

Laboratorium z Metod Numerycznych

Interpolacja.

Data zajęć 14.01.2017r
Data wykonania 27.01.2017r

mgr inż. Wojciech Łabaj

Sekcja 1 grupa 1
Skroboł Bartłomiej
[bartskr440@student.polsl](mailto:bartskr440@student.polsl.pl)

Treść zadania:

Napisać program wyznaczający wartość wielomianu interpolacyjnego Newtona w punktach leżących w przedziale $\langle a, b \rangle$ dla funkcji interpolowanej $f(x)$.

Funkcja interpolowana: $f(x) = |\cos(x) \cdot x|$

Przyjąć $(n + 1)$ węzłów dla $n = 7, 8, 15, 16$

a) równoodległych

b) dobranych optymalnie

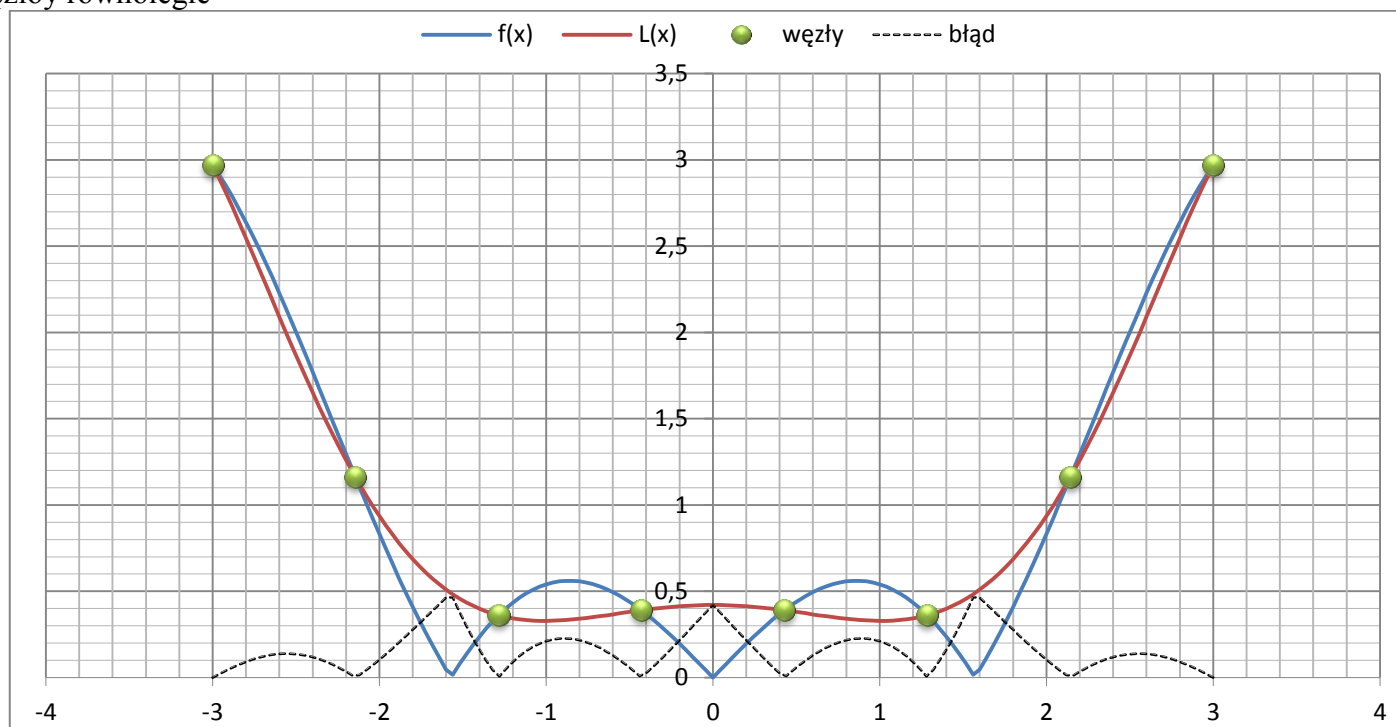
Przedziały

a) $x \in \langle -3; 3 \rangle$

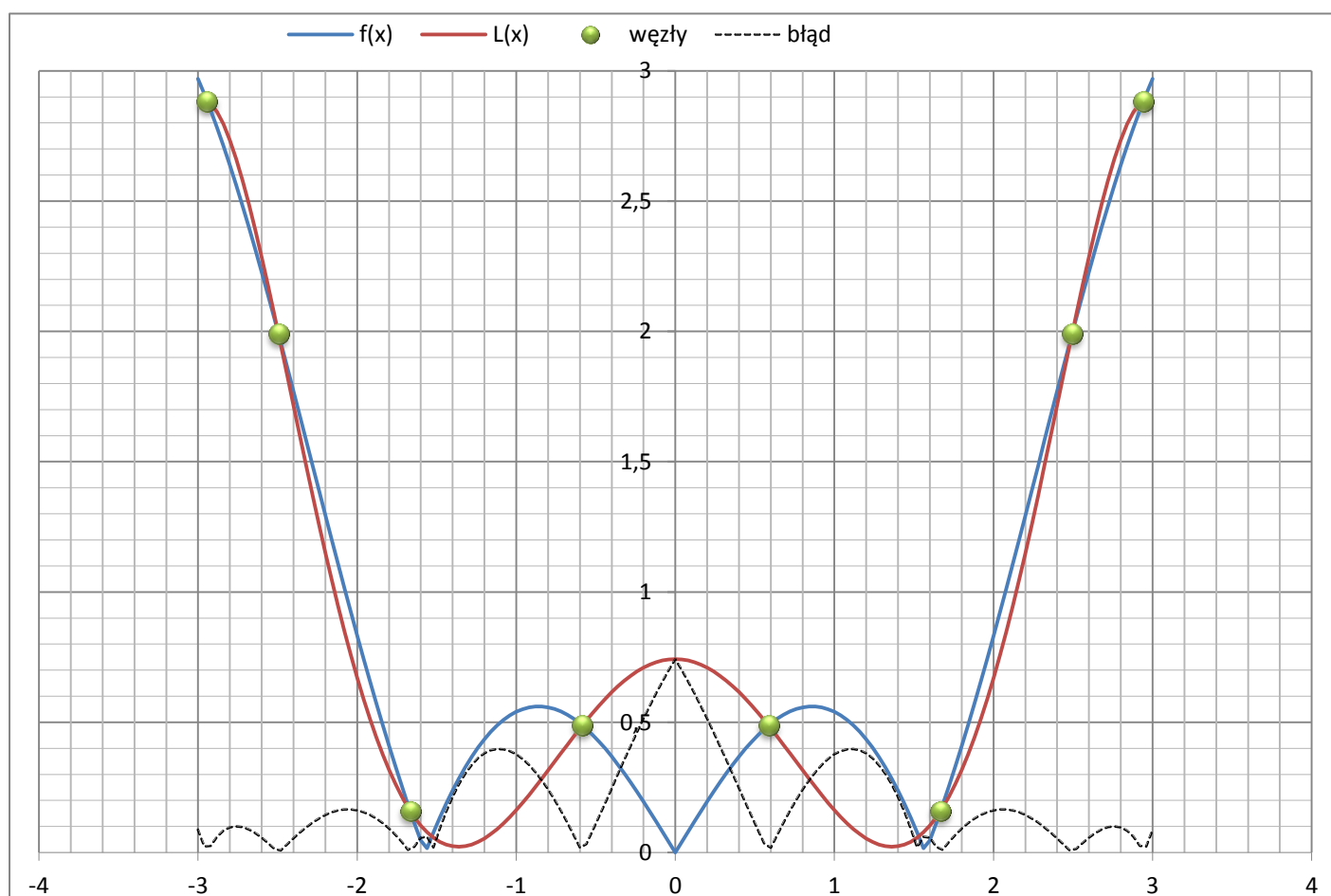
b) $x \in \langle -6; 6 \rangle$

Wyniki: $\langle -3, 3 \rangle$ $n=7$

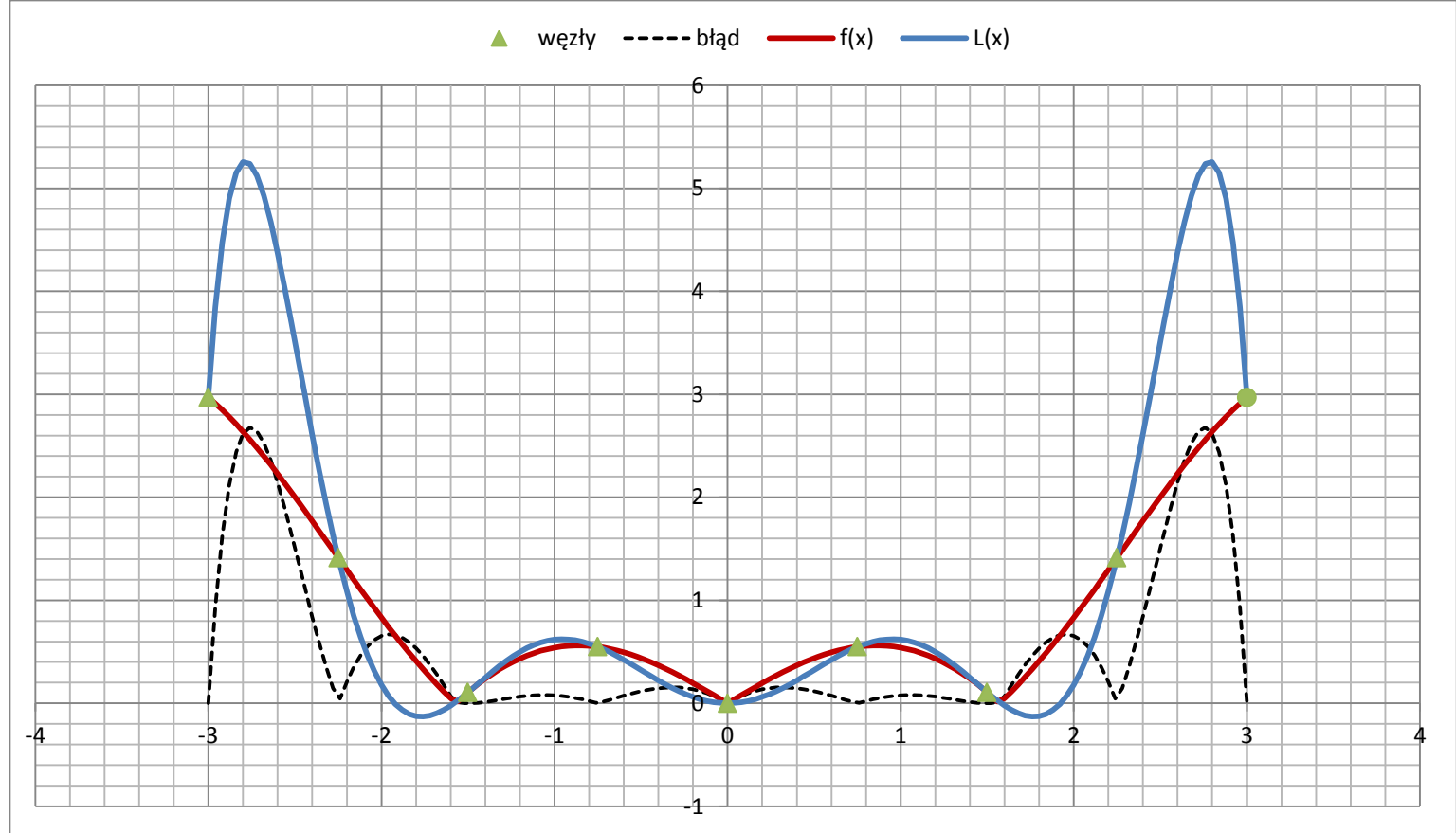
Węzły równoległe



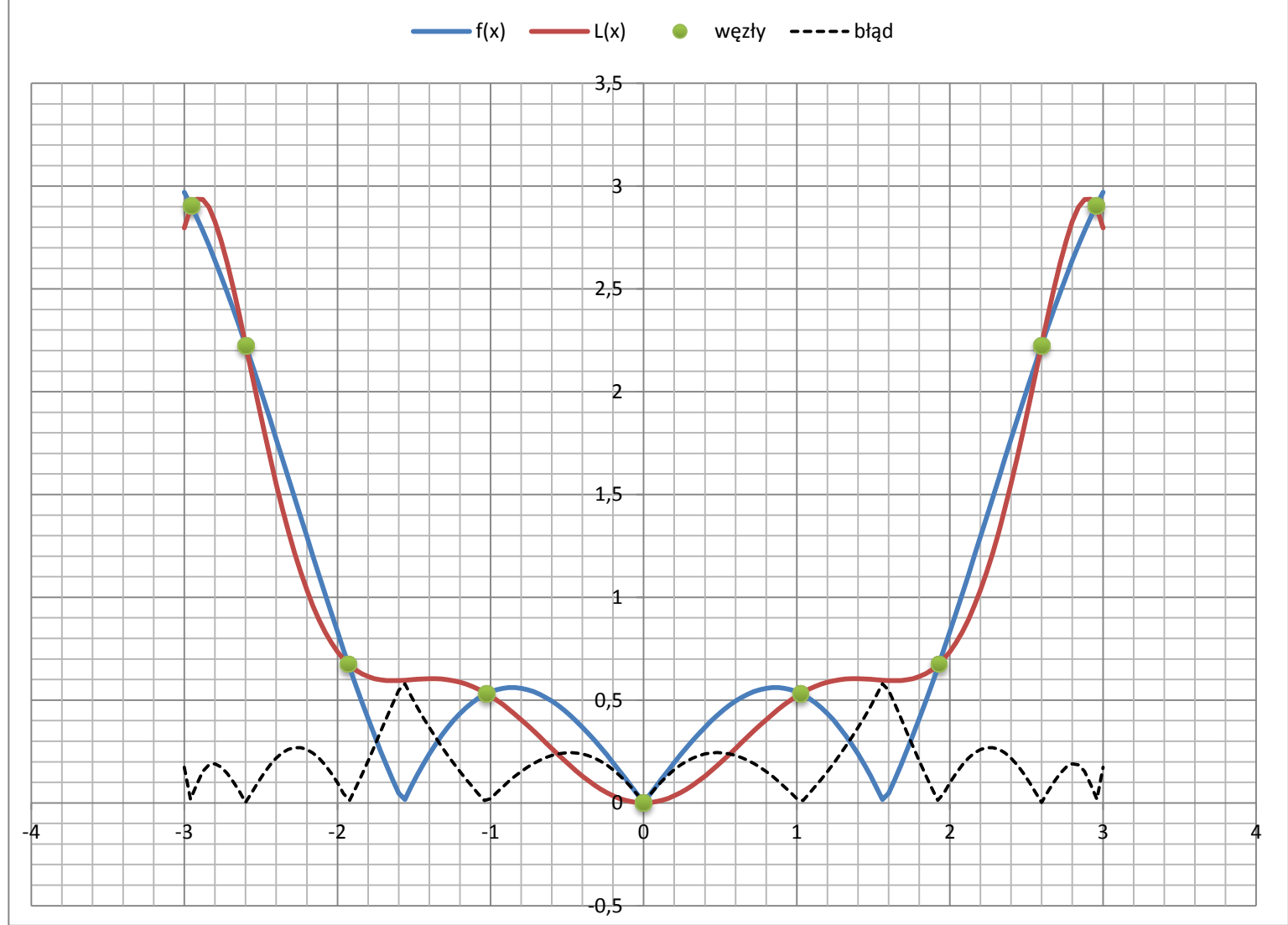
węzły optymalne



$\langle -3, 3 \rangle$ $n=8$
Węzły równoległe

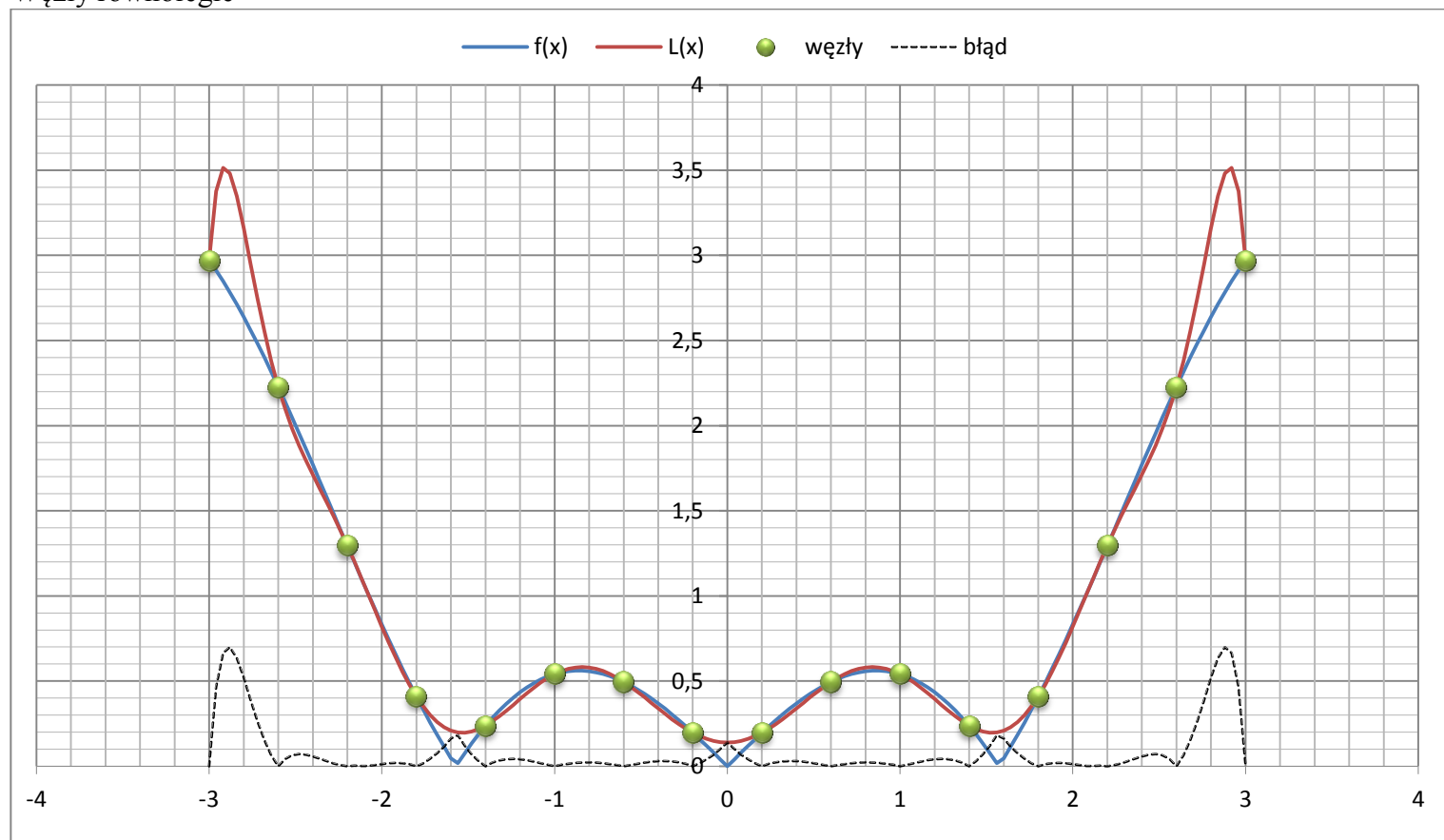


Węzły optymalne

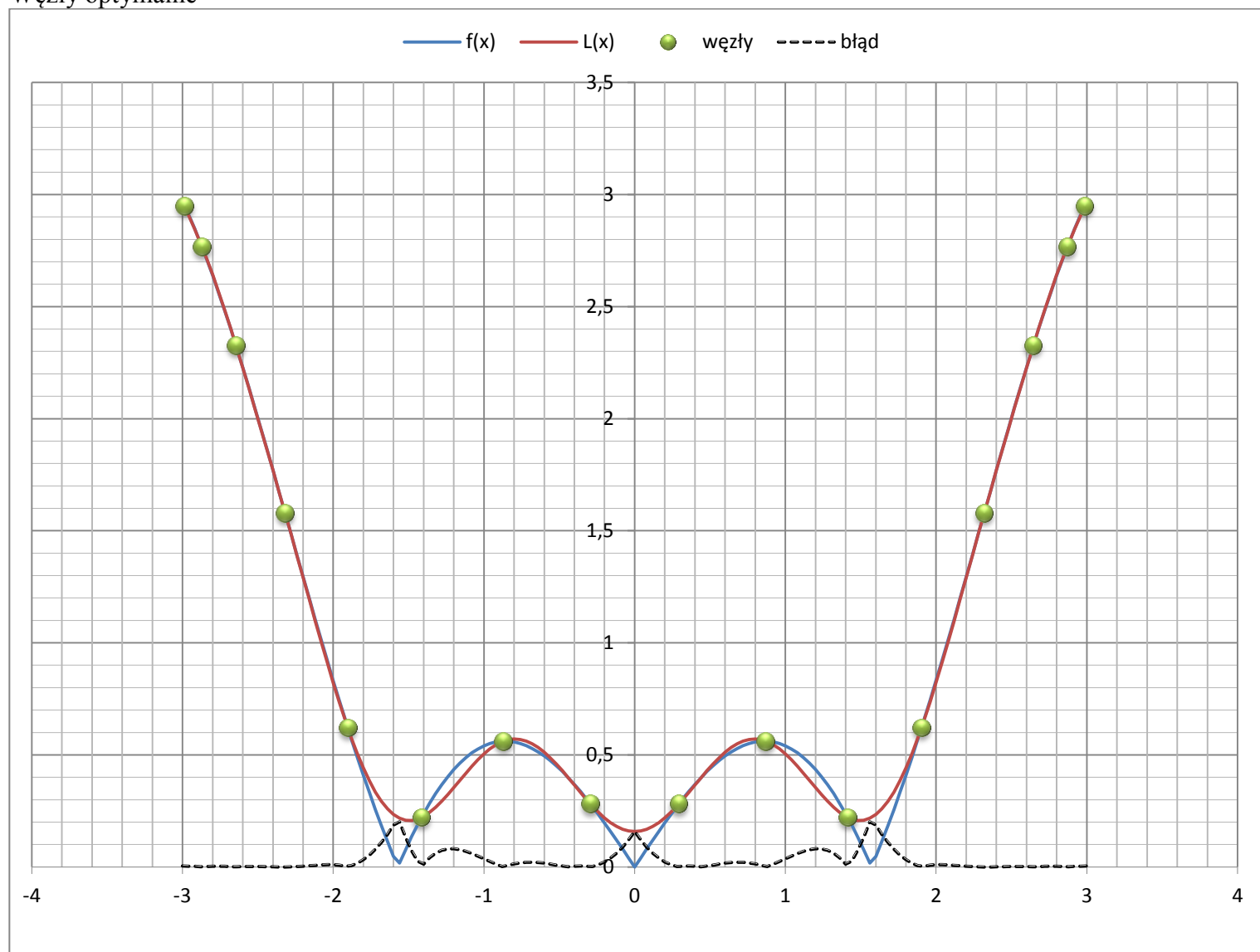


$\langle -3, 3 \rangle$ $n=15$

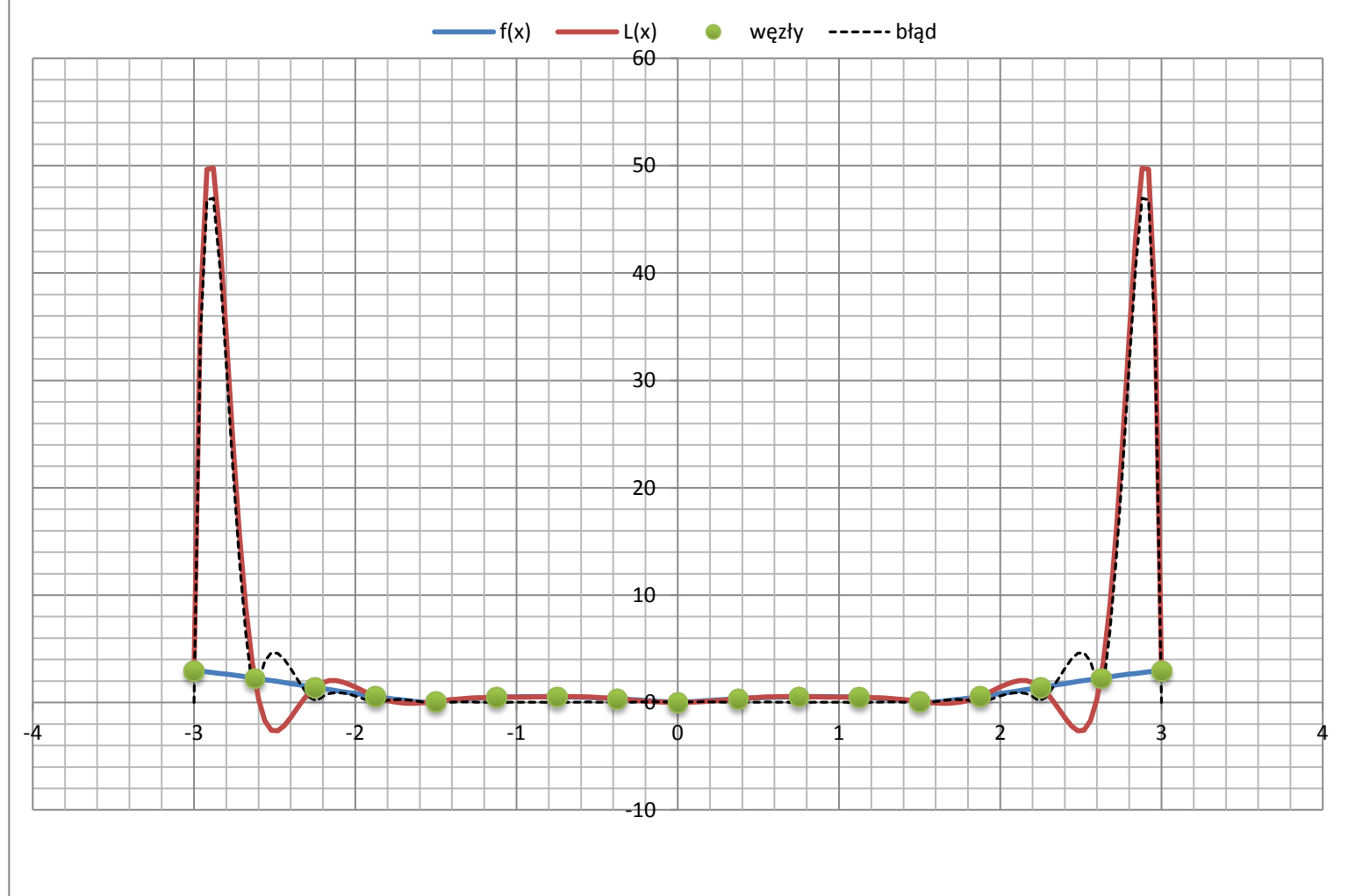
Węzły równoległe



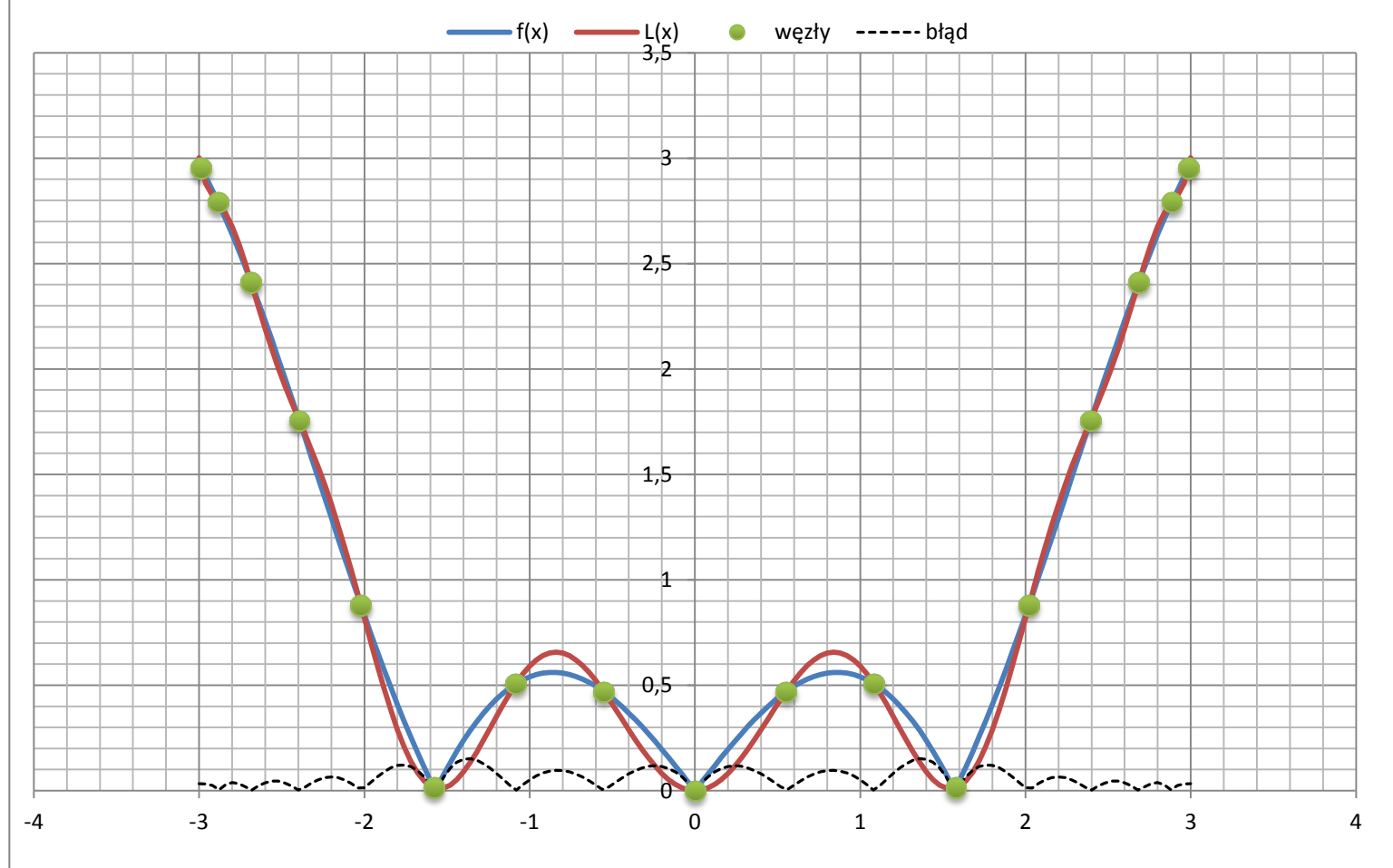
Węzły optymalne



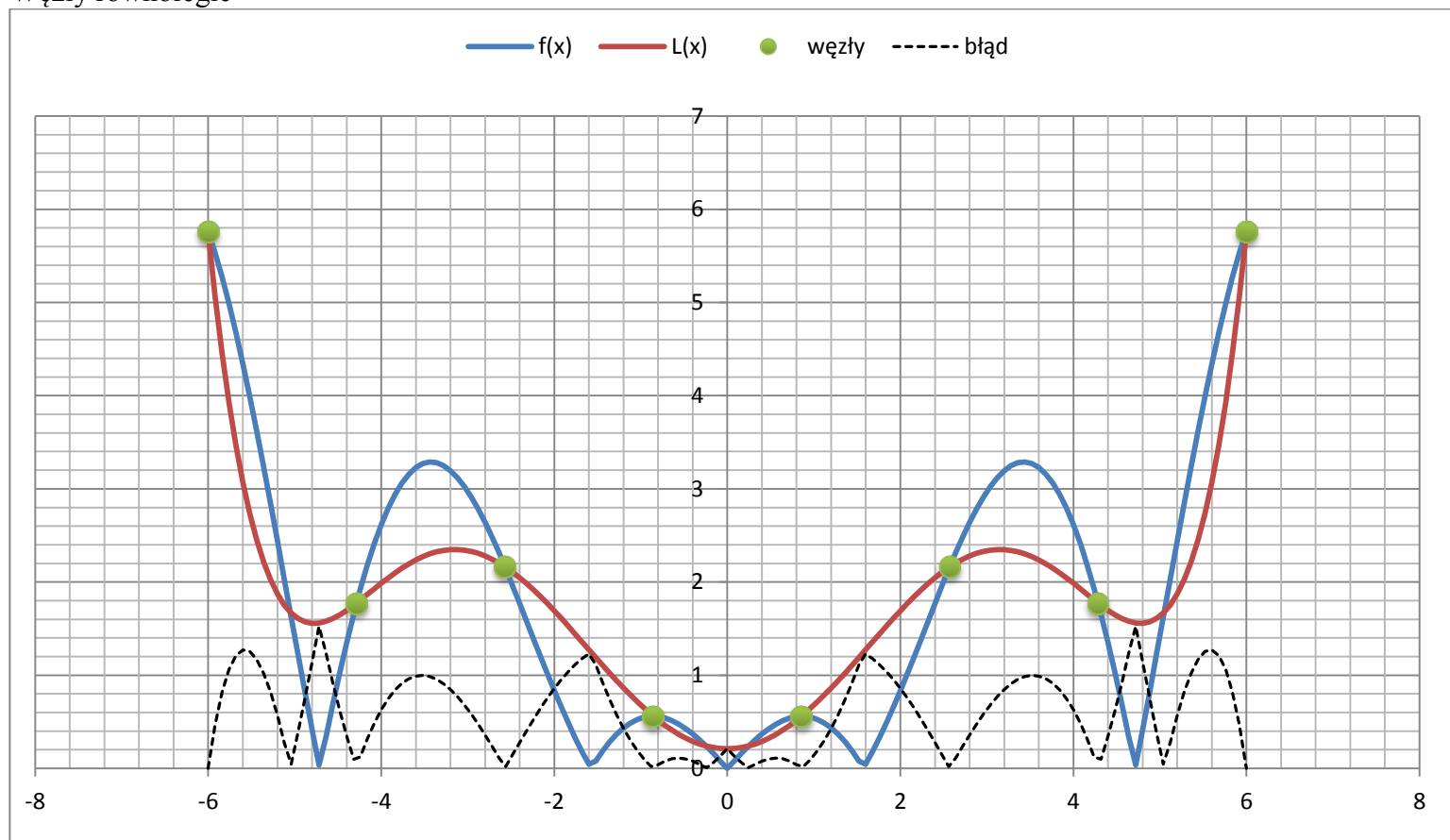
$\langle -3, 3 \rangle$ $n=16$
Węzły równoległe



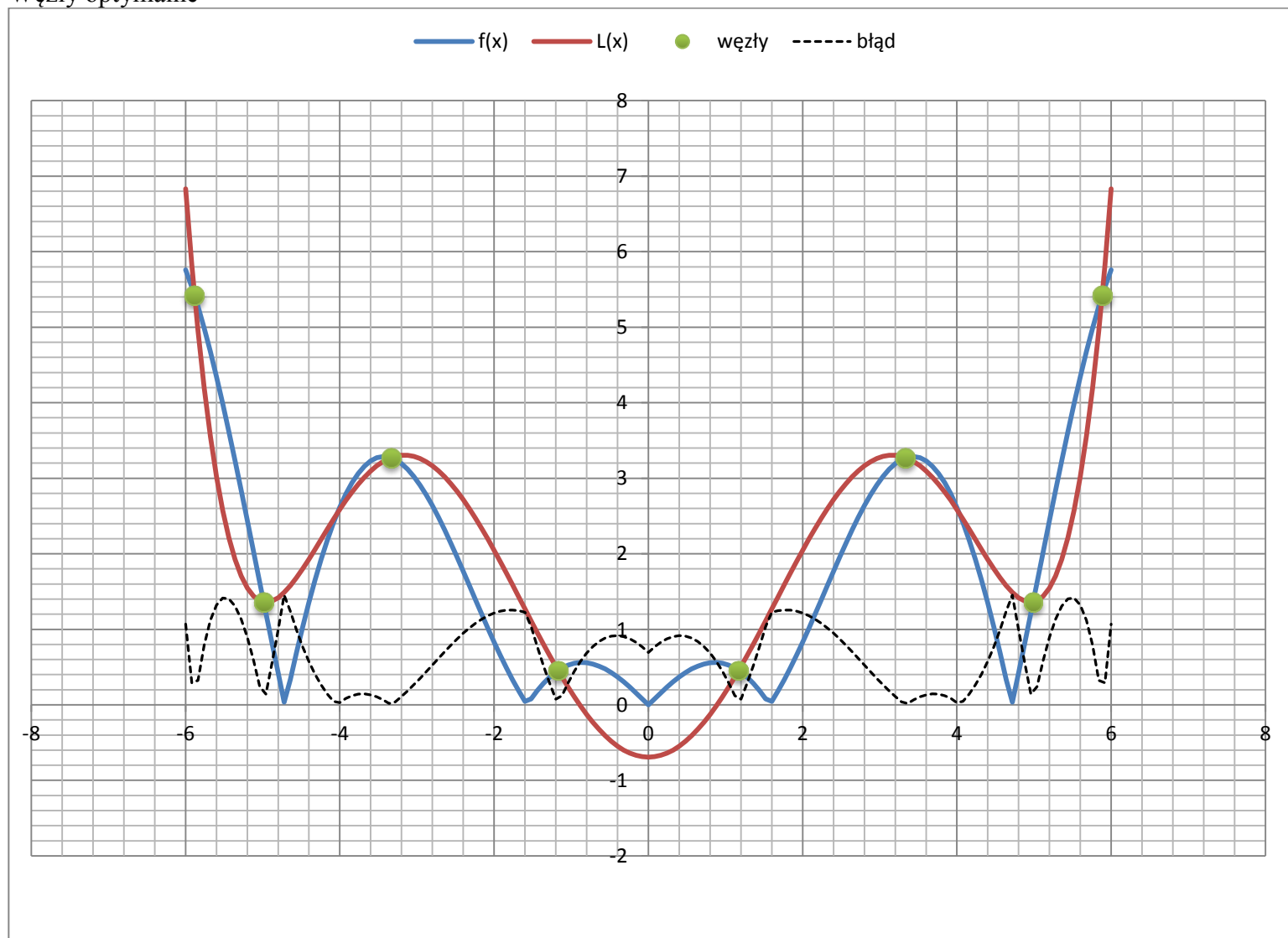
Węzły optymalne



$\langle -6, 6 \rangle$ $n=7$
Węzły równoległe

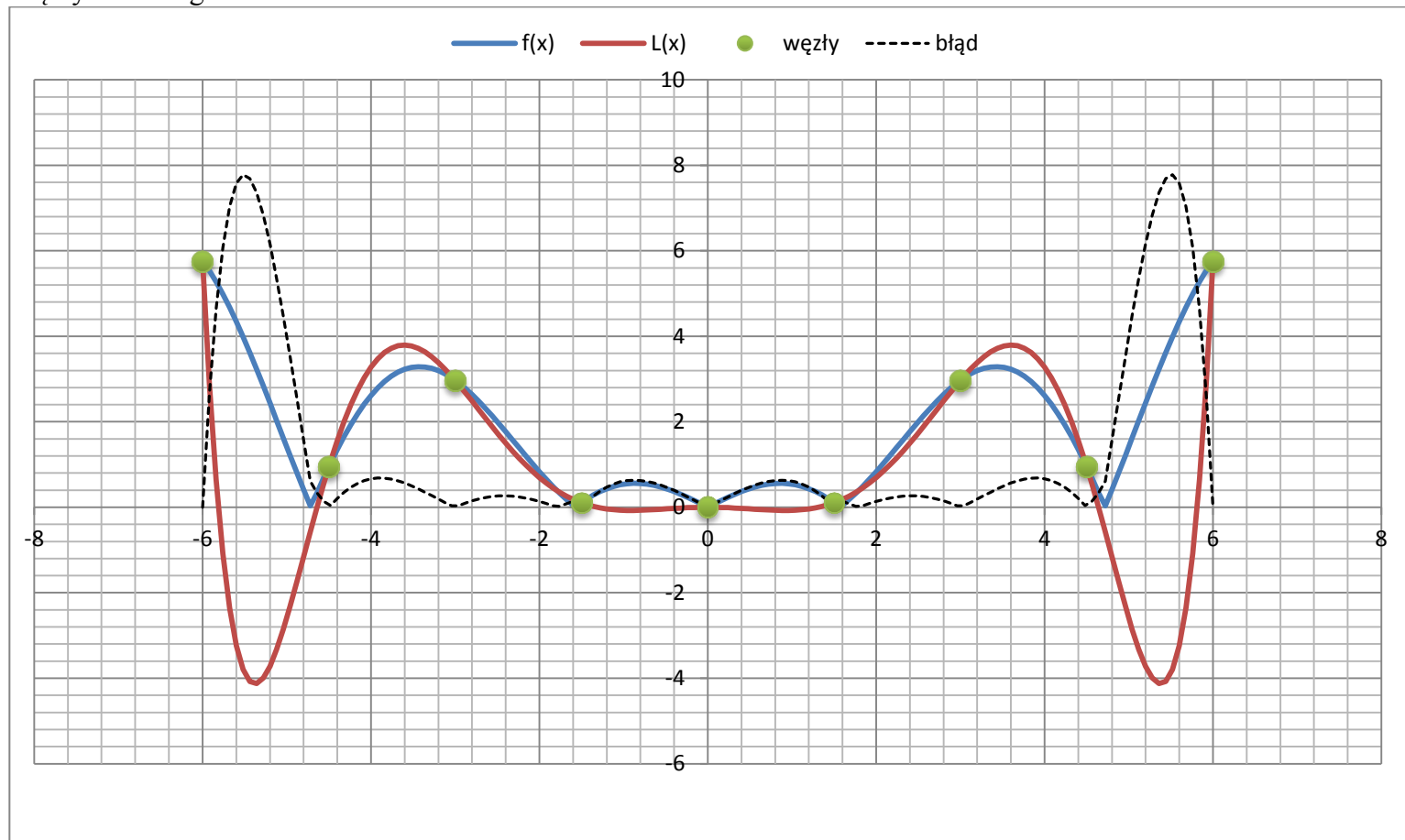


Węzły optymalne

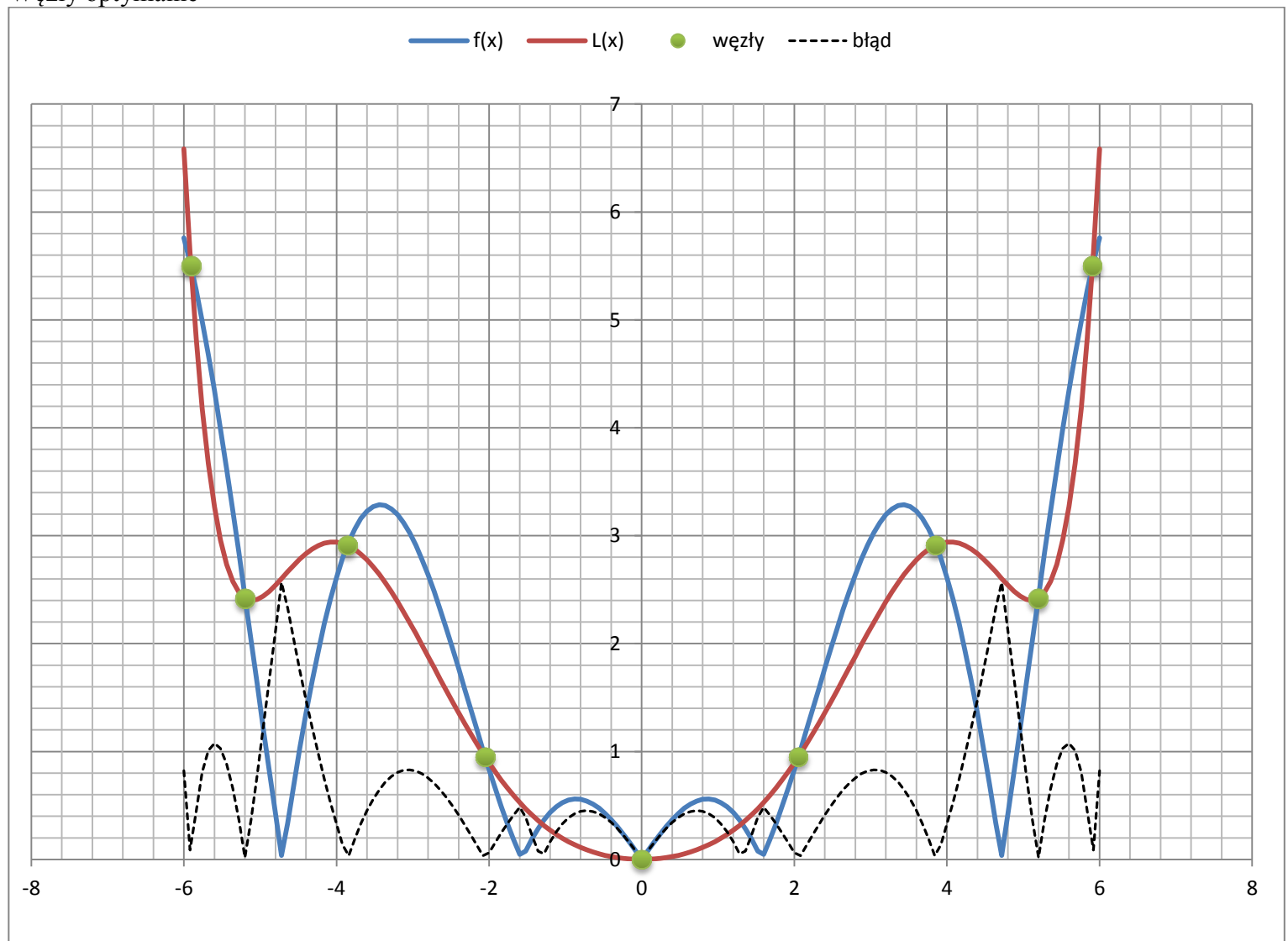


$\langle -6, 6 \rangle$ $n=8$

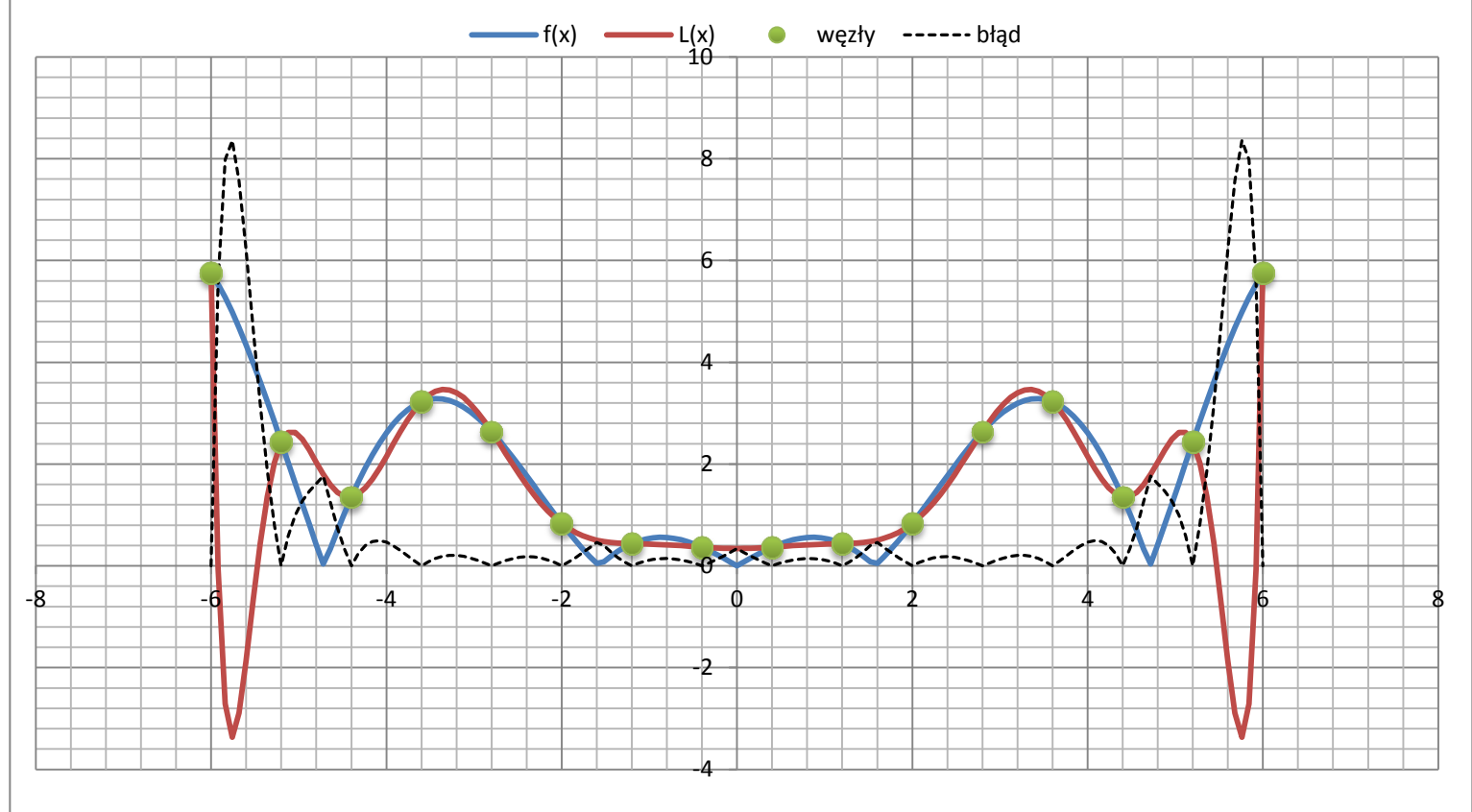
Węzły równoległe



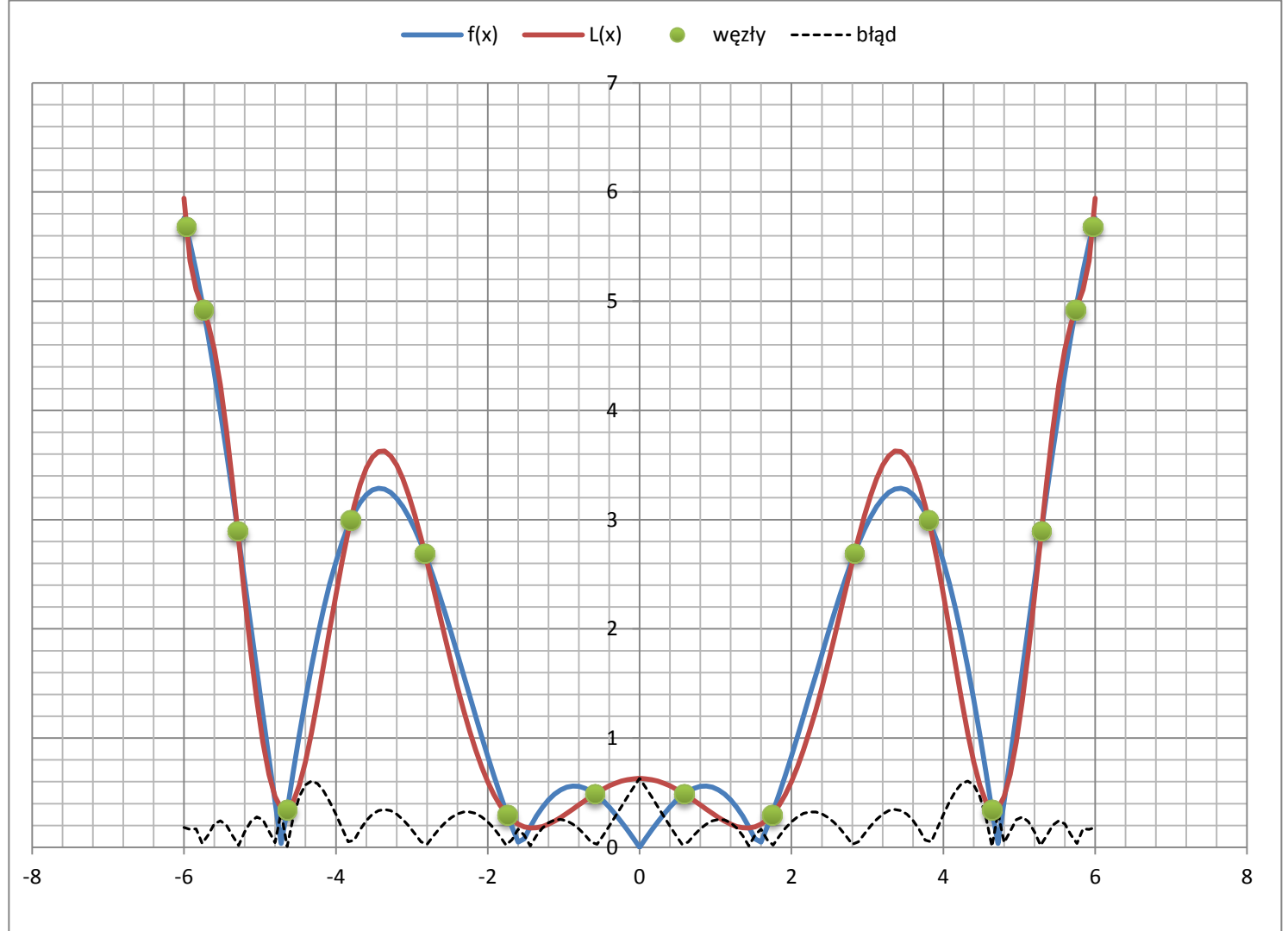
Węzły optymalne



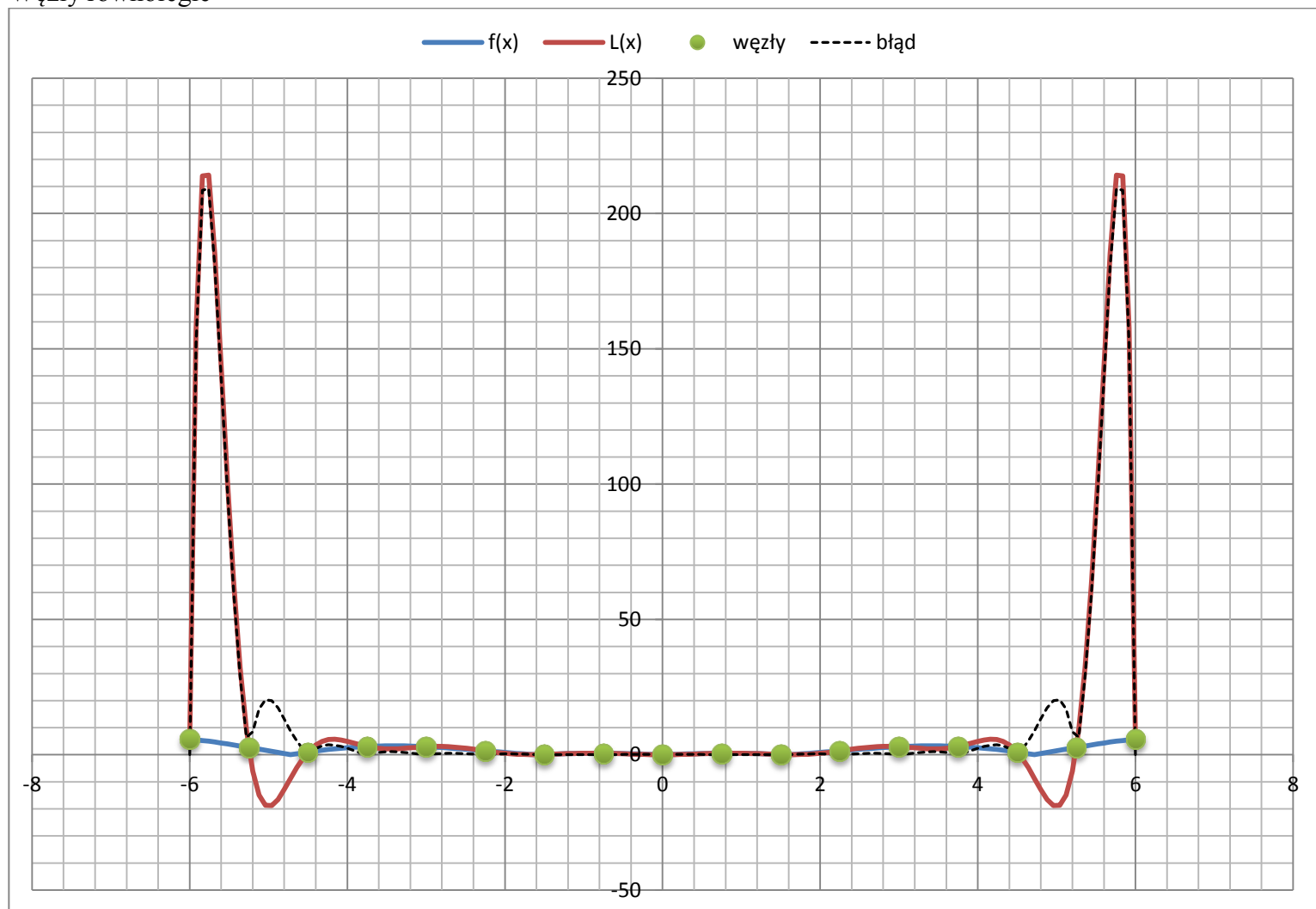
$\langle -6, 6 \rangle$ $n=15$
Węzły równoległe



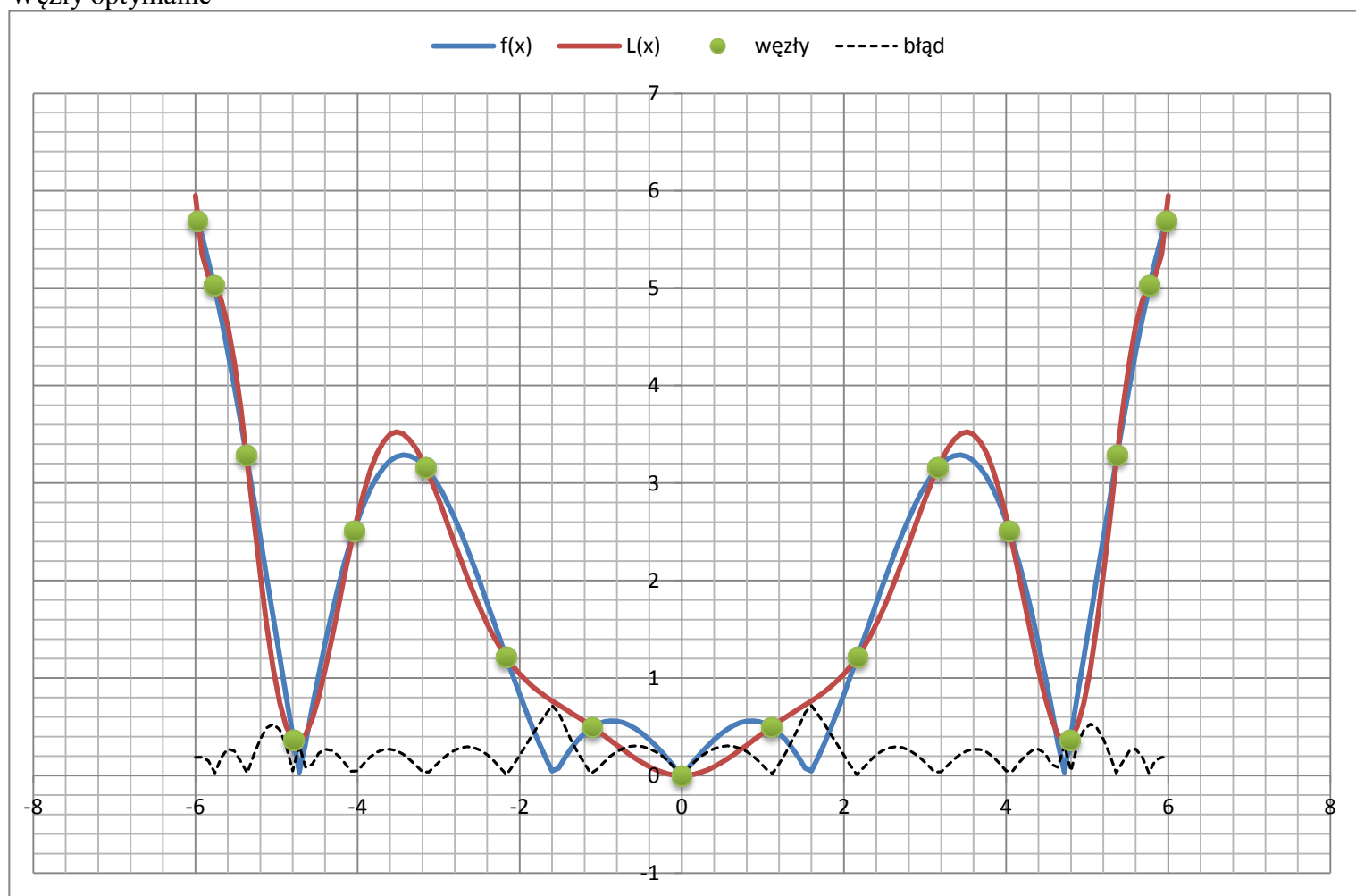
Węzły optymalne



$\langle -6, 6 \rangle$ $n=16$
Węzły równoległe



Węzły optymalne



Wnioski

Zauważyłem że podczas zwiększania ilości węzłów interpolacji w przypadku węzłów równooddalonych dostajemy lepsze przybliżenie funkcji interpolowanej w środku przedziału a na końcach coraz gorsze, szczególnie gdy liczba węzłów n jest parzysta. W przypadku węzłów dobranych optymalnie zwiększanie liczby węzłów wpływa na zmniejszanie się błędu, jednak w każdym przypadku przybliżenie było lepsze na krańcach przedziału, ponieważ węzły interpolacji zagęszczały się na końcach przedziałów.

Listing

„Interpolation.h”

```
#pragma once
#include <vector>
#include <algorithm>

const double PI = 3.141592653589793238463;

void parallelNodes(size_t nodesAmount, double downRange, double upRange, std::vector<double>& x);
void chebyszewNodes(size_t nodesAmount, double downRange, double upRange, std::vector<double>& x);

///blad bezwzględny
double absoluteError(double fx, double Lx);

class NewtonInterpolation
{
public:
    NewtonInterpolation(const std::vector<double>& x, const std::vector<double>& y);
    NewtonInterpolation(const double* x, const double* y, size_t n);
    ~NewtonInterpolation();

    void setArrays(const std::vector<double>& x, const std::vector<double>& y);
    void setArrays(const double* x, const double* y, size_t n);
    double compute(double x);
private:
    std::vector<double> mX;
    std::vector<double> mY;
    std::vector<double> mAi;
    size_t mNodes;
    ///wielomian czynnikowy wi
    ///stopnia degree
    double factorPolynomialial(size_t degree, double x);
    ///iloraz roznicowy ai
    ///rzędu n
    double differentialQuotient(size_t n);
    ///tworzy wektor ilorazow
    void differentialQuotient();
};
```

„Interpolation.cpp”

```

#include "Interpolation.h"
#include <iostream>

void parallelNodes(size_t nodesAmount, double downRange, double upRange, std::vector<double>& x)
{
    x.clear();
    double h = (upRange - downRange) / nodesAmount;
    for (size_t i = 0; i <= nodesAmount; ++i)
    {
        x.push_back(downRange + i*h);
    }
}

void czebyszewNodes(size_t nodesAmount, double downRange, double upRange, std::vector<double>& x)
{
    x.clear();
    double temp = 0.0;
    double tmp = 0.0;
    for (size_t i = 0; i <= nodesAmount; ++i)
    {
        tmp = (2 * (double)i + 1);
        tmp /= (2 * nodesAmount + 2);
        tmp *= PI;
        tmp = cos(tmp);
        temp = (upRange - downRange) / 2;
        temp *= cos(((2 * (double)i + 1) / (2 * nodesAmount + 2))*PI);
        //temp *= tmp;
        temp += (downRange + upRange) / 2;
        x.push_back(temp);
    }
}

double absoluteError(double fx, double Lx)
{
    return abs(fx - Lx);
}

NewtonInterpolation::NewtonInterpolation(const std::vector<double>& x, const std::vector<double>&
y):mX(x),mY(y)
{
    mNodes = mX.size()-1;
    differentialQuotient();
}

NewtonInterpolation::NewtonInterpolation(const double * x, const double * y, size_t n):mNodes(n-1)
{
    for (size_t i = 0; i < n; ++i)
    {
        mX.push_back(x[i]);
        mY.push_back(y[i]);
    }
    differentialQuotient();
}

NewtonInterpolation::~~NewtonInterpolation()
{
}

double NewtonInterpolation::factorPolynomialial(size_t degree, double x)
{
    if (degree == 0)
        return 1;

    double tmp = 1;
    for (int i = 1; i <= degree; ++i)
    {
        tmp *= (x - mX[i-1]);
    }
    //std::cout << "wielomian czynnikiowy stopnia " << degree << " " << tmp << std::endl;
    return tmp ;
}

```

```

double NewtonInterpolation::differentialQuotient(size_t n )
{
    double sum = 0.0;
    for (int i = 0; i <= n; ++i)
    {
        double quotient=1.0;

        for (int j = 0; j <= n; ++j)
        {
            if (i != j)
            {
                quotient *= (mX[i] - mX[j]);
            }
        }
        sum += (mY[i] / quotient);
    }
    //std::cout << "iloraz roznicy stopnia " << n << " " << sum << std::endl;
    return sum;
}

void NewtonInterpolation::differentialQuotient()
{
    mAi.clear();
    for (size_t i = 0; i <= mNodes; ++i)
    {
        mAi.push_back(differentialQuotient(i));
    }
}

void NewtonInterpolation::setArrays(const std::vector<double>& x, const std::vector<double>& y)
{
    mNodes = mX.size() - 1;
    mX.clear();
    mX = x;
    mY.clear();
    mY = y;
    differentialQuotient();
}

void NewtonInterpolation::setArrays(const double * x, const double * y, size_t size)
{
    mX.clear();
    mY.clear();
    mNodes = size-1;
    for (size_t i = 0; i <size; ++i)
    {
        mX.push_back(x[i]);
        mY.push_back(y[i]);
    }
    differentialQuotient();
}

double NewtonInterpolation::compute(double x)
{
    double temp = 0.0;
    for (size_t i = 0; i <= mNodes; ++i)
    {
        temp += (mAi[i]*factorPolynomialial(i, x));
    }

    return temp;
}

```

```

„main.cpp”
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <string>
#include <conio.h>
#include "Interpolation.h"

void reportMini(std::ostream& output, const std::vector<double>& x, const std::vector<double>& y);

void report(std::ostream& output, const std::vector<double>& x, const std::vector<double>& y, const
std::vector<double>& Lx, const std::vector<double>& error);

int main()
{
    //zadana funkcja
    auto fx = [](double x) {return abs(cos(x)*x); };

    int downRange=-3, upRange=3;
    size_t n = 7;
    size_t np = 150;
    std::vector<double> x;
    std::vector<double> y;
    std::vector<double> Ly;
    std::vector<double> error;//nazwa robocza
    std::stringstream sstream;

    std::cout << "Podaj dolny zakres:";
    std::cin >> downRange;
    std::cout << "Podaj gorny zakres:";
    std::cin >> upRange;
    std::cout << "Podaj liczbe wezlow:";
    std::cin >> n;
    sstream << "(" << downRange << " " << upRange << ")n" << n << ".txt";
    std::string fileName = sstream.str();
    std::cout << fileName<<"\n";
    //std::cout << "Podaj nazwe pliku dla raportu :";
    //std::cin >> fileName;

    //obliczanie wezlow rownoleglych
    parallelNodes(n, downRange, upRange, x);
    for (auto obj : x) {
        y.push_back(fx(obj));
    }
    NewtonInterpolation I(x, y);

    std::fstream raportFile(fileName, std::fstream::out);
    if (raportFile.is_open())
    {
        raportFile << "Wielomian interpolacyjny Newtona w punktach lezacych w przedziale <" <<
downRange << "," << upRange<< ">\n";
        raportFile << "dla funkcji f(x)=|cos(x)*x|\n";
        raportFile << "Wezly rownolegle n=" << n<< "\n";
        raportFile << "Wartosci w wezlach:\n";
        reportMini(raportFile, x, y);
    }
    reportMini(std::cout, x, y);

    //obliczanie wartosci funkcji interpolowanej i interpolujacej w rownoleglych wezlach
    parallelNodes(np, downRange, upRange, x);
    Ly.clear();
    y.clear();
    error.clear();
    for (auto obj : x) {
        Ly.push_back(I.compute(obj));
        y.push_back(fx(obj));
        error.push_back(absoluteError(y.back(), Ly.back()));
    }
    if(raportFile.is_open())
        report(raportFile, x, y, Ly, error);
    //report(std::cout, x, y, Ly, error);
}

```

```

    czebyszewNodes(n, downRange, upRange, x);
    y.clear();
    for (auto obj : x) {
        y.push_back(fx(obj));
    }
    I.setArrays(x, y);

    if (raportFile.is_open())
    {
        raportFile << "\nWezly optymalne n=" << n << "\n";
        raportFile << "Wartosci w wezlach:\n";
        reportMini(raportFile, x, y);
    }
    reportMini(std::cout, x, y);

    //obliczanie wartosci funkcji interpolowanej i interpolujacej w optymalnych wezlach
    parallelNodes(np, downRange, upRange, x);
    Ly.clear();
    y.clear();
    error.clear();
    for (auto obj : x) {
        Ly.push_back(I.compute(obj));
        y.push_back(fx(obj));
        error.push_back(absoluteError(y.back(), Ly.back()));
    }
    if (raportFile.is_open())
        report(raportFile, x, y, Ly, error);
    //report(std::cout, x, y, Ly, error);

    raportFile.close();
    std::cout << "\nWcisnij dowolny klawisz..";
    _getch();
    return 0;
}

void reportMini(std::ostream & output, const std::vector<double>& x, const std::vector<double>& y)
{
    size_t n = x.size();

    for (size_t i = 0; i < n; ++i)
    {
        output << "x" << i << ": " << std::left << std::setw(20) << x[i]; //<<std::scientific
        output << "y" << i << ": " << y[i] << "\n"; // << std::scientific
    }
}

void report(std::ostream & output, const std::vector<double>& x, const std::vector<double>& y, const
std::vector<double>& Lx, const std::vector<double>& error)
{
    size_t n = x.size();
    output << "\nWartosci w " << n-1 << " wezlach\n";
    output<< std::left << std::setw(6) << "i" ;
    output<< std::left << std::setw(8) << "x" ;
    output<< std::left << std::setw(20) << "f(x)" ;
    output<< std::left << std::setw(20) << "L(x)" ;
    output << "error\n";
    for (size_t i = 0; i < n; ++i)
    {
        output<< std::left << std::setw(6) << i;
        output<< std::left << std::setw(8) << x[i]; //<<std::scientific
        output<< std::left << std::setw(20) << y[i] ; // << std::scientific
        output<< std::left << std::setw(20) << Lx[i] ;
        output<< std::left << std::setw(20) << error[i]<< "\n";
    }
}

```