

# Maxima - przewodnik praktyczny

Rafał Topolnicki  
rtopolnicki@o2.pl

KNF Migacz  
Wydział Fizyki i Astronomii  
Uniwersytet Wrocławski

Wrocław, 15 października 2009



Początek prezentacji jest przydługi i raczej nieciekawny  
Proszę się nie poddawać ;)



## Plan:

- 1 Operatory, funkcje matematyczne
- 2 Listy
- 3 Modyfikowanie środowiska
- 4 Przekształcanie wyrażeń
- 5 Macierze liczby zespolone
- 6 Rozwiązywanie równań
- 7 Grafika
- 8 Operacje na plikach. "Analiza danych"
- 9 Granice, pochodne, całki oznaczone i nieoznaczone
- 10 Równania różniczkowe
- 11 Szeregi Fouriera
- 12 Elementy języka programownia
- 13 Przykłady:
  - Oscylator harmoniczny
  - Potencjał wewnątrz nieskończonej wnęki
  - Jednowymiarowa studnia potencjału
  - Przykład całkowania
  - Relatywistyczne składnie prędkości
  - Prosta animacja



## O programie

Maxima jest programem typu CAS (*Computer Algebra System*), wspomagającym wykonywanie obliczeń symbolicznych.

Możliwości:

- Różniczkowanie i całkowanie symboliczne
- Rozwiązywanie równań i układów równań algebraicznych
- Rozwiązywanie wybranych typów równań różniczkowych
- Upraszczenie wyrażeń algebraicznych
- Tworzenie wykresów 2D i 3D (za pośrednictwem Gnuplota)
- Szeregi Fouriera
- Operacje na macierzach
- Fitowanie (dopasowywanie wzorów funkcji do danych)
- Obliczenia dowolnej precyzji
- Eksport wyników do TeX'a
- Strukturalny język programowania (+Lisp)
- Wybrane operacje numeryczne
- Wybrane operacje statystyczne

i wiele innych ...



# Wady i zalety - subiektywny przegląd

## Zalety:

- Licencja GPL
- Dostępna pod wszystkie platformy zgodne z POSIX oraz Microsoft Windows
- Ogromna liczba rozszerzeń
- Względnie szczegółowa dokumentacja
- Prężnie działająca lista mailingowa

## Wady:

- Niektóre funkcje działają źle
- Wyniki działania niektórych funkcji należy poprawiać ręcznie
- Brak dokumentacji po polsku
- Duże braki w niektórych dziedzinach
- Trudna w obsłudze
- Słabo rozwinięty język programowania wewnątrz Maximy - konieczność odwoływania się do Lispa
- Mała popularność
- Lekkie zamieszanie z macierzami, tablicami i zagnieżdżonymi listami



# Wady i zalety - subiektywny przegląd

## Zalety:

- Licencja GPL
- Dostępna pod wszystkie platformy zgodne z POSIX oraz Microsoft Windows
- Ogromna liczba rozszerzeń
- Względnie szczegółowa dokumentacja
- Pręźnie działająca lista mailingowa

## Wady:

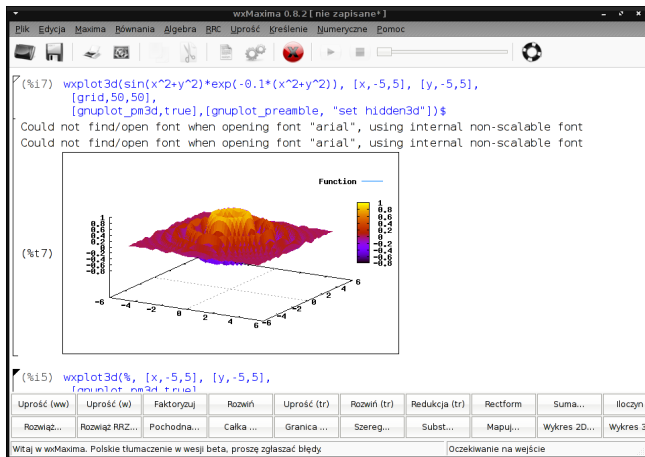
- Niektóre funkcje działają źle
- Wyniki działania niektórych funkcji należy poprawiać ręcznie
- Brak dokumentacji po polsku
- Duże braki w niektórych dziedzinach
- Trudna w obsłudze
- Słabo rozwinięty język programowania wewnątrz Maximy - konieczność odwoływania się do Lispa
- Mała popularność
- Lekkie zamieszanie z macierzami, tablicami i zagnieżdżonymi listami



# Maxima działa w trybie tekstowym. Istnieją graficzne nakładki:

## ■ wxMaxima

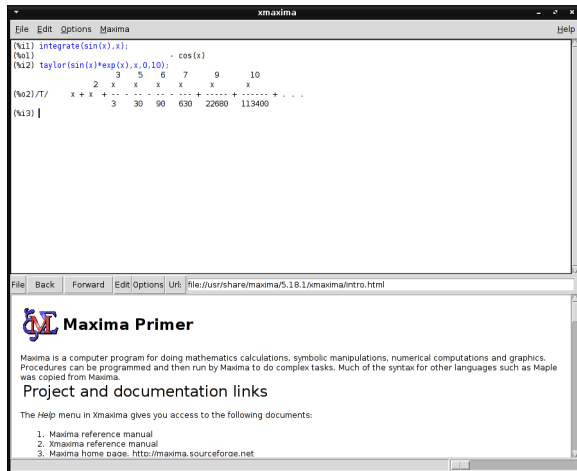
- Napisana w wxWidgets
- Bardzo szybko rozwijana co prowadzi do częstych zmian interfejsu
- Względnie spolonizowana
- Formatowanie wyników
- Od wersji 0.8.0 działa w formie "dokumentu" - podział na komórki



Maxima działa w trybie tekstowym. Istnieją graficzne nakładki:

## ■ xmaxima

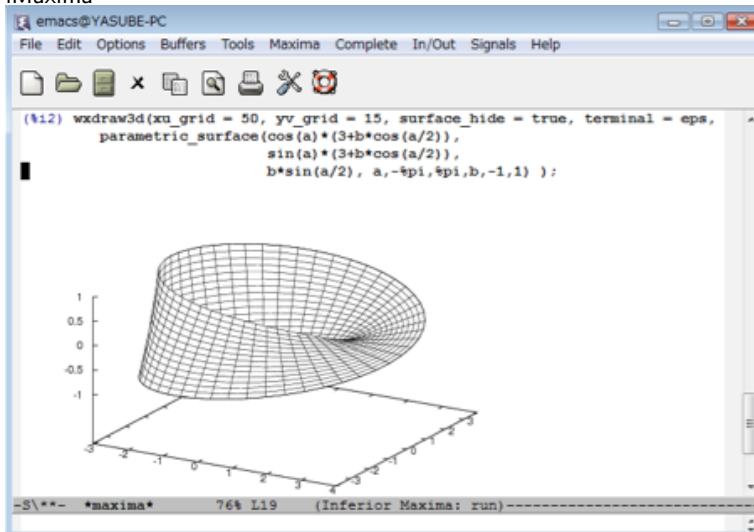
- Niezwykle prosta
- Środowisko nadal bardziej tekstowe niż graficzne
- Szybsza niż wxMaxima





Maxima działa w trybie tekstowym. Istnieją graficzne nakładki:

## ■ iMaxima



# Podstawy. Praca z programem

- Maxima działa w trybie wejść %i (od *input*) i wyjść %o (od *output*). Wejścia jak i wyjścia są numerowane, co w połączeniu z %i albo %o tworzy unikalny identyfikator
- Każdą instrukcję kończymy ; lub \$. Użycie tego drugiego powoduje, że wynik obliczeń nie pojawi się na ekranie
- Maxima rozróżnia wielkość liter, tak więc funkcje  $f(x)$  i  $F(x)$  nie są tym samym
- Zmienna % przechowuje wynik ostatniego polecenia
- Zmienna %% przechowuje natomiast wynik ostatniego wyrażenia w ciągu instrukcji ograniczonych () (coś na kształt potoku)

```
(%i1) (integrate(sin(x),x),diff(%%,x),ev(%%,x=0));  
(%o1) 0
```



# Podstawy. Praca z programem

- Maxima działa w trybie wejść %i (od *input*) i wyjść %o (od *output*). Wejścia jak i wyjścia są numerowane, co w połączeniu z %i albo %o tworzy unikalny identyfikator
- Każdą instrukcję kończymy ; lub \$. Użycie tego drugiego powoduje, że wynik obliczeń nie pojawi się na ekranie
- Maxima rozróżnia wielkość liter, tak więc funkcje  $f(x)$  i  $F(x)$  nie są tym samym
- Zmienna % przechowuje wynik ostatniego polecenia
- Zmienna %% przechowuje natomiast wynik ostatniego wyrażenia w ciągu instrukcji ograniczonych () (coś na kształt potoku)

```
(%i1) (integrate(sin(x),x),diff(%%,x),ev(%%,x=0));  
(%o1) 0
```



# Podstawy. Praca z programem

- Maxima działa w trybie wejść %i (od *input*) i wyjść %o (od *output*). Wejścia jak i wyjścia są numerowane, co w połączeniu z %i albo %o tworzy unikalny identyfikator
- Każdą instrukcję kończymy ; lub \$. Użycie tego drugiego powoduje, że wynik obliczeń nie pojawi się na ekranie
- Maxima rozróżnia wielkość liter, tak więc funkcje  $f(x)$  i  $F(x)$  nie są tym samym
- Zmienna % przechowuje wynik ostatniego polecenia
- Zmienna %% przechowuje natomiast wynik ostatniego wyrażenia w ciągu instrukcji ograniczonych () (coś na kształt potoku)

```
(%i1) (integrate(sin(x),x),diff(%%,x),ev(%%,x=0));  
(%o1) 0
```



# Podstawy. Praca z programem

- Maxima działa w trybie wejść %i (od *input*) i wyjść %o (od *output*). Wejścia jak i wyjścia są numerowane, co w połączeniu z %i albo %o tworzy unikalny identyfikator
- Każdą instrukcję kończymy ; lub \$. Użycie tego drugiego powoduje, że wynik obliczeń nie pojawi się na ekranie
- Maxima rozróżnia wielkość liter, tak więc funkcje  $f(x)$  i  $F(x)$  nie są tym samym
- Zmienna % przechowuje wynik ostatniego polecenia
- Zmienna %% przechowuje natomiast wynik ostatniego wyrażenia w ciągu instrukcji ograniczonych () (coś na kształt potoku)

```
(%i1) (integrate(sin(x),x),diff(%%,x),ev(%%,x=0));  
(%o1) 0
```



# Podstawy. Praca z programem

- Maxima działa w trybie wejść %i (od *input*) i wyjść %o (od *output*). Wejścia jak i wyjścia są numerowane, co w połączeniu z %i albo %o tworzy unikalny identyfikator
- Każdą instrukcję kończymy ; lub \$. Użycie tego drugiego powoduje, że wynik obliczeń nie pojawi się na ekranie
- Maxima rozróżnia wielkość liter, tak więc funkcje  $f(x)$  i  $F(x)$  nie są tym samym
- Zmienna % przechowuje wynik ostatniego polecenia
- Zmienna %% przechowuje natomiast wynik ostatniego wyrażenia w ciągu instrukcji ograniczonych () (coś na kształt potoku)

```
(%i1) (integrate(sin(x),x),diff(%%,x),ev(%%,x=0));  
(%o1) 0
```



## ■ Operatory

- arytmetyczne  $+$ ,  $-$ ,  $*$ ,  $\backslash$
- potęgowanie  $^$ ,  $**$
- mnożenie macierzy  $.$
- silnia  $!$ , podwójna silnia  $!!$

## ■ Operator przypisania

- `:` przypisanie wartość do zmiennej lub wyrażenia `x:3, y:%o3`
- `:=` definicja funkcji, wymagany conajmniej jeden argument `ln(x):=log(x)/log(%e)`
- operatory `::` i `:=`

## ■ Stałe

- `%e` - podstawa logarytmu naturalnego
- `%i` - jednostka urojona
- `%pi` -  $\pi$
- `inf` -  $\infty$
- `minf` -  $-\infty$  (lepiej używać `minf` niż `-inf`)
- `%gamma` - stała Eulera
- `%c %k1 %k2` - stałe całkowania równań różniczkowych pierwszego i drugiego rzędu



## ■ Operatory

- arytmetyczne  $+$ ,  $-$ ,  $*$ ,  $\backslash$
- potęgowanie  $^$ ,  $**$
- mnożenie macierzy  $.$
- silnia  $!$ , podwójna silnia  $!!$

## ■ Operator przypisania

- $:$  przypisanie wartości do zmiennej lub wyrażenia  $x:3$ ,  $y:\%o3$
- $:=$  definicja funkcji, wymagany co najmniej jeden argument  $\ln(x) := \log(x)/\log(\%e)$
- operatory  $::$  i  $::=$

## ■ Stałe

- $\%e$  - podstawa logarytmu naturalnego
- $\%i$  - jednostka urojona
- $\%pi$  -  $\pi$
- $\inf$  -  $\infty$
- $\minf$  -  $-\infty$  (lepiej używać  $\minf$  niż  $-\inf$ )
- $\%gamma$  - stała Eulera
- $\%c \ \%k1 \ \%k2$  - stałe całkowania równań różniczkowych pierwszego i drugiego rzędu





## ■ Operatory

- arytmetyczne  $+$ ,  $-$ ,  $*$ ,  $\backslash$
- potęgowanie  $^$ ,  $**$
- mnożenie macierzy  $.$
- silnia  $!$ , podwójna silnia  $!!$

## ■ Operator przypisania

- $:$  przypisanie wartość do zmiennej lub wyrażenia  $x:3$ ,  $y:\%o3$
- $:=$  definicja funkcji, wymagany conajmniej jeden argument  $\ln(x) := \log(x)/\log(\%e)$
- operatory  $::$  i  $::=$

## ■ Stałe

- $\%e$  - podstawa logarytmu naturalnego
- $\%i$  - jednostka urojona
- $\%pi$  -  $\pi$
- $\inf$  -  $\infty$
- $\minf$  -  $-\infty$  (lepiej używać  $\minf$  niż  $-\inf$ )
- $\%gamma$  - stała Eulera
- $\%c \ \%k1 \ \%k2$  - stałe całkowania równań różniczkowych pierwszego i drugiego rzędu



## Podstawowe funkcje matematyczne:

- $\sin()$ ,  $\cos()$ ,  $\sec() = \frac{1}{\cos x}$ ,  $\tan()$ ,  $\cot()$  ...
- $\operatorname{asin}()$ ,  $\operatorname{acos}()$ ,  $\operatorname{asec}()$ ,  $\operatorname{atan}()$ ,  $\operatorname{acot}()$  ...
- $\sinh()$ ,  $\cosh()$ ,  $\tanh()$ ,  $\coth()$  ...
- $\operatorname{asinh}()$ ,  $\operatorname{acosh}()$ ,  $\operatorname{atanh}()$ ,  $\operatorname{acoth}()$  ...
- $\log()$ ,  $\operatorname{abs}()$ ,  $\operatorname{sqrt}()$ ,  $\exp()$  ...
- $\operatorname{erf}()$ ,  $\operatorname{beta}()$ ,  $\operatorname{gamma}()$ ,  $\operatorname{bessel}()$  ...



Inne ważne operatory i funkcje:

- ' (apostrof) instrukcja którą poprzedza nie jest wykonywana

```
(%i1) 'diff(sin(x),x);
```

$$(\%o1) \frac{d}{dx} \sin(x)$$

```
(%i4) 'integrate(x/(1+x^3),x)=integrate(x/(1+x^3),x);
```

$$(\%o4) \int \frac{x}{x^3+1} dx = \frac{\log(x^2-x+1)}{6} + \frac{\operatorname{atan}\left(\frac{2x-1}{\sqrt{3}}\right)}{\sqrt{3}} - \frac{\log(x+1)}{3}$$

- '' ponowne wykonanie instrukcji



# Podstawy. Listy

Podstawowym sposobem reprezentowania danych są listy:

```
(%i1) l:[1,2,3,5];  
(%o1) [1, 2, 3, 5]  
(%i2) l[2];  
(%o2) 2
```

- listą jest wszystko ujęte w klamry []
- pierwszy element listy ma indeks 1
- $l[n] = n$ -ty element listy  $l$



## Podstawowe funkcje operujące na listach

`makelist(expr,i,start,stop),makelist(expr,i,lista)`

Tworzy listę której elementy powstają poprzez obliczenie wartości wyrażenie `expr` dla wszystkich całkowitych `i` od `start` do `stop`

```
(%i1) L:makelist(sin(n/2*%pi),n,0,4);  
(%o1) [0, 1, 0, - 1, 0]  
(%i2) P:makelist(%i^n,n,0,4);  
(%o2) [1, %i, - 1, - %i, 1]  
(%i3) losowe:makelist(random(10),i,1,5);  
(%o3) [2, 2, 4, 5, 4]
```

W drugim wariancie zmienna `i` przybiera kolejno wartości elementów z listy `lista`

```
(%i4) makelist(2^n,n,[1,2,3]);  
(%o4) [2, 4, 8]
```

`append(lista1,lista2,...,listan)`

"Skleja" listy będące jej argumentami w jedną

```
(%i5) append([1,2,3,4],[a,b,c],[A,B,D,E]);  
(%o5) [1, 2, 3, 4, a, b, c, A, B, D, E]
```



## Podstawowe funkcje operujące na listach

`makelist(expr,i,start,stop),makelist(expr,i,lista)`

Tworzy listę której elementy powstają poprzez obliczenie wartości wyrażenie `expr` dla wszystkich całkowitych `i` od `start` do `stop`

```
(%i1) L:makelist(sin(n/2*pi),n,0,4);  
(%o1) [0, 1, 0, - 1, 0]  
(%i2) P:makelist(%i^n,n,0,4);  
(%o2) [1, %i, - 1, - %i, 1]  
(%i3) losowe:makelist(random(10),i,1,5);  
(%o3) [2, 2, 4, 5, 4]
```

W drugim wariancie zmienna `i` przybiera kolejno wartości elementów z listy `lista`

```
(%i4) makelist(2^n,n,[1,2,3]);  
(%o4) [2, 4, 8]
```

`append(lista1,lista2,...,listan)`

"Skleja" listy będące jej argumentami w jedną

```
(%i5) append([1,2,3,4],[a,b,c],[A,B,D,E]);  
(%o5) [1, 2, 3, 4, a, b, c, A, B, D, E]
```



`map(f,lista1,lista2,...,listan)`

Wynikiem działania funkcji jest lista

`[f(lista1[1],...,listan[1]),...,f(lista1[m],...,listan[m])]`

gdzie  $m$ =długość wszystkich list.

```
(%i1) map(sin,[0,1,2]);  
(%o1) [0, sin(1), sin(2)]  
(%i2) map("+",[1,2,3],[a,b,c]);  
(%o2) [a + 1, b + 2, c + 3]
```

bo  $a + b = +(a, b)$ . Aby skorzystać z własnej funkcji  $f$  musimy użyć funkcji `lambda`

```
(%i3) map(lambda([x],x^2),[1,2,3]);  
(%o3) [1, 4, 9]  
(%i4) map(lambda([x,y],x^2+y),[1,2,3],[a,b,c]);  
(%o4) [a + 1, b + 4, c + 9]
```



## `apply(f, lista)`

Wynikiem funkcji jest wartość wyrażenia `f(lista)`

```
(%i1) L:makelist(x^n,n,0,3);
(%o1)          2  3
      [1, x, x , x ]
(%i2) apply("+",L);
(%o2)          3  2
      x  + x  + x + 1
(%i3) apply("*",L);
(%o3)          6
      x
(%i4) apply(min,[1,2,-3]);
(%o4)          - 3
(%i5) (assume(x>1),apply(min,L));
(%o5)          1
```





`sort(L), sort(L,P)`

`sort(L)` sortuje listę w kolejności rosnącej.

W drugim wariancie `P` jest predykatem, czyli funkcją będącą kryterium sortowania.

```
(%i1) L: [-1,2,-4,0,8,1];
(%o1)          [- 1, 2, - 4, 0, 8, 1]
(%i2) sort(L);
(%o2)          [- 4, - 1, 0, 1, 2, 8]
(%i3) porownaj(x,y):=(if abs(x)>abs(y) then true else false)$
(%i4) sort(L,porownaj);
(%o4)          [8, - 4, 2, - 1, 1, 0]
```

Do dyspozycji mamy jeszcze wiele innych funkcji np. `lmin()`, `lmax()`, `member()`, `listp()`, `cons()` .... Dokładny spis w manualu.



`declare(a_1,p_1,a_2,p_2,...)`

Przypisuje atomowi  $a_i$  właściwość  $p_i$  (atom = np. liczba,zmienna,tekst)

- constant - atom jest stałą
- even, odd - parzyste/nieparzyste
- real, imaginary, complex
- **integer**

```
(%i1) declare(n,integer,C,constant);
(%o1)                                     done
(%i2) sin(n*pi);
(%o2)                                     0
(%i3) cos(n*pi);
(%o3)                                     n
(%i4) [diff(C),diff(y)];
(%o4) [0, del(y)]
```



## assume(zalozenie1,...,zalozenien)

Dodaje założenie co do wielkości występujących w danej sesji

```
(%i1) assume(x>0,y<-1,z>=0);  
(%o1) [x > 0, y < - 1, z >= 0]  
(%i2) facts();  
(%o2) [x > 0, - 1 > y, z >= 0]  
(%i3) is(x>y);  
(%o3) true  
(%i4) is(x^2>y^2);  
(%o4) unknown  
(%i5) is(exp(x)>exp(y));  
(%o5) true  
(%i6) is(z>0);  
(%o6) unknown  
(%i7) is(z+x>0);  
(%o7) true
```

## is(expr)

Określa czy expr może być prawdziwe

## forget(zalozenie1,...,zalozenien)

Zapomina dane założenie poczynione za pomocą assume

```
(%i8) forget(x>0);  
(%o8) [x > 0]  
(%i9) is(x>y);  
(%o9) unknown
```

## `assume(zalozenie1,...,zalozenien)`

Dodaje założenie co do wielkości występujących w danej sesji

```
(%i1) assume(x>0,y<-1,z>=0);  
(%o1) [x > 0, y < - 1, z >= 0]  
(%i2) facts();  
(%o2) [x > 0, - 1 > y, z >= 0]  
(%i3) is(x>y);  
(%o3) true  
(%i4) is(x^2>y^2);  
(%o4) unknown  
(%i5) is(exp(x)>exp(y));  
(%o5) true  
(%i6) is(z>0);  
(%o6) unknown  
(%i7) is(z+x>0);  
(%o7) true
```

## `is(expr)`

Określa czy `expr` może być prawdziwe

## `forget(zalozenie1,...,zalozenien)`

Zapomina dane założenie poczynione za pomocą `assume`

```
(%i8) forget(x>0);  
(%o8) [x > 0]  
(%i9) is(x>y);  
(%o9) unknown
```

`ev(expr, arg1, arg2, argn)`

Wykonuje (=oblicza) wyrażenie `expr` zgodnie z argumentami:

- `simp`
- `expand`
- `diff` oblicza wszystkie pochodne w wyrażeniu
- `float`
- **nouns** wykonuje wszystkie wyrażenia nominalne w `expr` np. `'diff()`, `'integrate()`, `'sum()`
- **var1=a, var2=b, ...** oblicza wartość wyrażenia gdy zmienne `var1`, `var2` są równe odpowiednio `a`, `b`

```
(%i1) sin(x)+cos(y)+'diff(sin(w),w);
```

```
(%o1) cos(y) + sin(x) +  $\frac{d}{dw}(\sin(w))$ 
```

```
(%i2) ev(%, nouns);
```

```
(%o2) cos(y) + sin(x) + cos(w)
```

```
(%i3) ev(%, x=0, y=%pi/2);
```

```
(%o3) cos(w)
```



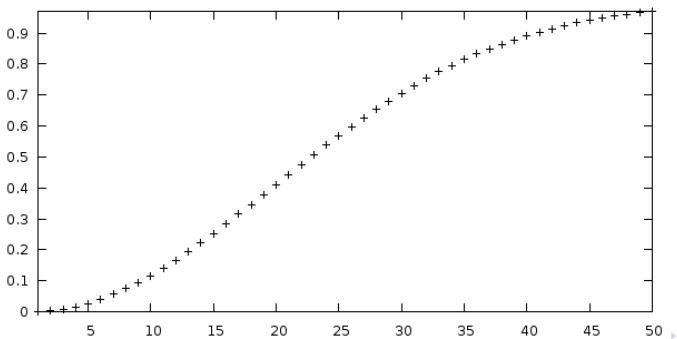
## Duże liczby - przykład

### Paradoks urodzinowy:

Jakie jest prawdopodobieństwo, że w grupie  $n$  osób przynajmniej dwie będą miały urodziny tego samego dnia

$$P(n) = 1 - \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365-n}{365} = 1 - \frac{1}{365^n} \frac{365!}{(365-n)!}$$

```
(%i1) P(n):=1-1/365^n*365!/(365-n)!$
(%i2) float(P(30));
(%o2) .7063162427192686
```



# Rozwijanie wyrażeń

`expand(expr), expand(expr,p,n)`

`expand()` jest podstawową funkcją używaną do rozwijania wyrażeń. Parametry `p` i `n` lokalnie zmieniają wartości zmiennych:

- `maxposex` *domyślnie: 1000* - gdy wykładnik wyrażenia jest większy niż `maxposex` wyrażenie nie jest rozwijane
- `maxnegex` *domyślnie: 1000* - gdy wykładnik wyrażenie jest mniejszy niż `-maxnegex` wyrażenie nie jest rozwijane

```
(%i1) expand((a+b)^5);
(%o1)      5      4      2 3      3 2      4      5
      b  + 5 a b  + 10 a  b  + 10 a  b  + 5 a  b + a
(%i2) expand((a+b)^5,3);
(%o2)                        5
                      (b + a)
```



Zachowanie programu określa również szereg zmiennych:

- `expop` *domyślnie*: 0 najwyższy wykładnik wyrażenie, które jest automatycznie rozwijane

```
(%i1) (a+b)^2;
```

```
(%o1)          2
          (b + a)
```

```
(%i2) (a+b)^2,expop=3;
```

```
(%o2)          2          2
          b  + 2 a b + a
```

- `expon` *domyślnie*: 0 -najniższy wykładnik wyrażenie, które będzie automatycznie rozwijane
- `logexpand` *domyślnie*: *true* kontroluje rozwijanie logarytmów  $\log(a \cdot b)$ ,  $\log(a^b)$ 
  - *true* wyrażenie typu  $\log(a^b)$  zostanie uproszczone do  $b \log(a)$
  - *all* dodatkowo wyrażenie typu  $\log(a \cdot b)$  zostanie uproszczone do  $\log(a) + \log(b)$
  - *false* brak uproszczeń
- `radexpand` *domyślnie*: *true* kontroluje upraszczanie pierwiastków





## ratexpand(expr)

Służy do rozwijania wielomianów. Szybsza od `expand()` oraz dokładniejsza w poszukiwaniu wspólnych dzielników:

```
(%i1) e1:(x-1)/(x+1)^2 + 1/(x-1);
```

```
(%o1)
      x - 1      1
      ----- + -----
            2      x - 1
      (x + 1)
```

```
(%i2) expand(e1);
```

```
(%o2)
      x          1          1
      ----- - ----- + -----
      2          2          x - 1
x  + 2 x + 1  x  + 2 x + 1
```

```
(%i3) ratexpand(e1);
```

```
(%o3)
      2          2
      2 x          2
      ----- + -----
      3      2      3      2
x  + x  - x - 1  x  + x  - x - 1
```



# Upraszczanie wyrażeń

`ratsimp(expr)`

Wykonuje uproszczenia na wyrażeniach wymiernych:

```
(%i1) sin(x/(x^2+x)) = exp((log(x)+1)^2-log(x)^2);
```

```
(%o1)          x      (log(x) + 1)2 - log(x)2
      sin(-----) = %e
            2
          x  + x
```

```
(%i2) ratsimp(%);
```

```
(%o2)          1      2
      sin(-----) = %e x
          x + 1
```



`fullratsimp(expr)`

Wielokrotnie stosuje uproszczenia wymiarne:

```
(%i1) expr: (x^(a/2) + 1)^2*(x^(a/2) - 1)^2/(x^a - 1);
```

```
(%o1)
      a/2      2      a/2      2
      (x  - 1) (x  + 1)
      -----
              a
              x  - 1
```

```
(%i2) ratsimp (expr);
```

```
(%o2)
      2 a      a
      x  - 2 x  + 1
      -----
              a
              x  - 1
```

```
(%i3) fullratsimp (expr);
```

```
(%o3)
      a
      x  - 1
```

```
(%i4) ratsimp(%o2);
```

```
(%o4)
      a
      x  - 1
```



## rootscontract(expr)

Zamienia iloczyn pierwiastków na pierwiastek iloczynu. O zachowaniu decyduje wartość zmiennej `rootsconmode`:

- *false* upraszcza jedynie pierwiastki których mianowniki są takie same
- *true* upraszcza również wtedy gdy mianowniki są swoimi wielokrotnościami
- *all* upraszcza zawsze, sprowadza mianowniki do NWW

```
(%i1) p1:x^(1/2)*y^(3/2)$
```

```
(%i2) p2:x^(1/2)*y^(1/4)$
```

```
(%i3) p3:x^(1/2)*y^(1/3)$
```

	<i>false</i>	<i>true</i>	<i>all</i>
p1	$\sqrt{xy^3}$	$\sqrt{xy^3}$	$\sqrt{xy^3}$
p2	$\sqrt{xy^{1/4}}$	$\sqrt{x\sqrt{y}}$	$(x^2y)^{1/4}$
p3	$\sqrt{xy^{1/3}}$	$\sqrt{xy^{1/4}}$	$(x^3y^2)^{1/6}$



## logcontract(expr)

Zamienia  $a1*\log(b1)+a2*\log(b2)+c$  na  $\log(\text{ratsimp}(b1^{a1}*b2^{a2}))+c$

```
(%i1) a*(a*log(x)+b*log(y)+a*d*log(z^2/z));
(%o1)      a (a d log(z) + b log(y) + a log(x))
(%i2) logcontract(%);
(%o2)      2              2
      a d log(z) + a b log(y) + a log(x)
```



## factor(expr)

Faktoryzuje wyrażenie tzn. zapisuje w takiej postaci aby ostatnim wykonywanym działaniem było mnożenie/dzielenie

```
(%i1) factor(2^65+3^65);
(%o1)          2
          5 11 79 131 4057 5981 149166625778220961
(%i2) factor(expand((x+1)^2*(x-1)^11));
(%o2)          11      2
          (x - 1) (x + 1)
(%i3) factor(1+x^12);
(%o3)          4      8      4
          (x  + 1) (x  - x  + 1)
(%i4) factor(x^2+1);
(%o4)          2
          x  + 1
```

## gfactor(expr)

Dopuszcza użycie liczb zespolonych przy faktoryzowaniu

```
(%i5) gfactor(x^2+1);
(%o5)          (x - %i) (x + %i)
(%i6) gfactor(x^4-1);
(%o6)          (x - 1) (x + 1) (x - %i) (x + %i)
(%i7) gfactor(x^6+1);
(%o7)          2      2
          (x - %i) (x + %i) (x  - %i x - 1) (x  + %i x - 1)
```

## factor(expr)

Faktoryzuje wyrażenie tzn. zapisuje w takiej postaci aby ostatnim wykonywanym działaniem było mnożenie/dzielenie

```
(%i1) factor(2^65+3^65);
(%o1)          2
              5 11 79 131 4057 5981 149166625778220961
(%i2) factor(expand((x+1)^2*(x-1)^11));
(%o2)          11      2
              (x - 1) (x + 1)
(%i3) factor(1+x^12);
(%o3)          4      8      4
              (x  + 1) (x  - x  + 1)
(%i4) factor(x^2+1);
(%o4)          2
              x  + 1
```

## gfactor(expr)

Dopuszcza użycie liczb zespolonych przy faktoryzowaniu

```
(%i5) gfactor(x^2+1);
(%o5)          (x - %i) (x + %i)
(%i6) gfactor(x^4-1);
(%o6)          (x - 1) (x + 1) (x - %i) (x + %i)
(%i7) gfactor(x^6+1);
(%o7)          2      2
              (x - %i) (x + %i) (x  - %i x - 1) (x  + %i x - 1)
```

`trigsimp(expr)`

Stosuje uproszczenia  $\sin^2 x + \cos^2 x = 1$  oraz  $\cosh^2 x - \sinh^2 x = 1$  do wyrażeń zawierających funkcje trygonometryczne i hiperboliczne

`trigreduce(expr)`

Zamienia potęgi i iloczyny wyrażeń trygonometrycznych i hiperbolicznych na krotności i sumy ich argumentów. Ruguje funkcje trygonometryczne z mianowników wyrażeń wymiernych

```
(%i1) trigreduce(-sin(x)^2+3*cos(x)^2+x);
```

```
(%o1)          cos(2 x)      cos(2 x)      1      1
      ----- + 3 (----- + -) + x - -
              2          2          2          2
```

```
(%i2) trigreduce(cos(x)*sin(y)+sin(x)*cos(y));
```

```
(%o2)          sin(y + x)
```





`trigsimp(expr)`

Stosuje uproszczenia  $\sin^2 x + \cos^2 x = 1$  oraz  $\cosh^2 x - \sinh^2 x = 1$  do wyrażeń zawierających funkcje trygonometryczne i hiperboliczne

`trigreduce(expr)`

Zamienia potęgi i iloczyny wyrażeń trygonometrycznych i hiperbolicznych na krotności i sumy ich argumentów. Ruguje funkcje trygonometryczne z mianowników wyrażeń wymiernych

```
(%i1) trigreduce(-sin(x)^2+3*cos(x)^2+x);
(%o1)
      cos(2 x)      cos(2 x)      1      1
----- + 3 (----- + -) + x - -
      2          2          2      2
(%i2) trigreduce(cos(x)*sin(y)+sin(x)*cos(y));
(%o2)
      sin(y + x)
```



## Pakiet ntrig

Po jego załadowaniu wyrażenia typu  $f(n\pi/10)$  gdzie  $f$  to  $\sin$ ,  $\cos$ ,  $\tan$ ,  $\csc$ ,  $\sec$ ,  $\cot$  będą domyślnie obliczane

```
(%i1) sin(11/10*%pi);
```

```
(%o1)          11 %pi  
          sin(-----)  
              10
```

```
(%i2) load(ntrig);
```

```
(%o2)      /usr/share/maxima/5.18.1/share/trigonometry/ntrig.mac
```

```
(%i3) sin(11/10*%pi);
```

```
(%o3)          1 - sqrt(5)  
          -----  
              4
```



Za uproszczenia trygonometryczne odpowiada również szereg zmiennych

- `trigsign true/false` wartość `true` powoduje uproszczenia typu  $\sin(-x) = -\sin x$
- `halfangles false/true`  $f(\frac{x}{2}) \Rightarrow F(f(x))$  gdzie  $f$  to dowolna funkcja trygonometryczna np:  
 $\sin(x/2)$ ;

$$\frac{\sqrt{1 - \cos x} (-1)^{\lfloor \frac{x}{2\pi} \rfloor}}{\sqrt{2}}$$

- `triginverses true/all/false` kontroluje upraszczanie złożenia funkcji trygonometrycznych/hyperbolicznych z ich funkcjami odwrotnymi
  - `all`:  $af(f(x))$  i  $f(af(x)) = x$
  - `true`:  $af(f(x))$  nie podlega uproszczeniu
  - `false`: żadne uproszczenia nie są wykonywane

gdzie  $f$  i  $af$  to odpowiednio dowolna funkcja trygonometryczna lub hiperboliczna i jej funkcja odwrotna.



`trigexpand(expr)`

Rozwija wyrażenia zawierające funkcje trygonometryczne. Zachowanie zależy od zmiennych:

- `halfangles`
- `trigexpandplus` *true/false* rozwija funkcje, których argumentem jest suma/różnica np.  $\sin(x+y)$
- `trigexpandtimes` *true/false* rozwija funkcje, których argumentem jest iloczyn  $\sin(2*x)$



# Macierze

`matrix([wiersz1],..., [wierszn])`

Tworzy macierz składającą się z  $n$  wierszy. Każdy wiersz jest równoliczną listą.  
Dozwolone operacje:

- $+$ ,  $-$ ,  $*$ ,  $\backslash$  odpowiadających sobie elementów
- $^$  potęgowanie elementów  $A^3 = A * A * A$
- $.$  mnożenie macierzy (nieprzemienne)
- $^^$  potęgowanie macierzy  $A^^3 = A . A . A$

`entermatrix(m,n)`

Interaktywny sposób wprowadzania macierzy  $m \times n$ .

```
(%i1) M:entermatrix(3,3);
```

```
Is the matrix  1. Diagonal  2. Symmetric  3. Antisymmetric  4. General
```

```
Answer 1, 2, 3 or 4 :
```

```
2;
```

```
Row 1 Column 1:
```



Wybrane funkcje operujące na macierzach:

- `determinant(M)` wyznacznik macierzy `M`
- `adjoint(M)` macierz dołączona do `M`
- `invert(M)` macierz odwrotna do `M`

(%i1)M:

```
matrix([a1,b1,c1,d1],[a2,b2,c2,d2],[a3,b3,c3,d3],[a4,b4,c4,d4])
```

$$\begin{pmatrix} a1 & b1 & c1 & d1 \\ a2 & b2 & c2 & d2 \\ a3 & b3 & c3 & d3 \\ a4 & b4 & c4 & d4 \end{pmatrix}$$

(%i2)ratsimp(M.invert(M))

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Wybrane funkcje operujące na macierzach cd:

- `charpoly(M, x)` wielomian charakterystyczny  $\det(M - xI)$
- `rank(M)` rząd macierzy  $M$
- `transpose(M)` macierz transponowana
- `mat_trace(M)` ślad macierzy



Po załadowaniu pakietu `eigen load(eigen)` do dyspozycji mamy również:

- `eigenvalues(M)` zwraca listę złożoną z dwóch list. Pierwsza z nich to lista wartości własnych, druga to ich krotności.

```
(%i1) load(eigen);
(%o1) /usr/share/maxima/5.18.1/share/matrix/eigen.mac
(%i2) A:matrix([0,0,1],[0,1,0],[1,0,0])$

(%i3) eigenvalues(A);
(%o3) [[- 1, 1], [1, 2]]
(%i4) charpoly(A,a);
(%o4) (1 - a) a2 + a - 1
(%i5) solve(%,a);
(%o5) [a = - 1, a = 1]
(%i6) multiplicities;
(%o6) [1, 2]
```

- `eigenvectors(M)` zwraca wartości i wektory własne:

■ **Maxima < 5.19.x**

Pierwszy element listy to wartości wynik polecenia `eigenvalues(M)` a kolejne to wektory własne

```
(%i6) eigenvectors(A);
(%o6) [[[- 1, 1], [1, 2]], [1, 0, - 1], [1, 0, 1], [0, 1, 0]]
```





Po załadowaniu pakietu `eigen` `load(eigen)` do dyspozycji mamy również:

- `eigenvalues(M)` zwraca listę złożoną z dwóch list. Pierwsza z nich to lista wartości własnych, druga to ich krotności.

```
(%i1) load(eigen);
(%o1) /usr/share/maxima/5.18.1/share/matrix/eigen.mac
(%i2) A:matrix([0,0,1],[0,1,0],[1,0,0])$

(%i3) eigenvalues(A);
(%o3) [[- 1, 1], [1, 2]]

(%i4) charpoly(A,a);
(%o4) (1 - a) a2 + a - 1

(%i5) solve(%,a);
(%o5) [a = - 1, a = 1]

(%i6) multiplicities;
(%o6) [1, 2]
```

- `eigenvectors(M)` zwraca wartości i wektory własne:

- **Maxima  $\geq 5.19.x$**

Pierwszy element listy to wartości wynik polecenia `eigenvalues(M)`. Następny to lista, której każdy element to wszystkie wektory własne odpowiadające danej wartości własnej

```
(%i6) eigenvectors(A);
(%o6) [[[-1,1], [1,2]], [[1,0,-1]], [[1,0,1], [0,1,0]]]
```



# Liczby zespolone

## `conjugate(x)`

Zwraca liczbę sprzężoną do  $x$

```
(%i1) z1:a+b*%i;  
(%o1)  $a + bi$   
(%i2) declare(R,real,I,imaginary);  
(%o2) done  
(%i3) conjugate([z1,R,I]);  
(%o3)  $[a - bi, R, -I]$ 
```

## `abs(expr)`

Zwraca wartość bezwzględna (moduł) wyrażenia  $expr$

## `realpart(expr)`, `imagpart(expr)`

Zwraca część rzeczywistą/urojoną  $expr$



# Liczby zespolone

## `conjugate(x)`

Zwraca liczbę sprzężoną do  $x$

```
(%i1) z1:a+b*i;  
(%o1) %i b + a  
(%i2) declare(R,real,I,imaginary);  
(%o2) done  
(%i3) conjugate([z1,R,I]);  
(%o3) [a - %i b, R, - I]
```

## `abs(expr)`

Zwraca wartość bezwzględną (moduł) wyrażenia  $expr$

## `realpart(expr)`, `imagpart(expr)`

Zwraca część rzeczywistą/urojoną  $expr$



# Liczby zespolone

## `conjugate(x)`

Zwraca liczbę sprzężoną do  $x$

```
(%i1) z1:a+b*i;  
(%o1) %i b + a  
(%i2) declare(R,real,I,imaginary);  
(%o2) done  
(%i3) conjugate([z1,R,I]);  
(%o3) [a - %i b, R, - I]
```

## `abs(expr)`

Zwraca wartość bezwzględną (moduł) wyrażenia  $expr$

## `realpart(expr), imagpart(expr)`

Zwraca część rzeczywistą/urojoną  $expr$



## carg(z)

Zwraca argument główny liczby  $z$

```
(%i1) carg(1);
(%o1) 0
(%i2) carg(exp(%pi*i));
(%o2) %pi
```

## polarform(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci trygonometrycznej  $r \exp(i\varphi)$  gdzie  $r, \varphi \in \mathbb{R}$

```
(%i1) polarform(a+%i*b);
(%o1) sqrt(b^2 + a^2) %e %i atan2(b, a)
(%i2) polarform(%e*(cos(2)-%i*sin(2)));
(%o2) sqrt(%e^2 sin(2)^2 + %e^2 cos(2)^2) %e %i (-atan2(sin(2), cos(2)) - %pi)
(%i3) trigreduce(%);
(%o3) 1 - 2 %i %e
```



## carg(z)

Zwraca argument główny liczby  $z$

```
(%i1) carg(1);
(%o1) 0
(%i2) carg(exp(%pi*i));
(%o2) %pi
```

## polarform(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci trygonometrycznej  $r \exp(i\varphi)$  gdzie  $r, \varphi \in \mathbb{R}$

```
(%i1) polarform(a+%i*b);
(%o1) sqrt(b^2 + a^2) %e %i atan2(b, a)
(%i2) polarform(%e*(cos(2)-%i*sin(2)));
(%o2) sqrt(%e^2 sin^2(2) + %e^2 cos^2(2)) %e %i (-atan2(sin(2), cos(2)) - %pi)
(%i3) trigreduce(%);
(%o3) 1 - 2 %i %e
```



### rectform(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci dwumianowej  $a + ib$  gdzie  $a, b \in \mathbb{R}$

```
(%i6) rectform(%o1);  
      2      2  
Is b + a  positive or zero? p;  
(%o6)      %i b + a
```

### demoivre(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci trygonometrycznej  $r(\cos \varphi + i \sin \varphi)$  gdzie  $r = |z|$ ,  $\varphi = \text{Arg}(z)$

```
(%i7) demoivre(%o2),trigreduce;  
(%o7)      %e %i sin(2) - %e cos(2)
```



### rectform(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci dwumianowej  $a + ib$  gdzie  $a, b \in \mathbb{R}$

```
(%i6) rectform(%o1);
      2      2
Is b + a positive or zero? p;
(%o6)          %i b + a
```

### demoivre(z)

Zapisuje liczbę  $z \in \mathbb{C}$  w postaci trygonometrycznej  $r(\cos \varphi + i \sin \varphi)$  gdzie  $r = |z|$ ,  $\varphi = \text{Arg}(z)$

```
(%i7) demoivre(%o2), trigreduce;
(%o7)          %e %i sin(2) - %e cos(2)
```





# Równania liniowe

`linsolve(lista_rownan, lista_niewiadomych)`

`linsolve` rozwiązuje układy równań liniowych:

$$\begin{cases} 2x + 4y + 0.2z + 11p = 8 \\ \frac{11}{12}x + y = 3 \\ \frac{3}{2}x + y + \frac{7}{2}z = -1 \\ x + y + z + p = 0 \end{cases}$$

```
(%i1) eq1:2*x+4*y+0.2*z+11*p=8$
```

```
(%i2) eq2:11/12*x+y=3$
```

```
(%i3) eq3:3/2*x+7/2*z+y=-1$
```

```
(%i4) eq4:x+y+z+p=0$
```

```
(%i5) rown:[eq1,eq2,eq3,eq4]$
```

```
(%i6) rozw:linsolve(rown,[x,y,z,p]);
```

```
rat: replaced 0.2 by 1/5 = 0.2
```

```
(%o6)      6996      7400      790      1194
      [x = - ----, y = ----, z = ---, p = - ----]
```

```
(%i7) ev(rown,rozw);
```

```
(%o7)      [8.0 = 8, 3 = 3, - 1 = - 1, 0 = 0]
```



# Równania. solve

## `solve(expr, var)`

Funckja `solve()` stara się rozwiązać symbolicznie dowolne równanie lub układ równań. Pierwszym argumentem jest wyrażenie zawierające niewiadomą `var`. Może być ono dane w trzech postaciach:

- $f(x)=g(x)$
- $f(x)=0$
- $f(x)$

Dwie ostatnie postaci są równoważne. Gdy równanie zawiera tylko jedną zmienną, można opuścić drugi argument. Wynik zwrócony jest w postaci listy.



```
(%i1) eq1:expand((x+1)^2*(x-1));
(%o1)          3      2
          x  + x  - x - 1
(%i2) solve(eq1);
(%o2)          [x = 1, x = - 1]
(%i3) multiplicities;
(%o3)          [1, 2]
```

Lista multiplicites przechowuje krotności znalezionych pierwiastków.

Maxima stara się znajdować dokładne rozwiązania. Czasami wiąże się to z utratą rozwiązań - wyświetlany jest wtedy stosowny komunikat:

```
(%i9) solve(sin(x)=1/2);

solve: using arc-trig functions to get a solution.
Some solutions will be lost.
```

```
(%o9)          %pi
          [x = ---]
          6
```

Rozwiązania nie muszą być rzeczywiste:

```
(%i11) solve(x^3-1);
(%o11)          sqrt(3) %i - 1      sqrt(3) %i + 1
          [x = -----, x = - -----, x = 1]
          2                        2
```



Rozwiążemy równanie kwadratowe i sprawdzimy rozwiązania:

```
(%i1) f(x):=a*x^2+b*x+c;
```

```
(%o1)          2
      f(x) := a x  + b x + c
```

```
(%i2) rozw:solve(f(x),x);
```

```
(%o2)          2          2
      sqrt(b  - 4 a c) + b      sqrt(b  - 4 a c) - b
[x = - ----, x = ----]
      2 a          2 a
```

```
(%i3) wart:map(rhs,rozw);
```

```
(%o3)          2          2
      sqrt(b  - 4 a c) + b      sqrt(b  - 4 a c) - b
[- ----, ----]
      2 a          2 a
```

```
(%i4) (map(f,wart),expand(%));
```

```
(%o4) [0, 0]
```



`solve([eqn1,...,eqnn],[x1,...,xn])`

Rozwiązuje układ  $n$  równań algebraicznych o  $n$  niewiadomych.

```
(%i1) rown:[x^2+y^2=1,x+3*y=0];
(%o1)

$$\begin{matrix} 2 & 2 \\ y & + & x & = & 1, & 3 & y & + & x & = & 0 \end{matrix}$$

(%i2) rozw:solve(rown,[x,y]),rootscontract;
(%o2)

$$\begin{matrix} 3 & 1 & 3 & 1 \\ [x = - \frac{\sqrt{10}}{3}, y = \frac{1}{\sqrt{10}}], & [x = \frac{\sqrt{10}}{3}, y = - \frac{1}{\sqrt{10}}] \end{matrix}$$

(%i3) for i thru 2 do
      for j thru 2 do (
        ev(rown[i],rozw[j]),disp(lhs(%)-rhs(%)) );
        0
        0
        0
        0
```



# Metody numeryczne

`realroots(W)`, `realroots(W,dokladnosc)`

Znajduje metodą bisekcji wszystkie rzeczywiste pierwiastki wielomianu. Współczynniki wielomianu muszą być liczbami wymiernymi (nie %pi, %e). Drugi parametr definiuje dokładność z jaką szukany jest pierwiastek - domyślnie jest to  $10^{-7}$ .

```
(%i1) W:-x^5-x^4-3*x^3+x^2+1$
(%i2) solve(W);
(%o2)          5      4      3      2
[0 = x  + x  + 3 x  - x  - 1]
(%i3) rozw:realroots(W);
(%o3)          23799551
[x = -----]
          33554432
(%i4) float(ev(W,rozw));
(%o4)          - 1.6116037715156115e-7
(%i5) rozw2:realroots(W,1e-12);
(%o5)          1559727313253
[x = -----]
          2199023255552
(%i6) float(ev(W,rozw2));
(%o6)          1.454998872258234e-12
```



## `allroots(W)`

Numerycznie znajduje przybliżone wartości rzeczywistych i zespolonych pierwiastków wielomianu  $W$  jednej niewiadomej. Gdy  $W$  jest rzeczywistym wielomianem użycie `allroots(%i*W)` może podać dokładniejsze wyniki niż `allroots(W)`

```
(%i7) allroots(W);  
(%o7) [x = .7092818638073147, x = 0.62408254372636 %i - .2232644778356121,  
x = - 0.62408254372636 %i - .2232644778356121,  
x = 1.676467898169772 %i - .6313764540680453,  
x = - 1.676467898169772 %i - .6313764540680453]
```

w tym wypadku `allroots(W)` i `allroots(%i*W)` zwracają praktycznie ten sam wynik

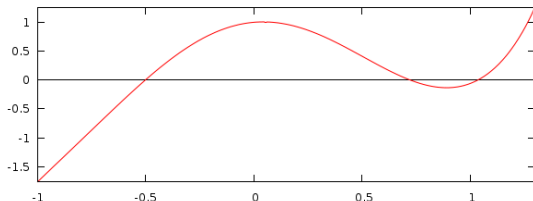


`find_root(f,a,b)`

Numerycznie znajduje pierwiastek funkcji  $f$  wewnątrz przedziału  $[a, b]$

- Gwarantuje znalezienie pierwiastka jeśli istnieje
- Kończy działanie po znalezieniu pierwszego (=najmniejszego) pierwiastka
- Wartości  $f(a)$  i  $f(b)$  muszą być różnych znaków tj.  $f(a)f(b) < 0$

Szukamy pierwiastków równania  $f(x) = \exp(x^2) \sin(\frac{x}{\pi}) + \cos(ex)$  w przedziale  $[-1, 1.5]$



```
(%i1) f(x):=exp(x^2)*sin(x/%pi)+cos(%e*x)$
(%i2) fpprintprec:9$
(%i3) find_root(f(x),-1,3/2);
(%o3) - .502014095
(%i4) float(f(%));
(%o4) 1.66533454e-16
```





# Metoda Newtona

`newton(expr, var, x_0, eps)`

Zwraca przybliżoną wartość pierwiastka wyrażenia `expr` zmiennej `var` znalezionej metodą Newtona (metodą stycznych). Poszukiwania zaczyna od `x=x_0` a kończy gdy `abs(expr)<eps`. Porównanie to, nakłada jedyny warunek na postać `expr`. Funkcja znajduje się w pakiecie `newton1.mac`

```
(%i6) load(newton1);
(%o6) /usr/share/maxima/5.18.1/share/numeric/newton1.mac
(%i7) newton(f(x),x,-1/2,1/1000);

                                0.5
                                .210078698 - 1.28402542 sin(---)
                                %pi
(%o7)  - ----- - 0.5
                                0.5
                                1.28402542 cos(---)
                                %pi
                                0.5
                                1.28402542 sin(---) + ----- + 2.65762187
                                %pi                                %pi

(%i8) float(%);
(%o8) - .502015933
(%i9) float(f(%));
(%o9) - 6.013395935e-6
```



# Wykresy

Maxima wszystkie wykresy tworzy domyślnie za pośrednictwem gnuplota. Komunikacja między programami odbywa się na dwa sposoby, w zależności od metody rysowania

- za pośrednictwem potoków - lubi nie działać w systemach MS Windows
- za pośrednictwem plików tymczasowych - mniej elegancja ale skuteczniejsza forma

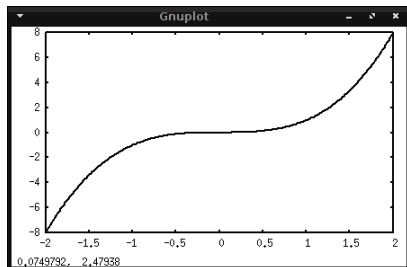
Omawiany pakiet `draw.mac` działa w oparciu o drugą metodę



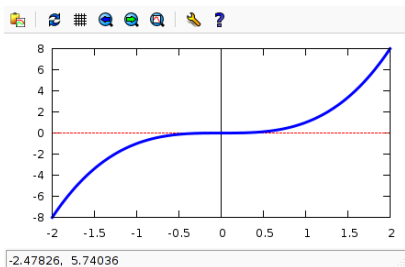
## Pakiet draw.mac

- Wymaga Gnupota  $> 4.2$ , Maxima  $> 5.14$
- Pakiet włączamy poleceniem `load(draw)` - trwa to chwile
- Możliwości
  - Wykresy 2D ( $\mathbb{R} \rightarrow \mathbb{R}$ )
  - Wykresy 3D ( $\mathbb{R}^2 \rightarrow \mathbb{R}$ )
  - Wykresy parametryczne
  - Wykresy funkcji uwikłanych
  - Kontury
  - "Animacje"
  - ...



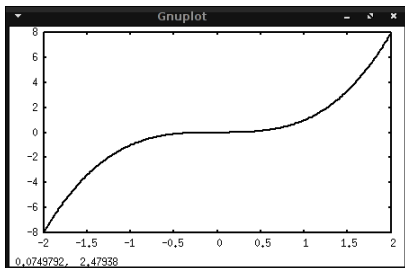


```
draw2d(explicit(x^3,x,-2,2));
```

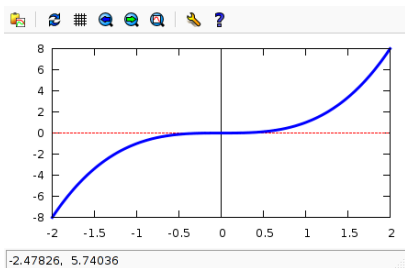


```
draw2d(
  line_width=3,
  color=blue,
  xaxis=true,
  xaxis_color=red,
  yaxis=true,
  yaxis_type=solid,
  explicit(x^3,x,-2,2),
  terminal=wxr)
```



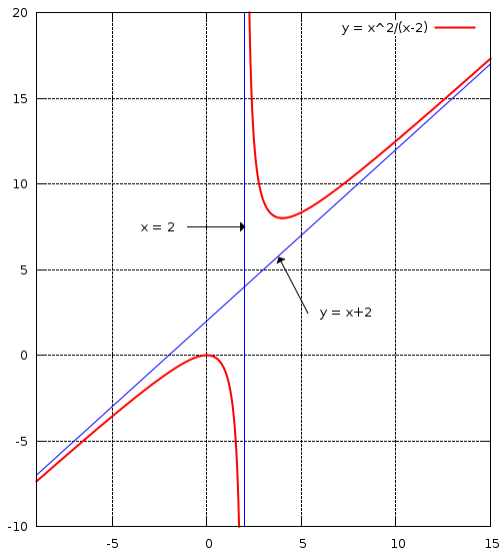


```
draw2d(explicit(x^3,x,-2,2));
```



```
draw2d(
  line_width=3,
  color=blue,
  xaxis=true,
  xaxis_color=red,
  yaxis=true,
  yaxis_type=solid,
  explicit(x^3,x,-2,2),
  terminal=wxt)
```





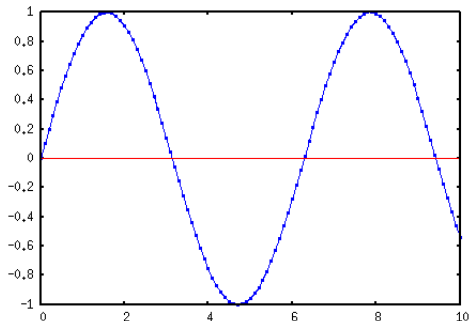
```
draw2d(
  line_width=2,
  grid = true,
  key   = "y = x^2/(x-2)",
  yrange = [-10,20],
  color = red,
  explicit(x^2/(x-2),x,-9,15),
/*asymptoty*/
  line_width=1,
  key       = "",
  line_type = solid,
  color     = blue,
  explicit(x+2,x,-9,15),
  nticks = 70,
  parametric(2,t,t,-10,20),
/*strzałki*/
  head_length = 0.3,
  color       = black,
  line_type   = solid,
  vector([5.35,2.45],[-1.53,3.25]),
  vector([-1,7.5],[3,0]),
  label_alignment = left,
  label(["y = x+2",6,2.5]),
  label_alignment = right,
  label(["x = 2",-1.7,7.5]),
terminal=wxt);
```



# Zbiory punktów

Do rysowania zbiorów punktów używamy polecenia `points(arg)` gdzie `arg` to współrzędne punktów w postaci:

- $[[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]]$
- $[[x_1, x_2, \dots, x_n], [y_1, y_2, \dots, y_n]]$



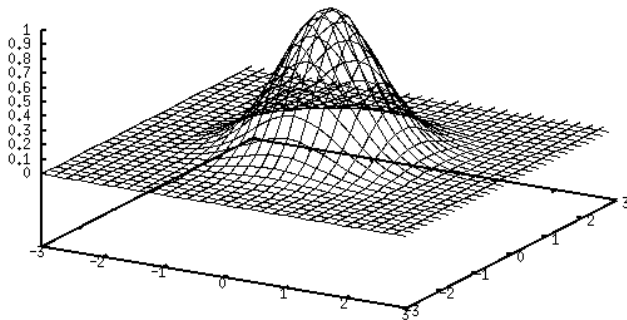
4.82778, 0.372613

```
L:=makelist([i/10,sin(i/10)],  
            i,0,100)$  
  
draw2d(  
  point_type=6,  
  color=blue,  
  points_joined=true,  
  points(L),  
  color=red,  
  explicit(0,x,0,10));
```



# Wykresy 3D

```
draw3d(explicit(exp(-x^2-y^2),x,-3,3,y,-3,3))
```



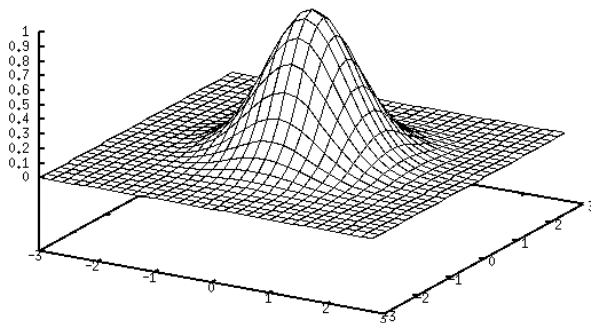
view: 60.0000, 30.0000 scale: 1.00000, 1.00000





Użycie opcji `hide_surface=true` powoduje ukrycie niewidocznych linii

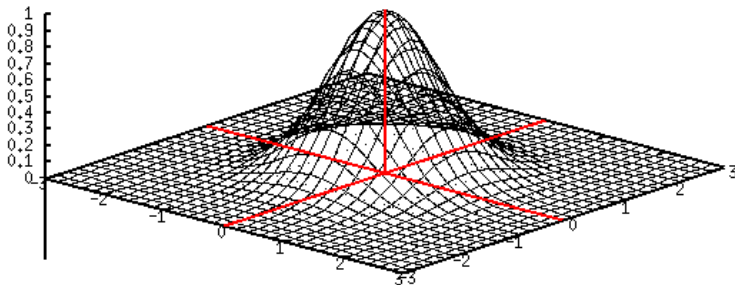
```
draw3d(hide_surface=true,explicit(exp(-x^2-y^2),x,-3,3,y,-3,3))
```



view: 60.0000, 30.0000 scale: 1.00000, 1.00000



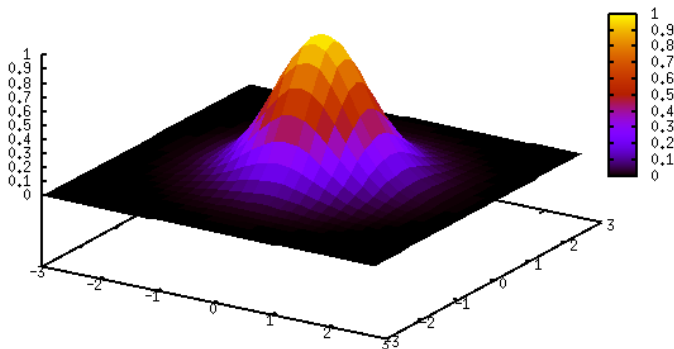
```
draw3d(  
  xaxis=true, xaxis_type=solid, xaxis_color=red, xaxis_width=2,  
  yaxis=true, yaxis_type=solid, yaxis_color=red, yaxis_width=2,  
  zaxis=true, zaxis_type=solid, zaxis_color=red, zaxis_width=2,  
  grid=true,  
  user_preamble="set xyplane at 0",  
  explicit(exp(-x^2-y^2),x,-3,3,y,-3,3));
```



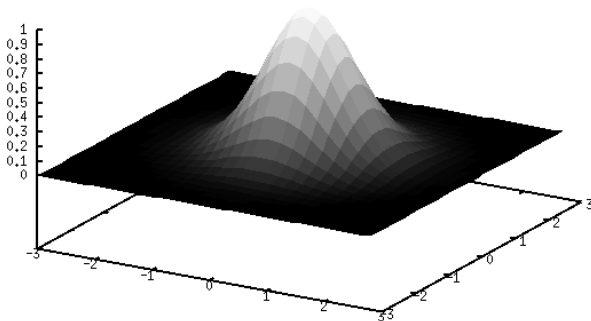
## Kolorowanie wykresu

Do kolorowania wykresu zarządza zmienna `enhanced3d`, odpowiednik opcji `setpm3d`, `sethidden3d` z `gnuplot`.

```
draw3d(enhanced3d=true,explicit(exp(-x^2-y^2),x,-3,3,y,-3,3))
```



```
draw3d(enhanced3d=true,palette=gray,colorbox=false,  
explicit(exp(-x^2-y^2),x,-3,3,y,-3,3))
```

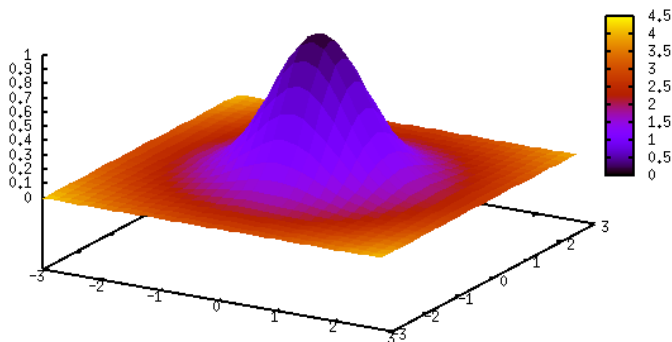


view: 60.0000, 30.0000 scale: 1.00000, 1.00000



Domyślnie kolor odpowiada wartości funkcji w danym punkcie. Do zmiennej `enhanced3d` możemy przypisać dowolną funkcję, której wartość będzie odwzorowywana w kolor

```
draw3d(enhanced3d=sqrt(x^2+y^2),explicit(exp(-x^2-y^2),x,-3,3,y,-3,3))
```

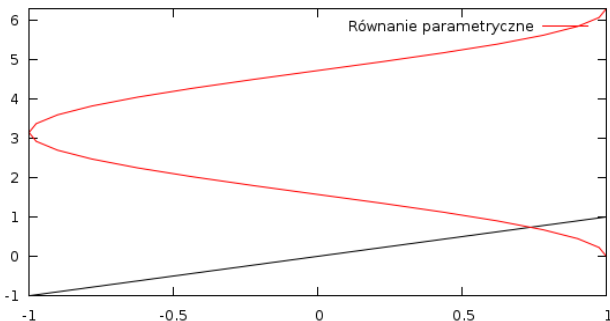


view: 60.0000, 30.0000 scale: 1.00000, 1.00000



# Wykresy parametryczne

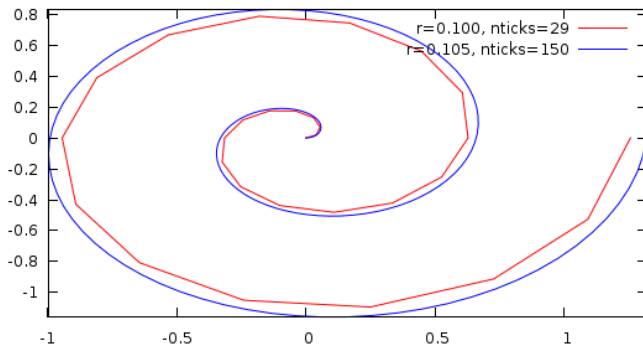
Użycie polecenia `parametric()` pozwala na tworzenie wykresów funkcji parametrycznych.



$$\begin{cases} y = 6x \\ x = \cos t \\ y = t \end{cases}$$

```
draw2d(explicit(6x,x,0,1),  
color=red,key="Rownanie parametryczne",parametric(cos(t),t,t,0,2*%pi),terminal=wxt);
```

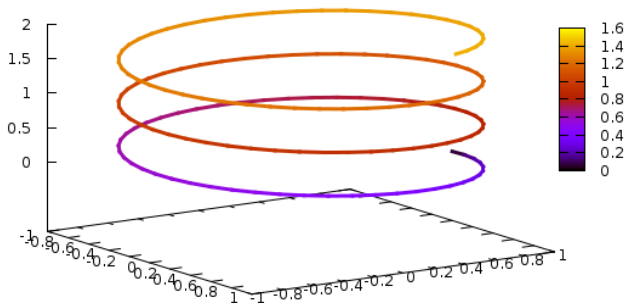




$$\begin{cases} x &= & rt \cos t \\ y &= & rt \sin t \end{cases}$$

```
draw2d(
color=red,key="r=0.100, nticks=29",
parametric(0.1*t*cos(t),0.1*t*sin(t),t,0,4*pi),
color=blue,key="r=0.105, nticks=150",nticks=150,
parametric(0.105*t*cos(t),0.105*t*sin(t),t,0,4*pi),
terminal=wxt);
```





$$\begin{cases} x &= & rt \cos t \\ y &= & rt \sin t \\ z &= & t \end{cases}$$

```
draw3d(enhanced3d=sqrt(t),line_width=3,nticks=150,
parametric(sin(10*t),cos(10*t),t,t,0,2),terminal=wxt);
```

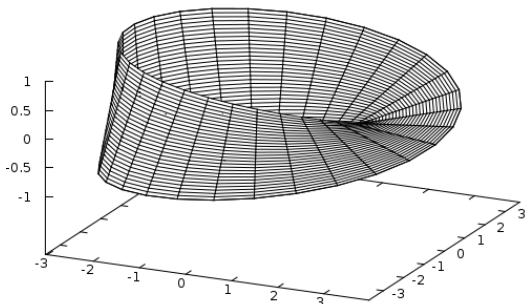




# Rysowanie powierzchni

Do rysowania powierzchni używamy polecenia

`parametric_surface(x(u,v),y(u,v),z(u,v),u,u_min,u_max,v,v_min,v_max)`

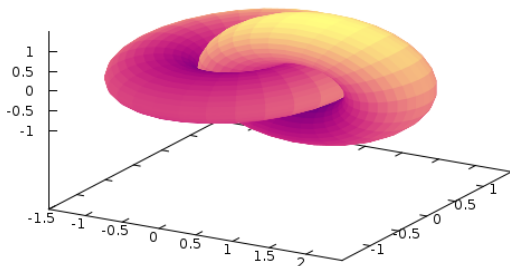


Wstęga Mobiusa:

$$\begin{cases} x &= \cos \varphi \left( 3 + r \cos \left( \frac{\varphi}{2} \right) \right) \\ y &= \sin \varphi \left( 3 + r \cos \left( \frac{\varphi}{2} \right) \right) \\ z &= r \sin \left( \frac{\varphi}{2} \right) \end{cases}$$

```
draw3d(surface_hide=true,
  parametric_surface(cos(a)*(3+b*cos(a/2)),
    sin(a)*(3+b*cos(a/2)),
    b*sin(a/2),
    a,-%pi,%pi,b,-1,1),terminal=wxt)$
```





Torus:

$$\begin{cases} x &= \cos u + \frac{1}{2} \cos u \cos v \\ y &= \sin u + \frac{1}{2} \sin u \cos v \\ z &= \frac{1}{2} \sin v \end{cases}$$

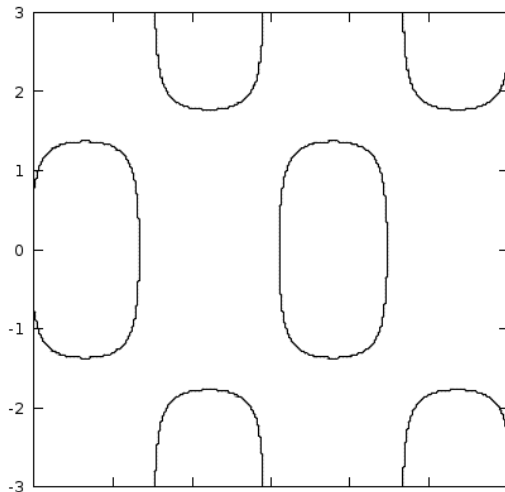
```
draw3d(enhanced3d = sin(u)+cos(v),
terminal    = wxt,
colorbox=false,
palette     = [8,4,1],
parametric_surface(cos(u)+.5*cos(u)*cos(v),
                    sin(u)+.5*sin(u)*cos(v),
                    .5*sin(v),
                    u, -%pi, %pi,
                    v, -%pi, %pi),
parametric_surface(1+cos(u)+.5*cos(u)*cos(v),
                    .5*sin(v),
                    sin(u)+.5*sin(u)*cos(v),
                    u, -%pi, %pi,
                    v, -%pi, %pi)) $
```



## Wykresy funkcji uwikłanych

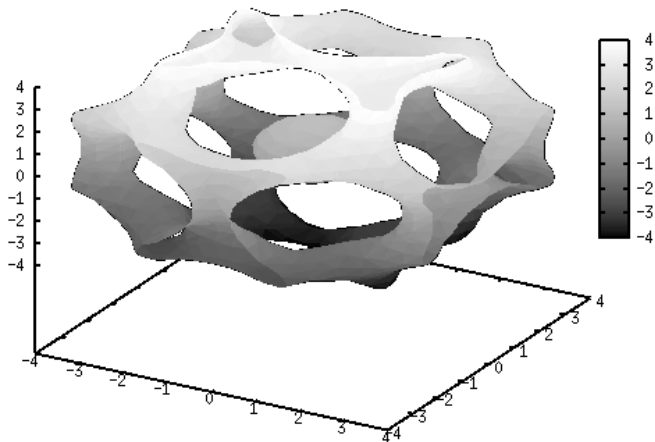
Pakiet `draw.mac` za pośrednictwem polecenie `implicit()` umożliwia rysowanie wykresów funkcji uwikłanych zarówno 2D jak i 3D

```
draw2d(user_preamble="set size ratio 1",implicit(sin(2*x)*cos(y)=0.2,x,-3,3,y,-3,3))
```



```
draw3d(x_voxel      = 20,  
       y_voxel      = 20,  
       z_voxel      = 20,  
       enhanced3d    = true,  
       palette       = gray,  
       surface_hide  = true,  
       user_preamble="set hidden3d",  
       implicit(2=(cos(x+%phi*y)+cos(x-%phi*y)  
                  +cos(y+%phi*z)+cos(y-%phi*z)  
                  +cos(z-%phi*x)+cos(z+%phi*x)),  
               x,-4,4,y,-4,4,z,-4,4))$
```





view: 60.0000, 30.0000 scale: 1.00000, 1.00000



# Operacja na plikach

Trzy zmienne definiują położenia w których maxima szuka plików:

- `file_search_maxima`
- `file_search_lisp`
- `file_search_demos`
- `maxima_userdir`
- bieżący katalog

Funckja `file_search("nazwa")` sprawdza czy w którymś z powyższych katalogów istnieje plik o danej nazwie.



```
printfile("nazwa")
```

Drukuje na ekran zawartość pliku nazwa:

```
(%i4) printfile("/etc/resolv.conf");  
nameserver 89.108.195.20  
nameserver 89.108.195.21  
(%o4)                               /etc/resolv.conf
```

```
with_stdout("plik", expr1, expr2, ..., exprn)
```

Wyniki wyrażeń `expr1 ... exprn` zostają zapisane do pliku `plik`. O trybie zapisu decyduje wartość zmiennej `file_output_append`

- `true` dopisywanie do pliku
- `inna` nadpisanie pliku

W obu wypadkach, gdy plik nie istnieje jest tworzony. Plik zapisywany jest w formacie unix - pojedynczy znak (LF) kończy linię. Korzystając z MS Windows musimy więc uzbroić się w odpowiedni edytor tekstu.



```
printfile("nazwa")
```

Drukuje na ekran zawartość pliku nazwa:

```
(%i4) printfile("/etc/resolv.conf");
nameserver 89.108.195.20
nameserver 89.108.195.21
(%o4)                                     /etc/resolv.conf
```

```
with_stdout("plik", expr1, expr2, ..., exprn)
```

Wyniki wyrażeń `expr1 ... exprn` zostają zapisane do pliku `plik`. O trybie zapisu decyduje wartość zmiennej `file_output_append`

- `true` dopisywanie do pliku
- `inna` nadpisanie pliku

W obu wypadkach, gdy plik nie istnieje jest tworzony. Plik zapisywany jest w formacie unix - pojedynczy znak (LF) kończy linie. Korzystając z MS Windows musimy więc uzbroić się w odpowiedni edytor tekstu.





```
write_data(ob,"plik")
```

Uniwersalna metoda zapisu. Zapisuje obiekt ob do plikuplik.

## Czytanie z pliku

Istnieje wiele funckji czytających dane z pliku:

- `read_list("plik")` zwraca zawartość pliku w postaci listy. Białe znaki w pliku oddzielają elementy listy
- `read_nested_list("plik")` zwraca zagnieżdżona listę. Wiersze odpowiadają kolejnym *podlistą*, których elementami jest zawartość wierszy
- `read_matrix("plik")` jw. z tym, że zwracana jest macierz

Wszystkie one występują w kilku wariantach pozwalających np. zdefiniować separator



```
write_data(ob,"plik")
```

Uniwersalna metoda zapisu. Zapisuje obiekt `ob` do pliku `plik`.

## Czytanie z pliku

Istnieje wiele funkcji czytających dane z pliku:

- `read_list("plik")` zwraca zawartość pliku w postaci listy. Białe znaki w pliku oddzielają elementy listy
- `read_nested_list("plik")` zwraca zagnieżdżoną listę. Wiersze odpowiadają kolejnym *podlistą*, których elementami jest zawartość wierszy
- `read_matrix("plik")` jw. z tym, że zwracana jest macierz

Wszystkie one występują w kilku wariantach pozwalających np. zdefiniować separator



```
(%i1) file_search("dane.dat");
(%o1)
false
(%i2) with_stdout("dane.dat", for i:1 thru 5 do print(i,i^2,i^3))$

(%i3) printfile("dane.dat");
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125

(%o3)
dane.dat
(%i4) L:read_list("dane.dat");
(%o4)
[1, 1, 1, 2, 4, 8, 3, 9, 27, 4, 16, 64, 5, 25, 125]
(%i5) NL:read_nested_list("dane.dat");
(%o5)
[[1, 1, 1], [2, 4, 8], [3, 9, 27], [4, 16, 64], [5, 25, 125]]
(%i6) M:read_matrix("dane.dat");

[ 1  1  1 ]
[      ]
[ 2  4  8 ]
[      ]
[ 3  9 27 ]
[      ]
[ 4 16 64 ]
[      ]
[ 5 25 125 ]
```



# Metoda najmniejszych kwadratów

Przez *fitowanie* rozumiemy dopasowywanie wzoru zawierającego niewiadome do zbioru punktów. Jest to zagadnienie numeryczne, niemniej jednak Maxima posiada odpowiednie narzędzia.

## Pakiet lsquares

Pakiet lsquares udostępni polecenie

`lsquares_estimates(macierz, [zmienne], rownanie, [parametry])`, które dla punktów zapisanych w *macierzy* (wiersz=punkt) metodą najmniejszych kwadratów szuka *równania* wiążącego *zmienne* i *parametry*.



# Regresja liniowa - prosty przykład

```
(%i1) L:makelist([i,%pi*i-%e],i,-10,10);
(%o1) [[- 10, - 10 %pi - %e], [- 9, - 9 %pi - %e], [- 8, - 8 %pi - %e],
[- 7, - 7 %pi - %e], [- 6, - 6 %pi - %e], [- 5, - 5 %pi - %e],
[- 4, - 4 %pi - %e], [- 3, - 3 %pi - %e], [- 2, - 2 %pi - %e],
[- 1, - %pi - %e], [0, - %e], [1, %pi - %e], [2, 2 %pi - %e], [3, 3 %pi - %e],
[4, 4 %pi - %e], [5, 5 %pi - %e], [6, 6 %pi - %e], [7, 7 %pi - %e],
[8, 8 %pi - %e], [9, 9 %pi - %e], [10, 10 %pi - %e]]
(%i2) M:float(apply(matrix,L))$
(%i3) load(lsquares);
(%o3) /usr/share/maxima/5.18.1/share/contrib/lsquares.mac
(%i4) lsquares_estimates(M,[x,y],y=a*x+b,[a,b]);
917750606692388452303686307283893961692709619434074271514259
(%o4) [[a = -----,
292129090655549561926648149573640677936898084137187201475850
1418618682683520483797785067016990821102284596004979274573090193
b = - -----]]
521880653299121413757540737900138880933961057363335739422020320
(%i5) float(%);
(%o5) [[a = 3.141592659029331, b = - 2.718281802008905]]
```



Jedną z miar jakości *fitu* jest średnie odchylenie kwadratowe zdefiniowane jako:

$$\frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$$

Wartość tą zwraca funkcja `lsquares_mse()`

```
(%i6) lsquares_mse(M,[x,y],y=a*x+b);
```

$$\frac{\sum_{i=1}^n (M_{i,2} - a M_{i,1} - b)^2}{21}$$

```
(%o6)
```

```
(%i7) ev(%,%o5[1],nouns);
```

```
(%o7) 1.784524095416561e-15
```



## Pakiet interpol

Pakiet `interpol.mac` udostępnia trzy funkcje służące do interpolacji wielomianowej. Pierwszym argumentem wszystkich z nich są punkty, które możemy zapisać na trzy sposoby:

- dwukolumnowa macierz `p:matrix([2,4],[5,6],[9,3])`
- lista par `p:[[2,4],[5,6],[9,3]]`
- lista liczb

Pakiet definiuje nową funkcję `charfun2(x, a, b)`:

$$\text{charfun2}(x, a, b) = \begin{cases} \text{true} & \text{dla } x \in [a, b) \\ \text{false} & \text{dla } x \notin [a, b) \end{cases}$$

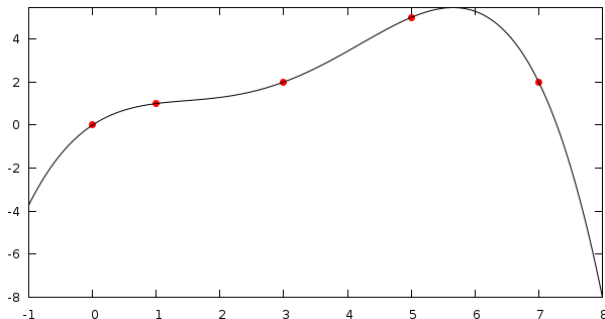


## lagrange(points)

```
(%i1) load(interpol);
(%o1) /usr/share/maxima/5.19.2/share/numeric/interpol.mac
(%i2) p: [[7,2],[1,1],[0,0],[5,5],[3,2]];
(%o2) [[7, 2], [1, 1], [0, 0], [5, 5], [3, 2]]
(%i3) L:lagrange(p);
```

$$\frac{(x-5)(x-3)(x-1)x}{168} - \frac{(x-7)(x-3)(x-1)x}{16} +$$

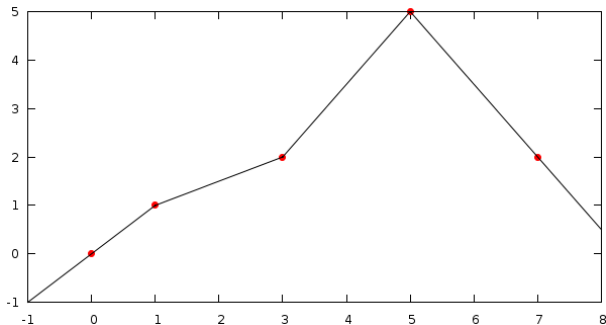
$$+ \frac{(x-7)(x-5)(x-1)x}{24} - \frac{(x-7)(x-5)(x-3)x}{48}$$





## linearinterpol(points)

```
(%i4) LI:lagrange(p);  
(%o4) x*charfun2(x,-inf,1)+(25/2-(3*x)/2)*charfun2(x,5,inf)  
      +((3*x)/2-5/2)*charfun2(x,3,5)+(x/2+1/2)*charfun2(x,1,3)
```



# Granice

Polecenie `limit(expr,x,val,dir)` znajduje granicę  $\lim_{x \rightarrow val} expr(x)$ . Opcjonalny argument `dir` określa czy szukamy granicy z prawej (plus) czy z lewej (minus) strony.

```
(%i1) limit(1/x,x,0);
(%o1)                                     infinity
(%i2) limit(1/x,x,0,plus);
(%o2)                                     inf
(%i3) limit(sin(x)/x,x,0);
(%o3)                                     1
(%i4) limit((1+1/x)^x,x,inf);
(%o4)                                     %e
(%i5) limit(sin(x),x,inf);
(%o5)                                     ind
(%i6) limit((sin(x+h)-sin(x))/h,h,0);
(%o6)                                     cos(x)
(%i7) limit((cos(x)+1)/sin(x)^2,x,%pi);
(%o7)                                     1
                                           -
                                           2
```



# Pochodne

Polecenia `diff()` używamy do znajdowania pochodnych. Występuję ono w kilku wariantach:

`diff(expr)`

Zwraca różniczkę zupełną, czyli sumę po pochodnych po wszystkich zmiennych

```
(%i1) diff(f(x)*f(y));
(%o1)      f(x) (--- (f(y))) del(y) + (--- (f(x))) f(y) del(x)
              dy              dx
```

gdzie  $\text{del}(x)$  oznacza  $\frac{\partial}{\partial x}$

`diff(expr,x,n)`

Wywołanie `diff(f(x),x,n)` odpowiada:

$$\frac{\partial^n}{\partial x^n} f(x)$$

Skrótowa forma `diff(expr,x)=diff(expr,x,1)`



# Pochodne

Polecenia `diff()` używamy do znajdowania pochodnych. Występują one w kilku wariantach:

`diff(expr, x1, n1, ..., xn, nm)`

Odpowiada zapisowi: `diff(...(diff(expr, xn, nm)...), x1, n1)` przy czym `expr` musi jawnie zależeć od wszystkich zmiennych `x1, x2, ..., xn`. Zależność taką tworzymy na dwa sposoby:

- deklarując `expr` jako funkcję `x1, x2, ..., xn`  $\Rightarrow$  `expr(x1, x2, ..., xn)`
- używając `depends(expr, [x1, x2, x3, ..., xn])`

`diff(expr(x, y, z), x, 1, y, 2, z, 3);`

$$\frac{d^6}{dx dy^2 dz^3} \text{expr}(x, y, z)$$



```
(%i1) diff(sin(x)*cos(x),x);
```

```
(%o1) cos(x)^2 - sin(x)^2
```

```
(%i2) f(x):=(2*x^2-5*x+2)/(3*x^2-10*x+3)$
```

```
(%i3) diff(f(x),x);
```

```
(%o3) (4 x - 5) / (3 x^2 - 10 x + 3) - (6 x - 10) (2 x^2 - 5 x + 2) / (3 x^2 - 10 x + 3)^2
```

```
(%i4) df(x):='';
```

```
(%o4) df(x) := (4 x - 5) / (3 x^2 - 10 x + 3) - (6 x - 10) (2 x^2 - 5 x + 2) / (3 x^2 - 10 x + 3)^2
```

```
(%i5) solve(df(x)=0,x);
```

```
(%o5) [x = - 1, x = 1]
```

```
(%i6) define(ddf(x),diff(f(x),x,2))$
```

$$\frac{4}{3x^2 - 10x + 3} - \frac{6(2x^2 - 5x + 2)}{(3x^2 - 10x + 3)^2} - \frac{2(4x - 5)(6x - 10)}{(3x^2 - 10x + 3)^2} + \frac{2(6x - 10)^2(2x^2 - 5x + 2)}{(3x^2 - 10x + 3)^3}$$

```
(%i7) ddf(1);
```

```
(%o7) - 5/8
```

```
(%i8) is(ddf(-1)>0);
```

```
(%o8) true
```

# Całkowanie symboliczne

Do obliczania zarówno całek oznaczonych jak i nieoznaczonych używamy polecenia `integrate()`

`integrate(expr,x)`

Oblicza całkę z nieoznaczoną `expr` względem zmiennej  $x$

$$\text{integrate}(f(x),x) = \int f(x)dx$$



# Całkowanie symboliczne

Do obliczania zarówno całek oznaczonych jak i nieoznaczonych używamy polecenia `integrate()`

`integrate(expr,x)`

Oblicza całkę z nieoznaczoną `expr` względem zmiennej `x`

$$\text{integrate}(f(x),x) = \int f(x)dx$$

`integrate(expr,x,a,b)`

Oblicza całkę oznaczoną:  $\int_a^b f(x)dx$  Wewnętrznie Maxima oblicza całkę nieoznaczoną i przykłada granicę:

$$\int_a^b f(x)dx = \lim_{x \rightarrow b} \int f(x)dx - \lim_{x \rightarrow a} \int f(x)dx$$



```
(%i1) integrate(sin(a*x),x);
```

```
(%o1)          cos(a x)
              - ----
                  a
```

```
(%i2) integrate(sin(x)/(b*cos(x)+a),x);
```

```
(%o2)          log(b cos(x) + a)
              - ----
                  b
```

```
(%i3) integrate(log(x)/x,x);
```

```
(%o3)          2
              log (x)
              -----
                  2
```

```
(%i4) integrate(exp(-a*x^2-b*x-c),x,minf,inf);
```

```
Is a positive, negative, or zero? p;
```

```
Is b positive, negative, or zero? p;
```

$$\frac{\sqrt{\pi} e^{-\frac{4ac-b^2}{4a}}}{\sqrt{a}}$$





Poprawność wyniku zawsze możemy sprawdzić licząc pochodną:

```
(%i1) integrate(x^2*exp(-x^2),x);
```

```
(%o1)      2
           - x
      sqrt(%pi) erf(x)  x %e
      ----- - -----
             4          2
```

```
(%i2) diff(%,x);
```

```
(%o2)      2      - x
           2      - x
      x %e
```

Wynik (%o1) zawiera funkcję specjalną  $\text{erf}(x)$ , będącą dystrybuantą rozkładu normalnego, zdefiniowaną jako:

$$\text{erf}(x) = \frac{2}{\pi} \int_0^x \exp(-t^2) dt$$



# Całkowanie numeryczne

Istnieje wiele funkcji umożliwiających całkowanie numeryczne:

- `romberg()` z pakietu o tej samej nazwie - bardzo prosty i wydajny algorytm. Wymaga aby całka była właściwa. Umożliwia numeryczne liczenie całek wielokrotnych
- funkcje `quad_qa*`
  - `quad_qag`
  - `quad_qagi`
  - `quad_qags`
  - `quad_qawc`
  - `quad_qawf`
  - `quad_qawo`
  - `quad_qaws`



# Całkowanie numeryczne

Istnieje wiele funkcji umożliwiających całkowanie numeryczne:

- `romberg()` z pakietu o tej samej nazwie - bardzo prosty i wydajny algorytm. Wymaga aby całka była właściwa. Umożliwia numeryczne liczenie całek wielokrotnych
- funkcje `quoad_qa*`
  - `quad_qag`
  - `quad_qagi`
  - `quad_qags`
  - `quad_qawc`
  - `quad_qawf`
  - `quad_qawo`
  - `quad_qaws`



`romberg(f, var, a, b)`

Numerycznie znajduje wartość  $\int_a^b f(x)dx$  używając metody Romberga<sup>1</sup>. Szukamy:

$$\int_0^{\pi} \exp(-2x^2) dx$$

```
(%i1) load(romberg);
(%o1) /usr/share/maxima/5.18.1/share/numeric/romberg.lisp
(%i2) f(x):=exp(-2*x^2);
(%o2) f(x) := exp((- 2) x ^ 2)
(%i3) romberg(f(x),x,0,%pi);
(%o3) .6266570689954041
```

Dla porównania to samo z użyciem `integrate()`

```
(%i4) (integrate(f(x),x,0,%pi),float(%));
(%o4) .6266570684498846
```

<sup>1</sup>[http://pl.wikipedia.org/wiki/Metoda\\_Romberga](http://pl.wikipedia.org/wiki/Metoda_Romberga)



Funkcje `romberg()` radzi sobie również z całkami wielokrotnymi:

$$g(x, y) = \int_0^3 dy \int_0^\pi \exp(-x^2 + y^2) \sin(xy)$$

```
(%i1) g(x,y):=exp(-x^2+y^2)*sin(x*y);  
                                     2      2  
(%o1)      g(x, y) := exp(- x  + y ) sin(x y)  
(%i2) romberg(romberg(g(x,y),x,0,%pi),y,0,3);  
(%o2)      655.5273722018377  
(%i3)
```

ZMIENNE ROMBERGTOL, ROMBERGIT, ...



# Pakiet QUADPACK

Funkcje quad\_qa\*:

- Przyjmują opcjonalne argumenty:
  - `epsrel` domyślnie:  $1e-8$   
Błąd względny  $< \text{epsrel}$
  - `epsabs` domyślnie: 0.0  
Błąd bezwzględny  $< \text{epsabs}$
  - `limit` domyślnie: 200  
Maksymalna ilość przedziałów na które dzielony jest obszar całkowania
- Użycie dowolnej z funkcji QUADPACK z nieznanym parametrem:

```
(%i1) quad_qags(a*x,x,0,1);  
(%o1) quad_qags(a x, x, 0, 1, epsrel = 1.e-8, epsabs = 0.0, limit = 200)
```

- Zwracają wynik w postaci listy `[A,B,C,D]` gdzie:
  - A wartość całki
  - B szacunkowy błąd bezwzględny
  - C liczba wywołań całkowanej funkcji
  - D kod błędu
    - 0 - wszystko OK
    - 3 - funkcja podcałkowa zachowuje się zbyt źle
    - 5 - funkcja podcałkowa jest rozbieżna lub zbiega zbyt wolno
    - 6 - błąd wejścia



# Pakiet QUADPACK

Funkcje quad\_qa\*:

- Przyjmują opcjonalne argumenty:
  - `epsrel` domyślnie:  $1e-8$   
Błąd względny  $< \text{epsrel}$
  - `epsabs` domyślnie: 0.0  
Błąd bezwzględny  $< \text{epsabs}$
  - `limit` domyślnie: 200  
Maksymalna ilość przedziałów na które dzielony jest obszar całkowania
- Użycie dowolnej z funkcji QUADPACK z nieznanym parametrem:

```
(%i1) quad_qags(a*x,x,0,1);  
(%o1) quad_qags(a x, x, 0, 1, epsrel = 1.e-8, epsabs = 0.0, limit = 200)
```

- Zwracają wynik w postaci listy `[A,B,C,D]` gdzie:
  - A wartość całki
  - B szacunkowy błąd bezwzględny
  - C liczba wywołań całkowanej funkcji
  - D kod błędu
    - 0 - wszystko OK
    - 3 - funkcja podcałkowa zachowuje się zbyt źle
    - 5 - funkcja podcałkowa jest rozbieżna lub zbiega zbyt wolno
    - 6 - błąd wejścia



# Pakiet QUADPACK

Funkcje quad\_qa\*:

- Przyjmują opcjonalne argumenty:
  - `epsrel` domyślnie:  $1e-8$   
Błąd względny  $< \text{epsrel}$
  - `epsabs` domyślnie: 0.0  
Błąd bezwzględny  $< \text{epsabs}$
  - `limit` domyślnie: 200  
Maksymalna ilość przedziałów na które dzielony jest obszar całkowania
- Użycie dowolnej z funkcji QUADPACK z nieznanym parametrem:

```
(%i1) quad_qags(a*x,x,0,1);  
(%o1) quad_qags(a x, x, 0, 1, epsrel = 1.e-8, epsabs = 0.0, limit = 200)
```

- Zwracają wynik w postaci listy `[A,B,C,D]` gdzie:
  - A wartość całki
  - B szacunkowy błąd bezwzględny
  - C liczba wywołań całkowanej funkcji
  - D kod błędu
    - 0 - wszystko OK
    - 3 - funkcja podcałkowa zachowuje się zbyt źle
    - 5 - funkcja podcałkowa jest rozbieżna lub zbiega zbyt wolno
    - 6 - błąd wejścia





`quad_qag(expr, var, a, b, key)`

Oblicza wartość całki  $\int_a^b f dx$  na skończonym przedziale  $[a, b]$ . Klucz `key`  $[0, 6]$  decyduje o użytym algorytmie, wysokie wartości warto stosować dla silnie oscylujących funkcji:

```
(%i1) quad_qag(sqrt(x)*log(1/x),x,0,1,0);
(%o1) [1.4444444444463946, 3.5460707082765625e-9, 555, 0]
(%i2) quad_qag(sqrt(x)*log(1/x),x,0,1,0,'epsrel=1e-2);
(%o2) [1.4444452203177388, .001852967083306165, 165, 0]
```

`quad_qagi(expr, var, a, b)`

Oblicza wartość całki oznaczonej, której przynajmniej jedna granica jest niewłaściwa

$$\int_{-\infty}^b f dx, \quad \int_a^{\infty} f dx, \quad \int_{-\infty}^{\infty} f dx$$

```
(%i1) quad_qagi(exp(-2*x^2),x,minf,inf);
(%o1) [1.253314137315502, 4.4674503117501196e-9, 270, 0]
(%i3) quad_qagi(exp(-x)*log(x),x,0,inf);
(%o2) [-.5772156649015293, 5.110526668516968e-9, 345, 0]
(%i3) float(%gamma);
(%o3) .5772156649015329
```



`quad_qag(expr, var, a, b, key)`

Oblicza wartość całki  $\int_a^b f dx$  na skończonym przedziale  $[a, b]$ . Klucz `key`  $[0, 6]$  decyduje o użytym algorytmie, wysokie wartości warto stosować dla silnie oscylujących funkcji:

```
(%i1) quad_qag(sqrt(x)*log(1/x),x,0,1,0);
(%o1)      [.4444444444463946, 3.5460707082765625e-9, 555, 0]
(%i2) quad_qag(sqrt(x)*log(1/x),x,0,1,0,'epsrel=1e-2);
(%o2)      [.4444452203177388, .001852967083306165, 165, 0]
```

`quad_qagi(expr, var, a, b)`

Oblicza wartość całki oznaczonej, której przynajmniej jedna granica jest niewłaściwa

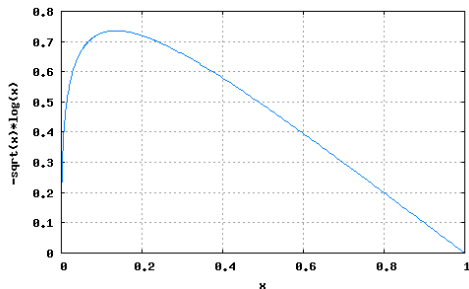
$$\int_{-\infty}^b f dx, \quad \int_a^{\infty} f dx, \quad \int_{-\infty}^{\infty} f dx$$

```
(%i1) quad_qagi(exp(-2*x^2),x,minf,inf);
(%o1)      [1.253314137315502, 4.4674503117501196e-9, 270, 0]
(%i3) quad_qagi(exp(-x)*log(x),x,0,inf);
(%o2)      [-.5772156649015293, 5.110526668516968e-9, 345, 0]
(%i3) float(%gamma);
(%o3)      .5772156649015329
```



`quad_qags(expr, var, a, b)`

Oblicza wartość całki na skończonym przedziale. Przystosowana do radzenia sobie z funkcjami mającymi osobliwości w przedziale całkowania.



$$f(x) = \sqrt{x} \ln\left(\frac{1}{x}\right)$$

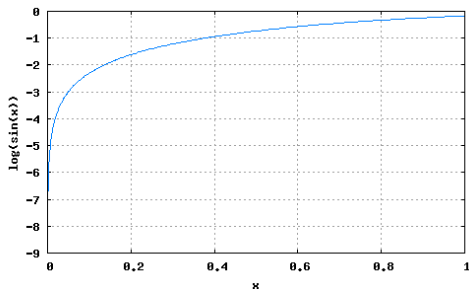
$$\int_0^1 f(x) dx = \frac{4}{9}$$

```
(%i2) quad_qag(f,x,0,1,0);
(%o2)  [.44444444444463946, 3.5460707082765625e-9, 555, 0]
(%i3) quad_qags(f,x,0,1);
(%o3)  [.44444444444444448, 1.110223024625157e-15, 315, 0]
```



`quad_qags(expr, var, a, b)`

Oblicza wartość całki na skończonym przedziale. Przystosowana do radzenia sobie z funkcjami mającymi osobliwości w przedziale całkowania.



$$g(x) = \ln(\sin x)$$

```
(%i5) quad_qags(g,x,0,%pi);  
(%o5)      [- 2.177586090303605, 2.797762022055394e-14, 399, 0]  
(%i7) quad_qag(g,x,0,%pi,0);  
(%o7)      [- 2.177586090263982, 1.7192795732040494e-8, 1665, 0]
```



`quad_qawo(expr, var, a, b, omega, trig)`

Funckja służy do obliczania wyrażeń typu:

$$\int_a^b f(x) \cos(\omega x) dx, \quad \int_a^b f(x) \sin(\omega x) dx$$

Obliczmy:

$$\int_{-1}^1 (x^2 + x) \cos(2x) \quad \int_1^{\pi} x \ln x \cos(2x)$$

```
(%i1) f:x^2+x$
(%i2) g:x*log(x)$
(%i3) float(integrate(f*cos(2*x),x,-1,1));
(%o3) .03850187686569839
(%i4) quad_qawo(f,x,-1,1,2,cos);
(%o4) [.03850187686569852, 2.595344524868247e-10, 15, 0]
(%i5) integrate(g*sin(2*x),x,1,%pi);
      %pi
      /
      [
(%o5) I      x log(x) sin(2 x) dx
      ]
      /
      1
(%i6) quad_qawo(f,x,1,%pi,2,sin);
(%o6) [- 7.249681724869196, 5.138589327224146e-16, 25, 0]
```

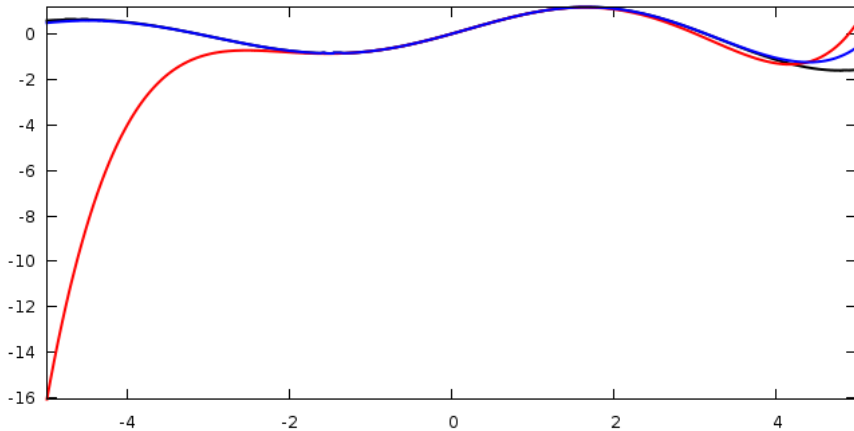
# Szeregi Taylora

`taylor(expr, var, pkt, n)`

Rozwija wyrażenie `expr` względem zmiennej `var` wokół punktu `pkt` w szereg Taylora stopnia `n`

```
(%i1) f(x):=sin(x)*exp(0.1*x);
(%o1) f(x) := sin(x) exp(0.1 x)
(%i2) T(n):=taylor(f(x),x,0,n);
(%o2) T(n) := taylor(f(x), x, 0, n)
(%i3) T(5);
              2      3      4      5
            x  97 x  33 x  1801 x
(%o3)/T/      + -- - ---- - ---- + ---- + . . .
            10   600  2000  240000
(%i4) load(draw)$
(%i8) draw2d(line_width=2,explicit(f(x),x,-5,5),
  color=red,explicit(T(5),x,-5,5),
  color=blue,explicit(T(10),x,-5,5),terminal=wxt)$
```





# Równania różniczkowe

## ode2(eqn, zvar, nvar)

Funkcja rozwiązuje zwyczajne równania różniczkowe (*Ordinary Differential Equation*) pierwszego i drugiego rzędu. Stałe całkowania reprezentowane są przez stałe %c dla równań pierwszego rzędu i stałe %k1 %k2 dla drugiego. zvar to zmienna zależna a nvar niezależna.

$$\frac{d}{dx}y = 3x^2 - 2x^3 + \lambda$$

```
(%i1) rr:'diff(y,x)=3*x^2-2*x^3+lambda;
                                dy          3      2
(%o1)      -- = lambda - 2 x  + 3 x
                                dx

(%i2) ode2(rr,y,x);

                                4
                                x      3
(%o2)      y = x lambda - -- + x  + %c
                                2

(%i3) ev(rr,%,nouns);

                                3      2          3      2
(%o3)      lambda - 2 x  + 3 x  = lambda - 2 x  + 3 x

(%i4) %-rhs(%);
(%o4)      0 = 0
```





Użycie apostrofu w %i1 było **konieczne**. W przeciwnym wypadku:

```
(%i5) rr:diff(y,x)=3*x^2-2*x^3+lambda;
```

```
(%o5)
              3      2
0 = lambda - 2 x  + 3 x
```

Inne sposoby:

- `diff(y(x),x)=...`

- `(%i6) depends(y,x);`

```
(%o6) [y(x)]
```

```
(%i7) rr:diff(y,x)=3*x^2-2*x^3+lambda;
```

```
(%o7)
      dy      3      2
-- = lambda - 2 x  + 3 x
dx
```

`ode2()` zwraca rozwiązanie ogólne. Aby otrzymać rozwiązanie szczególne nakładamy warunek początkowy:

`ic1(rozw_ogol,x=xval,y=yval)`

Nakłada na rozwiązanie ogółne równania pierwszego rzędu warunek, dla zmiennej niezależnej `x=xval` wartość zmiennej zależnej wynosi `y=yval`.



Przykłady rozwiązań równań różniczkowych pierwszego rzędu:

$$\frac{dy}{dx} \sin x + y \cos x = \sin 2x, \quad y\left(x = \frac{\pi}{2}\right) = 0$$

```
(%i1) rr:'diff(y,x)*sin(x)+y*cos(x)=sin(2*x)$
```

```
(%i2) ro:ode2(rr,y,x);
```

$$\%c - \frac{\cos(2x)}{2}$$

```
(%o2) y = -----
          sin(x)
```

```
(%i3) rs:ic1(ro,x=%pi/2,y=0);
```

```
(%o3) y = - \frac{\cos(2x) + 1}{2 \sin(x)}
```

```
(%i4) ev(rs,x=%pi/2);
```

```
(%o4) y = 0
```

```
(%i5) ev(rr,ro,nouns);
```

$$\begin{aligned} \sin(x) \left( \frac{\sin(2x)}{\sin(x)} - \frac{\cos(x) \left( \%c - \frac{\cos(2x)}{2} \right)}{\sin(x)} \right) + \frac{\cos(x) \left( \%c - \frac{\cos(2x)}{2} \right)}{\sin(x)} = \end{aligned}$$

$$\sin(2x)$$

```
(%i6) trigsimp(%);
```

```
(%o6) sin(2 x) = sin(2 x)
```

## Równanie Bernoulliego:

$$\frac{dy}{dx} + \frac{y}{x} = ay^2 \ln x, \quad y(x=1) = 1$$

```
(%i1) bern:'diff(y,x)+y/x=a*y^2*log(x)$
```

```
(%i2) ro:ode2(bern,y,x);
```

```
(%o2)
```

$$y = \frac{1}{x \left( \frac{a \log(x)}{2} - \frac{1}{2} \right)}$$

```
(%i3) rs:ic1(ro,x=1,y=1);
```

```
(%o3)
```

$$y = - \frac{2}{a x \log(x) - 2 x}$$

```
(%i4) ev(bern,ro,nouns);
```

```
(%o4)
```

$$\frac{a \log(x)}{x \left( \frac{a \log(x)}{2} - \frac{1}{2} \right)} = \frac{a \log(x)}{x \left( \frac{a \log(x)}{2} - \frac{1}{2} \right)}$$

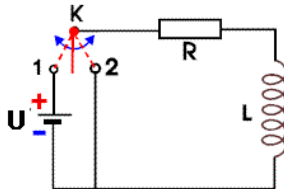
```
(%i5) ev(rs,x=1);
```

```
(%o5)
```

$$y = 1$$

Wyznacz przebieg zmian natężenia prądu  $i = i(t)$  po włączeniu obwodu do źródła.  
 $i(t = 0) = 0$

$$L \frac{di}{dt} + Ri = U$$



```
(%i1) kirch: L*'diff(i,t)+R*i=U$
```

```
(%i2) ro:ode2(kirch,i,t);
```

$$i = e^{-\frac{tR}{L}} \left( \frac{e^{\frac{tR}{L}} U}{R} + \%c \right)$$

```
(%i3) rs:ic1(ro,t=0,i=0);
```

$$i = \frac{e^{-\frac{tR}{L}} \left( e^{\frac{tR}{L}} - 1 \right) U}{R}$$

```
(%i4) assume(R>0,L>0,U>0);
```

```
(%o4) [R > 0, L > 0, U > 0]
```

```
(%i5) limit(rs,t,inf);
```

```
(%o5) U  
R
```

## Równania drugiego rzędu

Polecenie `ode2()` potrafi rozwiązywać wybrane typu równań różniczkowych drugiego rzędu.

`ic2(rozw_ogol, x=xval, y=yval, diff(y, x)=pval)`

Nakłada na rozwiązanie ogólne warunki początkowe

$$y(x_{val}) = y_{val} , \quad \left. \frac{dy}{dx} \right|_{x=x_{val}} = p_{val}$$

`bc2(rozw_ogol, x=x1, y=y1, x=x2, y=y2)`

Nakłada na rozwiązanie ogólne warunki brzegowe

$$y(x_1) = y_1 , \quad y(x_2) = y_2$$



$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + 2y = 2\sin x, \quad y(x=0) = 0, \quad \frac{dy}{dx}(x=0) = 0$$

```
(%i1) rr:'diff(y,x,2)+2*'diff(y,x)+2*y=2*sin(x)$
(%i2) ro:ode2(rr,y,x);
          - x          2 sin(x) - 4 cos(x)
(%o2)      y = %e      (%k1 sin(x) + %k2 cos(x)) + -----
                                                    5
(%i3) rs:ic2(%,x=0,y=0,diff(y,x)=0);
          2 sin(x) - 4 cos(x)      - x  2 sin(x)  4 cos(x)
(%o3)      y = ----- + %e  (----- + -----)
          5                      5          5
(%i4) (diff(rs,x),ev(%,x=0));
(%o4)                                0 = 0
```



## contrib\_ode

### contrib\_ode(eqn,zvar,nvar)

Rozwiązuje liniowe równania różniczkowe zwyczajne oraz wybrane typy równań nieliniowych:

$$x \left( \frac{dy}{dx} \right)^2 - (1 + xy) \frac{dy}{dx} + y = 0 \quad \frac{dy}{dx} = (x + y)^2$$

```
(%i1) load(contrib_ode);
(%o1) /usr/share/maxima/5.19.2/share/contrib/diffequations/contrib_ode.mac
(%i2) rr:x*'diff(y,x)^2-(1+x*y)*'diff(y,x)+y=0$
(%i3) ode2(rr,y,x);

          dy 2          dy
(%t3)      x (--)  - (x y + 1) -- + y = 0
          dx          dx

first order equation not linear in y'

(%o3)                                     false
(%i4) contrib_ode(rr,y,x);
          dy 2          dy
(%t4)      x (--)  - (x y + 1) -- + y = 0
          dx          dx

                                x
(%o4)      [y = log(x) + %c, y = %c %e ]
```

# contrib\_ode

## contrib\_ode(eqn,zvar,nvar)

Rozwiązuje liniowe równania różniczkowe zwyczajne oraz wybrane typy równań nieliniowych:

$$x \left( \frac{dy}{dx} \right)^2 - (1 + xy) \frac{dy}{dx} + y = 0 \quad \frac{dy}{dx} = (x + y)^2$$

```
(%i5) rr2:'diff(y,x)=(x+y)^2;
```

```
(%o5)
      dy
      -- = (y + x)
      dx
```

```
(%i6) contrib_ode(rr2,y,x);
```

```
(%o6) [[x = %c - atan(sqrt(%t)), y = - x - sqrt(%t)],
      [x = atan(sqrt(%t)) + %c, y = sqrt(%t) - x]]
```





## Rozwijanie w szereg Fouriera z definicji

Maximy, możemy używać również przy rozwijaniu danej funkcji  $f(x)$  w szereg Fouriera na odcinku  $(-l, l)$ . Pierwszy sposób polega na znalezieniu współczynników  $a_n$  i  $b_n$ , ręcznie bezpośrednio z definicji.

Korzystając z wzorów, dla ułatwienia na odcinku  $(-\pi, \pi)$ :

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx)$$

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx \quad a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx \quad b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx$$

Z oczywistych względów powyższa suma biegnie od  $n=1$  to jakiejś rozsądnej granicy, która wyznacza możliwości naszego komputera.



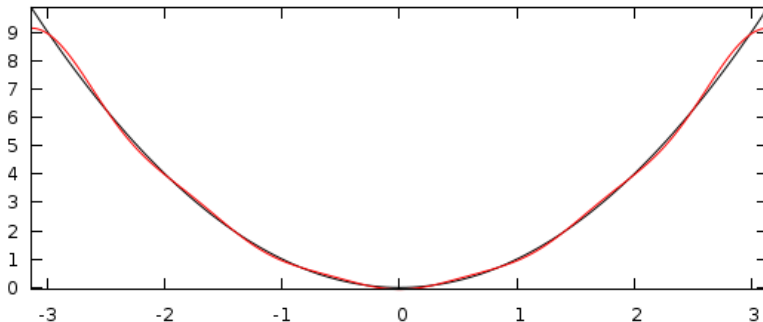
Zacznijmy od najprostszego przykłądu  $f(x) = x^2$

```
(%i1) declare(n, integer);
(%o1) done
(%i2) f(x):=x^2$
(%i3) a0:1/%pi*integrate(f(x),x,-%pi,%pi);
2
2 %pi
-----
3
(%o3) 1/%pi*integrate(f(x)*cos(n*x),x,-%pi,%pi);
n
4 (- 1)
-----
2
n
(%o4) a(n):='';
n
4 (- 1)
-----
2
n
(%o5) a(n) := -----
2
n
(%i6) 1/%pi*integrate(f(x)*sin(x),x,-%pi,%pi);
0
(%o6)
(%i7) fs(nmax):=sum(a(n)*cos(n*x),n,1,nmax)+a0/2$
```



Rysujemy wykresy funkcji  $f(x)$  oraz jej rozwinięcia z dokładnością do np. 5 składników sumy

```
(%i9) load(draw)
(%o9) /usr/share/maxima/5.18.1/share/draw/draw.lisp
(%i10) draw2d(line_width = 2, explicit(f(x), x, - %pi, %pi), color = red,
            explicit(fs(5), x, - %pi, %pi), terminal = wxt)
(%o10) [gr2d(explicit, explicit)]
```



# Pakiet fourie.mac

Pakiet fourie.mac umożliwia zautomatyzowanie tego procesu.

```
(%i2) load(fourie)
(%o2) /usr/share/maxima/5.18.1/share/calculus/fourie.mac
(%i3) f(x):=x^2$
(%i4) flist : fourier(f(x), x, %pi)

(%t4)

$$a = \frac{\frac{\pi^2}{0} - \frac{\pi^2}{3}}{3}$$



$$2 \left( \frac{\frac{\pi^2 \sin(\pi n)}{n} - \frac{2 \sin(\pi n)}{3} + \frac{2 \pi \cos(\pi n)}{n} \right)$$


(%t5)

$$a = \frac{\frac{\pi^2}{n} - \frac{2 \sin(\pi n)}{3} + \frac{2 \pi \cos(\pi n)}{n}}{\pi}$$


(%t6)

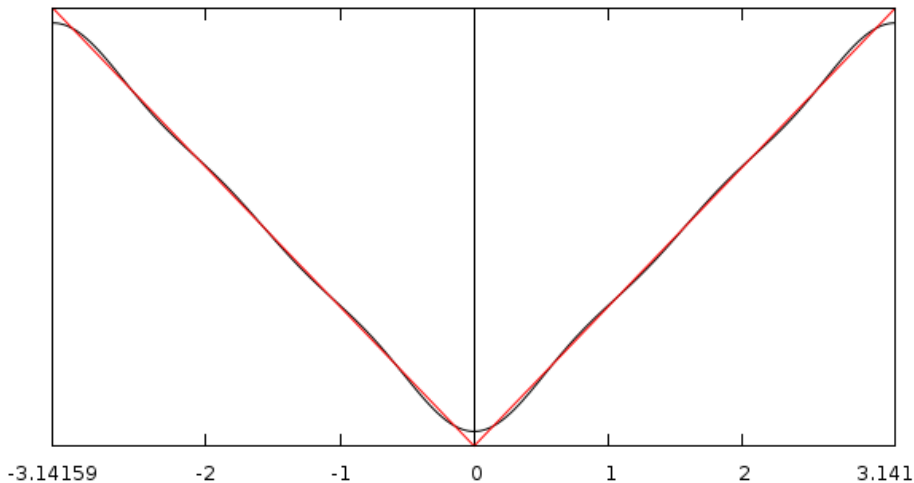
$$b = 0$$


(%o6)
[%t4, %t5, %t6]
(%i7) fs(nmax) := fourexpand(flist, x, %pi, nmax)$
```

Używamy ten sam wynik, znacznie szybciej.



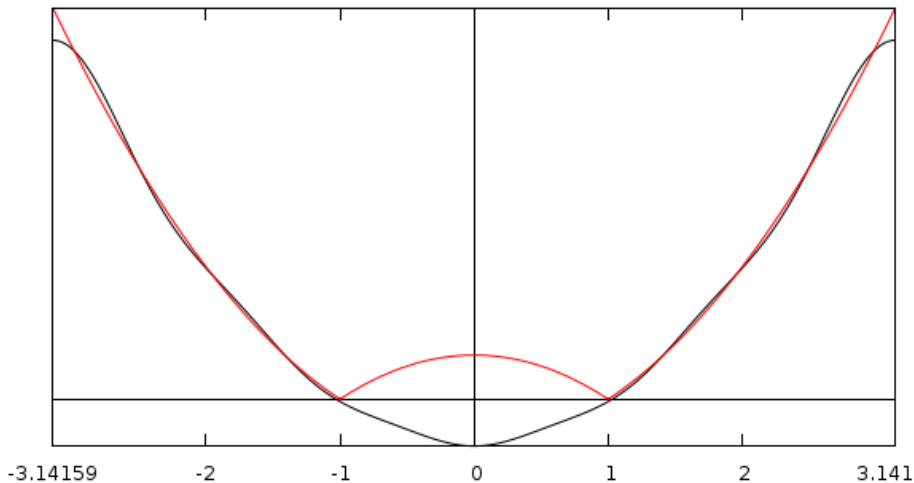
Pakiet fourie powinien sobie radzić z wyrażeniami zawierającymi  $| \cdot |$ :



$$f(x) = |x|$$



Pakiet fourie powinien sobie radzić z wyrażeniami zawierającymi  $| \cdot |$ :



$$f(x) = |x^2 - 1|$$



Próba rozwinięcia funkcji  $\sin(x)$  (!) kończy się błędem:

```
(%i2) f(x):=sin(x)$
(%i3) flist:fourier(f(x),x,%pi);
(%t3)
a = 0
0

(%t4)
a = 0
n

sin(%pi n) sin(%pi n)
2 (----- - -----)
2 n + 2 2 n - 2

(%t5) b = -----
n %pi

[%t3, %t4, %t5]
(%i6) fs(nmax):=fourexpend(flist,x,%pi,nmax);
(%o6) fs(nmax) := fourexpend(flist, x, %pi, nmax)
(%i7) ev(fs(10),x=0);
(%o7) 0
(%i8) ev(fs(10),x=1);

Division by 0
#0: fourexpend(1=[%t3,%t4,%t5],x=1,p=%pi,nn=10)
#1: fs(nmax=10)
-- an error. To debug this try debugmode(true);
```



W wypadku bardziej skomplikowanych funkcji, zmieniających się w wielu przedziałach należy użyć pakietu pw.mac. Źródła, przykłady oraz dokumentacje można znaleźć na stronie <http://mysite.verizon.net/res11w2yb/id2.html>.





# Podstawy

Dwa sposoby programowania:

- Użycie wewnętrznego języka programowania. Zwyczajowo kod zapisywany jest w plikach **.mac** lub **.mc**
- Użycie Lispa, pliki **.lisp**

Podstawowe zasady:

- Instrukcje proste kończymy " ," za wyjątkiem ostatniej
- Nawiasy ( ) zamykają kilka instrukcji w blok { } w C
- Maxima cały kod zwiija do jednej linii  $\Rightarrow$  czasami wymagana jest spacja, sam znak nowej linii nie wystarczy
- Komentarzem jest wszystko pomiędzy znakami /\* i \*/
- Pliki wsadowe (skrypty) wczytujemy poleceniem batch



## Wyprowadzanie tekstu na ekran

Trzy podstawowe funkcje: disp, display, print:

```
(%i1) r1:x+y=1$
(%i2) r2:x+y-1$
(%i3) tekst:" ma kota"$
(%i4) display(r1,r2,"Ala",tekst);

          y + x = 1

          r2 = y + x - 1

          Ala = Ala

          tekst =  ma kota

(%o4)
          done
(%i5) disp(r1,r2,"Ala",tekst);

          y + x = 1

          y + x - 1

          Ala

          ma kota

(%o5)
          done
(%i6) print(r1,r2,"Ala",tekst);
y + x = 1 y + x - 1 Ala  ma kota
(%o6)
          ma kota
```

## Prosty przykład

$$\begin{aligned}0! &= 1 \\ n! &= n(n-1)!\end{aligned}$$

```
(%i1) silnia(n):=if n = 0 then 1 else n*silnia(n-1);  
(%o1)      silnia(n) := if n = 0 then 1 else n silnia(n - 1)  
(%i2) silnia(7);  
(%o2)      5040  
(%i3) 7!;  
(%o3)      5040
```



`block([lista],expr1,expr2,...,exprn)`

Wykonuje wyrażenia `expr1`-`exprn`. Zwraca wynik ostatniego wyrażenia. Kolejność wykonywania instrukcji można zmieniać za pomocą `go`, `throw`, `return`.

`[lista]` zawiera listę lokalnych zmiennych niedostępnych poza instrukcją `block`.

```
(%i1) x1:3;
(%o1)
3
(%i2) f(x):=block([x1,temp],
    x1:4,
    temp:2*x,
    x)$
(%i3) f(4);
(%o3)
4
(%i4) x1;
(%o4)
3
(%i5) temp;
(%o5)
temp
```



# Instrukcja warunkowa

## Instrukcja warunkowa

```
if warunek_1  
  then expr1  
  else expr2
```

```
if warunek_1 then expr1  
  elseif warunek_2 then expr2  
  elseif warunek_3 then expr3  
  ....  
  else expr0
```

Wszystkie wyrażenia `warunek_n` muszą mieć określoną wartość *true/false*. Wartość jest nieokreślona (np. `a1a>kot`) zachowanie Maximy zależy do zmiennej `prederror`

- *true* wyświetlany jest błąd
- *false* brak błędu

!!!!OPERATORY!!!!



```

(%i1) wieksze(x,y):=if x>y then print(x," wieksze niz ",y)
      else print(x," nie wieksze niz",y)$
(%i2) wieksze(1,2);
1   nie wieksze niz 2
(%o2)
2
(%i3) wieksze(4.5,3);
4.5   wieksze niz  3
(%o3)
3
(%i4) wieksze(exp(3),sin(1));
3
%e   wieksze niz  sin(1)
(%o4)
sin(1)
(%i5) wieksze(y^2,y);
      2      2
(%o5) if y  > y then print(y ,  wieksze niz , y)
      2
      else print(y ,  nie wieksze niz, y)
(%i6) assume(y>1);
(%o6)
[y > 1]
(%i7) wieksze(y^2,y);
2
y   wieksze niz  y
(%o7)
y

```



# Pętla for

## Klasyczna pętla [od,do]:

```
(%i1) for i:1 thru 3 do
    print(i, " ", i^2);
1      1
2      4
3      9
(%o1)      done
```

## Zmiana warunku

```
(%i3) for i:-1 unless i>3 do
    print(i, " ", i^2);
- 1      1
0      0
1      1
2      4
3      9
(%o3)      done
```

zamiast unless można użyć while i zanegować warunek

## Zmiana kroku:

```
(%i2) for i:3 step -0.5 thru 1 do
    print(i, " ", i^2);
3      9
2.5    6.25
2.0    4.0
1.5    2.25
1.0    1.0
(%o2)      done
```

## Iterowanie po elementach listy

```
(%i4) L: [1,2,3]$
(%i5) for i in L do
(
    print(i),
    print(2^i)
);
1
2
2
4
3
8
```



# Pętla for

## Klasyczna pętla [od,do]:

```
(%i1) for i:1 thru 3 do
    print(i, " ", i^2);
1      1
2      4
3      9
(%o1)      done
```

## Zmiana warunku

```
(%i3) for i:-1 unless i>3 do
    print(i, " ", i^2);
- 1      1
0      0
1      1
2      4
3      9
(%o3)      done
```

zamiast unless można użyć while i zanegować warunek

## Zmiana kroku:

```
(%i2) for i:3 step -0.5 thru 1 do
    print(i, " ", i^2);
3      9
2.5    6.25
2.0    4.0
1.5    2.25
1.0    1.0
(%o2)      done
```

## Iterowanie po elementach listy

```
(%i4) L: [1,2,3]$
(%i5) for i in L do
(
    print(i),
    print(2^i)
);
1
2
2
4
3
8
```





# Pętla for

## Klasyczna pętla [od,do]:

```
(%i1) for i:1 thru 3 do
    print(i, " ", i^2);
1      1
2      4
3      9
(%o1)      done
```

## Zmiana warunku

```
(%i3) for i:-1 unless i>3 do
    print(i, " ", i^2);
- 1      1
0      0
1      1
2      4
3      9
(%o3)      done
```

zamiast unless można użyć while i zanegować warunek

## Zmiana kroku:

```
(%i2) for i:3 step -0.5 thru 1 do
    print(i, " ", i^2);
3      9
2.5    6.25
2.0    4.0
1.5    2.25
1.0    1.0
(%o2)      done
```

## Iterowanie po elementach listy

```
(%i4) L: [1,2,3]$
(%i5) for i in L do
(
    print(i),
    print(2^i)
);
1
2
2
4
3
8
```



# Pętla for

## Klasyczna pętla [od,do]:

```
(%i1) for i:1 thru 3 do
    print(i, " ", i^2);
1      1
2      4
3      9
(%o1)      done
```

## Zmiana warunku

```
(%i3) for i:-1 unless i>3 do
    print(i, " ", i^2);
- 1      1
0      0
1      1
2      4
3      9
(%o3)      done
```

zamiast unless można użyć while i zanegować warunek

## Zmiana kroku:

```
(%i2) for i:3 step -0.5 thru 1 do
    print(i, " ", i^2);
3      9
2.5    6.25
2.0    4.0
1.5    2.25
1.0    1.0
(%o2)      done
```

## Iterowanie po elementach listy

```
(%i4) L: [1,2,3]$
(%i5) for i in L do
(
    print(i),
    print(2^i)
);
1
2
2
4
3
8
```



## Prosty przykład - silnia

```
(%i1) s(n):=block([temp],
    temp:1,
    for i:1 thru n do
        temp:temp*i,
    return(temp)
)$
(%i2) s(5);
(%o2) 120
```

## Prosty przykład - dzielniki

/Przykład wybitnie nieoptymalny/

```
(%i8) s(n):=block([l:[]],
    for i:1 thru n do(
        if mod(n,i)=0 then
            l:endcons(i,l)
    ),
    return(l)
)$
(%i9) s(120);
(%o9) [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
```



## Prosty przykład - silnia

```
(%i1) s(n):=block([temp],
    temp:1,
    for i:1 thru n do
        temp:temp*i,
    return(temp)
)$
(%i2) s(5);
(%o2) 120
```

## Prosty przykład - dzielniki

/Przykład wybitnie nieoptymalny/

```
(%i8) s(n):=block([l:[]],
    for i:1 thru n do(
        if mod(n,i)=0 then
            l:endcons(i,l)
    ),
    return(l)
)$
(%i9) s(120);
(%o9) [1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120]
```



Kolejny przykład: rozwijanie w szereg Taylora wokół 0:

```
(%i1) Taylor0(f,x,n):=block([i],
  s:1,
  wyraz:f,
  for i:1 unless i>n do(
    wyraz:diff(wyraz,x)/i,
    s:ev(wyraz,x=0)*x^i+s
  ),
  return(s)
)$
```

```
(%i2) Taylor0(sin(x)*exp(cos(x)),x,8);
```

```
(%o2)
          7          5          3
      379 %e x    31 %e x    2 %e x
- ---- + ---- - ---- + %e x + 1
  5040    120    3
```



## Przykład - wielomiany Legendre'a

Wzór rekurencyjny:  $P_0(x) = 1$      $P_1(x) = x$

$$P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x)$$

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$P_2(x) = \frac{1}{2} (3x^2 - 1)$$

$$P_3(x) = \frac{1}{2} (5x^3 - 3x)$$

$$P_4(x) = \frac{1}{8} (35x^4 - 30x^2 + 3)$$

```
(%i1) legendre1(x,n):=block(
  if n = 0
    then 1
  else
    if n = 1
      then x
    else ratsimp((2*n-1)/n*x*legendre1(x,n-1)-(n-1)/n*legendre1(x,n-2))
)$
(%i2) legange1(w,4)
```

$$\frac{35w^4 - 30w^2 + 3}{8}$$



## Wielomiany Legendre'a - cd

Algorytm iteracyjny:

```
(%i2) legendre2(x,n):=block([po,ppo,pb],
  if n = 0 then
    return(1),
  if n = 1 then
    return(x),
  ppo:1,
  po:x,
  for k:2 thru n do(
    pb:(2*k-1)/k*x*po-(k-1)/k*ppo,
    ppo:po,
    po:pb
  ),
  return(ratsimp(pb))
)$
(%o3) legendre2(x,4);
```

$$\frac{35x^4 - 30x^2 + 3}{8}$$



## Wielomiany Legendre'a - cd

Algorytm iteracyjny:

```
(%i4) legendre3(x,n):=block([Pn:[1,x]],
  if n = 0 then
    return([1]),
  for k:2 thru n do(
    pb:(2*k-1)/k*x*Pn[k]-(k-1)/k*Pn[k-1],
    Pn:endcons(pb,Pn)
  ),
  return(ratsimp(Pn))
)$
(%i5) legendre3(x,5);
```

$$\left[1, x, \frac{3x^2 - 1}{2}, \frac{5x^3 - 3x}{2}, \frac{35x^4 - 30x^2 + 3}{8}, \frac{63x^5 - 70x^3 + 15x}{8}\right]$$





## Wielomiany Legendre'a - cd

Wzór bezpośredni:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2 - 1)^n]$$

```
(%i6) legengre4(x,n):=ratsimp(1/(2^n*n!)*diff((x^2-1)^n,x,n))$
(%i7) legengre4(x,11);
```

$$\frac{88179 x^{11} - 230945 x^9 + 218790 x^7 - 90090 x^5 + 15015 x^3 - 693 x}{256}$$

## legendre\_p(n,x)

Zwraca wielomian Legendre'a pierwszego rodzaju stopnia  $n$  względem zmiennej  $x$

```
(%i8) legendre_p(5,x);
```

$$-\frac{63(1-x)^5}{8} + \frac{315(1-x)^4}{8} - 70(1-x)^3 + \frac{105(1-x)^2}{2} - 15(1-x) + 1$$



## Wielomiany Legendre'a - porównanie

Wyznaczanie  $P_{30}(x)$ 

Metoda	z upraszczeniem	bez upraszczania
legendre1	5min8s (5433MB)	3m8s (2758MB)
legendre2	17s (740MB)	0,005s (75kB)
legendre3	45s (1939MB)	0,005 (84kB)
legendre4	0,026s (580kB)	0,025s (535kB)
legendre_p	0,003s (203kB)	0,002s (54kB)
legendre_p(1000, x)	10,25s (1177MB)	1,3s (17MB)



## Wielomiany Legendre'a - porównanie

Wyznaczanie  $P_{30}(x)$

Metoda	z upraszczeniem	bez upraszczania
legendre1	5min8s (5433MB)	3m8s (2758MB)
legendre2	17s (740MB)	0,005s (75kB)
legendre3	45s (1939MB)	0,005 (84kB)
legendre4	0,026s (580kB)	0,025s (535kB)
legendre_p	0,003s (203kB)	0,002s (54kB)
legendre_p(1000, x)	10,25s (1177MB)	1,3s (17MB)



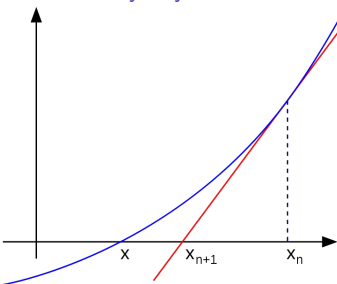
## Przykład bisekcja

```
(%i1) p(f,x,a,b,iteracje):=block([xp],
  if ev(f,x=a)*ev(f,x=b) > 0 then
  (
    print("Takie same znaki!"),return()
  ),
  xp:(a+b)/2,
  for i:1 thru iteracje do(
    if ev(f,x=xp)=0 then
      return(xp),
    if ev(f,x=a)*ev(f,x=xp)<0 then
      xp:(a+xp)/2
    else
      xp:(b+xp)/2
  ),
  return(xp)
)$

(%i2) float(p(sin(x)*cos(x)-1/3,x,-1,1,20));
(%o2) .3548383712768555
```

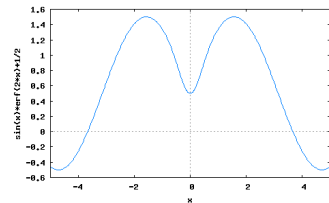


## Metoda stycznych



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
niuton(expr,var,x0,eps):=block([xn:x0,D],
D:diff(expr,var),
while abs(subst(xn,var,expr))>eps do
  xn:xn-subst(xn,var,expr)/subst(xn,var,D),
return(xn)
);
```



```
(%i2) niuton(sin(x)*erf(2*x)+1/2,x,4,1/19);
```

$$4 - \frac{\sin(4) \operatorname{erf}(8) + \frac{1}{2}}{\cos(4) \operatorname{erf}(8) + \frac{4e^{-64} \sin(4)}{\sqrt{\pi}}}$$

```
(%i3) float(%)
```

```
(%o3) 3.607121545883621
```



# Pozostałe elementy języka

## ■ Pętla while:

```
while warunek do  
(  
instrukcje  
)
```

```
NWD(a,b):=block([c],  
while b # 0 do  
(  
    c:mod(a,b),  
    a:b,  
    b:c  
) ,  
return(a)  
);
```

## ■ Pętla unless:

```
unless warunek do  
(  
instrukcje  
)
```

```
unless w = 0 do
```



## ■ Instrukcja skoku:

```
....  
etykieta,  
    ....  
go (etykieta),  
....
```

```
NWD(a,b):=block([c],  
petla,  
(  
    c:mod(a,b),  
    a:b,  
    b:c  
) ,  
if b # 0 then go (petla),  
return(a)  
);
```



## ■ Funkcje:

### ■ Brak polimorfizmu:

```
(%i1) f(x,y):=x+y;
(%o1)          f(x, y) := x + y
(%i2) f(x):=x;
(%o2)          f(x) := x
(%i3) f(3);
(%o3)          3
(%i4) f(1,2);

Too many arguments supplied to f(x):
[1, 2]
-- an error.  To debug this try debugmode(true);
```

### ■ Przekazywanie przez wartość:

```
(%i1) f(x):=x:x+1;
(%o1)          f(x) := x : x + 1
(%i2) f(3);
(%o2)          4
(%i3) a:3;
(%o3)          3
(%i4) f(a);
(%o4)          4
(%i5) a;
(%o5)          3
```





## ■ Funkcje:

### ■ Brak polimorfizmu:

```
(%i1) f(x,y):=x+y;
(%o1)
f(x, y) := x + y
(%i2) f(x):=x;
(%o2)
f(x) := x
(%i3) f(3);
(%o3)
3
(%i4) f(1,2);

Too many arguments supplied to f(x):
[1, 2]
-- an error. To debug this try debugmode(true);
```

### ■ Przekazywanie przez wartość:

```
(%i1) f(x):=x:x+1;
(%o1)
f(x) := x : x + 1
(%i2) f(3);
(%o2)
4
(%i3) a:3;
(%o3)
3
(%i4) f(a);
(%o4)
4
(%i5) a;
(%o5)
3
```



## ■ Funkcje:

### ■ Dostęp do zmiennych "globalnych":

```
(%i1) f(x):=(z:3);  
(%o1) f(x) := z : 3  
(%i2) z;  
(%o2) z  
(%i3) z:1;  
(%o3) 1  
(%i4) f(11);  
(%o4) 3  
(%i5) z;  
(%o5) 3
```

### ■ Usuwanie funkcji: remfunction(nazwa)

```
(%i6) remfunction(f);  
(%o6) [f]  
(%i7) f(1);  
(%o7) f(1)
```



## ■ Funkcje:

### ■ Dostęp do zmiennych "globalnych":

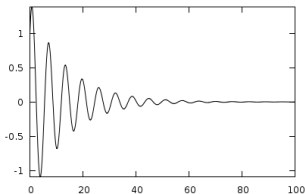
```
(%i1) f(x):=(z:3);  
(%o1)                                f(x) := z : 3  
(%i2) z;  
(%o2)                                z  
(%i3) z:1;  
(%o3)                                1  
(%i4) f(11);  
(%o4)                                3  
(%i5) z;  
(%o5)                                3
```

### ■ Usuwanie funkcji: remfunction(nazwa)

```
(%i6) remfunction(f);  
(%o6)                                [f]  
(%i7) f(1);  
(%o7)                                f(1)
```



# Oscylator harmoniczny



$$m\ddot{x} + \alpha\dot{x} + kx = 0$$

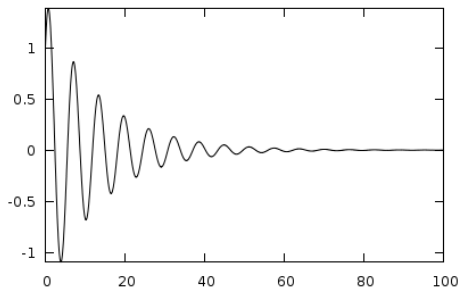
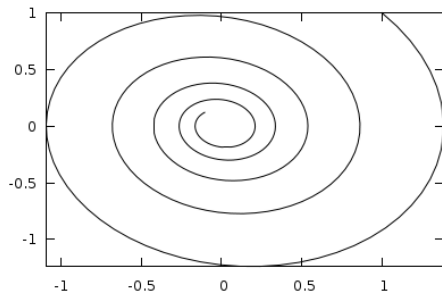
Szukamy rozwiązania:

$$x(t=0) = 1 \quad \dot{x}(t=0) = 1$$

$$k = 1 \quad \alpha = 0,15 \quad m = 1$$

```
load(draw);
m:1;
k:1;
alfa:0.15;
rr:m*'diff(x,t,2)+alfa*'diff(x,t)+k*x=0;
ro:ode2(rr,x,t);
rs:ic2(ro,t=0,x=1,'diff(x,t)=1);
X:rhs(rs);
V:diff(X,t);
draw2d(explicit(X,t,0,100));
draw2d(nticks=300,parametric(X,V,t,0,30));
```



Zależność  $x(t)$ 

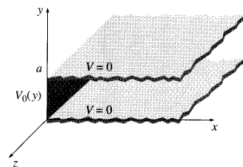
Zależność prędkości od wychylenia



# Potencjał wewnątrz nieskończonej wnęki

Warunki brzegowe:

- 1  $V = 0$  dla  $y = 0$
- 2  $V = 0$  dla  $y = a$
- 3  $V = V_0(y)$  dla  $x = 0$



Postulujemy rozwiązanie spełniające  $\nabla^2 V = 0$  w postaci  $V(x, y) = X(x)Y(y)$ . Po podstawieniu otrzymujemy:

$$\frac{d^2 X(x)}{dx^2} = k^2 X(x) \quad \frac{d^2 Y(y)}{dy^2} = -k^2 Y(y)$$

```
(%i1) depends(X,x,Y,y);
(%o1) [X(x), Y(y)]
(%i2) (ode2(diff(X,x,2)=k^2*X,X,x),ev(%%,%k1=A,%k2=B));
      - k x      k x
(%o2) X = %e B + %e A
(%i3) (ode2(diff(Y,y,2)=-k^2*Y,Y,y),ev(%%,%k1=C,%k2=D));
(%o3) Y = cos(k y) D + sin(k y) C
(%i4) V:rhs(%o2*%o3);
      - k x      k x
(%o4) (%e B + %e A) (cos(k y) D + sin(k y) C)
```

Warunki brzegowe implikują  $A = 0$ ,  $D = 0$ ,  $\sin ka = 0 \Rightarrow k = \frac{n\pi}{a}$ ,  $n \in \mathbb{N}$

$$V(x, y) = \sum_{n=1}^{\infty} C(n) e^{-\frac{\pi n x}{a}} \sin\left(\frac{\pi n y}{a}\right)$$

Korzystamy z liniowości równania Laplace'a:

$$V(x, y, N) = \sum_{n=1}^N C(n) e^{-\frac{\pi n x}{a}} \sin\left(\frac{\pi n y}{a}\right)$$

W szczególności dla  $x = 0$  otrzymujemy warunek brzegowy  $V(0, y) = V_0(y)$ :

$$\sum_{n=1}^N C(n) \sin\left(\frac{\pi n y}{a}\right) = V_0(y)$$

Mnożymy powyższe równanie przez  $\sin\left(\frac{m\pi y}{a}\right)$ ,  $m \in \mathbb{N}_+$  i całkujemy od 0 do  $a$



```
(%i6) declare(n,integer)$
(%i7) declare(m,integer)$
(%i8) sum('integrate('ev(V,C=C(n),x=0)*sin(m*pi*y/a),y,0,a),n,1,N)
      =integrate(V0(y)*sin(m*pi*y/a),y,0,a);
```

$$\sum_{n=1}^N C(n) \int_0^a \sin\left(\frac{\pi m y}{a}\right) \sin\left(\frac{\pi n y}{a}\right) dy = \int_0^a V_0(y) \sin\left(\frac{\pi m y}{a}\right) dy$$

```
(%i9) ev(%,nouns);
```

$$0 = \int_0^a V_0(y) \sin\left(\frac{\pi m y}{a}\right) dy \quad (\text{dla } n \neq m)$$

```
(%i10) ev(%i8,n=m,nouns);
```

$$\frac{a}{2} \sum_{n=1}^N C_n = \int_0^a V_0(y) \sin\left(\frac{\pi m y}{a}\right) dy \quad (\text{dla } n = m)$$

Zadajemy jakąś nietrywialną postać potencjału:

```
(%i11) V0(y):=y*(y-a)$
(%i12) (ev(%o8,nouns),factor(%));
```

$$\frac{a}{2} \sum_{n=1}^N C(n) = \frac{2 a^3 ((-1)^n - 1)}{\pi^3 n^3}$$





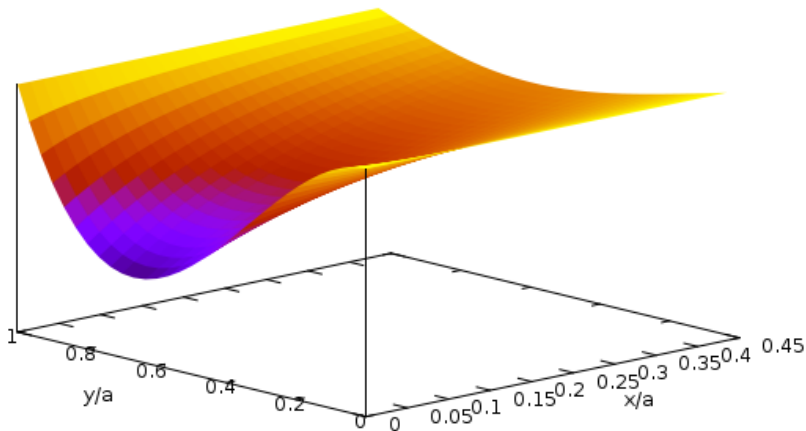
Otrzymujemy wzór na  $C_n$

$$C(n) = \frac{4a^2 ((-1)^n - 1)}{\pi^3 n^3}$$

$$V(x, y, N) = \sum_{i=1}^N \frac{4a^2 ((-1)^n - 1)}{\pi^3 n^3} e^{-\frac{\pi n x}{a}} \sin\left(\frac{\pi n y}{a}\right)$$

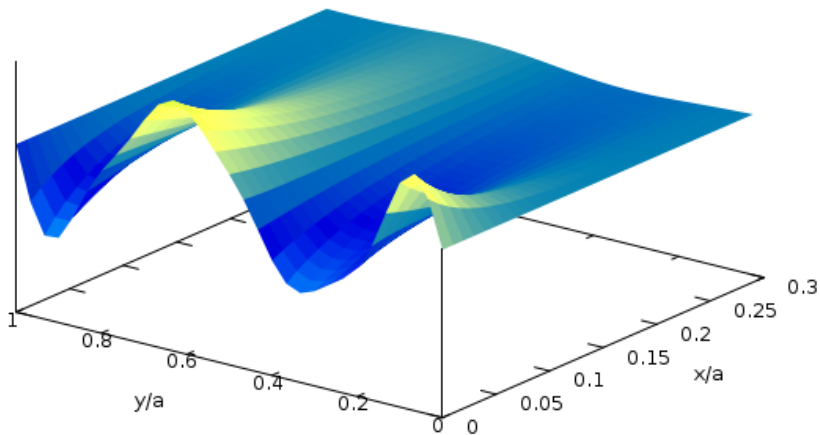
```
(%i13) draw3d(surface_hide=true, ztics='none,
xlabel="x/a", ylabel="y/a",
colorbox=false, enhanced3d=true,
explicit(ev(V(x,y,10),a=1),x,0,0.45,y,0,1),
terminal=wxt);
```





$$V_0(y) = y(y - a)$$





$$V_0(y) = \cos(10x)$$



# Jednowymiarowa studnia potencjału

Tworzymy procedurę obliczającą iloczyn skalarny w przestrzeni  $L^2$ :

```
(%i1) iloczyn(f,g,x,a,b):=(integrate(conjugate(f)*g,x,a,b))$
```

oraz normującą funkcję:

```
(%i2) normuj(f,x,Const,a,b):=block([temp],
temp:iloczyn(f,f,x,a,b)=1,
temp:abs(rhs(solve(temp,Const)[1])),
return(subst(temp,Const,f)))$
```

Rozwiązujemy niezależne od czasu równanie Schroedingera:

$$-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \Psi_n = E_n \Psi_n$$

```
(%i3) declare(n,integer)$
(%i4) assume(h>0,m>0,E(n)>0,L>0,n>=0)$
(%i5) rSch:-h^2/(2*m)*'diff(Psi,x,2)=E(n)*Psi$
(%i6) ro:rootscontract(ode2(rSch,Psi,x));
```

$$\Psi = \%k1 \sin\left(\frac{\sqrt{2mE(n)}x}{\hbar}\right) + \%k2 \cos\left(\frac{\sqrt{2mE(n)}x}{\hbar}\right)$$

Wprowadzamy zmienną pomocniczą  $k_n$  oraz nakładamy warunki brzegowe  $\Psi_n(x=0) = \Psi_n(x=L) = 0 \Rightarrow \%k2 = 0, \quad k_n = \frac{n\pi}{L}$



```
(%i7) rs:scsimp(ro,%k2=0,sqrt(2*m*E(n))/h=k(n));
```

$$\Psi = \%k1 \sin(k(n) x)$$

```
(%i7) k(n):=n*pi/L$
```

```
(%i8) rhs(''%o7)$
```

```
(%i9) Psi(x,n):='';
```

$$\Psi(x, n) = \%k1 \sin\left(\frac{\pi n x}{L}\right)$$

```
(%i10) normuj(Psi(x,n),x,%k1,0,L)$
```

```
(%i11) Psi(x,n):='';
```

$$\Psi(x, n) := \frac{\sqrt{2} \sin\left(\frac{\pi n x}{L}\right)}{\sqrt{L}}$$

```
(%i12) rhs(solve(sqrt(2*m*E(n))/h=k(n),E(n))[1])$
```

```
(%i13) E(n):='';
```

$$E(n) := \frac{\pi^2 \hbar^2 n^2}{2 m L^2}$$

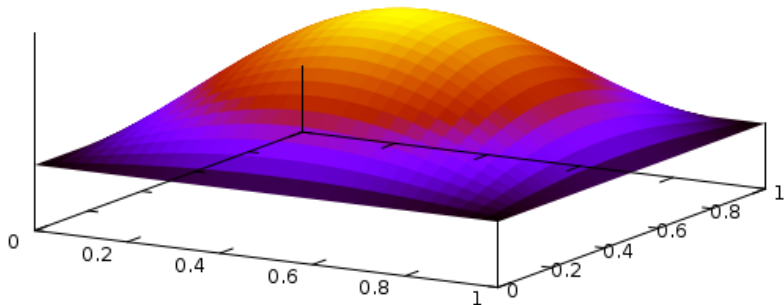


W wypadku dwuwymiarowym otrzymujemy:

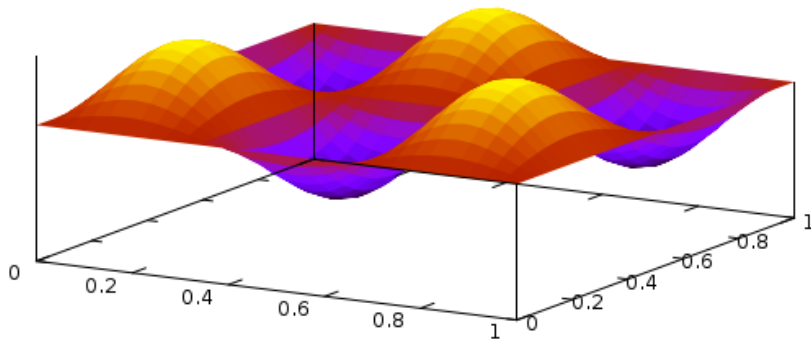
$$\Psi_{n_x, n_y} = \sqrt{\frac{4}{L_x L_y}} \sin\left(\frac{n_x \pi x}{L_x}\right) \sin\left(\frac{n_y \pi y}{L_y}\right)$$

$$E(n_x, n_y) = \frac{\pi^2 \hbar^2}{2m} \left[ \left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 \right]$$

Stan podstawowy



Stan wzbudzony  $n_x=3$ ,  $n_y=2$



## "Ruchy Browna"

- Badamy ruch drobinki znajdującej się początkowo w punkcie (0,0,0)
- W każdej iteracji, z równym prawdopodobieństwem może ona zmienić położenie  $\pm 1\hat{x}, \pm 1\hat{y}, \pm 1\hat{z}$

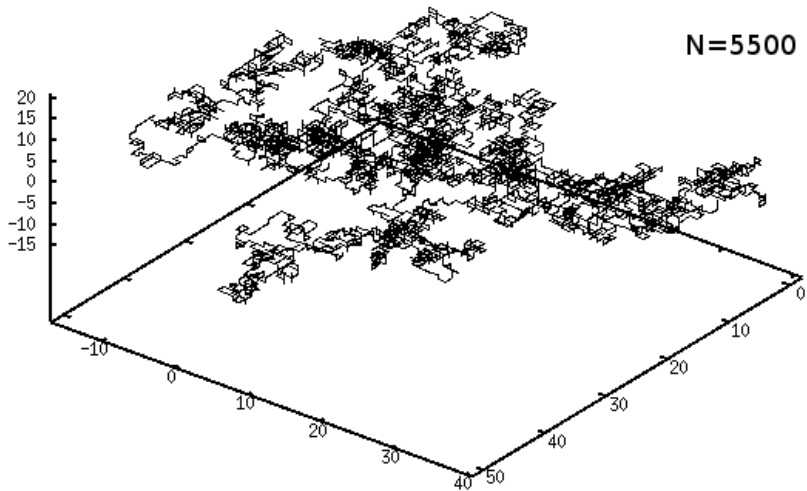
```
(%i1) RB(N):=block([polozienie:[[0,0,0]],los,temp],
    for i:1 thru N do(
        los:random(3)+1,
        temp:copylist(last(polozienie)),
        temp[los]:temp[los]+random(3)-1,
        polozienie:endcons(temp,polozienie)
    ),
    return(polozienie)
)$
```

```
(%i2) load(draw)$
```

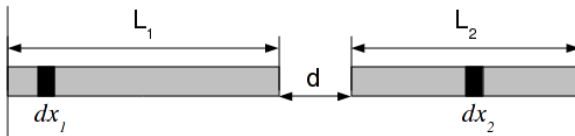
```
(%i3) draw3d(title="N=5500",point_type=0,points_joined=true,points(RB(5500)));
```







Znajdź siłę z jaką przyciągają się dwa współliniowe pręty, naładowane ładunkami  $Q_1$  i  $Q_2$



$$F = \int_0^{L_1} dx_1 \int_{L_1+d}^{L_1+d+L_2} dx_2 \frac{\lambda_1 \lambda_2}{4\pi\epsilon_0 (x_2 - x_1)^2}$$

```
(%i1) dF:lambda1*lambda2/(4*pi*eps0*(x2-x1)^2)$
(%i2) assume(L1>0,L2>0,d>0);
(%o2) [L1 > 0, L2 > 0, d > 0]
(%i3) F:integrate(integrate(dF,x1,0,L1),x2,L1+d,L1+d+L2);
Is L1 - x2 positive, negative, or zero? n;
lambda1 lambda2 (- log(L2 + L1 + d) + log(L2 + d) + log(L1 + d) - log(d))
(%o3) -----
4 %pi eps0

(%i4) logcontract(%);

lambda1 lambda2 log(-----)
d L2 + d L1 + d
2
(L1 + d) L2 + d L1 + d
2
(%o4) - -----
4 %pi eps0
```

```
(%i5) ev(%,lambda1=Q1/L1,lambda2=Q2/L2);
```

$$-\frac{\ln\left(\frac{d L_2 + d L_1 + d^2}{(L_1 + d) L_2 + d L_1 + d^2}\right) Q_1 Q_2}{4 \pi \varepsilon_0 L_1 L_2}$$

```
(%i6) assume(Q1>0,Q2>0,eps0>0);
```

```
(%o6) [Q1 > 0, Q2 > 0, eps0 > 0]
```

```
(%i7) is(%o5>0);
```

```
(%o7) true
```



# Gwiazdy

Gwiazda oddala się od obserwatora z prędkością  $v = 0.6c$ . Druga gwiazda oddala się od pierwszej również z prędkością  $v$ . Jaka jest prędkość  $N$ -tej gwiazdy względem nieruchomego obserwatora? Wzór na dodawanie prędkości

$$v_n = \frac{v + v_{n-1}}{1 + \frac{v v_{n-1}}{c^2}}$$

Algorytm iteracyjny:

```
(%i1) v(N,V):=block(L:[V],
  for i:2 thru N do(
    temp:ratsimp((L[i-1]+V)/(1+L[i-1]*V/c^2)),
    L:endcons(temp,L)
  ),
  return(L)
)$
(%i2) v(5,a*c);
```

$$\left[ a c, \frac{2 a c}{a^2 + 1}, \frac{(a^3 + 3 a) c}{3 a^2 + 1}, \frac{(4 a^3 + 4 a) c}{a^4 + 6 a^2 + 1}, \frac{(a^5 + 10 a^3 + 5 a) c}{5 a^4 + 10 a^2 + 1} \right]$$



Porównanie z wzorem teoretycznym. Definiujemy pośpieszność:

$$y = \operatorname{artgh}\left(\frac{v}{c}\right)$$

Z addytywności  $y$  mamy:

$$y_n = n \operatorname{artgh}\left(\frac{ac}{c}\right) = n \operatorname{artgh}(a)$$

Zachodzi przy tym:

$$\gamma = \cosh y \Rightarrow \cosh(n \operatorname{artgh}(a)) = \gamma = \left(\sqrt{1 - \frac{v_n^2}{c^2}}\right)^{-1}$$

$$\left(\frac{1}{\cosh(n \operatorname{artgh}(a))}\right)^2 = 1 - \frac{v_n^2}{c^2}$$

i ostatecznie:

$$v_n = \sqrt{\left(1 - \left(\frac{1}{\cosh(n \operatorname{artgh}(a))}\right)^2\right)} c^2$$



Porównanie obu wyników:

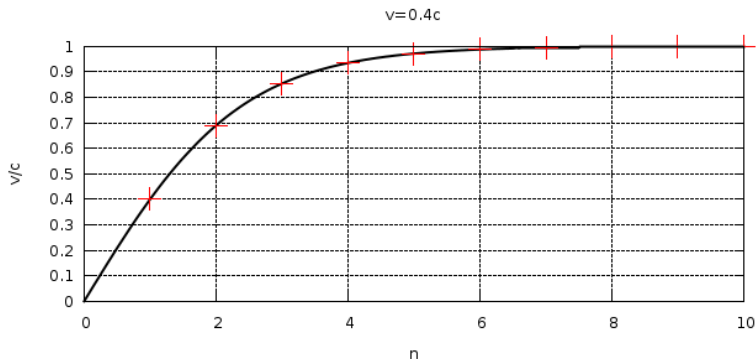
```
(%i3) u(n):=sqrt((1-(1/cosh(n*atanh(0.6))^2))*c^2)$
```

```
(%i4) assume(c>0)$
```

```
(%i5) fpprintprec:3$
```

```
(%i6) makelist(u(n)/c,n,1,10)-ev(v(10,a*c)/c,a=0.6),ratprint:false;
```

$[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7.45 \cdot 10^{-9}, 2.328 \cdot 10^{-9}, 6.11 \cdot 10^{-10}, 1.55 \cdot 10^{-10}]$



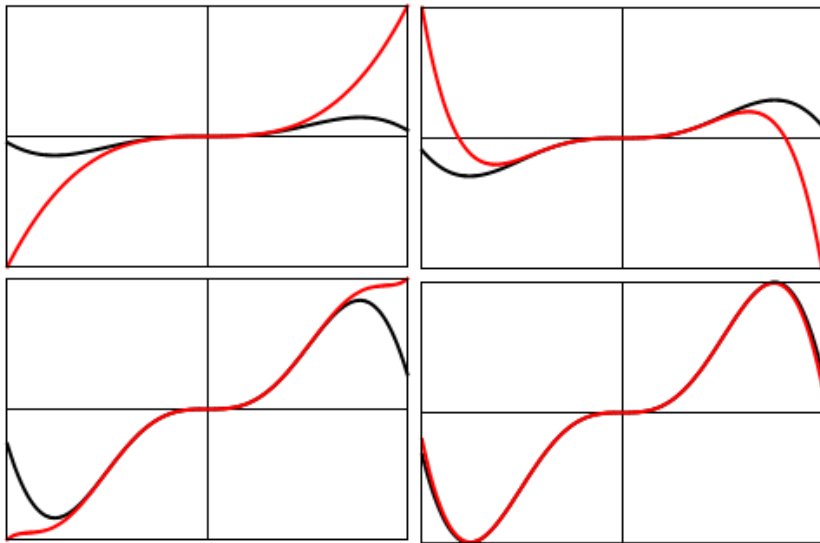
# Prosta animacja

Kolejne przybliżenia funkcji  $f(x)$  na przedziale  $[a, b]$  szeregiem Maclaurina

```
animacja(f,var,kroki,a,b):=block([T],
for i:1 thru kroki do
(
?sleep(1),
T:taylor(f,var,0,i),
draw2d(xtics='none,ytics='none,xaxis=true,
xaxis_type=solid,yaxis=true,yaxis_type=solid,
line_width=2,explicit(f,var,a,b),color=red,
explicit(T,var,a,b),terminal=wxt)
)
);
```



```
(%i2) animacja(sin(x)*x^2,x,10,-3,3);
```





Nie omówione możliwości programu:

- Statystyka
- Obliczenia zmiennoprzecinkowe
- Numeryczne rozwiązywanie równań różniczkowych
- Szeregi
- Wiele innych ...



## Literatura

- 1 Edwin L. Woollett *Maxima by Example* [www.csulb.edu/~woollett/](http://www.csulb.edu/~woollett/)
- 2 Cyprian T. Lachowicz *Matlab Scilab Maxima, Opisy i przykłady zastosowań*
- 3 Paulo Ney de Souza *The Maxima Book*  
[michel.gosse.free.fr/documentation/fichiers/maximabook.pdf](http://michel.gosse.free.fr/documentation/fichiers/maximabook.pdf)
- 4 [www.telefonica.net/web2/biomates/maxima/gpdraw/](http://www.telefonica.net/web2/biomates/maxima/gpdraw/)
- 5 [www.eng.ysu.edu/~jalam/engr6924s07/sessions/session22/session22.htm](http://www.eng.ysu.edu/~jalam/engr6924s07/sessions/session22/session22.htm)

