Temat:			
Szybka transformata sinusowa			
Wykonał: Marcin Fabrykowski	Wydział: FiIS	Kierunek Inf. Stos.	Grupa: grupa 3

1. Wstęp
   Szybka transformata sinusowa to liniowa i odwracalna funkcja prowadząca z $R^N \to R^N$. Definiuje się ją jako:

$$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N+1}(n+1)(k+1)\right],\ k = 0, 1, 2, \ldots, N-1$$

2. Wykonanie ćwiczenia
   Zaczynamy od wygenerowania sygnału okresowego zgodnie ze wzorem:

$$y_0(i) = \sin(\omega \cdot i) + \sin(2\omega \cdot i) + \sin(3\omega \cdot i)$$

   przy założeniu że $\omega = 2\frac{2\pi}{n}$
   Następnie do naszego sygnału generujemy szum. Szum będzie w zakresie $a \in (-1, 1)$. Generujemy go wykorzystując zmienną losową

$$X = \frac{rand()}{RAND\_MAX + 1.0}$$

$$a = 2 * X - 1$$

   Nasz sygnał z szumem będzie miał postać: $y(i) = y_0(i) + a$.
   Zapisujemy nasz sygnał do wektora. Długość wektora wynosi $n = 2^k$.
   Następnie wykonujemy szybką transformatę sinusową. Do tego celu używamy funkcji **sinft** z biblioteki Numerical Reciples.
   Wyznaczamy wartość maksymalną, a następnie zerujemy wartości mniejsze niż 25% wartości maksymalnej.
   Wykonujemy odwrotną transformatę sinusową. Wykonuje się ją poprzez ponowne wykorzystanie funkcji **sinft**, a następnie pomnożenie wyniku przez $\frac{2}{n}$
   Powyższe zadanie wykonuje program:

Listing 1: main.cpp

```
#include <iostream>
#include <math.h>
#include <nrutil.h>
#include <nrutil.c>
#include <sinft.c>
#include <realft.c>
#include <four1.c>
#include <time.h>
using namespace std;
float omega;
long n;
float y(float i);
int main(int argc, char* argv[])
{
        srand(time(NULL));
        if(argc<2)
        {
                cerr<<"Usage: "<<argv[0]<<" <k>"<<endl;
                return -1;
        };
        int k=atoi(argv[1]);
        n=pow(2,k);
        omega=4*M_PI/(n);
        long x;
        float *data=new float[n];
        FILE *plik=fopen("f.dat","w+");
        FILE *plik2=fopen("f3.dat","w");
        if(!plik)
        {
                cerr<<"File error"<<endl;
                return -2;
        };
        for(x=0;x<n;x++)
        {
                data[x]=y(x);
                fprintf(plik2,"%d %f\n",x,data[x]);
                float r=rand();
                float szum=r/(RAND_MAX+1.0);
                if(((float)rand()/(RAND_MAX+1.0)<0.5))
                        szum*=-1;
                data[x]+=szum;
//              cout<<x<<" "<<data[x]<<endl;
                fprintf(plik,"%d %f\n",x,data[x]);
        };
        fclose(plik);
//      cout<<endl;
        sinft(data,n);
        plik=fopen("f2.dat","w+");
```

```
        for(x=0;x<n;x++)
        {
                fprintf(plik,"%d %f\n",x,data[x]);
//              cout<<x<<" "<<data[x]<<endl;
        };
        float max=0;
        for(x=0;x<n;x++)
        {
                if(data[x]>max)
                        max=data[x];
        };
        for(x=0;x<n;x++)
        {
                if(data[x]<0.25*max)
                        data[x]=0;
        };
        sinft(data,n);
        fclose(plik);
        plik=fopen("f4.dat","w+");
        for(x=0;x<n;x++)
        {
                data[x]*=2.0/n;
                fprintf(plik,"%d %f\n",x,data[x]);
//              cout<<x<<" "<<data[x]<<endl;
        };
        fclose(plik);
        fclose(plik2);
        return 0;
};
float y(float i)
{
        return sin(omega*i)+sin(2*omega*i)+sin(3*omega*i);
};
```

którego argumentem jest wartość k.
Wykorzystując skrypty:

Listing 2: plot.sh

```
#!/usr/bin/gnuplot
set term jpeg
set size square
set out "f.jpg"
set xlabel "x"
set ylabel "f(x)"
set grid
show grid
plot "f.dat" using 1:2  title "f(x)" w l
replot
```

Listing 3: plot2.sh

```
#!/usr/bin/gnuplot
set term jpeg
set size square
set out "f2.jpg"
set xlabel "x"
set ylabel "f(x)"
set grid
show grid
set xrange [0:50]
plot "f2.dat" using 1:2  title "f(x)" w l
replot
```

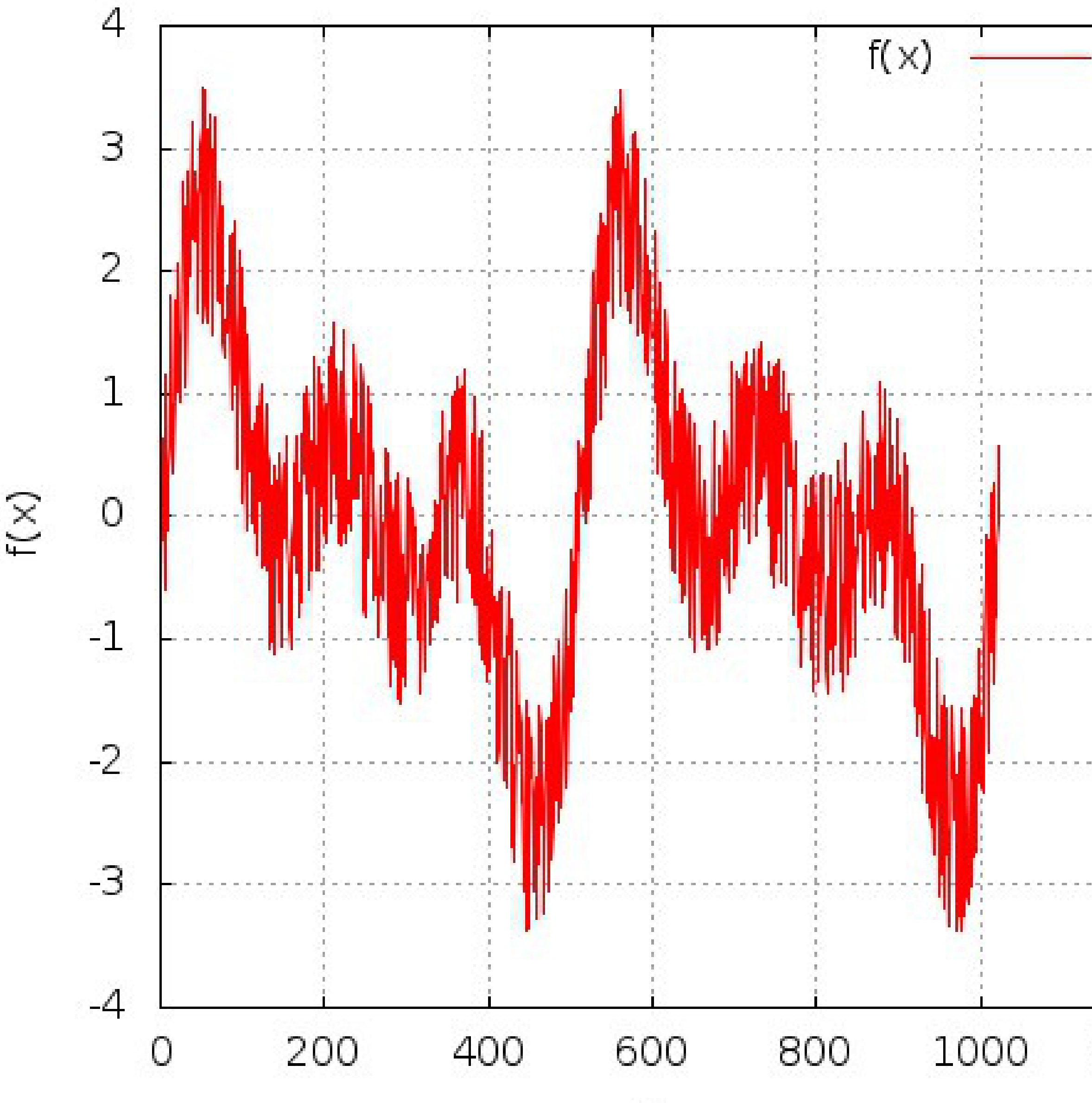
Listing 4: plot3.sh

```
#!/usr/bin/gnuplot
set term jpeg
#set size square
set out "f3.jpg"
set xlabel "x"
set ylabel "f(x)"
set grid
set xrange [0:80]
show grid
plot "f3.dat" using 1:2  title "f(x)" w l, "f4.dat" using
    1:2 title "f2(x)" w l
#plot "f4.dat" using 1:2 title "f2(x)" w l
replot
```

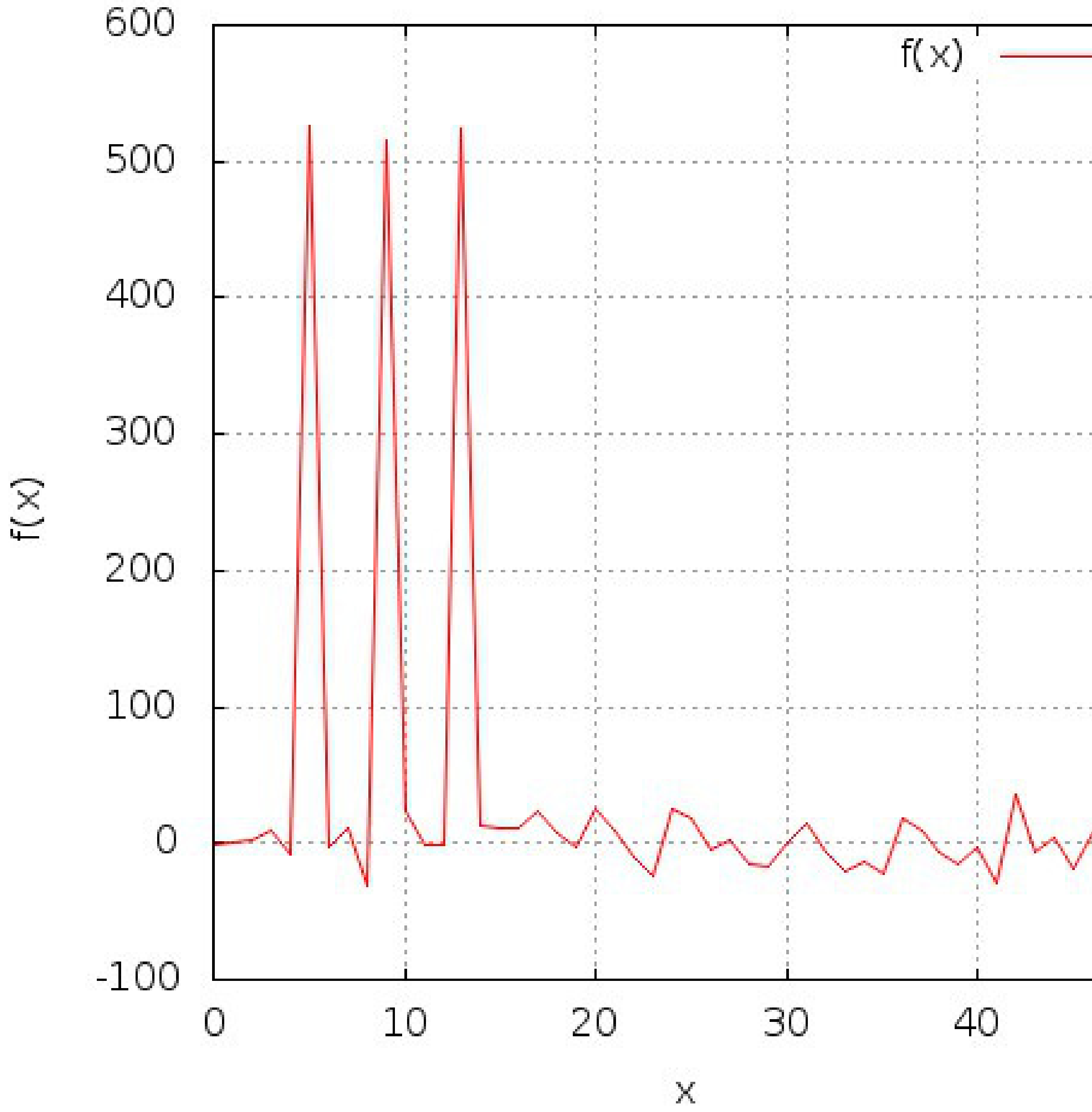
wykonujemy wykresy.

3. Wnioski
   Jak widać, metoda transformaty sinusowej sprawdza się w odszumianiu sygnałów. Dokładność ta rośnie wraz z k. Przy $k = 10$ widać praktycznie idealne nałożenie się sygnału pierwotnego i odszumionego. Dla $k = 6$ zauważamy pewnie niedokładności jednak ich poziom oceniam jako dopuszczalny.

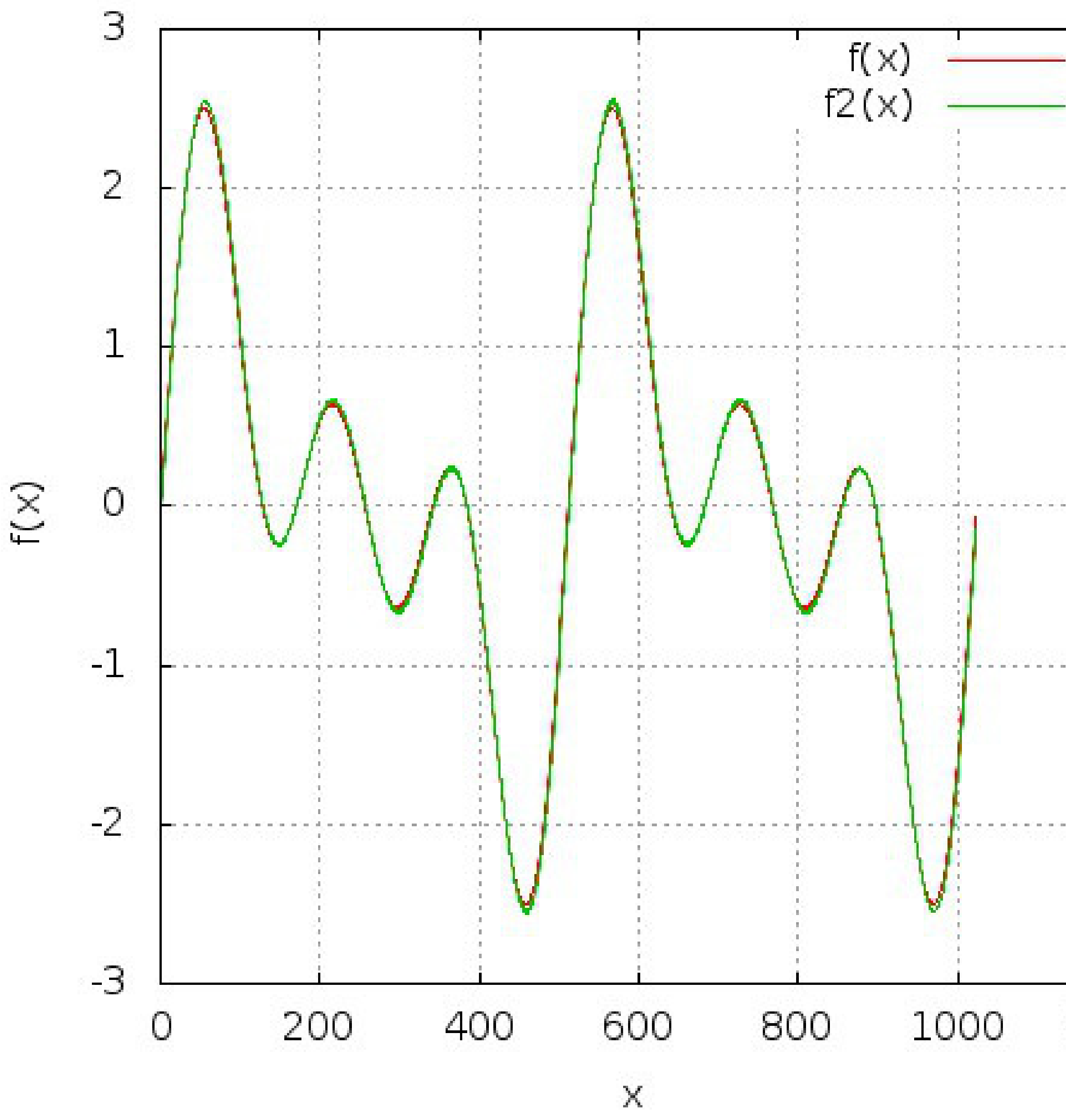
Rysunek 1: k=10, pełny zakres sygnału zaszumionego



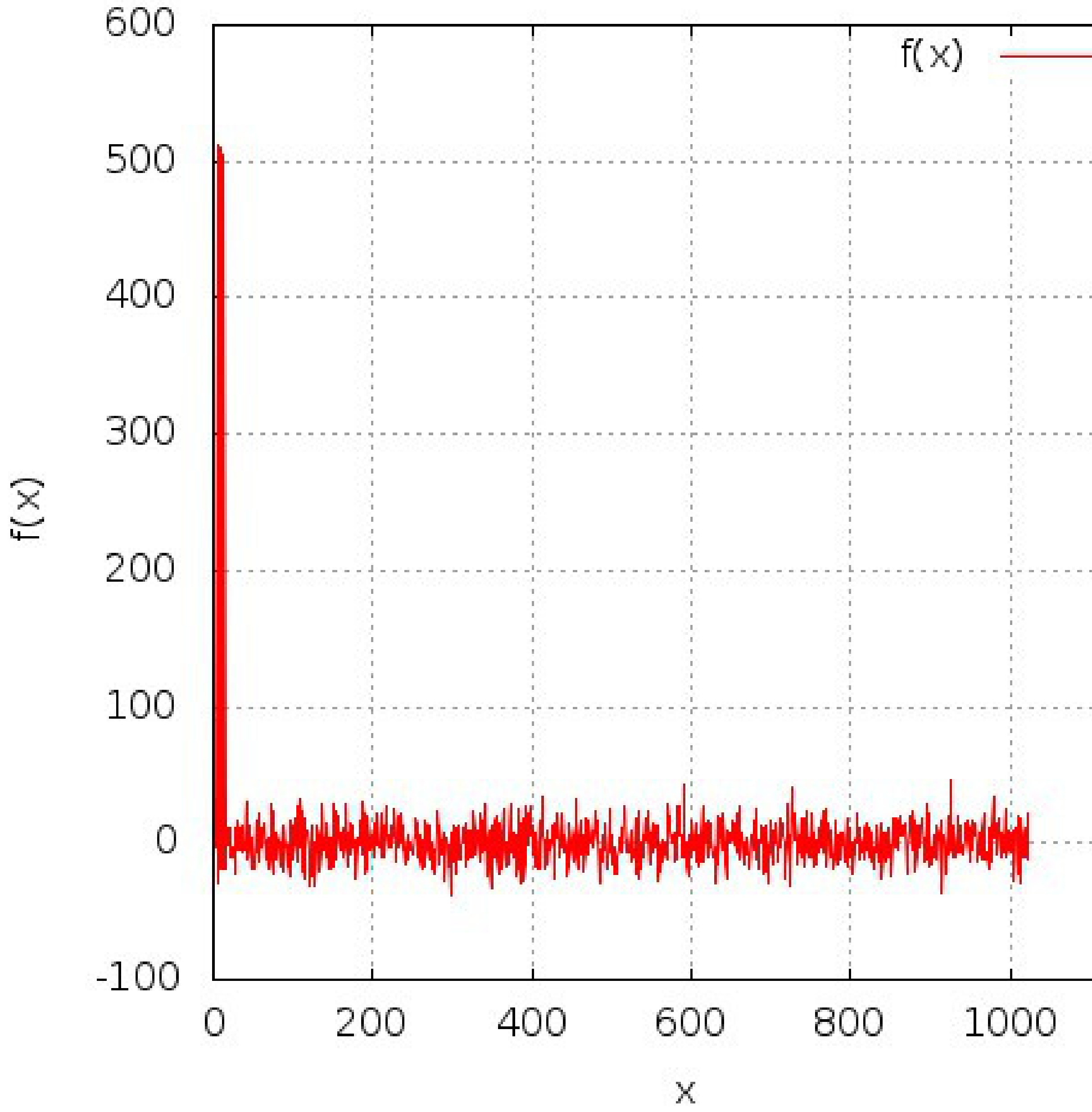
Rysunek 2: k=10, transformata z widocznymi pikami



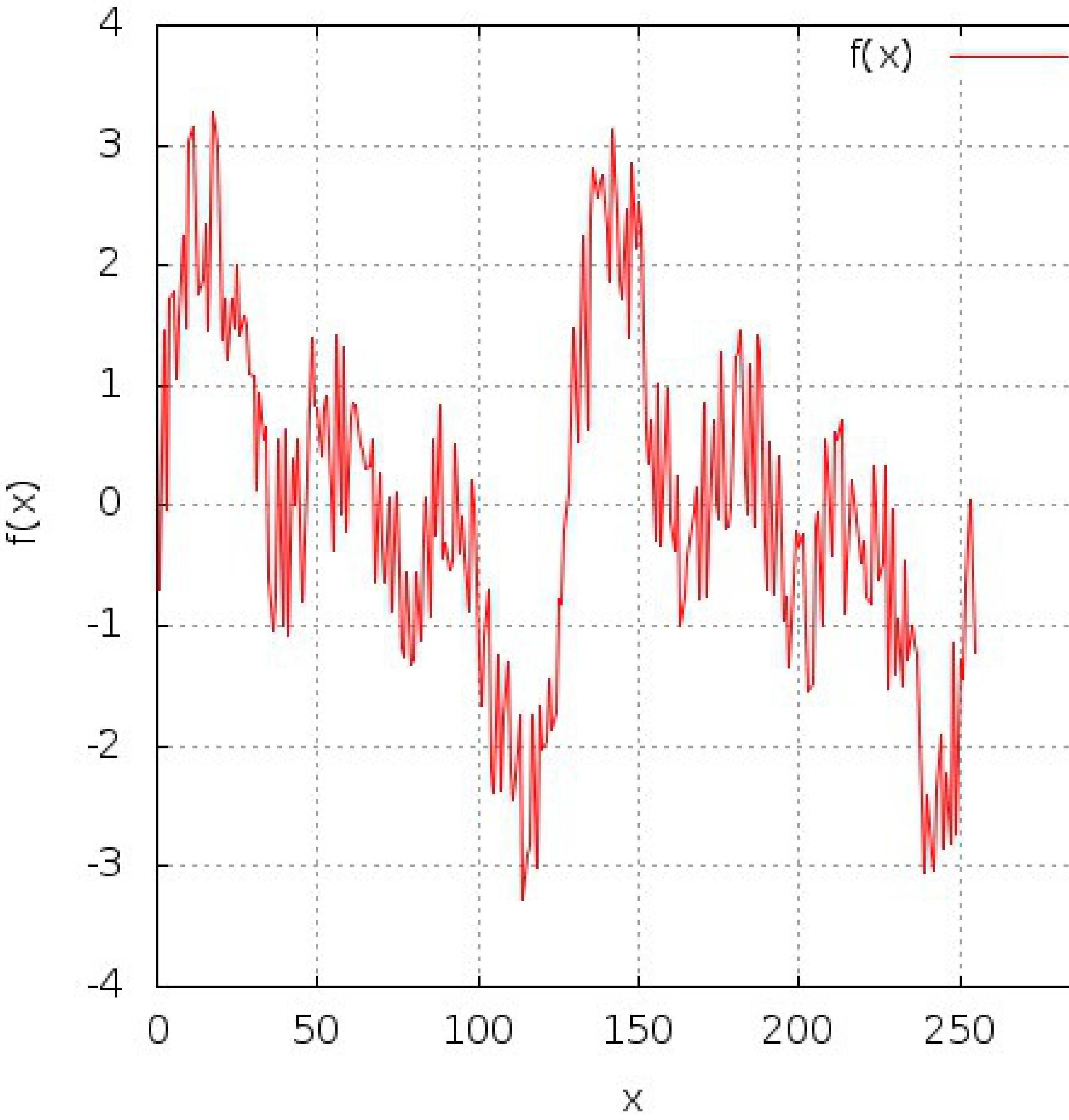
Rysunek 3: k=10, sygnał niezaszumiony i odszumiony



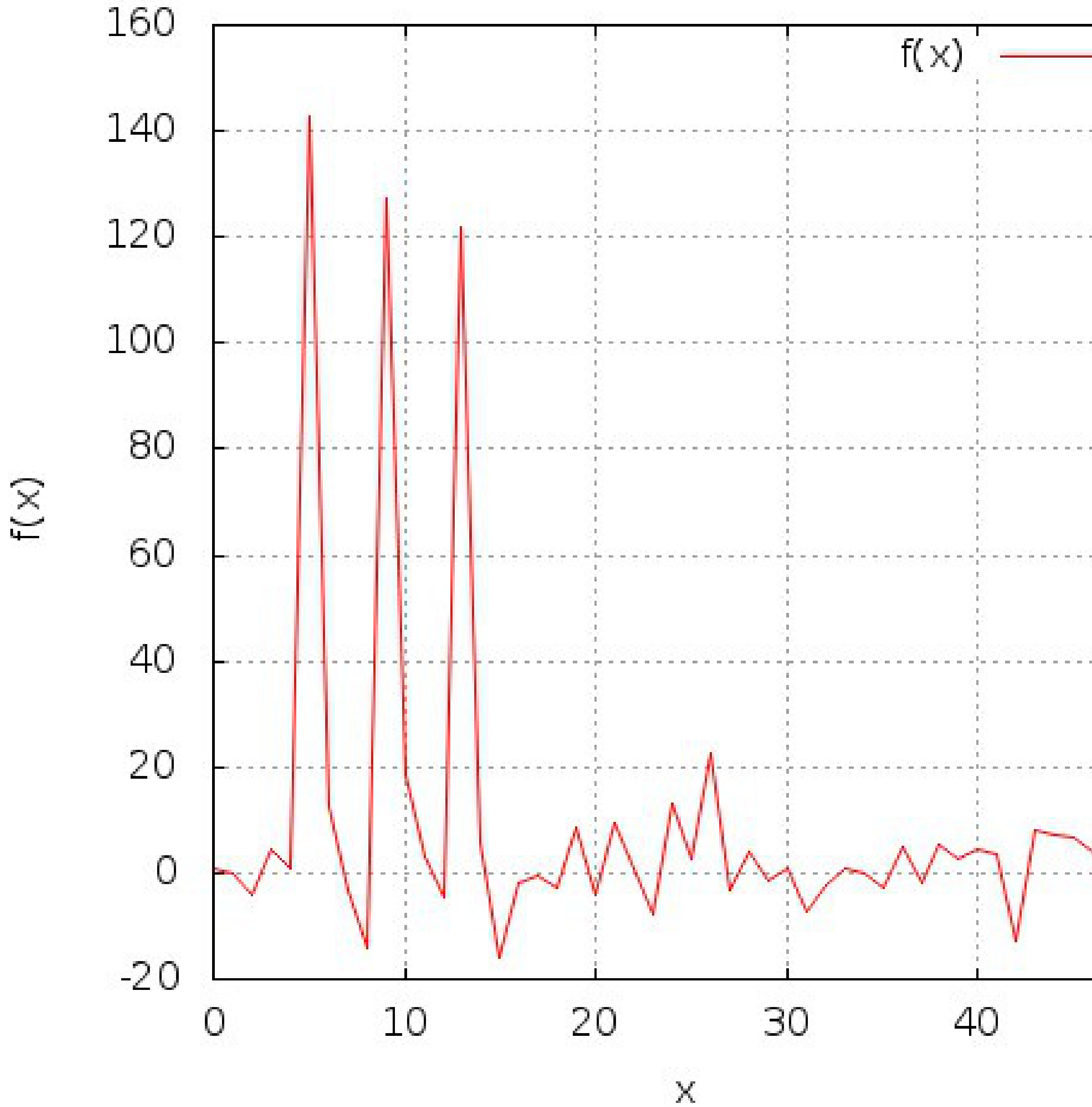
Rysunek 4: k=10, transformata w pełnym zakresie



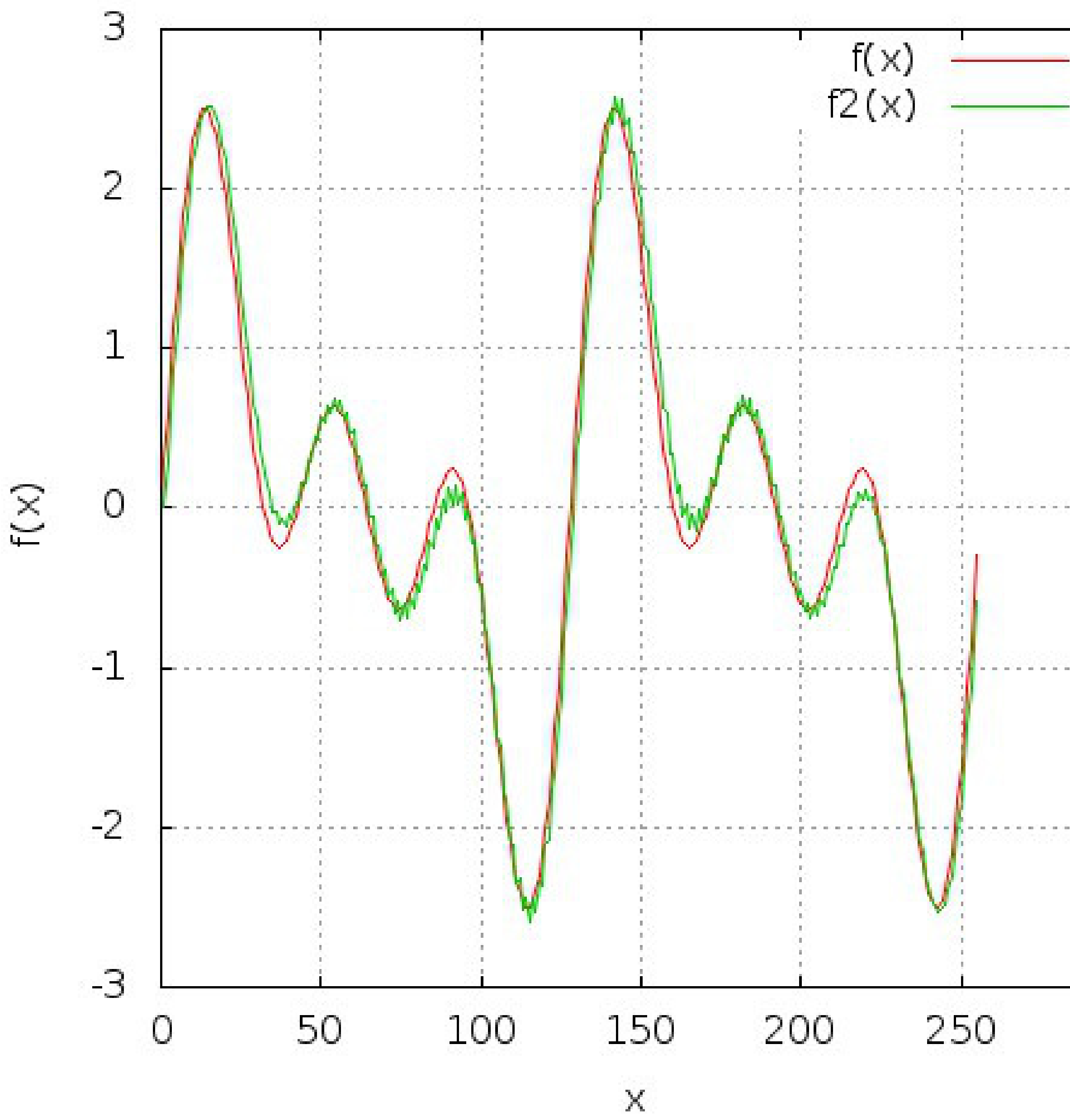
Rysunek 5: k=8, transformata w pełnym zakresie



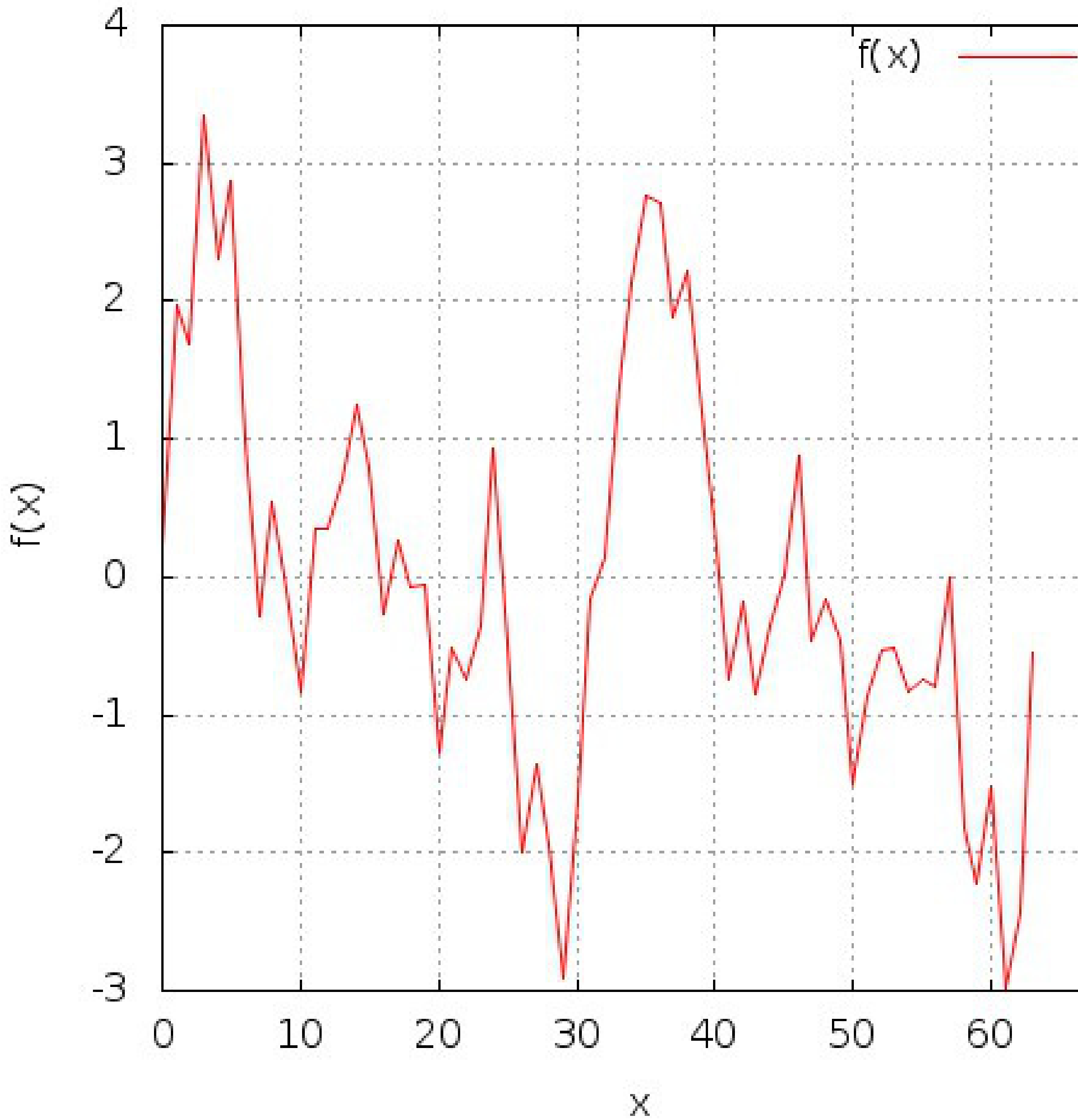
Rysunek 6: k=8, transformata z widocznymi pikami



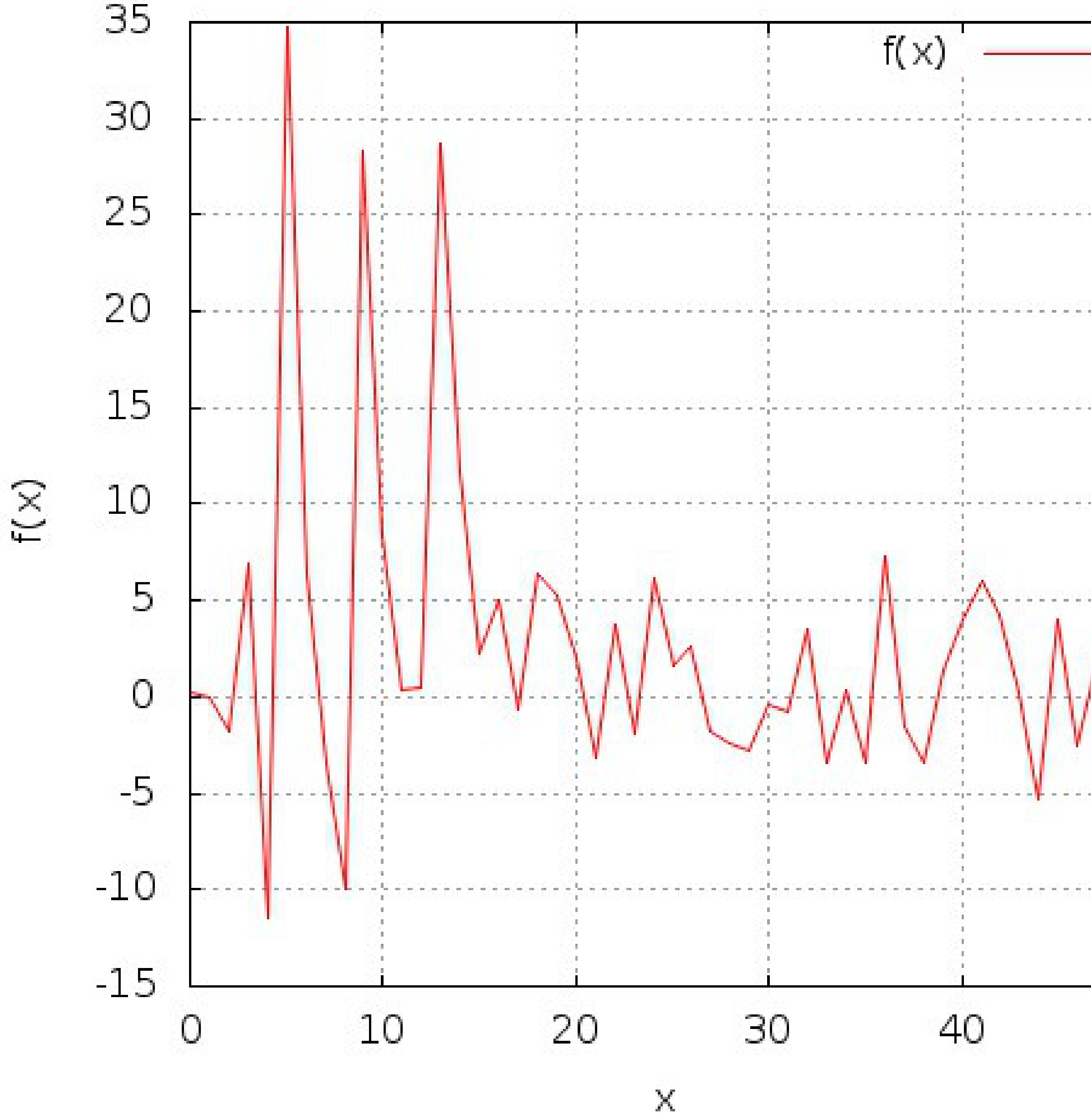
Rysunek 7: k=8, sygnał niezaszumiony i odszumiony



Rysunek 8: k=6, transformata w pełnym zakresie



Rysunek 9: k=6, transformata z widocznymi pikami



Rysunek 10: k=6, sygnał niezaszumiony i odszumiony