) - x;

}

double d2f1(double x)

{

*return* -cos(x);

}

double f2(double x)

{

*return* *1* / (*1* + *12* \* std::pow(x, *4*));

}

double d2f2(double x)

{

*return* (*2880* \* std::pow(x, *6*) - *144* \* x \* x) / (std::pow(*1* + *12* \* std::pow(x, *4*), *3*));

}

std::*vector*<double> NewtonInterpolation(*const* std::*vector*<double> *&*x, *const* std::*vector*<double> *&*y)

{

  int n = x.size();

  std::*vector*<double> dividedDifferences = y;

  std::*vector*<double> res;

*for* (int i = *1*; i <= n; ++i)

  {

    std::*vector*<double> nextDividedDifferences;

*for* (int j = *0*; j < n - i; ++j)

    {

      double dividedDifference = (dividedDifferences[j + *1*] - dividedDifferences[j]) / (x[j + i] - x[j]);

      nextDividedDifferences.push\_back(dividedDifference);

    }

    res.push\_back(dividedDifferences[*0*]);

    dividedDifferences = nextDividedDifferences;

  }

*return* res;

}

std::*function*<double(double)> CreatePolynomial(*const* std::*vector*<double> *&*x, *const* std::*vector*<double> *&*y)

{

  std::*vector*<double> coeffs = NewtonInterpolation(x, y);

*if* (x.size() == *3*)

  {

    std::cout << '*\n*';

    std::cout << "Analytical representation of the 2nd-degree polynomial: ";

    std::cout << coeffs[*0*] << " + " << coeffs[*1*] << " \* (x - " << x[*0*] << ") + "

              << coeffs[*2*] << " \* (x - " << x[*0*] << ") \* (x - " << x[*1*] << ")";

  }

*return* [coeffs, x](double point)

  {

    double res = *0*;

*for* (int i = *0*; i < x.size(); i++)

    {

      double term = coeffs[i];

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

      {

        term \*= (point - x[j]);

      }

      res += term;

    }

*return* res;

  };

}

std::*vector*<double> GetEqualPoints(int n)

{

  double step = static\_cast<double>(b - a) / n;

  std::*vector*<double> points;

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

  {

    points.push\_back(a + i \* step);

  }

*return* points;

}

std::*vector*<double> GetChebyshevPoints(int n)

{

  double step = (b - a) / *2*;

  double middle = (a + b) / *2*;

  std::*vector*<double> points;

*for* (int i = *0*; i < n + *1*; i++)

  {

    points.push\_back(middle + step \* std::cos(((*2* \* i + *1*) \* M\_PI) / (*2* \* (n + *1*))));

  }

*return* points;

}

void Write(*const* std::string *&*filename, *const* std::*function*<double(double)> *&*function, double l, double r)

{

  std::ofstream file(filename);

*if* (file.is\_open())

  {

*for* (double i = l; i <= r; i += *0.01*)

    {

      file << i << " " << function(i) << std::endl;

    }

    file.close();

  }

}

void ForwardRunThrough(std::*vector*<double> *&*b, std::*vector*<double> *&*c, std::*vector*<double> *&*a, std::*vector*<double> *&*vector)

{

*const* int n = b.size();

  vector[*0*] /= c[*0*];

  b[*0*] /= -c[*0*];

*for* (int i = *1*; i < n; ++i)

  {

    b[i] /= -(c[i] + a[i - *1*] \* b[i - *1*]);

    vector[i] = (vector[i] - a[i - *1*] \* vector[i - *1*]) / (c[i] + a[i - *1*] \* b[i - *1*]);

  }

  vector[n] = (vector[n] - a[n - *1*] \* vector[n - *1*]) / (c[n] + a[n - *1*] \* b[n - *1*]);

}

std::*vector*<double> ReverseRunThrough(*const* std::*vector*<double> *&*b, *const* std::*vector*<double> *&*vector)

{

*const* int n = b.size();

  std::*vector*<double> solution(n + *1*);

  solution[n] = vector[n];

*for* (int i = n - *1*; i >= *0*; --i)

  {

    solution[i] = b[i] \* solution[i + *1*] + vector[i];

  }

*return* solution;

}

std::*vector*<double> SolveSystemUsingRunThroughMethod(std::*vector*<double> b, std::*vector*<double> c, std::*vector*<double> a, std::*vector*<double> vector)

{

  ForwardRunThrough(b, c, a, vector);

*return* ReverseRunThrough(b, vector);

}

std::*vector*<std::*function*<double(double)>> GetSpline(*const* std::*vector*<double> *&*x, *const* std::*function*<double(double)> *&*f, *const* std::*function*<double(double)> *&*d2f)

{

*const* int n = x.size();

*const* double h = x[*1*] - x[*0*];

  std::*vector*<double> vector(n);

  std::*vector*<double> ac(n - *1*, *0*);

  std::*vector*<double> cc(n, *0*);

  std::*vector*<double> bc(n - *1*, *0*);

  std::*vector*<std::*vector*<double>> matrix(n, std::*vector*<double>(n, *0*));

  cc[*0*] = *1*;

  cc[n - *1*] = *1*;

  vector[*0*] = d2f(a);

  vector[n - *1*] = d2f(b);

*for* (int i = *1*; i <= n - *2*; ++i)

  {

    vector[i] = *6* \* ((f(x[i + *1*]) - f(x[i])) / h - (f(x[i]) - f(x[i - *1*])) / h);

    ac[i - *1*] = h;

    cc[i] = *2* \* (h + h);

    bc[i] = h;

  }

*const* std::*vector*<double> c = SolveSystemUsingRunThroughMethod(bc, cc, ac, vector);

  std::*vector*<std::*function*<double(double)>> functions;

*for* (int i = *1*; i <= n - *1*; ++i)

  {

    int temp = i;

    functions.push\_back([f, c, x, h, temp](double point)

                        {

*const* double a = f(x[temp]);

*const* double b = (f(x[temp]) - f(x[temp - *1*])) / h + c[temp] \* h / *3* + c[temp - *1*] \* h / *6*;

*const* double d = (c[temp] - c[temp - *1*]) / h;

*return* a + b \* (point - x[temp]) + c[temp] / *2* \* (point - x[temp]) \* (point - x[temp]) + d / *6* \* (point - x[temp]) \* (point - x[temp]) \* (point - x[temp]); });

  }

*return* functions;

}

void CreateFolders()

{

*if* (!std::filesystem::exists("task1"))

  {

    std::filesystem::path folderPath("task1");

    std::filesystem::create\_directory(folderPath);

  }

*if* (!std::filesystem::exists("task2"))

  {

    std::filesystem::path folderPath("task2");

    std::filesystem::create\_directory(folderPath);

  }

*if* (!std::filesystem::exists("task3"))

  {

    std::filesystem::path folderPath("task3");

    std::filesystem::create\_directory(folderPath);

  }

*if* (!std::filesystem::exists("real"))

  {

    std::filesystem::path folderPath("real");

    std::filesystem::create\_directory(folderPath);

  }

}

void Run()

{

  CreateFolders();

  Write("real/f1.txt", f1, a, b);

  Write("real/f2.txt", f2, a, b);

*for* (int deg : degrees)

  {

    std::*vector*<double> x1 = GetEqualPoints(deg);

    std::*vector*<double> x2 = GetChebyshevPoints(deg);

    std::*vector*<double> y1\_1;

    std::*vector*<double> y2\_1;

    std::*vector*<double> y1\_2;

    std::*vector*<double> y2\_2;

*for* (int i = *0*; i < x1.size(); i++)

    {

      y1\_1.push\_back(f1(x1[i]));

      y2\_1.push\_back(f2(x1[i]));

      y1\_2.push\_back(f1(x2[i]));

      y2\_2.push\_back(f2(x2[i]));

    }

    std::*function*<double(double)> poly1\_1 = CreatePolynomial(x1, y1\_1);

    std::*function*<double(double)> poly2\_1 = CreatePolynomial(x1, y2\_1);

    std::*function*<double(double)> poly1\_2 = CreatePolynomial(x2, y1\_2);

    std::*function*<double(double)> poly2\_2 = CreatePolynomial(x2, y2\_2);

    Write("task1/poly1\_" + std::to\_string(deg) + "\_deg.txt", poly1\_1, a, b);

    Write("task1/poly2\_" + std::to\_string(deg) + "\_deg.txt", poly2\_1, a, b);

    Write("task2/poly1\_" + std::to\_string(deg) + "\_deg.txt", poly1\_2, a, b);

    Write("task2/poly2\_" + std::to\_string(deg) + "\_deg.txt", poly2\_2, a, b);

    std::*vector*<std::*function*<double(double)>> functions1 = GetSpline(x1, f1, d2f1);

    std::*vector*<std::*function*<double(double)>> functions2 = GetSpline(x1, f2, d2f2);

*for* (int i = *0*; i < functions1.size(); ++i)

    {

      Write("task3/spline1\_" + std::to\_string(i) + "\_" + std::to\_string(deg + *1*) + "\_deg.txt", functions1[i], x1[i], x1[i + *1*]);

      Write("task3/spline2\_" + std::to\_string(i) + "\_" + std::to\_string(deg + *1*) + "\_deg.txt", functions2[i], x1[i], x1[i + *1*]);

    }

  }

}

*// sorry for hardcode*

void MakePlots()

{

  std::system("output*\\*plot.exe real/f1.txt task1/poly1\_2\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task1/poly1\_4\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task1/poly1\_8\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task1/poly1\_16\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task1/poly2\_2\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task1/poly2\_4\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task1/poly2\_8\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task1/poly2\_16\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task2/poly1\_2\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task2/poly1\_4\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task2/poly1\_8\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task2/poly1\_16\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task2/poly2\_2\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task2/poly2\_4\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task2/poly2\_8\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task2/poly2\_16\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task3/spline1\_0\_3\_deg.txt task3/spline1\_1\_3\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task3/spline1\_0\_5\_deg.txt task3/spline1\_1\_5\_deg.txt task3/spline1\_2\_5\_deg.txt task3/spline1\_3\_5\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task3/spline1\_0\_9\_deg.txt task3/spline1\_1\_9\_deg.txt task3/spline1\_2\_9\_deg.txt task3/spline1\_3\_9\_deg.txt task3/spline1\_4\_9\_deg.txt task3/spline1\_5\_9\_deg.txt task3/spline1\_6\_9\_deg.txt task3/spline1\_7\_9\_deg.txt");

  std::system("output*\\*plot.exe real/f1.txt task3/spline1\_0\_17\_deg.txt task3/spline1\_1\_17\_deg.txt task3/spline1\_2\_17\_deg.txt task3/spline1\_3\_17\_deg.txt task3/spline1\_4\_17\_deg.txt task3/spline1\_5\_17\_deg.txt task3/spline1\_6\_17\_deg.txt task3/spline1\_7\_17\_deg.txt task3/spline1\_8\_17\_deg.txt task3/spline1\_9\_17\_deg.txt task3/spline1\_10\_17\_deg.txt task3/spline1\_11\_17\_deg.txt task3/spline1\_12\_17\_deg.txt task3/spline1\_13\_17\_deg.txt task3/spline1\_14\_17\_deg.txt task3/spline1\_15\_17\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task3/spline2\_0\_3\_deg.txt task3/spline2\_1\_3\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task3/spline2\_0\_5\_deg.txt task3/spline2\_1\_5\_deg.txt task3/spline2\_2\_5\_deg.txt task3/spline2\_3\_5\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task3/spline2\_0\_9\_deg.txt task3/spline2\_1\_9\_deg.txt task3/spline2\_2\_9\_deg.txt task3/spline2\_3\_9\_deg.txt task3/spline2\_4\_9\_deg.txt task3/spline2\_5\_9\_deg.txt task3/spline2\_6\_9\_deg.txt task3/spline2\_7\_9\_deg.txt");

  std::system("output*\\*plot.exe real/f2.txt task3/spline2\_0\_17\_deg.txt task3/spline2\_1\_17\_deg.txt task3/spline2\_2\_17\_deg.txt task3/spline2\_3\_17\_deg.txt task3/spline2\_4\_17\_deg.txt task3/spline2\_5\_17\_deg.txt task3/spline2\_6\_17\_deg.txt task3/spline2\_7\_17\_deg.txt task3/spline2\_8\_17\_deg.txt task3/spline2\_9\_17\_deg.txt task3/spline2\_10\_17\_deg.txt task3/spline2\_11\_17\_deg.txt task3/spline2\_12\_17\_deg.txt task3/spline2\_13\_17\_deg.txt task3/spline2\_14\_17\_deg.txt task3/spline2\_15\_17\_deg.txt");

}

int main()

{

  Run();

  MakePlots();

*return* *0*;

}

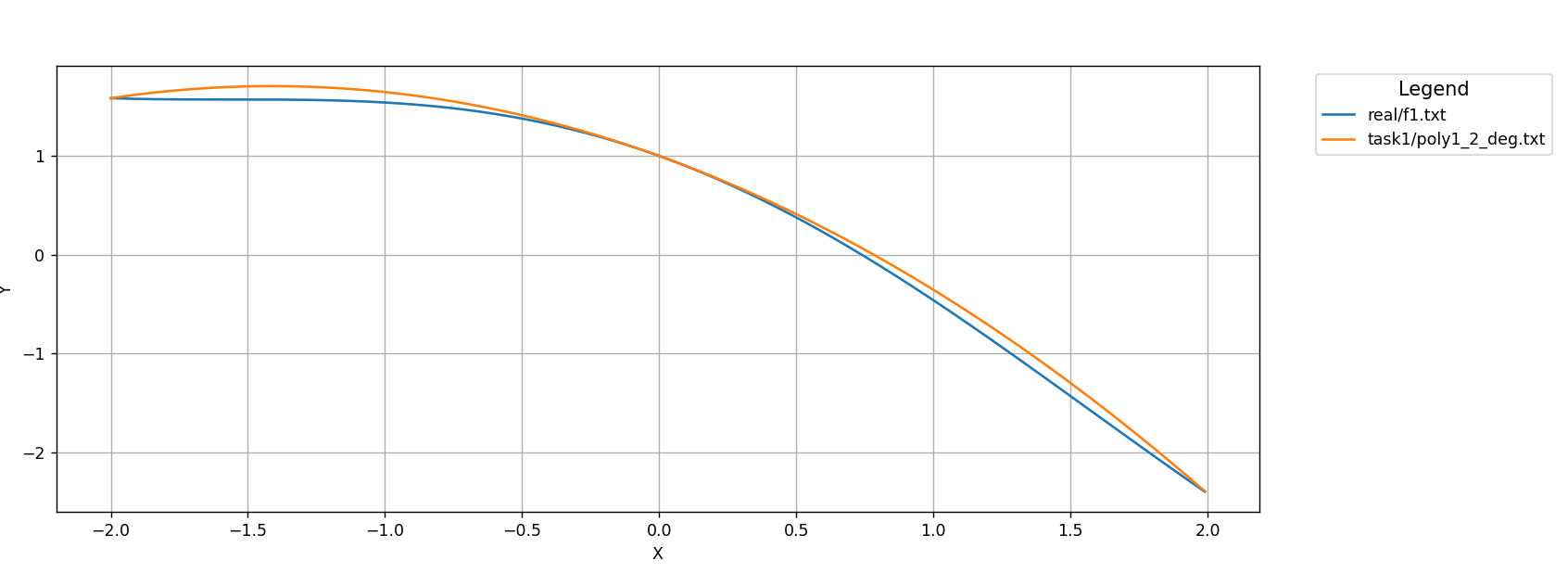
**Выходные данные.**

Задание 1

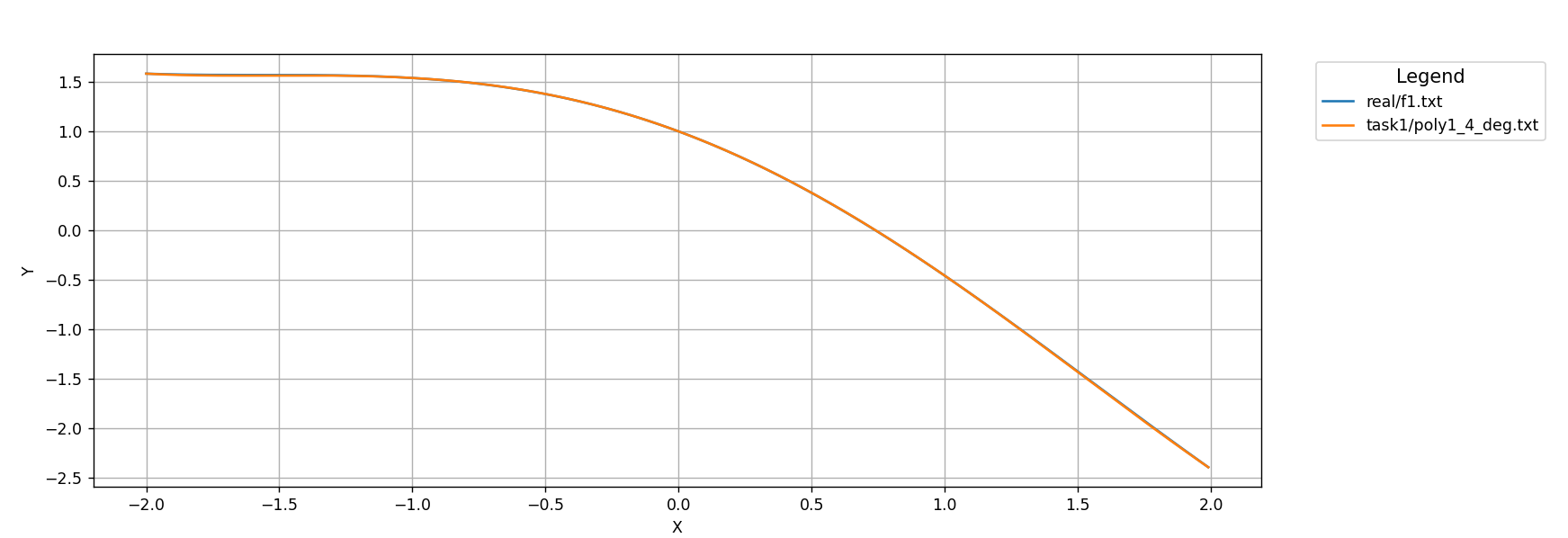


Analytical representation of the 2nd-degree polynomial: 1.58385 + -0.291927 \* (x - -2) + -0.354037 \* (x - -2) \* (x - 0)

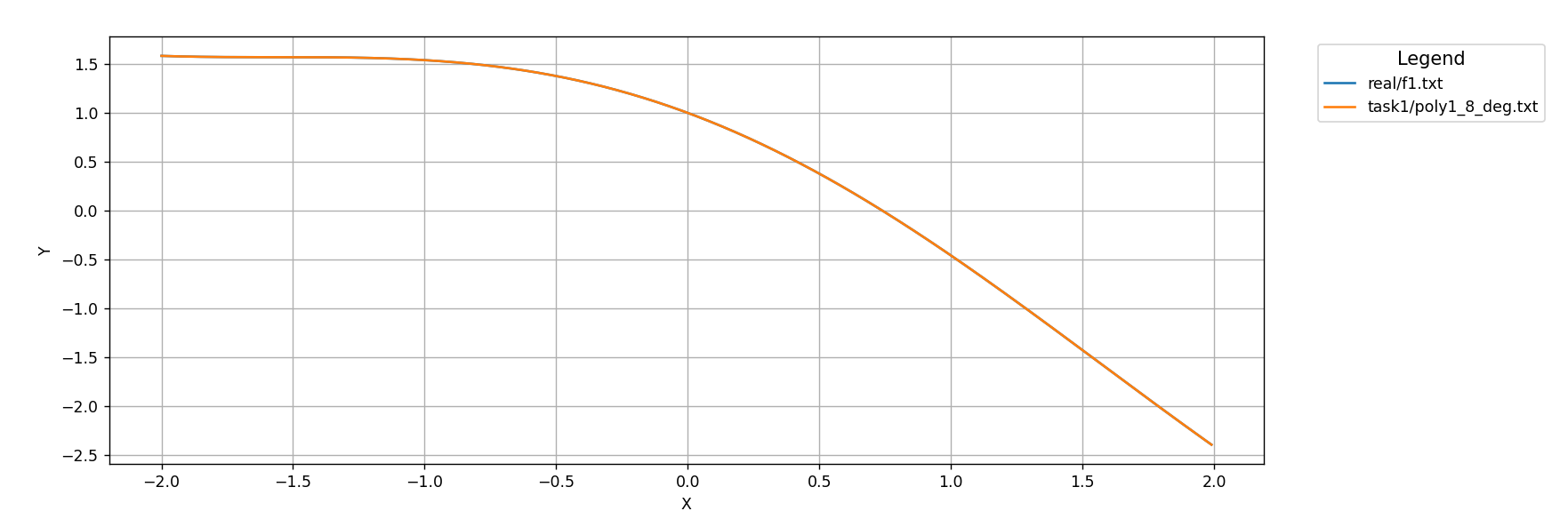
n=2



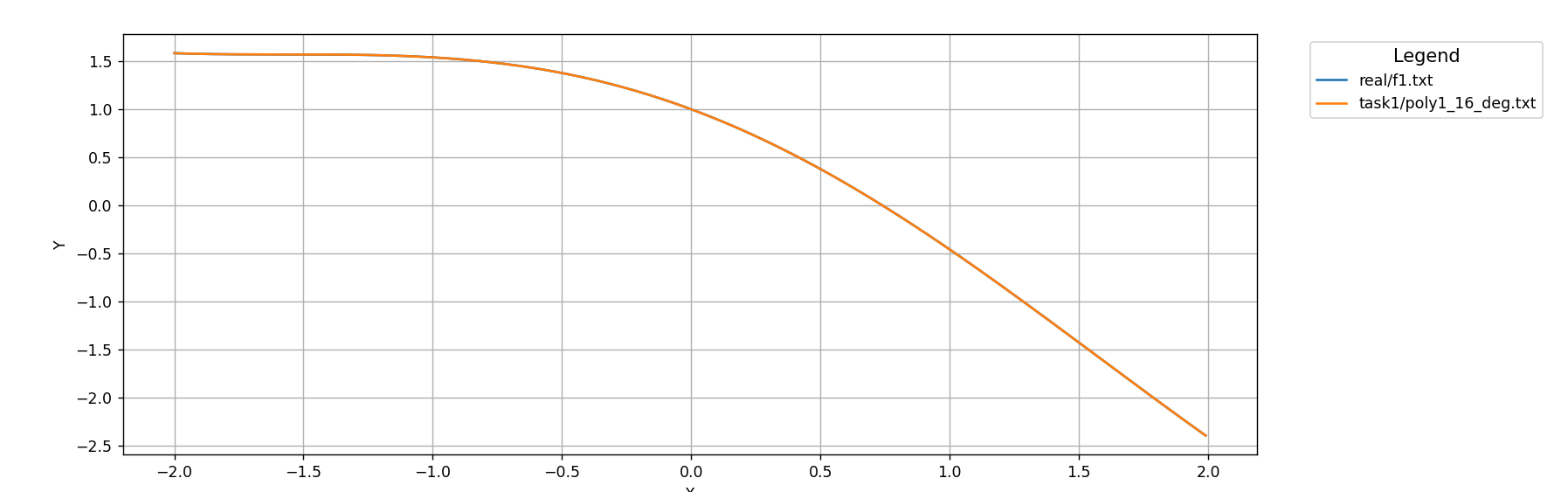
n=4



n=8



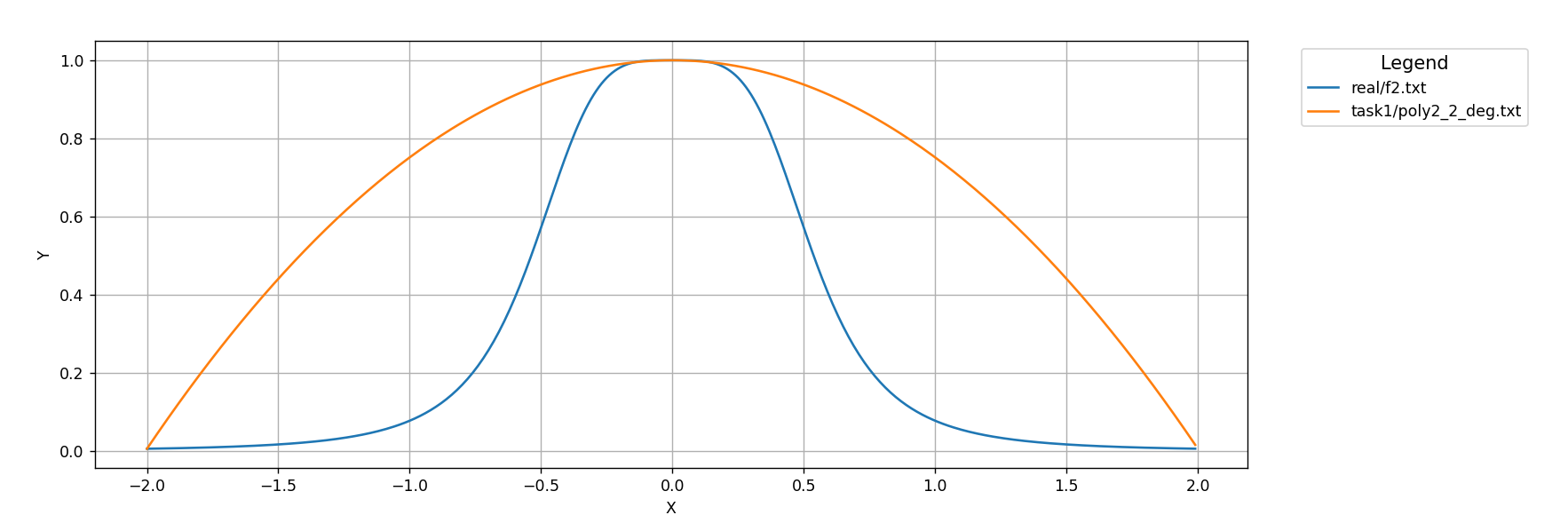
n=16



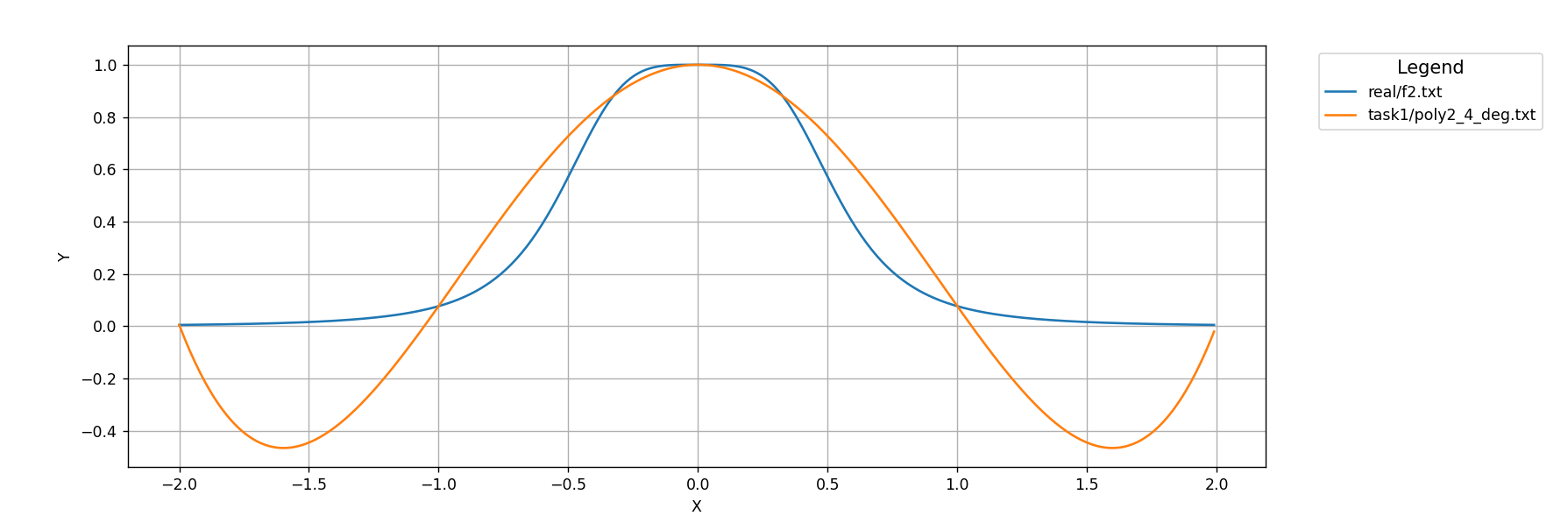


Analytical representation of the 2nd-degree polynomial: 0.00518135 + 0.497409 \* (x - -2) + -0.248705 \* (x - -2) \* (x - 0)

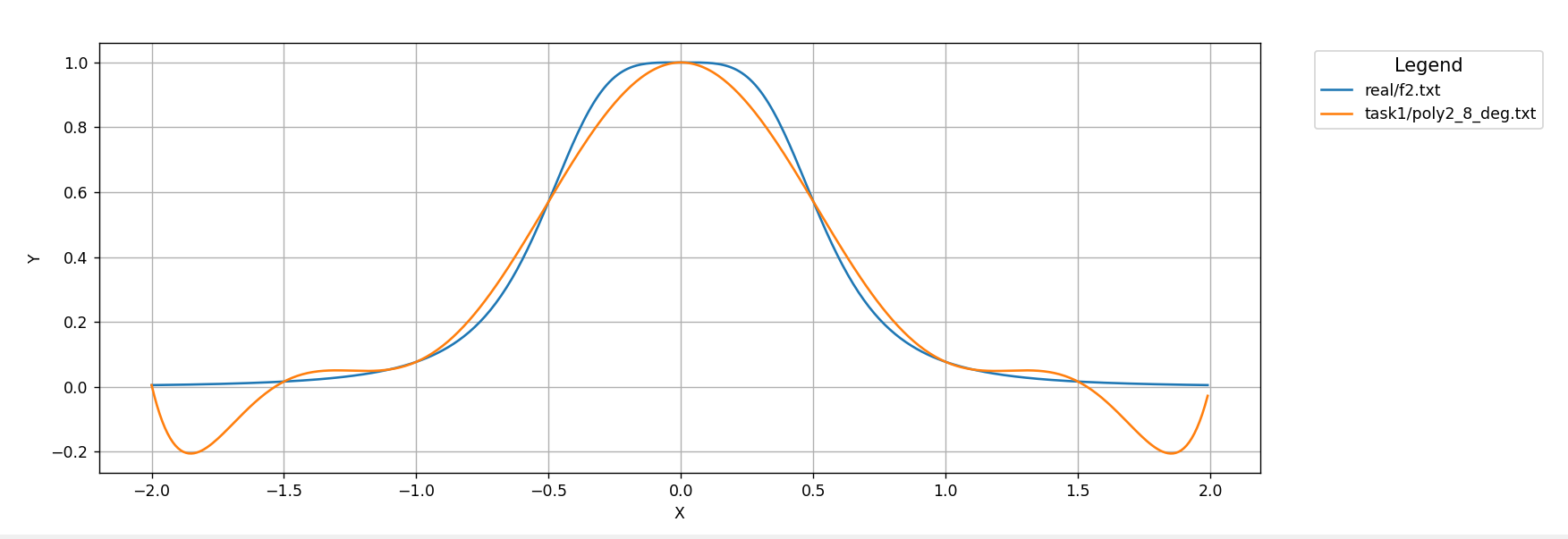
n=2



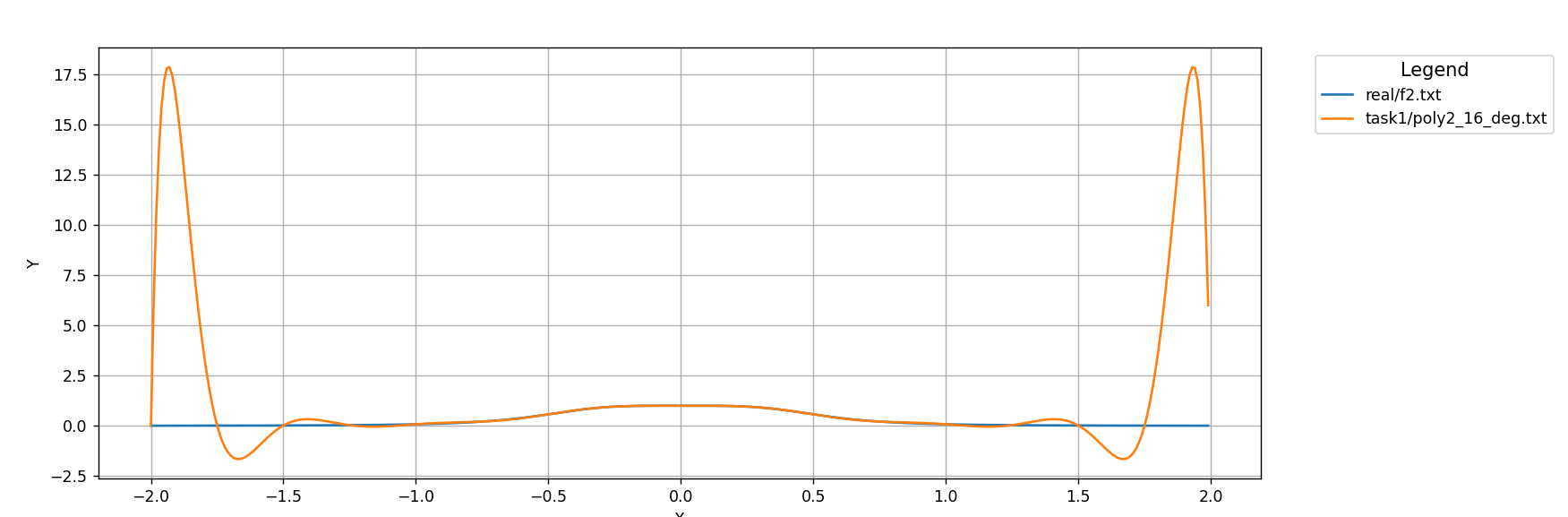
n=4



n=8



n=16

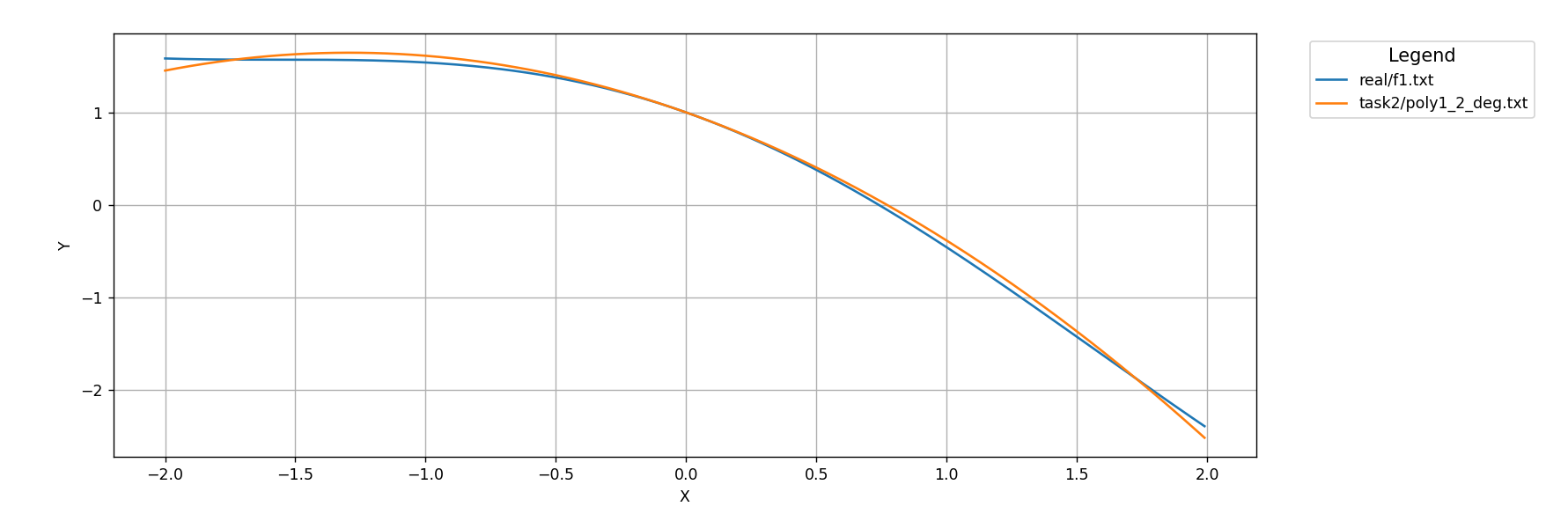


Задание 2

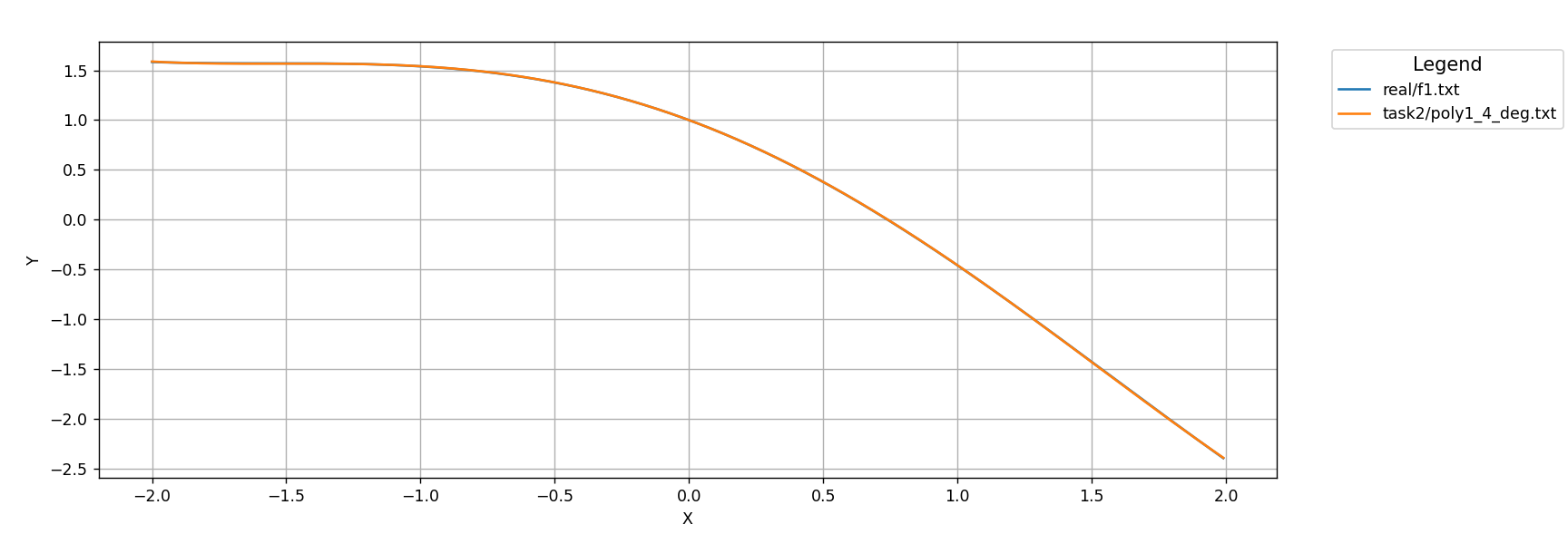


Analytical representation of the 2nd-degree polynomial: -1.89261 + -1.67005 \* (x - 1.73205) + -0.386852 \* (x - 1.73205) \* (x - 1.22465e-16)

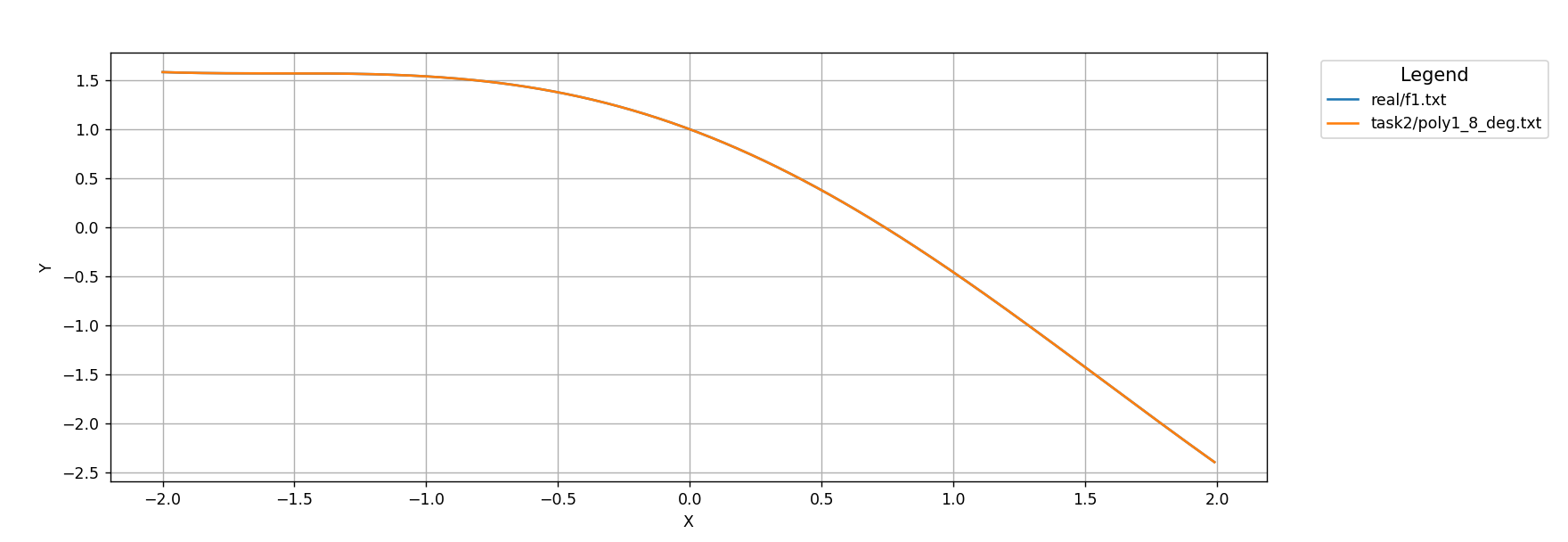
n=2



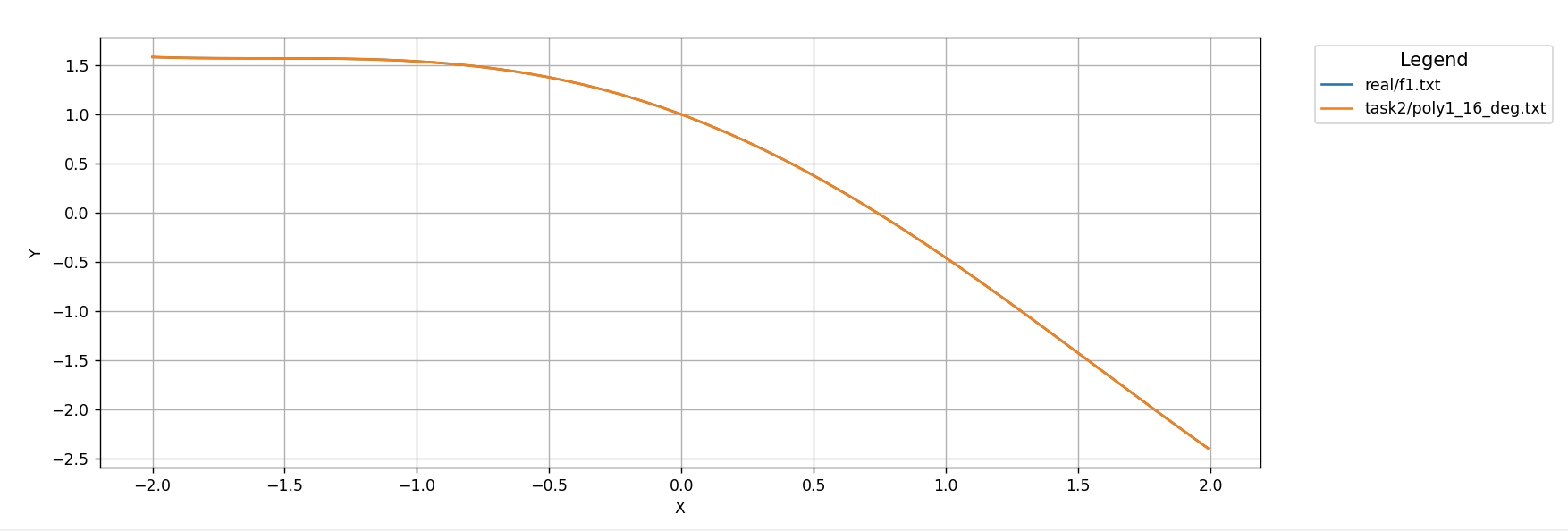
n=4



n=8



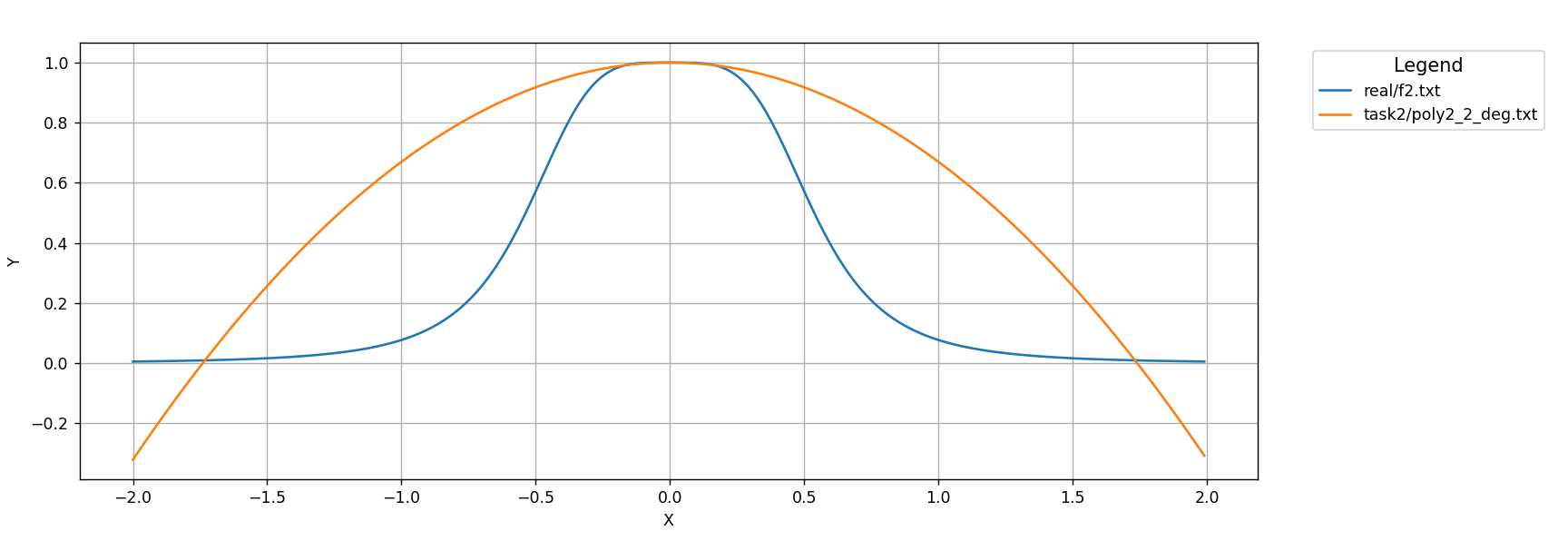
n=16



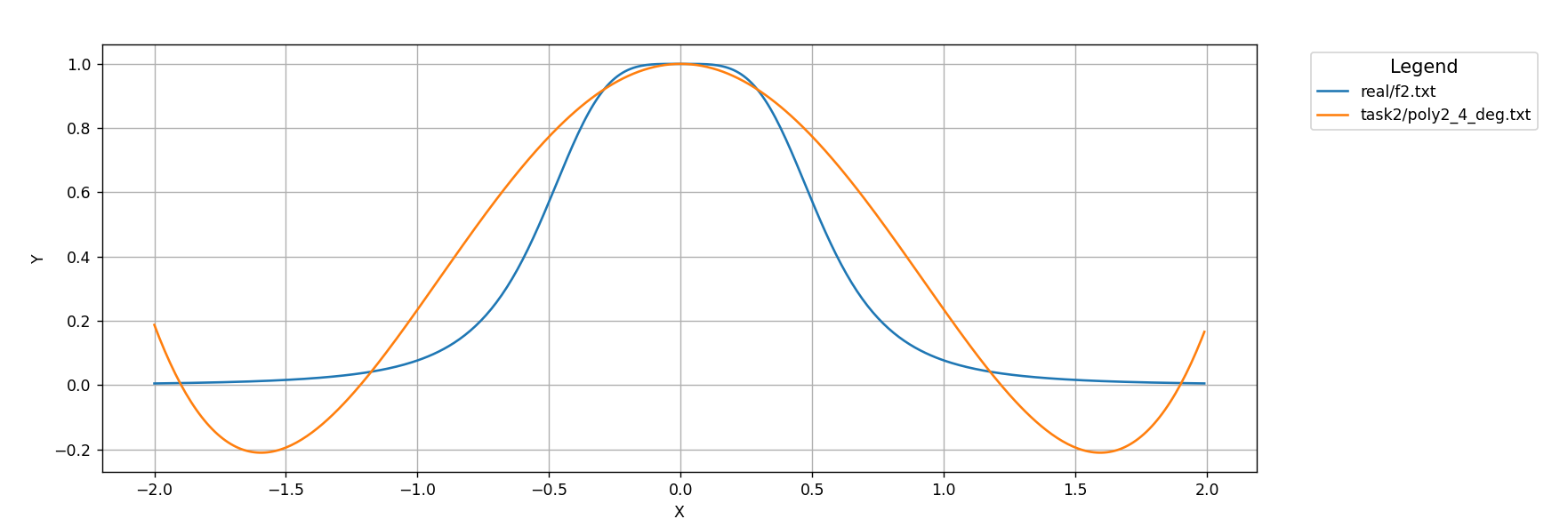


Analytical representation of the 2nd-degree polynomial: 0.00917431 + -0.572053 \* (x - 1.73205) + -0.330275 \* (x - 1.73205) \* (x - 1.22465e-16)

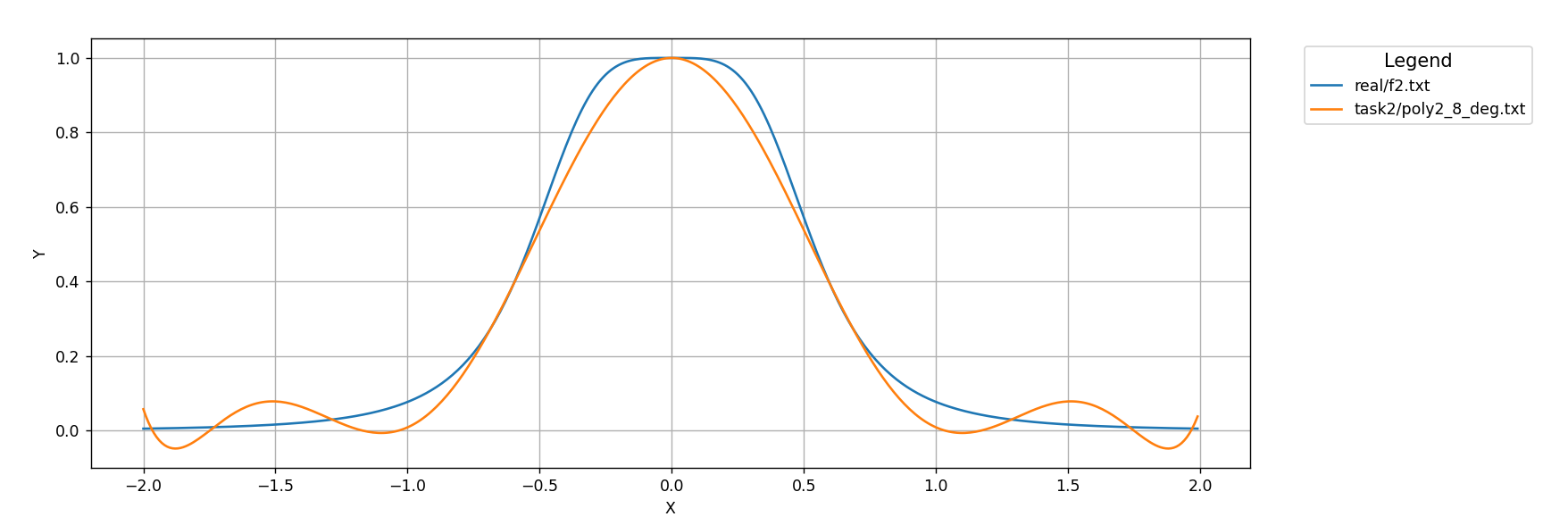
n=2



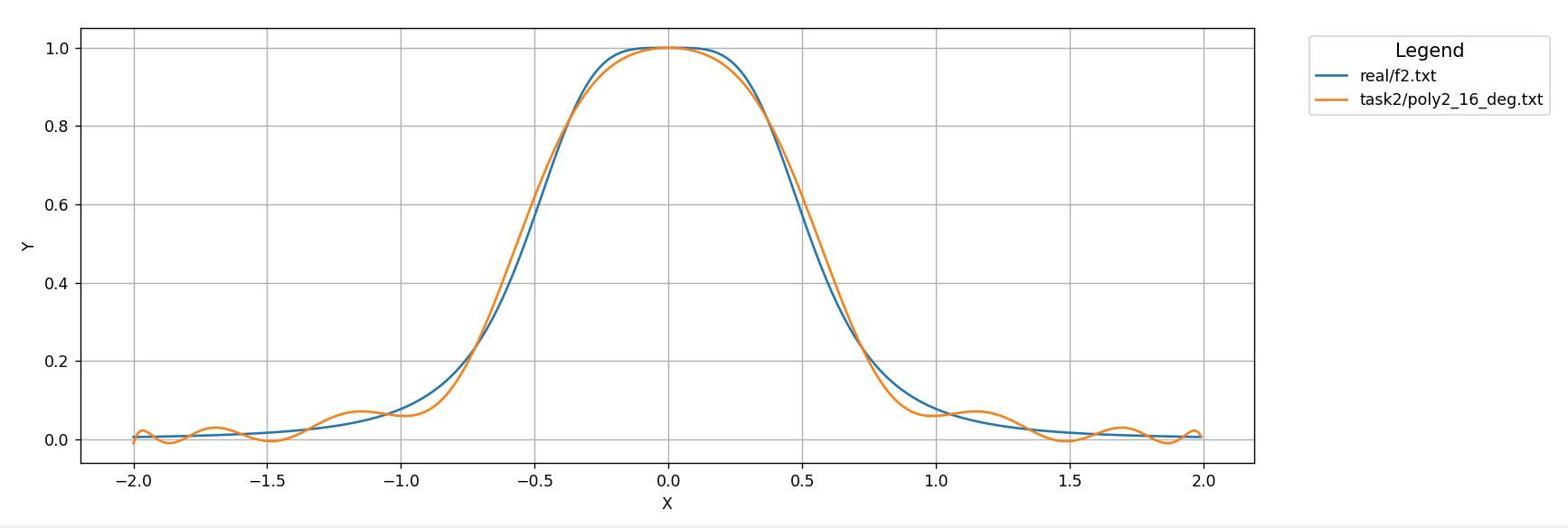
n=4



n=8



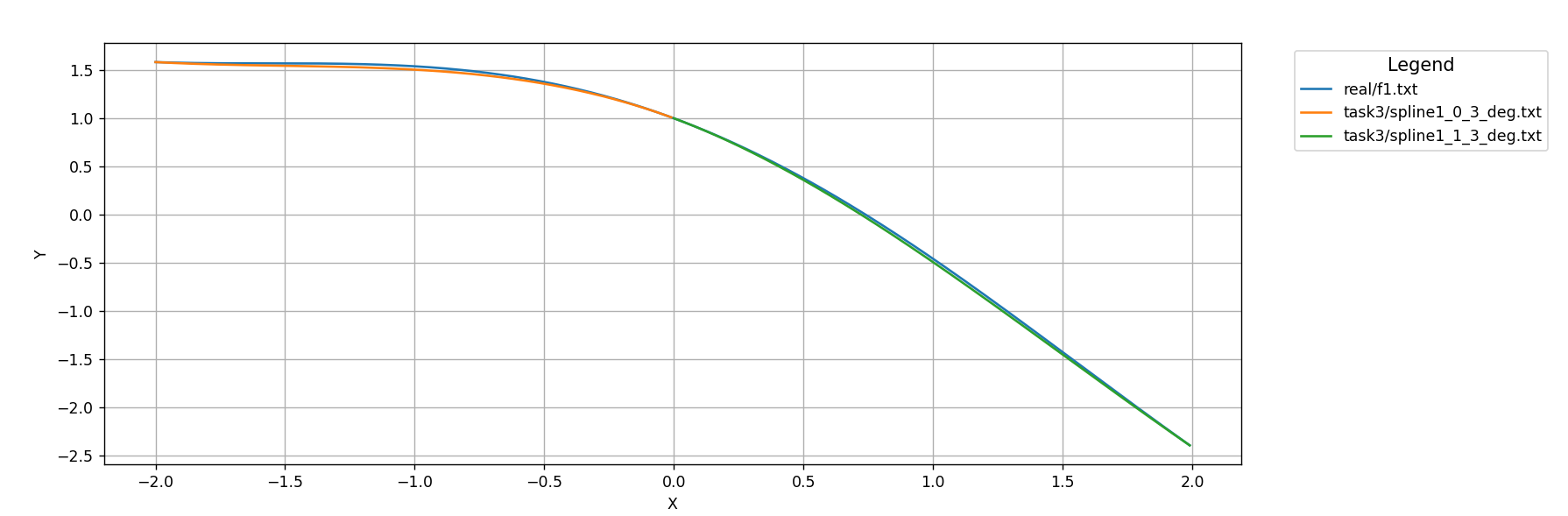
n=16



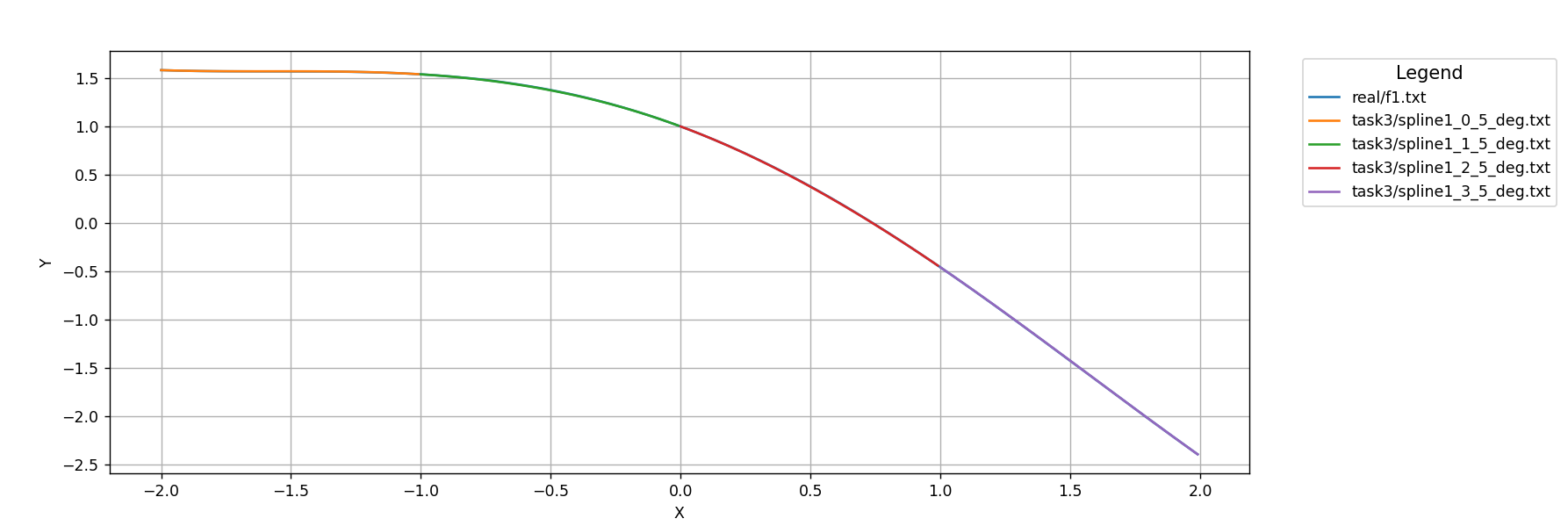
Задание 3



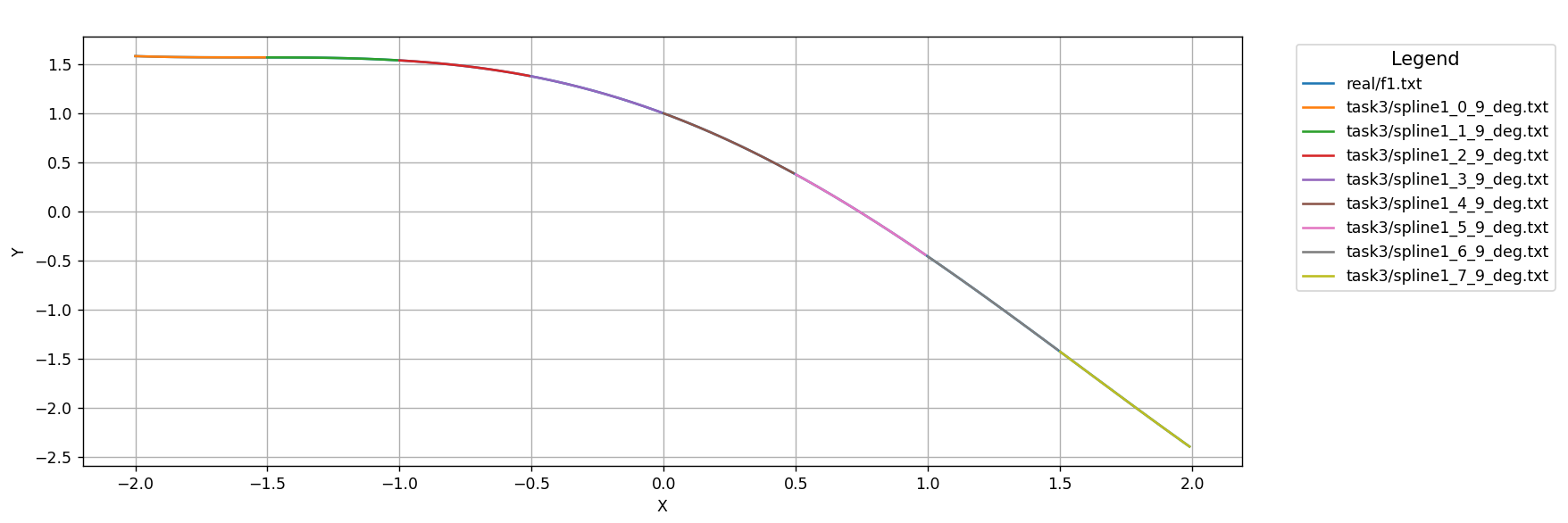
n=2



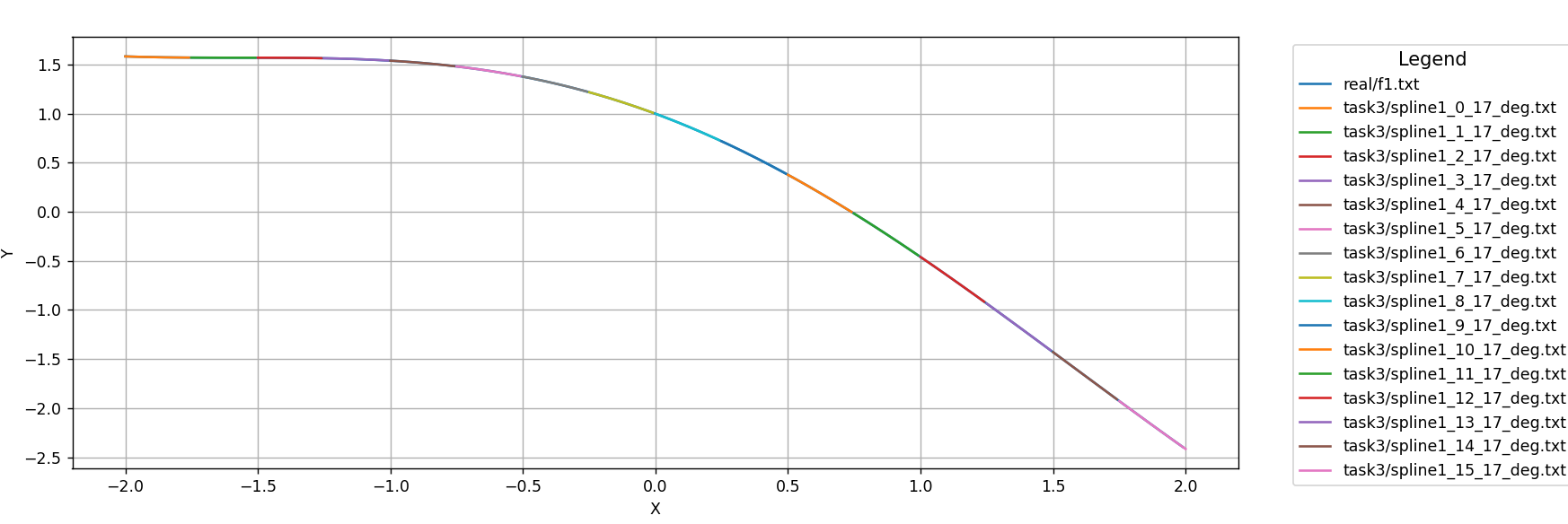
n=4



n=8

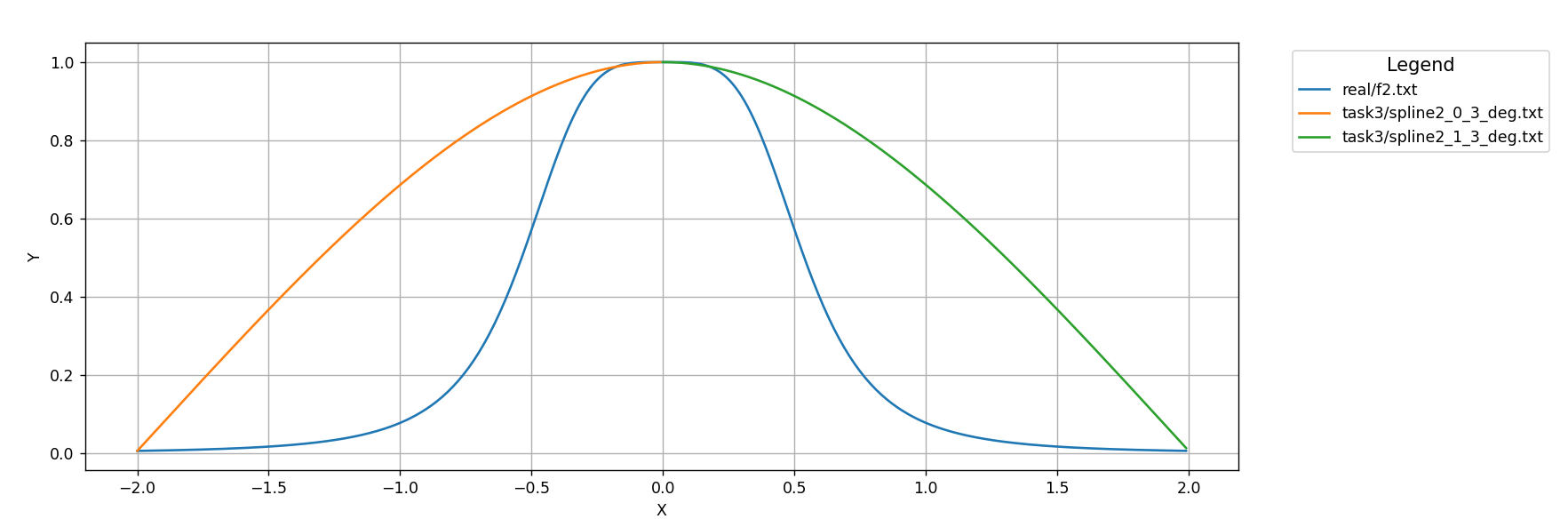


n=16

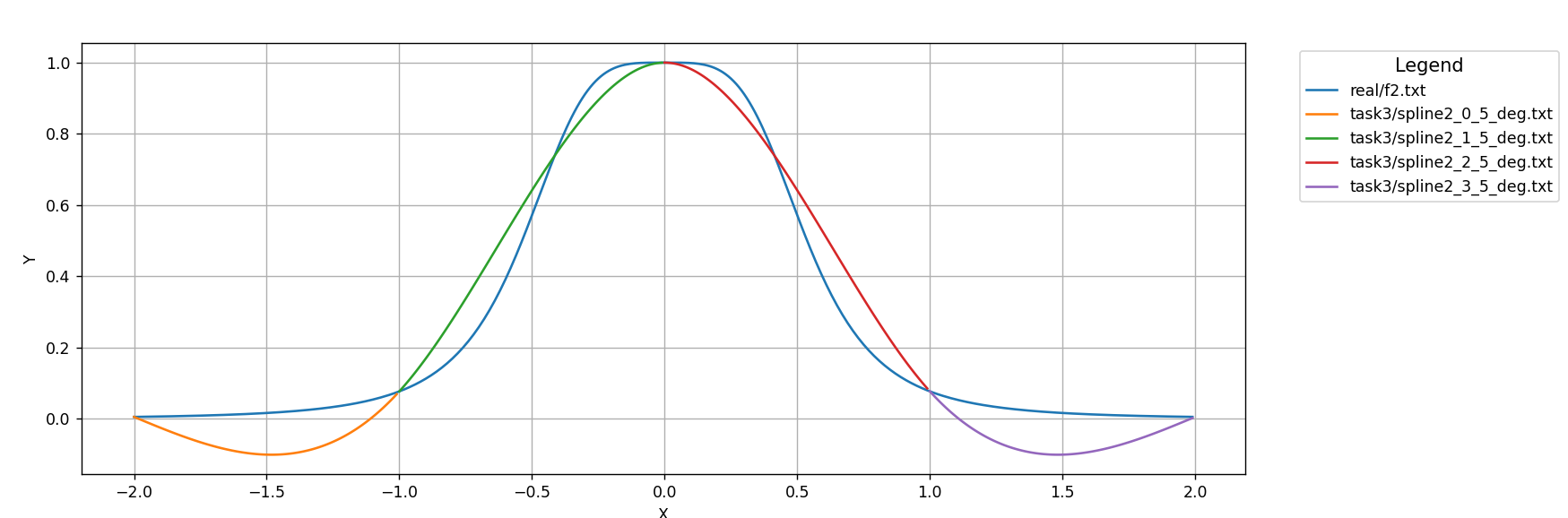




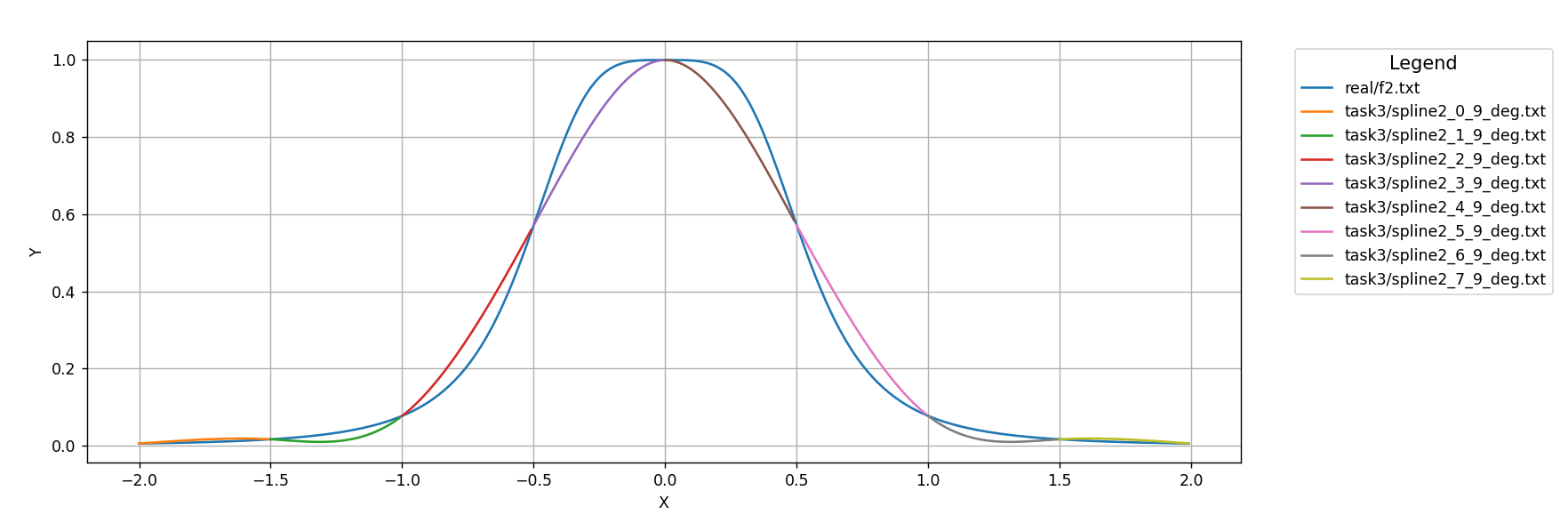
n=2



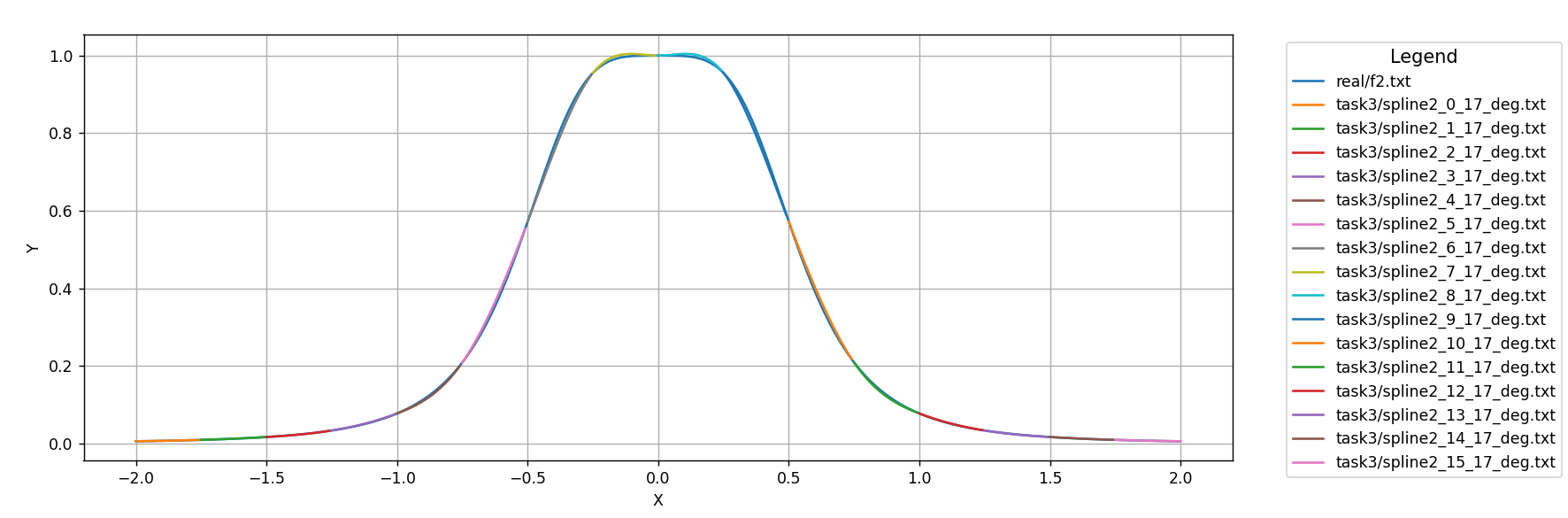
n=4



n=8



n=16



Задание 4

Сравнивая f1 и f2 можно заметить, что все методы приближения работают лучше для f1, так как на отрезке [-2,2] она является более «простой». Сравнивая методы приближения лучше всего сработал метод из Задания 3. Также стоит отметить тот факт, что при увелечении n увеличивается точность приближения. Касаемо выбора построения сетки узлов сложно однозначно определить лучший, но не исключено преимущество какого-либо из построений на функциях, отличных от f1 и f2.