1. Funkcje

1. $f_1(x) = cos(x)cosh(x) - 1, [\frac{3}{2}\pi, 2\pi]$

```
long double f1(long double x){
    return cos(x)*cosh(x)-1;
}

long double f1_derivative(long double x){
    return -cosh(x)*sin(x) + cos(x)*sinh(x);
}
```

2. $f_2(x) = \frac{1}{x} - tan(x), [0, \frac{\pi}{2}]$

```
long double f2(long double x){
    return 1/x - tan(x);
}

long double f2_derivative(long double x){
    return -1/pow(x,  y: 2) - pow( x: 1/cosh(x),  y: 2);
}
```

3. $f_3(x) = 2^{-x} + e^x + 2cos(x) - 6, [1, 3]$

```
long double f3(long double x){
    return pow( x: 2.0, -x) + exp(x) + 2*cos(x) - 6;
}

long double f3_derivative(long double x){
    return pow( x: 2.0, -x)*log( x: 2) + exp(x) - 2*sin(x);
}
```

2. Metoda bisekcji

a. Kod

```
long double bisection_method(long double (*f)(long double), long double from, long double to, long double abs_err) {
    long double x1 = (from + to) / 2;
    long int i = 0;
    while (abs((*f)(x1)) > abs_err) {
        if ((*f)(from) * (*f)(x1) < 0) {
            to = x1;
        } else {
            from = x1;
        }
        long double last_x1 = x1;
        x1 = (from + to) / 2;
        if(last_x1==x1){
            cout<<"Precision is to small.\n";
            break;
        }
        i++;
    }
    cout << "Ilosc iteracji: " << i << endl;
    return x1;
}
```

b. Wyniki

```
~~~~~~~~F1~~~~~~~~
-----BISECTION-METHOD-----
Precision: 1e-07
Ilosc iteracji: 28
4.73004
Precision: 1e-15
Ilosc iteracji: 54
4.73004
Precision: 1e-33
Precision is to small.
Ilosc iteracji: 60
4.73004
```

```
~~~~~~~~F3~~~~~~~~
-----BISECTION-METHOD-----
Precision: 1e-07
Precision is to small.
Ilosc iteracji: 62
6.28319
Precision: 1e-15
Precision is to small.
Ilosc iteracji: 62
6.28319
Precision: 1e-33
Precision is to small.
Ilosc iteracji: 62
6.28319
```

```
~~~~~~~~F2~~~~~~~~
-----BISECTION-METHOD-----
Precision: 1e-07
Precision is to small.
Ilosc iteracji: 62
4.71239
Precision: 1e-15
Precision is to small.
Ilosc iteracji: 62
4.71239
Precision: 1e-33
Precision is to small.
Ilosc iteracji: 62
4.71239
```

c. Uzyskiwanie pierwszych k dodatnich pierwiastków:

Po pierwsze należy usunąć ujemną część przedziału podawanego do funkcji oraz dodać do parametrów funkcji liczbę reprezentującą liczbę znalezionych pierwiastków(przekazywanych przez referencje) oraz wartość poprzedniego x1. Następnie funkcje wykonywać rekurencyjnie dla obu przedziałów (po lewej i prawej x1), zaczynając zawsze od lewego przedziału. Jeśli wartość (x1-last_x1) jest równa zeru to wywołujemy brake, w przeciwnym wypadku sprawdzamy czy wartość abs(f(x1)) jest mniejsza od dokładności to zwiększamy ilość znalezionych pierwiastków o 1 i wypisujemy x1. Po powrocie lewego przedziału sprawdzamy czy znaleźliśmy odpowiednią ilość miejsc zerowych i jeśli tak to wywołujemy brake.

3. Metoda Newtona

a. Kod

```
long double newtons_method(long double (*f)(long double), long double (*f_derivative)(long double), int max_iters, long double epsilon){
    long double x0 = 0.1;
    long double xn;
    int i;
    for(i=0; i<max_iters; i++){
        xn = x0 - (f(x0))/f_derivative(x0);
        if(abs( x: x0-xn)<epsilon){
            x0=xn;
            break;
        }
        x0=xn;
    }
    cout << "Ilosc iteracji: " << i << endl;
    return x0;
}
```

b. Wyniki

```
-----NEWTONS-METHOD-----
Precision: 1e-07
Ilosc iteracji: 30
8.47747e-06
Precision: 1e-15
Ilosc iteracji: 30
8.47747e-06
Precision: 1e-33
Ilosc iteracji: 30
8.47747e-06
```

```
-----NEWTONS-METHOD-----
Precision: 1e-07
Ilosc iteracji: 100
0.866132
Precision: 1e-15
Ilosc iteracji: 100
0.866132
Precision: 1e-33
Ilosc iteracji: 100
0.866132
```

```
-----NEWTONS-METHOD-----
Precision: 1e-07
Ilosc iteracji: 100
0.866132
Precision: 1e-15
Ilosc iteracji: 100
0.866132
Precision: 1e-33
Ilosc iteracji: 100
0.866132
```

c. Zbieżność

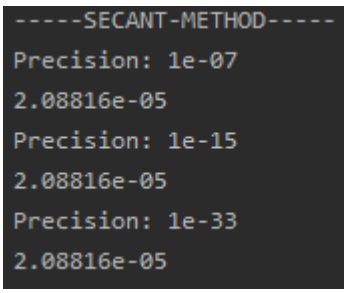
Zbieżność metody Newtona jest kwadratowa a metody bisekcji jest liniowa. Metoda liniowa ma dwukrotnie mniejszą szybkość zbieżności.

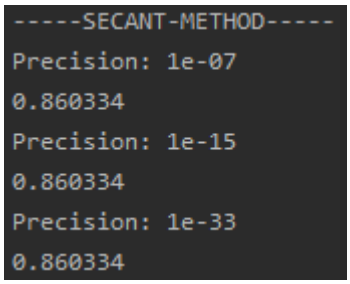
4. Metoda siecznych

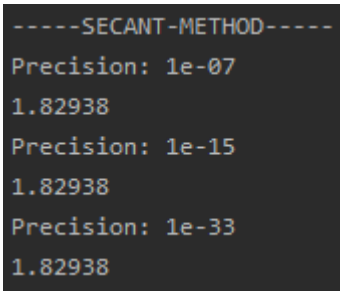
a. Kod

```
long double secant_method(long double (*f)(long double), int max_iters, long double epsilon){
    long double x0 = 0.1, x1 = 1, x2;
    for(int i=0; i<max_iters; i++){
        x2 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
        if(isinf(x2)){
            x2 = x1;
            break;
        }
        if(abs( x: x2-x1)<epsilon){
            break;
        }
        x0 = x1;
        x1 = x2;
    }
    return x2;
}
```

b. Wyniki







c. Zbieżność

Metoda siecznych ma zbieżność prawie kwadratową, jednak dla funkcji dostatecznie gładkich, wykładnik zbieżności spada do wartości (1 + 5^(½)) / 2 = 1.618. Ma ona zatem zbieżność większą od metody bisekcji oraz mniejszą od metody Newtona.