# MOwNiT Lab3 - sprawozdanie

## Autor: Michał Flak

### Zadanie 1

*Proszę zastosować aproksymację Czebyszewa dla funkcji:*
*y = exp((x* ***2) w przedziale od -1 do 1***
***y = abs(x+x*** *3) w przedziale od -1 do 1*
*y = sign(x) w przedziale od -1 do 1*
*Do aproksymacji należy użyć biblioteki GSL.*
*Dla każdej funkcji proszę:*
*· narysować wykres funkcji aproksymowanej i aproksymującej (gnuplot)*
*· sprawdzić, jak wynik zależy od stopnia wielomianu aproksymującego - przedstawić odpowiednie wykresy*

Napisano program implementujący te funkcje. Liczy on ich wartości na określonym przedziale bezpośrednio, jak również z pomocą GSL ich aproksymacje Czebyszewa o stopniu określonym w definicji:

```
#define ORDER 8
```

Program zapisuje pliki o nazwie:

```
output/fun{{numer funkcji}}ord{{stopień wielomianu}}.txt
na przykład
output/fun1ord3.txt
```

Program podaje na wyjściu informację w formacie:

```
stopien f1_err   f2_err   f3_err
1       0.455703 0.564391 0.414243
```

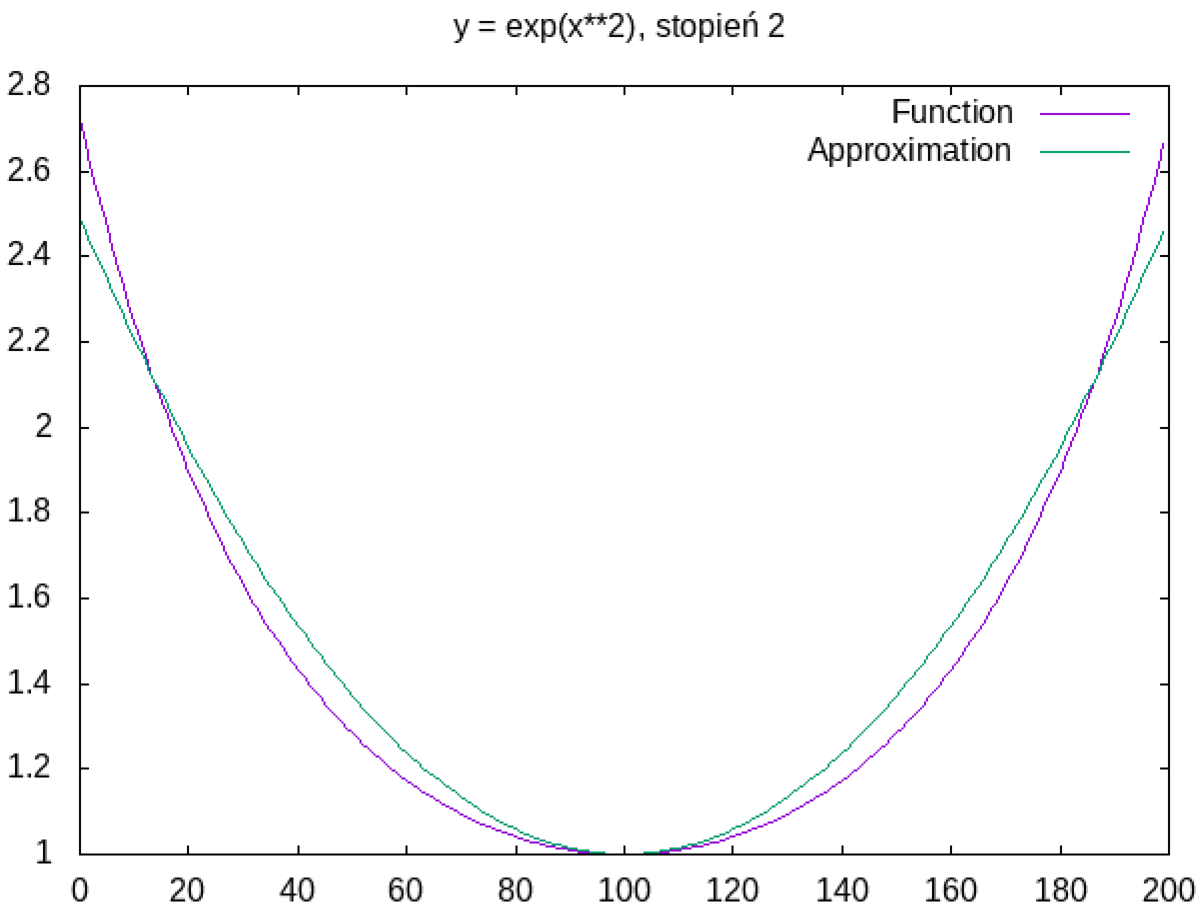
gdzie fX_err to średni błąd względny przybliżenia.

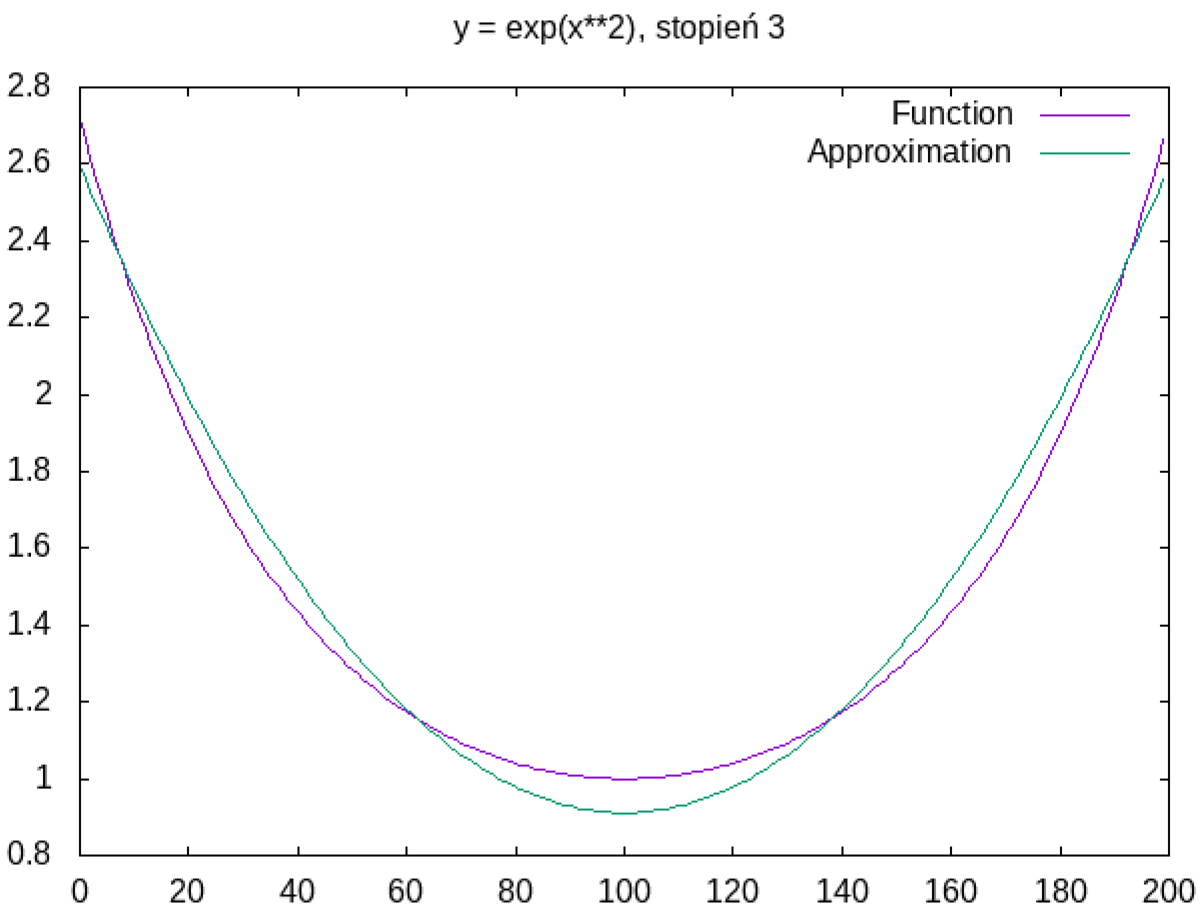
Fragment kodu odpowiadający za liczenie wartości i błędu funkcji i aproksymacji:

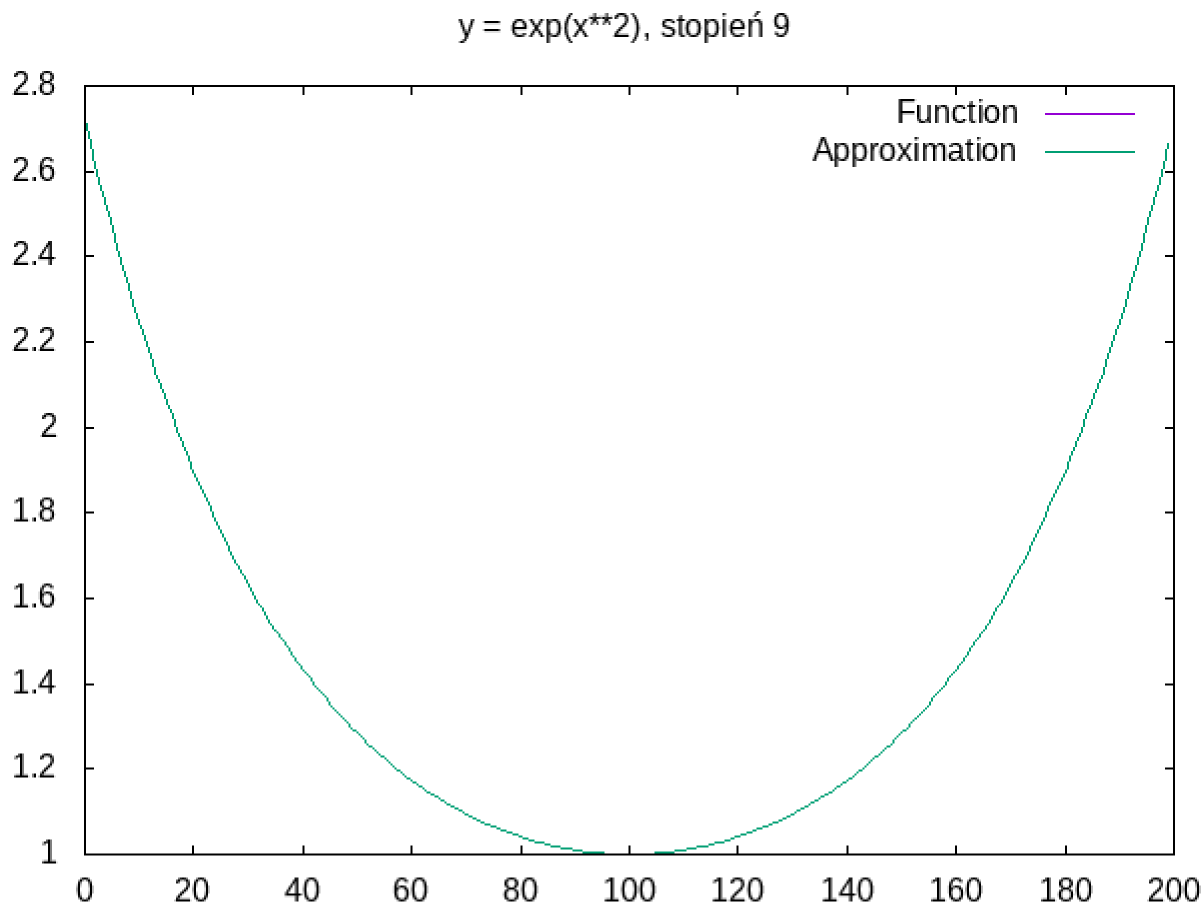
```
int main (void)
{
  const double a = -1.0;
  const double b = 1.0;
  double xi, yi, ayi, aerr, absoluteErrorSum;
  int passes;
  gsl_function F[3];
  double meanAbsoluteErrors[3];
```

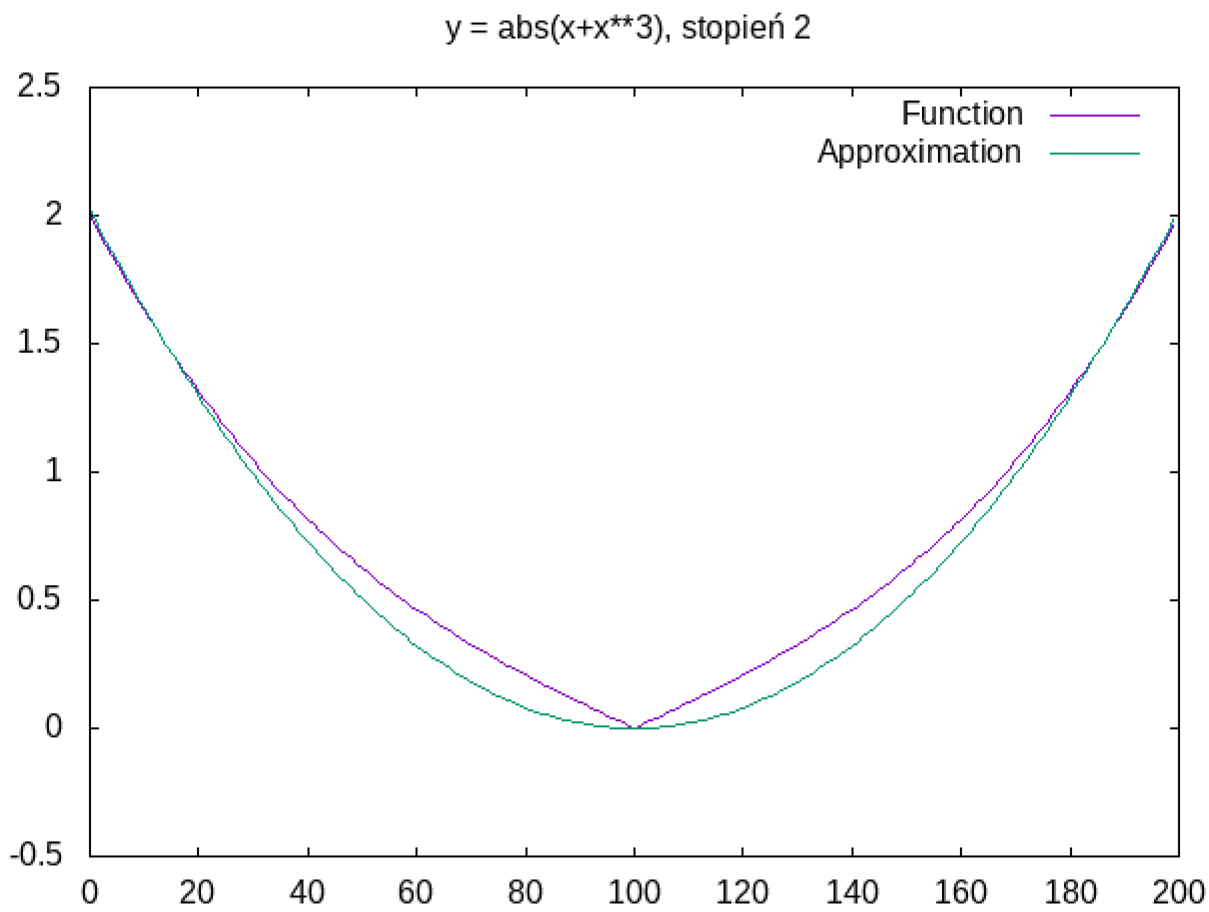
```c
    F[0].function = fun1;
    F[1].function = fun2;
    F[2].function = fun3;

    FILE *output[3];
    output[0] = fopen(NAME( 1, ORDER ), "w");
    output[1] = fopen(NAME( 2, ORDER ), "w");
    output[2] = fopen(NAME( 3, ORDER ), "w");

    for(int fi = 0; fi < 3; fi++)
    {
      passes = 0;
      absoluteErrorSum = 0.0;
      gsl_cheb_series* cs = gsl_cheb_alloc(ORDER);
      gsl_cheb_init(cs, &F[fi], a, b);

      for (xi = a; xi <= b; xi += 0.01)
      {
          yi = F[fi].function(xi, NULL);
          gsl_cheb_eval_err(cs, xi, &ayi, &aerr);
          fprintf (output[fi],"%g %g %g %g\n", xi, yi, ayi, aerr);
          passes++;
          absoluteErrorSum += fabs(yi - ayi);
      }
      meanAbsoluteErrors[fi] = absoluteErrorSum / (double)passes;
    }
    printf("%d %g %g %g\n",
      ORDER,
      meanAbsoluteErrors[0],
      meanAbsoluteErrors[1],
      meanAbsoluteErrors[2]
      );
    return 0;
  }
```

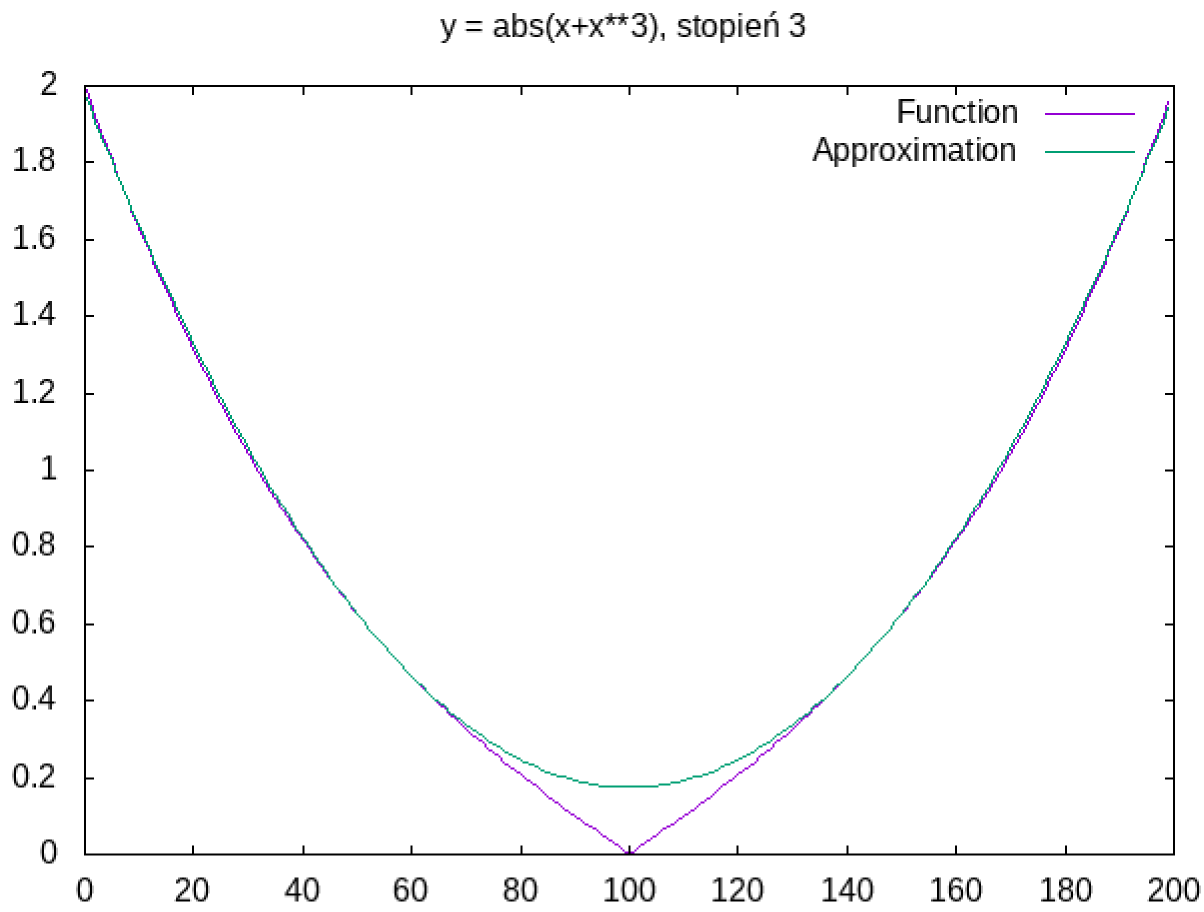
Używając skryptów oraz programu gnuplot przygotowano dane i wykresy dla wszystkich funkcji oraz stopni wielomianu [1..50]. Przykładowe uzyskane obrazy:


y = exp(x**2), stopień 2
Function
Approximation
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
20
40
60
80
100
120
140
160
180
200

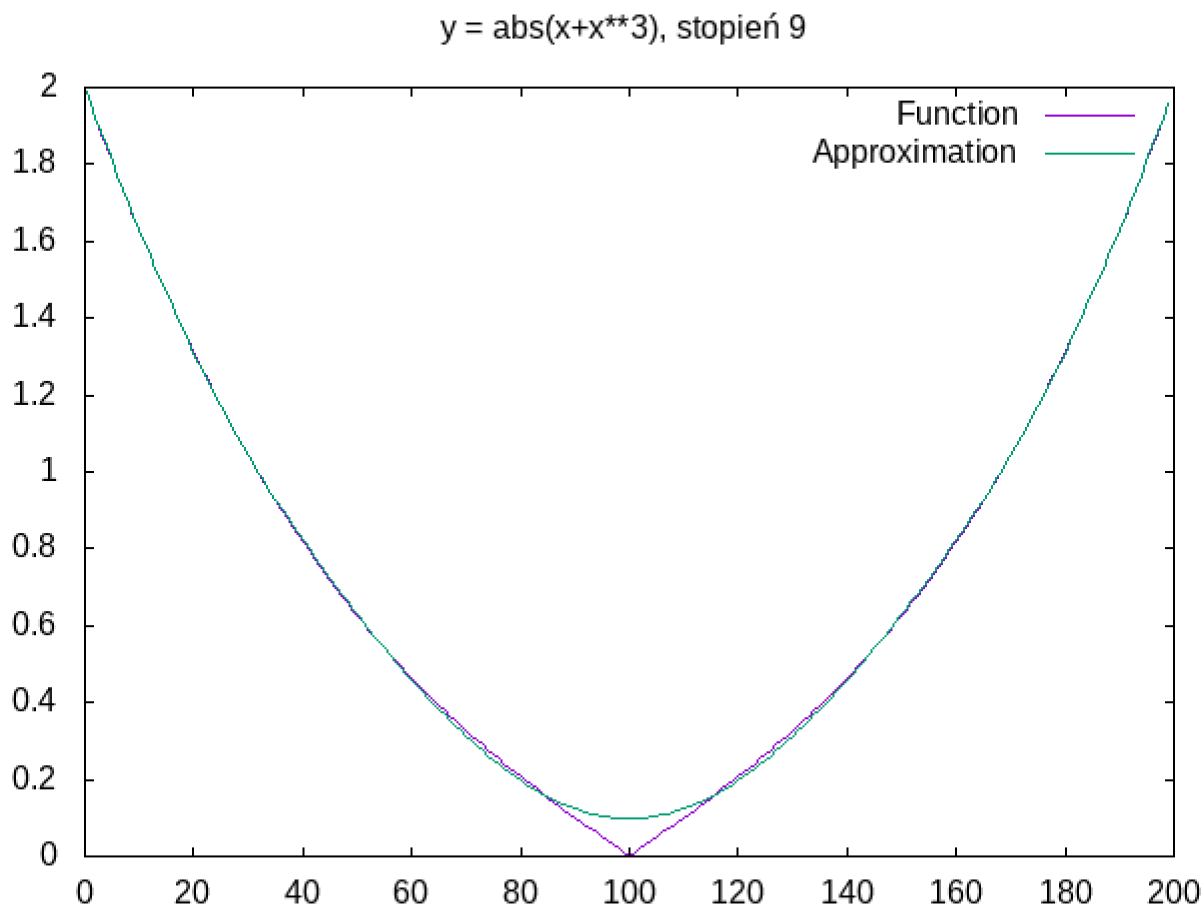

y = exp(x**2), stopień 3
Function
Approximation
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
20
40
60
80
100
120
140
160
180
200


y = exp(x**2), stopień 9
Function
Approximation
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
20
40
60
80
100
120
140
160
180
200

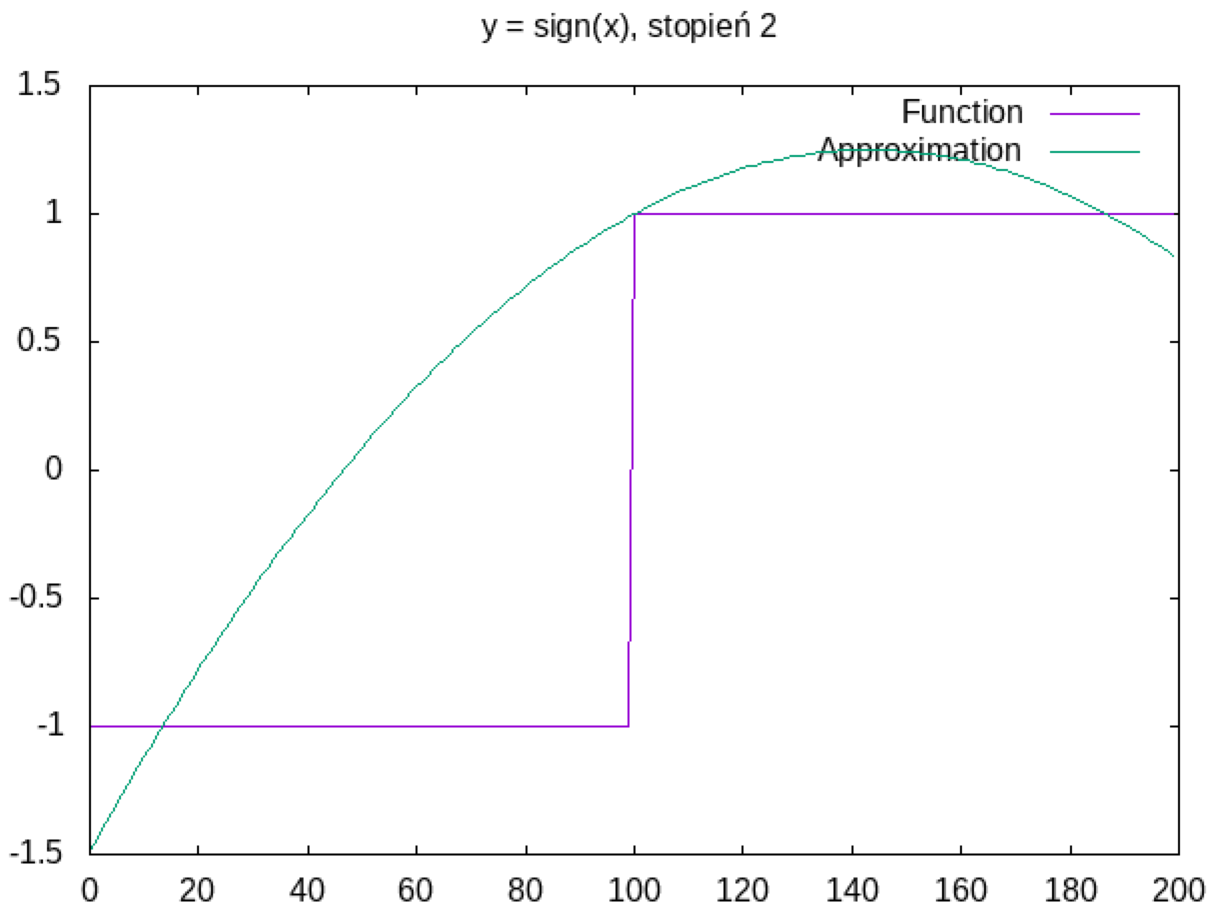

y = abs(x+x**3), stopień 2
Function
Approximation
2.5
2
1.5
1
0.5
0
-0.5
0
20
40
60
80
100
120
140
160
180
200

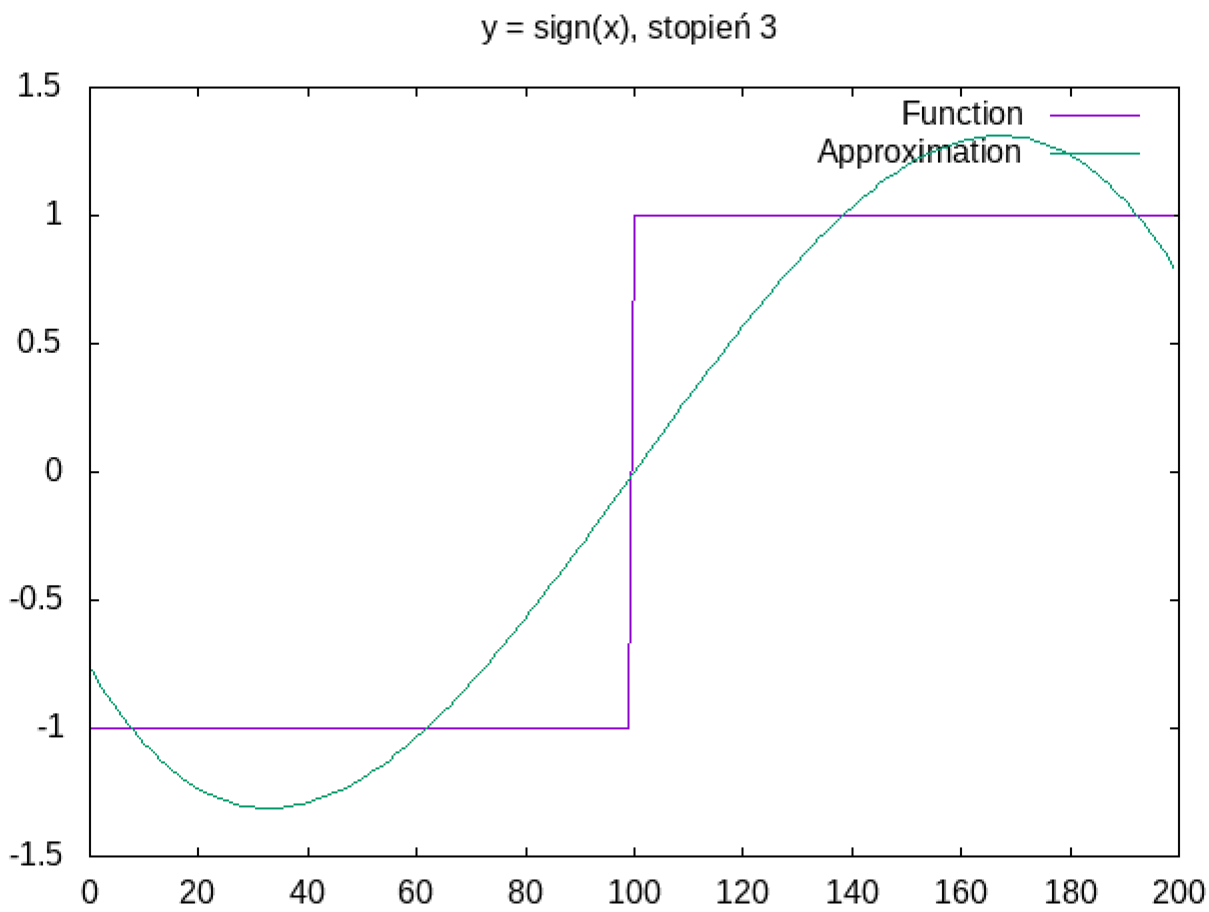


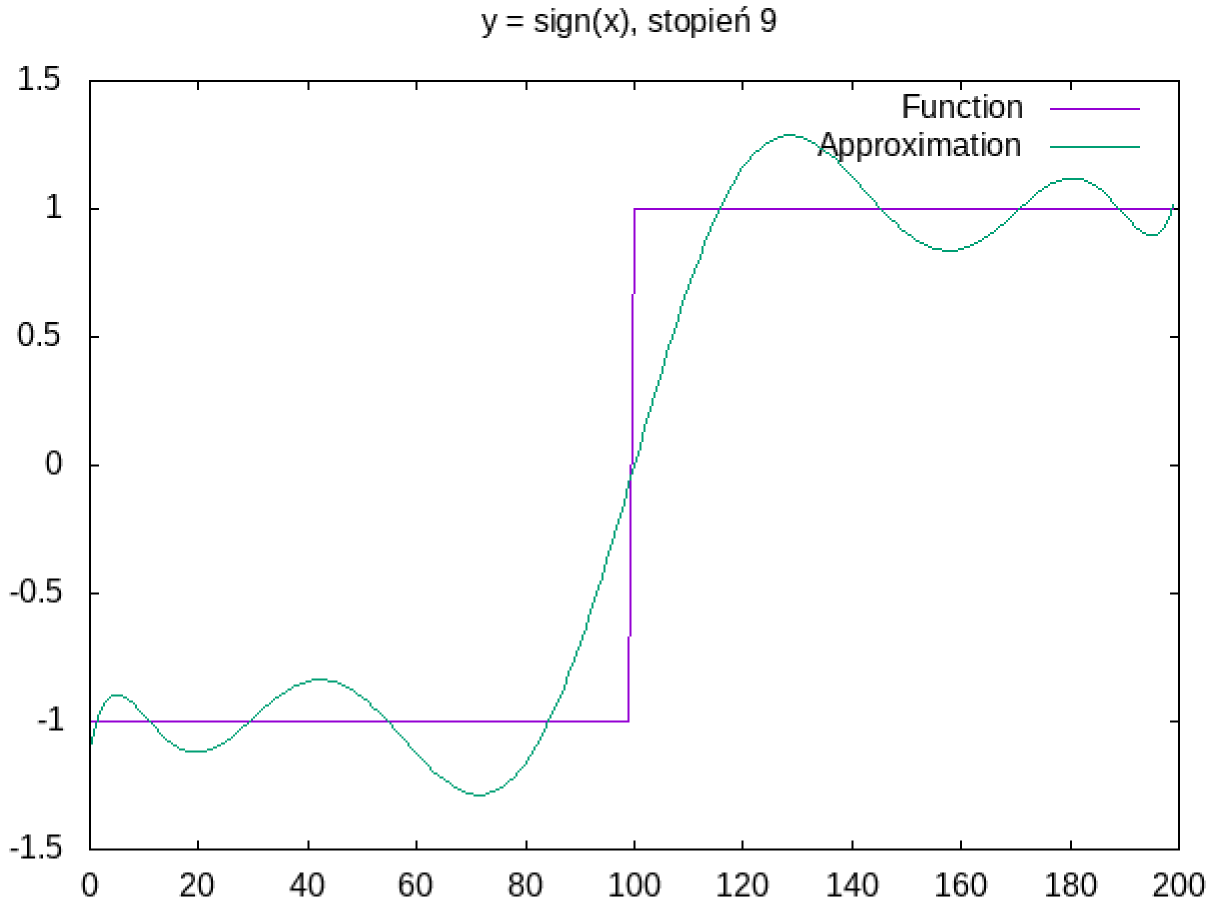
y = abs(x+x**3), stopień 3
Function
Approximation
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
160
180
200

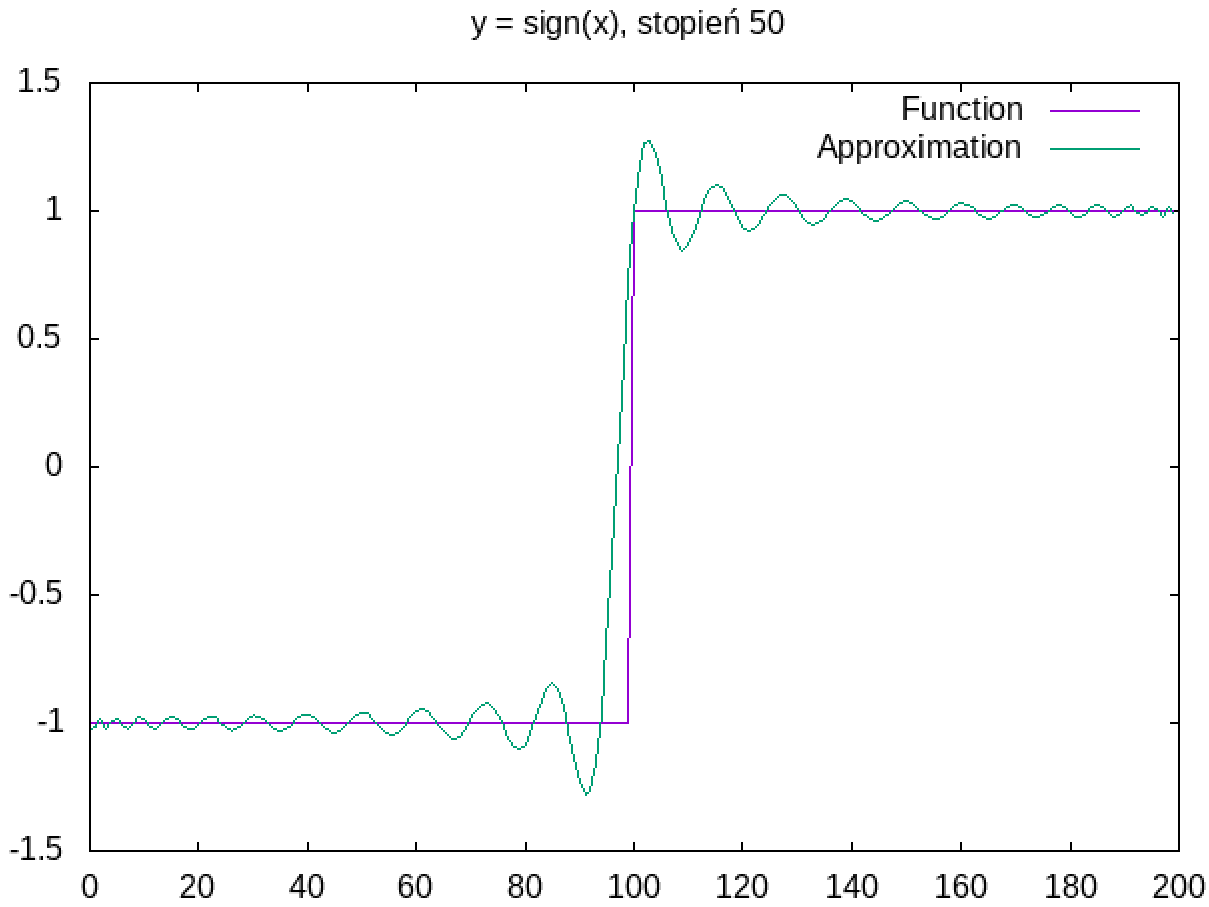



y = abs(x+x**3), stopień 9
Function
Approximation
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
160
180
200

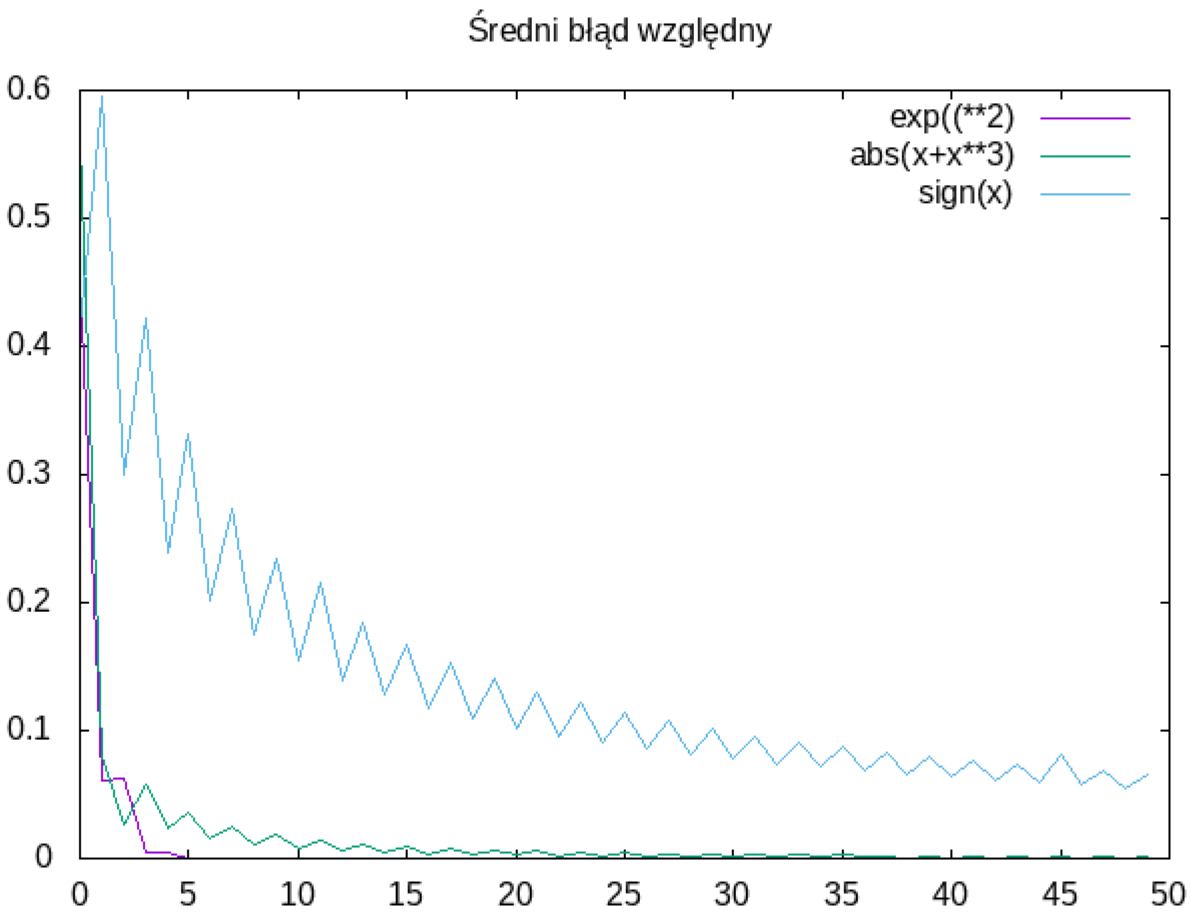

y = sign(x), stopień 2
Function
Approximation
1.5
1
0.5
0
-0.5
-1
-1.5
0
20
40
60
80
100
120
140
160
180
200


y = sign(x), stopień 3
Function
Approximation
1.5
1
0.5
0
-0.5
-1
-1.5
0
20
40
60
80
100
120
140
160
180
200




Można zauważyć, że im większy stopień wielomianu, tym bardziej przybliżenie pasuje do oryginalnego wykresu. Potwierdza to również wykres błędu od stopnia wielomianu:




Widzimy wyraźną tendencję spadkową, jednak wraz z rosnącym stopniem wielomianu, pożytek ze zwiększania go maleje. Najgorzej aproksymować daje się funkcja signum.

### Zadanie 2, 3

*Wykonać aproksymację funkcji f(x)=(1/2)^( x^2+2x), gdzie x należy do przedziału (-1,1) przy pomocy aproksymacji średniokwadratowej*

*Znaleźć aproksymację tej funkcji przy pomocy aproksymacji jednostajnej i porównać tę aproksymację z wynikiem z p1.*

Wybrałem regresję liniową jako algorytm aproksymacji średniokwadratowej, realizowaną przez funkcję GSL `gsl_fit_linear`.
Wybrałem aproksymację Czebyszewa jako aproksymację jednostajną, zrealizowaną przez funkcję GSL `gsl_cheb_eval`. Użyłem stopnia 5 i 50.

Napisałem program liczący wartość podanej funkcji w zakresie [-1..1], generujący plik `output2/comparison.txt` w formacie:

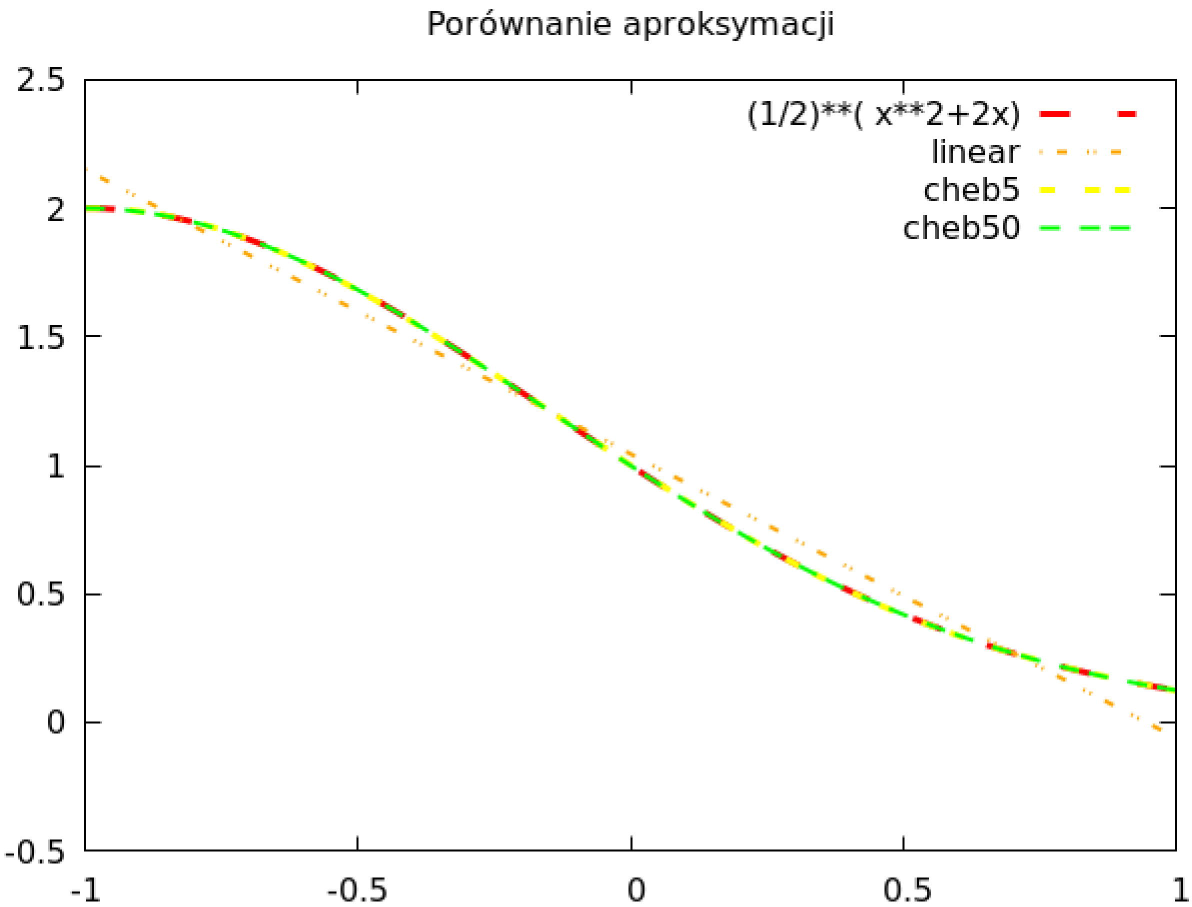
```
x,     y,      linY,   chebY5, chebY50
1      2       2.15076 1.99847 2
-0.999 2       2.14965 1.99852 2
-0.998 1.99999 2.14855 1.99857 1.99999
```

Program na wyjściu daje średni błąd względny dla każdej metody przybliżenia:

```
meanAbsErrLin, meanAbsErrCheb5, meanAbsErrCheb50
0.0633058      0.00114041       4.80441e-15
```

Widzimy więc, że najlepsze wyniki daje przybliżenie Czebyszewa - im większy stopień, tym mniejszy błąd.

Narysowano wykres prezentujący metody aproksymacji:




Kod programu do wglądu:

```
static double fun1(double x, void* params)
{
  return pow(0.5, pow(x,2.0) + 2.0*x);
}

void compareFit()
{
    FILE *output = fopen("output2/comparison.txt", "w");
  //setup linear fit
    const int stepsCount = 1000;
    double xArray[stepsCount * 2];
    double yArray[stepsCount * 2];

    for (int i = -stepsCount; i <= stepsCount; i++)
    {
        double x = (double)i / (double)stepsCount;
```

```
        double y = fun1(x, NULL);

        xArray[i + stepsCount] = x;
        yArray[i + stepsCount] = y;
    }

    double c0 = 0;
    double c1 = 0;
    double cov00 = 0;
    double cov01 = 0;
    double cov11 = 0;
    double sumsq = 0;
    gsl_fit_linear(
      xArray,
      1,
      yArray,
      1,
      stepsCount * 2,
      &c0,
      &c1,
      &cov00,
      &cov01,
      &cov11,
      &sumsq
      );

    //setup Chebyshev approx
    const double a = -1.0;
    const double b = 1.0;
    gsl_function F;
    F.function = fun1;

    gsl_cheb_series* cs5 = gsl_cheb_alloc(5);
    gsl_cheb_series* cs50 = gsl_cheb_alloc(50);

    gsl_cheb_init(cs5, &F, a, b);
    gsl_cheb_init(cs50, &F, a, b);
    //end setup

    //error calculating setup
    int passes = 0;
    double absoluteErrorSumLin = 0.0;
    double absoluteErrorSumCheb5 = 0.0;
    double absoluteErrorSumCheb50 = 0.0;
    double meanAbsoluteErrorLin = 0.0;
    double meanAbsoluteErrorCheb5 = 0.0;
    double meanAbsoluteErrorCheb50 = 0.0;
    //end setup

    for (int i = -stepsCount; i <= stepsCount; i++)
    {
        double x = (double)i / (double)stepsCount;
        double y = fun1(x, NULL);
        double linY = c0 + c1 * x;
```

```
        double chebY5 = gsl_cheb_eval(cs5, x);
        double chebY50 = gsl_cheb_eval(cs50, x);

        passes++;
        absoluteErrorSumLin += fabs(y - linY);
        absoluteErrorSumCheb5 += fabs(y - chebY5);
        absoluteErrorSumCheb50 += fabs(y - chebY50);

        fprintf (output,"%g %g %g %g %g\n", x, y, linY, chebY5, chebY50);
    }

    meanAbsoluteErrorLin    = absoluteErrorSumLin    / (double)passes;
    meanAbsoluteErrorCheb5  = absoluteErrorSumCheb5  / (double)passes;
    meanAbsoluteErrorCheb50 = absoluteErrorSumCheb50 / (double)passes;

    printf ("meanAbsErrLin, meanAbsErrCheb5, meanAbsErrCheb50\n");
    printf (
      "%g %g %g\n",
      meanAbsoluteErrorLin,
      meanAbsoluteErrorCheb5,
      meanAbsoluteErrorCheb50
    );

    gsl_cheb_free(cs5);
    gsl_cheb_free(cs50);
}
```