Obliczenia Naukowe - laboratorium 3

Łukasz Machnik

20 listopada 2023

1 Zadanie 1

Zadanie polega na napisaniu funkcji rozwiązującej równanie $f(x) = 0$ metodą bisekcji. Funkcja ma być zdefiniowana następująco:

```
function mbisekcji(f, a::Float64, b::Float64, delta::Float64, epsilon::Float64)
```

Dane:

f	-	funkcja $f(x)$ zadana jako anonimowa funkcja
a, b	-	końce przedziału początkowego
delta, epsilon	-	dokładność obliczeń

Zwracany wynik: Czwórka (r, v, it, err), taka że:

r	-	przybliżenie pierwiastka równania $f(x) = 0$ w przedziale $[a, b]$
v	-	wartość $f(r)$
it	-	liczba wykonanych iteracji
err	-	sygnalizacja błędu 0 - brak błędu 1 - funkcja nie zmienia znaku w przedziale $[a, b]$

Funkcja ta działa jeśli f jest funkcją ciągłą i jeżeli $f(a)f(b) < 0$ - a zatem f zmienia znak w przedziale $[a, b]$, a zatem funkcja ma w tym przedziale zero.

Metoda bisekcji działa następująco: oblicza środek c przedziału $[a, b]$. Jeżeli $f(c) = 0$ to znaleźliśmy rozwiązanie. W przeciwnym przypadku jeżeli $f(a)$ i $f(c)$ mają różne znaki to w następnej iteracji przyglądamy się przedziałowi $[a, c]$. Jeżeli $sgn(f(a)) = sgn(f(c))$ to oznacza, że $sgn(f(c)) \neq sgn(f(b))$ a zatem w następnej iteracji przyglądamy się przedziałowi $[c, b]$.

Moja implementacja tego algorytmu prezentuje się następująco:

```
function mbisekcji(f, a::Float64, b::Float64, delta::Float64, epsilon::Float64)
    if(abs(f(a)) <= epsilon)
        return (a, f(a), 0, 0)
    elseif(abs(f(b)) <= epsilon)
        return (b, f(b), 0, 0)
    elseif(sign(f(a)) == sign(f(b)))
        return (0, 0, 0, 1)
    end
    it = 0
    while true
        c = (b - a) / 2 + a
        if b - a <= delta || abs(f(c)) <= epsilon
            return (c, f(c), it, 0)
        end
        it = it + 1
        if(sign(f(a)) != sign(f(c)))
            b = c
        else
            a = c
        end
    end
end
```

Na początku (*2, 4*) sprawdzamy czy zero funkcji (a właściwie odpowiednio bliska mu wartość) nie znajduje się na jednym z końców przedziału. Następnie (*6*) sprawdzamy czy końce przedziałów są różnych znaków - jeżeli nie to zwracamy kod błędu 1. W linijce *9* inicjalizujemy zmienną liczącą ile iteracji przeprowadziliśmy. Następnie w nieskończonej pętli powtarzamy następujące czynności: obliczamy środek przedziału $[a, b]$ (*11*). Warto zauważyć, że nie robimy tego wykonując działanie $\frac{a+b}{2}$ ponieważ takie działanie mogłoby w wyjątkowych przypadkach sprawić że wynik byłby nawet poza przedziałem $[a, b]$. Zamiast tego obliczamy połowę długości obecnego przedziału i dodajemy do początku tego przedziału - tak jest bezpieczniej. W linijce *12* sprawdzamy czy obecna długość przedziału jest mniejsza niż zadana przez użytkownika, albo czy wartość funkcji w połowie tego przedziału jest odpowiednio bliska 0. Jeżeli tak to zwracamy wynik i kończymy działanie funkcji, jeżeli nie to inkrementujemy licznik iteracji i sprawdzamy (*16*) czy znak zmienia się na przedziale $[a, c]$ czy $[c, b]$. Zmieniamy obecnie rozpatrywany przedział tak żeby na jego końcach zmieniał się znak i przechodzimy do kolejnej iteracji. Warto też zauważyć że różnice znaków na dwóch krańcach przedziału sprawdzamy za pomocą wbudowanej funkcji $sign()$ zamiast obliczając czy $f(a)f(c) < 0$ ponieważ takie niepotrzebne mnożenie mogłoby sprawić że otrzymamy niedomiar albo nadmiar, co mogłoby doprowadzić do wybrania błędnego przedziału. Należy pamiętać że aby funkcja działała poprawnie podane wartości delta i epsilon powinny być dodatnie - im będą mniejsze tym więcej może potencjalnie wykonać iteracji i dokładniejszy podać wynik. Algorytm oprócz późniejszych zadań przetestowałem również dla czterech różnych funkcji:

$$\begin{aligned} f(x) &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8) \\ g(x) &= tan(x) \\ h(x) &= log(x) \\ i(x) &= e^x - 1 \end{aligned}$$

Algorytm wywoływałem z $delta = epsilon = 0.00001$ otrzymując następujące wyniki (z zaokrągleniem do 4 liczb znaczących):

funkcja	x0	x1	r	x_z	$\lvert f(r)\rvert$	it
$f(x)$	3.25	4.6	4.0	4	$4.12 \cdot 10^{-5}$	18
$g(x)$	1.6	4.0	3.142	π	$8.909 \cdot 10^{-6}$	11
$h(x)$	0.0001	913.0	1.0	1	$4.656 \cdot 10^{-6}$	22
$i(x)$	-4.0	1024.0	$2.384 \cdot 10^{-7}$	0	$2.384 \cdot 10^{-7}$	23

Gdzie x_z to najbliższe znalezionego dokładne miejsce zerowe

2 Zadanie 2

Kolejne zadanie polega na napisaniu funkcji rozwiązującej równanie $f(x) = 0$ metodą Newtona (metodą stycznych). Funkcja ma być zdefiniowana następująco:

```
function mstycznych(f, pf, x0::Float64, delta::Float64, epsilon::Float64, maxit::Int)
```

Dane:

f, pf	-	funkcja $f(x)$ oraz pochodna $f'(x)$ zadane jako anonimowe funkcje
x0	-	przybliżenie początkowe
delta, epsilon	-	dokładność obliczeń
maxit	-	maksymalna dopuszczalna liczba iteracji

Zwracany wynik: Czwórka (r, v, it, err), taka że:

r	-	przybliżenie pierwiastka równania $f(x) = 0$
v	-	wartość $f(r)$
it	-	liczba wykonanych iteracji
err	-	sygnalizacja błędu 0 - brak błędu 1 - nie osiągnięto wymaganej dokładności w *maxit* iteracji 2 - pochodna bliska zeru

Algorytm ten wyznacza miejsce zerowe korzystając z pochodnej badanej funkcji. Zaczyna od zadanego przez użytkownika punktu $x0$, korzystając z pochodnej wyznacza styczną do wykresu w punkcie $(x0, f(x0))$. Podstawiając pod $x0$ miejsce zerowe wyznaczonej stycznej powtarza procedurę aż do osiągnięcia satysfakcjonującego wyniku. Moja implementacja tego algorytmu wygląda następująco:

```
function mstycznych(f, pf, x0::Float64, delta::Float64, epsilon::Float64, maxit::Int)
    v = f(x0)
    if abs(v) < epsilon
        return (x0, v, 0, 0)
    end
    for k = 1:maxit
        if(pf(x0) == 0)
            return (0, 0, k, 2)
        end
        x1 = x0 - v / pf(x0)
        v = f(x1)
        d = abs(x1 - x0)
        if(d < delta || abs(v) < epsilon)
            return(x1, v, k, 0)
        end
        x0 = x1
    end
    return (x0, v, maxit, 1)
end
```

Tak jak w poprzednim algorytmie tak i w tym na początku sprawdzamy czy wartość w punkcie początkowym nie jest już wystarczającym przybliżeniem miejsca zerowego (*3*). Następnie wykonując maksymalnie *maxit* iteracji wykonujemy następujące czynności: sprawdzamy czy pochodna w punkcie $x0$ nie jest równa 0 (*7*). Jeżeli tak to zwracamy kod błędu 2. W przeciwnym przypadku obliczamy punkt przecięcia z osią X stycznej do wykresu badanej funkcji w punkcie $(x0, f(x0))$ (*10*) i zapamiętujemy jako $x1$. Zapamiętujemy również jako v wartość funkcji w tym punkcie (*11*). Następnie jeżeli odległość nowo znalezionego $x1$ od $x0$ jest mniejsza na moduł niż zadana przez użytkownika *delta* albo jeżeli wartość funkcji w punkcie $x1$ jest mniejsza na moduł niż zadany *epsilon* (*13*) to zwracamy uzyskane wartości. W przeciwnym wypadku zamieniamy x0 na x1 (*16*) i przechodzimy do kolejnej iteracji. Po wyjściu z pętli - a zatem jeżeli nie uzyskaliśmy w zadanej liczbie iteracji odpowiednio precyzyjnych wyników zwracamy kod błędu 1 (*18*). Algorytm oprócz późniejszych zadań przetestowałem również dla tych samych czterech funkcji co w zadaniu 1:

$$\begin{aligned} f(x) &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8) \\ g(x) &= tan(x) \\ h(x) &= log(x) \\ i(x) &= e^x - 1 \end{aligned}$$

Oczywiście dla tego algorytmu musiałem również wyznaczyć pochodne tych funkcji:

$$\begin{aligned} f'(x) &= 4(2x^7 - 63x^6 + 819x^5 - 5670x^4 + 22449x^3 - 50463x^2 + 59062x - 27396) \\ g'(x) &= sec^2(x) \\ h'(x) &= \tfrac{1}{x} \\ i'(x) &= e^x \end{aligned}$$

Algorytm wywoływałem z $delta = epsilon = 0.00001$ oraz $maxit = 64$ otrzymując następujące wyniki (z zaokrągleniem do 4 liczb znaczących):

funkcja	x0	r	x_z	$\lvert f(r)\rvert$	it	err
$f(x)$	3.25	3.0	3	$4.201 \cdot 10^{-7}$	4	0
$g(x)$	1.6	3.142	π	$2.664 \cdot 10^{-7}$	8	0
$h(x)$	0.0001	1.0	1	$2.388 \cdot 10^{-9}$	9	0
$i(x)$	50.0	$1.219 \cdot 10^{-10}$	0	$1.219 \cdot 10^{-10}$	54	0
$i(x)$	100.0	36	0	$4.311 \cdot 10^{15}$	64	1

Gdzie x_z to najbliższe znalezionego dokładne miejsce zerowe. W wypadku tego algorytmu trzeba uważnie dobrać wartość początkową $x0$. Dla funkcji eksponencjalnej jeśli zaczniemy od zbyt dużego $x0$ (na przykład 100) to w 64 iteracjach zbliżając się powoli do wartości 0 dojdziemy tylko do $x0 = 36$. Jest to spowodowane faktem że funkcja eksponencjalna rośnie bardzo szybko więc styczne do jej wykresu są prawie pionowe dla dużych x a zatem zbliżanie się do 0 jest bardzo powolne. Dla niektórych funkcji algorytm może też w ogóle nie zbliżać się do poprawnego rozwiązania(dając rozbieżny ciąg potencjalnych rozwiązań i kończąc wykonywanie z kodem błędu 1). Problemem w zastosowaniu tego algorytmu jest też to że możemy napotkać na punkt w którym pochodna będzie równa 0 - wtedy algorytm zwróci błąd. Można zatem wywnioskować że użycie tego algorytmu nie jest wskazane dla wszystkich funkcji.

3 Zadanie 3

Ostatni z algorytmów rozwiązujących równanie $f(x) = 0$, który mamy zaimplementować to metoda siecznych. Funkcja ma być zdefiniowana następująco:

```
function msiecznych(f, x0::Float64, x1::Float64, delta::Float64, epsilon::Float64, maxit::Int)
```

Dane:

f	-	funkcja $f(x)$ zadana jako anonimowa funkcja
x0, x1	-	przybliżenia początkowe
delta, epsilon	-	dokładność obliczeń
maxit	-	maksymalna dopuszczalna liczba iteracji

Zwracany wynik: Czwórka (r, v, it, err), taka że:

r	-	przybliżenie pierwiastka równania $f(x) = 0$
v	-	wartość $f(r)$
it	-	liczba wykonanych iteracji
err	-	sygnalizacja błędu 0 - metoda zbieżna 1 - nie osiągnięto wymaganej dokładności w *maxit* iteracji

Algorytm ten powstał aby wyeliminować jeden z problemów metody stycznych mianowicie potrzebę wyznaczenia pochodnej funkcji. Tutaj zastępujemy pochodną w obliczeniach jej następującym przybliżeniem: $f'(x_n) \approx \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}$. W tym celu potrzebujemy jednak dwóch punktów początkowych, poza tym algorytm jest analogiczny, z kilkoma dodatkowymi zmianami:

```
1  function msiecznych(f, x0::Float64, x1::Float64, delta::Float64, epsilon::Float64, maxit::Int)
2      fa = f(x0)
3      if(abs(fa) < epsilon)
4          return (x0, fa, 0, 0)
5      end
6      fb = f(x1)
7      if(abs(fb) < epsilon)
8          return (x1, fb, 0, 0)
9      end
10     for k = 1:maxit
11         if(abs(fa) > abs(fb))
12             swap(x0, x1)
13             swap(fa, fb)
14         end
15         s = (x1 - x0)/(fb - fa)
16         x1 = x0
17         fb = fa
18         x0 = x0 - fa * s
19         fa = f(x0)
20         d = abs(x1 - x0)
21         if(d < delta || abs(fa) < epsilon)
22             return (x0, fa, k, 0)
23         end
24     end
25     return (x0, fa, maxit, 1)
26 end
```

Tutaj również (*2-9*) zaczynamy od sprawdzenia czy punkt zerowy nie znajduje się w dwóch podanych punktach początkowych. Jeżeli nie to maksymalnie *maxit* razy wykonujemy co następuje: upewniamy się że funkcja w punkcie $x0$ jest "bliżej" osi X niż w punkcie $x1$ (jeżeli nie to zamieniamy wartości odpowiednich zmiennych - dla przejrzystości w sprawozdaniu zastąpiłem zamianę tych wartości pojedynczą funkcją $swap(a, b)$)(*11-14*). Obliczamy odwrotność przybliżenia pochodnej w punkcie zgodnie z podanym wcześniej wzorem (odwrotność, bo potem w programie dzielilibyśmy przez pochodną więc dla ułatwienia już teraz liczę odwrotność a potem pomnożę przez tą wartość)(*15*). Następnie "przesuwam" obecnie badane punkty - pod $x1$ podstawiam wartość $x0$, a pod $x0$ wartość wyznaczoną za pomocą przybliżenia pochodnej (*16, 18*). Aktualizuję również odpowiednio wartości fa i fb (*17, 19*). Jeżeli odległość między nowymi rozważanymi punktami jest mniejsza

niż zadana przez użytkownika *delta*, lub jeżeli wartość funkcji w nowo wyznaczonym punkcie jest mniejsza niż podany *epsilon* (*21*) to zwracamy odpowiednie wartości(*22*), w przeciwnym przypadku przechodzimy do kolejnej iteracji. Jeżeli po wykonaniu *maxit* iteracji nie zakończymy działania funkcji to zwracamy kod błędu 1(*25*). Algorytm oprócz późniejszych zadań przetestowałem również dla tych samych czterech funkcji co w poprzednich zadaniach:

$$\begin{aligned} f(x) &= (x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8) \\ g(x) &= tan(x) \\ h(x) &= log(x) \\ i(x) &= e^x - 1 \end{aligned}$$

Wywołując algorytm z $delta = epsilon = 0.00001$ oraz $maxit = 64$ otrzymałem następujące wyniki (z zaokrągleniem do 4 liczb znaczących):

funkcja	x0	x1	r	x_z	$\lvert f(r)\rvert$	it	err
$f(x)$	3.25	3.5	3.0	3	$-5.343 \cdot 10^{-9}$	7	0
$g(x)$	1.6	1.5	$3.16 \cdot 10^{-8}$	0	$3.16 \cdot 10^{-8}$	11	0
$h(x)$	0.00001	0.0001	1.0	1	$-4.376 \cdot 10^{-6}$	12	0
$i(x)$	5.0	10.0	$4.854 \cdot 10^{-8}$	0	$4.854 \cdot 10^{-8}$	13	0
$i(x)$	10.0	25.0	10	0	22030	1	0

Zauważyć można że w przypadku tego algorytmu jeszcze uważniej trzeba dobierać punkty. Na przykład dla funkcji eksponencjalnej (która już wcześniej sprawiała problemy) nawet stosunkowo małe początkowe wartości $x0$ i $x1$ (odpowiednio 10 i 25) sprawiają (w połączeniu z faktem że funkcja ta rośnie bardzo szybko) że obliczony $x2$ jest równy $x0$ - jest to spowodowane utratą precyzji. Jako że $x2 - x0 = 0$ to algorytm w tym miejscu kończy działanie nie zwracając żadnego błędu a dając wartość bardzo daleką od miejsca zerowego. Dlatego algorytmu tego należy używać z dużą ostrożnością.

4 Zadanie 4

Zadanie polega na wyznaczeniu rozwiązania równania $f(x) = sin(x) - (\frac{1}{2}x)^2 = 0$ przy pomocy metod zaprogramowanych w zadaniach 1-3. W celu skorzystania z metody stycznych jest nam również potrzebna pochodna tej funkcji wyglądająca następująco: $f'(x) = cos(x) - \frac{x}{2}$. Dla każdej z metod użyłem $delta = epsilon = \frac{1}{2}10^{-5}$ a dla metod które tego wymagały użyłem $maxit = 64$. Uzyskane wyniki prezentują się następująco:

Metoda	$x0$	$x1$	r	$\lvert f(r)\rvert$	it
bisekcji	1.5	2	1.9337539672851562	$2.7027680138402843e-7$	15
stycznych	1.5	N/A	1.933753779789742	$2.2423316314856834e-8$	4
siecznych	1	2	1.933753644474301	$1.564525129449379e-7$	4

Jak widać wszystkie metody zwróciły bardzo podobne i bliskie dokładnemu miejscu zerowemu wyniki. Jednak metoda bisekcji potrzebowała do tego dużo więcej iteracji gdyż jej zbieżność jest liniowa w porównaniu do metody siecznych, której zbieżność jest nadliniowa i metody stycznych, której zbieżność jest kwadratowa. Wartość najbliższą 0 uzyskała natomiast metoda stycznych, jednak różnice - jak wspomniałem - są niewielkie.

5 Zadanie 5

W tym zadaniu należy znaleźć punkt przecięcia wykresów dwóch funkcji: $y = 3x$ i $y = e^x$ za pomocą metody bisekcji. W tym celu należy rozwiązać równanie $f(x) = e^x - 3x = 0$. W poleceniu została podana dokładność obliczeń której należy użyć: $delta = epsilon = 10^{-4}$. Tak prezentują się wyniki otrzymane w zależności od wartości początkowych $x0$ i $x1$:

$x0$	$x1$	r	$\lvert f(r)\rvert$	it	err
0.0	1.0	0.619140625	$9.066320343276146 \cdot 10^{-5}$	8	0
0.0	2.0	0	0	0	1
1.5	10.0	1.5121917724609375	$8.792167774984705 \cdot 10^{-5}$	14	0
1.0	100.0	1.5121002197265625	$5.274503124397256 \cdot 10^{-5}$	15	0

Dużą trudnością w tym zadaniu jest fakt że funkcja $f(x)$ jest ujemna na bardzo małym przedziale - mniej więcej od $x = 0.619$ do $x = 1.512$ - są to dwa miejsca zerowe. Zatem metodą prób i błędów może być trudno zgadnąć takie $x0$ i $x1$ żeby wartości funkcji w tych dwóch punktach były dwóch różnych znaków. Jest to zatem przykład zadania w którym metoda bisekcji nie jest najlepszym wyborem, chyba że znamy mniej więcej położenie dwóch miejsc zerowych.

6 Zadanie 6

Celem ostatniego zadania było znalezienie za pomocą każdej z trzech zaprogramowanych metod miejsc zerowych dwóch funkcji: $f_1(x) = e^{1-x} - 1$ oraz $f_2(x) = xe^{-x}$. Wamagana dokładność obliczeń to $delta = epsilon = 10^{-5}$. Maksymalną liczbę iteracji ustawiłem na 64. Obie te funkcje mają dokładnie jedno miejsce zerowe. Dla f_1 jest to $x_z = 1$, a dla f_2 jest to $x_z = 0$. Poniżej wyniki wykonanych obliczeń dla pierwszej funkcji:

f_1

Metoda	$x0$	$x1$	r	$\|f(r)\|$	it	err
bisekcji	2.0	4.0	0.0	0.0	0	1
	0.0	8.0	1.0	0.0	2	0
stycznych	0.0	N/A	1.0	0.0	4	0
	2.0	N/A	1.0	0.0	5	0
	8.0	N/A	NaN	NaN	64	1
siecznych	4.0	8.0	4.0	-0.95021	2	0
	2.0	4.0	1.0	-0.0	9	0

Jak widać każda metoda jest w stanie zwrócić poprawny wynik o ile dostanie odpowiednie dane początkowe. Dla metody bisekcji $x0$ i $x1$ muszą znajdować się po dwóch stronach miejsca zerowego - w przypadku tej konkretnej funkcji jest to warunek dość łatwy do spełnienia nawet dobierając wartości startowe "na ślepo". Dla metody stycznych, jako że funkcja jest eksponencjalna, nie można zacząć od zbyt dużej wartości: dla $x0 = 0 \vee x0 = 2$ w kilku iteracjach otrzymamy oprawny wynik, jednak dla $x0 = 8$ pętla wykona maksymalną możliwą liczbę iteracji i zwróci kod błędu. Jest to najprawdopodobniej spowodowane tym że wykonując działania na tak dużych wartościach przez błąd zaokrąglenia podzielimy w pewnym momencie przez 0 (bo styczne są prawie równoległe do osi X) otrzymując NaN. Metoda siecznych również nie może zacząć od zbyt dużych wartości $x0$ i $x1$. Jak widać w powyższej tabelce mimo że działanie programu dla $x0 = 4$ i $x1 = 8$ zakończy się szybko (bo po 2 iteracjach) to wynik będzie dość mocno odbiegał od prawdy. Tak samo jak przy metodzie stycznych jest to spowodowane faktem że funkcja ta od pewnego momentu jest prawie stała.

Poniżej tabela uzyskanych wyników dla drugiej funkcji:

f_2

Metoda	$x0$	$x1$	r	$\|f(r)\|$	it	err
bisekcji	2.0	4.0	0.0	0.0	0	1
	-2.5	13.0	-0.0	0.0	19	0
stycznych	-20.0	N/A	-0.0	0.0	27	0
	0.5	N/A	-0.0	0.0	5	0
	1.0	N/A	0.0	0.0	1	2
	2.0	N/A	14.39866	$1.0e-5$	10	0
siecznych	4.0	8.0	14.48124	$1.0e-5$	9	0
	-4.0	-8.0	-0.0	0.0	14	0

Tutaj również funkcje zwracają poprawny wynik tylko z odpowiednimi danymi startowymi. W przypadku metody bisekcji wystarczy że początek przedziału jest mniejszy, a koniec większy niż 0. Inaczej się ma sytuacja w przypadku metody stycznych. Tutaj niestety jeśli przyjmiemy $x0 = 1$ to już przy pierwszej iteracji pochodna w punkcie będzie zbyt bliska 0 i otrzymamy kod błędu. Jeżeli $x0$ będzie większe (n.p. $x0 = 2$) to również nie otrzymamy poprawnego wyniku. Kolejno wyliczone wartości $x0$ i $x1$ będą zbyt bliskie siebie i mimo braku wykrycia błędu program zwróci niepoprawny wynik. Dla wartości początkowych $x0 < 1$ zwracane wyniki wydają się być poprawne. W przypadku metody siecznych sytuacja wygląda podobnie. Nie napotkamy problemu z pochodną bliską 0 gdyż nie korzystamy z pochodnej, ale dla zbyt dużych wartości początkowych $x0$ i $x1$ napotkamy moment w którym nowe wartości $x0$ i $x1$ będą zbyt bliskie siebie i program zwróci wynik daleki od prawdziwego.

Końcowe wnioski są zatem następujące: wszystkie te funkcje są użyteczne i potrafią znaleźć miejsce zerowe. Jednak aby to zrobić trzeba wybrać im odpowiednie dane początkowe. Aby znaleźć takie dane początkowe można używać tych funkcji hybrydowo: na przykład za pomocą metody bisekcji znaleźć mniej więcej wartość miejsca zerowego, a następnie doprecyzować ją za pomocą metody stycznych lub siecznych. Wszystko zależy jednak od tego jak wygląda funkcja którą badamy