Laboratorium nr 3

Wykonaj w formie programistycznej implementacji poniżej przedstawione zadania.

1) Napisz funkcję realizującą Dyskretną Transformatę Fouriera.

Funkcja realizująca DFT*:*

```
QVector<std::complex<double>> Lab3::calculateDFT(QVector<double> signal)
{
    int steps = signal.length();
    QVector<std::complex<double>> temp;
    for(int k = 0; k<steps; k++)
    {
        std::complex<double> sum(0,0);
        for(int n = 0; n<steps; n++)
        {
            double phase = (2*M_PI*k*n)/steps;
            std::complex<double> w(cos(phase), -sin(phase));
            sum+=signal.at(n)*w;
        }
        temp.append(sum);
    }
    return temp;
}
```

2) Użyj funkcji z poprzednich zajęć i wyznacz dyskretny sygnał tonu prostego $x(n)$. Wygeneruj wykres dla $n \in \langle 0; \widehat{A}\widehat{B}\widehat{C}\rangle$, jako parametry inicjalizujące przyjmij: $A = 1.0$ [V], $f = \widehat{B}$ [Hz], $\varphi = \widehat{C} \cdot \pi$ [rad].

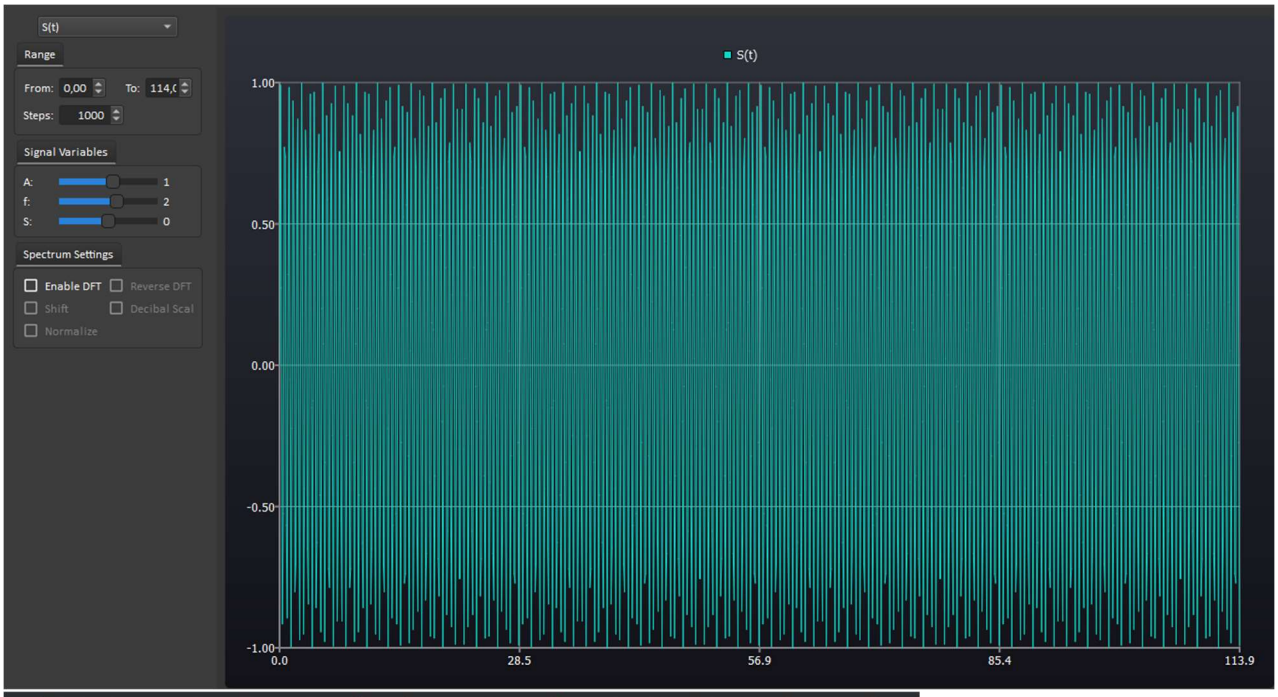
Użyj przekształcenia DFT z zadania pierwszego i dla uzyskanej reprezentacji sygnału $x(n)$ w dziedzinie czasu wyznacz sygnał w dziedzinie częstotliwości $X(k)$.

Oblicz widmo amplitudowe jako $M(k) = \sqrt{Re[X(k)]^2 + Im[X(k)]^2}$

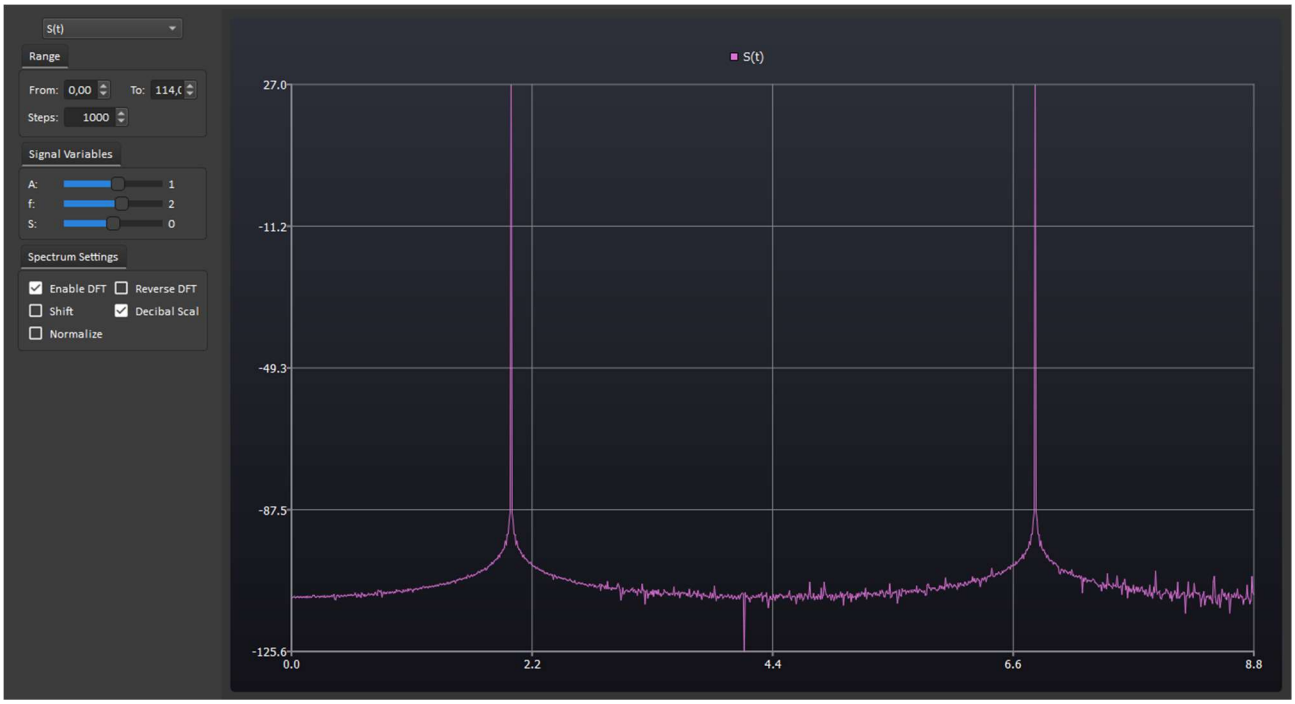
Wartość amplitudy przedstawić w sakli decybelowej $M'(k) = 10 \cdot \log_{10} M(k)$

Wyznacz skalę częstotliwości $f_k = k \cdot \frac{f_s}{N}$.

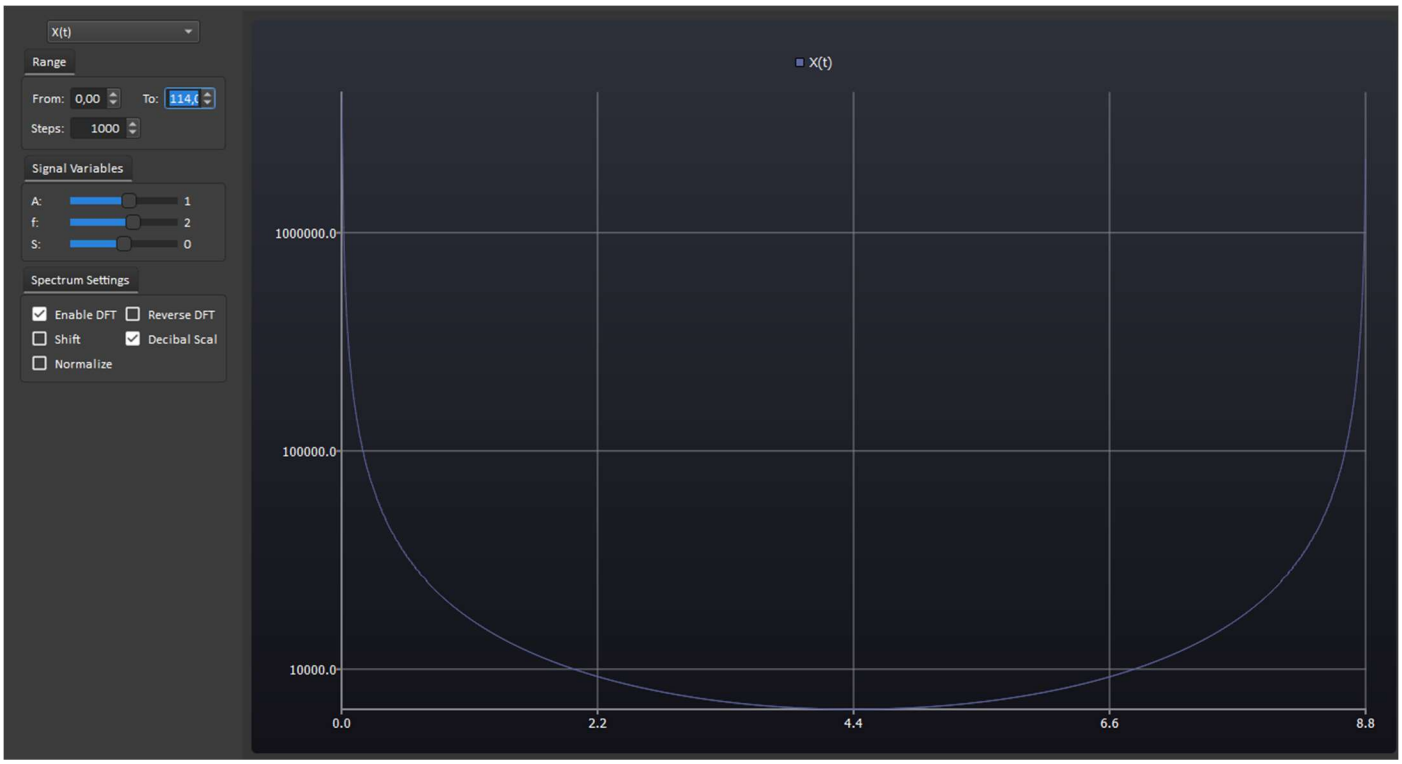
Wykreślić wykres widma amplitudowego M' (k), (wartosci f_k oznaczają częstotliwości prążków widma.

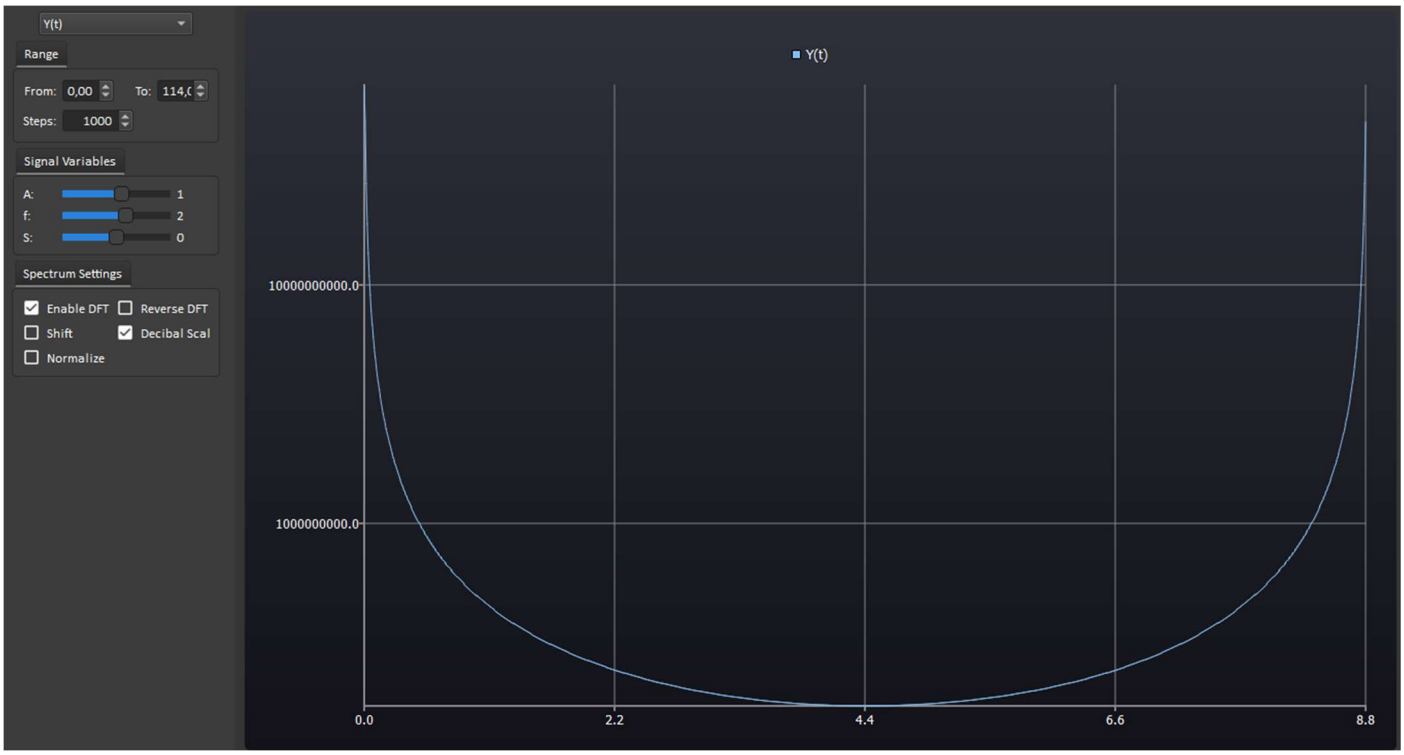


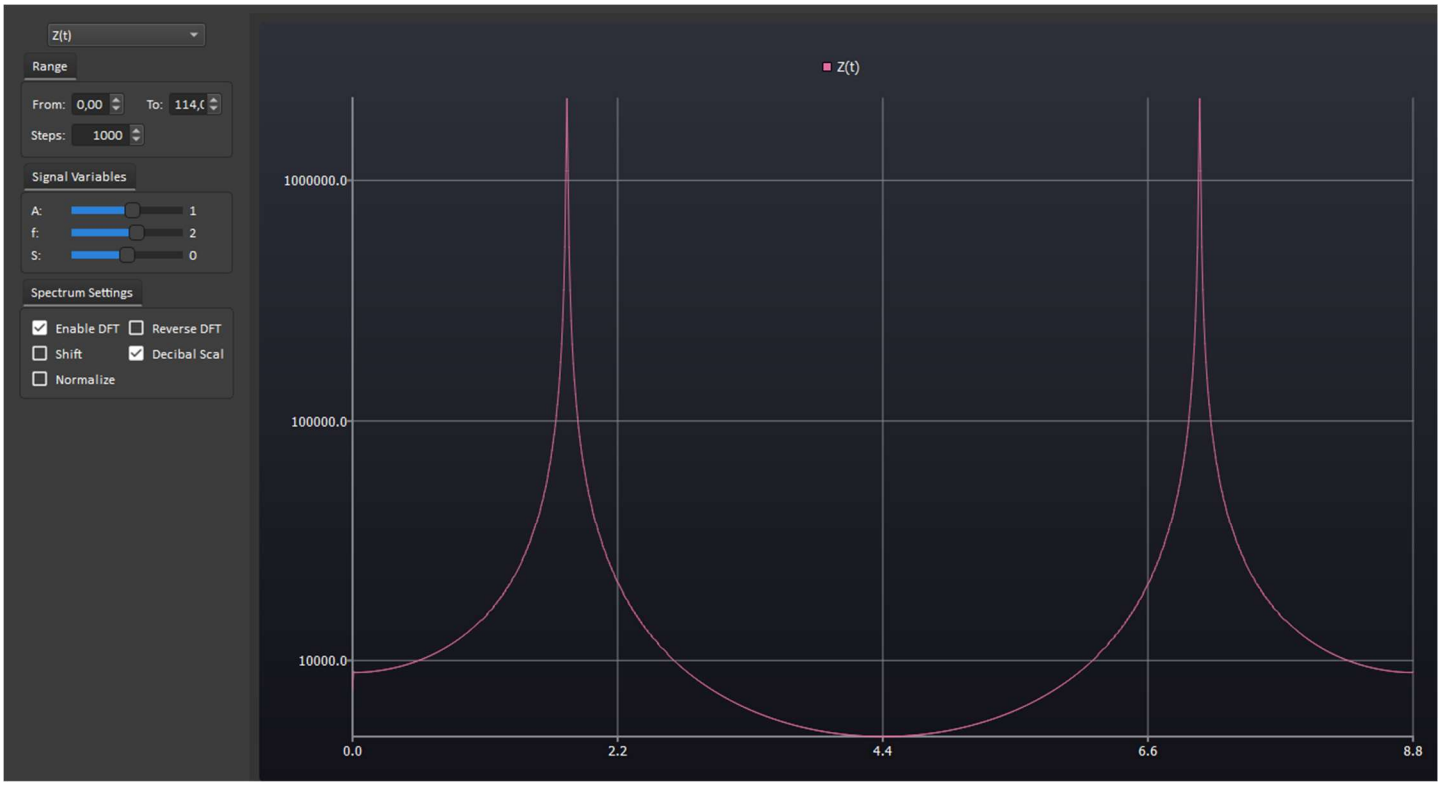

```cpp
void Lab3::calculateSeries(std::function<double(double)> foo)
{
    double rangeF = rangeFrom->value();
    double rangeT = rangeTo->value();
    int stepsVal = steps->value();
    double step = (rangeT-rangeF)/stepsVal; // deltaT
    double fs = 1/step;

    QVector<double> temp;
    for(double x = rangeF; x<=rangeT; x+=step)
    {
        temp.append(foo(x));
    }
    QVector<std::complex<double>> results = calculateDFT(temp);
    for(int k = 0; k<results.length(); k++)
    {
        double y = abs(results.at(k));
        y=10*log10(y);
        double x = (k*(fs/stepsVal));
        series->append(x, y);
    }
```

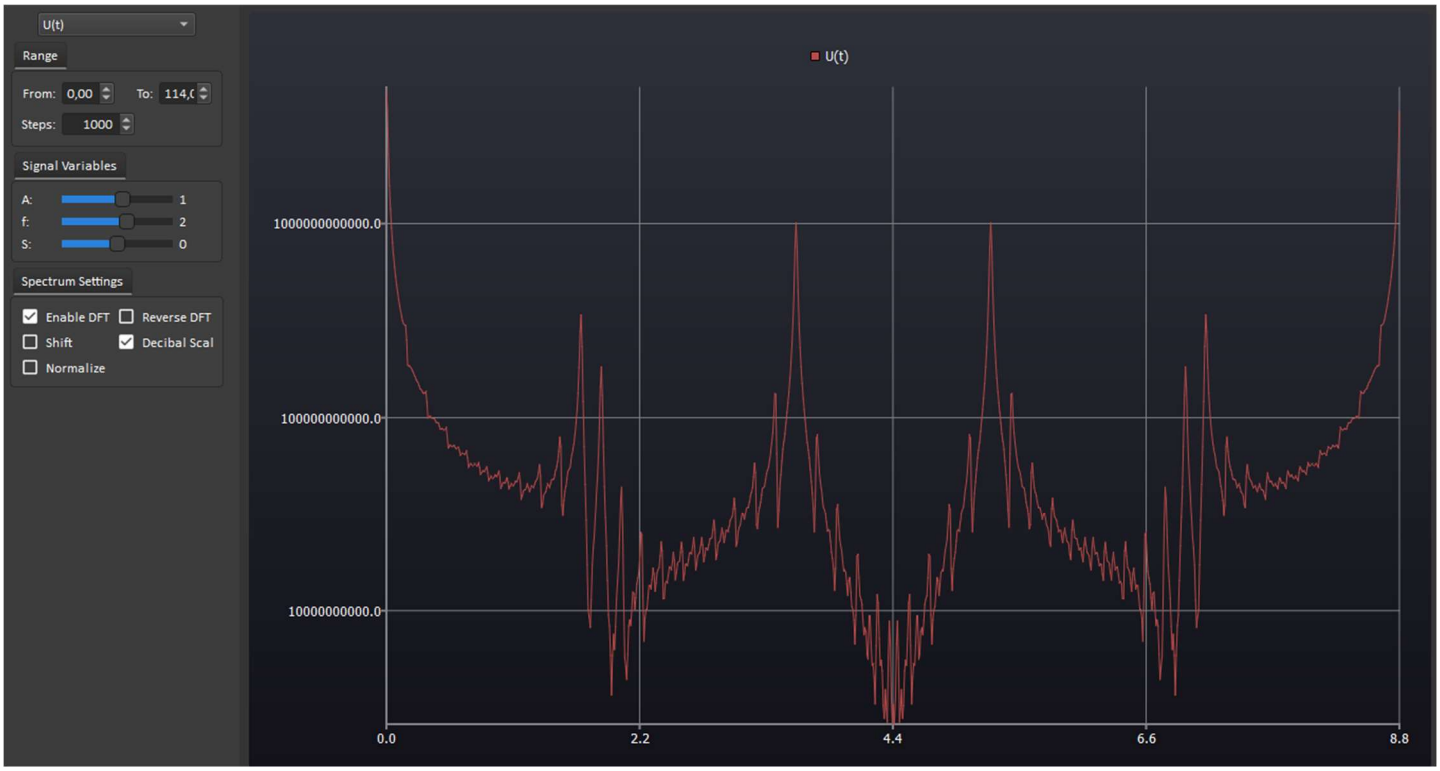


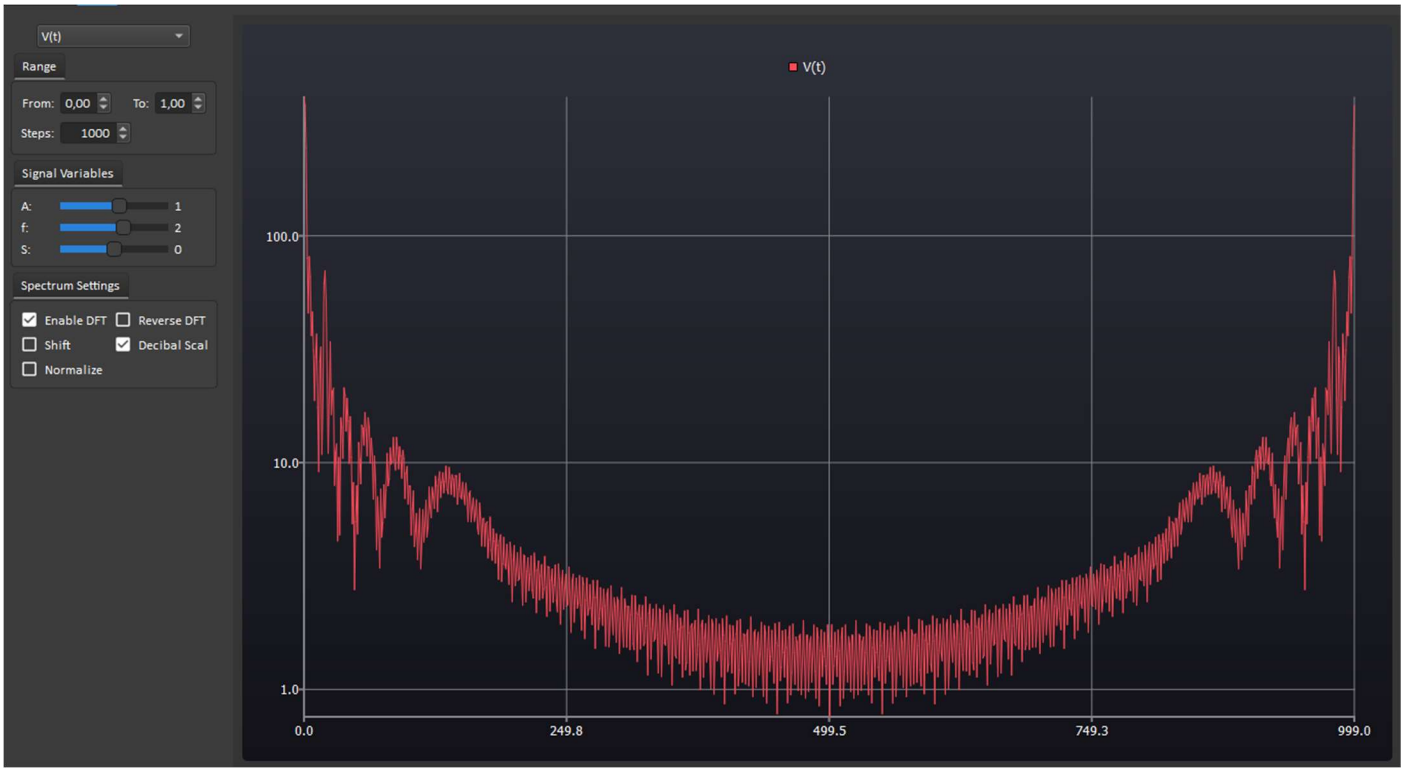
3) Dla sygnałów uzyskanych na pierwszych laboratoriach obliczyćwidma amplitudowe . Należy tak dobrać skale (liniową lub logarytmiczną) osi pionowych i poziomych aby jak najwięcej prążków widma było widocznych na wykresie.

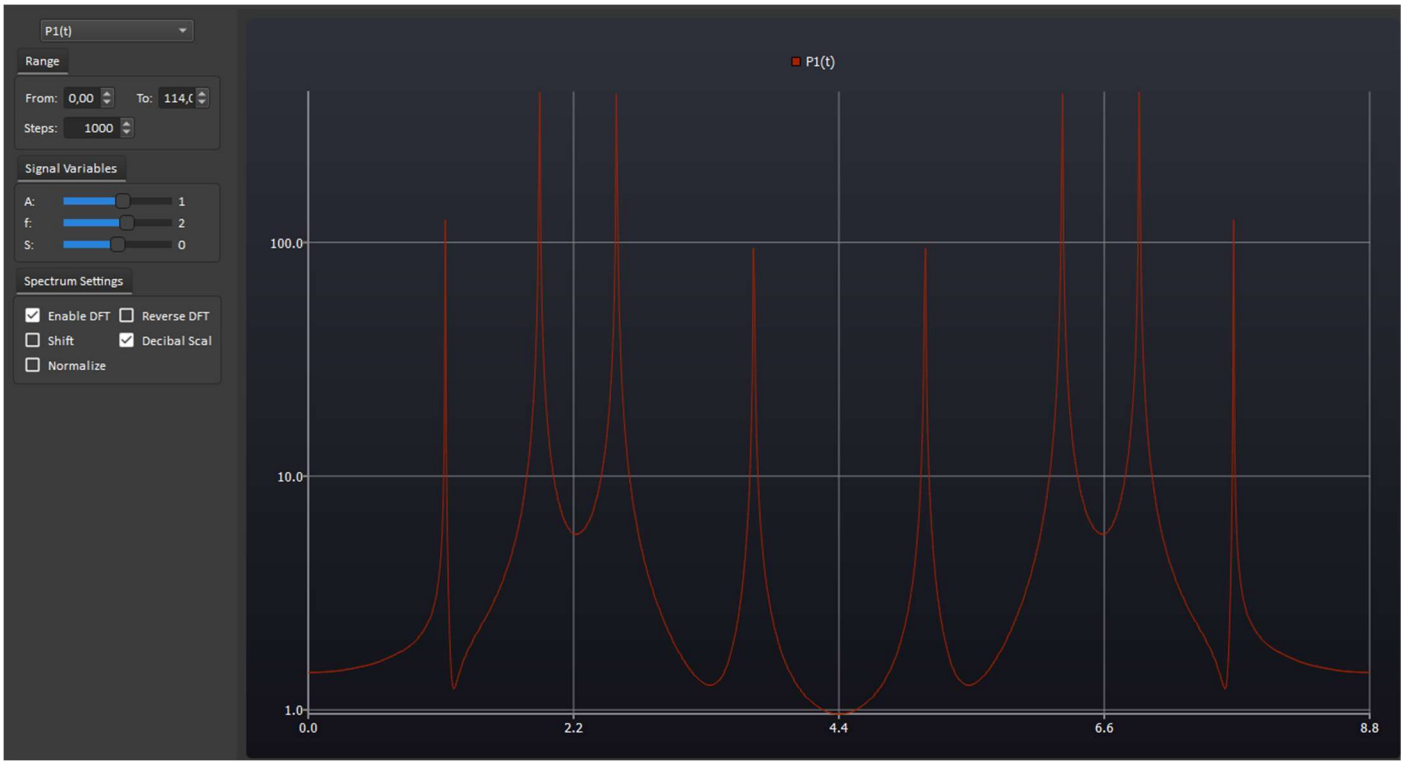


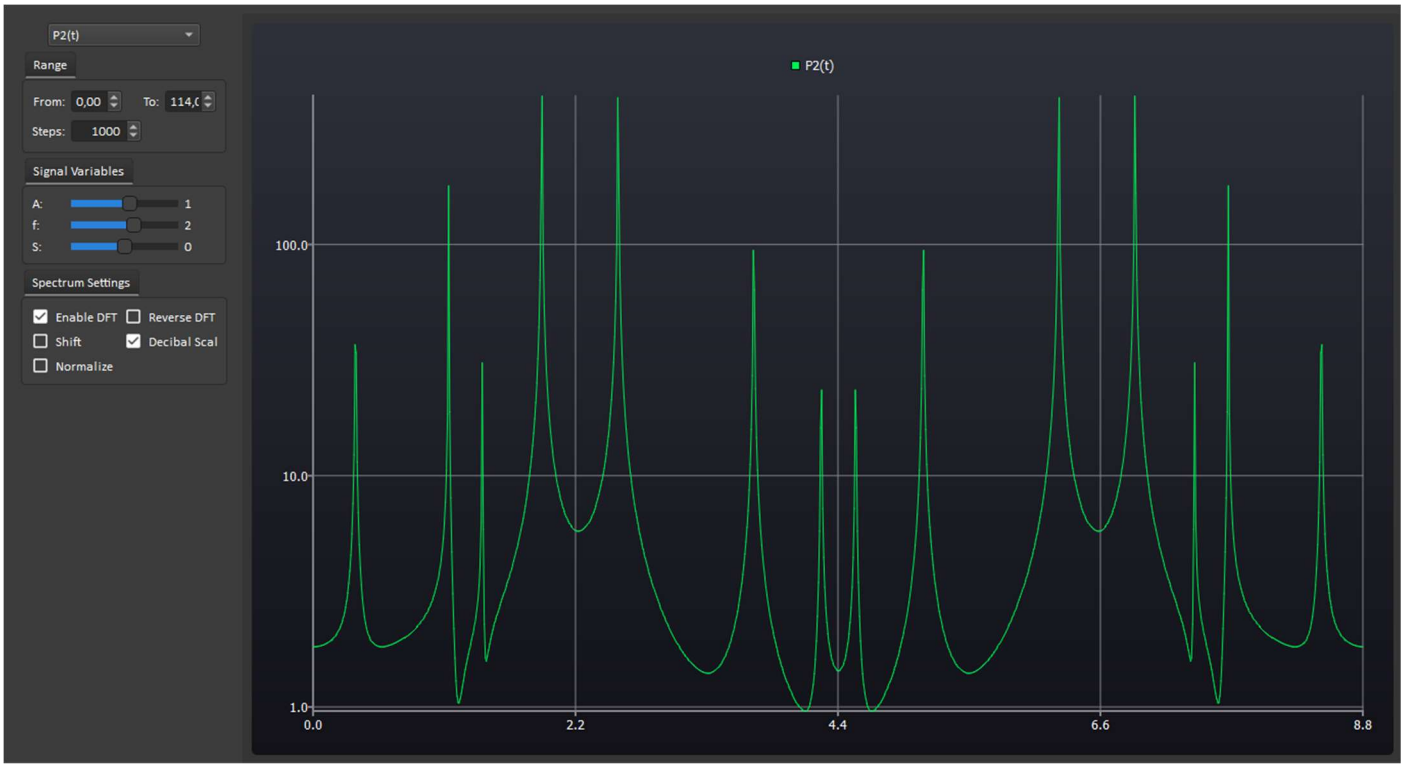


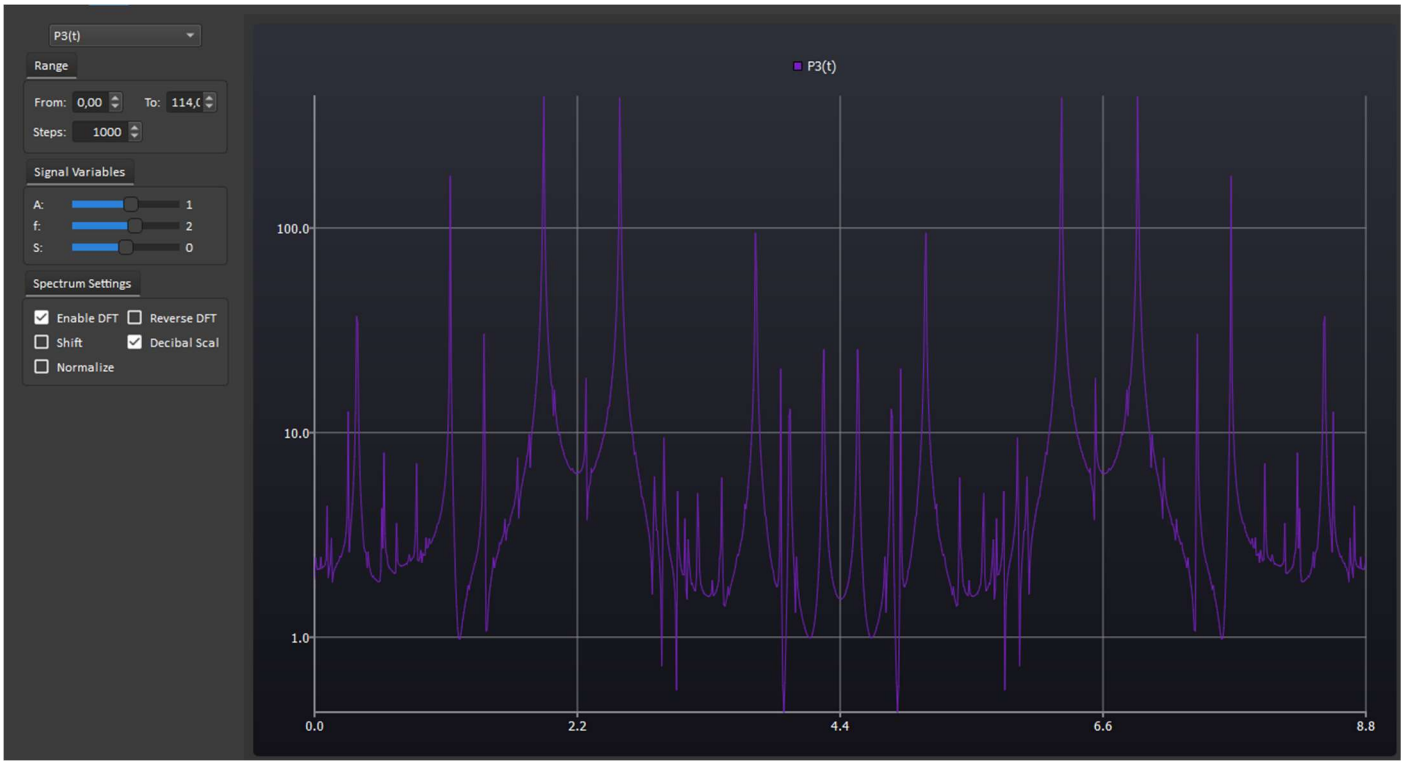

Z(t)
Range
From: 0,00
To: 114,(
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
Z(t)
1000000.0
100000.0
10000.0
0.0
2.2
4.4
6.6
8.8


U(t)
Range
From: 0,00
To: 114,(
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
U(t)
1000000000000.0
100000000000.0
10000000000.0
0.0
2.2
4.4
6.6
8.8


V(t)
Range
From: 0,00
To: 1,00
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
V(t)
100.0
10.0
1.0
0.0
249.8
499.5
749.3
999.0


P1(t)
Range
From: 0,00
To: 114,0
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
P1(t)
100.0
10.0
1.0
0.0
2.2
4.4
6.6
8.8


P2(t)
Range
From: 0,00
To: 114,(
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
P2(t)
100.0
10.0
1.0
0.0
2.2
4.4
6.6
8.8

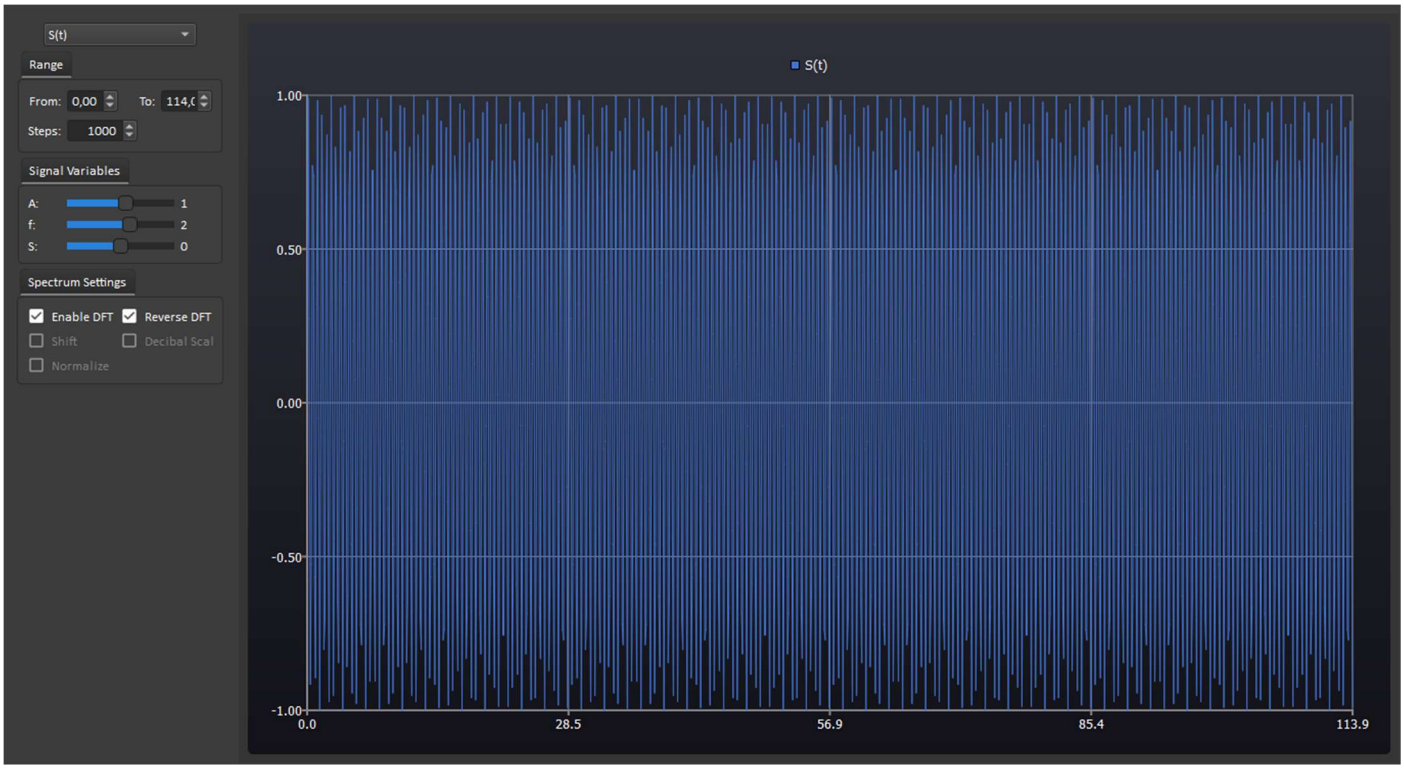

P3(t)
Range
From: 0,00
To: 114,(
Steps: 1000
Signal Variables
A: 1
f: 2
S: 0
Spectrum Settings
Enable DFT
Reverse DFT
Shift
Decibal Scal
Normalize
P3(t)
100.0
10.0
1.0
0.0
2.2
4.4
6.6
8.8

4) Napisz funkcję realizującą Odwrotną Dyskretną Transformatę Fouriera.

Zweryfikuj poprawność jej działania odwracając sygnał z dziedziny częstotliwości do dziedziny czasu (wykorzystaj sygnał użyty w zadaniu drugim).

```
QVector<double> Lab3::reverseDFT(QVector<std::complex<double> > dft)
{
    int steps = dft.length();
    QVector<double> temp;
    for(int n = 0; n<steps; n++)
    {
        double sum = 0;
        for(int k = 0; k<steps; k++)
        {
            double phase = (2*M_PI*k*n)/steps;
            sum += cos(phase) * dft[k].real() - sin(phase) * dft[k].imag();
        }
        temp.append(sum/steps);
    }
    return temp;
}
```

```
if(reverse->checkState()==2)
{
    auto revResults = reverseDFT(results);
    for(double x = rangeF; x<=rangeT; x+=step)
    {
        series->append(x,revResults.at(qRound(x/step)));
    }
}
```



Poniżej funkcje wykorzystane do wykonania wykresów:

```
double Lab3::sFunction(double x)
{
    double amp = static_cast<double>(amplitude->value())/100;
    double freq = static_cast<double>(frequency->value())/100;
    double pShift = static_cast<double>(phaseShift->value())/100;
    return amp*sin(2*M_PI*freq*x+pShift);
}

double Lab3::xFunction(double x)
{
    return 1*x*x+1*x+4;
}

double Lab3::yFunction(double x)
{
    return 2*(xFunction(x))*(xFunction(x))+12*qCos(x);
}

double Lab3::zFunction(double x)
{
    return qSin(2*M_PI*7*x)*xFunction(x) - 0.2*log10(abs(yFunction(x))+M_PI);
}

double Lab3::uFunction(double x)
{
    return sqrt(abs(yFunction(x)*yFunction(x)*zFunction(x)))
           - 1.8*sin(0.4*x*zFunction(x)*xFunction(x));
}

double Lab3::vFunction(double x)
{
    if(x<0.22)
        return (1-7*x)*qSin((2*M_PI*x*10)/(x+0.04));
    else if(x>=0.22 && x<=0.7)
        return 0.63*x*qSin(125*x);
    else
        return qPow(x, -0.662)+0.77*qSin(8*x);
}

double Lab3::p1Function(double x)
{
    double y=0;
    for(int number=1; number<=2; number++)
    {
        y += (qCos(12*x*number*number) + qCos(16*x*number))/(number*number);
    }
    return y;
}
```

```
double Lab3::p2Function(double x)
{
    double y=0;
    for(int number=1; number<=4; number++)
    {
        y += (qCos(12*x*number*number) + qCos(16*x*number))/(number*number);
    }
    return y;
}

double Lab3::p3Function(double x)
{
    double y=0;
    for(int number=1; number<=99; number++)
    {
        y += (qCos(12*x*number*number) + qCos(16*x*number))/(number*number);
    }
    return y;
}
```