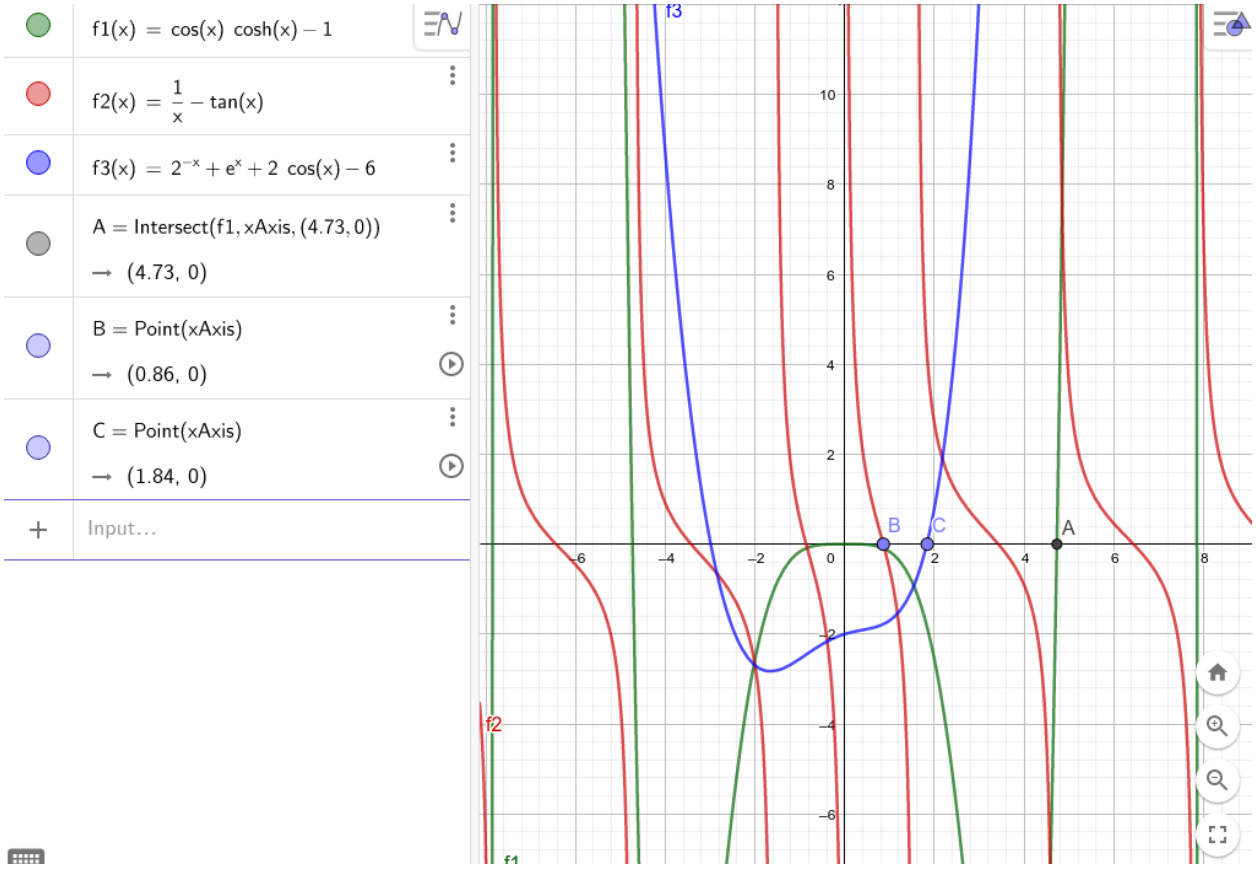
Funkcje do testów

1. $f_1(x) = \cos(x)\cosh(x) - 1,\ [\frac{3}{2}\pi, 2\pi]$
2. $f_2(x) = \frac{1}{x} - \tan(x),\ [0, \frac{\pi}{2}]$
3. $f_3(x) = 2^{-x} + e^x + 2\cos(x) - 6,\ [1, 3]$




```python
def f1(x):
    return mpmath.cos(x) * mpmath.cosh(x) - 1

def f2(x):
    return 1/x - mpmath.tan(x)

def f3(x):
    return mpmath.power(2, x) + mpmath.exp(x) + 2 * mpmath.cos(x) - 6
```

Pochodne funkcji przydatne w metodzie Newtona

```python
def f1_derivative(x):
    return mpmath.cos(x) * mpmath.sinh(x) + mpmath.sin(x) * mpmath.cosh(x)
```

```python
def f2_derivative(x):
    return mpmath.sec(x) - 1/(mpmath.power(x, 2))

def f3_derivative(x):
    return  mpmath.exp(x) - mpmath.power(2, -x)*mpmath.ln(2) - 2 * mpmath.sin(x)
```

1.Metoda bisekcji

Metoda bisekcji (inaczej połowienia przedziału lub równych podziałów)

Metoda służy do wyznaczenia miejsca zerowego danej funkcji i polega na cyklicznym połowieniu zadanego z góry przedziału (w którym znajduje się pierwiastek) aż do osiągnięcia zadanej dokładności.

Aby można było zastosować metodę, muszą być spełnione założenia:

1. funkcja f(x) jest ciągła w przedziale domkniętym [a; b]
2. funkcja przyjmuje różne znaki na końcach przedziału: f(a)f(b) < 0

Realizacja algorytmu

Uwaga do wszystkich metod: Została ustalona maksymalna ilość iteracji, aby rozwiązać problem, gdy precyzja nie pozwala na zbyt dokładne obliczenie miejsca zerowego

```python
def bisection(precision, start, end, abs_error, function = f1):
    number_iteration = 1000000
    mpmath.mp.dps = precision
    u = function(mpmath.mpf(start))
    v = function(mpmath.mpf(end))
    e = end - start
    # print(start, end, u, v)
    if mpmath.sign(u) == mpmath.sign(v):
        print("sgn(function(start)) = sgn(function(end))")
        return
    else:
        for k in range(1, number_iteration + 1):
            e = e / 2
            c = start + e
            w = function(c)
```

```
            # print("Iteration = {0}, center partition c = {1}, f(c) = {2},
length of 1/2 partition = {3}".format(k, c, w, e))
            if abs(w) < abs_error:
                print("Iteration = {0}, center partition c = {1}, f(c) = {2},
length of 1/2 partition = {3}"
                      .format(k, c, w, e))
                return #c, w, k
            if mpmath.sign(w) != mpmath.sign(u):
                end = c
                v = w
            else:
                start = c
                u = w
```

Test metody bisekcji dla określonych błędów bezwzględnych i precyzji

```
def test_bisection():
    epsilon_arr = [10e-8, 10e-16, 10e-34]
    prec_arr = [10, 25, 60]
    for prec in prec_arr:
        mpmath.mp.dps = prec
        for epsilon in epsilon_arr:
            epsilon = mpmath.mpf(epsilon)
            print(f"{bcolors.HEADER}precision = {prec} \t epsilon = {epsilon}
{bcolors.ENDC}")
            print(f"{bcolors.OKBLUE}function = cos(x)*cosh(x) - 1
{bcolors.ENDC}")
            bisection(prec, 3 * mpmath.pi / 2, 2 * mpmath.pi, epsilon, f1)
            print(f"{bcolors.OKBLUE}function = 1/x - tan(x) {bcolors.ENDC}")
            bisection(prec, 0 + epsilon, mpmath.pi / 2, epsilon, f2)
            print(f"{bcolors.OKBLUE}function = 2^x + e^x + 2*cos(x)
{bcolors.ENDC}")
            bisection(prec, 1, 3, epsilon, f3)
            print("")
```

Wynik testu:

```
/home/pawel/Desktop/Git/Mownit/venv5/bin/python /home/pawel/Desktop/Git/Mownit/lab5/lab5.py
preccision = 10      epsilon = 1.0e-7
function = cos(x)*cosh(x) - 1
Iteration = 29, center partition c = 4.730040746, f(c) = 6.465415936e-8, length of 1/2 partition = 2.925836159e-9
function = 1/x - tan(x)
Iteration = 24, center partition c = 0.8603336008, f(c) = -4.362664185e-8, length of 1/2 partition = 9.362675111e-8
function = 2^x + e^x + 2*cos(x)
Iteration = 24, center partition c = 1.0794812440872192, f(c) = 8.539063856e-8, length of 1/2 partition = 1.1920928955078125e-07

preccision = 10      epsilon = 1.0e-15
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
function = 2^x + e^x + 2*cos(x)
Iteration = 36, center partition c = 1.0794812118110713, f(c) = 0.0, length of 1/2 partition = 2.9103830456733704e-11

preccision = 10      epsilon = 1.0e-33
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
function = 2^x + e^x + 2*cos(x)
Iteration = 36, center partition c = 1.0794812118110713, f(c) = 0.0, length of 1/2 partition = 2.9103830456733704e-11

preccision = 25      epsilon = 9.999999999999999547481118e-8
function = cos(x)*cosh(x) - 1
Iteration = 29, center partition c = 4.730040745960150028204928, f(c) = 6.32628460663148821077060 9e-8, length of 1/2 partition = 2.92583615853431936210267 6e-9
function = 1/x - tan(x)
sgn(function(start)) = sgn(function(end))
function = 2^x + e^x + 2*cos(x)
Iteration = 24, center partition c = 1.0794812440872192, f(c) = 8.535218293144516400263538e-8, length of 1/2 partition = 1.1920928955078125e-07

preccision = 25      epsilon = 1.0000000000000000777054e-15
function = cos(x)*cosh(x) - 1
Iteration = 55, center partition c = 4.730040744862704020461049, f(c) = -3.20681981648453401321781 2e-16, length of 1/2 partition = 4.35983562251078987434905 1e-17
function = 1/x - tan(x)
sgn(function(start)) = sgn(function(end))
function = 2^x + e^x + 2*cos(x)
Iteration = 51, center partition c = 1.0794812118123174, f(c) = 8.74040785717381583373221 1e-16, length of 1/2 partition = 8.881784197001252e-16

preccision = 25      epsilon = 1.0000000000000000596731e-33
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
sgn(function(start)) = sgn(function(end))
function = 2^x + e^x + 2*cos(x)

preccision = 60      epsilon = 0.000000099999999999999954748111825886258685613938723690807819366455
function = cos(x)*cosh(x) - 1
Iteration = 29, center partition c = 4.730040745968150028204927777745322057664160555209774196296 94, f(c) = 0.000000863262846866314876920372337441127574177788067309229143071814 5, length
function = 1/x - tan(x)
Iteration = 24, center partition c = 0.860333600798209189302600799674224893503152008541040288850688, f(c) = -0.000000043606024994867307102906802196071114947598009231216709474038 7, length
function = 2^x + e^x + 2*cos(x)
Iteration = 24, center partition c = 1.0794812440872192, f(c) = 0.000000085352182931445164056449722682397912412205337084325732206315 3, length of 1/2 partition = 1.1920928955078125e-07

preccision = 60      epsilon = 0.000000000000001000000000000000777053998766610792383071856011950151454925 6
function = cos(x)*cosh(x) - 1
Iteration = 55, center partition c = 4.730040744862704020461048754814738392962546063636808775565904, f(c) = -0.000000000000000320681992749878701274937449803751953746489757821718505997201
function = 1/x - tan(x)
Iteration = 51, center partition c = 0.860333589019379994427577531218923331548321376160966614604563, f(c) = -0.000000000000000085867123825924294404493338003199024201654526460721816326049 7
function = 2^x + e^x + 2*cos(x)
Iteration = 51, center partition c = 1.0794812118123174, f(c) = 0.000000000000000874040785722700598821538731354040422104291820762231309796158, length of 1/2 partition = 8.881784197001252

preccision = 60      epsilon = 1.000000000000000055967309976241901934452242603237480063296 89e-33
function = cos(x)*cosh(x) - 1
Iteration = 112, center partition c = 4.730040744862704026024048100833884811297605332307942153549 43, f(c) = -4.957939302610917157980123759612558730611408097964135092666 42e-34, length of
function = 1/x - tan(x)
Iteration = 110, center partition c = 0.860333589019379762483893424137662524838864252982204428324743, f(c) = -7.086756532774061486442443702480871443685268754222409210163 91e-34, length o
function = 2^x + e^x + 2*cos(x)
```

Wnioski:

1)Gdy precyzja była mniejsza od błędu bezwzględnego funkcja nie zawsze była w stanie wyliczyć miejsce zerowe(co jest oczywiste)

2)Ilośc iteracji dla poszczególnych błędów była zgodna ze wzorem

$$n = \left\lceil \frac{\log \frac{b-a}{\varepsilon}}{\log 2} \right\rceil$$

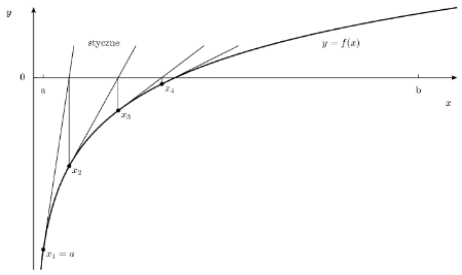
3)Wzrost dokładności pociąga za sobą wzrost liczby iteracji
4)Liczenie metodą bisekcji jest umiarkowanie szybkie wymagało maks 112 iteracji
5)Jest liniowo zbieżną metodą

2.Metoda Newtona

Opis metody

W metodzie Newtona przyjmuje się następujące założenia dla funkcji ***f***:

1. W przedziale **[*a*,*b*]** znajduje się dokładnie jeden pierwiastek.
2. Funkcja ma różne znaki na krańcach przedziału, tj. $f(a) \cdot f(b) < 0$.
3. Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.



Ilustracja działania metody Newtona, pokazane zostały 4 pierwsze kroki.

W pierwszym kroku metody wybierany jest punkt startowy $\mathbf{x_1}$ (zazwyczaj jest to wartość **a, b, 0** lub **1**), z którego następnie wyprowadzana jest styczna w $\mathbf{f(x_1)}$. Odcięta punktu przecięcia stycznej z osią OX jest pierwszym przybliżeniem rozwiązania (ozn. $\mathbf{x_2}$).

Jeśli to przybliżenie nie jest satysfakcjonujące, wówczas punkt $\mathbf{x_2}$ jest wybierany jako nowy punkt startowy i wszystkie czynności są powtarzane. Proces jest kontynuowany, aż zostanie uzyskane wystarczająco dobre przybliżenie pierwiastka

Kolejne przybliżenia są dane rekurencyjnym wzorem:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Realizacja algorytmu

```
def newton_method(precision, x0, abs_error, function = f1, der_function
=f1_derivative):
    number_iteration = 1000000
    mpmath.mp.dps = precision
    v = function(mpmath.mpf(x0))
    k = 0
    if abs(v) < abs_error:
        print("Iteration = {0}".format(k))
        return
    else:
        for k in range(1, number_iteration+1):
            x1 = x0 - v/(der_function(x0))
            v = function(x1)
            if abs(v) < abs_error:
                print("Iteration = {0}\t approximate value of the element =
{1}".format(k, v))
                return
            else:
                x0 = x1
```

Funkcja testująca

```
def test_newton_method():
    epsilon_arr = [10e-8, 10e-16, 10e-34]
    prec_arr = [10, 25, 60]
    for prec in prec_arr:
        mpmath.mp.dps = prec
        for epsilon in epsilon_arr:
            epsilon = mpmath.mpf(epsilon)
            print(f"{bcolors.HEADER}precision = {prec} \t epsilon = {epsilon}
{bcolors.ENDC}")

            print(f"{bcolors.OKBLUE}function = cos(x)*cosh(x) - 1\tx0 = 3*pi/2
{bcolors.ENDC}")
            newton_method(prec, 3 * mpmath.pi / 2,  epsilon, f1)
            print(f"{bcolors.OKBLUE}function = cos(x)*cosh(x) - 1\tx0 = 2*pi
{bcolors.ENDC}")
            newton_method(prec, 2 * mpmath.pi , epsilon, f1)

            print(f"{bcolors.OKBLUE}function = 1/x - tan(x) \t x0 = {epsilon}
{bcolors.ENDC}")
            newton_method(prec, epsilon,epsilon, f2, f2_derivative)
            print(f"{bcolors.OKBLUE}function = 1/x - tan(x) \t x0 = pi/2
{bcolors.ENDC}")
            newton_method(prec, mpmath.pi / 2, epsilon, f2, f2_derivative)

            print(f"{bcolors.OKBLUE}function = 2^x + e^x + 2*cos(x)\t x0 =
1{bcolors.ENDC}")
            newton_method(prec, 1, epsilon, f3, f3_derivative)
            print(f"{bcolors.OKBLUE}function = 2^x + e^x + 2*cos(x)\t x0 =
3{bcolors.ENDC}")
            newton_method(prec, 3, epsilon, f3, f3_derivative)

            print("")
```

Wyniki testu:

```
/home/pawel/Desktop/Git/Mownit/venv5/bin/python /home/pawel/Desktop/Git/Mownit/lab5/lab5.p
precision = 10   epsilon = 1.0e-7
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
Iteration = 59021    approximate value of the element = -1.553416951e-8
function = cos(x)*cosh(x) - 1   x0 = 2*pi
Iteration = 13149    approximate value of the element = -8.455390343e-8
function = 1/x - tan(x)      x0 = 1.0e-7
Iteration = 34   approximate value of the element = 6.640402717e-8
function = 1/x - tan(x)      x0 = pi/2
Iteration = 7    approximate value of the element = -7.788912626e-9
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 10   epsilon = 1.0e-15
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 1.0e-15
function = 1/x - tan(x)      x0 = pi/2
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 10   epsilon = 1.0e-33
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 1.0e-33
function = 1/x - tan(x)      x0 = pi/2
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3
```

```
precision = 25   epsilon = 9.999999999999999547481118e-8
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
Iteration = 2050     approximate value of the element = -8.740976244284294754698885e-8
function = cos(x)*cosh(x) - 1   x0 = 2*pi
Iteration = 2920     approximate value of the element = -6.225876372071657607521836e-10
function = 1/x - tan(x)      x0 = 9.999999999999999547481118e-8
Iteration = 34   approximate value of the element = 6.641500657467070988427197e-8
function = 1/x - tan(x)      x0 = pi/2
Iteration = 7    approximate value of the element = -7.790548036598485736271344e-9
function = 2^x + e^x + 2*cos(x)  x0 = 1
Iteration = 673111   approximate value of the element = 6.696376372563755298627 8e-8
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 25   epsilon = 1.000000000000000777054e-15
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
Iteration = 104034   approximate value of the element = -1.767128174109971150137837e-16
function = 1/x - tan(x)      x0 = 1.000000000000000777054e-15
Iteration = 62   approximate value of the element = -8.643221162808997058817677e-16
function = 1/x - tan(x)      x0 = pi/2
Iteration = 12   approximate value of the element = 6.659477035510535428220 04e-16
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 25   epsilon = 1.000000000000000005596731e-33
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 1.000000000000000005596731e-33
function = 1/x - tan(x)      x0 = pi/2
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3
```

```
precision = 60   epsilon = 0.000000099999999999999954748111825886258685613938723690807819366455
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
Iteration = 136  approximate value of the element = -0.000000082448200042654912998280749633367309960340054593131520041794
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 0.000000099999999999999954748111825886258685613938723690807819366455
Iteration = 34   approximate value of the element = 0.000000066415006574670709910434746648981887648773481046644534078251 5
function = 1/x - tan(x)      x0 = pi/2
Iteration = 7    approximate value of the element = -0.000000007790548036598485714322191562664118607207268265837962342974 72
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 60   epsilon = 0.000000000000001000000000000000777053998766610792383071856011950151454925 6
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 0.000000000000001000000000000000777053998766610792383071856011950151454925 6
Iteration = 62   approximate value of the element = -0.000000000000000864322116277185505372244736334780379306827314547124541370036
function = 1/x - tan(x)      x0 = pi/2
Iteration = 12   approximate value of the element = 0.000000000000000665947703552030575736389637928825274133876855632311615000234
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3

precision = 60   epsilon = 1.000000000000000055967309976241901934452242603237480063296 89e-33
function = cos(x)*cosh(x) - 1   x0 = 3*pi/2
function = cos(x)*cosh(x) - 1   x0 = 2*pi
function = 1/x - tan(x)      x0 = 1.000000000000000055967309976241901934452242603237480063296 89e-33
Iteration = 135  approximate value of the element = -5.075421213615567410577101087187144206407082210478130171692 73e-34
function = 1/x - tan(x)      x0 = pi/2
Iteration = 25   approximate value of the element = -2.794522586659178914764482862386882744757433370874583567717 24e-34
function = 2^x + e^x + 2*cos(x)  x0 = 1
function = 2^x + e^x + 2*cos(x)  x0 = 3
```

Wnioski:

1)Precyzja nie zawsze pozwala nam na obliczenie miejsca zerowego
2)W kilku przypadkach jest rozbieżna
3)Wymaga małej liczby iteracji
4)Jest bardzo szybką metodą(jest zbieżna kwadratowo)
5)Wymaga znajommości pochodnej funkcji

3.Metoda Siecznych

Metoda siecznych (metoda Eulera) – metoda numeryczna, służąca do rozwiązywania równania nieliniowego z jedną niewiadomą.

Metoda siecznych to algorytm interpolacji liniowej. W literaturze polskojęzycznej nazywana czasem bywa metodą cięciw[1]. Polega na przyjęciu, że funkcja ciągła na dostatecznie małym odcinku w przybliżeniu zmienia się w sposób liniowy. Możemy wtedy na odcinku $\langle a, b \rangle$ krzywą $y = f(x)$ zastąpić sieczną. Za przybliżoną wartość pierwiastka przyjmujemy punkt przecięcia siecznej z osią OX.

Metodę siecznych dla funkcji $f(x)$, mającej pierwiastek w przedziale $\langle a, b \rangle$ można zapisać następującym wzorem rekurencyjnym:

$$\begin{cases} x_0 = a \\ x_1 = b \\ x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} \end{cases}$$

Metoda siecznych ma tę zaletę, że do wykonania interpolacji za jej pomocą niepotrzebna jest znajomość pochodnej danej funkcji, gdyż przybliżamy ją za pomocą powyższego wzoru. Aby metoda się powiodła dla każdego n musi zachodzić $f(x_n)f(x_{n-1}) < 0$, gdyż tylko wtedy sieczna przechodząca przez punkty $(x_n, f(x_n))$ i $(x_{n-1}, f(x_{n-1}))$ przecina oś OX. Metoda ta nie zawsze jest zbieżna.

```
def secand_method(precision, start, end, abs_error, function = f1):
    number_iteration = 1000000
    mpmath.mp.dps = precision
    fstart = function(start)
    fend = function(end)
    for k in range(2, number_iteration+1):
        if abs(fstart) > abs(fend):
            start, end = end, start
            fstart, fend = fend, fstart
        if (fend -fstart) == 0:
            return
        else:
            s = (end - start)/(fend - fstart)
            end = start
            fend = fstart
            start = start - fstart * s
            fstart = function(start)
            if abs(fstart) < abs_error:
                print("Iteration = {0}\t approximate value of the element =
{1}".format(k-1, fstart))
                return
```

Test:

```
def test_secand_method():
    epsilon_arr = [10e-8, 10e-16, 10e-34]
    prec_arr = [10, 25, 60]
    for prec in prec_arr:
        mpmath.mp.dps = prec
        for epsilon in epsilon_arr:
            epsilon = mpmath.mpf(epsilon)
            print(f"{bcolors.HEADER}precision = {prec} \t epsilon = {epsilon}
{bcolors.ENDC}")

            print(f"{bcolors.OKBLUE}function = cos(x)*cosh(x) - 1
{bcolors.ENDC}")
            secand_method(prec, 3 * mpmath.pi / 2, 2 * mpmath.pi, epsilon, f1)
            print(f"{bcolors.OKBLUE}function = 1/x - tan(x) {bcolors.ENDC}")
            secand_method(prec, 0 + epsilon, mpmath.pi / 2, epsilon, f2)
            print(f"{bcolors.OKBLUE}function = 2^x + e^x + 2*cos(x)
{bcolors.ENDC}")
            secand_method(prec, 1, 3, epsilon, f3)

            print("")
```

Wynik testu:

```
precision = 10   epsilon = 1.0e-7
function = cos(x)*cosh(x) - 1
Iteration = 4    approximate value of the element = -2.929300535e-8
function = 1/x - tan(x)
Iteration = 27   approximate value of the element = 1.455191523e-11
function = 2^x + e^x + 2*cos(x)
Iteration = 5    approximate value of the element = 0.0

precision = 10   epsilon = 1.0e-15
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
function = 2^x + e^x + 2*cos(x)
Iteration = 5    approximate value of the element = 0.0

precision = 10   epsilon = 1.0e-33
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
function = 2^x + e^x + 2*cos(x)
Iteration = 5    approximate value of the element = 0.0

precision = 25   epsilon = 9.999999999999999547481118e-8
function = cos(x)*cosh(x) - 1
Iteration = 4    approximate value of the element = -2.869431456116665581933918e-8
function = 1/x - tan(x)
Iteration = 38   approximate value of the element = 2.681130232963893202808356e-10
function = 2^x + e^x + 2*cos(x)
Iteration = 5    approximate value of the element = 1.911079512250500852763732e-11
```

```
precision = 25   epsilon = 1.0000000000000000777054e-15
function = cos(x)*cosh(x) - 1
Iteration = 6    approximate value of the element = -3.27124082870580163296802 4e-23
function = 1/x - tan(x)
Iteration = 59   approximate value of the element = 1.16839261523115159075586 5e-23
function = 2^x + e^x + 2*cos(x)
Iteration = 6    approximate value of the element = -1.51692423575218067940679 9e-18

precision = 25   epsilon = 1.0000000000000000596731e-33
function = cos(x)*cosh(x) - 1
function = 1/x - tan(x)
function = 2^x + e^x + 2*cos(x)
Iteration = 7    approximate value of the element = 0.0

precision = 60   epsilon = 0.000000099999999999999954748111825886258685613938723690807819366455
function = cos(x)*cosh(x) - 1
Iteration = 4    approximate value of the element = -0.000000028694314561166653834265936807966156290762793011128339358646 7
function = 1/x - tan(x)
Iteration = 38   approximate value of the element = 0.000000000268113023296388126921907550116622687218088687799200869486788
function = 2^x + e^x + 2*cos(x)
Iteration = 5    approximate value of the element = 0.000000000019110795122505007826760485796002396425010757675536132978479 7

precision = 60   epsilon = 0.000000000000001000000000000000777053998766610792383071856011950151454925 6
function = cos(x)*cosh(x) - 1
Iteration = 6    approximate value of the element = -3.352380663791355634652259944270707632644764847881875574848 78e-23
function = 1/x - tan(x)
Iteration = 78   approximate value of the element = -6.264680800570469311278791630995440941259362987339974004902 77e-25
function = 2^x + e^x + 2*cos(x)
Iteration = 6    approximate value of the element = -0.000000000000000001516924170775711708277069263000842483954132956183825456110 37

precision = 60   epsilon = 1.000000000000000055967309976241901934452242603237480063296 89e-33
function = cos(x)*cosh(x) - 1
Iteration = 7    approximate value of the element = -3.916614241418818477049039460840252447146252210517505897441 79e-38
function = 1/x - tan(x)
Iteration = 143  approximate value of the element = 1.669010561991052403428873163314300600181233404500157629820 01e-48
function = 2^x + e^x + 2*cos(x)
Iteration = 8    approximate value of the element = 2.043132686190139451720409088511413745276290351646498384441 3e-48
```

Wnioski:

1)Precyzja nie zawsze pozwala nam na obliczenie miejsca zerowego
2)Wymagana ilość iteracji jest stosunkowa nieduża(wynosi około 1,6)
3)Jest stosunkowo szybka
4)W pewnych przypadkach nie jest zbieżna

Wnioski z wszystkich testów

1) Metoda bisekcji jest wolniejsza, ale daje nam pewność otrzymania dobrego wyniku
2)Najszybszą metodą jest metoda Newtona, druga to metoda siecznych, a najwolniejsza jest metoda bisekcji
3)Im szybsza metoda tym ma większe problemy może powodować
-metoda Newtona (czasami nie jest zbieżna, wymaga znajomości pochodnej)
-metoda siecznych(czasami nie jest zbieżna)
-metoda bisekcji(działa, ale najwolniejsza)