

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

def f1\_derivative(x):  
 return mpmath.cos(x) \* mpmath.sinh(x) + mpmath.sin(x) \* mpmath.cosh(x)  
  
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

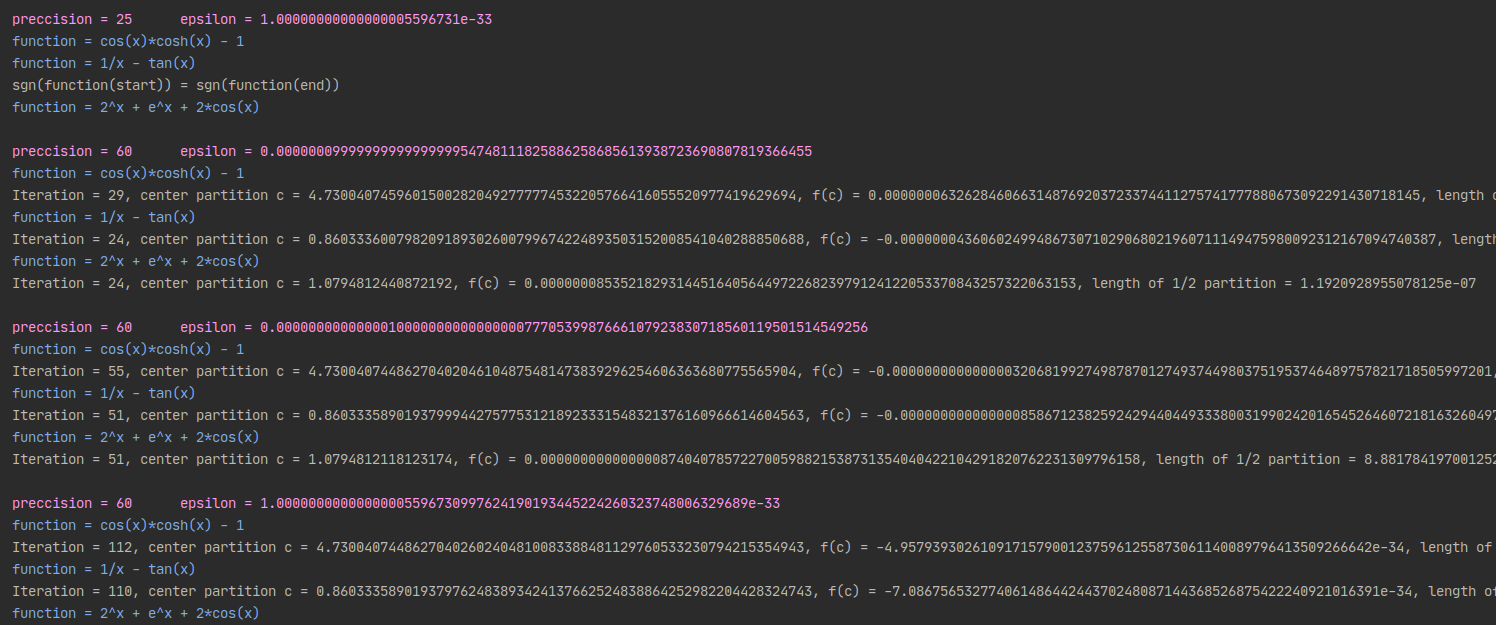
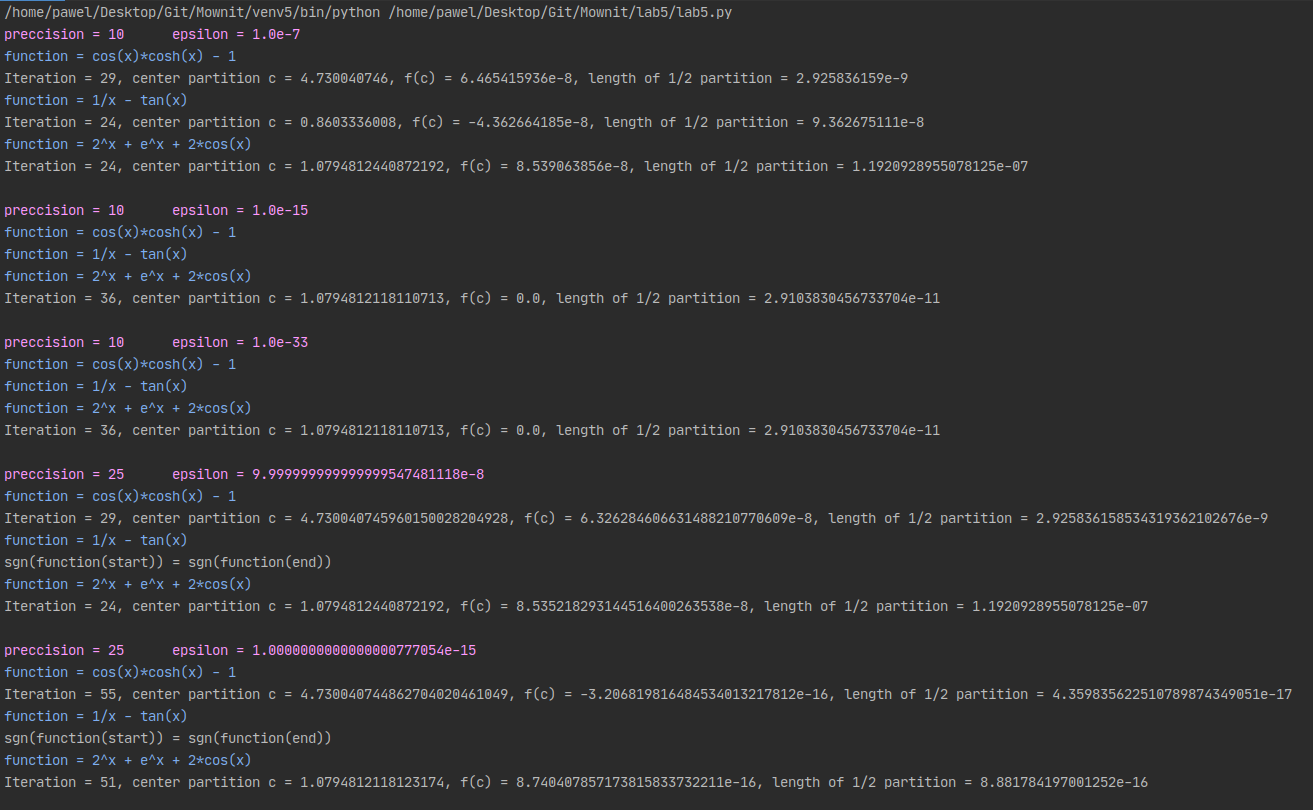
Uwaga do wszystkich metod: Została ustalona maksymalna ilość iteracji,  
aby rozwiązać problem, gdy precyzja nie pozwala na zbyt dokładne obliczenie miejszca zerowego

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

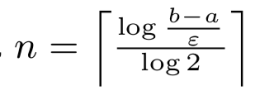
Test metody bisekcji dla określonych błędów bezwzględnych i precyzji

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:

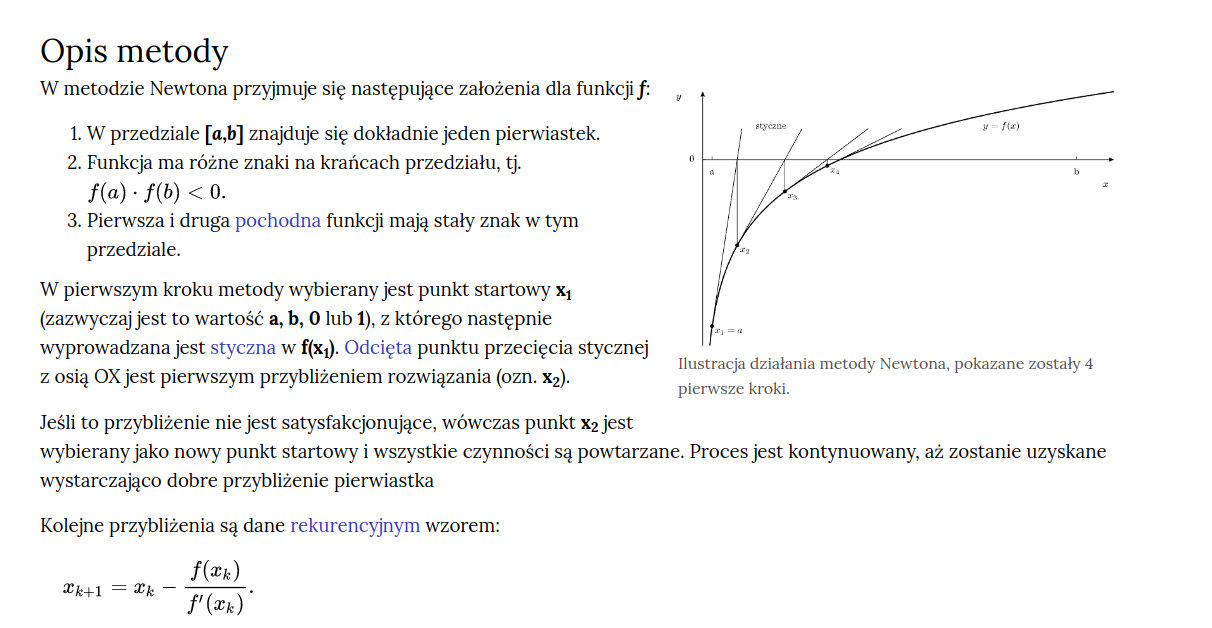


**Wnioski:**1)Gdy precyzja była mniejsza od błędu bezwzględnęgo funkcja nie zawsze była w stanie wyliczyć miejsce zerowe(co jest oczywiste)2)Ilośc iteracji dla poszczególnych błędów była zgodna ze wzorem



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**



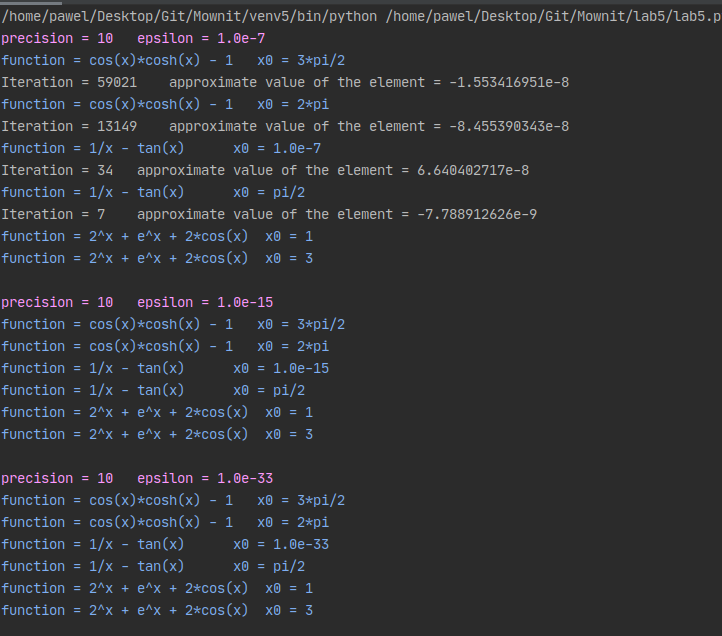
Realizacja algorytmu

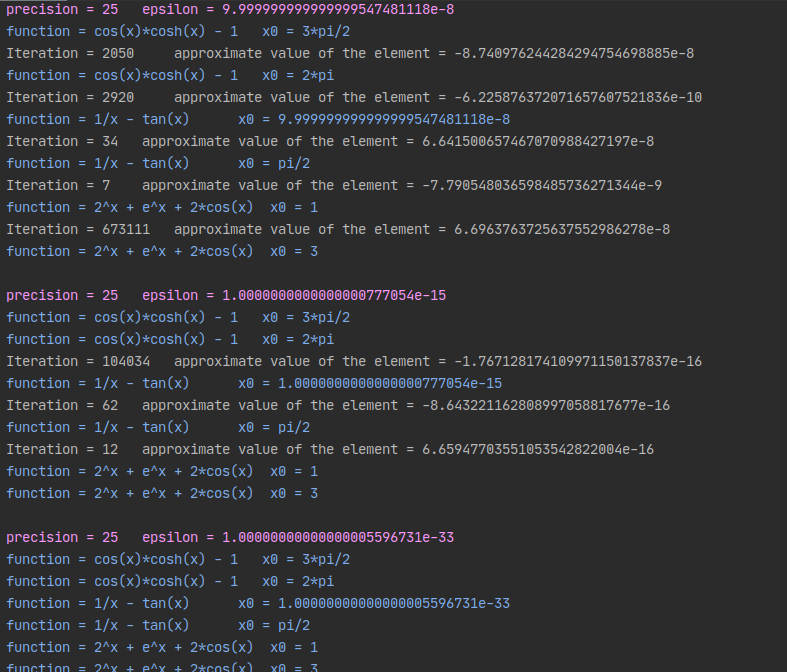
def newton\_method(precision, x0, abs\_error, function = f1, der\_function =f1\_derivative):  
 number\_iteration = 1000000  
 mpmath.mp.dps = precision  
 v = function(mpmath.mpf(x0))  
 k = 0  
 if abs(v) < abs\_error:  
 print("Iteration = {0}".format(k))  
 return  
 else:  
 for k in range(1, number\_iteration+1):  
 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

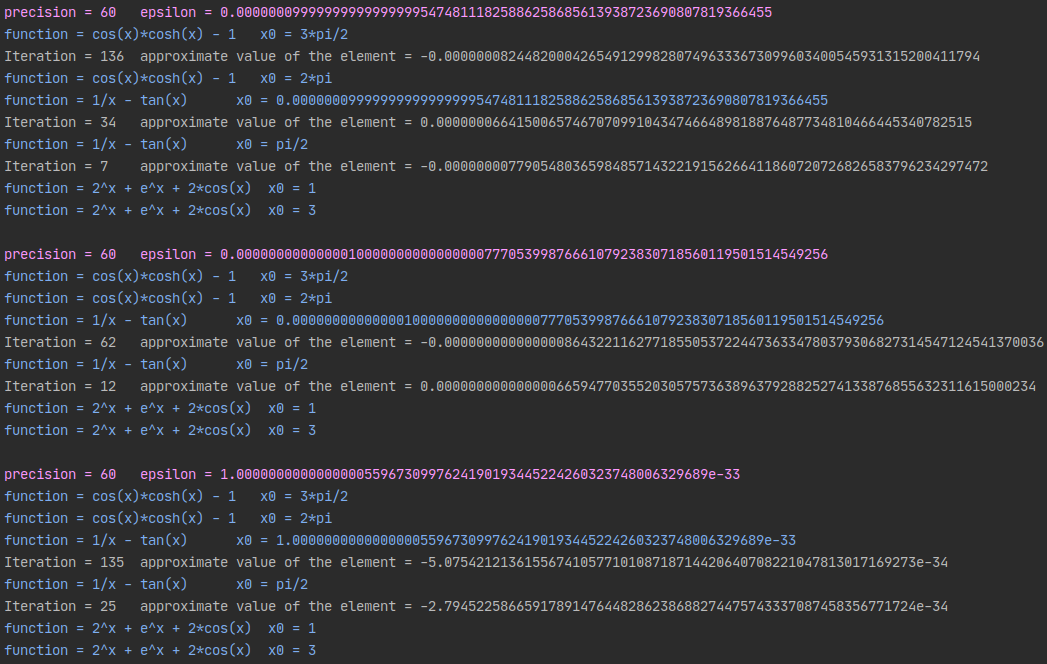
Funkcja testująca

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:

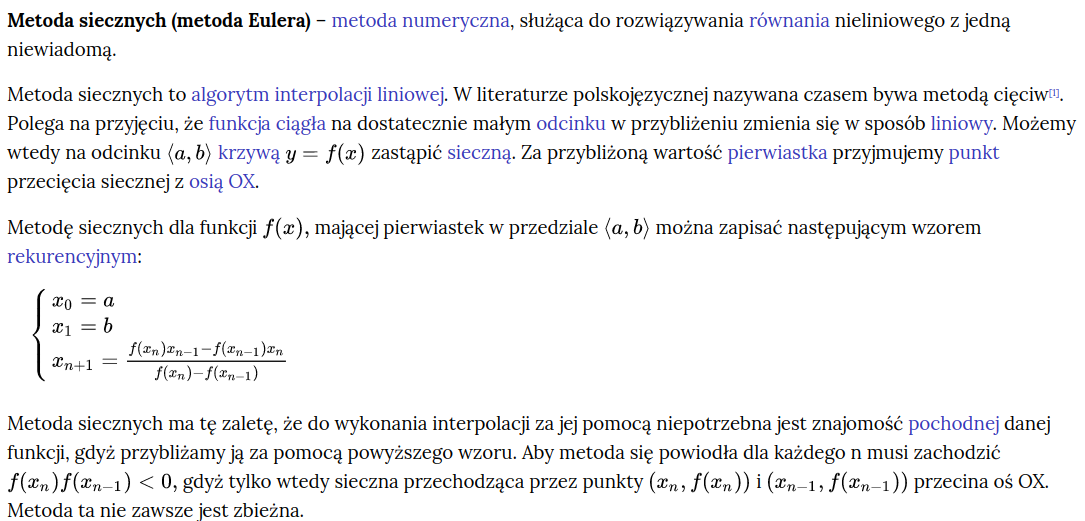






**Wnioski:**1)Precyzja nie zawsze pozwala nam na obliczenie miejsca zerowego  
2)W kilku przypadkach jest rozbieżna3)Wymaga małej liczby iteracji4)Jest bardzo szybką metodą(jest zbieżna kwadratowo)  
5)Wymaga znajommości pochodnej funkcji

**3.Metoda Siecznych**

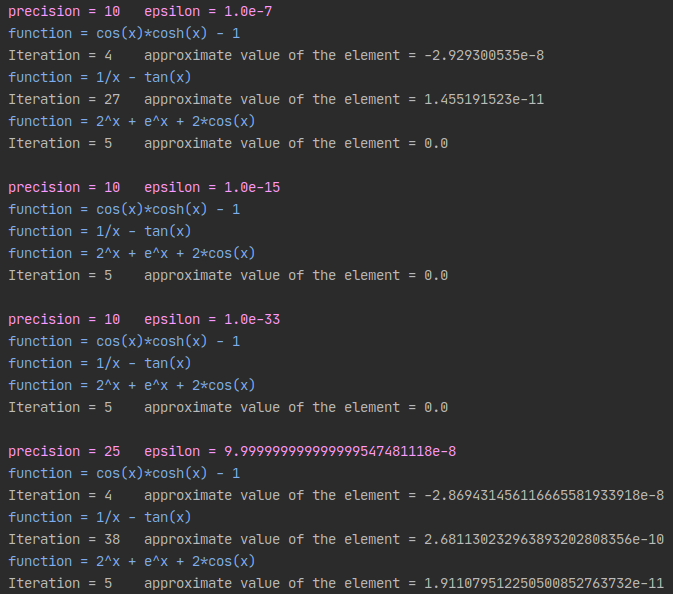


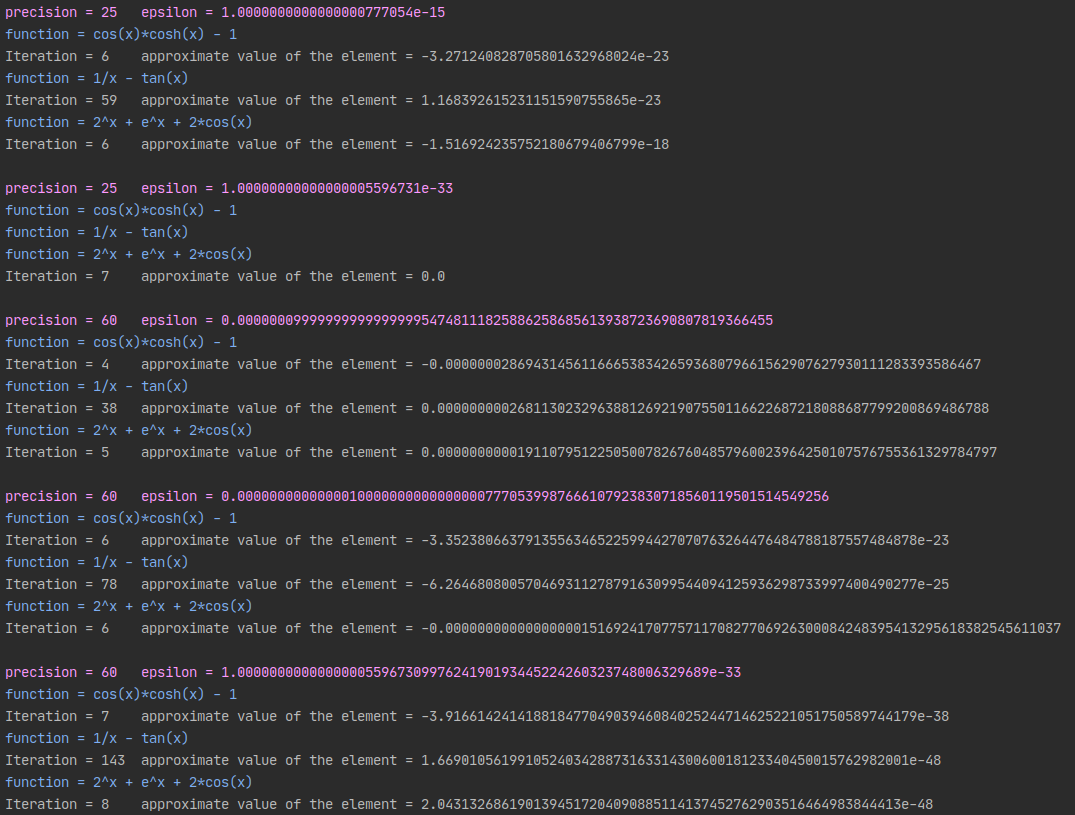
def secand\_method(precision, start, end, abs\_error, function = f1):  
 number\_iteration = 1000000  
 mpmath.mp.dps = precision  
 fstart = function(start)  
 fend = function(end)  
 for k in range(2, number\_iteration+1):  
 if abs(fstart) > abs(fend):  
 start, end = end, start  
 fstart, fend = 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:





**Wnioski:**1)Precyzja nie zawsze pozwala nam na obliczenie miejsca zerowego2)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 wyniku2)Najszybszą metodą jest metoda Newtona, druga to metoda siecznych, a najwolniejsza jest metoda bisekcji3)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)