Analiza matematyczna dla informatyków.

Mieczysław Cichoń, ver. 4.3/2023

Mieczysław Cichoń - WMI UAM

Wzory na asymptoty ukośne.

Pytanie: jak w ogólnym przypadku znaleźć **wzory** na asymptoty?

Twierdzenie. *Warunkiem koniecznym i wystarczającym na to, aby funkcja* $y = mx + n$ *była asymptotą ukośną funkcji* f *dla* $x \longrightarrow +\infty$ $[x \longrightarrow -\infty]$, *jest aby:*

$$m = \lim_{x \to +\infty} \frac{f(x)}{x} \qquad \textit{oraz} \qquad n = \lim_{x \to +\infty} (f(x) - mx) \ ,$$

odpowiednio: $$\left[m = \lim_{x \to -\infty} \frac{f(x)}{x} \qquad \textit{oraz} \qquad n = \lim_{x \to -\infty} (f(x) - mx) \right] \ .$$

Należy jeszcze wyjaśnić co to za granice...

Granice w nieskończoności.

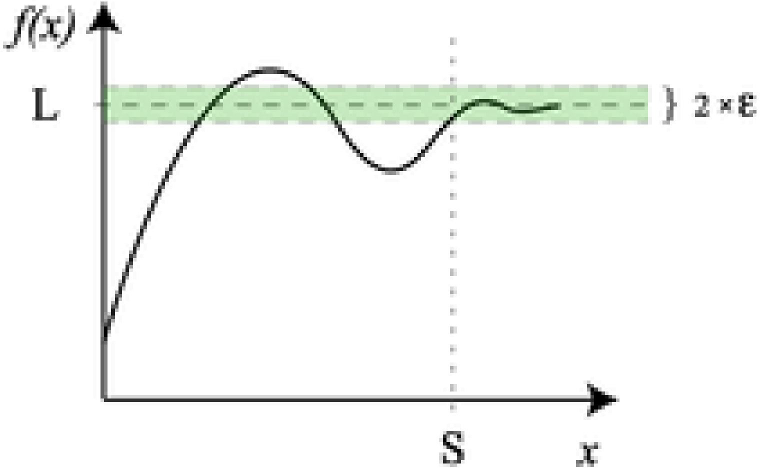
Definicja. Niech funkcja f będzie przy pewnym $a > 0$ określona w przedziale $[a, \infty)$. Liczbę $g \in \mathbb{R}$ nazywamy **granicą funkcji** f **przy** x **dążącym do** $+\infty$ (co zapiszemy $g = \lim_{x \to +\infty} f(x)$ lub $f(x) \longrightarrow g$ dla $x \to +\infty$) gdy dla dowolnej liczby $\varepsilon > 0$ istnieje taka liczba $A > 0$, że jeśli $x > A$, to $| f(x) - g |< \varepsilon$

$$\underset{\varepsilon>0}{\forall} \quad \underset{A>a}{\exists} \quad \underset{x\in[a,\infty)}{\forall} \quad x > A \quad \Longrightarrow \quad | f(x) - g |< \varepsilon \ .$$

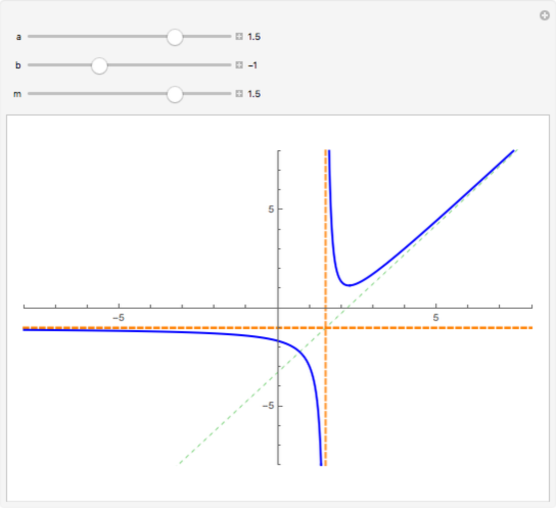
Analogicznie dla funkcji f określonej w $(-\infty, a]$ liczbę g nazwiemy **granicą funkcji** f **przy** x **dążącym do** $-\infty$ ($g = \lim_{x\to-\infty} f(x)$ lub $f(x) \to g$ przy $x \to -\infty$), gdy

$$\underset{\varepsilon>0}{\forall} \quad \underset{A>0}{\exists} \quad \underset{x\in(-\infty,a]}{\forall} \quad x < -A \quad \Longrightarrow \quad | f(x) - g |< \varepsilon \ .$$

Granica w nieskończoności.



Przykłady asymptot.



Asymptota (pozioma) w $-\infty$ oraz ukośna w $+\infty$.
Jedna asymptota pionowa.

Przykład.

$$f(x) = x + \frac{1}{x}\ , \qquad x \neq 0$$

Funkcja ta jest ciągła w każdym punkcie swojej dziedziny (sprawdzić !). Mamy więc:

$$\lim_{x \to 0-} f(x) = -\infty$$

$$\lim_{x \to 0+} f(x) = +\infty$$

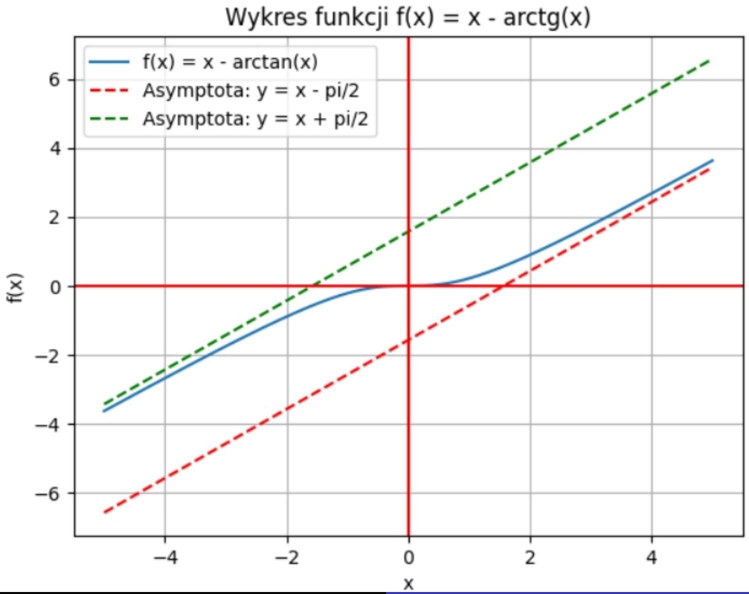
Czyli prosta $x = 0$ jest (obustronną) asymptotą pionową. Rozpatrujmy prostą $y = mx + n$

$$m = \lim_{x \to \pm\infty} \frac{f(x)}{x} = \lim_{x \to \pm\infty} (1 + \frac{1}{x^2}) = 1\ ,$$

$$n = \lim_{x \to \pm\infty} (f(x) - mx) = \lim_{x \to \pm\infty} \frac{1}{x} = 0\ .$$

Stąd $y = x$ jest asymptotą ukośną funkcji f (jedyną).

Ale mogą być dwie...



Zadanie samodzielne.

Wyznacz asymptoty funkcji:

(a) $f(x) = \dfrac{x^2 - 6x + 3}{x - 3}$,

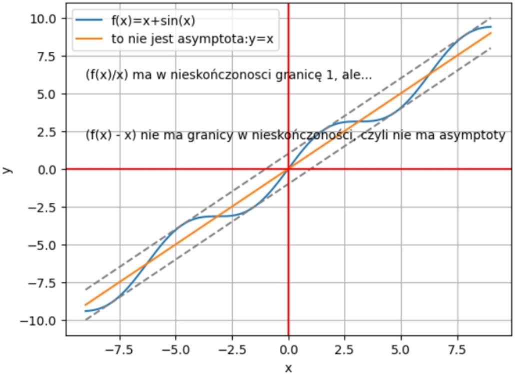
(b) $f(x) = x \cdot \text{arc ctg } x$,

(c) $f(x) = \ln(4 - x^2)$.

Prosimy zwrócić uwagę na istnienie **OBU** współczynników w definicji asymptoty ukośnej !!!

Przykład. Dla funkcji $f(x) = x + \sin x$ mamy:
$m = \lim_{x\to\pm\infty} \dfrac{f(x)}{x} = 1$

ALE
$n = \lim_{x\to\pm\infty}(f(x) - x) = \lim_{x\to\pm\infty} \sin x$ nie istnieje, a więc funkcja **nie ma** asymptot ukośnych ...



Nieciągłość w informatyce...

A czy musimy w informatyce badać funkcje nieciągłe? Choć to trudniejsze niż można by się spodziewać (zwłaszcza w przypadkach braku granicy w pewnych punktach), to jednak też ich potrzebujemy (choć niekiedy wymaga to odrębnych algorytmów). Oto kilka przykładów:

1. **Algorytmy optymalizacyjne** - funkcje nieciągłe często pojawiają się w problemach optymalizacyjnych, gdzie trzeba znaleźć minimum lub maksimum danej funkcji. Przykładowo, funkcje skokowe i kawałkami liniowe są często stosowane w projektowaniu algorytmów do minimalizacji kosztów produkcji lub maksymalizacji wydajności procesów.

2. **Symulacje systemów dynamicznych** - funkcje nieciągłe są często używane do modelowania systemów dynamicznych, takich jak sieci neuronowe, układy sterowania, czy procesy przepływu. Dzięki funkcjom nieciągłym można modelować zachowanie systemu w odpowiedzi na zdarzenia krytyczne, takie jak awarie, zmiany warunków środowiskowych, itp.

3. **Analiza sygnałów** - funkcje nieciągłe są często stosowane w analizie sygnałów, np. takich jak obrazy, dźwięki, czy sygnały biomedyczne. Przykładowo, funkcje skokowe są często używane do wykrywania krawędzi w obrazach (w pewnych algorytmach), czy segmentacji sygnałów EEG. Sygnały w świecie rzeczywistym często są nieciągłe, na przykład sygnały o zmiennej częstotliwości, sygnały impulsowe itp. Funkcje takie są wykorzystywane do modelowania tych sygnałów i analizy ich właściwości. Funkcje nieciągłe, takie jak funkcja progowa, są powszechnie stosowane w przetwarzaniu obrazów do detekcji krawędzi i segmentacji obrazów.

4. **Kryptografia** - funkcje nieciągłe są wykorzystywane w algorytmach szyfrowania, które opierają się na trudności odwrócenia funkcji jednokierunkowej. Przykładem takiej funkcji jest funkcja skokowa, która jest trudna do odwrócenia, gdyż nie ma jednoznacznie zdefiniowanego odwrotnego działania.

5. **W grach komputerowych** - funkcje nieciągłe, takie jak skoki, są często stosowane w programowaniu gier komputerowych do modelowania *ruchu* postaci i przeszkód.

6. **Symulacje zachowań społecznych** - funkcje nieciągłe są często stosowane w modelowaniu zachowań społecznych, takich jak strategie gry, zachowania konsumentów, czy interakcje między osobami. Funkcje nieciągłe pozwalają modelować skomplikowane zachowania, takie jak zmiany preferencji lub sytuacje konfliktowe.

7. **W kompresji danych** - wiele algorytmów kompresji danych, takich jak kompresja różnicowa, wykorzystuje funkcje nieciągłe do kodowania zmian między kolejnymi wartościami danych.

8. **W uczeniu maszynowym** - funkcje nieciągłe, takie jak funkcja skoku, są często stosowane w sieciach neuronowych jako funkcje aktywacji. Te funkcje pozwalają na modelowanie nieciągłości w danych i wprowadzanie nieliniowych przekształceń.

9. **W analizie numerycznej** - funkcje nieciągłe są często wykorzystywane do testowania i walidacji algorytmów numerycznych, stanowią wyzwanie dla tych algorytmów i mogą prowadzić do błędów numerycznych.

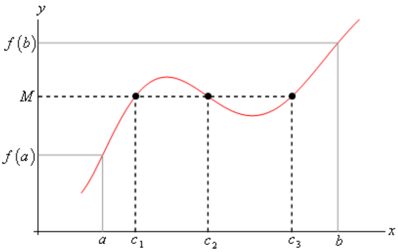
Mówimy, że funkcja f jest ciągła w zbiorze A, jeżeli jest ciągła w każdym punkcie x_0 tego zbioru. Jeżeli przy tym $A = [a, b]$, to w punkcie $x_0 = a$ rozważamy ciągłość prawostronną, a w punkcie $x_0 = b$ - ciągłość lewostronną.

Twierdzenie. *Funkcja ciągła w przedziale domkniętym jest w tym przedziale ograniczona.*

Twierdzenie. *Funkcja ciągła w przedziale domkniętym osiąga w nim swoje kresy.*

Własność Darboux.

Twierdzenie. (własność Darboux) *Jeżeli funkcja f jest określona i ciągła w przedziale $P = [a, b]$ oraz $x_1, x_2 \in P$, $x_1 < x_2$ będą takie, że $y_1 = f(x_1) \neq f(x_2) = y_2$, to funkcja f przyjmuje w przedziale $[x_1, x_2]$ wszystkie wartości pośrednie między y_1 i y_2.*



https://www.geogebra.org/m/CXEN5xM3

Bisekcja.

A tu już coś korzystającego z własności funkcji - na początek **ciągłość.**

To prosta (czyżby?) metoda znajdowania przybliżonego rozwiązania równania.

Weźmy równanie (dla ułatwienia jest to wielomian o współczynnikach całkowitych):

$$W(x) = x^5 - 8x^4 + x + 11.$$

Na początek jest łatwo: wiemy (dzięki matematyce!), że mamy od 1 do 5 rozwiązań rzeczywistych (można wykonać wykres, o ile potrafimy...).

Obliczamy kilka wartości w różnych punktach (zastanowić się jak je wybierać). Np. $W(-2) = -151 < 0$, $W(0) = 11 > 0$, $W(2) = -83 < 0$, $W(10) = ... > 0$. Zlokalizowaliśmy co najmniej 3 rozwiązania (a ile ich jest?).

Metoda bisekcji

Niech $g : [a, b] \to \mathbb{R}$ będzie ciągła i $g(a) \cdot g(b) \leqslant 0$.

Wtedy z **własności Darboux** istnieje $p \in (a, b)$ takie, że

$$g(p) = 0 \quad \text{(miejsce zerowe).}$$

Punkt p można aproksymować dzieląc systematycznie przedziały na pół i sprawdzając, w której połowie musi leżeć pewne miejsce zerowe g:

$$[a_0, b_0] := [a, b],$$

$$[a_{n+1}, b_{n+1}] := \begin{cases} \left[a_n, \frac{a_n+b_n}{2}\right], & \text{gdy } g(a_n) \cdot g\left(\frac{a_n+b_n}{2}\right) \leqslant 0, \\ \left[\frac{a_n+b_n}{2}, b_n\right], & \text{w przeciwnym wypadku.} \end{cases}$$

Wbrew pozorom przedstawiony algorytm na ogół nie lokalizuje pierwiastka g najbliższego środkowi $\frac{a+b}{2}$.

Odrzucane przedziały mogą zawierać miejsca zerowe g.

Założenia metody.

Dwa główne założenia metody gwarantujące zbieżność iteracji do miejsca zerowego to:

1) f jest ciągła,

2) f zmienia znak w $[a, b]$, czyli $f(a) \cdot f(b) < 0$,

Oczywiście - jak już mówilismy nie pozwala ona znaleźć wszystkich miejsc zerowych (tak byłoby gdyby miała dodatkową własność):

3) f jest ściśle monotoniczna na $[a, b]$.

Czy założenia są konieczne? Tak: np. dla funkcji $f(x) = \frac{1}{x}$ dla $[-1, 1]$ i $f(0) = 1$ mamy $f(1) \cdot f(-1) < 0$, ale funkcja nie ma miejsc zerowych. Drugie założenie jest oczywiste: np. $g(x) \equiv 1$ jest ciągła na dowolnym przedziale, ale nie ma miejsc zerowych.

Bisekcja - c.d.

Teraz zastosujemy znany algorytm bisekcji (mozna napisać program w dowolnym języku), **ALE ...** postawimy pytania:

1. Ile jest rozwiązań?
2. Czy na pewno w przedziałach $[-2, 0]$, $[0, 2]$ i $[2, 10]$ mamy rozwiązanie i jest ono **jedyne**?
3. Czy (i jak?) można oszacować błąd przybliżenia?
4. O ile przyjmiemy zakładaną dokładność przez ε, to co się stanie z algorytmem, gdy $|x_1 - x_2| < \varepsilon$ (x_1, x_2 - rozwiązania)?

Dobry pomysł? No to równanie

$$\sin \frac{2\pi}{x} = 0 \qquad x \in \left[\frac{1}{10000000}, 1\right]$$

i powodzenia (proszę zrobić symulację w dowolnym programie)...
Problemem jest też powolna zbieżność metody...

Pomoże matematyka (i analiza matematyczna)...

Bisekcja - problemy.

Aby stosować ten algorytm musimy kontrolować własności funkcji f w równaniu $f(x) = 0$ w $[a, b]$. Zakładamy $f(a) \cdot f(b) < 0$. (jak zawsze - istnienie i jedyność)

(1) musi **istnieć** rozwiązanie w tym przedziale (dzięki ciągłości możemy mieć własność Darboux, a to jest warunkiem wystarczającym),

(2) aby wyznaczyć rozwiązanie musimy wyizolować **jedyne** rozwiązanie w pewnym podprzedziale (tu mogą pomóc inne własności jak moduł ciągłości, albo - o czym później - własności pochodnej...),

(3) metoda nie gwarantuje znalezienia wszystkich rozwiązań.

... ale nawet wtedy algorytm nie musi być skuteczny (uwaga na przybliżenia...). Proponuję przeczytać: [K] str. 24-25 i może **sprawdzić** tamte informacje?

Ciągłość jednostajna.

Zgodnie z definicją ciągłości funkcji f w punkcie x_0, wybór liczby δ może być zależny od ε i od x_0. Jeśli uda się dobrać δ niezależnie od wyboru punktu x_0, to takie funkcje (a nie są to wszystkie funkcje ciągłe) będą miały szczególne własności.

Definicja. Funkcję f określoną w niepustym zbiorze $A \subset \mathbb{R}$ nazywamy **jednostajnie ciągłą**, gdy dla każdego $\varepsilon > 0$ istnieje $\delta > 0$ taka, że dla dowolnych $x, x_0 \in A$ spełniających warunek $\mid x - x_0 \mid < \delta$, zachodzi nierówność $\mid f(x) - f(x_0) \mid < \varepsilon$.

Zestawimy tu powtórnie obie definicje:

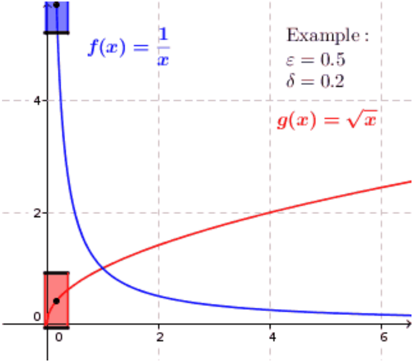
$$\underset{x_0}{\forall} \quad \underset{\varepsilon>0}{\forall} \quad \underset{\delta>0}{\exists} \quad \underset{x\in A}{\forall} \quad \mid x - x_0 \mid < \delta \implies \mid f(x) - f(x_0) \mid < \varepsilon \ ,$$

$$\underset{\varepsilon>0}{\forall} \quad \underset{\delta>0}{\exists} \quad \underset{x_0\in A}{\forall} \quad \underset{x\in A}{\forall} \quad \mid x - x_0 \mid < \delta \implies \mid f(x) - f(x_0) \mid < \varepsilon \ .$$

Przykładem funkcji ciągłej jednostajnie na $A = \mathbb{R}$ jest $f(x) = \sin x$, ale np. dla $A = (0, 1)$ funkcja $f(x) = \frac{1}{x}$ nie jest ciągła jednostajnie.

Ta własność jest silniejsza niż ciągłość, ale mamy twierdzenie:

Twierdzenie. *Funkcja ciągła w przedziale domkniętym jest w tym przedziale jednostajnie ciągła.*



W informatyce...

Własność jednostajnej ciągłości wydaje się trudna i zbędna w informatyce. Błąd!

Tylko takie funkcje są przydatne w obliczeniach! Zauważmy, że obliczając $f(x_0)$ (por. *Funkcje 2.*) wyznaczamy x_0 z pewną dokładnością, powiedzmy δ. Zgodnie z definicją ciągłości wartość $f(x_0)$ wyznaczymy z pewnym błędem ε. Oczekujemy, że w **innych** punktach x błąd **powinien być taki sam**. Ale to - to właśnie jednostajna ciągłość funkcji!!
Błąd oszacowania wartości $f(x)$ jest zależny od błędu oszacowania x, ale jest jednakowy dla wszystkich wartości $x \in A$ tylko dla funkcji jednostajnie ciągłej na A.

Stąd będziemy zwracali uwagą na warunki wystarczające ciągłości jednostajnej. Jeden już był, a drugi pojawi się po wprowadzeniu pochodnych (może już teraz: ograniczona pochodna na A, albo warunek Lipschitz'a). To często *"ukryte" założenie algorytmów* obliczania wartości funkcji...

Dla przypomnienia: błędy reprezentacji - powstają, gdy występuje konieczność reprezentacji liczby na komputerze z wykorzystaniem skończonej długości słów binarnych (ciągów bitów), co wymusza zaokrąglanie. Każda liczba typu rzeczywistego jest więc wyznaczana z błędem reprezentacji, a jego wielkość zależy od precyzji obliczeń.

Przykład. Oznaczmy przez δ błąd reprezentacji liczby x. Jak będzie wyglądało oszacowanie błędu obliczenia wartości funkcji f w punkcie x? To zależy jednak od funkcji...

Problemy arytmetyki liczb zmiennoprzecinkowych tu pominiemy, ale w realnych programach trzeba je uwzględnić.

Niech $f(x) = x^2 + 1$, $x \in [0, 3]$.

W praktyce otrzymamy liczbę $f(z)$, gdzie $z \in [x - \delta, x + \delta]$. Możliwy do szacowania błąd to

$$\epsilon(x) = \max\{|f(s) - f(t)| : \ s, t \in [x - \delta, x + \delta]\}.$$

Tu mamy (co wyjaśnimy nieco później):

$$\epsilon(x) = \max\{|f(s) - f(t)| : \ s, t \in [x - \delta, x + \delta]\} \leqslant 2|x| \cdot \delta.$$

Ponieważ $x \in [0, 3]$, to istnieje wspólne oszacowanie błędu dla **wszystkich** x jednocześnie

$$\epsilon \leqslant 6 \cdot \delta.$$

I tu klucz do wyjaśnienia pojęcia jednostajnej ciągłości funkcji: niezależnie od tego jaki punkt x wybierzemy, to oszacowanie uzyskanej wartości jest takie samo i zależy tylko od δ.

Ćwiczenie: określić błąd obliczenia dla $f(e) = e^2 + 1$ w Pythonie:

```
import math

result = math.exp(2) + 1

print(result)
```

A dla zainteresowanych: jak działa funkcja *exp* z biblioteki *math*?

Albo zmieniać wartość δ zmieniajac precyzję (czyli: δ) (lub definicję *exp*):

```
from decimal import *

getcontext().prec = 30 # ustawienie precyzji na 30 cyfr

e = Decimal('2.718281828459045235360287471352')

result = e ** 2 + 1
```

A teraz inna funkcja: $f(x) = \frac{1}{x^2} + 1$ dla $x \in (0,3]$.

Teraz po obliczeniach otrzymamy liczbę $f(z)$, gdzie $z \in [x-\delta, x+\delta]$. Możliwy do szacowania błąd formalnie jest tak samo szacowany

$$\epsilon(x) = \max\{|f(s) - f(t)| s, t \in [x-\delta, x+\delta]\}.$$

Tu mamy (o czym nieco później):

$$\epsilon(x) = \max\{|f(s) - f(t)| s, t \in [x-\delta, x+\delta]\} \leqslant \frac{2}{x^3} \cdot \delta.$$

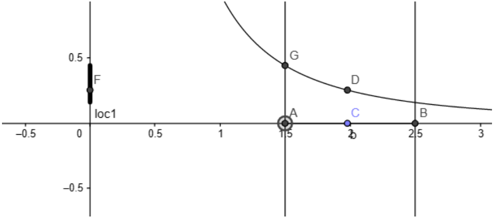
Ale teraz nie mamy wspólnego oszacowania na $L(x) = \frac{2}{x^3}$ w przedziale $(0,3]$!

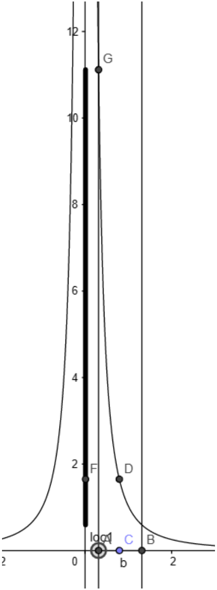
Np. mamy $L(1) = 2$, $L(2) = \frac{1}{4} = 0,25$. Proszę obliczyć wartość tego wyrażenia $L(x)$ w: $x = 0,001$ oraz w $x = 0,000001$.

I już widać jak duże są to stałe! Co więcej

$$\sup_{x \in (0,3]} L(x) = \infty.$$

Ta funkcja **nie jest jednostajnie ciągła na zbiorze** $(0,3]$ (choć ciągła) i *nie istnieje wspólne oszacowanie błędu obliczeń wartości tej funkcji na tym zbiorze przy zadanej precyzji* δ.





dla $x = 2$ dla $x < 1$

Korzystając zaś z definicji Heinego i znanych już twierdzeń dla ciągów liczbowych można (oprócz powyższego) udowodnić kolejne twierdzenia zdecydowanie ułatwiające obliczanie granic:

Twierdzenie. *Jeżeli* $\lim_{x\to x_0} f(x) = 0$ *i funkcja* g *jest ograniczona w pewnym zbiorze* $(x_0 - a, x_0) \cup (x_0, x_0 + a)$ *(dla pewnego* $a > 0$*) to* $\lim_{x\to x_0} f(x) \cdot g(x) = 0$*.*

Twierdzenie. *Jeżeli* $\lim_{x\to x_0} f(x) = a$ *i* $\lim_{x\to x_0} g(x) = b$ *to*

$$lim_{x\to x_0} f(x) \cdot g(x) = a \cdot b.$$

Twierdzenie. (o granicy ilorazu). *Jeżeli funkcję* f *i* g *mają granice w punkcie* x_0 *i* $\lim_{x\to x_0} g(x) \neq 0$*, to funkcja* $\frac{f}{g}$ *ma też granicę w punkcie* x_0 *oraz*

$$\lim_{x\to x_0} \frac{f(x)}{g(x)} = \frac{\lim_{x\to x_0} f(x)}{\lim_{x\to x_0} g(x)}\ .$$

Granice niewłaściwe.

Definicja. Niech funkcja f będzie określona dla takich x, że $0 <| x - x_0 |$ przy pewnym $a > 0$. Mówimy, że funkcja f ma w punkcie x_0 **granicę niewłaściwą** $+\infty$, gdy dla dowolnej liczby $M > 0$ istnieje taka $\delta > 0$, że jeśli $0 <| x - x_0 |< \delta$, to $f(x) > M$.

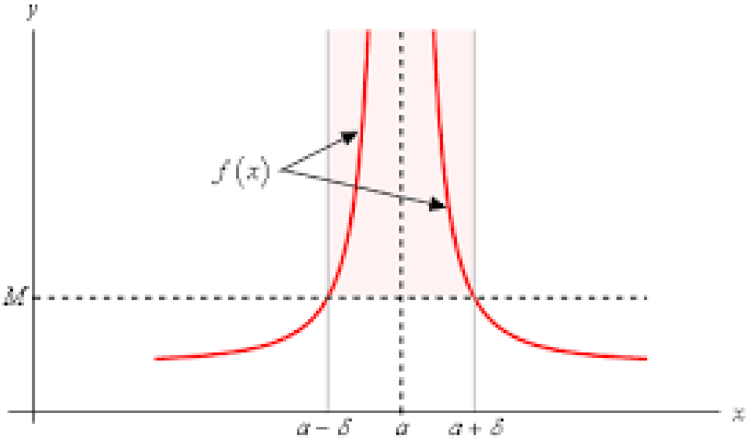
Fakt ten zapisujemy $\lim_{x \to x_0} f(x) = +\infty$.

$$\underset{M>0}{\forall} \quad \underset{\delta>0}{\exists} \quad 0 <| x - x_0 |< \delta \quad \Longrightarrow \quad f(x) > M \ .$$

Analogicznie: funkcja f ma w punkcie x_0 **granicę niewłaściwą** $-\infty$, gdy

$$\underset{M>0}{\forall} \quad \underset{\delta>0}{\exists} \quad 0 <| x - x_0 |< \delta \quad \Longrightarrow \quad f(x) < -M$$

Granica niewłaściwa.



Błędy...

Proces aproksymacji to ważny punkt *analizy matematycznej*. Obliczanie przez komputer wyrażenia z zadaną dokładnością nie jest banalne gdy **obliczamy wartości rzeczywiste** x. Przecież już w punkcie wyjścia mamy wartość przybliżoną (np. przekątna kwadratu o boku 1 ...). Przykłady metod ograniczania błędów - korzystanie z funkcji "łatwych obliczeniowo" (f - "trudna", g - "łatwa" obliczeniowo):

$$f(x) = x \cdot \sin x \quad \text{oraz} \quad g(x) = x^2$$

mają "bliskie" wartości dla x w otoczeniu zera

$$f(x) = \frac{x^2 + 1}{x} \quad \text{oraz} \quad g(x) = x$$

mają "bliskie" wartości dla "dostatecznie" dużych x.

Ograniczamy błedy...

To co da nam *mniejszy błąd*: obliczenia wartości funkcji obciążone błędem obcięcia (np. dla funkcji f "trudnych obliczeniowo" powyżej), czy wartości w tych punktach "bliskich" funkcji g?

```
def f(x):
    return (x**2 + 1) / x

print(f(983646))
```

co daje 98364.0000010167. Oczywiście $g(x) = x$ daje nam **bez dodatkowych obliczeń** $g(983646) = 983646$. Co więcej *print(f(9999999999))* daje nam 9999999999.0.

A jak rola analizy matematycznej? Koniec z cudzysłowami, skorzystamy z **granic i asymptot** (symbole o "małe" i O "duże").

Czyli wyjaśnimy, co to jest "bliska funkcja". I do tego potrzebujemy **granic**.

Funkcje asymptotycznie niewiększe.

Prawie przy każdej okazji przedstawiania algorytmu podany będzie np. rząd jego złożoności obliczeniowej: "O" duże = symbol Landau'a, np. sortowanie o złożoności $O(n \log n)$, co posłuży do oceny i porównywania algorytmów. *Alternatywą* są nierówności pomiędzy ciągami dowodzone poprzez indukcję matematyczną...

Funkcja asymptotycznie niewiększa od funkcji $g(n)$ to taka funkcja $f : \mathbb{N} \to \mathbb{R}$, dla której istnieją $c > 0$ i $n_0 \in \mathbb{N}$, że $|f(n)| \leqslant c \cdot |g(n)|$ dla (prawie) wszystkich $n \geqslant n_0$.

Podstawowym zastosowaniem notacji asymptotycznej **w informatyce** jest szacowanie długości działania programów, w szczególności procedur rekurencyjnych, których złożoność łatwo opisać równaniem rekurencyjnym.

Patrz też notacje: "duże Theta" $\Theta(n)$ i "duże Omega $\Omega(n)$" (ich warunki wystarczające w języku granic ciągów)...

Czasowa złożoność obliczeniowa.

Oznacza to, że $|f(n)| \leqslant c \cdot |g(n)|$ zachodzi dla (prawie wszystkich) liczb naturalnych n, czyli po prostu (warunek wystarczający)

$$f = O(g) \Leftarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

lub nawet niekiedy warunek stosowany ogólniej:

$$\limsup_{x\to\infty} \frac{f(x)}{g(x)} < \infty.$$

Zbiór funkcji asymptotycznie nie większych niż $g(n)$ oznaczamy przez $O(g(n))$. Przykładowe zastosowanie w informatyce: **twierdzenie o rekursji uniwersalnej** (szacowanie długości działania programu wraz ze wzrostem ilości danych - oczywiście asymptotyczne oszacowanie).

link: polecane materiały...

Python

Mamy oczywiście w Pythonie (biblioteka *SymPy*) "obliczanie" granic. Czasami wystarczy...

```
def ratio(x):
    return f(x) / g(x)

from sympy import limit, Symbol
x = Symbol('x')result = limit(ratio(x), x, float('inf'))
print(result)
```

gdzie np.

```
def f(x):
    return x ** 2 + 3*x - 1

def g(x):
    return 2*x ** 2 - x + 5
```

Ta funkcja wykorzystuje bibliotekę *SymPy*, która umożliwia "obliczanie" granic i operacji matematycznych na symbolicznych wyrażeniach. Wynik tego wywołania to granica stosunku $f(x)/g(x)$ dla x dążącego do nieskończoności.

Sprawdź czy wynik granicy jest skończony i niezerowy:

a) Jeśli granica jest skończona i niezerowa, to f(x) jest rzędu O(g(x)).

b) Jeśli granica jest równa zero, to f(x) jest rzędu o mniejszym rządzie niż g(x).

c) Jeśli granica jest nieskończona, to f(x) jest rzędu o wyższym rządzie niż g(x).

Funkcje asymptotycznie podobne (równe).

Jeżeli $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1 < \infty$, to funkcje są asymptotycznie równe (czyli $f(n) \sim g(n)$). Studenci matematyki uczą się np. **wzoru Stirliga** wykorzystywanego praktycznie zawsze winformatyce, gdy mielibyśmy obliczać silnię

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

(o liczbie e powiemy oczywiście na wykładzie...), a w informatyce (**kryptografia**) np. przybliżenie na ilość liczb pierwszych nie większych niż n

$$\pi(n) \sim \frac{n}{ln\ n}.$$

Możliwe zastosowanie wzoru Stirlinga dla **informatyków**: np. oszacowanie liczby cyfr rozwinięcia dziesiętnego liczby 999!.

Funkcje asymptotycznie mniejsze.

Kolejny szczególnie ciekawy przypadek:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0,$$

czyli symbol "o" małe..., czyli istnieje n_0, takie, że dla dowolnego $c > 0$ nierówność $|f(n)| < c \cdot |g(n)|$ zachodzi dla wszystkich liczb naturalnych $n \geqslant n_0$.

W **informatyce** - np. przydatne w badaniach złożoności obliczeniowej, jak w twierdzeniach o hierarchii czasowej i pamięciowej czy w szacowaniu reszty we wzorze Taylora = błędu lub w badaniach złożoności czasowej algorytmów (np. istotny wynik: dla każdego k mamy $\log_2 n = o(n^k)$ - algorytm przeszukiwania połówkowego), patrz też - później - tw. Stolza i reguła de l'Hôspitala....

Poza tym dzięki twierdzeniu: jeżeli $f(n) = o(g(n))$, to $f(n) = O(g(n))$, pojęcie będzie przydatne bezpośrednio.

Symbole Landaua.

Zależności algebraiczne O, o, Ω, ω, Θ.

zapis .... warunek wystarczający

$f(x) \in O(g(x))$: $\lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| < \infty$

$f(x) \in o(g(x))$: $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$

$f(x) \in \Omega(g(x))$: $\lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| > 0$

$f(x) \in \omega(g(x))$: $\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty$

$f(x) \in \Theta(g(x))$: $0 < \lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| < \infty$

$f(x) \in O(1)$ – funkcja $f(x)$ jest ograniczona,

$f(x) \in O(\log n)$ – funkcja $f(x)$ jest ograniczona przez funkcję logarytmiczną,

$f(x) \in O(n)$ – funkcja $f(x)$ jest ograniczona przez funkcję liniową,

$f(x) \in O(n \log n)$ - w informatyce, funkcja $f(x)$ jest ograniczona przez funkcję quasi-liniową,

$f(x) \in O(n!)$ – funkcja $f(x)$ jest ograniczona przez silnię.

Zastosowanie: badanie złożoności obliczeniowej algorytmów. Najczęstszym zastosowaniem asymptotycznego tempa wzrostu jest szacowanie złożoności problemów obliczeniowych, w szczególności algorytmów. Oszacowanie rzędów złożoności obliczeniowej funkcji pozwala na porównywanie ilości zasobów (np. czasu, pamięci), jakich wymagają do rozwiązania problemu opisanego określoną ilością danych wejściowych. W dużym uproszczeniu: im niższy rząd złożoności obliczeniowej algorytmu, tym będzie on wydajniejszy przy coraz większym rozmiarze problemu (np. ilości danych).

Niezbędnik informatyka!

To warto mieć zawsze “pod ręką”:

Concrete mathematics, a foundation for computer science, Ronald L. Graham, Donald E. Knuth, Oren Patashnik. 625 stron, 1989. (są nowsze wydania i jest wersja polska)

M.in. zagadnienie symboli Landaua i ich rola w informatyce: rozdział 9.

Przykład.

Złożoność czasowa algorytmu. Rozważmy następujący algorytm, który znajduje maksymalny element w tablicy n elementowej:

1. Ustaw maksimum na pierwszy element tablicy.

2. Przeglądaj kolejne elementy tablicy i porównuj je z maksimum.

3. Jeśli napotkasz element większy od maksimum, ustaw maksimum na ten element.

4. Zwróć maksimum.

Aby oszacować złożoność czasową tego algorytmu, można skorzystać z *symboli Landaua*.

Zauważmy, że krok 1 wymaga jednej operacji porównania, a kroki 2-4 wymagają n porównań. Zatem całkowita liczba porównań wykonywanych przez algorytm to $n+1$. Oznaczmy tę liczbę przez $T(n)$.

To prosty przykład, więc łatwo o oszacowania. Dla dużych wartości n, wartość $T(n)$ jest "zdominowana" przez wyrażenie n, ponieważ n jest większe niż 1, a wartość 1 nie ma dużego wpływu na $T(n)$. Zatem $T(n)$ można zapisać jako $T(n) = O(n)$, co oznacza, że złożoność czasowa tego algorytmu to złożoność liniowa.

A teraz, mniej opisowo, a bardziej precyzyjnie. Możemy teraz oszacować $T(n)$ za pomocą granic i symboli Landaua. Innym sposobem oszacowania złożoności tego algorytmu jest oczywiście użycie granic. Możemy wyrazić $T(n)$ jako:

$$T(n) = n + 1$$

W granicy, gdy n dąży do nieskończoności mamy

$$\lim_{n \to \infty} \frac{T(n)}{n} = \lim_{n \to \infty} \frac{n+1}{n} = 1.$$

Oznacza to, że złożoność czasowa tego algorytmu to $O(n)$, czyli złożoność jest liniowa, co oznacza, że czas działania algorytmu rośnie liniowo wraz z rozmiarem danych wejściowych (w tym przypadku rozmiarem tablicy).

Zastosowanie.

Badanie złożoności obliczeniowej algorytmów. Najczęstszym zastosowaniem asymptotycznego tempa wzrostu jest szacowanie złożoności problemów obliczeniowych, w szczególności algorytmów. Oszacowanie rzędów złożoności obliczeniowej funkcji pozwala na porównywanie ilości zasobów (np. czasu, pamięci), jakich wymagają do rozwiązania problemu opisanego określoną ilością danych wejściowych. W dużym uproszczeniu: im niższy rząd złożoności obliczeniowej algorytmu, tym będzie on wydajniejszy przy coraz większym rozmiarze problemu (np. ilości danych).

Algorytm sortowania jest jednym z podstawowych problemów informatycznych i polega na uporządkowaniu zbioru danych wejściowych według określonego porządku. Istnieje wiele różnych algorytmów sortowania, z których każdy ma swoją złożoność czasową i przestrzenną.

Dla przykładu, algorytm sortowania bąbelkowego ma złożoność czasową $O(n^2)$, co oznacza, że liczba operacji wykonywanych przez ten algorytm rośnie kwadratowo wraz ze wzrostem liczby danych wejściowych. Dla dużych rozmiarów danych wejściowych ten algorytm staje się bardzo wolny i nieefektywny.

Z kolei algorytm sortowania szybkiego ma złożoność czasową $O(n \log n)$, co oznacza, że liczba operacji wykonywanych przez ten algorytm rośnie liniowo-logarytmicznie wraz ze wzrostem liczby danych wejściowych. Dla dużych rozmiarów danych wejściowych ten algorytm działa znacznie szybciej niż algorytm sortowania bąbelkowego.

W obu przypadkach używamy symboli Landaua, aby określić tempo wzrostu złożoności czasowej wraz ze wzrostem liczby danych wejściowych.

Uwaga. A teraz konieczne jest krótkie wyjaśnienie dlaczego umieściliśmy te rozważania dotyczące granic ciągów w dziale dotyczącym granic funkcji (w nieskończoności).

To wynika z definicji Heinego granicy funkcji, ale przede wszystkim z faktu, iż pojawi się sporo przypadków, gdy dużo łatwiej będzie obliczać granice funkcji niż ciągów! Metodą jest zbadanie ilorazu funkcji, czyli rozpatrzenie dziedziny $[1, \infty)$ i granicy

$$\lim_{x \to \infty} \frac{T(x)}{g(x)}.$$

To wynika z definicji Heinego: $x \to \infty$ oznacza, że możemy rozparywać DOWOLNY ciąg (x_n) taki, że $x_n \to \infty$ i badać $\lim_{n\to\infty} \frac{T(x_n)}{g(x_n)}$, a nas interesuje jeden z takich ciągów: $x_n = n$...

Na kolejnych wykładach pojawi się rachunek różniczkowy i doskonałe narzędzie: reguła de l'Hôspitala. Wtedy wrócimy do tematu, a na razie przykład ilustrujący problem.

Rozważmy algorytm sortowania przez scalanie (merge sort), który chcemy wykazać, ze działa w czasie $O(n \log n)$ dla n elementów.

Pierwszym krokiem jest podzielenie tablicy na dwie równe części, co zajmuje czas $O(1)$. Następnie rekurencyjnie sortujemy obie połowy, co daje czas $2T(n/2)$, gdzie $T(n)$ oznacza czas sortowania tablicy o n elementach. Następnie scalamy obie posortowane połowy w jedną, co również zajmuje czas $O(n)$. Możemy zapisać powyższe zależności w postaci równania rekurencyjnego:

$$T(n) = 2T(n/2) + O(n)$$

Mając takie równanie, możemy użyć symbolu Landaua do oszacowania złożoności czasowej algorytmu. Aby to zrobić, powinniśmy obliczyć granicę

$$\lim_{n\to\infty} \frac{T(n)}{n \log n}$$

Jeśli granica ta istnieje i jest skończona, to oznacza to, że złożoność algorytmu wynosi $O(n \log n)$. Tylko jak? W badaniach złożoności czasowych przedstawione zapewne dwie metody (bezpośrednio lub poprzez gotowe twierdzenie o granicy funkcji rekurencyjnych = matematyka!, czyli na razie niewiele mówiący gotowy wzór (?)

$$T(n) = O(n \log n) + n \log n \cdot \int_1^n \frac{T(x)}{x^2 \cdot (x \log x)} \, dx$$

(to całka! - we wzorze to sumowanie kosztów wywołań rekurencyjnych, a jak się wkrótce okaże to da się właśnie uogólnić do całki Riemanna - no i proszę musimy omówić całki na wykładzie :-)).

Czyli: jeśli obliczymy granicę funkcji, to znajdziemy też poszukiwaną granicę ciągu i będą one równe. W tym przypadku stosujemy tzw. metody drzewa rekursji (ang. recursion tree method), która jest jedną z metod rozwiązywania równań rekurencyjnych. Metoda ta polega na reprezentacji działań algorytmu za pomocą drzewa rekursji i obliczeniu złożoności na podstawie sumy kosztów działań na każdym poziomie drzewa.

W przypadku algorytmu sortowania przez scalanie, drzewo rekursji będzie miało logarytmicznie wiele poziomów, ponieważ w każdym etapie algorytm dzieli tablicę na dwie równe części. Na każdym poziomie drzewa, koszt działania to $O(n)$, ponieważ każda wartość z tablicy jest porównywana i przepisywana.

Możemy teraz policzyć całkowity koszt algorytmu, sumując koszty działań na każdym poziomie drzewa. Na poziomie 0, koszt to $O(n)$, na poziomie 1, koszt to $2 \cdot O(n/2)$, na poziomie 2, koszt to $4 \cdot O(n/4)$, i tak dalej. W ogólności, na poziomie i, koszt to $2^i \cdot O(n/2^i)$.

Sumując koszty na wszystkich poziomach, otrzymujemy:

$$T(n) = O(n) + 2 \cdot O(n/2) + 4 \cdot O(n/4) + ... + n \cdot O(n/n).$$

Możemy teraz uprościć powyższe wyrażenie stosując wzór na sumę skończoną ciągu geometrycznego:
$2^0 + 2^1 + 2^2 + ... + 2^k = 2^(k+1) - 1$. Przyjmując $k = \log_2(n)$, otrzymujemy:

$$T(n) = O(n \log n) + n \cdot O(1).$$

Ostatecznie, granica $T(n)/n \log n$ dąży do stałej $C = 2$, co oznacza, że złożoność algorytmu sortowania przez scalanie wynosi $O(n \log n)$.

Podsumowując, zastosowanie granic i symboli Landaua pozwala na oszacowanie złożoności czasowej algorytmów, co jest bardzo ważne w analizie wydajności programów i systemów informatycznych.

Tajne/poufne.

- Własność Darboux: rozpatrz 2 przypadki (funkcje ciągłe i pochodne funkcji). Uzasadnij konieczność jej wykoerzystania w metodzie bisekcji. Podaj inny przykład, gdy w algorytmie numerycznym korzystamy z tej własności.
- W oszacowaniach złożoności algorytmów występuje symbol Landaua "o małe" (funkcje asymptotycznie mniejsze): $f(n) = o(g(n))$. W praktyce oczywiście funkcje $f(n)$ i $g(n)$ są rozbieżne do niekończoności , więc korzystnie jest rozszerzyć funkcje f i g jako zdefiniowane na $\mathbb{R}$ i wykorzystać regułę de l'Hôspitala. **Podaj ją i sprawdź, że** dla dowolnego $\alpha > 0$ $f(n) = \log_2(n) = o(n^\alpha)$.
- Wiemy, że funkcja $f(x)$ jest rzędu $o(x^2)$ przy $x \to 0$. Wybierz, które z funkcji podanych poniżej spełniają taki warunek - **wykonaj obliczenia**:

  [a] $f_1(x) = x^3$,
  [b] $f_2(x) = 274x^2$,
  [c] $f_3(x) = 2x^{\frac{3}{2}}$,
  [d] $f_4(x) = (\sin x)^2$.

- Obliczając za pomocą komputera wartość pewnej funkcji $f : [a, b] \to \mathbb{R}$ w punkcie $x_0 \in [a, b]$ popełnimy na ogół błąd polegający na konieczności wykonania obliczeń przez komputer na liczbach rzeczywistych.
  Oznaczmy przez $\varepsilon > 0$ akceptowalną dokładność obliczeń wartości funkcji f, a przez $\delta > 0$ możliwą dokładność wyznaczania wartości liczby rzeczywistej x_0. Jaka własność funkcji f pozwala, przy ustalonym $\varepsilon > 0$ na uzyskanie **wspólnej dla wszystkich punktów** x_0 wielkości $\delta > 0$? Jaka klasa funkcji ciągłych f to zapewnia i podaj 2 przypadki (twierdzenia), pozwalające zbadać zachodzenie tej własności.
- Oblicz, czy $2^{n+1} = O(2^n)$? A czy $2^{2n} = O(2^n)$? (wsk. : symbole Landaua "O duże")